AURA: Asymptotically Optimal Uncertainty-Robust Replanning Algorithm for Kinodynamic Systems

Seyedali Golestaneh, Zhuoyun Zhong, Donghyung Lee, and Constantinos Chamzas

***Abstract*—Sampling-based motion planners offer a practical and scalable approach to kinodynamic motion planning, notably for high-dimensional, underactuated, or non-holonomic systems. However, these planners are typically used offline, requiring execution to begin only after the trajectory has been computed. In addition, the planned trajectory may not be accurately tracked in the presence of motion uncertainty, leading to deviations from the nominal solution. In this work, these limitations were addressed within a unified framework, AURA, an asymptotically-optimal meta-planner framework that improves both path quality and tracking performance during execution. In addition to the main execution thread, this framework comprises a replanning method that continuously explores the state space and refines the trajectory during execution, and an optimization process that refines future control inputs to reduce tracking error. Together, these components enable AURA to leverage asymptotically optimal planning online while improving execution accuracy under motion uncertainty. The proposed approach is evaluated in both simulation and real-world environments across multiple systems, demonstrating consistent improvements in trajectory quality, tracking accuracy, and overall performance compared with baselines. The source code is available at https://github.com/elpis-lab/AURA.**

***Index Terms*—Motion and Path Planning, Kinodynamic Planning, Planning Under Uncertainty**

I. INTRODUCTION

SAMPLING-BASED kinodynamic planners provide a general and theoretically sound framework for motion planning in systems with complex dynamics [1]. By relying solely on forward propagation, they avoid the need for steering functions, which are often intractable for arbitrary dynamical systems. Moreover, asymptotically optimal (AO) variants [2] offer *probabilistic completeness* and *asymptotic optimality*, ensuring that the probability of finding a feasible trajectory approaches one and that the cost converges almost surely to the global optimal as the number of samples increases.

Despite these guarantees, two practical issues hinder the execution of trajectories computed by kinodynamic planners. Consider the kinodynamic planning problem of pushing an object in Fig. 1. First, the initial trajectory is often *suboptimal* due to limited planning time. Second, *motion uncertainty*, arising from model mismatch and actuation errors, inevitably induces deviation from the initial trajectory during open-loop execution. Online replanning strategies [3]–[5] attempt to handle trajectory suboptimality by recomputing trajectories from the latest observed state. However, for kinodynamic AO planners, frequent replanning is computationally expensive, may discard prior exploration progress, and without a steering function, reconnecting to the original search tree is not trivial [6].

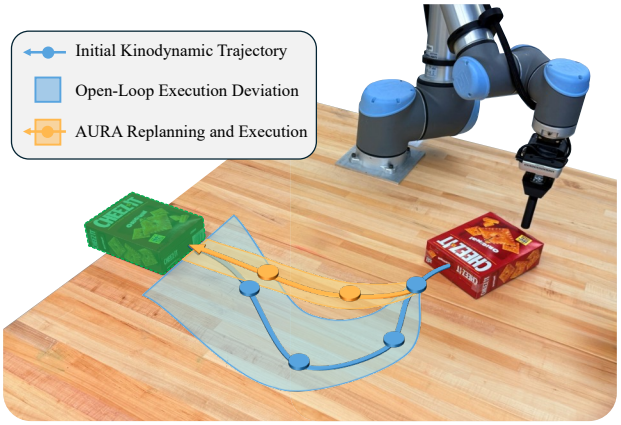


Fig. 1. **Illustration of open-loop and AURA execution for non-prehensile manipulation.** Under a limited time budget and with an approximated dynamics model, open-loop execution of the planned path yields a suboptimal trajectory and accumulates large execution deviation (blue). In contrast, AURA performs online global replanning and local optimization, resulting in a better overall trajectory and smaller execution error (orange).

Meanwhile, receding-horizon control [7]–[9] mitigates motion uncertainty through closed-loop tracking of a reference trajectory. However, the reference is typically fixed and not improved during execution. Moreover, these methods often rely on high-frequency state feedback [10]. In certain tasks—such as the non-prehensile manipulation scenario shown in Fig. 1—state observations may only be available at low-frequency discrete action boundaries. (e.g., after a pushing action is executed due to in-hand sensing).

In this work, we propose AURA, **A**symptotically Optimal **U**ncertainty-Robust **R**eplanning **A**lgorithm that combines the global exploration and asymptotic guarantees of AO kinodynamic planners [6], [11] with the local robustness of receding-horizon control. Rather than treating planning and execution independently, AURA operates as a meta-layer on top of any forward-propagation-based AO planner, enabling: (i) continuous online refinement of trajectory quality during execution, and (ii) local control optimization to mitigate execution deviation. Concretely, the contributions of AURA are:

- A novel online replanning framework for kinodynamic AO planners that continuously improves trajectory quality online without requiring a steering function.
- A GPU-accelerated local control optimization module to empirically mitigate execution deviation up to 72%, together with an existence proof that a recovering control always exists under mild assumptions.

This work was supported in part by Amazon WPI RBE Research Award 2025, NSF CRII Grant No. 2451108, and WPI funds. All authors are affiliated with the Department of Robotics Engineering, Worcester Polytechnic Institute (WPI), Worcester, MA 01609, USA {sgolestaneh, zzhong3, dlee5, cchamzas} @ wpi.edu.

- Comprehensive evaluation across multiple dynamics models in simulated and real-world tasks, including learned models, demonstrating up to 50% reduction in total task time over receding-horizon and planning baselines.

II. Related Work

This section reviews prior work most relevant to this method from three perspectives: asymptotically-optimal kinodynamic planning, online replanning, and robust control under uncertainty. AURA lies at their intersection, combining AO kinodynamic planners with global replanning and local optimization.

A. Asymptotically-Optimal Kinodynamic Planners

Sampling-based planners for kinodynamic systems must satisfy system dynamics and constraints [12]. These methods have been successfully applied to a variety of systems, while avoiding the need for an explicit steering function. Among such methods, SST* achieves asymptotic optimality through pruning and best-node selection [6]. More generally, the meta-algorithm AO-X [11] lifts any kinodynamic planner, e.g., RRT [13] or EST [14], into asymptotically optimal planners by augmenting the search in state-cost space.

However, AO planners guarantee convergence to the optimal path only in probability as computation time increases [2]. Therefore, under limited time scenarios, the resulting solution may remain substantially suboptimal. To address this challenge, AURA exploits the extra time during execution, enabling the planner to further refine and improve the trajectory quality.

B. Online Replanning

AO planning is typically cast as an offline problem, since the planning terminates once a trajectory is selected for execution. In contrast, real-time planning methods couple planning and execution, enabling online adaptation based on the current state during execution, but generally fail to offer optimality [15]. Early work in this direction, RAMP for non-holonomic systems [3], [16], leverages replanning during execution to adapt to environmental changes and action uncertainty. Moreover, RRT$^{\text{X}}$ [4] enables reconnecting to the original tree at runtime when environment changes. However, these methods require a steering function for local rewiring.

