

Domain Randomized Deployment of Massively Parallel MPC

Domänen-randomisierte Einsatz von massiv paralleler MPC

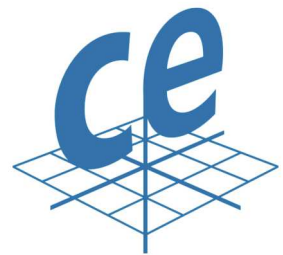
Master thesis in the field of study "Computational Engineering" by Magnus Dierking

Date of submission: March 10, 2026

1. Review: João Carvalho, Ph.D.
2. Review: An Thai Le, Ph.D.
3. Review: Prof. Jan Peters, Ph.D.
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Magnus Dierking, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 10. März 2026

M. Dierking

Abstract

This thesis designs and deploys a real-to-sim-to-real pipeline that integrates Sampling-based Model Predictive Control with GPU-accelerated MuJoCo MJX simulation and a Franka Research 3 robot for contact-rich manipulation. Using Bugtrap Escape and the Push-T benchmark in both simulation and hardware, we evaluate the Model Tensor Planning (MTP) algorithm against MPPI, CEM, and Predictive Sampling under practical real-time constraints. Across settings, MTP reliably handles challenging local minima and mode-switching behavior, demonstrating robust sim-to-real transfer at planning frequencies up to 10 Hz. In addition, we study domain randomization (DR) for sampling-based MPC via parallel rollouts over randomized physics domains with feedback-based reweighting. We find that randomizing global physics parameters (e.g., mass, friction) often yields weak adaptation signals in contact-dominated regimes. This lack of signal is primarily due to the fact that observed behavior is dominated by constraint resolution rather than underlying physics—a direct consequence of the coarse timesteps and solver adaptations required to meet real-time planning speed requirements. In contrast, contact-initiation parameters (e.g., collision margins) produce more interpretable, mode-dependent weight evolution. Overall, the work validates MTP as an effective sampling-based MPC method for contact-rich tasks and clarifies practical limits and opportunities for adaptive domain randomization in real-time planning.

Zusammenfassung

Diese Arbeit entwirft und implementiert eine Real-to-Sim-to-Real-Pipeline, die eine auf sampling-basierter modellprädiktiver Regelung (MPC) mit einer GPU-beschleunigten MuJoCo MJX-Simulation und einem Franka Research 3-Roboter für kontaktreiche Manipulationen integriert. Unter Verwendung von Simulations- und Hardwareversuchen bewerten wir den Model Tensor Planning (MTP)-Algorithmus im Vergleich zu MPPI, CEM und Predictive Sampling unter praktischen Echtzeitbeschränkungen. Über alle Einstellungen hinweg bewältigt MTP zuverlässig schwierige lokale Minima und Moduswechselverhalten und demonstriert einen robusten Sim-to-Real-Transfer bei Planungsfrequenzen von bis zu 10 Hz. Darüber hinaus untersuchen wir die Domänenrandomisierung (DR) für die sampling-basiertes MPC über parallele Rollouts über randomisierte Physikdomänen mit feedback-basierter Neugewichtung. Wir stellen fest, dass die Randomisierung globaler Physikparameter (z. B. Masse, Reibung) in kontaktreichen Szenarien oft zu schwachen Anpassungssignalen führt. Dieses Fehlen von Signalen ist in erster Linie darauf zurückzuführen, dass das beobachtete Verhalten eher durch die Auflösung von mathematischen Nebenbedingungen als durch die zugrunde liegende Physik dominiert wird – eine direkte Folge der groben Zeitschritte und Solver-Anpassungen, die erforderlich sind, um die Anforderungen an die Echtzeit-Planungsgeschwindigkeit zu erfüllen. Im Gegensatz dazu erzeugen Kontaktinitiierungsparameter (z. B. Kollisionsspielräume) eine besser interpretierbare Entwicklung der Gewichtungen. Insgesamt bestätigt die Arbeit MTP als eine effektive sampling-basierte MPC-Methode für kontaktreiche Aufgaben und verdeutlicht die praktischen Grenzen und Möglichkeiten der adaptiven Domänenrandomisierung in Echtzeit-Planungen.

Acknowledgments

I express my deepest gratitude to João for his excellent supervision and unwavering support throughout this thesis. I also thank An for suggesting this thesis topic and providing regular input from the other side of the world.

Jan and the IAS team deserve special thanks for their pro-student mentality, granting me lab access and resources in an environment that fostered exploration and growth. This extends to my fellow students and PhD candidates, whose lab discussions were invaluable.

Along with An and João, I am grateful to Theo and Ali for their mentorship during my time as a student assistant, allowing me to extend far beyond the curriculum of my master's program. I also thank Georgia for enabling me to pursue my interests through a student project.

Finally, my family and friends provided endless support and encouragement, for which I am profoundly thankful.

AI Disclosure: The author used an AI-assisted grammar and style tool (ChatGPT, Gemini, Perplexity, DeepL) solely to improve spelling, grammar, and formal consistency. All scientific content, structure, data, analysis, and final wording were authored, reviewed, edited, and approved by the author alone.

Contents

1. Introduction	2
1.1. Motivation	2
1.2. Structure of the Document	3
2. Foundations	4
2.1. Dynamical Systems	4
2.2. From Optimal Control to Model Predictive Control	5
2.3. Optimization- and Sampling-Based MPC	6
2.4. Trajectory Parametrization with Splines	13
2.5. Domain Randomization	16
2.5.1. Adaptive Domain Randomization	17
3. Related Work	19
4. Methodology	23
4.1. Model Tensor Planning	23
4.1.1. Extensions	25
4.2. Domain Randomization	26
5. Experimental Setup	28
5.1. Parallelized Planning Framework	28
5.1.1. JAX	28
5.1.2. Physics Simulation with Mujoco and MJX	30
5.1.3. Hydrax	32
5.2. Extending Hydrax	35
5.2.1. Simulation Environments	35
5.2.2. Framework Extension and Optimization	37
5.2.3. MJX Performance Optimizations	38
5.2.4. MTP and Extensions	39
5.2.5. Domain Randomization	40
5.3. Sim-to-Sim Pipeline	40
5.4. Real-to-Sim-to-Real Pipeline	42
5.4.1. Franka Research 3 Robotic Arm	42
5.4.2. ROS 2 Middleware	42
MoveIt2 and Servo	43
5.4.3. OptiTrack Motion Capture System	44
5.4.4. System Overview	45

6. Experimental Results	48
6.1. Bugtrap Escape	48
6.2. Push T Sim-2-Sim	49
6.2.1. Annealed MTP	52
6.3. Planning Pipeline Scaling	52
6.4. Push T on the FR3	54
6.5. Domain Randomization	58
7. Conclusion and Future work	63
7.1. Conclusion	63
7.2. Future Work	63
A. Appendix	69
A.1. Experimental Parameters	69
A.1.1. Sim-to-sim	69
A.1.2. Sim-to-real	72

Figures and Tables

List of Figures

2.1. Side-by-side comparison of MPPI, CEM and PS in the notation of Section 2.3. Key differences are highlighted: sampling strategy , weighting vs. elite selection , and update rule	12
2.2. Comparison of interpolation methods for exemplary control points.	15
4.1. Comparison of MPPI and MTP samples in the simulated Push T task.	24
5.1. Comparison of a simple Python function and its jaxpr, obtained via <code>jax.make_jaxpr(f)</code>	29
5.2. Illustration of warp execution on a GPU, Figure taken from [38].	29
5.3. UML of the basic Hydrax codebase. Italic fonts indicate abstract methods.	33
5.4. Data evolution over a single planning step. During expansion each data point consist of an element of the control input space U and we sample N trajectories of length T , copied over D domains. After parallel evaluation, we get the same structure for the experienced costs. To make a decision on the next action and distribution update, we accumulate information over trajectories (1), domains (2) and samples (3).	34
5.5. Bugtrap environment with physical walls. The green sphere marks the goal, light grey traces indicate samples rolled out via MJX.	36
5.6. Push-T environment. The green mocap object marks the goal and does not take part in physics computations.	37
5.7. Simplification of the collision meshes for the FR3 XML.	39
5.8. Effect of applying a SavGol filter to MPPI samples in simulation.	40
5.9. Sim-to-sim system overview. A CPU MuJoCo simulation provides the current state, which is copied to MJX for accelerator-resident planning. The resulting control sequence is then executed back in MuJoCo until the next replanning step.	41
5.10. Franka Research 3 robotic arm in the IAS lab with a custom end-effector and OptiTrack markers on the T-shaped object.	42

5.12. Basic functionality of MoveIt 2 Servo. Given a desired end-effector twist \mathcal{V}_{des} and the current joint positions \mathbf{q} , MoveIt 2 Servo computes joint velocity commands using the Jacobian. These commands are sent to the joint trajectory controller. In a separate thread, typically running at a lower frequency, collision checking and singularity handling are performed. If a potential collision or singularity is detected, the commands are scaled down to safely decelerate or stop the robot.	43
5.11. MoveIt2 collision boxes used for safety in the Push-T task.	43
5.13. Push-T experimental system overview. Blue arrows indicate ROS 2 topic-based communication between processes. Purple arrows indicate OptiTrack NatNet protocol streaming. The three processes (control, bridge, planning) are loosely coupled via asynchronous messaging, allowing the GPU-accelerated planning node to operate independently.	47
6.1. Empirical state visitation distributions for the bugtrap escape task over 10000 planning steps, aggregated over 6 seeds with noise on the initial particle position. Base MPPI and CEM remain stuck in the local minimum inside the trap in all seeds, while Predictive Sampling escapes only once and otherwise behaves similarly to CEM. The global samples together with elite perturbations enable MTP to escape and reach the goal in every run. All algorithms plan at 25 Hz.	49
6.2. Scenarios under test for our sim-to-sim evaluation of the Push-T task. Colors indicate difficulty: yellow is easiest, dark red hardest. The green cross marks the initial end-effector position. . .	50
6.3. Sim-to-sim results for Push T over long and short horizons. All methods use 256 samples at 20 Hz.	51
6.4. Runtime statistics over 200 planning steps for increasing numbers of samples or domains. The box plots show the median (middle line), 25th to 75th percentile (box), and the furthest data points (whiskers). For sample scaling, we fix the algorithm to use 1 domain; for domain scaling, we use 128 samples. Both sweeps use a simulation timestep of 0.01 s and 12 horizon steps with 4 simulation steps each.	53
6.5. Runtime statistics over 200 planning steps for various horizons. The box plots show the median (middle line), 25th to 75th percentile (box), and the furthest data points (whiskers). For each trial, we fix 128 samples across 6 domains and use a simulation timestep of 0.01 s.	54
6.6. GPU trace of a single jit-ed optimization step as in Algorithm 5 running base MPPI over 15 horizon steps on the FR3 Push T task. The 15 steps encapsulating <code>sim_steps_per_control_step</code> physics steps (small purple chunks) dominate the runtime.	54
6.7. Comparison of CEM, MPPI, MTP, and PS on the 3 DoF Push T task. All methods use 1024 samples at 10 Hz planning. Hyperparameters are reported in Appendix A.5.	55
6.8. Real-to-sim-to-real results with the 7 DoF T model across 6 seeds with pose/end-effector variations. All methods use 1024 samples at 8 Hz. Hyperparameters are reported in Appendix A.5.	57
6.9. Simulation rollouts under domain randomization. Differences across domains are largely driven by contact initiation and constraint resolution rather than by a smooth dependence on global physics parameters.	58
6.10. Evolution of domain weights under adaptive domain randomization for mass parameters. . . .	59
6.11. Real-system performance over 6 seeds with 160 samples each for 8 domains at 5 Hz. Left: final object pose relative to the goal pose; right: state cost over planning steps.	60

6.12. Exemplary evolution of domain weights for the domain-randomized Push-T task under margin randomization. Time is encoded by color and weight magnitude by transparency. Margins are randomized within ± 1 cm around the end-effector.	60
6.13. Effect of kinematic and frame-alignment errors: for the same nominal setting, different workspace regions can exhibit qualitatively different contact behavior.	61

List of Tables

A.1. Hyperparameters for the BugTrap escape task comparing MTP, CEM, MPPI, and Predictive Sampling.	69
A.2. Key MuJoCo/MJX simulation settings for the Push-T scenes (free 6-DoF object vs. jointed 3-DoF object) used in sim-to-sim testing.	70
A.3. FR3 Push-T sim-to-sim hyperparameters for the 3 DoF vs 7 DoF settings.	71
A.4. Key MuJoCo/MJX simulation settings for the Push-T scenes (free 6-DoF object vs. jointed 3-DoF object) used in real-to-sim-to-real testing.	72
A.5. FR3 Push-T real-to-sim-to-real hyperparameters for the 3 DoF vs 7 DoF settings.	73

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
AnMTP	Annealed Model Tensor Planning
CEM	Cross Entropy Method
IK	Inverse Kinematics
IT-MPC	Information-Theoretic Model Predictive Control
JAX	JAX
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
MJX	MuJoCo for JAX
MPC	Model Predictive Control
MPPI	Model Predictive Path Integral
MTP	Model Tensor Planning
PS	Predictive Sampling
QP	Quadratic Program
ROS 2	Robot Operating System 2
SMPC	Sampling-Based Model Predictive Control
XLA	Accelerated Linear Algebra

List of Symbols

Notation	Description
Σ	Input covariance
θ	Domain parameters
$\mathbf{1}$	Indicator function
Δt	Time step
$\dot{\mathbf{q}}$	Generalized velocities
λ	Temperature (inverse-cost scale)
\mathbb{P}	Prior measure
\mathbb{Q}	Controlled/trajectory measure
$\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$	Bias forces
$\mathbf{C}(x)$	Spline curve
$\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$	Constraint/contact forces
$\mathbf{J}(\mathbf{q})$	Constraint Jacobian
$\mathbf{M}(\mathbf{q})$	Mass matrix
\mathbf{q}	Generalized coordinates
\mathbf{t}	Knot vector
\mathbf{u}	Input, $\mathbf{u} \in \mathbb{R}^m$
\mathbf{U}	Mean input sequence, $\mathbf{U} = (\mathbf{u}_0, \dots, \mathbf{u}_{T-1})$
\mathbf{V}	Sampled input sequence, $\mathbf{V} = (\mathbf{v}_0, \dots, \mathbf{v}_{T-1})$
\mathbf{x}_{init}	Initial state
\mathbf{x}	State, $\mathbf{x} \in \mathbb{R}^n$
\mathbf{z}_i	Spline control points
$\text{KL}(\cdot \parallel \cdot)$	KL divergence
D	Number of domains
g	Discrete dynamics, $\mathbf{x}_{k+1} = g(\mathbf{x}_k, \mathbf{u}_k)$
h	Measurement model, $\mathbf{y}_k = h(\mathbf{x}_k, \mathbf{u}_k)$
I_f	Terminal cost
I	Stage cost
$N_{i,p}(x)$	B-spline basis function
N	Number of samples
p	B-spline degree

S	Trajectory cost
T	Horizon length (steps)
w_i	Sample weight

1. Introduction

1.1. Motivation

Model predictive control (MPC) [18, 42] has long been a staple in process engineering and robotics due to its ability to incorporate explicit system dynamics and constraints into the control loop. Classical optimization-based MPC formulations typically require strong assumptions like linearity, convexity, or smooth differentiability to ensure real-time feasibility. These assumptions break down in many robotic scenarios, in particular in contact-rich manipulation tasks, where dynamics are hybrid, nonsmooth, and subject to substantial model uncertainty. As a result, there has been increased interest in sampling-based variants that trade analytical structure for direct forward simulation of candidate control sequences.

Moreover, these methods have become increasingly attractive due to the widespread availability of massively parallel hardware such as GPUs. Efficient CUDA-based implementations of sampling-based MPC methods already exist [50] and can achieve impressive real-time performance, but they typically rely on hand-crafted or simplified dynamics models that are expensive to design, difficult to scale, and often inaccurate outside a narrow operating regime. This design burden makes them ill-suited for rapid prototyping and raises the question of whether we can instead directly leverage full-featured physics engines as dynamics backends.

Recent work has begun to explore this direction by coupling sampling-based MPC algorithms with highly parallel physics simulators, such as MuJoCo/MJX [9] and Isaac Gym [33], enabling large-scale rollouts over many trajectories in a single planning step. At the same time, the Model Tensor Planning (MTP) algorithm [28] was proposed as a hybrid sampling-based controller that combines elements of MPPI, CEM, and Predictive Sampling with a structured global sampling strategy to systematically escape local minima. MTP has been demonstrated in simulation, where it outperforms standard baselines on tasks with pronounced local minima and multi-modal solution strategies.

However, both the original Hydrax framework [23] underlying these simulations and the initial MTP experiments remain purely simulation-based, leaving open critical questions about real-world applicability. First, it is unclear whether MTP’s exploration benefits persist when planning frequencies are constrained by real-time requirements and physical hardware interfaces. Second, the interaction between contact-rich dynamics, soft-constraint solvers, and sampling-based planning under practical computational budgets has not been thoroughly characterized. Third, while domain randomization has been proposed for sim-to-real transfer in sampling-based MPC, prior work often underspecifies implementations and protocols [39], evaluates mainly on simple proof-of-concept successes, and leaves the meaning/validity of randomized parameters ambiguous [1], leaving it unclear when and why DR reliably improves real-world performance.

The goals of this thesis are therefore threefold:

Algorithmic evaluation: Extend our understanding of Model Tensor Planning relative to established sampling-based MPC baselines such as CEM, MPPI, and Predictive Sampling, both in challenging simulation scenarios and

in real-world experiments. We aim to validate whether MTP’s structured exploration translates to improved performance and robustness when deployed on physical hardware under real-time constraints.

Real-to-sim-to-real pipeline: Design and implement a complete real-to-sim-to-real pipeline that couples MTP with a GPU-accelerated physics engine (MuJoCo/MJX) and a physical Franka Research 3 robotic arm, using the Push-T manipulation task as a benchmark for assessing performance, robustness, and failure modes across simulation and hardware. This pipeline must address practical challenges including asynchronous control, state estimation via motion capture, solver tuning for real-time performance, and handling computational bottlenecks in batched physics simulation.

Domain randomization: Investigate DR as an additional degree of freedom in sampling-based MPC. We examine what types of physics parameters provide meaningful feedback signals at MPC replanning frequencies, and how to trade off the sample budget between parallel domains and trajectory diversity without sacrificing control quality.

Our evaluations across extensive sim-to-sim experiments, real-world deployment on the FR3, and an analysis of domain randomization strategies validate MTP’s practical utility. At the same time, while we demonstrate that domain randomization can be made to work in this MPC setting, our findings underscore that robust and systematic sim-to-real scaling remains difficult due to contact sensitivity, weak/ambiguous adaptation signals, and tight real-time constraints.

1.2. Structure of the Document

This thesis is organized as follows. Chapter 2 provides theoretical foundations in dynamical systems, optimal control, MPC variants, and domain randomization. Chapter 3 reviews related work, focusing on key inspirations for Model Tensor Planning. Chapter 4 formalizes the proposed MTP methodology and its extensions. Chapter 5 describes the implementation and experimental setup, spanning simulation and real-robot pipelines. Chapter 6 presents results from benchmarking, scaling analysis, and real-world evaluation. Chapter 7 concludes with findings and future directions.

2. Foundations

This chapter introduces the theoretical background for this thesis. We begin with a general formulation of dynamical state-space models and use this framework to motivate the principles of optimal control, from which model predictive control (MPC) arises as a receding-horizon approximation. Building on this, we discuss sampling-based variants of MPC that replace explicit optimization with stochastic evaluation and selection mechanisms. Finally, we introduce the concept of *domain randomization* as a framework for improving the transfer of simulation-trained policies to the real world. We define various techniques for aggregating information across multiple domains and describe how real-world feedback can be incorporated to perform statistical updates of a domain distribution.

2.1. Dynamical Systems

A *dynamical system* describes the time evolution of a system's state. In the context of control theory, the state variable is typically denoted by $\mathbf{x} \in \mathbb{R}^n$, evolving according to an autonomous or controlled differential equation. For practical purposes, and because all systems of ordinary differential equations (ODEs) can be transformed into first-order systems via state augmentation, we consider first-order systems without loss of generality. The general, continuous-time, controlled system evolution is described by

$$\dot{\mathbf{x}} = \mathbf{g}^c(\mathbf{x}(t), \mathbf{u}(t))$$

where $\mathbf{g}: \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is the system dynamics and $\mathbf{u} \in \mathbb{R}^m$ a control input vector. In most real-world applications, control actions are implemented digitally. Therefore, even when the underlying physical system is continuous-time, one must discretize it for control synthesis [14, Ch. 4]. This discretization (e.g. via zero-order hold or Euler approximation) yields the state update

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{g}^c(\mathbf{x}_k, \mathbf{u}_k),$$

Additionally, for nonlinear systems, it is often convenient to employ local linearization around an operating point $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$ [8, Sec. 2.4]. The linearized continuous-time dynamics are:

$$\dot{\mathbf{x}} \approx \mathbf{J}_{\mathbf{x}} \delta \mathbf{x} + \mathbf{J}_{\mathbf{u}} \delta \mathbf{u}, \quad \mathbf{J}_{\mathbf{x}} = \left. \frac{\partial \mathbf{g}^c}{\partial \mathbf{x}} \right|_{(\bar{\mathbf{x}}, \bar{\mathbf{u}})}, \quad \mathbf{J}_{\mathbf{u}} = \left. \frac{\partial \mathbf{g}^c}{\partial \mathbf{u}} \right|_{(\bar{\mathbf{x}}, \bar{\mathbf{u}})}$$

where $\delta \mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}$ and $\delta \mathbf{u} = \mathbf{u} - \bar{\mathbf{u}}$. The discretization of the linearized system yields:

$$\delta \mathbf{x}_{k+1} = \mathbf{J}_{\mathbf{x},d} \delta \mathbf{x}_k + \mathbf{J}_{\mathbf{u},d} \delta \mathbf{u}_k,$$

where $\mathbf{J}_{\mathbf{x},d} \in \mathbb{R}^{n \times n}$ and $\mathbf{J}_{\mathbf{u},d} \in \mathbb{R}^{n \times m}$ are the respective discrete-time system matrices obtained from discretizing the Jacobians. These transformations—linearization and discretization—are approximations and

inherently introduce modeling errors. Unless the system is exactly linear and time-invariant (LTI) and an exact discretization is used, the resulting model does not exactly capture the true system dynamics. In any case, we base this thesis on an (Euler-) discretized nonlinear model of the form

$$\mathbf{x}_{k+1} = \mathbf{g}(\mathbf{x}_k, \mathbf{u}_k). \quad (2.1)$$

In many applications, the full state \mathbf{x}_k is not directly measurable. Instead, one obtains outputs

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{u}_k), \quad (2.2)$$

with $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^p$, from which the state must be reconstructed. This motivates the use of algorithms such as the Luenberger observer or the Kalman filter, which estimate \mathbf{x}_k based on measured outputs and the above dynamics [27].

2.2. From Optimal Control to Model Predictive Control

In many engineering and scientific applications, we are not merely interested in observing the natural evolution of a dynamical system, but in actively controlling it to achieve specific goals like minimizing energy, reducing error, or stabilizing motion. This gives rise to the field of optimal control, which seeks the best possible sequence of actions under given dynamics and constraints.

Mathematically, the control objective is encoded in a cost function that must be minimized subject to the system's dynamic equations and admissibility conditions on states and inputs. The most fundamental, albeit idealized, formulation is the infinite-horizon optimal control problem [18, Ch. 4]:

$$\begin{aligned} J^*(\mathbf{x}_0) &= \min_{\mathbf{U}} \sum_{i=0}^{\infty} I(\mathbf{x}_i, \mathbf{u}_i) \\ \text{subj. to } &\mathbf{x}_{i+1} = \mathbf{g}(\mathbf{x}_i, \mathbf{u}_i), \quad i = 0, 1, 2, \dots \\ &\mathbf{x}_i \in \mathcal{X}, \quad \mathbf{u}_i \in \mathcal{U}, \quad i = 0, 1, 2, \dots \\ &\mathbf{x}_0 = \mathbf{x}_{\text{init}} \end{aligned}$$

where $I : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}_{\geq 0}$ is the stage cost, and $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$ are admissible state and control sets, respectively. Apart from specific cases where the control input can be shown to be a smooth function of the state and time, this problem is computationally intractable. In practice, control actions are planned over a finite time horizon, reflecting physical, computational, or operational limitations. For this reason, a natural extension of the above is the *finite-horizon* optimal control formulation

$$\begin{aligned} J^*(\mathbf{x}_0) &= \min_{\mathbf{U}} I_f(\mathbf{x}_T) + \sum_{i=0}^{T-1} I(\mathbf{x}_i, \mathbf{u}_i) \\ \text{subj. to } &\mathbf{x}_{i+1} = \mathbf{g}(\mathbf{x}_i, \mathbf{u}_i), \quad i = 0, \dots, T-1 \\ &\mathbf{x}_i \in \mathcal{X}, \quad \mathbf{u}_i \in \mathcal{U}, \quad i = 0, \dots, T-1 \\ &\mathbf{x}_T \in \mathcal{X}_f \\ &\mathbf{x}_0 = \mathbf{x}_{\text{init}}, \end{aligned}$$

which trades the infinite horizon for additional terminal conditions over a finite prediction horizon of length T . Here, $I_f : \mathbb{R}^n \mapsto \mathbb{R}_{\geq 0}$ is a terminal cost, and $\mathcal{X}_f \subseteq \mathbb{R}^n$ is a terminal constraint set, both chosen to approximate

the cost-to-go and constraint set beyond the finite horizon.

This formulation leads directly to the paradigm of *Model Predictive Control (MPC)* [18, Ch. 3], which solves a finite-horizon optimal control problem at every time step. Specifically, at time step k , MPC solves:

$$J_{k \rightarrow k+T|k}^*(\mathbf{x}_k) = \min_{\mathbf{U}} I_f(\mathbf{x}_{k+T|k}) + \sum_{i=k}^{T+k-1} I(\mathbf{x}_{i|k}, \mathbf{u}_{i|k}) \quad (2.3)$$

$$\text{subj. to } \mathbf{x}_{i+1|k} = \mathbf{g}(\mathbf{x}_{i|k}, \mathbf{u}_{i|k}) \quad (2.4)$$

$$\mathbf{x}_{i|k} \in \mathcal{X}, \quad \mathbf{u}_{i|k} \in \mathcal{U}, \quad i = k, \dots, k+T-1 \quad (2.5)$$

$$\mathbf{x}_{k+T|k} \in \mathcal{X}_f \quad (2.6)$$

$$\mathbf{x}_{k|k} = \mathbf{x}_k, \quad (2.7)$$

Key Concept

Model Predictive Control (MPC) solves a finite-horizon optimal control problem online at every time step, using the current state estimate as the initial condition. Only a subset of the optimal sequence is applied before re-solving at the next step. This process is known as the *receding horizon* strategy.

2.3. Optimization- and Sampling-Based MPC

Optimization-Based Approaches

The MPC formulation in Equation (2.3) defines, at each time step, a constrained nonlinear optimization problem. This formulation is intractable in general due to several factors: the system dynamics are often nonlinear, complicating closed-form solutions and the presence of state and input constraints can break the smoothness and differentiability properties needed to compute optimal solutions. To obtain tractable solutions, the literature has historically focused on special cases that admit closed-form or efficiently solvable formulations. These approaches, however, rely on simplifying assumptions.

A canonical example is the Linear Quadratic Regulator (LQR) [29], which provides a closed-form feedback control law even for the infinite-horizon case. In LQR, the system dynamics are assumed to be linear,

$$\mathbf{x}_{i+1|k} = \mathbf{A}\mathbf{x}_{i|k} + \mathbf{B}\mathbf{u}_{i|k},$$

and the cost is quadratic,

$$J(\mathbf{x}_{k|k}) = \sum_{i=k}^{\infty} \mathbf{x}_{i|k}^{\top} \mathbf{Q} \mathbf{x}_{i|k} + \mathbf{u}_{i|k}^{\top} \mathbf{R} \mathbf{u}_{i|k},$$

with $\mathbf{Q} \succeq 0$ and $\mathbf{R} \succ 0$. Under these assumptions, the optimal control law takes the form $\mathbf{u}_i = -\mathbf{K}\mathbf{x}_i$, where the gain \mathbf{K} is obtained from the discrete-time algebraic Riccati equation. While elegant, this formulation requires restrictive assumptions: exact linearity, quadratic cost, and the absence of constraints. The Linear Quadratic Gaussian (LQG) [31] controller generalizes this setup to systems subject to Gaussian process and

measurement noise by combining LQR with a Kalman filter for state estimation. However, LQG inherits the same structural limitations, namely the reliance on linear system dynamics and exact model knowledge.

When constraints are incorporated while maintaining linear dynamics and quadratic costs, the finite-horizon MPC problem can be posed as a Quadratic Program (QP). Specifically, for horizon T and at time step k , one obtains

$$\begin{aligned} \min_{\mathbf{u}} \quad & \mathbf{x}_{k+T|k}^\top \mathbf{Q} \mathbf{x}_{k+T|k} + \sum_{i=k}^{k+T-1} \mathbf{x}_{i|k}^\top \mathbf{Q} \mathbf{x}_{i|k} + \mathbf{u}_{i|k}^\top \mathbf{R} \mathbf{u}_{i|k} \\ \text{s.t.} \quad & \mathbf{x}_{i+1|k} = \mathbf{A} \mathbf{x}_{i|k} + \mathbf{B} \mathbf{u}_{i|k}, \\ & \mathbf{x}_{i|k} \in \mathcal{X}, \quad \mathbf{u}_{i|k} \in \mathcal{U}, \quad i = k, \dots, k+T-1, \\ & \mathbf{x}_{k|k} = \mathbf{x}_k, \end{aligned}$$

where \mathcal{X} and \mathcal{U} are required to be polyhedral sets. This QP can be solved efficiently using interior-point or active-set methods, and the resulting controller is often referred to as *Linear MPC*. Such controllers are well suited for applications with moderate state and input dimensions, and have found widespread use in process control [44]. Many more variants like these exist based on favourable assumption that enable reformulation to standard problems of (convex) optimization, please refer to [42, Ch. 2] for more information.

To mitigate model uncertainty and disturbances, robust MPC variants have been proposed [42, Ch. 3]. A prominent example is Tube MPC [26], which computes a nominal trajectory $\{\bar{\mathbf{x}}_{i|k}, \bar{\mathbf{u}}_{i|k}\}_{i=k}^{k+N}$ and defines a feedback policy of the form

$$\mathbf{u}_{i|k} = \bar{\mathbf{u}}_{i|k} + \mathbf{K}(\mathbf{x}_{i|k} - \bar{\mathbf{x}}_{i|k}),$$

where \mathbf{K} is a stabilizing feedback gain. The resulting closed-loop trajectories are guaranteed to remain within an invariant tube

$$\mathbf{x}_{i|k} \in \bar{\mathbf{x}}_{i|k} \oplus \mathcal{E}_{\text{inv}},$$

where \mathcal{E}_{inv} is a disturbance-invariant set and \oplus denotes the Minkowski sum, i.e. $\bar{\mathbf{x}}_{i|k} \oplus \mathcal{E}_{\text{inv}} = \bar{\mathbf{x}}_{i|k} + e \mid e \in \mathcal{E}_{\text{inv}}$. This construction ensures constraint satisfaction even under bounded disturbances. However, Tube MPC depends on an accurate disturbance model, introduces conservatism by shrinking the feasible region, and increases computational cost due to the need for set-based computations.

Despite their tractability, these formulations face important limitations. Real robotic systems are rarely exactly linear, particularly in contact-rich manipulation tasks where the dynamics are hybrid or discontinuous. Differentiable cost structures, although mathematically convenient, exclude many objectives of practical interest, such as max-norm penalties, sparsity-inducing criteria, or discontinuous rewards. Similarly, simple constraint sets like polyhedrons cannot accurately model nonlinear actuator characteristics, torque-speed envelopes, or collision-avoidance constraints, which are inherently non-convex.

Moreover, the resulting controller is only optimal with respect to the chosen specification of the cost function and constraints, which themselves are idealized models of the system and its environment and are often based on heuristics. Any mismatch between the modeled and actual dynamics, unmodeled constraints, or unanticipated disturbances can therefore lead to performance degradation. Additionally, the cost function plays a central role in determining the controller behavior, yet mapping a desired behavior to an appropriate cost function is a complex engineering task, further complicated by restrictions on the functional form. As a result, controllers derived from these formulations are fundamentally limited and may exhibit poor performance or even infeasibility when applied to complex systems.

Sampling-Based MPC Framework

These challenges have motivated the development of alternative formulations that relax some of the analytical and computational requirements of traditional MPC. Sampling-based MPC refers to a family of methods that draw inspiration from statistical physics and information theory, rather than from classical optimal control theory. Instead of explicitly solving a nonlinear optimization problem, these approaches rely on stochastic sampling of control trajectories and determine statistically optimal actions through appropriate estimators. This perspective enables the direct treatment of nonlinear and nonconvex dynamics under stochasticity and uncertainty, and avoids many of the differentiability requirements of gradient-based optimization.

Arguably the most prominent formulations are the Model Predictive Path Integral (MPPI) [54] control and the Information-Theoretic MPC (IT-MPC) [56] frameworks. In this work, we focus on the latter and only sketch out MPPI. We do this because IT-MPC provides a more general theoretical foundation that encompasses MPPI as a special case while requiring fewer assumptions on the system model and cost structure. In the following, all densities are with respect to the Lebesgue measure.

The key idea in sampling-based MPC is to model the control inputs as a random variable

$$\mathbf{x}_{i+1|k} = \mathbf{g}(\mathbf{x}_{i|k}, \mathbf{v}_{i|k}), \quad \mathbf{v}_{i|k} \sim \mathcal{N}(\mathbf{u}_{i|k}, \boldsymbol{\Sigma}_{i|k}), \quad (2.8)$$

where $\mathbf{x}_{i|k} \in \mathbb{R}^n$ over a fixed finite horizon $i \in \{k, \dots, k+T-1\}$ and $\boldsymbol{\Sigma}_{i|k} \in \mathbb{R}^{m \times m}$, $\boldsymbol{\Sigma}_{i|k} \succ \mathbf{0}$. Input and mean control action sequences are denoted

$$\mathbf{V}_k \in \mathbb{R}^{m \times T} = (\mathbf{v}_{k|k}, \dots, \mathbf{v}_{k+T-1|k}) \quad (2.9)$$

$$\mathbf{U}_k \in \mathbb{R}^{m \times T} = (\mathbf{u}_{k|k}, \dots, \mathbf{u}_{k+T-1|k}). \quad (2.10)$$

respectively. Based on this and the assumed covariance, the probability density function of actual control inputs is

$$q(\mathbf{V}_k | \mathbf{U}_k, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{mN}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \sum_{i=k}^{k+T-1} (\mathbf{v}_{i|k} - \mathbf{u}_{i|k})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{v}_{i|k} - \mathbf{u}_{i|k})\right). \quad (2.11)$$

The corresponding measure is denoted $\mathbb{Q}_{\mathbf{U}_k, \boldsymbol{\Sigma}}$. In many applications, $\boldsymbol{\Sigma}$ is obtained by simply stacking T copies $\boldsymbol{\Sigma}_{i|k}$ in a block diagonal matrix. The Gaussian formulation captures two essential aspects while enforcing maximum entropy: the mean represents the nominal trajectory around which the input fluctuates, while the covariance quantifies how much confidence we place in this nominal value for every control input.

This stochastic perspective is advantageous in scenarios with nonlinearities, imperfect models, or contact dynamics, where precise prediction is inherently unreliable. Furthermore, robotic systems typically rely on smoothing via filters and low-level controllers running at very high frequencies, leading to additional uncertainty. This is naturally covered in this setting.

Mathematically, this theoretical framework is based on the duality between the so-called free energy and the relative entropy. The free energy of a cost function $S(x): \mathbb{R}^d \mapsto \mathbb{R}$ with respect to \mathbb{P} at scale $\lambda > 0$ is defined as

$$\mathcal{F}(S, p, \lambda) = \log \left(\mathbb{E}_{x \sim \mathbb{P}} \left[\exp \left(-\frac{1}{\lambda} S(x) \right) \right] \right) \quad (2.12)$$

From an information-theoretic point of view, free energy captures the "effective value" of minimizing the cost $S(x)$ when accounting for uncertainty about the system's state. As such, the base measure \mathbb{P} encodes prior belief about the system state, while λ controls how much we trust this prior.

Intuition: Free Energy

For high λ , we put more trust on \mathbb{P} by blurring the cost landscape and allow the weighting given by the measure to have a larger effect on the expectation. Conversely, low λ puts more emphasize on costs instead of the state. This corresponds to higher entropy on the system state.

Using importance sampling and Jensen's Inequality, we can derive the inequality

$$-\lambda\mathcal{F}(S, p, \lambda) \leq \mathbb{E}_{x \sim \mathbb{Q}}[S(x)] + \lambda\text{KL}(\mathbb{Q}||\mathbb{P}), \quad (2.13)$$

based on another measure \mathbb{Q} , which –in its most general form– is called the Gibbs Variational Principle. The inequality in (2.13) shows that in addition to the definition above, the free energy is the best compromise we can do to minimize the cost while limiting information gain / stay close to the prior. The tradeoff is again guided by the inverse temperature λ .

Intuition: Gibbs Variational Principle

For $\lambda \rightarrow 0$, the cost dominates. We disregard the prior distribution and simply perform hard minimization, selecting the state with the lowest cost. Information gain relative to the base measure does not matter since the measure is not trusted.

For $\lambda \rightarrow \infty$, the prior dominates. The cost is ignored, and minimizing information gain means leaving the base measure unchanged. The result is simply the expected cost under the prior \mathbb{P} .

The Gibbs Variational Principle tells us that if we want to minimize a cost function S by picking a distribution that is focused on low cost states, any \mathbb{Q} that fulfills Equation (2.13) with equality is optimal.

A general approach to derive an algorithm is to assume a certain family of distributions for the prior and the general form of the cost function. Combining this with our form of control input distribution in Equation (2.11), this allows us to determine the form of the optimal distribution $q^*(x)$ that achieves the lower bound in Equation (2.13). While we cannot directly sample from q^* , we can solve

$$\theta^* = \arg \min_{\mathcal{U}} \text{KL}(\mathbb{Q}^*||\mathbb{Q}), \quad (2.14)$$

by iteratively updating its parameters θ based on expectations evaluated via importance sampling with a proposal distribution \mathbb{W} with density w . In most cases the parameter is the mean of a Gaussian distribution while control input constraints can be integrated by clipping, allowing a simple mean update

$$\mathbb{E}_{\mathbb{Q}^*}[\mathbf{V}_k] = \mathbb{E}_{\mathbb{W}}[w(\mathbf{V}_k)\mathbf{V}_k], \quad w(\mathbf{V}_k) = \frac{q^*(\mathbf{V}_k)}{w(\mathbf{V}_k)}. \quad (2.15)$$

IT-MPC and MPPI

The general IT-MPC [55] is based on a state-dependent cost function

$$S(\mathbf{V}_k; \mathbf{x}_k) = C(\mathbf{x}_k, \mathbf{g}(\mathbf{x}_k, \mathbf{v}_{0|k}), \mathbf{g}(\mathbf{g}(\mathbf{x}_k, \mathbf{v}_{0|k}), \mathbf{v}_{1|k}), \dots). \quad (2.16)$$

We assume a prior distribution with density of the form

$$p(\mathbf{V}_k) = q(\mathbf{V}_k | \tilde{\mathbf{U}}_k, \Sigma), \quad (2.17)$$

which enables a closed-form solution of the optimal update distribution,

$$q^*(\mathbf{V}_k) = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda} S(\mathbf{V}_k)\right) p(\mathbf{V}_k), \quad (2.18)$$

where $\lambda > 0$ is a temperature parameter and η a normalization constant. Substituting this into the Gibbs Variational Principle (2.13) reveals that IT-MPC is equivalent to minimizing a stochastic optimal control objective with an implicit quadratic-affine control cost induced by the KL divergence between the proposal and optimal distributions.

Choosing the previous iterate as the proposal distribution, $\mathbb{W} = \mathbb{Q}_{\hat{\mathbf{U}}_k, \Sigma}$, the update law takes the form

$$\mathbf{U}_k = \mathbb{E}_{\mathbf{V}_k \sim \mathbb{Q}_{\hat{\mathbf{U}}_k, \Sigma}}[\mathbf{w}(\mathbf{V}_k) \mathbf{V}_k], \quad (2.19)$$

with importance weights

$$w(\mathbf{V}_k^i) = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda} \left(S(\mathbf{V}_k^i) + \lambda(\hat{\mathbf{U}}_k - \tilde{\mathbf{U}}_k)^\top \Sigma^{-1} \mathbf{V}_k^i\right)\right). \quad (2.20)$$

Some authors also update the covariance based on this via the standard update formula. Thus, IT-MPC iteratively refines the distribution parameters by exponentially reweighting sampled trajectories according to their cost. In practice, expectations are approximated via Monte Carlo samples.

Model Predictive Path Integral control (MPPI) [54] is obtained from the IT-MPC formulation by making the following assumptions:

1. Additive Gaussian control perturbations.

$$\mathbf{V}_k^i = \hat{\mathbf{U}}_k + \boldsymbol{\epsilon}^i, \quad \boldsymbol{\epsilon}^i \sim \mathcal{N}(\mathbf{0}, \Sigma) \text{ i.i.d. over } t, i, \quad (2.21)$$

i.e., candidates are zero-mean perturbations around the current mean $\hat{\mathbf{U}}_k$ with fixed covariance Σ .

2. Uncontrolled Gaussian prior and proposal distribution.

$$p(\mathbf{V}_k) = w(\mathbf{V}_k) = q(\mathbf{V}_k | \mathbf{0}, \Sigma_k). \quad (2.22)$$

With this choice, the regularization term induced by $\text{KL}(\mathbb{Q}_{\mathbf{U}, \Sigma} \| \mathbb{Q}_{\mathbf{0}, \Sigma})$ becomes a quadratic penalty on the mean controls:

$$\text{KL}(\mathcal{N}(\mathbf{U}_k, \Sigma) \| \mathcal{N}(\mathbf{0}, \Sigma)) = \frac{1}{2} \mathbf{U}_k^\top \Sigma^{-1} \mathbf{U}_k + \text{const}. \quad (2.23)$$

Thus a *quadratic control cost is enforced implicitly* by the (zero-mean) prior through the KL term. However, this also means that the sampling covariance Σ is not arbitrary but tied to the control penalty: directions with larger variance correspond to cheaper control effort, while directions with smaller variance correspond to expensive control effort.

In the original paper, the authors rely on arguments from stochastic optimal control and partial differential equations, which require additional assumptions such as smooth dynamics and specific structure of the control costs.

Ultimately, practical versions of both IT-MPC and MPPI arrive at the same update law by incorporating the last iteration, but MPPI has to do this through heuristic manipulation of the result, whereas IT-MPC enables direct incorporation in form of the prior

$$\tilde{p}(\mathbf{V}_k) = p(\mathbf{V}_k | \hat{\mathbf{U}}_k, \Sigma). \quad (2.24)$$

For $\alpha = 0$, this corresponds to the base MPPI algorithm, while $\alpha = 1$ realizes a trust region around the last iterate. The weights are thereby computed via

$$w(\mathbf{V}_k^i) = \frac{1}{\bar{\eta}} \exp\left(-\frac{1}{\lambda} \left(S(\mathbf{V}_k^i) + \lambda(1 - \alpha)(\hat{\mathbf{U}}_k)^\top \Sigma^{-1} \mathbf{V}_k^i\right)\right). \quad (2.25)$$

which additionally makes tuning of the temperature parameter λ easier. In Equation (2.20), setting high temperature values, e.g. when expecting good control authority can result in unstable behavior, because most samples will have vanishing weights. With the new prior, we can work against this by setting higher values for α , balancing smoothness and control cost. Additionally, the formulation itself is globally optimal in the limit of infinite exploration, while MPPI only guarantees convergence to local minima [56].

Key Concept: IT-MPC

The IT-MPC formulation uses minimal assumptions on costs, dynamics and constraints, requiring no smoothness or even continuity. Uncertainty is encoded in a prior distribution of the control input. While this provides flexibility in algorithm design, it comes at the expense of deterministic guarantees, which are replaced by stochastic ones that are susceptible to errors arising from finite-sample Monte Carlo estimation of expectations.

Cross-Entropy Method

The Cross-Entropy Method (CEM) [41] is another sampling-based approach that can be interpreted as a special case of importance sampling. Instead of using exponential reweighting as in IT-MPC, CEM reformulates the objective as a rare-event estimation problem. The optimal distribution is the indicator distribution

$$\tilde{q}^*(\mathbf{V}_k) = \mathbf{1}_{\{S(\mathbf{V}_k) \leq \gamma\}}, \quad (2.26)$$

where γ is a cost threshold and $\mathbf{1}_{\cdot}$ is the indicator function. Intuitively, this distribution assigns probability mass only to the set of “elite” samples whose costs fall below γ .

In practice, the actual numerical value of γ does not matter and it is estimated adaptively at each iteration. Additionally, all distributions are assumed to be Gaussian, yielding closed-form updates. Given N sampled trajectories $\{\mathbf{V}_k^i\}_{i=1}^N \sim \mathcal{N}(\mathbf{U}_k, \Sigma_k)$ with general costs $S^i = S(\mathbf{V}_k^i)$, the elite set \mathcal{E} is defined as the top $E < N$ trajectories with the lowest costs:

$$\mathcal{E} = \{\mathbf{V}_k^i \mid S^i \leq \gamma, i \in \{1, \dots, E\}\}. \quad (2.27)$$

The distribution parameters are then updated by maximum likelihood estimation using only this elite set:

$$\mathbf{U}'_k = \frac{1}{E} \sum_{\mathbf{V}_k^i \in \mathcal{E}} \mathbf{V}_k^i, \quad (2.28)$$

$$\Sigma'_k = \frac{1}{E} \sum_{\mathbf{V}_k^i \in \mathcal{E}} (\mathbf{V}_k^i - \mathbf{U}'_k)(\mathbf{V}_k^i - \mathbf{U}'_k)^\top. \quad (2.29)$$

Unlike IT-MPC, which smoothly reweights all samples, CEM discards suboptimal samples and updates its distribution using only the best-performing trajectories. This focus on elites often improves sample efficiency but can make the method more susceptible to local minima.