A recently proposed kinodynamic replanning framework, KRAFT [5], avoids the need for the steering function. Nevertheless, it re-propagates the full tree after the state deviates from the nominal trajectory, which cannot guarantee collision-free execution without access to the true dynamics model. In contrast, AURA computes a recovery control that steers the system back to the collision-free planned trajectory without discarding the planning progress or requiring to the true dynamics model.

C. Control and Optimization under Uncertainty

On the execution side, several control strategies have been developed to improve tracking accuracy and robustness under motion uncertainty, including Model Predictive Control (MPC) [7], and robust control [8]. Among these, MPC and its variants can explicitly handle system dynamics and constraints [17], [18]. Related extensions include differentiable MPC formulations that enable online parameter adaptation [19], and sampling-based stochastic control methods, such as MPPI [20]. Despite their strong execution performance, they are susceptible to local minima in long-horizon tasks.

A common solution is to follow a pre-planned reference trajectory [21], [22], but these trajectories remain fixed throughout execution and are not refined. AURA combines AO kinodynamic planning with predictive optimization during execution to improve execution accuracy under motion uncertainty while continuing to refine the overall trajectory quality.

III. Problem Statement

In this work, the kinodynamic motion planning problem under bounded execution uncertainty is considered. Let $\mathcal{X}$ and $\mathcal{U}$ denote the state space and control space, respectively. The state space $\mathcal{X}$ is partitioned into two disjoint subsets, $\mathcal{X} = \mathcal{X}_{\text{obs}} \cup \mathcal{X}_{\text{free}}$, where $\mathcal{X}_{\text{obs}}$ represents the invalid state region and $\mathcal{X}_{\text{free}}$ is the valid state region. The *nominal dynamics model* f describes the evolution of the system:

$$x(T) = x(0) + \int_0^T f(x(t), u(t))dt, \tag{1}$$

where $x(t) \in \mathcal{X}$ and $u(t) \in \mathcal{U}$ denote the state and control input of the system at time t. In practice, forward propagation is commonly performed over time intervals Δt_i using piecewise-constant controls [1]. Let $\Gamma(\cdot,\cdot)$ denote the *nominal state transition function* over one such interval:

$$x_{i+1} = \Gamma(x_i, u_i) = x_i + \int_0^{\Delta t_i} f(x(t), u_i)dt, \quad x(0) = x_i \tag{2}$$

where x_i is the start state and u_i is the constant control applied for propagation duration $\Delta t_i \in [t_{\min}, t_{\max}]$.

Let $\pi = (x_0, u_0, \ldots, x_{T-1}, u_{T-1}, x_T)$ denote a trajectory, where $\boldsymbol{x} = (x_i)_{i=0}^{T} \in \mathcal{X}_{\text{free}}^{T+1}$ is the state trajectory generated by applying the control sequence $\boldsymbol{u} = (u_i)_{i=0}^{T-1} \in \mathcal{U}^T$ from x_0, satisfying Eq. 2. Let $J(\cdot)$ represent the *cost function* that typically measures the quality of a trajectory, defined as:

$$J(\pi) = \textstyle\sum_{i=0}^{T-1} \ell(x_i, u_i), \tag{3}$$

where $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}^+$ is the running cost (e.g., path length, time, or energy consumption).

During *execution*, the true system state may deviate from the nominal state x_{i+1} due to model mismatch, unmodeled dynamics, and actuation errors. This deviation is modeled as:

$$x_{i+1}^{\text{gt}} = \Gamma(x_i, u_i) + w_i, \tag{4}$$

where x_{i+1}^{gt} denotes the observed ground-truth system state after applying control u_i for Δt_i, and w_i is the *bounded execution uncertainty*, satisfying $\|w_i\| \le \delta$. Accordingly, the *tracking error* along trajectory π is defined as:

$$E(\pi) = \textstyle\sum_{i=0}^{T} \|x_i - x_i^{\text{gt}}\|, \tag{5}$$

where x_i and x_i^{gt} denote the nominal and observed states after execution, respectively.

The robust kinodynamic planning problem under motion uncertainty is formalized as follows: Given an initial state x_{start},

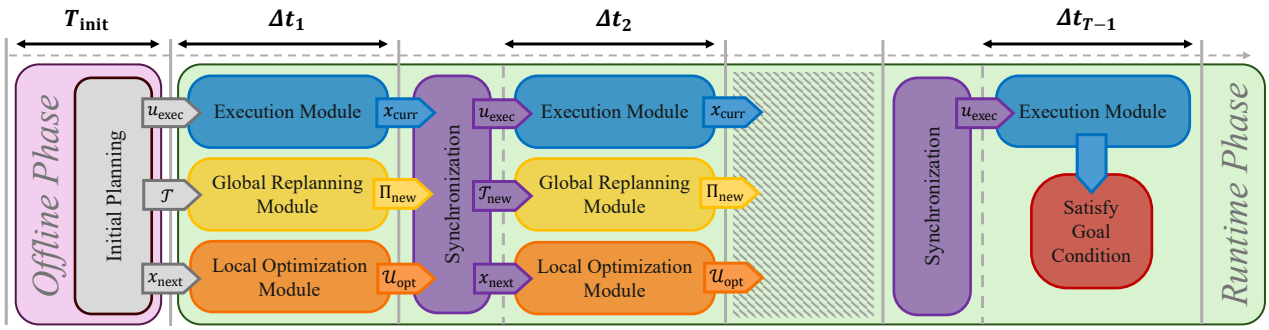


Fig. 2. **Time frame of AURA algorithm**, which includes *offline* and *runtime* phase. The *offline* phase provides an initial guide for the next phase. At *runtime*, three modules work concurrently for each Δt_i interval: (a) Execution module applies the next control and returns the resulting state, (b) global replanning module expands the existing planning tree, and (c) optimization module predicts the possible future controls based on the next state. (d) At the end of the cycle, all the data is synced to provide the inputs for the next cycle.

a goal region $\mathcal{X}_{\text{goal}}$, and the nominal propagation model $\Gamma(\cdot,\cdot)$, AURA aims to determine a nominal trajectory π, such that:

$$x_{i+1} = \Gamma(x_i, u_i),\ i = 0, \dots, T-1,$$
$$x_0 = x_{\text{start}},\ x_T \in \mathcal{X}_{\text{goal}},\ u_{0:T-1} \in \mathcal{U}^T,\ x_{0:T}, x^{\text{gt}}_{0:T} \in \mathcal{X}^{T+1}_{\text{free}}. \tag{6}$$

while seeking to minimize the trajectory cost in Eq. 3 and the tracking error in Eq. 5, thereby improve trajectory quality and robustness to bounded motion uncertainty during execution.

IV. AURA FRAMEWORK

The main idea of AURA is shown in Alg. 1 and Fig. 2. Given an AO planner $\mathcal{P}$ and the nominal forward propagator Γ, AURA aims to produce a trajectory that satisfies the goal condition while minimizing a cost function and execution error. Let $\mathcal{T}$ denote the search tree maintained by $\mathcal{P}$, and $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$ denote the set of root-to-goal plans in $\mathcal{T}$. In the *Offline Phase*, $\mathcal{P}$ plans for planning time T_{init}, and selects the best trajectory $\pi^\star \in \Pi$ to start execution (Alg. 1, line 1 - line 5). Afterward, the *Runtime Phase* begins, which consists of three parallel modules running together: **(a)** Execution Module, **(b)** Global Replanning Module, and **(c)** Local Optimization Module. Rather than operating sequentially, these modules run concurrently during trajectory execution. This procedure repeats iteratively until the system reaches the goal region.

A. Execution Module

This module, shown as blue blocks in Fig. 2, is responsible for applying the first control action u_{exec} of the current best trajectory for its associated duration Δt_i and observing the resulting ground-truth state $x^{\text{gt}}_{\text{curr}}$, during which the system evolves based on Eq. 4 (Alg. 1, line 7). In simulation, the system evolves based on the physics engine after applying the control input. In the real world, the consequent state results directly from the system's physical response to the executed control. In both cases, the resulting ground-truth state is then observed and passed to the synchronization module. In this work, the observation is assumed to be accurately available at the end of each execution interval without measurement noise.

Algorithm 1: AURA Framework

Input: AO planner $\mathcal{P}$; Transition function $\Gamma(\cdot,\cdot)$; Start state x_{start}; Goal region $\mathcal{X}_{\text{goal}}$; Initial planning time T_{init}; Neighborhood sampling radius σ;
Result: Executed control sequence driving x_{start} to $\mathcal{X}_{\text{goal}}$

1 $(\Pi, \mathcal{T}) \leftarrow \text{PLAN}(\mathcal{P}, \Gamma, x_{\text{start}}, \mathcal{X}_{\text{goal}}, \text{time} = T_{\text{init}})$
2 $x^{\text{gt}}_{\text{curr}} \leftarrow x_{\text{start}}$
3 $\pi^\star \leftarrow \text{GETBESTPLAN}(\Pi)$
4 $u_{\text{exec}} \leftarrow \text{GETFIRSTCONTROL}(\pi^\star)$
5 $x_{\text{next}} \leftarrow \text{GETNEXTPLANNEDSTATE}(\pi^\star)$
6 **while** $x_{curr} \notin \mathcal{X}_{goal}$ **do**
/* Execution Thread */
7 $x^{\text{gt}}_{\text{curr}} \leftarrow \text{EXECUTE}(x^{\text{gt}}_{\text{curr}}, u_{\text{exec}})$
8 /* Replanning Thread */ ▷ Alg. 2
9 $(\Pi, \mathcal{T}) \leftarrow \text{REPLANNING}(\mathcal{P}, \Gamma, \mathcal{T}, x_{\text{next}}, \mathcal{X}_{\text{goal}}, \Delta t_{u_{\text{exec}}})$
10 /* Optimization Thread */ ▷ Alg. 3
11 $(\mathcal{X}_{\text{child}}, \mathcal{U}_{\text{child}}) \leftarrow \text{GETCHILDREN}(x_{\text{next}}, \mathcal{T})$
12 $\mathcal{X}_{\mathcal{B}} \leftarrow \text{SAMPLENEARBY}(x_{\text{next}}, \sigma)$
13 $\mathcal{U}_{\text{opt}} \leftarrow \text{OPTIMIZATION}(\mathcal{X}_{\mathcal{B}}, \mathcal{X}_{\text{child}}, \mathcal{U}_{\text{child}}, \Gamma)$
/* Synchronization */
14 AWAITTHREADS()
15 $\Pi \leftarrow \text{RECALCULATECOST}(x^{\text{gt}}_{\text{curr}}, \Pi)$
16 $\pi^\star \leftarrow \text{GETBESTPLAN}(\Pi)$
17 $x_{\text{next}} \leftarrow \text{GETNEXTPLANNEDSTATE}(\pi^\star)$
18 $x^\star \leftarrow \text{NEAREST}(x^{\text{gt}}_{\text{curr}}, \mathcal{X}_{\mathcal{B}} \cup \{x_{\text{next}}\})$
19 $u_{\text{exec}} \leftarrow \mathcal{U}_{\text{opt}}[x^\star, x_{\text{next}}]$

B. Global Replanning Module

As noted earlier, the key property of the AO planner $\mathcal{P}$ is asymptotic convergence to the optimal trajectory as the number of samples increases [2]. This motivates resuming planning from the existing tree to find higher-quality trajectories (Fig. 3). Two main operations occur in this module: *Pruning* and *Replanning* (line 9 of Alg. 1, i.e., Alg. 2).

Pruning: After selecting u_{exec}, the search tree needs to be updated to remain kinodynamically consistent. As shown in Fig. 3, during execution toward x_1, branches that are not descendants of x_1 (faded nodes) can no longer be used. Finding a suitable control to connect x_1 to these nodes constitutes a *2-Point Boundary Value Problem (2PBVP)* that requires a steering function to solve, to which this work does not assume access. The next state of $\pi^\star$, x_1, is set as the new root (Alg. 2, line 1),

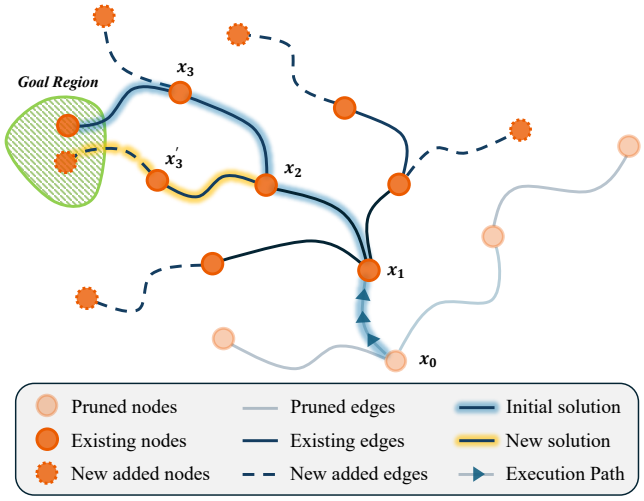


Fig. 3. **Tree update during the runtime phase.** As the first segment of the solution is executed from x_0 to x_1, nodes and edges that are no longer reachable are pruned (faded), and new samples are added to the tree (dashed). If a lower-cost solution (golden) is found, it becomes the new best trajectory $\pi^\star$.