Algorithm 1 MPPI

```
1: Initialize  $(\mathbf{U}, \Sigma)$ 
2: for  $k \leftarrow 1$  to  $\dots$  do
3:   Sample trajectories  $\mathbf{V}^i \sim \mathcal{N}(\mathbf{U}, \Sigma)$ 
4:   Roll out dynamics, compute costs  $S(\mathbf{V}^i)$ 
5:   Compute weights  $w^i \propto \exp(-S(\mathbf{V}^i)/\lambda)$ 
6:    $\mathbf{U} \leftarrow \sum_i w^i \mathbf{V}^i$ 
7:    $\Sigma \leftarrow \sum_i w^i (\mathbf{U} - \mathbf{V}^i)^2$ 
8: end for
9: Apply  $\mathbf{u}_0$ 
```

Algorithm 2 CEM

```
1: Initialize  $(\mathbf{U}, \Sigma)$ 
2: for  $k \leftarrow 1$  to  $\dots$  do
3:   Sample trajectories  $\mathbf{V}^i \sim \mathcal{N}(\mathbf{U}, \Sigma)$ 
4:   Roll out dynamics, compute costs  $S(\mathbf{V}^i)$ 
5:   Select elite set  $\mathcal{E} = \{\mathbf{V}^i \mid S(\mathbf{V}^i) \leq \gamma\}$ 
6:    $\mathbf{U} \leftarrow \frac{1}{E} \sum_{\mathbf{V}^i \in \mathcal{E}} \mathbf{V}^i$ 
7:    $\Sigma \leftarrow \frac{1}{E} \sum_{\mathbf{V}^i \in \mathcal{E}} (\mathbf{V}^i - \mathbf{U})^2$ 
8: end for
9: Apply  $\mathbf{u}_0$ 
```

Algorithm 3 PS

```
1: Initialize  $\mathbf{U}$ 
2: for  $k \leftarrow 1$  to  $\dots$  do
3:   Sample trajectories  $\mathbf{V}^i \sim \mathcal{N}(\mathbf{U}, \Sigma)$ 
4:   Roll out dynamics, compute costs  $S(\mathbf{V}^i)$ 
5:   Select best sample  $\mathbf{V}^* = \arg \min_{\mathbf{V}^i} \{S(\mathbf{V}^i)\}$ 
6:    $\mathbf{U} \leftarrow \mathbf{V}^*$ 
7: end for
8: Apply  $\mathbf{v}_0$  // best sample
```

Figure 2.1.: Side-by-side comparison of MPPI, CEM and PS in the notation of Section 2.3. Key differences are highlighted: **sampling strategy**, **weighting vs. elite selection**, and **update rule**.

Predictive Sampling

Another frequently used method is Predictive Sampling (PS). The idea is to simply select the sample with best performance in simulation as the new mean. Afterwards, variations are sampled using a pre-defined covariance at the next iteration. This yields a more aggressive optimizer with high variance and susceptibility to mode switching, yet it can help escaping local minima. [24] and [3] provide theoretical justification for this by framing the update law of IT-MPC as a denoising process. Predictive Sampling thereby provides a maximum likelihood estimate for a Dirac delta at the optimum based on the current samples.

Discussion

Optimization-based and sampling-based formulations of MPC represent two fundamentally different trade-offs between guarantees and flexibility. Classical optimization-based MPC methods, such as LQR, linear MPC, or robust MPC, provide strong theoretical guarantees of stability, feasibility, and, in some cases, optimality. These guarantees are valuable in safety-critical applications. However, they come at the cost of restrictive modeling assumptions in form of linear dynamics and well-behaved cost functions and constraint sets. In practice, these assumptions reduce the controller's ability to capture the complexities of real robotic systems. Contact-rich manipulation is particularly problematic, as it introduces discontinuities, non-smooth dynamics, and amplified model errors. Moreover, integrating nonlinear or learned dynamics models, such as neural

networks, remains challenging within an optimization-based MPC framework due to non-convexity and the lack of tractable solvers.

In contrast, sampling-based MPC relaxes many of these assumptions by evaluating control sequences directly through forward simulations of the system dynamics. This paradigm requires minimal structural constraints on the dynamics or cost function and is therefore well suited for domains with complex physics, uncertainties, or learned models. It also avoids solver-specific issues by relying on stochastic or population-based optimization, making it easier to incorporate discontinuous rewards, nonlinear objectives, or control-space constraints.

Nevertheless, sampling-based methods introduce their own specific challenges. Reliable performance typically requires evaluating a large number of samples at each control step, which can be computationally demanding. Moreover, while optimization-based methods can guarantee stability and feasibility under their assumptions, sampling-based MPC generally provides only probabilistic performance guarantees. Enforcing hard constraints in the state space remains particularly difficult: if infeasible states are penalized, e.g., through infinite costs, sampling efficiency can deteriorate significantly, leading to overly conservative behavior. Furthermore, the controller can become trapped in local minima when the cost landscape is highly non-convex, especially with insufficient sample diversity. Although Information-Theoretic MPC is theoretically globally optimal, achieving this optimality would require perfect evaluation of the expectation, which is impossible with a finite number of samples.

Certain problems are inherent to the MPC paradigm and are shared across all its variants. For instance, rollouts are susceptible to the sim-to-real gap. Additionally, the design of cost and reward functions remains a significant engineering challenge that strongly influences controller behavior and overall performance, regardless of the specific MPC formulation chosen.

2.4. Trajectory Parametrization with Splines

In many optimal control and trajectory optimization frameworks, candidate trajectories are represented explicitly as sequences of states and control inputs at discrete time steps. While this direct representation is simple, it often leads to trajectories that are piecewise-constant or discontinuous in higher-order derivatives, which is undesirable for robotic systems where smoothness is critical for safety, efficiency, and hardware feasibility. For example, sudden changes in velocity or torque can lead to actuator saturation, mechanical stress, or unstable interactions in contact-rich tasks.

A common strategy to mitigate these issues is to represent trajectories using smooth basis functions rather than raw sampled sequences. Among the most popular approaches are spline-based parameterizations, which provide a compact representation of trajectories while enforcing continuity and smoothness by construction. This not only improves the quality of the resulting motions but also reduces the dimensionality of the underlying optimization problem, since the trajectory is described by a smaller set of parameters (e.g., control points) instead of every sample along the horizon.

B-Splines

A widely used approach for trajectory parameterization is based on *B-splines* [40], which allow one to describe a continuous curve using a finite number of control points and basis functions.

Formally, let p denote the degree of the B-spline basis functions and $p + 1$ the corresponding order. A B-spline curve is constructed from a set of $n + 1$ control points $\mathbf{z}_0, \dots, \mathbf{z}_n \in \mathbb{R}^d$ and a non-decreasing knot vector $\mathbf{t} = (t_0, \dots, t_m)$ of length $m + 1$. The relation between these quantities is given by

$$m = n + p + 1. \quad (2.30)$$

The B-spline curve $\mathbf{C}(x)$ is then defined as a linear combination of basis functions and control points,

$$\mathbf{C}(x) = \sum_{i=0}^n N_{i,p}(x) \mathbf{z}_i, \quad (2.31)$$

where $N_{i,p}(x)$ denotes the i -th basis function of degree p .

The basis functions are defined recursively. For degree $p = 0$, the functions reduce to indicator functions on the knot intervals,

$$N_{i,0}(x) = \begin{cases} 1, & \text{if } t_i \leq x < t_{i+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (2.32)$$

and for higher degrees $p \geq 1$, they are given by the recursion

$$N_{i,p}(x) = \frac{x - t_i}{t_{i+p} - t_i} N_{i,p-1}(x) + \frac{t_{i+p+1} - x}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(x), \quad (2.33)$$

with the convention that fractions are set to zero whenever their denominator vanishes. In practice, the evaluation of B-spline curves can be written compactly in matrix form. For a set of pre-determined evaluation points $\{x_0, x_1, \dots, x_m\}$, one pre-computes the non-zero basis function values

$$(\mathbf{N})_{j,i} = N_{i,p}(x_j), \quad 0 \leq j \leq m, \quad 0 \leq i \leq n, \quad (2.34)$$

and stores them in the matrix

$$\mathbf{N} = \begin{bmatrix} N_{0,p}(x_0) & N_{1,p}(x_0) & \dots & N_{n,p}(x_0) \\ N_{0,p}(x_1) & N_{1,p}(x_1) & \dots & N_{n,p}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ N_{0,p}(x_m) & N_{1,p}(x_m) & \dots & N_{n,p}(x_m) \end{bmatrix}. \quad (2.35)$$

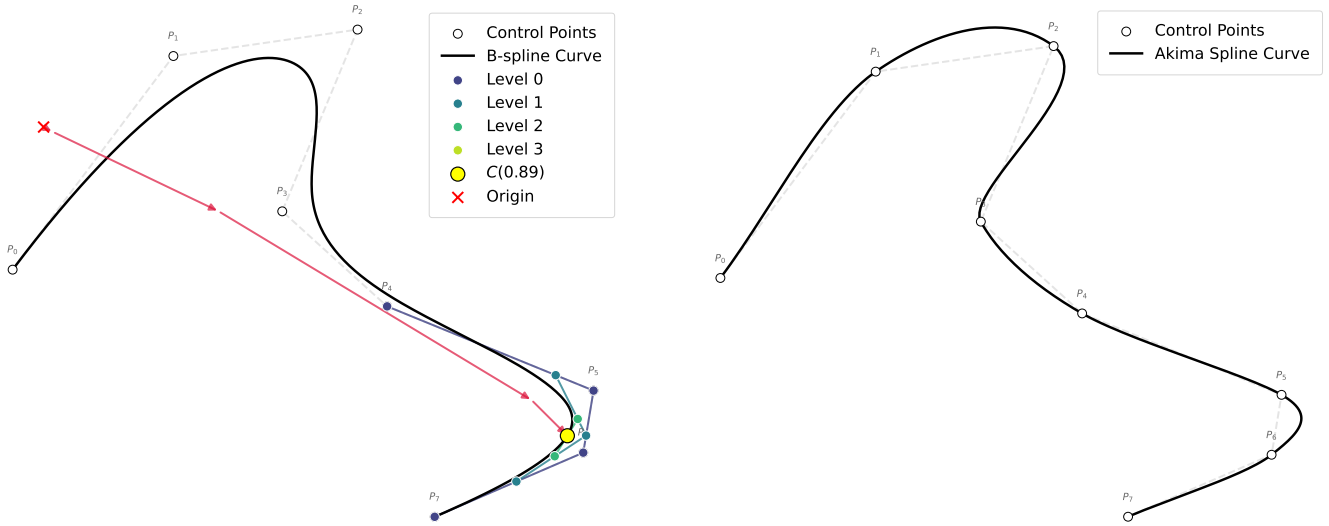
Collecting the control points into the vector

$$\mathbf{P} = \begin{bmatrix} \mathbf{z}_0^\top \\ \mathbf{z}_1^\top \\ \vdots \\ \mathbf{z}_n^\top \end{bmatrix}, \quad (2.36)$$

the evaluation of the curve at all points U can be written as

$$\begin{bmatrix} \mathbf{C}(u_0)^\top \\ \mathbf{C}(u_1)^\top \\ \vdots \\ \mathbf{C}(u_m)^\top \end{bmatrix} = \mathbf{N} \cdot \mathbf{P}. \quad (2.37)$$

In this construction, the control points \mathbf{z}_i serve as decision variables in an optimization problem, while the knot vector \mathbf{t} and the degree p determine the smoothness and flexibility of the resulting curve. B-splines guarantee continuity of derivatives up to order $p - 1$ at the knots, making them particularly well suited for control applications where smooth input signals are required. Efficient evaluation of B-spline curves at varying phase values can be performed using De Boor's algorithm.



(a) B-Spline interpolation of exemplary control points. The levels indicate the recursive linear interpolations used in the Cox-De Boor algorithm, while the red arrows show the linear combination of active control points in Equation (2.31) used to compute the collocation matrix entries.

(b) Akima spline interpolation of the same control points. The piecewise cubic polynomials are determined using weighted slopes, ensuring local control and avoiding oscillations typical of high-order polynomials. Notice how each control point is part of the resulting spline curve.

Figure 2.2.: Comparison of interpolation methods for exemplary control points.

Akima Splines

An alternative to B-splines, which rely on a global knot vector and recursive basis functions, are *Akima splines* [2]. These provide a piecewise cubic interpolation scheme that emphasizes local control and robustness to outliers in the data. The construction is based on a dataset of M knot-control point pairs (t_i, \mathbf{z}_i) , where t_i denotes the knot positions and $\mathbf{z}_i \in \mathbb{R}^d$ the associated control points.

For each interval $x \in [t_i, t_{i+1}]$, Akima splines define a cubic polynomial

$$\mathbf{c}_i(x) = \mathbf{d}_i(x - t_i)^3 + \mathbf{c}_i(x - t_i)^2 + \mathbf{b}_i(x - t_i) + \mathbf{a}_i, \quad (2.38)$$

where the coefficients $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i, \mathbf{d}_i \in \mathbb{R}^d$ are determined from the control points and their slopes.

First, piecewise slopes between consecutive knots are computed as

$$\mathbf{m}_i = \frac{\mathbf{z}_{i+1} - \mathbf{z}_i}{t_{i+1} - t_i}. \quad (2.39)$$

These are then used to define the local spline slopes \mathbf{s}_i by a weighted average of neighboring differences,

$$\mathbf{s}_i = \frac{|\mathbf{m}_{i+1} - \mathbf{m}_i| \mathbf{m}_{i-1} + |\mathbf{m}_{i-1} - \mathbf{m}_{i-2}| \mathbf{m}_i}{|\mathbf{m}_{i+1} - \mathbf{m}_i| + |\mathbf{m}_{i-1} - \mathbf{m}_{i-2}|}. \quad (2.40)$$

The polynomial coefficients are then obtained as

$$\mathbf{a}_i = \mathbf{z}_i, \quad (2.41)$$

$$\mathbf{b}_i = \mathbf{s}_i, \quad (2.42)$$

$$\mathbf{c}_i = \frac{3\mathbf{m}_i - 2\mathbf{s}_i - \mathbf{s}_{i+1}}{t_{i+1} - t_i}, \quad (2.43)$$

$$\mathbf{d}_i = \frac{\mathbf{s}_i + \mathbf{s}_{i+1} - 2\mathbf{m}_i}{(t_{i+1} - t_i)^2}. \quad (2.44)$$

To complete the definition, special handling is required at the endpoints. The boundary slopes are chosen as

$$\mathbf{s}_1 = \mathbf{m}_1, \quad \mathbf{s}_2 = \frac{1}{2}(\mathbf{m}_1 + \mathbf{m}_2), \quad (2.45)$$

$$\mathbf{s}_{M-1} = \frac{1}{2}(\mathbf{m}_{M-1} + \mathbf{m}_{M-2}), \quad \mathbf{s}_M = \mathbf{m}_{M-1}. \quad (2.46)$$

Compared to classical cubic splines, the Akima formulation avoids oscillations in regions with large slope changes, as the interpolation depends primarily on local data rather than global conditions.

2.5. Domain Randomization

Because collecting real-world data and conducting extensive experiments on physical hardware are often costly, time-consuming, or even unsafe, simulation-based training has become an essential component of modern control and learning research. When deploying such algorithms on real-world systems for control, perception, or decision-making tasks, a key challenge arises from the discrepancy between the simulated environment and the physical world. This mismatch, often referred to as the *sim-to-real gap*, can lead to degraded performance or even outright failure of policies trained exclusively in simulation. A widely adopted technique to mitigate this issue is *Domain Randomization* (DR) [34], which deliberately introduces variability in the simulation during training.

The idea is that by exposing the learning agent to a wide range of randomized environmental conditions and system parameters, the resulting policy becomes more robust and generalizes better when transferred to the real world. Instead of fixing the environment parameters, we consider a set of possible parameter tuples

$$\boldsymbol{\xi} \in \Xi,$$

where Ξ denotes the set of all physically plausible parameter combinations of the system. Each $\boldsymbol{\xi}$ corresponds to a specific instance of the simulated environment, characterized by quantities such as masses, friction coefficients, sensor noise levels, initial states, or actuator delays. Training across multiple domains can then be interpreted as optimizing a policy over a finite subset,

$$\{\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots\},$$

with each $\boldsymbol{\xi}_i$ is corresponding to one fixed simulation setting. During training, trajectories are rolled out for each selected environment, and the policy is optimized jointly over the accumulated data. This procedure is

easy to implement and often improves robustness, but it treats the chosen environments as isolated points. There is no systematic notion of neighborhood or similarity between parameters, and thus no principled way to incorporate additional information.

An similar idea, which we use in our SMPC setting instead rolls out the same trajectory over all domains $\{\xi_i\}_{i=1}^D$, and then aggregates the information into one final performance measure. Formally, this requires us to endow Ξ with additional structure—for example, viewing it as a subset of \mathbb{R}^n equipped with a notion of distance (metric) or similarity (kernel), turning Ξ into a parameter space.

The central challenge here is that defining this additional structure is not straightforward and a whole research area in itself: The components of ξ often represent heterogeneous physical quantities such as masses and geometric lengths, each with distinct units and scales. Constructing a single metric / kernel that captures distance / similarity across such mixed quantities is inherently difficult. Furthermore, the system dynamics may depend on ξ in a highly nonlinear fashion, so small parameter perturbations can induce large or discontinuous changes in trajectories.

Once a metric or kernel is chosen, the expected cost under domain randomization can be written, based on the cost definition in Equation (2.3), as

$$J_{k \rightarrow k+N|k}^*(\mathbf{x}_k, h(\xi)) = \mathbb{E}_{\xi \sim h(\xi)} \left[J_{k \rightarrow k+N|k}^*(\mathbf{x}_k) \right], \quad (2.47)$$

where $h(\xi)$ denotes a density over randomized parameters. In practice, this expectation must be approximated from a finite number of samples $\xi_i \sim h(\xi)$, yielding an aggregated estimate

$$\widehat{J}(\mathbf{x}_k) = \mathcal{A}(J^*(\mathbf{x}_k; \xi_1), \dots, J^*(\mathbf{x}_k; \xi_M)), \quad (2.48)$$

where $\mathcal{A}(\cdot)$ is an *aggregation operator* that defines how information from different domains is combined. A variety of aggregation rules can be employed, depending on whether the goal is to emphasize average, worst-case, or risk-sensitive performance.

2.5.1. Adaptive Domain Randomization

Building on this formulation, Adaptive Domain Randomization (ADR) [34, Sec. 5.2] methods update the sampling distribution \mathbb{H} dynamically based on discrepancies between simulated and real-world behavior. The underlying idea is to progressively shift \mathbb{H} toward regions of the parameter space that better explain the observed real-world data, improving the sim-to-real alignment. This can be done based on reweighting, resampling or a combination of both. The nature of the feedback signal used to update \mathbb{H} depends on the information accessible from the physical system. In general, this feedback defines a notion of *similarity* between real and simulated outcomes. Let $\mathcal{O}_{\text{real}}$ denote a set of real-world observations—such as state trajectories, control inputs, or scalar performance values and $\mathcal{O}_{\text{sim}}(\xi)$ the corresponding simulated outcomes generated under domain parameters ξ . We then define a similarity kernel

$$k(\mathcal{O}_{\text{real}}, \mathcal{O}_{\text{sim}}(\xi)) = \exp\left(-\frac{1}{2} d^2(\mathcal{O}_{\text{real}}, \mathcal{O}_{\text{sim}}(\xi))\right),$$

where $d(\cdot, \cdot)$ is a task-dependent distance or discrepancy measure. This kernel quantifies how well parameters ξ reproduce the observed real-world behavior: larger values of k indicate higher consistency between simulation and reality.

Different choices of $d(\cdot, \cdot)$ yield different forms of feedback:

- **State-based feedback** When full or partial state trajectories are available, the discrepancy can be expressed as

$$d_{\text{state}}^2 = \sum_t \|\mathbf{x}_{t+1}^{\text{real}} - \mathbf{g}(\mathbf{x}_t^{\text{real}}, \mathbf{u}_t^{\text{real}}; \boldsymbol{\xi})\|^2, \quad (2.49)$$

where $f(\cdot)$ denotes the simulated dynamics model. The resulting kernel $k_{\text{state}} = \exp(-d_{\text{state}}^2/2\sigma^2)$ assigns higher weight to parameters that produce dynamics close to those observed in the real system.

- **Task-based feedback** When only aggregated task-level performance (e.g., reward, tracking error) is observable, the distance can be defined directly in the reward or cost space as

$$d_{\text{task}} = |J_{\text{real}} - J_{\text{sim}}(\boldsymbol{\xi})|,$$

yielding a kernel $k_{\text{task}} = \exp(-\lambda d_{\text{task}})$ that emphasizes parameters leading to similar task outcomes.

Both cases can be unified through a general *kernel-weighted update rule*

$$\mathbb{H}_{k+1}(\boldsymbol{\xi}) \propto \mathbb{H}_k(\boldsymbol{\xi}) k(\mathcal{O}_{\text{real}}, \mathcal{O}_{\text{sim}}(\boldsymbol{\xi})), \quad (2.50)$$

which effectively re-weights the sampling distribution according to the similarity between real and simulated outcomes. Note that the validity of this approach relies on the simulations ability to generate a separable signal for different $\boldsymbol{\xi}$. This reweighting can be used on its own or to resample new domain parameter combinations, which can be interpreted as a Bayesian posterior when k is viewed as a likelihood function, or as a gradient-free optimization step that shifts mass in \mathbb{H} toward regions of higher agreement.

3. Related Work

The previous chapter introduced representative sampling-based MPC algorithms in form of IT-MPC /MPPI, Predictive Samples (PS) and the Cross-Entropy Method (CEM) as baseline formulations derived from a common stochastic optimal control perspective. These methods form the foundation for a large body of subsequent research that extends, adapts, or hybridizes them to address practical challenges in robotics.

This chapter reviews these developments, emphasizing how prior work has built upon the baseline formulations to improve sample efficiency, robustness, and applicability to complex robotic systems. We begin by outlining advances within sampling-based MPC itself, followed by approaches leveraging full physics simulators for parallelized rollouts, and conclude with works that integrate online domain randomization to improve generalization.

Sampling Based Model Predictive Control

Sampling-based or zero-order optimization has a long history under various names such as Bayesian Optimization [15] and Evolutionary Strategies [53]. These methods share the property of requiring no gradient information and only minimal assumptions about the objective, making them broadly applicable. In robotics, zero-order methods have been employed for tasks ranging from control and sensor design to morphological optimization [12].