Algorithm 2: Replanning

Input: AO planner $\mathcal{P}$; Transition function $\Gamma(\cdot,\cdot)$; Tree $\mathcal{T}_{\text{curr}}$; Next state x_{next}; Goal region $\mathcal{X}_{\text{goal}}$; Replanning time T_{replan};

Output: Updated plans Π_{new}, Updated tree $\mathcal{T}_{\text{new}}$

1 $x_{\text{start}} \leftarrow x_{\text{next}}$

2 $\mathcal{T}_{\text{pruned}} \leftarrow \textsc{PruneUnreachableNodes}(\mathcal{T}_{\text{curr}}, x_{\text{start}})$

3 $(\Pi_{\text{new}}, \mathcal{T}_{\text{new}}) \leftarrow \textsc{Plan}\left(\mathcal{P}, x_{\text{start}}, \mathcal{X}_{\text{goal}}, \Gamma, T_{\text{replan}}, \mathcal{T}_{\text{pruned}}\right)$

4 **return** $(\Pi_{\text{new}}, \mathcal{T}_{\text{new}})$

and all nodes and branches that are not descendants of this new root are removed to construct the tree $\mathcal{T}_{\text{pruned}}$ (Alg. 2, line 2).

Replanning: The planner then continues refining the trajectory, and runs for the remaining execution interval T_{replan} with a new start state, starting with the existing tree $\mathcal{T}_{\text{pruned}}$. As $\mathcal{P}$ is an AO planner, there is a non-zero probability that a lower-cost trajectory can be found [2], an example of which is illustrated by the highlighted golden trajectory in Fig. 3.

C. Local Optimization Module

This module employs a novel precomputation-based optimization strategy to reduce the tracking error defined in Eq. 5. In contrast to MPC and other receding-horizon methods, which compute the control after observing the updated state, AURA computes candidate recovery controls for the next cycle before the actual state is observed, during the current execution interval. The resulting optimized control set $\mathcal{U}_{\text{opt}}$ serves as the candidate pool for selecting the execution control in the next cycle.

In the presence of motion uncertainty (Eq. 4), executing the control will lead the system to deviate from the nominal state x_i to the true state $x_i^{\text{gt}} \in \mathcal{B}_\delta(x_i)$, where $\mathcal{B}_\delta(x_i)$ denotes the neighborhood of x_i defined as a ball of radius δ. As shown by the solid red line in Fig. 4 **(a)**, this deviation can accumulate along the trajectory. Proposition 1 in Sec. V proves that there exists a recovery trajectory segment, generated by a control function u_{rec}, from the perturbed state back toward the nominal trajectory. This motivates the use of numerical optimization

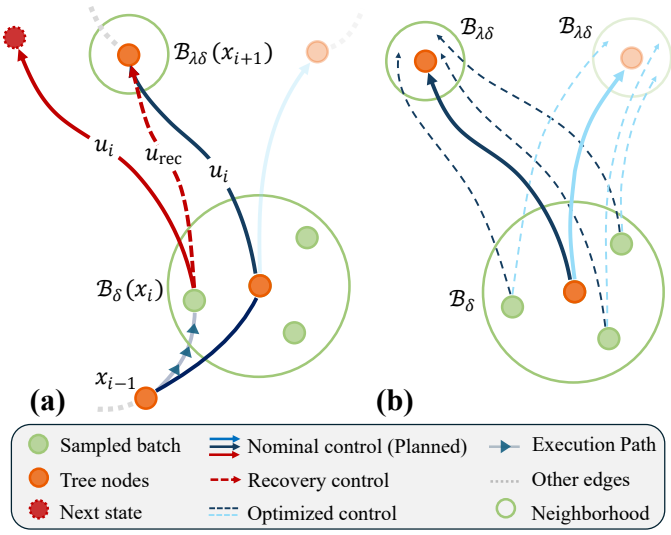


Fig. 4. **Batched optimization process**. **(a)** This is a trajectory segment. When the executed state deviates from the desired state (x_i), applying the nominal control (red arrow) can further amplify the deviation. The optimization module seeks to approximate the optimal recovery control (dashed red arrow). **(b)** For each sample in the neighborhood of x_i (green nodes) and each child on the tree (orange nodes), AURA calculated an optimized control (dashed lines).

Algorithm 3: Optimization

Input: Sampled local nodes $\mathcal{X}_{\mathcal{B}}$; Children nodes $\mathcal{X}_{\text{child}}$; Corresponding controls $\mathcal{U}_{\text{child}}$; Transition $\Gamma(\cdot,\cdot)$;

Output: Optimized controls $\mathcal{U}_{\text{opt}}$

1 $(X_{\text{start}}, X_{\text{child}}, U^0) \leftarrow \textsc{AssembleBatch}(\mathcal{X}_{\mathcal{B}}, \mathcal{X}_{\text{child}}, \mathcal{U}_{\text{child}})$

2 **for** $k \leftarrow 0$ **to** N **do**

3 $\quad \mathcal{L} \leftarrow \left\| \Gamma\left(X_{\text{start}}, U^k\right) - X_{\text{children}} \right\|^2$

4 $\quad U^{k+1} \leftarrow \text{CLIP}_{\mathcal{U}}\left(U^k - \alpha \dfrac{\partial \mathcal{L}}{\partial U^k}\right)$

5 **return** U^N

to search for a local control that best approximates such a recovery by minimizing the terminal propagation error.

While the execution is underway, this module samples a batch of states $\mathcal{X}_{\mathcal{B}}$ in the neighborhood of the next successor state (Alg. 1, line 11 - line 12) as candidates for possible execution outcomes, illustrated as green states in Fig. 4. For each sampled candidate, the next planned control provided by the planner (solid lines in Fig. 4 **(b)**) serves as an initial guess for an optimization process that adjusts control inputs to get closer to the next children (dashed line in Fig. 4 **(b)**).

Alg. 3 shows the optimization procedure over all pairs between the b sampled states in $\mathcal{X}_{\mathcal{B}}$ and the c child states in $\mathcal{X}_{\text{child}}$. The optimizer computes one recovery control for each sampled state–child pair, producing a set of $b \times c$ optimized controls. To enable efficient GPU-based parallel optimization, the sample–child inputs are vectorized into aligned matrices. Each sampled state is repeated c times, once for each child state, while the child states and corresponding planned controls are repeated for each of the b sampled states. This forms three aligned matrices with $b \times c$ entries: initial states X_{start}, target states X_{child}, and planned controls U^0 (Alg. 3, line 1).

To numerically estimate u_{rec} for each sample-child combina-

tion and minimize the error $\|\Gamma(X_{\text{start}}, U) - X_{\text{child}}\|$, this module employs gradient descent with step size α using the dynamics gradient with respect to the control inputs (Alg. 3, line 2 - line 4). After each update, the controls are projected onto the admissible set $\mathcal{U}$. This formulation makes AURA compatible with any differentiable dynamics, including analytical functions and learned models, while gradients for unknown dynamics can be approximated using batched finite differences.

D. Synchronization

At this stage of the process, the execution thread has advanced the system to a new state x_{curr}, the global replanning process has expanded the tree from the new root, and the optimization process has generated a $\mathcal{U}_{\text{opt}}$ set to mitigate deviation.

After the execution, any small deviations can alter the relative costs of previously computed trajectories. Therefore, at the end of each cycle, the costs of the stored candidate solutions are recalculated (Alg. 1, line 15) based on the current state. Accordingly, the next state along the current best trajectory $\pi^\star$ is chosen as the next target state (Alg. 1, line 16 - line 17). Additionally, the state in $\mathcal{X}_{\mathcal{B}}$ that is nearest to x_{curr} is selected as the best estimate (Alg. 1, line 18). Based on this estimate and the target state, the best optimized control is selected from $\mathcal{U}_{\text{opt}}$ to be executed in the next cycle (Alg. 1, line 19). If the target state is a newly added child, its recovery control is optimized on demand, since the local optimization module had not included the new child in the precomputed batch.

V. Recovery Control Existence Proof

This section proves the existence of a recovery control that takes a perturbed state back toward the nominal successor along a planned trajectory, as shown in Fig. 4 **(a)**.

Assumption 1. *The system dynamics Γ is Lipschitz continuous in both the state and control variables. Formally, $\exists K_x^\Gamma, K_u^\Gamma \geq 0$ such that, $\forall x, x' \in \mathcal{X}$, $\forall u, u' \in \mathcal{U}$:*

$$\begin{aligned} \|\Gamma(x,u) - \Gamma(x',u)\| &\leq K_x^\Gamma \|x - x'\|, \\ \|\Gamma(x,u) - \Gamma(x,u')\| &\leq K_u^\Gamma \|u - u'\|. \end{aligned} \tag{7}$$

Assumption 2. *The system satisfies Chow's condition [6].*

Chow's condition implies *small-time local accessibility (STLA)*, meaning that the reachable set from a state contains an open neighborhood around that state.

Such assumptions are standard in the analysis of kinodynamic sampling-based planners. Under these assumptions, we borrow the following result from [6, Lemma 6, Definition 7].

Lemma 1. *Let $\boldsymbol{x}$ be a feasible state trajectory for a system satisfying Assumption 1 and Assumption 2. Then there exists a positive value δ_0, termed dynamic clearance, such that for every $\eta \in (0, \eta_0]$, every $x_0' \in \mathcal{B}_\delta(x_0)$, and every $x_T' \in \mathcal{B}_\delta(x_T)$, there exists a dynamically feasible state trajectory $\hat{\boldsymbol{x}}$ satisfying $\hat{x}_0 = x_0'$ and $\hat{x}_T = x_T'$. The trajectory is called δ-robust if both its obstacle clearance ϵ, i.e., minimum distance from obstacles over all trajectory states, and its dynamic clearance η_0 are greater than δ.*

With Lemma 1, we can obtain the following proposition.

Proposition 1 (Existence of a Recovery Segment)**.** *Let $\overline{x_i x_{i+1}}$ be a segment from x_i to x_{i+1} along a δ_0-robust nominal state trajectory. Then for any perturbed state $x_i' \in \mathcal{B}_\delta(x_i)$ with $\delta \leq \delta_0$, there exists a dynamically feasible state trajectory segment $\overline{x_i' x_{i+1}}$ from x_i' to x_{i+1}.*

Proof. Consider a nominal δ_0-robust trajectory segment $\overline{x_i x_{i+1}}$. By Lemma 1, for any $\delta \in (0, \delta_0]$, any initial state $\hat{x}_0 \in \mathcal{B}_\delta(x_i)$, and any terminal state $\hat{x}_T \in \mathcal{B}_\delta(x_{i+1})$, there exists a dynamically feasible trajectory segment connecting $\hat{x}_0$ to $\hat{x}_T$. Setting $\hat{x}_0 = x_i'$ and $\hat{x}_T = x_{i+1}$, since $x_{i+1} \in \mathcal{B}_\delta(x_{i+1})$ for any $\delta > 0$, there exists a dynamically feasible trajectory from x_i' to x_{i+1}. □

Proposition 1 guarantees the existence of a local recovery trajectory segment, together with an admissible recovery control function u_{rec}. The proposed optimization module does not claim to recover u_{rec} exactly from the nominal initialization. Instead, it is a practical local search that attempts to approximate it with u_{opt} by minimizing the terminal propagation error $\|\Gamma(x_i', u_{\text{opt}}) - x_{i+1}\|^2$. Therefore, the theory motivates the objective of the optimization, while the optimization module (Sec. IV-C) serves as a numerical approximation mechanism.

Additionally, Proposition 2 shows that exact recovery is not required, and provides a conditional sufficient criterion. If the local optimization produces a control whose predicted successor is sufficiently close to the nominal target, and the remaining execution error is bounded, the executed successor remains inside the original δ-neighborhood. If the execution errors were routinely on the order of δ, then a planned δ-robust trajectory would itself be unreliable, since the system could easily leave the clearance region and collide with obstacles.

Proposition 2 (Approximate Recovery Suffices)**.** *Let $x_i' \in \mathcal{B}_\delta(x_i)$, and suppose the local optimization module returns a control u_{opt} such that $\Gamma(x_i', u_{\text{opt}}) \in \mathcal{B}_{\lambda\delta}(x_{i+1})$ for some $\lambda \in (0,1)$. If the execution error is bounded by $(1-\lambda)\delta$, that is, $\|x_{i+1}' - \Gamma(x_i', u_{\text{opt}})\| \leq (1-\lambda)\delta$, where x_{i+1}' denotes the executed successor state, then we have $x_{i+1}' \in \mathcal{B}_\delta(x_{i+1})$.*

Proof. Applying the triangle inequality gives

$$\begin{aligned} \|x_{i+1}' - x_{i+1}\| &\leq (1-\lambda)\delta + \|\Gamma(x_i', u_{\text{opt}}) - x_{i+1}\| \\ &\leq (1-\lambda)\delta + \lambda\delta = \delta. \end{aligned} \tag{8}$$

Hence $x_{i+1}' \in \mathcal{B}_\delta(x_{i+1})$. □

VI. Experiments and Results

This section presents a detailed experimental evaluation of AURA on multiple systems with different dynamics in both simulation and real-world environments. The results demonstrate the effect of different modules on the overall performance. All the experiments were run on a workstation with an Ultra7-265KF CPU, an NVIDIA RTX4070 Ti Super GPU, and 32GB of RAM.