In recent years, the widespread adaptation of deep learning based methods in most engineering disciplines has led to highly capable GPUs being widely available. With the potential for easy parallelization of sampling based MPC we outlined in Section 2.3, this has led to the emergence of sampling based control as an active field of research, particularly in robotics. Vlahov et al. [50] provide a generic C++/CUDA implementation of several sampling-based MPC algorithms, enabling real-time planning with custom dynamics models. Similarly, STORM [5] extends MPPI to robotic control by sampling trajectories in both joint and task space and evaluating them via handcrafted physics rollouts. Using experimental setups like these, a plethora of works explore extensions of the basic MPPI/IT-MPC and CEM algorithms. For MPPI, [49] propose to generate temporally correlated Gaussian noise by sampling in the frequency domain. Using a Fourier identity, they efficiently produce colored noise trajectories through scaled inverse FFTs. The authors show that this can improve sample quality, particularly in tasks like autonomous driving or flight, where decorrelated, high-frequency noise can cause unstable behavior. Similarly, Pinneri et al. [41] use the same idea for the Cross-Entropy Method (CEM), further combining the correlated noise with population decay and an elite-memory mechanism for enhanced sample efficiency. Trevisan et al. [47] exploit the flexibility in choosing a cost function for the derivation in Section 2.3. Their adapted derivations follow exactly the same arguments, but replaces the Gaussian prior in Equation (2.18) with

$$\tilde{S}(\mathbf{V}_k) = S(\mathbf{V}_k) + \lambda \log \left(\frac{p(\mathbf{V}_k)}{q_s(\mathbf{V}_k)} \right). \quad (3.1)$$

yielding a new importance sampling procedure with update

$$\mathbb{E}_{\mathbb{Q}^*}[\mathbf{V}_k] = \mathbb{E}_{\mathbb{Q}_s}[\mathbf{w}(\mathbf{V}_k)\mathbf{V}_k], \quad \mathbf{w}(\mathbf{V}_k) = \frac{1}{\eta} \exp\left(-\frac{1}{\lambda}S(\mathbf{V}_k)\right) \quad (3.2)$$

that allows for arbitrary sampling distributions. However, this introduces a bias towards this distribution, whose interpretation and performance is highly task-dependent.

Wang et al. [51] uses a different measure of statistical similarity between prior and posterior in Equation (2.13) via the Tsallis divergence, allowing for additional parameter shaping of the likelihood function compared to Equation (2.18). The authors show that this can be advantageous to control risk sensitivity. As mentioned in Section 2.3, the minimal assumptions on IT-MPC also limit the theoretical guarantees that can be made. Theoretical work therefore tends to balance adding as few assumptions as possible that lead to stronger guarantees. For example, Yi et al. [60] provide convergence guarantees for MPPI under certain cost-function assumptions, deriving an optimal covariance update. However, in practice the computational cost requires offline covariance estimation via a nominal controller. The analysis also assumes unconstrained state and action spaces. Lambert et al. [25] reformulate MPC as a Bayesian inference problem over policy parameters, deriving updates via Stein Variational Gradient Descent (SVGD), thereby allowing the use of gradient information when available. [4] uses the stochastic nature of MPPI to estimate a safety region at every step, which is then integrated into the objective of an optimization based MPC controller via convex hull construction, albeit only being tested on toy examples. Regarding the performance drawbacks of SMPC, Homburger et al. [20] shows that the suboptimality of sampling based approaches grows quadratically with sampling variance, analysis only for smooth and unconstrained case.

More recently, incorporating generative models into the sampling approach has become an interest. Bruder-muller et al. [7] replaces the sampling distribution with a generative model conditioned on the state history that is trained offline based on data generated from baseline SMPC algorithms. Kurtz et al. [24] does this in a more structured way by alternating between training and data collection and mixing samples from a base implementation and the generative model. Additionally, their generative model is conditioned on the current state and mean update.

Full-Simulator-Based Sampling MPC

For modeling the dynamics, the methods above rely on hand-crafted models. While effective for well-understood systems, such models are difficult to design and scale to complex or high-dimensional dynamics, requiring expert domain knowledge. Meanwhile, the demand for large-scale data in e.g. reinforcement learning [45] has driven the development of highly parallel physics simulators such as Brax [16], MuJoCo MJX and Warp [46], and Isaac Gym [30].

CPU-based simulation used in earlier works such as [22] already greatly simplifies model design by offloading it to the physics engine, but performance is limited by the number of available CPU cores and confined to simulation environments, where sub-real-time operation is acceptable. However, with the recent GPU-accelerated implementations of these simulators, the use of full physics simulation for real-time sampling-based MPC has become practically feasible. Pezzato et. al. [39] use GPU accelerated Isaac Gym to simulate rollouts at scale, enabling real world experiments with planning frequencies of up to 25 Hz. Additionally, the authors introduce tuning of the inverse temperature by keeping track of the normalization constant to balance

exploration and exploitation by tuning it online to stay in $[\eta_{\min}, \eta_{\max}]$ via

$$\lambda_{k+1} = \begin{cases} 0.9\lambda_k, & \text{if } \eta > \eta_{\max}, \\ 1.2\lambda_k, & \text{if } \eta < \eta_{\min}, \\ \lambda_k, & \text{else.} \end{cases} \quad (3.3)$$

Multi-Modal MPPI [62] uses a similar setup and keeps track of multiple separate control input sequences to allow switching between strategies. However, it remains unclear how to avoid mode collapse or how to combine multiple strategies into a next action in a meaningful way.

With DIAL-MPC, Xue et al. [58] reinterpret MPC updates as denoising steps within a score-based diffusion process. The method introduces covariance annealing across both time steps and algorithm iterations to gradually shift from exploration to exploitation. Specifically:

a) Trajectory-level Annealing

The covariance is progressively reduced across distribution updates to promote broad exploration in early iterations and convergence in later ones:

$$\det(\Sigma_{k:k+T}^i) \propto \exp\left(-\frac{\tilde{N}-i}{\kappa_1 N} Tm\right), \quad i = \tilde{N}, \dots, 1,$$

where m is the control dimension and \tilde{N} is the number of iterations per planning step.

b) Action-level Annealing

The covariance for actions further in the horizon is increased to encourage exploratory behavior in the distant future while maintaining stability for near-term actions:

$$\det(\Sigma_{k+t}^i) \propto \exp\left(-\frac{T-t}{\kappa_2 T} m\right), \quad t = 0, \dots, T.$$

To fully realize the benefits of this double-annealing strategy, multiple refinement iterations of the same planning step are required. Consequently, computation times can limit the control frequency in dynamic scenarios such as quadruped jumping on a box. Inspection of the source code [57] also revealed highly customized solver settings, which effectively turn off slipping in favor of fast normal contact resolution, which only works for scenarios without sliding friction effects.

To further simplify setting up a sampling-based MPC pipeline, Hess et al. [19] combine basic MPPI rolled out on the GPU via MuJoCo with VLA created cost functions to drive a tendon driven hand. To achieve this, they provide a task description in natural language and videos of the rollouts to the VLA.

Online Domain Randomization

Despite the simpler design and computational advantages of GPU-based simulators, most of the mentioned works assume a fixed and well-calibrated dynamics model. In practice, however, modeling errors and unmodeled disturbances can significantly degrade controller performance when transferring from simulation to reality. To mitigate this gap, several recent approaches integrate *domain randomization* into sampling-based MPC frameworks, aiming to improve robustness and generalization across varying environments and system parameters.

Pezzato et al. [39] incorporate online domain randomization into their GPU-accelerated Isaac Gym framework by running an unspecified number of randomized environments in parallel and aggregating rollouts via the arithmetic mean of the cost. More formally, Abraham et al. [1] provide a theoretical formulation showing that the Information-Theoretic MPC framework naturally extends to randomized domains by introducing an additional expectation over the parameter distribution:

$$-\lambda\tilde{\mathcal{F}}(S, p, \lambda, h) \leq \mathbb{E}_{h(\boldsymbol{\xi})} \left[\mathbb{E}_{x \sim \mathbb{Q}}[S(\mathbf{V})] + \lambda \text{KL}(\mathbb{Q} \parallel \mathbb{P}) \right], \quad (3.4)$$

where $h(\boldsymbol{\xi})$ denotes the distribution over randomized domain parameters. This formulation effectively averages the free-energy objective across multiple environments, promoting solutions that generalize beyond a single dynamics model. However, the authors do not provide implementation details, and no open-source code is available. It is also worth noting that both studies evaluate only on relatively simple manipulation tasks, such as sliding a rectangular block to a target position in a straight line. For more complex scenarios, including drawer opening or tilting, results are reported only for successful trials and presented primarily as proof-of-concept demonstrations.

4. Methodology

The following chapter briefly introduces the MTP algorithm introduced by Le et al. [28] we want to test against the baselines from Section 2.3.

4.1. Model Tensor Planning

MTP combines elements of CEM, MPPI and PS with a global sampling technique to enforce exploration. The idea is to add smooth white noise trajectories to enforce exploration in a systematic way, working around the local minima issues the algorithms discussed in Section 2.3 face.

Global Graph Samples

In addition to the local samples obtained with the Gaussian formulation in Equation (2.21), MTP mixes global samples obtained via a simple graph sampling strategy. Based on a graph of M layers, each consisting of N uniform samples from control space \mathcal{U} . Edge set is formed by forward pair-wise connections between layers

$$\{(\tilde{\mathbf{v}}_i, \tilde{\mathbf{v}}_{i+1}) | \forall \tilde{\mathbf{v}}_i \in L_i, \tilde{\mathbf{v}}_{i+1} \in L_{i+1}, 1 \leq i \leq N\},$$

where $L_i = \{\tilde{\mathbf{v}}_j \in \mathcal{U} | \tilde{\mathbf{v}}_j \sim \mathcal{U}\}_{j=1}^N$. To sample paths, we can simply sample an integer between 1 and N for a tensor of shape (B, M) and then identify the associated vertex in each layer. Based on this, both the Akima and B-Splines shown in Section 2.4 can be used to interpolate these control points to any resolution.

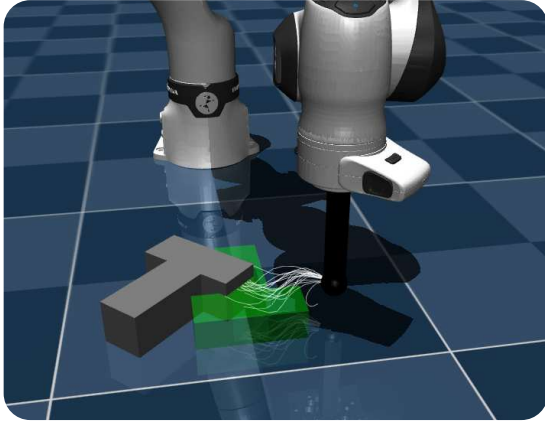
MTP uses elites as in CEM and then applies a performance-based weighting as in MPPI. Additionally, the best sample is rolled out as in PS, compare Algorithm 4. The idea is that near optima, the more structured MPPI samples dominate the elites and guide the process in a stable way, ignoring the global samples, resulting in a behavior that is similar to MPPI. Near small local minima, these global samples can introduce samples that pull the algorithm out of minima, compare Figure 4.1. This balance is subject to hyperparameter tuning, namely β , Σ as well as the graph parameters N , M and spline parameters.

Algorithm 4 MTP

```
1: Initialize  $(\mathbf{U}, \Sigma)$ 
2: for  $k \leftarrow 1$  to  $K$  do
3:   Sample tensor paths  $\tilde{\mathbf{V}}^G \sim \mathcal{G}_{M,N}$  + local  $\mathbf{V}^L \sim \mathcal{N}(\mathbf{U}, \alpha \Sigma)$ 
4:   Interpolate  $\tilde{\mathbf{V}}^G$  (B-spline/Akima)
5:    $\mathbf{V} \leftarrow [\mathbf{V}^G; \mathbf{V}^L]$ 
6:   Roll out dynamics, compute costs  $S(\mathbf{U}^i)$ 
7:   Select elite set  $\mathcal{E}$ , weights  $w^i \propto \exp(-S^i/\lambda)$ 
8:    $\mathbf{U} \leftarrow \sum w^i \mathbf{V}_{\mathcal{E}}^i$ 
9:    $\Sigma \leftarrow \max(\sum w^i (\mathbf{U} - \mathbf{V}_{\mathcal{E}}^i)^2, \Sigma_{\min})$ 
10: end for
11: Apply  $\mathbf{v}_0^*$  // best sample
```

Key Concept

MTP mixes MPPI-styled sampling with maximum entropy global samples and evaluates an elite fraction. This realizes an exploration-exploitation tradeoff: Local sampling exploits the best known sample by trying out small variations, while the global samples allow quick mode switching to escape from local minima.



(a) Samples generated by MPPI.

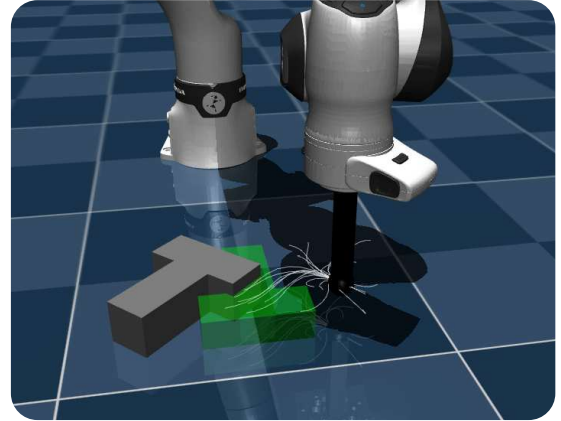
(b) Samples generated by MTP with $\beta = 0.35$.

Figure 4.1.: Comparison of MPPI and MTP samples in the simulated Push T task.

The original MTP paper [28] based its update rule on heuristics. However, we can easily integrate it into the IT-MPC framework of Section 2.3. Based on the introduction above, we see that the update rule uses the sample weighting of Biased MPPI [47] in form of Equation (3.2). Additionally, in contrast to CEM, a softmax weighting of elites is used. For this thesis, we evaluated various update weighting regiments, which always resulted in performance degradation. Continuing the arguments from Section 2.3 and Biased MPPI, it is easy to see why:

MTP uses a mixing strategy controlled via β . Since the expectation is linear, we can split the sample contribution.

Naively using a CEM update yields

$$\mathbb{E}[(1 - \beta)\mathbf{V}_{\text{Local}} + \beta\mathbf{V}_{\text{Global}}] = (1 - \beta)\mathbb{E}[\mathbf{V}_{\text{Local}}] + \beta\mathbb{E}[\mathbf{V}_{\text{Global}}] \quad (4.1)$$

$$= \frac{(1 - \beta)}{E_{\text{Local}}} \sum_{\mathbf{V}_{\text{Local}}^i \in \mathcal{E}} \mathbf{V}_{\text{Local}}^i + \frac{(\beta)}{E_{\text{Global}}} \sum_{\mathbf{V}_{\text{Global}}^i \in \mathcal{E}} \mathbf{V}_{\text{Global}}^i, \quad (4.2)$$

where $E_{\text{Local}}, E_{\text{Global}}$ denote the number of elites from the respective sampling technique. This means the sampling biases the result towards the expectation of the graph samples in accordance to the arguments of Biased MPPI [47], which is the centre of the control set. On the other hand, using a softmax weighting as in MPPI results in

$$\frac{(1 - \beta)}{\lfloor (1 - \beta)N \rfloor} \sum_{i=0}^{\lfloor (1 - \beta)N \rfloor} w(\mathbf{V}_{\text{Local}}^i) \mathbf{V}_{\text{Local}}^i + \frac{(\beta)}{\lceil \beta N \rceil} \sum_{j=\lfloor (1 - \beta)N \rfloor}^N w(\mathbf{V}_{\text{Global}}^j) \mathbf{V}_{\text{Global}}^j. \quad (4.3)$$

While this can discover modes far away from the current mean, it suffers from the same issues MPPI does: incorporating all samples into the update results in slow adaptation to a few good samples, making it prone to get stuck in local minima and slow to react. In essence, the global samples add a white noise floor and inflate the variance. Using the weighted elite samples as in [28] mitigates this effect by only allowing global samples to affect the update if they outperform the local MPPI samples. If this happens, the algorithm greedily changes its mean, depending how much better the global sample performed and inflates the variance until the MPPI samples dominate again.

4.1.1. Extensions

To explore possible improvements to the base MTP implementation, we add the following capabilities:

Rolling Arrays

Currently all algorithms evaluated in [28] assume zero-order hold, which means they only roll out the first control input of the updated mean / best sample. In the evaluated simulation tasks this is no restriction, because the sub-realtime rollouts in simulation enable high enough frequencies on all the tasks. However, we expect to lower the frequency for real system rollouts. This can be problematic, since the base implementation would repeatedly apply the first action, even if the sample was chosen based on the premise that the input varies according to the rolled out sample. To mitigate this, we roll the elites / mean forward, cutting off initial timesteps and adding either random ones to the end or repeating the last one.

Improving Sample Smoothness

While the global samples in MTP are smooth by construction, the locally sampled trajectories used by MPPI and CEM do not impose any temporal regularity and can therefore exhibit high-frequency jitter. To mitigate this, we implement two smoothing strategies for the local samples. First, we parameterize local control sequences using the same spline-based approximation as for the global component, which enforces temporal coherence directly at the level of the sampling distribution. Alternatively, we apply a lightweight Savitzky–Golay [43] filter to the sampled control sequences as a post-processing step.

AnMTP - Annealing of the β -Mixing

In MTP, the exploration–exploitation trade-off is governed by the mixing coefficient β , which controls the relative contribution of the global (exploratory) and local (exploitative) sampling components. In the original implementation, β is fixed for the duration of an episode, which forces the same exploration level in early, mid, and late stages of the task. We therefore introduce an annealed variant, AnMTP, that allows β to vary online. Intuitively, higher exploration is desirable when the optimizer is far from a good mode (e.g., when escaping local minima), whereas more exploitative sampling is preferable once a promising strategy has been identified.

4.2. Domain Randomization

Our implementation evaluates all candidate control trajectories across a fixed number of parallel domains. For each trajectory, this yields a set of per-domain costs, which we aggregate into a single scalar objective using an aggregation operator as in Section 2.5.1. Concretely, we support worst- and best-case aggregation, weighted expectations with adjustable domain weights, and percentile-based criteria to interpolate between risk-neutral and risk-averse behavior.

Directly comparing real-world and simulated trajectory costs is impractical due to the sim-to-real gap. Instead, we use state-based feedback at replanning times: for the selected action sequence, we predict the object pose at the next replanning time under each domain and compare this prediction to the subsequently observed pose. This requires a fixed planning frequency and a consistent mapping between planning steps and simulation time. During rollouts (Algorithm 4), we therefore store the predicted future object poses at the expected replanning time for all domains. Note that only MTP and PS can be combined with these strategies, as CEM and MPPI may execute an action sequence that was not explicitly rolled out in simulation.

We quantify the prediction–observation discrepancy using a standard rigid-pose error that combines translational and rotational components with tunable weights. Let $\hat{\mathbf{T}}_{k+1}^{(d)} = (\hat{\mathbf{R}}_{k+1}^{(d)}, \hat{\mathbf{p}}_{k+1}^{(d)}) \in \text{SE}(3)$ denote the predicted object pose at the next replanning time under domain d , and let $\mathbf{T}_{k+1} = (\mathbf{R}_{k+1}, \mathbf{p}_{k+1})$ denote the observed pose. We compute

$$e_{\text{pos}}^{(d)} = \left\| \hat{\mathbf{p}}_{k+1}^{(d)} - \mathbf{p}_{k+1} \right\|_2, \quad e_{\text{rot}}^{(d)} = \theta \left(\hat{\mathbf{R}}_{k+1}^{(d)} \mathbf{R}_{k+1}^\top \right),$$

where $\theta(\mathbf{R}) = \arccos\left(\frac{\text{trace}(\mathbf{R})-1}{2}\right)$ is the geodesic rotation angle on $\text{SO}(3)$. The final scalar mismatch is

$$e^{(d)} = w_{\text{pos}} e_{\text{pos}}^{(d)} + w_{\text{rot}} e_{\text{rot}}^{(d)},$$

with $w_{\text{pos}}, w_{\text{rot}} \geq 0$ controlling the relative importance of translation and rotation.

We use this discrepancy to adapt the domain distribution online. Specifically, at each replanning step we compute the per-domain pose errors $\{e^{(d)}\}_d$ between the previous-step prediction and the current observation, convert these errors into scores via a scaling, and update domain weights using a softmax. To avoid degenerate updates when domains make nearly identical predictions, we only recompute weights when the inter-domain variation in predicted object pose exceeds a threshold. In addition, we apply exponential smoothing to the weight updates to reduce sensitivity to observation noise and occasional outlier steps.

Beyond weight updates, we also provide optional mechanisms to resample domain parameters based on (i) an evolutionary-style update rule and (ii) fitting a Gaussian to the current weighted domain samples, which can be used to shift the domain proposal distribution over time.

5. Experimental Setup

To assess the effectiveness of the proposed approach, experiments are conducted first in a controlled simulation environment and subsequently on a real robotic platform. This section details the corresponding experimental setup, including the hardware and software components, the task definitions, and the data collection and evaluation procedures used in both domains.

5.1. Parallelized Planning Framework

Modern sampling-based MPC methods require evaluating hundreds to thousands of candidate control sequences per replanning step, making parallel rollout execution the main computational bottleneck in practice. In this thesis, we therefore build a planning stack that keeps the full SMPC loop—sampling, batched simulation, cost evaluation, and distribution update—on accelerator hardware. The core idea is to express the planner in a form that can be compiled and vectorized end-to-end: JAX provides composable program transformations, while MJX provides a JAX-compatible physics simulator so that rollouts can be executed as large batched array programs. The remainder of this section introduces these components and motivates the design constraints that arise from JIT compilation, which directly influence how we implement and scale the planning pipeline.

5.1.1. JAX

JAX [6] is a Python-based numerical computing library that combines a NumPy-compatible API with support for just-in-time (JIT) compilation, automatic differentiation, and parallel execution. Built on top of XLA (Accelerated Linear Algebra), JAX compiles array operations into optimized, low-level code that can be executed on CPUs, GPUs, or TPUs.

Central to JAX is a system of *function transformations*, which are higher-order operators that modify Python functions to enable new behaviors. The most commonly used transformations are:

- `grad`: computes gradients of scalar-valued functions with respect to their inputs, supporting both forward- and reverse-mode automatic differentiation.
- `jit`: traces a Python function and compiles it into an optimized XLA executable. This is done by fusing many subsequent array operations into a single GPU kernel, allowing intermediate results to stay on-chip to optimize memory traffic and kernel launch overhead.
- `vmap`: automatically vectorizes a function across a batch dimension.
- `pmap`: distributes computations across multiple devices (e.g., multiple GPUs or TPU cores) for large-scale parallel training or simulation.

Listing (5.1) Python function.

```
def f(x):
    return 4*x**3 + 3*x**2 + 2*x + 1
```

Listing (5.2) Resulting jaxpr.

```
{ lambda ; a: f32 []. let
  b: f32 [] = integer_pow [y=3] a
  c: f32 [] = mul 4.0: f32 [] b
  d: f32 [] = integer_pow [y=2] a
  e: f32 [] = mul 3.0: f32 [] d
  f: f32 [] = add c e
  g: f32 [] = mul 2.0: f32 [] a
  h: f32 [] = add f g
  i: f32 [] = add h 1.0: f32 []
in (i, ) }
```

Figure 5.1.: Comparison of a simple Python function and its jaxpr, obtained via `jax.make_jaxpr(f)`.