To demonstrate the generalizability of the proposed method, systems with different nominal dynamics and state spaces were investigated, including:

- **6D Double Integrator:** A single-point acceleration control model used for forward propagation, with state space

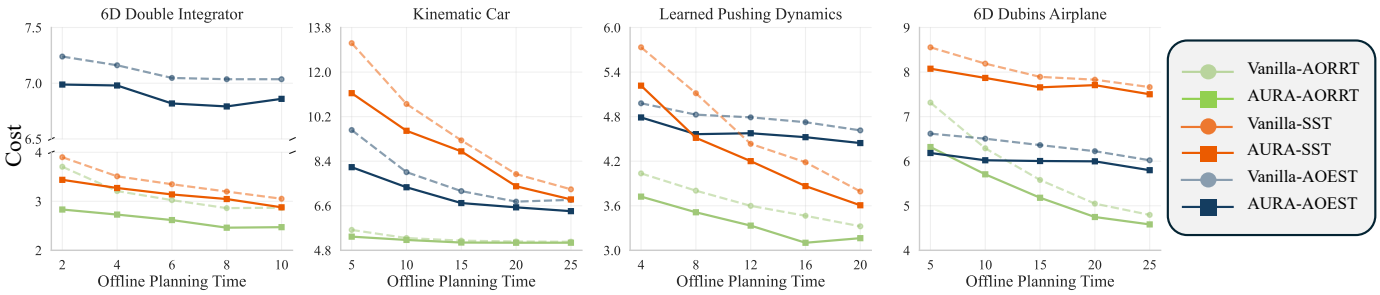


Fig. 5. **Trajectory quality comparison.** The final cost value of the solution path is represented for three planners across four dynamics averaged over 100 trials. When integrated with AURA, AO planners can further improve trajectory quality during execution in the runtime phase, given any initial planning time.

$x = [p_x, p_y, p_z, \dot{p}_x, \dot{p}_y, \dot{p}_z]^\top \in \mathcal{X} \subseteq \mathbb{R}^6$ where p denotes position, and control space $u = [\ddot{p}_x, \ddot{p}_y, \ddot{p}_z]^\top \in \mathcal{U} \subseteq \mathbb{R}^3$.

- **Kinematic Car:** A planar reduced-order kinematic bicycle model with state $x = [p_x, p_y, \theta]^\top \in \mathcal{X} \subseteq \mathrm{SE}(2)$ and control $u = [v, \phi]^\top \in \mathcal{U} \subseteq \mathbb{R}^2$, where θ, v, and ϕ denote yaw, longitudinal velocity, and steering angle, respectively. The dynamics follow the model in [23].
- **Learned Non-prehensile Pushing:** A learned dynamics of contact-rich planar pushing following the model in [24], with state $x = [p_x, p_y, \theta]^\top \in \mathcal{X} \subseteq \mathrm{SE}(2)$ and control $u = [p_c, u_d]^\top \in \mathcal{U} \subseteq \mathbb{R}^2$, where p_c continuously parameterizes an object-relative contact point along the object boundary, and u_d specifies the pushing distance.
- **Nonlinear 6D Dubins Airplane:** A nonlinear 3D flight model with state $x = [p_x, p_y, p_z, \psi, \gamma, v]^\top \in \mathcal{X} \subseteq \mathbb{R}^6$ and control $u = [a, \omega_\psi, \omega_\gamma]^\top \in \mathcal{U} \subseteq \mathbb{R}^3$, where ψ, γ, and v denote yaw, pitch, and speed, and a, ω_ψ, and ω_γ denote acceleration, yaw-rate, and pitch-rate, respectively [25].

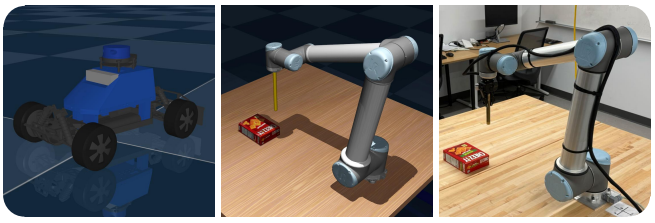
The robustness of AURA was evaluated across environments with different forms of execution uncertainty w (Eq. 4).

- **Gaussian Noise (GN):** Execution uncertainty is modeled as bounded additive Gaussian noise applied to the nominal motion of each system.
- **Mujoco Simulation:** Uncertainty arises from the mismatch between the nominal dynamics and the MuJoCo simulation dynamics [26], evaluated using a MuSHR car model [27] with a simplified reduced-order model (shown in Fig. 6 (a)) and a UR10 manipulation task (Fig. 6 (b)).
- **Real-World Robot:** Hardware inaccuracies and unmodeled dynamics introduce uncertainty in a tabletop manipulation task (Fig. 6 (c)).

For each system and environment, three AO planners implemented in the *Open Motion Planning Library (OMPL)* [28] were used: SST [6], AO-RRT, and AO-EST [11].

A. Runtime Trajectory Quality Improvement

As discussed in Sec. IV-B, AURA utilizes each execution interval Δt_i to continue the global search from the retained planning tree, and progressively improves the cost of the candidate trajectory. This experiment evaluates the attained improvement in this cost value $J(\pi)$ under a noise-free execution setting ($\|w\| = 0$) on a matched initial planning budget T_{init}, thereby isolating the effect of runtime global replanning on solution quality from the effects of tracking error.



(a) Car in Simulation (b) Manipulator in Simulation (c) Manipulator in Real-World

Fig. 6. **Evaluation environments.** (a) MuJoCo MuSHR car, (b) MuJoCo UR10 non-prehensile task, and (c) Real-world UR10 non-prehensile task.

The baseline is the trajectory generated by the corresponding vanilla AO planner , as shown in Fig. 5. As the execution deviation is zero, no recovery is required for this experiment.

For all the systems and planners, AURA consistently produced lower-cost trajectories than the corresponding vanilla planners, given the same offline planning time. The benefit is more pronounced for shorter planning budgets and generally decreases as the offline planning time increases, since longer planning already returns lower-cost trajectories. A similar trend is observed for the learned dynamics model, where finding a solution is more challenging, and the nonlinear 6D Dubins airplane, where 3D dynamics are strongly coupled. This experiment therefore demonstrates AURA 's ability to reduce initial computation while continuing trajectory refinement during execution, which is consistent with the theoretical properties of asymptotic optimality [2].

TABLE I
MEAN STEP-WISE TRACKING ERROR

System	Environment	AURA	MPPI	Open-loop Execution
Double Integrator	GN	0.00175	0.00194	0.00368
Kinematic Car	GN	0.03913	0.03146	0.13798
Pushing Dynamics	GN	0.02903	0.02671	0.05856
Dubins Airplane	GN	0.06404	0.06390	0.14991
Kinematic Car	MuJoCo	0.04477	0.03941	0.15385
Pushing Dynamics	MuJoCo	0.02015	0.01942	0.03621
Pushing Dynamics	Real-World	0.08313	0.07426	0.18560

B. Tracking Robustness

To examine how local optimization affects execution under uncertainty, step-wise tracking error was measured using Eq. 5. This evaluation includes execution uncertainty, where for each