Importantly, these transformations are composable; for instance, one can compute gradients through a just-in-time compiled function, or vectorize the evaluation of a gradient across many inputs.

The JIT compilation process relies on *tracing*. When a function is wrapped with `jit`, JAX does not execute the function directly. Instead, it runs a special trace using example inputs, recording the sequence of primitive operations applied to the arrays. This trace is lowered into JAX's intermediate representation called `jaxpr`, a symbolic, purely functional expression of the computation, exemplified in Figure 5.1. The `jaxpr` is then passed to XLA, which compiles it into an optimized executable for the target hardware. Subsequent calls with inputs of the same shape and type reuse the compiled executable, amortizing the cost of tracing and compilation.

An important abstraction in this workflow is the use of *pytrees*, JAX's general data structure for organizing arrays within nested Python containers such as lists, tuples, or dictionaries. During tracing, *pytrees* are flattened into a canonical representation suitable for compilation, and after execution results are automatically reassembled into the original structure. This mechanism allows JAX to handle transformations of hierarchical inputs and outputs such as nested model parameters or trajectory data.

In order to be able to fuse and optimize large code hierarchies into GPU kernels, JAX imposes several constraints that can complicate its use in practice. First, functions must be expressed in a functional, side-effect-free style to be compatible with JAX transformations such as `jit` or `grad`. This programming paradigm requires users to avoid common Python constructs such as in-place updates, dynamic control flow, or side effects such as I/O within transformed functions. For many cases like if-else constructs, JAX provides its own optimized implementations, but these often inherit system-side challenges from parallel execution on GPUs.

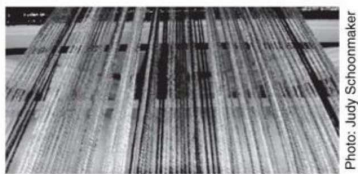


Photo: Judy Schoonmaker

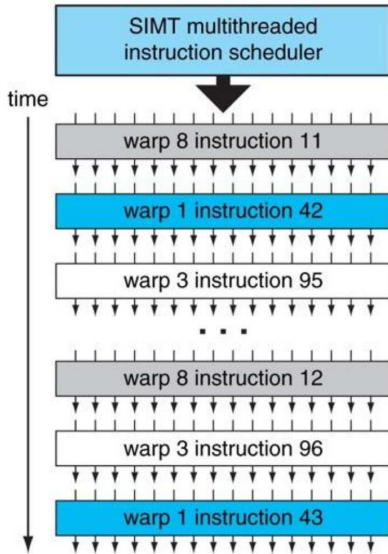


Figure 5.2.: Illustration of warp execution on a GPU, Figure taken from [38].

Modern GPUs group threads into units called *warps*, where multiple threads execute the same instruction in lockstep (Figure 5.2). This model is highly efficient for uniform, data-parallel workloads but struggles with

branching control flow. When different threads within a warp follow different execution paths (e.g., due to an if-statement whose condition varies across data elements), the warp must serialize the divergent branches, resulting in under-utilization of compute resources. This means that threads within a warp that do not satisfy the branch condition remain inactive while the other branch is executed. Even for a simple binary if-else statement with equal branch lengths, effective utilization can be reduced by 50%.

In addition, the requirement that inputs maintain consistent shapes and data types across calls to a jit-compiled function can reduce flexibility in some applications. Handling dynamically sized data often requires padding or specialized batching strategies. Should any traced shapes or types change during execution, this triggers recompilation of the entire function, which can take several minutes. Moreover, debugging compiled jit functions is less transparent than debugging standard Python code, since execution happens within an optimized XLA kernel rather than line-by-line in the interpreter.

5.1.2. Physics Simulation with Mujoco and MJX

MuJoCo [46] (Multi-Joint dynamics with Contact) is a physics engine for simulating articulated rigid-body systems with contact interactions. Since this thesis relies on contact-rich rollouts for sampling-based planning, we focus on the parts of MuJoCo that most strongly affect rollout behavior and performance: the soft-constraint contact model, friction modeling, and the basic stepping interface.

A key modeling choice in MuJoCo is to treat contacts and other constraints as *soft* constraints that generate forces based on constraint violation, rather than enforcing hard position-level constraints. This leads to the following dynamics form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} + \mathbf{J}(\mathbf{q})^\top \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}) \quad (5.1)$$

Here, $\mathbf{M}(\mathbf{q})$ is the generalized mass matrix, $\mathbf{c}(\mathbf{q}, \dot{\mathbf{q}})$ collects bias terms (e.g., Coriolis and gravity), and $\mathbf{J}(\mathbf{q})^\top \mathbf{f}(\cdot)$ maps constraint/contact forces into generalized coordinates. Forward dynamics are computed in continuous time to obtain accelerations, which are then integrated numerically to update the state.

A central computational step is therefore the evaluation of the constraint force $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau})$, which depends on collision detection and the constraint solver.

A typical workflow constructs an `mj.Model` from a MuJoCo XML description and instantiates an `mj.Data` object holding the simulation state. Simulation then proceeds by repeated calls to

$$\text{mj.step}(\text{mj_model}, \text{mj_data}) \mapsto \text{mj_data}'$$

which advances the state by one simulation step. In this work, contact resolution and friction modeling are particularly important, and we summarize both next.

Soft Contact Modelling

Constraint forces are based on a residual r , usually generated by two bodies colliding, which causes small penetrations after updating the discrete positions in the previous solver step. This residual is then used to update constraint-free accelerations a_0 with a reference acceleration

$$a + d(r) \cdot (-a_{\text{ref}}) = (1 - d(r)) \cdot a_0. \quad (5.2)$$

Different behaviors can be modeled by tweaking the `solref` and `solimp` parameters. Additionally, a `margin` and `gap` determine when a constraint start to generate forces, i.e. when the normal distance (negative residual) is below `margin - gap`. The corrective reference acceleration is modeled as a spring-damper system via the two `solref` parameters `timeconst` and `dampratio`. The former controls the decay of the residual, higher values correspond to softer constraints, while very low values model stiff contacts. The latter is usually left to 1 for critical damping, but can be altered for springy behaviors. The influence of the resulting corrective acceleration is then controlled via the impedance $d(r) \in (0, 1)$. For $d \rightarrow 1$, the reference acceleration completely takes over, while $d \rightarrow 0$ deactivates the constraints effects.

The response behavior can be further shaped via 5 parameters (`d0`, `dwidth`, `width`, `midpoint`, `power`) of the `solimp` option. The first 3 values are used to define smooth variation of the impedance over the residual $[0, \text{width}]$ with

$$d(0) = d_0, \quad d(\text{width}) = d_{\text{width}}.$$

The remaining two shape a sigmoid function, with `midpoint` delaying the initial slope towards higher residual values.

Friction Modelling

The forces a contact point can create depend on the `condim`, where 1 corresponds to normal forces only, 3 adds tangential, 4 torsional friction around the contact normal and 6 rolling friction around tangential directions. Additionally, a global `cone` parameter sets the geometry of the friction model. `elliptic` friction cones model the true Coulomb model using a smooth circular cone

$$\mathcal{K}_{\text{elliptic}} := \left\{ \mathbf{f} \in \mathbb{R}^n : f_1 \geq 0, f_1^2 \geq \sum_{i=2}^n \frac{f_i^2}{\mu_{i-1}^2} \right\},$$

and friction coefficients μ_i , resulting in true isotropic behavior. This means the total contact force $(f_{\text{normal}}, f_{t_1}, f_{t_2})$ can be simply expressed in the identity basis

$$\mathbf{E}_{\text{elliptic}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

However, the second order structure of conic constraints, albeit still convex, are numerically more involved than simple linear inequalities, which is why MuJoCo offers an alternative in `pyramidal` cones. These replace the ℓ_2 -norm constraint with linear inequalities

$$\mathcal{K}_{\text{pyramidal}} := \left\{ \mathbf{f} \in \mathbb{R}^{2(n-1)} : f_i \geq 0 \right\}.$$

To enable this reduction to first order constraints, they rely on the entangled per-direction basis

$$\mathbf{E}_{\text{pyramidal}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \mu & -\mu & 0 & 0 \\ 0 & 0 & \mu & -\mu \end{bmatrix}.$$

MuJoCo provides a global `impratio` parameter that controls the friction-to-normal impedance ratio, allowing users to prioritize slip resistance over penetration resistance or vice versa. For elliptic cones, `impratio` has a clean interpretation: it uniformly scales friction impedance across all tangential directions while leaving normal impedance fixed. This can provide a simple strategy to tune overall system behavior without considering per geometry advanced options. In contrast, pyramidal cones inherently mix normal and frictional components within each basis vector (e.g., each edge direction has both a normal component of 1 and a tangential component of $\pm\mu$), making `impratio` scaling ambiguous and direction-dependent. Consequently, non-default `impratio` values with pyramidal cones can degrade both numerical stability and physical accuracy, as the scaling affects the normal force behavior in un-intuitive ways.

MJX

In this work, we also use a JAX-based reimplementation of the MuJoCo engine called MJX that is compatible with JAX transformations such as `jit` and `vmap`, allowing simulation routines to be compiled and batched efficiently. At its core, MJX relies on two key abstractions:

- `mjx.Model`: a lightweight, immutable container of system parameters (geometry, joint definitions, actuator models, etc.). It is conceptually equivalent to the `mjModel` structure in MuJoCo but reformulated as a JAX pytree. This makes it compatible with transformations such as `jit` and `vmap`.
- `mjx.Data`: a mutable state container based on `mj.Data`, reinterpreted in MJX as a purely functional object. State updates are expressed as side-effect-free transformations of this object.

In the beginning, a `mjx.Data` instance is generated from a regular `mujoco` instance. The update step follows the same logic as base MuJoCo. This design principle called *train-state pattern* is particularly advantageous in the context of JAX, since it is well suited for the pure functional paradigm: all states are passed explicitly as data, can be transformed by pure functions, and returned as updated states. Because both model and data are JAX pytrees, this stepping routine can be compiled with `jit` and / or vectorized with `vmap` across multiple data instances in parallel. This enables us to compile the simulator step and vectorize it across large batches of trajectories and randomized domains, which is essential for the throughput required by SMPC. However, it also implies that many model properties must remain static (or be randomized only through precompiled, tensor-valued parameters), shaping the methods and the overall planner design used in the following sections.

5.1.3. Hydrax

We base our implementation on Hydrax [23], which has also been used as the foundation for [28]. Hydrax provides a lightweight yet flexible structure of abstract base classes and basic implementations for three central components:

- Sampling-based MPC algorithms specifying sampling and parameter updates
- Tasks specifying an XML model and a reward function
- Domain randomization aggregation strategies

Additionally, Hydrax follows the *train-state pattern* of the previous Section. The algorithmic state is encapsulated in a `datastruct` object that is iteratively passed through the algorithm, which updates its fields to store quantities such as the current mean and covariance of the sampling distribution or the best-performing control sequence.

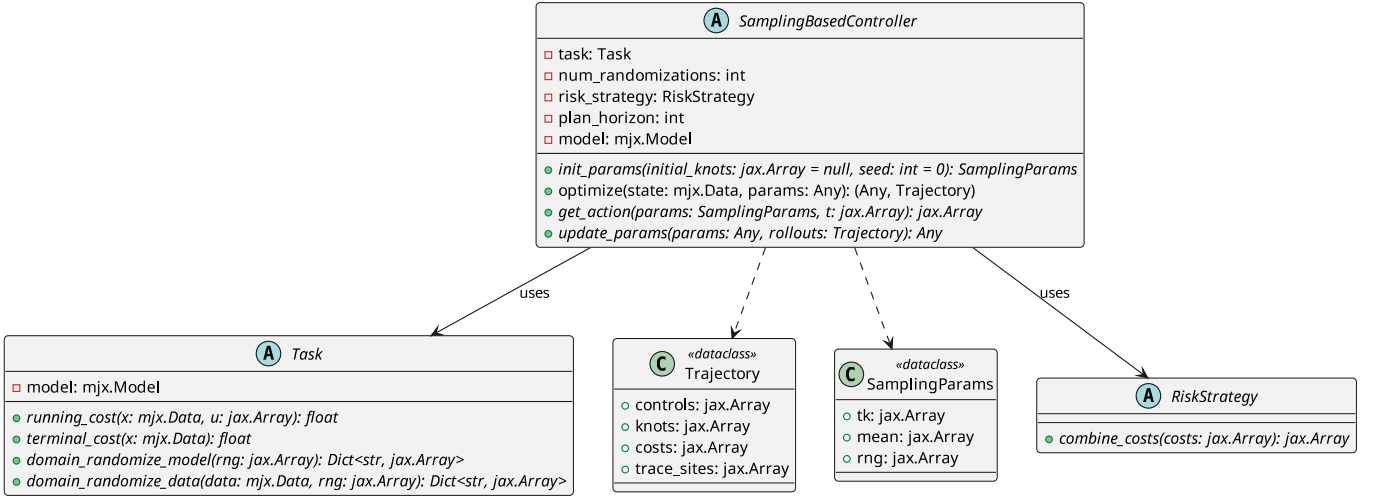


Figure 5.3.: UML of the basic Hydrax codebase. Italic fonts indicate abstract methods.

Figure 5.3 shows a UML of the basic components provided by Hydrax. Internally, the base class for the sampling based MPC algorithms simply uses a batched MJX simulation and evaluates the task on each sampled control input in parallel. This design stands in contrast to earlier approaches for parallelizing stochastic model predictive control (SMPC). Traditionally, parallel rollouts were implemented by explicitly stepping the discrete dynamics Equation (2.1) in hand-written batched form. While efficient for simplified dynamics, this approach becomes brittle when incorporating contact-rich physics, actuator models, and constraints. By contrast, MJX and a JAX-based controller permit end-to-end parallelization of the entire planning pipeline using a full physics engine. More concretely, the idea is to be able to JIT-compile the entire procedure outlined in Algorithm 5.

Algorithm 5 JIT-compiled planning step: OPTIMIZE

Input: Current simulator state \mathbf{x}_k (mjx.Data);

Controller parameters θ_k (e.g., mean U_k , covariance Σ_k , temperature λ , PRNG key)

Output: Updated parameters θ_{k+1} ; rollout data \mathcal{D}_k

```

@jax.jit
1: OPTIMIZE( $\mathbf{x}_k, \theta_k$ ):
2: ( $V_{1:N}, \theta_k$ )  $\leftarrow$  SAMPLETRAJECTORIES( $\theta_k$ ) //  $V_{1:N} \in \mathbb{R}^{N \times T \times m}$ ; updates PRNG state in  $\theta_k$ 
3:  $V_{1:N} \leftarrow$  clip( $V_{1:N}, u_{\min}, u_{\max}$ ) // elementwise
4:  $\mathcal{D}_k \leftarrow$  ROLLOUTWITHRANDOMIZATIONS( $\mathbf{x}_k, V_{1:N}, \text{dr\_rng}$ ) // batched over samples  $\times$  domains
5:  $\theta_{k+1} \leftarrow$  UPDATEPARAMS( $\theta_k, \mathcal{D}_k$ )
6: return ( $\theta_{k+1}, \mathcal{D}_k$ )
  
```

In each planning step, we expand the current policy distribution into a batched control tensor on the accelerator. Concretely, conditioned on the previous iterate’s parameters, we sample N candidate control sequences of horizon length T and evaluate each sequence under D domain randomizations, yielding a 3D array (up to a control-dimension factor). The entire batch is then rolled out in parallel using MJX, producing a cost tensor of the same layout. Finally, the planner aggregates information along the sample, horizon, and domain axes to select the next action and update the distribution parameters; see Figure 5.4 for an illustration of this

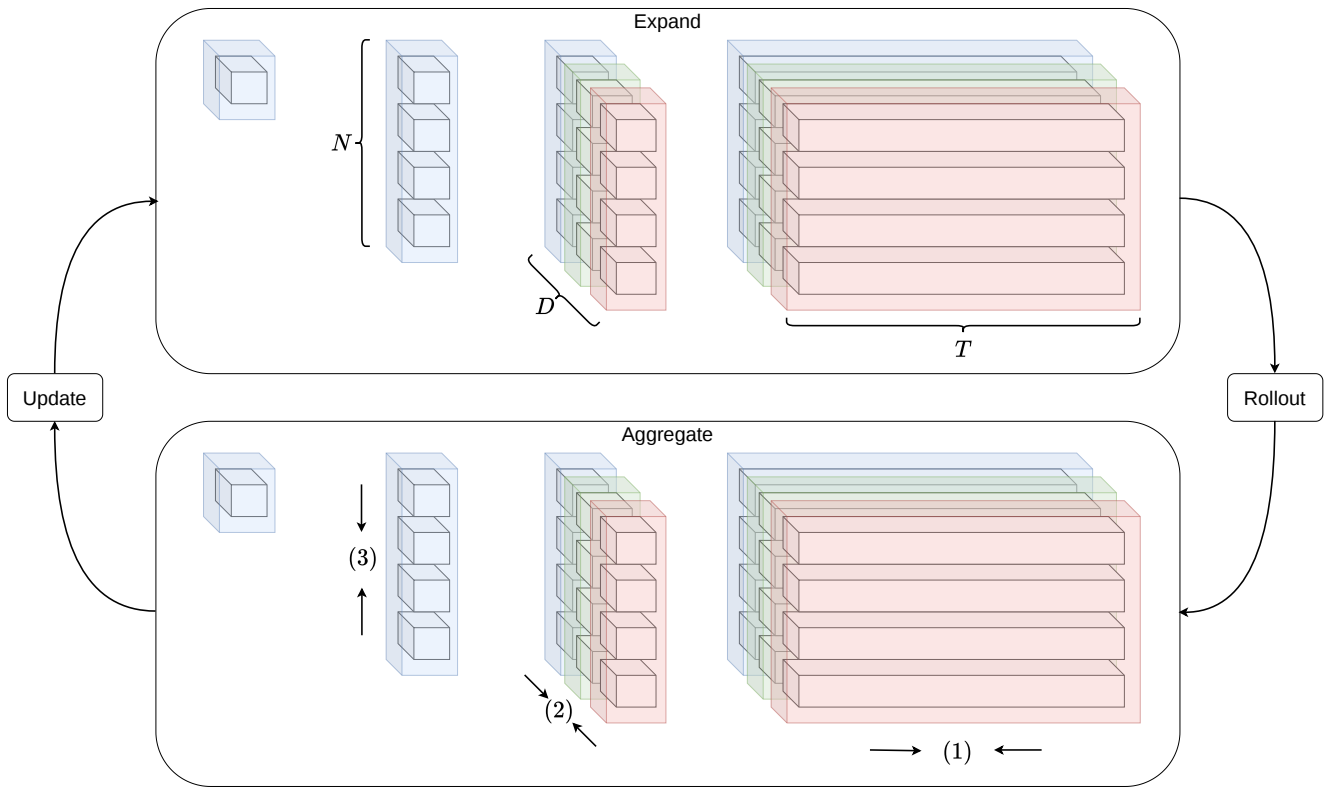


Figure 5.4.: Data evolution over a single planning step. During expansion each data point consist of an element of the control input space U and we sample N trajectories of length T , copied over D domains. After parallel evaluation, we get the same structure for the experienced costs. To make a decision on the next action and distribution update, we accumulate information over trajectories (1), domains (2) and samples (3).

dataflow.

5.2. Extending Hydrax

Hydrax provides a convenient foundation for this work, but requires extensions for sim-to-real transfer, structured domain randomization, and efficient large-batch rollouts. We introduce new simulation environments, performance optimizations, and algorithmic enhancements while preserving end-to-end accelerator execution. This section details our modifications to the framework.

5.2.1. Simulation Environments

We extend the codebase by two additional simulation environments and tasks used throughout our experiments: *Bugtrap Escape* as a controlled local-minimum benchmark, and *Push T* as a contact-rich, multimodal manipulation task.

Bugtrap

As a first example, the Bugtrap environment is implemented. As shown in Figure 5.5, it represents an extreme local minimum in which the agent can easily become trapped. The implementation is based on the particle navigation task in Hydrax, but both the physical setup and the task specification are substantially redesigned.

The original Hydrax task uses a signed distance field with kinematic shadow walls, which provides strong global information on how to escape the trap. Moreover, the reward function is almost entirely defined in terms of this distance field, with a goal-attractor term added only at the final timestep. This makes the task comparatively simple: with sufficiently long horizons, trajectories can either leave the trap or avoid it altogether when initialized outside by navigating around the obstacle. In the experiments of [28], the initial state is placed to the left of the trap, so the main challenge is to avoid entering the local minimum instead of escaping it. While the additional exploration is helpful in this simple case, we argue that additional evaluation is needed. To clearly evaluate whether MTP can escape local minima through its mixing strategy and its ability to rapidly switch between modes, a more challenging variant, *Bugtrap Escape*, is introduced. The environment uses high-mass box primitives for the walls to obtain more realistic contact interactions. The per-timestep cost over the planning horizon is defined as

$$\alpha_1 \cdot \mathbf{1}(\mathbf{f}_{\text{ext}} > 0) + \alpha_2 \cdot \|\mathbf{x}_{\text{particle}} - \mathbf{x}_{\text{goal}}\|_2^2, \quad (5.3)$$

where \mathbf{f}_{ext} is the external (x, y) -force acting on the particle, e.g. caused by collisions. The same structure is used for the terminal cost, scaled by an additional factor.

This formulation results in a considerably more difficult task. The initial state of the particle is now placed inside the bugtrap, so that the system always starts in the local minimum rather than having to avoid it. In contrast to the original setting, where the dominant difficulty is to prevent falling into the trap, the controller must now reliably discover escape strategies from within it. In addition, the force penalty introduces discontinuities due to the indicator function, so the environment only satisfies the minimal regularity assumptions required by the sampling-based MPC formulation discussed in Section 2.3.

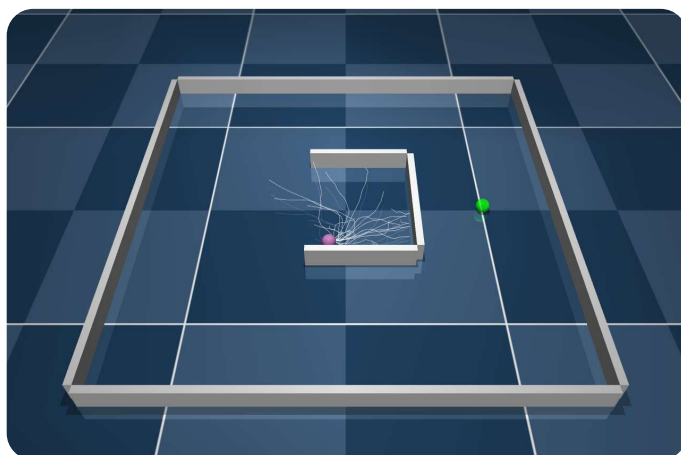


Figure 5.5.: Bugtrap environment with physical walls. The green sphere marks the goal, light grey traces indicate samples rolled out via MJX.