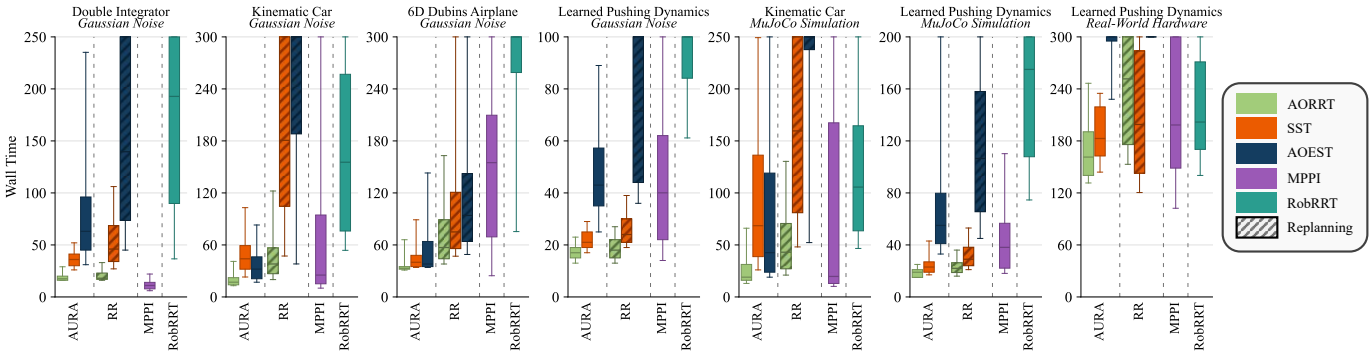


Fig. 7. **Task time comparison.** The distribution of overall task completion time (in seconds) over multiple trials for AURA, Restart Replanning, MPPI, and Robust-RRT is shown over 100 trials. The offline planning times were 10s for the double integrator, kinematic car, and Dubins airplane, and 5s for the pushing system. Lower values indicate faster task completion and better performance.

trial, an arbitrary reference trajectory is generated by propagating 10 randomly sampled piecewise-constant controls through the nominal dynamics (Eq. 2), with each control applied for one execution interval. For real-world trials, this is reduced to 5 pushes. Results are averaged over 100 trials for simulation and 20 for real-world. The comparison is made against open-loop execution, in which the nominal controls are applied without feedback, and MPPI, which observes the current state, optimizes a finite-horizon control sequence online, and executes its first control. In AURA, the best optimized control is selected at each step from $\mathcal{U}_{\text{opt}}$ as described in Sec. IV-C.

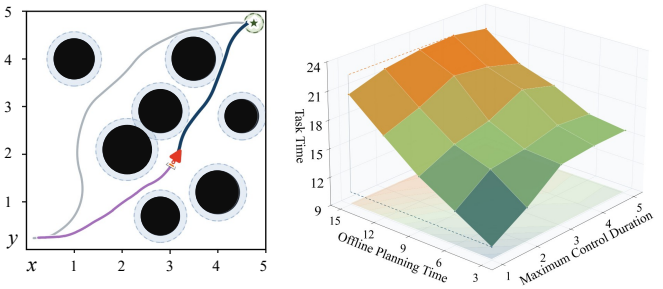
As reported in Table I, AURA substantially decreases tracking error relative to open-loop execution across all systems, and remains comparable to MPPI. Unlike MPPI, which performs optimization after observing the state, AURA precomputes candidate recovery controls during the execution and selects the best one at the synchronization point.

C. End-to-End Task Time Efficiency

To evaluate the overall task performance, wall time was measured as the total elapsed time required for each system to reach the goal, under motion uncertainty. This includes both the initial offline planning time and the total execution time. The baselines includes Restart Replanning (RR), Model Predictive Path Integral (MPPI) [20] and Robust-RRT (RobRRT) [29]. Following common practice in replanning under uncertainty [30], RR checks the step-wise deviation from its nominal trajectory. When this deviation exceeds a threshold, it discards the existing search tree and replans from the current observed state to the goal. MPPI provides receding-horizon control optimization. RobRRT incorporates execution uncertainty directly into planning using sampling-based reachable-set approximations [31].

Fig. 7 demonstrate that AURA reliability reduces wall time by utilizing global replanning and local optimization modules. Although RR uses the latest ground-truth state for planning, each restart requires a costly trajectory recomputation. In more complex settings, such as MuJoCo simulation and real-world setting, RR exhibits higher variance and longer execution times.

While MPPI outperforms AURA on the double integrator, indicating that finite-horizon optimization is sufficient for simple dynamics, its wall time increases substantially in more complex systems. This is because MPPI optimizes over a limited horizon without explicit awareness of the global state space or long-term constraints. Therefore, the controller aligns locally feasible states and converges to local minima. On the other hand, Robust-RRT have greater planning overhead with multiple forward propagations for each tree expansion, and it does not provide an asymptotic-optimality guarantee. In contrast, AURA retains global exploration and trajectory refinement, supporting more efficient task completion in complex systems and environments.



(a) Experiment Environment (b) Wall Time Over Different Hyperparameters

Fig. 8. **Hyperparameters Analysis.** (a) *XY* plane with obstacles (black) and their inflated boundaries (dashed lines). The red simulated car is tasked with reaching the goal region (Green). The gray trajectory is the offline planning output, the purple segment shows the executed trajectory, and the blue line shows the best current trajectory computed by AURA. (b) Effect of different offline planning times and maximum control durations on the overall wall time.

D. Hyperparameter Analysis

This section analyzes the sensitivity of AURA to two critical hyperparameters: *Offline Planning Time* T_{init} and *Maximum Control Duration* $t_{\max}$, which set the initial computation budget and upper bound on the sampled control duration, respectively. This experiment was conducted using the kinematic car with AO-RRT in the environment shown in Fig. 8 (a), where the average wall time was measured over 50 trials under different hyperparameter settings. For each planning edge, the control duration is sampled as $\Delta t_i \in [t_{\min}, t_{\max}]$. In this environment, the optimal solution passes through the narrow passage on the right, while a longer trajectory through the wider region on the left is more likely to be found during early planning.

As shown in Fig. 8 (b), larger offline planning time generally increases wall time. Since the reported wall time includes T_{init}, additional offline computation improves end-to-end efficiency only when the resulting trajectory improvement compensates for the added planning delay. This supports beginning execution once a feasible trajectory is available and continuing trajectory refinement during runtime.

The maximum control duration also influences wall time by changing the range of durations available to the planner and, consequently, the frequency of execution and replanning cycles. Across the tested settings, smaller $t_{\max}$ values generally result in lower wall time, although the trend is not strictly monotonic across offline planning budgets. The lowest average wall time, approximately $10s$, is obtained at $T_{\text{init}} = 3s$ and $t_{\max} = 1s$.

Overall, this analysis indicates that AURA's wall time is generally better with short offline planning budgets. Since $t_{\max} = 1$ s is the smallest value evaluated, it represents the best tested setting rather than a general optimum; the preferred maximum control duration depends on the system.

VII. Limitation and Future Work

This paper presents AURA, a meta-planning framework for robust motion planning under execution uncertainty. AURA augments kinodynamic AO planners with two complementary online modules: **(i)** a *Global Replanning Module* that continue refining the trajectory during execution, and **(ii)** a *Local Optimization Module* that precomputes recovery controls for possible execution outcomes. By combining online global exploration and local recovery optimization, AURA improves trajectory quality while mitigating tracking errors without requiring extensive offline planning. Experimental results across analytical and learned dynamics demonstrate the generality and practical effectiveness of this method under motion uncertainty.

Despite the advantages of the proposed method, some limitations remain. The local optimization module performance depends on the sampled batch and nominal dynamics accuracy. Consequently, tracking performance may degrade with poor optimizer convergence, or disturbances beyond the motion-uncertainty bound δ. Moreover, the recovery guarantee assumes that a nominal trajectory with clearance greater than δ exists. When the environment contains passages narrower than the uncertainty tube, such a trajectory may not exist with this method. The current formulation also assumes static obstacles and noise-free observation. Future work, will investigate parallelizing planning computations for both offline and runtime phases [32], accelerating the overall pipeline and better support systems requiring shorter control durations.

References

[1] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 7, no. 1, pp. 285–310, 2024. https://www.annualreviews.org/content/journals/10.1146/annurev-control-061623-094742

[2] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robot., and Autom. Syst.*, vol. 4, no. 1, pp. 295–318, 2021. https://www.annualreviews.org/content/journals/10.1146/annurev-control-061920-093753

[3] J. Vannoy and J. Xiao, "Real-time adaptive motion planning (ramp) of mobile manipulators in dynamic environments with unforeseen changes," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1199–1212, 2008. https://ieeexplore.ieee.org/document/4648454

[4] M. Otte and E. Frazzoli, "Rrt x: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 461–478. https://link.springer.com/chapter/10.1007/978-3-319-16595-0_27

[5] A. Sivaramakrishnan, S. Tangirala, D. M. Ramesh, E. Granados, and K. E. Bekris, "Kraft: Sampling-based kinodynamic replanning and feedback control over approximate, identified models of vehicular systems," *arXiv preprint arXiv:2409.11522*, 2024. https://arxiv.org/abs/2409.11522

[6] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Intl. J. of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016. https://journals.sagepub.com/doi/10.1177/0278364915614386

[7] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model predictive heuristic control," *Automatica (journal of IFAC)*, vol. 14, no. 5, pp. 413–428, 1978. https://dl.acm.org/doi/abs/10.1016/0005-1098(78)90001-8

[8] K. Zhou and J. C. Doyle, *Essentials of robust control*. Prentice Hall Upper Saddle River, NJ, 1998, vol. 104. https://www.ece.lsu.edu/kemin/saveold/books/essentials/solution_part1.ps

[9] W.-H. Chen, J. Yang, L. Guo, and S. Li, "Disturbance-observer-based control and related methods—an overview," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 1083–1095, 2015. https://ieeexplore.ieee.org/abstract/document/7265050

[10] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *IEEE Intl. Conf. Robot. Autom.*, 2021, p. 7330–7336. https://ieeexplore.ieee.org/document/9560990

[11] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1431–1443, 2016. https://ieeexplore.ieee.org/document/7588078

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. https://lavalle.pl/planning/

[13] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *Intl. J. of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001. https://ieeexplore.ieee.org/document/770022

[14] D. Hsu, J.-C. Latombe, R. Motwani, and P. Agarwal, "Path planning in expansive configuration spaces." *Intl. J. of Computational Geometry & Applications*, vol. 9, 1999. https://ieeexplore.ieee.org/abstract/document/619371

[15] K. I. Tsianos and L. E. Kavraki, "Replanning: A powerful planning strategy for hard kinodynamic problems," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.* IEEE, 2008, pp. 1667–1672. https://ieeexplore.ieee.org/document/4650965

[16] S. McLeod and J. Xiao, "Real-time adaptive non-holonomic motion planning in unforeseen dynamic environments," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, 2016, pp. 4692–4699. https://ieeexplore.ieee.org/document/7759690

[17] D. Mayne and H. Michalska, "Receding horizon control of nonlinear systems," *IEEE Trans. on Aut. Control*, vol. 35, no. 7, pp. 814–824, 1990. https://ieeexplore.ieee.org/document/57020

[18] Y. Song, B. Zhang, C. Wen, D. Wang, and G. Wei, "Model predictive control for complicated dynamic systems: a survey," *International Journal of Systems Science*, vol. 56, no. 9, pp. 2168–2193, 2025. https://www.tandfonline.com/doi/full/10.1080/00207721.2024.2439473

[19] A. Oshin, H. Almubarak, and E. A. Theodorou, "Differentiable robust model predictive control," *Robotics: Science and Syst.*, 2024. https://roboticsconference.org/2024/program/papers/3/

[20] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE Intl. Conf. Robot. Autom.* IEEE, 2016, pp. 1433–1440. https://ieeexplore.ieee.org/document/7487277

[21] D. Q. Mayne, E. C. Kerrigan, E. Van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *Int. J. of robust and nonlinear control*, vol. 21, no. 11, pp. 1341–1353, 2011. https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.1758?msockid=1bf0cd3899016aaf06bad9cc988e6b6c

[22] J. Köhler, D. Zhang, R. Soloperto, A. Carron, and M. Zeilinger, "An mpc framework for efficient navigation of mobile robots in cluttered environments," *arXiv preprint arXiv:2509.15917*, 2025. https://arxiv.org/abs/2509.15917

[23] T. Hellström, "Kinematics equations for differential drive and articulated steering," 2011. https://webapps.cs.umu.se/uminf/reports/2011/021/part1.pdf

[24] Z. Zhong, S. Golestaneh, and C. Chamzas, "Activepusher: Active learning and planning with residual physics for nonprehensile manipulation," in *IEEE Intl. Conf. Robot. Autom.*, 2026. https://elpislab.org/assets/pdf/zhong2026activepusheractivelearningplanning.pdf

[25] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a dubins airplane," in *2007 46th IEEE Conf. on decision and control*, pp. 2379–2384. https://ieeexplore.ieee.org/document/4434966

[26] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, 2012, pp. 5026–5033. https://ieeexplore.ieee.org/abstract/document/6386109

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

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

[29] A. Wu, T. Lew, K. Solovey, E. Schmerling, and M. Pavone, "Robust-rrt: Probabilistically-complete motion planning for uncertain nonlinear systems," in *Intl. Symp. on Rob. Research*. Springer, 2022, pp. 538–554. https://link.springer.com/chapter/10.1007/978-3-031-25555-7_36

[30] W. Sun, S. Patil, and R. Alterovitz, "High-frequency replanning under uncertainty using parallel sampling-based motion planning," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 104–116, 2015. https://ieeexplore.ieee.org/document/7027233

[31] T. Lew, L. Janson, R. Bonalli, and M. Pavone, "A simple and efficient sampling-based algorithm for general reachability analysis," in *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 1086–1099. https://proceedings.mlr.press/v168/lew22a.html

[32] N. Perrault, Q. H. Ho, and M. Lahijanian, "Kino-pax+: Near-optimal massively parallel kinodynamic sampling-based motion planner," *arXiv preprint arXiv:2602.02846*, 2026. https://arxiv.org/abs/2602.02846