Push T

Beyond toy problems, our primary evaluation environment is the widely used Push T task: a highly multimodal benchmark that requires contact-rich interactions and remains safe to deploy on a real robot. We integrate an FR3 model adapted from the `mujoco_menagerie` repository [61] with velocity control. In practice, this proves to result in smoother samples than position control and reduces computational overhead by avoiding full inverse kinematics inside the rollout loop. We further add a custom end-effector and implement gravity compensation for the robot geometries. We evaluate two distinct T modelling choices:

1. **3 DoF:** In this model, the T is constrained to planar motion (x, y translation and yaw rotation) through sliding and hinge joints. This highly artificial setting serves as a baseline but differs substantially from the real world. The original Hydrax implementation places the planar joint at the geometric center of the T’s bottom face; we instead align it with the center of mass to improve contact behavior under sim-to-real transfer.
2. **7 DoF:** The T operates as a fully free-floating rigid body with complete table interaction. This formulation captures realistic friction, mass distribution, and contact dynamics.

Details about both settings can be found in Appendix A.2. For reward design, our implementation balances task completion, contact initiation, and safety:

$$\alpha_1 d_{SO(3) \times \mathbb{R}^3}(\mathbf{T}_{\text{goal}}, \mathbf{T}^\top) + \alpha_2 \|\mathbf{x}_{\text{ee}} - \mathbf{x}^\top\|_2 + \alpha_3 \mathbf{1}(\|\mathbf{x}_{\text{ee}} - \mathbf{x}_{\text{goal}}\| > \delta), \quad (5.4)$$

where the primary term aligns the T pose with the goal, the secondary encourages end-effector proximity to the object when distant, and the optional third term retracts the end-effector if it extends too far, preventing singularities. Extensive experimentation across all algorithms and numerous cost variants yield several general insights into SMPC design, motivating our simple approach in Equation (5.4):

Additional cost components, such as velocity penalties, hard constraints, or end-effector orientation terms, consistently degrade sampling efficiency. The Monte Carlo nature of SMPC (Section 2.3) discards trajectories violating any single term, even if they excel elsewhere, creating a multiplicative failure mode. We expect this to become particularly acute in high-dimensional action spaces. In particular, action regularization terms

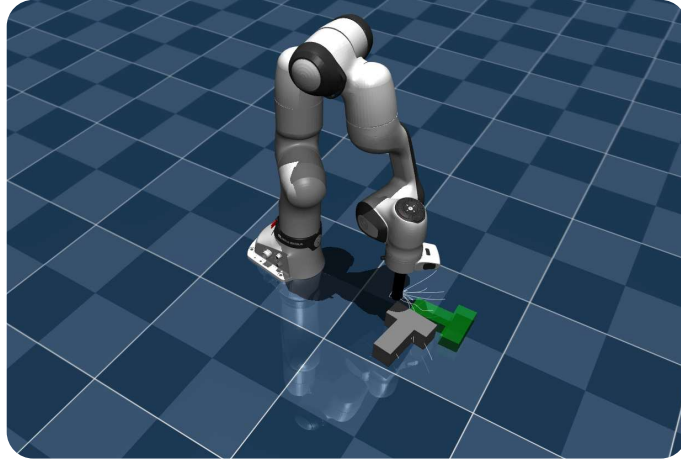


Figure 5.6.: Push-T environment. The green mocap object marks the goal and does not take part in physics computations.

lead to unwanted effects when other cost signals vanish. For example, near goal completion, control costs dominate T-pose error and trap the end-effector in suboptimal configurations, preventing re-approaches from novel angles.

The planning horizon H engages in a delicate interaction with the cost structure. Short horizons promote myopic greediness by limiting foresight, e.g. causing acceleration toward immediately cheaper T configurations and overshooting. For insufficiently long horizons and/or sparse reward structures, stationary policies emerge as local optima, preventing task initiation entirely. Post-collision free sliding further confuses short-horizon planners, as T motion becomes decoupled from robot action within the lookahead. Conversely, longer horizons (≥ 20 steps) amplify Monte Carlo variance: a single poor timestep, whether from collision or poor initialization, renders entire trajectories uncompetitive, starving elite selection.

5.2.2. Framework Extension and Optimization

We extend Hydrax while preserving full GPU residency throughout execution and eliminating recompilation overhead. Building on Hydrax’s general function structure, we introduce optimizations and additional functionality, most notably through a control mapper that disentangles the MuJoCo control array from the sampling space. This enables task-space sampling while retaining joint-space control.

For the Push T task (Section 5.2.1), the mapper implements differential inverse kinematics. We first compute the task-space Jacobian via `jax.grad` on an MJX forward kinematics pass, then calculate the damped Moore–Penrose pseudoinverse

$$\mathbf{J}^\dagger := \mathbf{J}^\top (\mathbf{J}\mathbf{J}^\top + \lambda \mathbf{I})^{-1}, \quad (5.5)$$

to map end-effector twists to joint velocities. To ensure JAX compatibility, we implement the mapper at the task level as a returnable function invoked optionally during rollouts. The function executes exactly once during initialization with constant usage patterns, enabling the MuJoCo XML optimizer to eliminate unused branches.

Hydrax provides a solid JIT-compiled foundation but suffers from excessive memory copying and lacked some of our required functionality. We modify the abstract base implementation in `alg_base.py` to minimize

rollout evaluation overhead. The original stored complete state trajectories during `vmap` scanning; our optimized version eliminates states from scan outputs, retaining only costs and a subset of trace sites. The `Trajectory` dataclass no longer stores controls during intermediate computations, avoiding duplication across domain randomizations since control sequences remain identical. We replace multiple sequential `jnp.append` and `jnp.concatenate` into a single `.replace()` call with pre-allocated shapes during initialization, achieving approximately 10% speedup in planning throughput.

For the `bugtrap` environment, we implement a custom rollout script that aggregates system states into discretized spatial bins to collect information about the system state distribution.

5.2.3. MJX Performance Optimizations

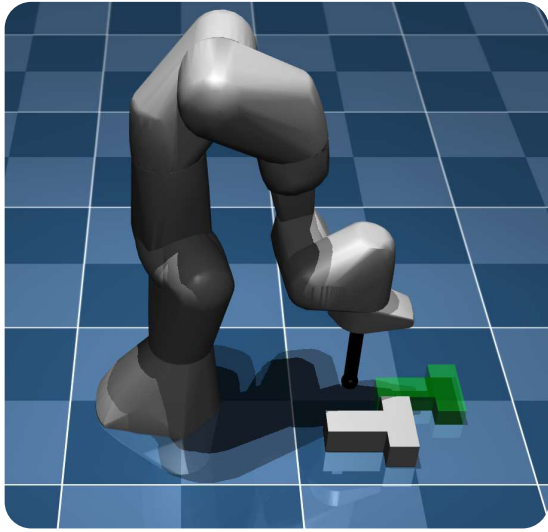
We identify the primary bottleneck to be the computational cost of batched physics solver iterations. Consequently, optimizing both the simulation parameters and the underlying model configuration are central to this work. The fundamental challenge lies in balancing three competing objectives: the simulation timestep, solver computation time, and the planning horizon of the SMPC algorithm. Achieving high planning frequencies requires short horizons and coarse timesteps to minimize the number of solver iterations. However, this optimization is constrained by performance considerations on the horizon length and numerical stability on the timestep size. Following best practices from [11], we configured the physics solver as follows.

The timestep is set as coarse as possible while maintaining simulation stability, and the solver parameters `iterations` and `ls_iterations` are kept at the minimum threshold that prevents simulation divergence. We employ the Newton solver with dense Jacobian computation across all experiments, as this configuration provides superior stability for our control tasks. Based on recommendations from [9], we further customize the numerical integrator by disabling damping and enabling `implicitfast`, which avoids full resolution of gyroscopic and centripetal forces, which unnecessary for our setting where very fast rotations are not expected. The implicit ratio parameter `impratio` is additionally tuned as described in Section 5.1.2.

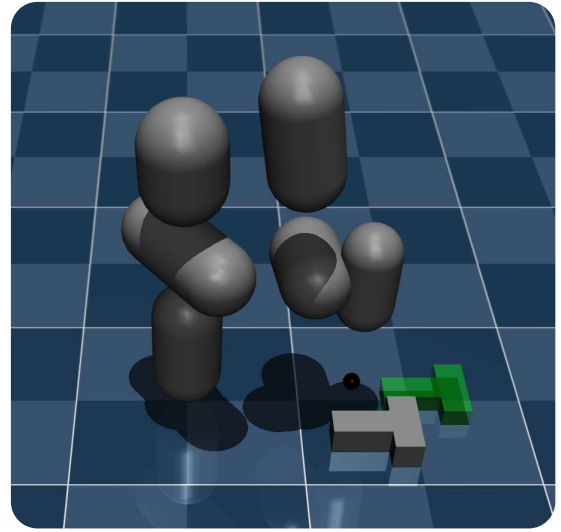
MJX’s contact detection represents a significant computational bottleneck. We address this through two complementary strategies. First, we limit the maximum number of vertices in convex hull decompositions for contact geometry and cap the total number of contact points via the `max_contact_points` numeric parameter. This reduces solver overhead and enables static contact arrays suitable for XLA compilation.

Second, we substantially simplify the robot collision mesh (Figure 5.7), replacing complex meshes with capsule geometries and disabling contacts between kinematically infeasible geometry pairs. This yields large speedups without degrading performance in Push T, which does not require detailed self-collision modeling.

At the runtime level, we additionally apply a set of optimizations to reduce kernel execution overhead and compilation latency. We enable more efficient XLA GPU GEMM kernels and activate JAX’s persistent compilation cache (`jax.config.update`) to avoid repeated recompilation across runs. To ensure that also smaller compiled artifacts are reused, we lower the persistent-cache thresholds (`jax_persistent_cache_min_*`), and we enable an XLA autotune cache so that kernel-tuning decisions can be retained between executions.



(a) Original collision model of the FR3 robotic arm model.



(b) Simplified collision model for MJX performance.

Figure 5.7.: Simplification of the collision meshes for the FR3 XML.

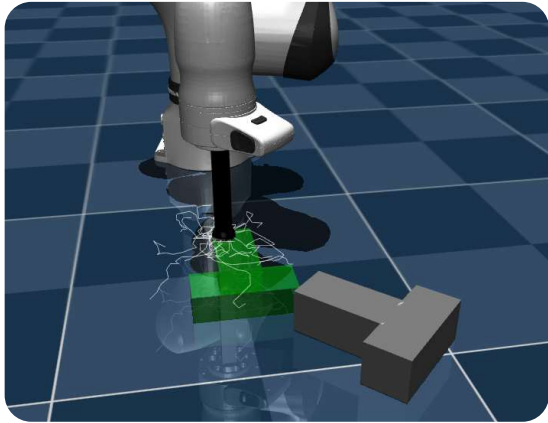
5.2.4. MTP and Extensions

We implement the extensions introduced in Section 4.1.1 with a focus on maintaining JAX/XLA compatibility, i.e., avoiding shape polymorphism and preventing recompilation inside the control loop.

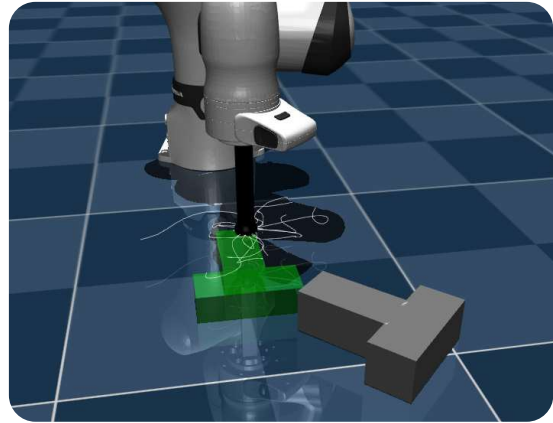
To obtain temporally smooth control sequences while keeping the implementation JAX-friendly, we represent the B-spline interpolation as a fixed matrix multiplication as in Eq. (2.35). In practice, this allows us to map a small set of control points to a dense control trajectory using a single batched linear operation, which is well supported by XLA and benefits from kernel fusion. Compared to the original MTP implementation, we slightly adjust the parametrization by introducing one additional control point in form of the last taken action. This ensures that the resulting spline has the desired boundary behavior over the planning horizon while keeping all tensor shapes fixed. Additionally, we implement a simple Savitzky-Golay (SavGol) filter that is JIT-compatible, which can be used to smoothen trajectories after sampling (see Figure 5.8).

To extend the planner beyond a strict zero-order hold execution model, we shift the control sequence between replanning steps. While this can be implemented efficiently using `jnp.roll`, a key constraint is that the roll offset must be known at compile time to preserve XLA optimization and avoid recompilation (cf. Section 5.1.1). We therefore require an assumed planning frequency as an initialization parameter.

Implementing AnMTP requires varying the global/local mixture coefficient β online, but JAX/XLA places strong constraints on dynamic shapes and data-dependent control flow. To avoid recompilation when β changes, we use a fixed-shape masking strategy. In each planning iteration, we sample a full batch of N local and N global rollouts, independent of the current β . We then apply masks that zero out the unused fraction of samples before combining both sets into the final candidate batch. Since both the sampled arrays and the masks have static shapes, this approach preserves compilation stability while still allowing β to be adapted.



(a) Samples generated without filter



(b) Samples generated by with high smoothing

Figure 5.8.: Effect of applying a SavGol filter to MPPI samples in simulation.

5.2.5. Domain Randomization

Hydrax includes a basic domain randomization utility that applies simple scaling to selected simulation parameters, primarily friction-related coefficients. In our setting, a central limitation arises from MJX’s compilation model: after compiling an XML model and JIT-compiling the simulation step, only parameters represented as JAX arrays can be changed efficiently, while many structural parameters (e.g., geometry dimensions, some solver settings) would require recompiling the model. Moreover, MuJoCo computes and caches derived quantities during model compilation; for mass-related fields (e.g., mass, density, center of mass), changing only a subset of values post hoc can therefore lead to inconsistent physics.

To address this, we implement a structured domain randomization interface that supports per-body, per-geom, and per-joint randomizations with explicit ranges or fixed values. Where it is required for consistent derived quantities, we compile and initialize a separate MuJoCo model for each domain tuple ξ and extract the resulting (derived) parameters into the batched pytree representation used by the planner. Finally, we augment the algorithm state with a domain-weight vector that can be used to reweight the aggregation step during optimization, either on-device inside the compiled planning step or externally.

5.3. Sim-to-Sim Pipeline

For simulation-only studies, we follow Hydrax’s standard execution structure with only minimal extensions for visualization and the domain randomization mechanisms described in Section 5.2.5. At every replanning step, the current MuJoCo state is copied into an `mjx.Data` instance, the planner generates a new control sequence via a JIT-compiled optimization step, and the first part of this sequence is then executed in MuJoCo until the next replanning event, compare Figure 5.9 and Algorithm 6.

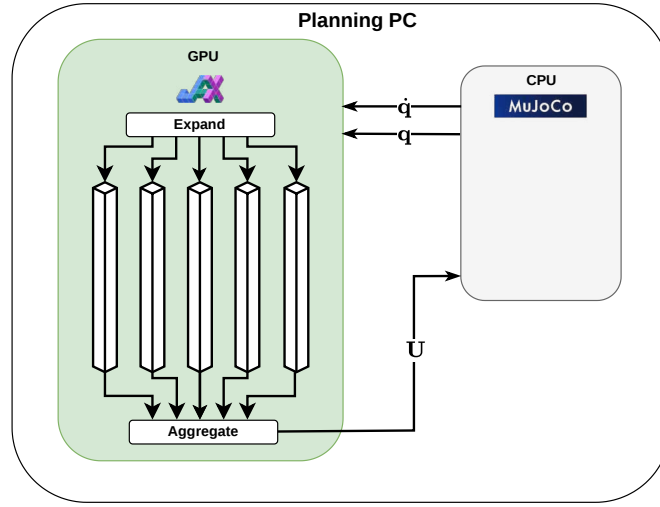


Figure 5.9.: Sim-to-sim system overview. A CPU MuJoCo simulation provides the current state, which is copied to MJX for accelerator-resident planning. The resulting control sequence is then executed back in MuJoCo until the next replanning step.

Algorithm 6 Deterministic sim-to-sim pipeline

Input: MuJoCo model M , MuJoCo data D ;
 Planning frequency f_{ctrl} ;
 S MPC planner exposing OPTIMIZE (Alg. 5)
Output: Logged rollout data (optional)

```

1: Initialize MJX state  $\mathbf{x}_0 \leftarrow \text{ToMJXSTATE}(D)$  //  $\mathbf{x}_k \in \text{mjx.Data}$ 
2: Initialize controller parameters  $\theta_0 \leftarrow \text{INITPARAMS}(\text{seed})$ 
3: JIT-compile OPTIMIZE( $\mathbf{x}_0, \theta_0$ ) and run one warm-up call
4: while  $k < \text{max\_step}$  do
5:   Synchronize state:  $\mathbf{x}_k \leftarrow \text{UPDATEMJXSTATE}(\mathbf{x}_{k-1}, D)$  // copy qpos, qvel, mocap, time, ...
6:    $(\theta_{k+1}, \mathcal{D}_k) \leftarrow \text{OPTIMIZE}(\mathbf{x}_k, \theta_k)$ 
7:   Extract planned control sequence  $U_k \leftarrow \text{GETPLAN}(\theta_{k+1})$  //  $U_k \in \mathbb{R}^{T \times m}$ 
8:   for  $i = 0$  to  $\text{sim\_steps\_per\_replan}-1$  do
9:     Select within-horizon index  $t_i$  // discrete index or time-based mapping
10:     $u_{k,i} \leftarrow \text{GETACTION}(U_k, t_i)$ 
11:     $u_{k,i} \leftarrow \text{CONTROLMAP}(u_{k,i})$ 
12:    Apply and step simulation:  $(M, D) \leftarrow \text{STEPMUJoCo}(M, D, u_{k,i})$ 
13:   end for
14:    $k \leftarrow k + 1$ 
15: end while

```

5.4. Real-to-Sim-to-Real Pipeline

To evaluate sampling-based model predictive control algorithms on real systems with MJX, we implement the Push-T experiment on a physical robotic platform.

5.4.1. Franka Research 3 Robotic Arm

The Franka Research 3 (FR3) is a 7-DoF collaborative manipulator designed for safe human–robot interaction and contact-rich manipulation. Its redundant kinematics provide flexibility for obstacle avoidance and natural motion generation. The arm is torque-controlled and equipped with joint torque sensing, enabling force estimation and compliant interaction. Through the Franka Control Interface (FCI), the robot exposes low-level control and sensor access at 1 kHz. For non-real-time orchestration, the Franka Desk API provides higher-level interfaces for configuring skills and task sequences. The system includes built-in supervision mechanisms such as self-collision monitoring, external collision detection, and singularity checks. The FR3 has a reach of approximately 855 mm and a payload capacity of up to 3 kg. It ships with a two-finger gripper; for pushing, we replace this with a custom 3D-printed end-effector, see Figure 5.10. More detailed information can be found in the official manual [13].

5.4.2. ROS 2 Middleware

ROS 2 (Robot Operating System 2 [32]) serves as the middleware framework for the robotic system presented in this work. It provides a standardized communication infrastructure built on the Data Distribution Service (DDS), which supports real-time, distributed coordination among modular software components. In contrast to its predecessor, ROS 2 introduces improved support for multi-threading, deterministic execution, and integration with real-time operating systems. These features are critical for robotics applications that demand predictable timing and high reliability. The core communication abstractions of ROS 2—nodes, topics, services, and actions—enable a clean separation of concerns across system components while ensuring interoperability and scalability in multi-agent or heterogeneous environments. Within the scope of this thesis, ROS 2 orchestrates communication between perception, control, and simulation modules.

For low-level actuation and high-level motion planning, this work leverages the `ros2_control` framework in combination with MoveIt2. The `ros2_control` framework standardizes the interface between robot hardware (or its simulated equivalent) and software controllers. It provides a controller manager responsible for loading, configuring, and executing real-time controllers, including position-, velocity-, and effort-based control loops. Hardware interfaces define how joint states are read and commands are issued, which allows the same control logic to be reused across different physical robots or simulation backends.

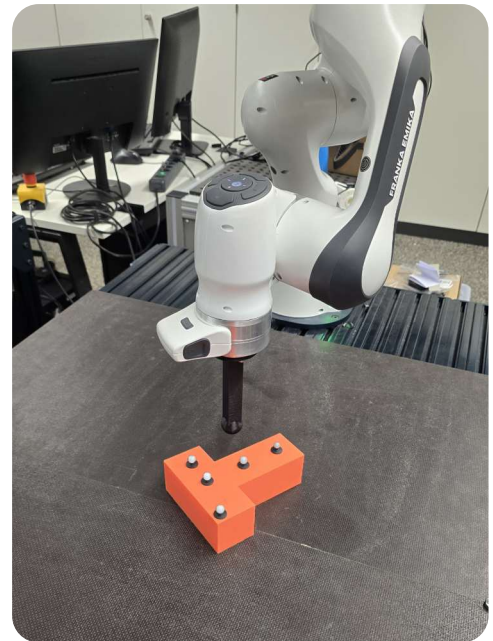


Figure 5.10.: Franka Research 3 robotic arm in the IAS lab with a custom end-effector and OptiTrack markers on the T-shaped object.

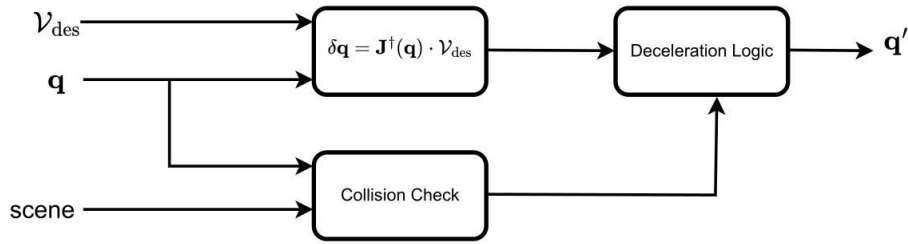


Figure 5.12.: Basic functionality of MoveIt 2 Servo. Given a desired end-effector twist \mathcal{V}_{des} and the current joint positions \mathbf{q} , MoveIt 2 Servo computes joint velocity commands using the Jacobian. These commands are sent to the joint trajectory controller. In a separate thread, typically running at a lower frequency, collision checking and singularity handling are performed. If a potential collision or singularity is detected, the commands are scaled down to safely decelerate or stop the robot.

MoveIt2 and Servo

MoveIt [17] is a widely used open-source framework for robot motion planning, manipulation, and control in the ROS / ROS2 ecosystem. It provides a high-level interface for tasks such as inverse kinematics, motion planning, trajectory execution, and collision avoidance, building on libraries like the Open Motion Planning Library (OMPL) for sampling-based planners. In ROS 2, MoveIt2 has been re-engineered to take advantage of the improved communication middleware (DDS) and real-time capabilities, while maintaining its modular design. MoveIt2 abstracts the underlying robot model through the Unified Robot Description Format (URDF) and Semantic Robot Description Format (SRDF), allowing generic tools to be applied across different manipulators. It integrates planning scene representations that account for the robot’s kinematics, dynamics, and environment geometry for collision-aware motion planning and constraint handling.

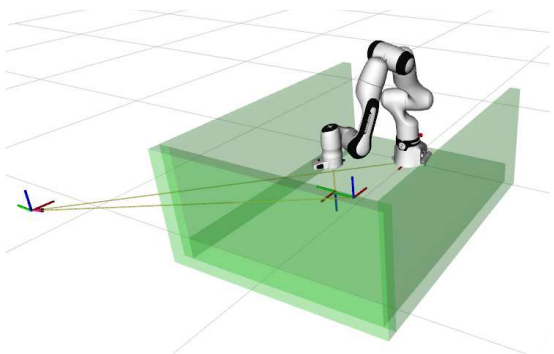


Figure 5.11.: MoveIt2 collision boxes used for safety in the Push-T task.

While traditional motion planning pipelines in MoveIt2 operate on a plan–then–execute principle, many applications require real-time responsiveness to external input. For these scenarios, MoveIt2 provides the *MoveIt2 Servo* package. MoveIt2 Servo enables continuous, low-latency control of robot end-effectors by streaming Cartesian or joint-space velocity commands that are executed in real time. Unlike precomputed trajectories, Servo computes small incremental motions at high frequency, regularly checks for collisions against the planning scene, and sends updated commands to the controller. The functionality is summarized in Figure 5.12. In our setup, MoveIt2 handles initialization motions to predefined configurations and provides collision avoidance through continuously updated

planning scenes. MoveIt2 Servo generates low-level velocity commands from the SMPC planning algorithm, which are then executed by the underlying `JointTrajectoryController` at 1 kHz.

5.4.3. OptiTrack Motion Capture System

OptiTrack is a commercial optical motion capture (mocap) system widely used in robotics, biomechanics, and animation. It provides high-precision 6-DoF pose estimates of tracked objects at low latency. A typical setup consists of three main hardware components: infrared cameras, retroreflective markers, and a central processing workstation running OptiTrack's proprietary *Motive* software.

The capture volume is monitored by multiple infrared (IR) cameras arranged around the workspace. Each camera integrates IR LEDs that actively illuminate the scene, while retroreflective markers mounted on tracked objects return the light back to the camera sensors. We use the Flex 3 system [35], which operates at 100 Hz, with spatial resolutions of around 0.3 MP. The vendor specifies pose estimation accuracy in the order of < 0.5 mm in position and $< 0.5^\circ$ in orientation with end-to-end delays of approximately 10 ms.

The system tracks small, lightweight retroreflective spherical markers. Individual markers are grouped into predefined constellations known as *rigid bodies*, which can be manually adjusted using the Motive software interface. By recognizing and fitting these constellations, the system can reconstruct the full 7-DoF pose (position and orientation) of the rigid body, even when a subset of markers is temporarily occluded. The use of multiple cameras provides redundancy for robust tracking in cluttered environments.

All camera data is streamed to the central workstation, where Motive performs camera calibration, marker detection, rigid body identification, and pose estimation. The resulting rigid body poses are broadcast in real time over a local area network using the NatNet (NaturalPoint Network) protocol.

The capture volume is expressed in a global OptiTrack coordinate frame, established during a calibration procedure using a wand and ground-plane tool. By convention, OptiTrack defines this global frame as Y-up.

OptiTrack-ROS 2 Bridge and Calibration

The integration of a motion capture system with a robotic platform requires bridging differences in coordinate conventions and establishing a common reference frame. In this work, the OptiTrack system is used to provide ground-truth pose information. However, OptiTrack and ROS 2 employ different coordinate conventions and communication protocols.

We implemented a lightweight ROS 2 node using the NatNet C++ SDK that connects to the OptiTrack server at a fixed IPv4 address and tracks rigid bodies by name (since IDs may change) and republishes the information of a predefined set of objects of interest as `geometry_msgs/TransformStamped` messages, using `optitrack` as the parent frame. The robot in ROS 2 maintains its own internal world frame, which must be related to the OptiTrack world frame in order to express both systems in a common coordinate system. To this end, a hand-eye calibration procedure based on Tsai's method [48] is applied. A calibration marker consisting of multiple reflectors is physically attached to the robot and registered within the OptiTrack system as a rigid body called `fr3-calibration-object`. The algorithm then commands the robot to a sequence of pre-defined calibration poses, storing two measurements for each pose i :

$${}_{\text{fr3_link0}}\mathbf{T}_{\text{fr3_link5}}(i), \quad {}_{\text{optitrack}}\mathbf{T}_{\text{calibration}}(i). \quad (5.6)$$

The first transform is obtained from the robot's kinematic model and joint encoders, while the second is provided by OptiTrack. The objective is to determine the fixed extrinsic calibration transform

$$\mathbf{T}^* = {}_{\text{fr3_link0}}\mathbf{T}_{\text{optitrack}}, \quad (5.7)$$

which aligns both frames of reference. Since the calibration object is rigidly attached to the robot link with a fixed offset, relative motions between pairs of poses are considered. For each pair (i, j) , the relative motions are computed as

$$\mathbf{A}_{ij} = \left({}^{\text{fr3_link0}}\mathbf{T}_{\text{fr3_link5}}(j) \right)^{-1} \cdot {}^{\text{fr3_link0}}\mathbf{T}_{\text{fr3_link5}}(i), \quad (5.8)$$

$$\mathbf{B}_{ij} = \left({}^{\text{optitrack}}\mathbf{T}_{\text{calibration}}(j) \right)^{-1} \cdot {}^{\text{optitrack}}\mathbf{T}_{\text{calibration}}(i), \quad (5.9)$$

expressing the same physical motion in their respective frames. Assuming \mathbf{T}^* to be a fixed transformation, the change-of-basis constraint is given by

$$\mathbf{A}_{ij} = \mathbf{T}^* \cdot \mathbf{B}_{ij} \cdot (\mathbf{T}^*)^{-1}. \quad (5.10)$$

In practice, this constraint is solved by dividing the problem into two least-squares subproblems: one for rotation, and another for translation conditioned on the estimated rotation. For a more detailed derivation and comprehensive illustrations of the frame relationships, we refer the reader to e.g. [21]. The resulting transformation \mathbf{T}^* is used to express pose information provided by the tracking in the same reference frame as the robot.

5.4.4. System Overview

For the full experimental setup, we assemble all components above into one experimentation pipeline: two ROS 2 nodes running on a control PC with a real-time kernel for the ROS 2 MoveIt 2 robot interface detailed in Section 5.4.2 and the bridge from Section 5.4.3, and two more nodes on a planning PC connecting all the components and running the JIT-compiled planning loop on an Nvidia 5090 GPU [36].

For the robot interface, we adapt the MoveIt 2 example of the official `franka_ros2` repository [59]. We customize the robot description to include our custom end effector and add the Servo interface with its required configurations to the launch file. Additionally, we launch the `JointTrajectoryController` and link it to MoveIt 2.

To interface with the exposed ROS 2 topics, we implement an abstract controller base class derived from `relpy.node.Node`. Internally, we adapt the `MoveIt2` and `MoveIt2Servo` interfaces provided by the `pymoveit2` repository [37] to ROS 2 Jazzy, applying minor API and configuration changes. This base class wraps the motion-planning and servoing services to provide methods for querying the current joint positions, velocities, and efforts, as well as for sending joint-space targets to the planner and Cartesian twist commands to the servo controller. In addition, the class subscribes to end-effector pose, twist, joint states, and TF transforms, and stores these in local buffers for thread-safe access by higher-level controllers.

After running the hand–eye calibration procedure detailed in Section 5.4.3, we inject the resulting extrinsic transformation between the robot base frame and the OptiTrack world frame by manually publishing it in the planning node’s initialization. Additionally, we account for the discrepancy between the T-object frame in MuJoCo and the rigid body frame in OptiTrack, which we determine experimentally. To adjust for systematic errors, we additionally implement a visualization node that allows calibration of a manual offset by testing contact initiation via keyboard control.

For planning, we encapsulate the planning step from Algorithm 5 along with state synchronization and action extraction into a single function, which we JIT-compile during initialization. To circumvent issues arising from the Python GIL not allowing true multithreading, we publish the full optimal action trajectory \mathbf{U}_k after each planning step and implement a separate servoing node. Based on the time passed since the last planning step,

this node computes the respective index to select an action from \mathbf{U}_k . This allows control at arbitrary planning frequencies while keeping servo frequency constant. For safety, the servo node automatically publishes a stop command when the current index exceeds the horizon length, effectively constraining the planning frequency to be high enough to cover the planning horizon.

Algorithm 7 FR3 PushT rollout initialization

Input: SMPC controller; control frequency f_{ctrl} ; robot IP

- 1: Add collision primitives // MoveIt2 via pymoveit2
 - 2: Move to initial pose
 - 3: Bind TF buffer/listener; publish static TFs // OptiTrack → Robot, ROS2 → MuJoCo
 - 4: **while** TF unavailable **do**
 - 5: Spin ROS once
 - 6: **end while**
 - 7: **while** robot/object states missing **do**
 - 8: Spin ROS; update object pose \mathbf{T}_{obj} , robot state (q, \dot{q})
 - 9: **end while**
 - 10: Initialize MJX state $\mathbf{x}_0 \leftarrow \text{ToMJXSTATE}(D)$
 - 11: Warm-start simulation to resolve initial contacts/state
 - 12: Initialize controller parameters $\theta_0 \leftarrow \text{INITPARAMS}(\text{seed})$
 - 13: JIT-compile controller step $\text{STEP}(\mathbf{x}_0, \theta_0, q, \dot{q}, \mathbf{T}_{obj})$
 - 14: Start timers: planning $(1/f_{ctrl})$, timeout
-

Algorithm 7 details the initialization procedure for the FR3 Push-T system. After creating the ROS 2 node, we add collision primitives (table, workspace boundaries) to MoveIt’s planning scene for safety. We synchronize the three coordinate frames (robot base, OptiTrack world, MJX simulation) by publishing static transforms and waiting for the TF buffer to resolve them. Once the TF tree is complete and robot/object states are observed, we initialize the MJX simulator to match the physical state, perform a forward step to resolve contact constraints, initialize policy parameters from the control distribution, and finally JIT-compile the planning step and warm-start.

Additionally, we add simple data logging inside the planning loop to record robot states, object poses, actions, and costs for offline analysis. A counter automatically stops the rollout after a set time. An overview of the full system integration is provided in Figure 5.13.

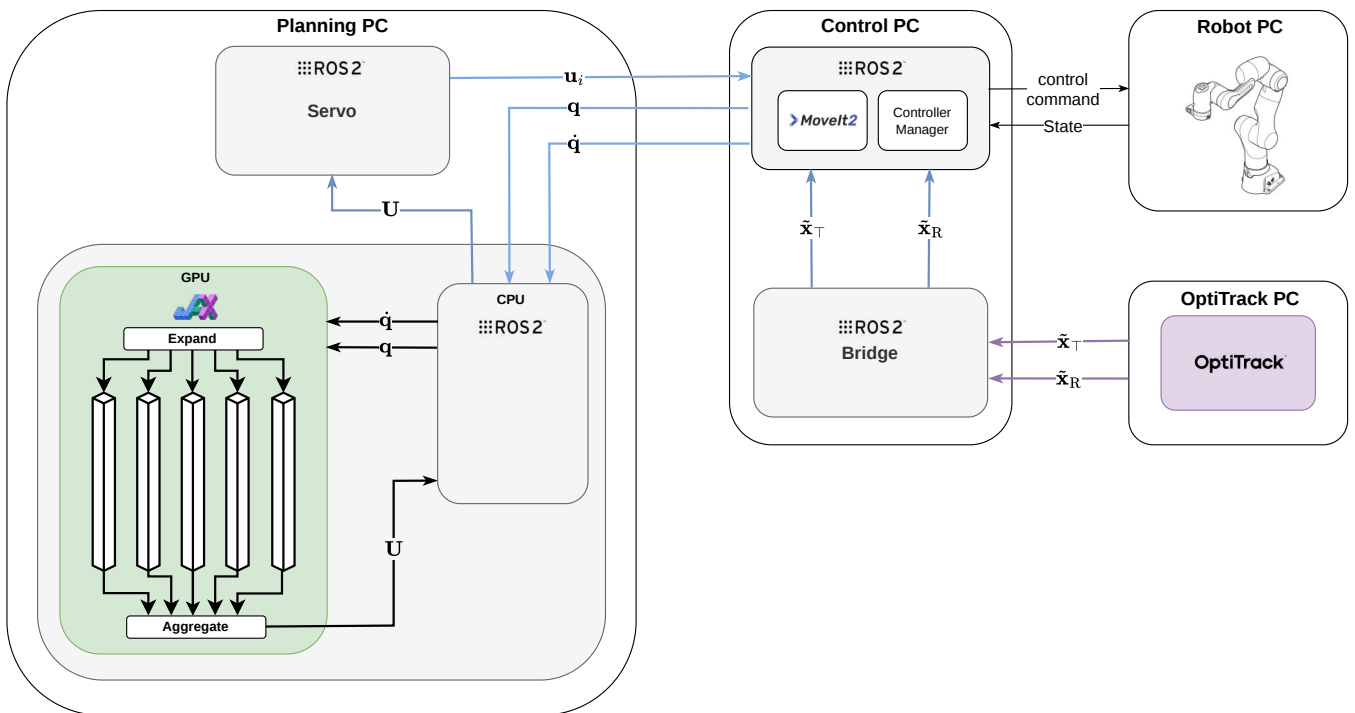


Figure 5.13.: Push-T experimental system overview. Blue arrows indicate ROS 2 topic-based communication between processes. Purple arrows indicate OptiTrack NatNet protocol streaming. The three processes (control, bridge, planning) are loosely coupled via asynchronous messaging, allowing the GPU-accelerated planning node to operate independently.

6. Experimental Results

This chapter evaluates the proposed sampling-based MPC methods across a set of benchmark tasks in simulation and on real hardware. We first validate implementations and isolate algorithmic effects in controlled MuJoCo/MJX settings with the system outlined in Section 5.3, and then assess transfer to the physical system in the real-to-sim-to-real pipeline of Section 5.4. To extend the results of [28], we evaluate MTP against baseline implementations of MPPI, CEM, and PS as described in Section 2.3. Hyperparameters and implementation details are provided in the appendix.

6.1. Bugtrap Escape

We begin with the bugtrap escape task introduced in Section 5.2.1. Figure 6.1 visualizes the empirical state visitation distribution induced by each algorithm over 10000 planning steps, aggregated over 6 seeds. To enable fair comparison, we selected hyperparameters for each baseline to be as similar as possible; detailed values are provided in Appendix A.1.

Both CEM and MPPI fail to escape the trap across all seeds, but their failure modes differ. CEM greedily pursues low-cost states near the inner wall based on elite samples that initially avoid contact penalties. Once contact becomes sufficiently likely, the associated costs exceed the state penalty, forcing the algorithm to steer away from the wall. This can induce oscillatory behavior: elite samples alternate between trajectories with significant contact and those with no contact, causing the particle to become trapped in a limit cycle near the wall. In contrast, MPPI incorporates all samples when updating its parameters. Collision costs from wall contact therefore influence the update more uniformly, driving the particle away from the walls. Simultaneously, the attractor term in Equation (5.3) prevents leftward motion, and the particle equilibrates near the center of the trap.

Predictive Sampling exhibits behavior similar to CEM, but with higher variance that can occasionally enable escape. The state distribution indicates, however, that escape occurs rarely; the particle spends the majority of rollouts near the wall, mirroring CEM’s dominant mode.

MTP demonstrates fundamentally different exploration characteristics. Rather than remaining confined to either the walls or the center, MTP explores the interior more systematically and can identify escape routes in which the combined costs of wall collisions and goal distance are overcome by discovering the necessary detour.

This example validates the findings of [28] in a setting that explicitly demands the SMPC formulation for its non-smooth cost-landscape. MTP enforces systematic exploration and therefore reliably solves a challenging problem characterized by local minima and narrow escape corridors.

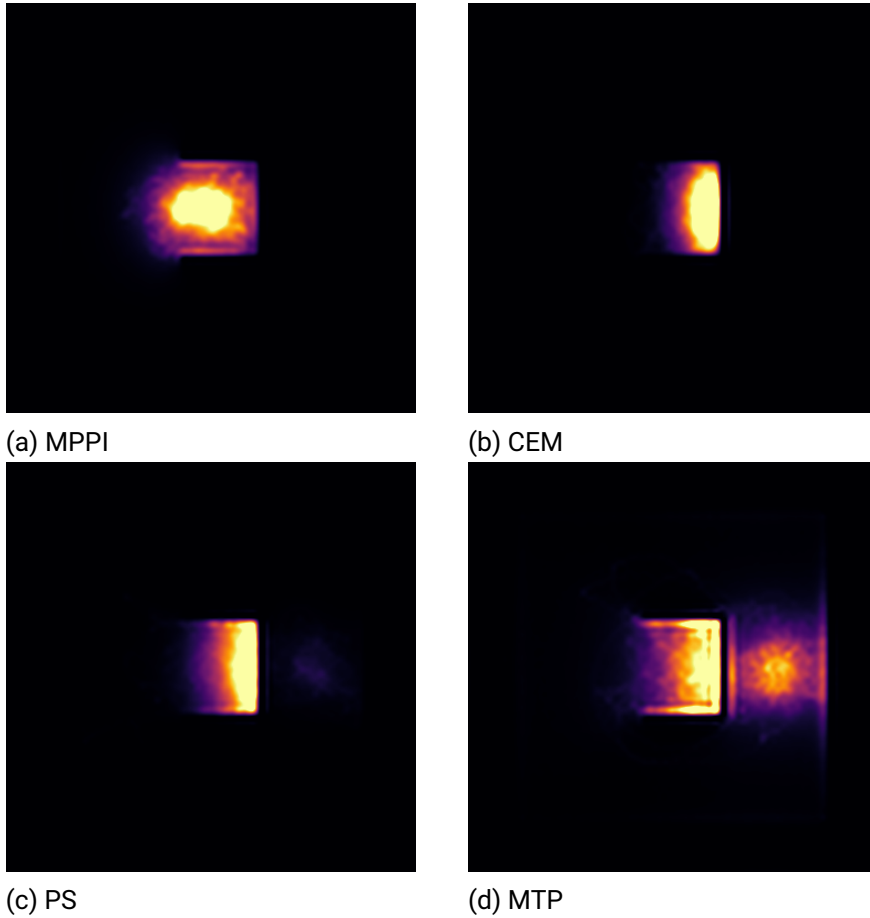


Figure 6.1.: Empirical state visitation distributions for the bugtrap escape task over 10000 planning steps, aggregated over 6 seeds with noise on the initial particle position. Base MPPI and CEM remain stuck in the local minimum inside the trap in all seeds, while Predictive Sampling escapes only once and otherwise behaves similarly to CEM. The global samples together with elite perturbations enable MTP to escape and reach the goal in every run. All algorithms plan at 25 Hz.

6.2. Push T Sim-2-Sim

Next, we investigate the Push-T task in simulation. To keep results representative of real-world deployment, we use the same settings that are intended to enable real-time control, but benchmark at higher planning frequencies to gain an understanding of the effect of these coarse timesteps. Since the difficulty and approach to this task varies dramatically depending on the initial position of the end-effector and the orientation of both goal and T object, ranging from a simple straight push to complicated manipulation trajectories, we rely on deterministic seeds for comparison rather than averaging across random initializations. Simple straight-line pushes are not particularly informative for distinguishing algorithm capabilities, so we focus on challenging scenarios that require mode switching, re-establishment of contact, or navigation around local minima. Figure 6.2 shows the scenarios under test, all of which present nontrivial challenges. The scenarios are color-coded by difficulty: yellow provides strong local reward signal but requires careful approach angles, red scenarios exhibit local minima or necessitate pose correction maneuvers, and dark red scenarios either place the system in a pronounced local minimum (seed 5) or are highly likely to enter one after initial pose

correction (seed 3).

Initial tests across both the 3 DoF (jointed) and 7 DoF (free-floating) models, various hyperparameter combinations, and different initial conditions reveal several consistent patterns:

Elite selection robustness: MTP can handle a wide range of elite fractions, remaining stable and effective across settings. In contrast, CEM performance degrades substantially when the elite set is chosen too large or too small.

Covariance update: Disabling covariance updates slightly benefits CEM by preventing variance collapse, while having minimal effect on other algorithms. For consistency and fair comparison, we fix the covariance across all methods in subsequent experiments.

Sample smoothing: After reimplementing B-spline-based trajectory smoothing for both global and local samples, we cannot observe significant performance differences between Akima and B-spline smoothing in simulation. Moreover, applying smoothing to MPPI samples does not yield measurable benefits in the sim-to-sim setting.

Since practical control frequencies may necessitate shorter horizons on hardware, we run all methods for two horizon settings across all seeds to quantify the resulting performance trade-offs.

Figure 6.3 presents results in form of the final pose error after 100 planning iterations. Several trends emerge:

As expected, the 3 DoF jointed setting is substantially easier than the free-floating 7 DoF case. The constrained kinematics eliminate table-object tipping interactions and idealize contact dynamics. All algorithms fail on the 7 DoF setting for seed 5, where there is no local reward signal and the T object is positioned such that the end-effector must execute a complex repositioning maneuver before establishing productive contact.

Short horizons have negative consequences on performance across all algorithms. As shown in Figure 6.3b, reducing the planning horizon degrades average final state error for all methods. Interestingly, in some individual seeds, a shorter horizon slightly improves MTP and PS performance. We explain this via the Monte Carlo sampling efficiency problem we discussed in Section 2.3: with long-horizon rollouts, each trajectory has a higher probability of encountering at least one bad timestep (e.g., collision, poor initialization, or singularity approach), which causes the entire trajectory to be discarded or heavily downweighted. Shorter horizons reduce this multiplicative failure mode, but at the cost of reduced foresight and increased myopic behavior.

MPPI is by far the worst-performing algorithm. It frequently fails to initiate contact, particularly in scenarios requiring approach from non-obvious angles. This behavior is consistent with MPPI's update rule, which incorporates all samples into the mean via softmax weighting. When the majority of samples produce high costs due to poor contact or lack of task progress, the algorithm struggles to commit to the few promising trajectories, resulting in conservative, oscillatory, or entirely passive behavior.

CEM performs better than MPPI, especially in the 3 DoF jointed setting where contact dynamics are simplified. However, CEM remains prone to local minima. Once elite samples converge to a suboptimal mode—such as hovering near the T without making contact—the algorithm can become trapped, as the elite set reinforces the current strategy without sufficient exploration to discover alternatives.

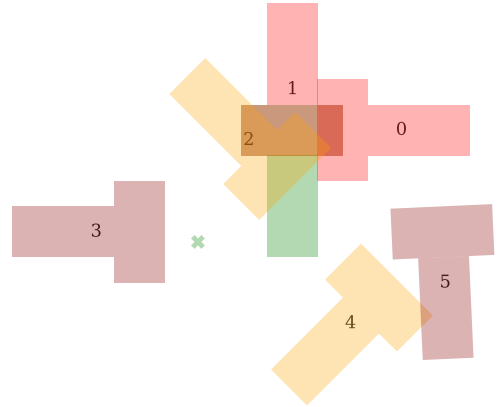
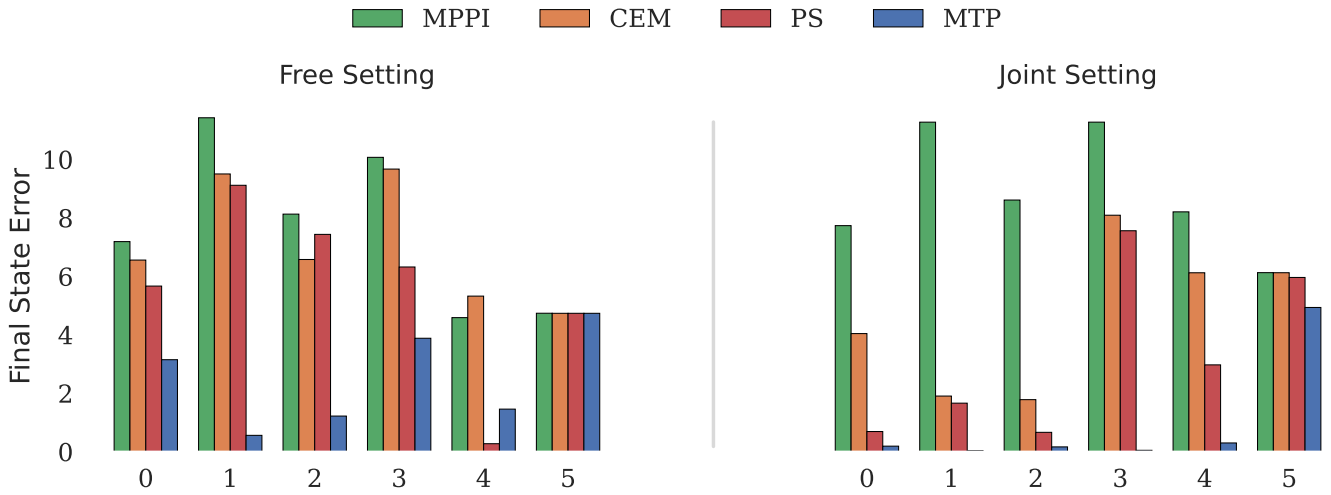
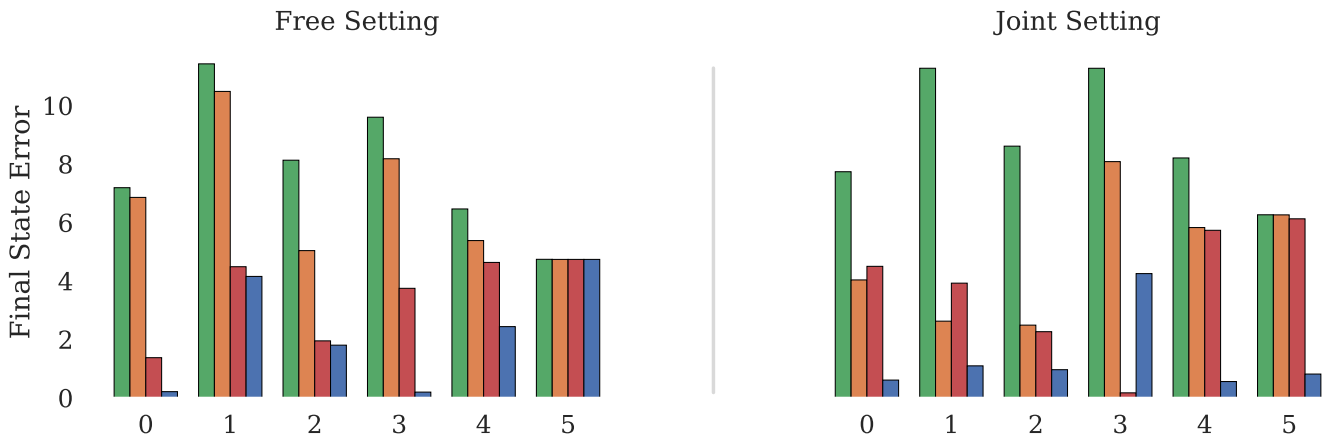


Figure 6.2.: Scenarios under test for our sim-to-sim evaluation of the Push-T task. Colors indicate difficulty: yellow is easiest, dark red hardest. The green cross marks the initial end-effector position.



(a) Long horizon results (0.5 sec).



(b) Short horizon results (0.375 sec).

Figure 6.3.: Sim-to-sim results for Push T over long and short horizons. All methods use 256 samples at 20 Hz.

PS is surprisingly effective in several scenarios, leveraging its greedy execution of the single best sample to achieve rapid task progress when a good trajectory is found. However, inspection of the rollout visualizations reveals a critical weakness: PS exhibits overly aggressive behavior that can lead to unrealistic contacts, effectively sticking the end effector to the T. This is particularly evident in the free-floating 7 DoF setting, where aggressive end-effector motion can induce T object tilting or edge contacts that lead to unrealistic behavior. Therefore, we do not expect this performance to translate to real systems, where the sim-to-real gap makes such aggressive, contact-exploiting behavior brittle; small modeling and sensing errors can turn it into contact loss, slip, or overshooting.

MTP is the most robust algorithm across varying initial states for both horizons, and it is the only method that succeeds on the hardest seed in any setting. While PS outperforms MTP in two scenarios, we do not expect this advantage to carry over to the real system for the reasons stated above.

6.2.1. Annealed MTP

We evaluated AnMTP to determine if a dynamic mixing coefficient β could better balance global exploration and local precision. By scaling β based on the current task error, we aimed to encourage mode-discovery when far from the goal and exploitative smoothness during final alignment. Two scheduling strategies were tested:

- **Momentum-based adaptation:** Increasing β when the cost remains stagnant (indicating a local minimum) and decreasing it when cost is successfully reduced.
- **Error-based decay:** Linear or exponential scaling from a maximal β_{max} to a minimal β_{min} based on the distance to the target pose.

While these schedules allowed adapted behavior, they did not provide a statistically significant performance boost over a well-tuned fixed mixing coefficient. For the Push-T task, we find that a **constant modest** $\beta \in [0.25, 0.4]$ provides the best trade-off.

Results: Simulation Studies

MTP successfully identifies escape routes in the Bugtrap environment where local-sampling baselines like CEM, MPPI and PS oscillate or get trapped, validating the necessity of the global samples for exploration. In the Push-T task, MTP is the most robust algorithm and demonstrates unique resilience to shortened planning horizons.

6.3. Planning Pipeline Scaling

Before deploying to the real robotic system, we investigate the computational cost of the planning step, as execution time directly constrains the planning frequency achievable by the real-to-sim-to-real pipeline introduced in Section 5.4. We benchmark CEM, MPPI, and MTP across varying problem scales. We exclude Predictive Sampling from this comparison because it is essentially CEM with a single elite sample. Additionally, we evaluate AnMTP to validate the overhead introduced by our masking implementation. To ensure a fair comparison, we fix the initial system state and employ the same similar parameters as in the sim-to-sim experiments. In all scaling experiments, we use the 3 DoF model version.

Figure 6.4 shows favorable scaling behavior over the tested range: median runtimes increase moderately with problem size. When doubling the number of samples, the median planning frequency decreases by less than 20% cumulatively across the two first scaling steps. All algorithms exhibit similar scaling trends, indicating that our annealed masking implementation (Section 5.2.4) introduces negligible overhead. Moreover, increasing the number of samples within a single domain and increasing the number of domains at fixed samples lead to comparable runtime scaling, suggesting that the implementation generalizes across both parameterizations.

However, several concerns remain. First, while median scaling is favorable, the interquartile range widens substantially with increasing problem size. This variability can become problematic when stable planning rates are required during real-system rollouts. Second, the absolute planning frequencies achieved in this experiment are lower than desirable for contact-rich manipulation. Although our asynchronous controller can repeat actions at a fixed servo rate, performance degrades when planning updates arrive too slowly, since the executed action sequence becomes increasingly stale.

The main driver of these low frequencies becomes evident in Figure 6.5, which characterizes runtime scaling with respect to the planning horizon length. We fix 128 samples distributed across 6 domains and measure

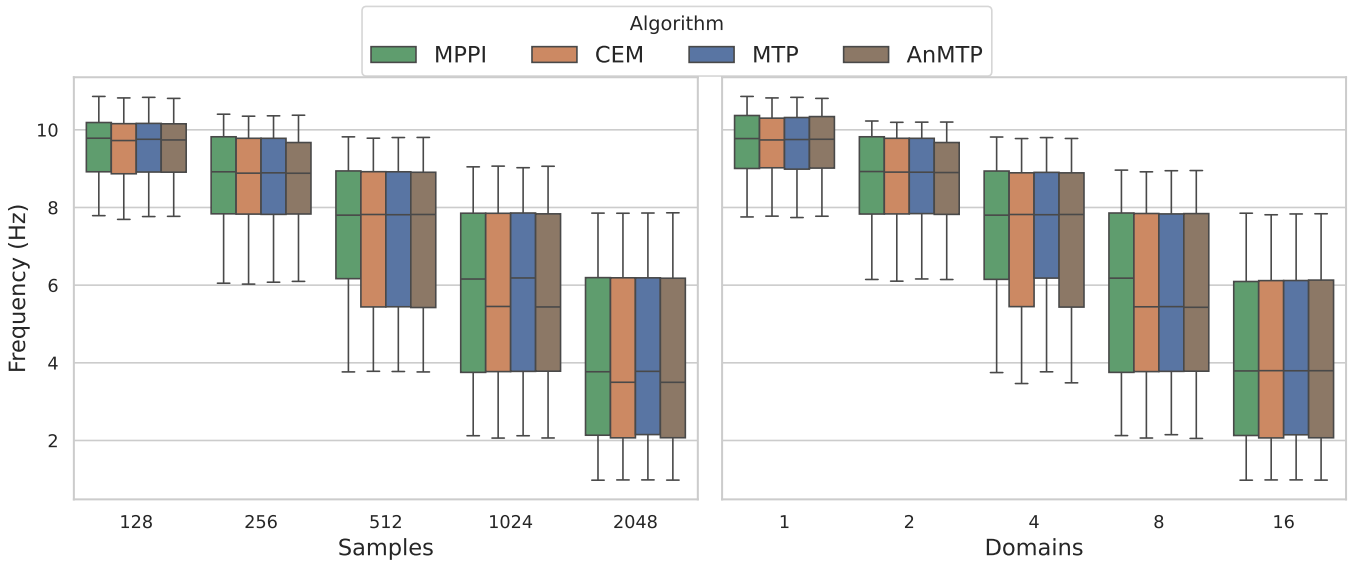


Figure 6.4.: Runtime statistics over 200 planning steps for increasing numbers of samples or domains. The box plots show the median (middle line), 25th to 75th percentile (box), and the furthest data points (whiskers). For sample scaling, we fix the algorithm to use 1 domain; for domain scaling, we use 128 samples. Both sweeps use a simulation timestep of 0.01 s and 12 horizon steps with 4 simulation steps each.

execution time across various horizons. Investigation of the GPU trace in Figure 6.6 reveals that the dominant contributor to runtime is advancing the MJX physics solver.

Many related works [19, 39] report similar computational bottlenecks. Their proposed solutions typically rely on short planning horizons or low planning frequencies, very small sample budgets (e.g., [3] uses only 30–50 samples), or exclusively simulation-based experiments, where computational limitations can be mitigated by operating in sub-realtime.

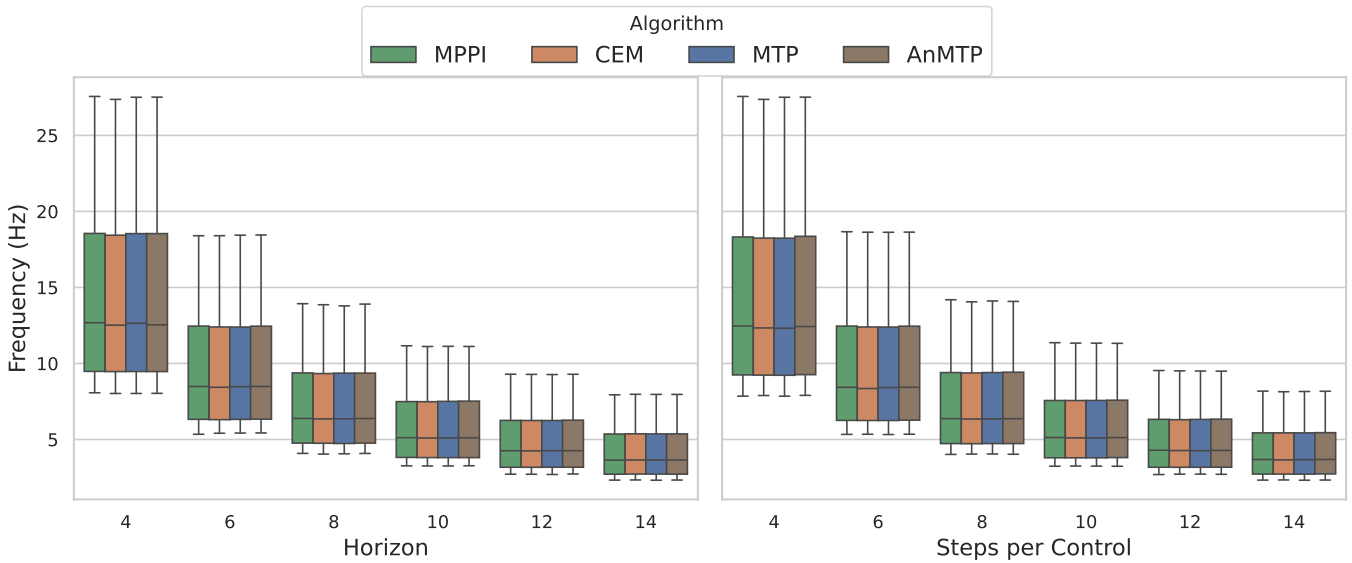


Figure 6.5.: Runtime statistics over 200 planning steps for various horizons. The box plots show the median (middle line), 25th to 75th percentile (box), and the furthest data points (whiskers). For each trial, we fix 128 samples across 6 domains and use a simulation timestep of 0.01 s.



Figure 6.6.: GPU trace of a single jit-ed optimization step as in Algorithm 5 running base MPPI over 15 horizon steps on the FR3 Push T task. The 15 steps encapsulating `sim_steps_per_control_step` physics steps (small purple chunks) dominate the runtime.

6.4. Push T on the FR3

Based on the scaling analysis from Section 6.3, we evaluate CEM, MPPI, PS, and MTP on the real robotic system described in Section 5.2.1. Unlike simulation, where planning frequency can be increased by reducing solver timesteps (and compensating with longer horizons), real-world deployment imposes strict real-time constraints. In the Push-T experiments, we target planning rates on the order of 8–10 Hz to avoid executing overly stale action sequences between replans. The scaling results in Section 6.3 show that achieving these planning rates requires fewer solver iterations and coarser timesteps. This shortens effective horizons and tends to degrade performance across all algorithms, yielding a trade-off between computational budget and contact fidelity.

With larger timesteps, contact dynamics in MuJoCo become more sensitive to solver and contact-parameter choices. In particular, larger inter-frame geometry displacements can lead to deep single-iteration penetrations, after which simulation may be dominated by constraint resolution via the acceleration model in Equation (5.2).

In our experiments, this manifested as spiky normal reaction forces and an artificially sticky contact regime that obscures frictional effects. Mitigating these artifacts required extensive parameter tuning. For example, in our MuJoCo setup, timesteps above 0.025 s consistently destabilized rollouts. At the same time, we aim to maintain at least 1000 samples in parallel to accommodate future domain randomization budgets.

Related work reports similar computational bottlenecks. For instance, [39] uses a 0.04 s timestep in Isaac Gym while limiting sample counts to a few hundred. Likewise, [58] reports ~ 50 Hz planning only through a heavily simplified quadruped model (impratio=100, single-iteration solver), while complex tasks (e.g., box-jumping) can require 30 s per step, which is consistent with our scaling observations.

Early hardware trials confirmed that Rolling Control Arrays (Section 4.1.1) are essential for maintaining stability; at planning frequencies of 8–10 Hz, a standard zero-order hold without sequence shifting leads to the execution of stale actions, causing noticeable performance degradation and lag. Furthermore, in contrast to simulation environments where noise is minimal, the application of Savitzky-Golay filtering to the control sequences yielded objectively less jittery behavior on the physical hardware without compromising task performance. Spline-based smoothing for local samples proved significantly more difficult to tune and offered no clear advantage over the Savitzky-Golay approach in our setup.

3 DoF setting For the 3 DoF Push-T setup, we idealize T dynamics through joint modeling and focus tuning on end-effector–T interactions, largely neglecting table contacts. Optimal performance uses low joint friction and damping. Pyramidal friction cones enable a 0.015 s timestep. To reduce artificial stickiness, we decrease solimp parameters (preserving frictional acceleration despite penetrations) and increase the cone width for smoother responses, while using slightly more solver iterations than recommended.

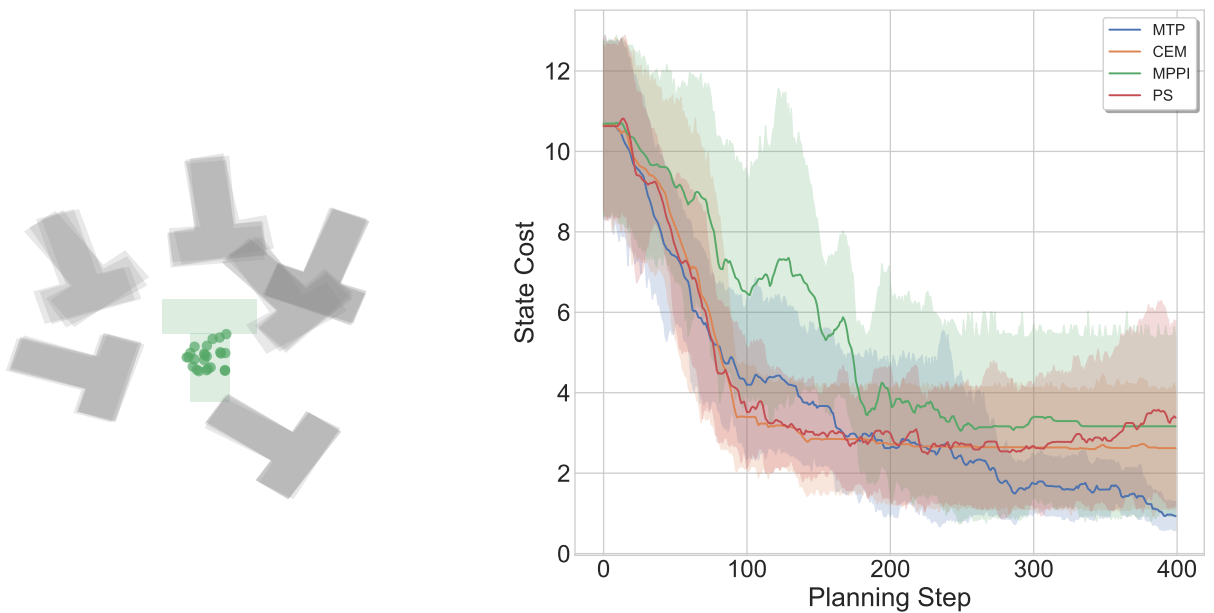


Figure 6.7.: Comparison of CEM, MPPI, MTP, and PS on the 3 DoF Push T task. All methods use 1024 samples at 10 Hz planning. Hyperparameters are reported in Appendix A.5.

In simple straight-line pushing, all algorithms perform comparably. We therefore again analyze challenging configurations that require strategy adaptation (e.g., maneuvering to re-establish contact). Across 6 configurations with initial position noise, Figure 6.7 is consistent with the simulation trends: CEM, MPPI, and PS

struggle with adaptation—converging to local minima (MPPI/CEM) or overshooting (PS). MTP achieves the lowest final error and variance, although the attractor term prevents exact zero-error convergence. Real-world contact penetrations are slightly lower than in sim-to-sim rollouts, since OptiTrack state updates correct accumulated inter-step drift.

Two MTP-specific effects become more prominent than in simulation. First, MTP trajectories exhibit higher variance, approaching PS rather than CEM-like smoothness. Second, MTP occasionally produces violent near-contact interactions that are not observed in MPPI/CEM. We attribute both to global samples combined with coarse timesteps: high-velocity loops can emerge in global distributions despite conservative elite perturbations, exceeding solver resolution and temporarily preventing effective T displacement. MTP typically recovers quickly, but can temporarily underperform (e.g., steps 100–200).

7 DoF setting While the 3 DoF setting transfers well, it is a significant simplification. T dynamics depend on task-specific joint friction/damping parameters, assume idealized table contacts, and omit full rigid-body physics. The 7 DoF setting restores rigid-body dynamics without task-specific simplifications, but introduces substantial new challenges.

Pose estimation is far more demanding than 3 DoF tracking. Small OptiTrack errors can cause table contact loss or penetration-induced normal-force spikes; tilting exacerbates both, widening the sim-to-real gap. During rollouts, simulated T tilting into the table can trigger reactive normal forces. Instead of sliding after stiction is overcome, the simulated T may “hop”, repeatedly breaking contacts and increasing unpredictability. Elliptic friction cones over slightly longer horizons mitigate this effect to some extent, but this requires us to increase timesteps to 0.025 s since more simulation steps hurt performance too much. This slows solver convergence and requires more iterations. As a result, we reduce planning to 8 Hz for 1024 samples. Low impratio settings stabilize table interactions but risk missing brief edge contacts under coarse timesteps, which can induce large, unexpected pose changes due to long moment arms about the COM.

Overall, all algorithms exhibit degraded performance in the 7 DoF scenario, primarily due to sim-to-real discrepancies that manifest as jittery contacts, small overshoots, and mismatches between expected and observed T behavior. These artifacts can lead to singularities or self-collisions when the T is pushed too close to the robot base or too far outward. We therefore focus evaluation on challenging configurations and compute statistics across multiple runs with small variations in T pose and end-effector initial position.

Figure 6.8 suggests that MTP still performs best on average, albeit with a reduced margin. The remaining gap is largely explained by occasional initiation of unwanted near-contact interactions (e.g., at the beginning of the third setting). Variance increases substantially across all methods despite only small perturbations in initial conditions. MPPI often avoids contact entirely, consistent with its behavior in simulation. PS starts strongly but degrades due to over-aggressiveness. CEM produces smoother samples and more stable contacts but remains prone to local minima, particularly after initial pose correction exhausts local reward signals. MTP performs best overall and achieves the lowest variance in final performance.

Results: Real-to-Sim-to-Real Deployment

MTP achieves best average performance and lowest variance on real FR3 hardware at 8-10 Hz planning frequencies with 1024 samples. MPPI often fails to initiate contact at all, PS exhibits over-aggressive behavior leading to degradation and CEM produces smooth contacts but gets trapped in local minima. MTP’s structured exploration enables reliable mode-switching in contact-rich manipulation under real-time constraints.

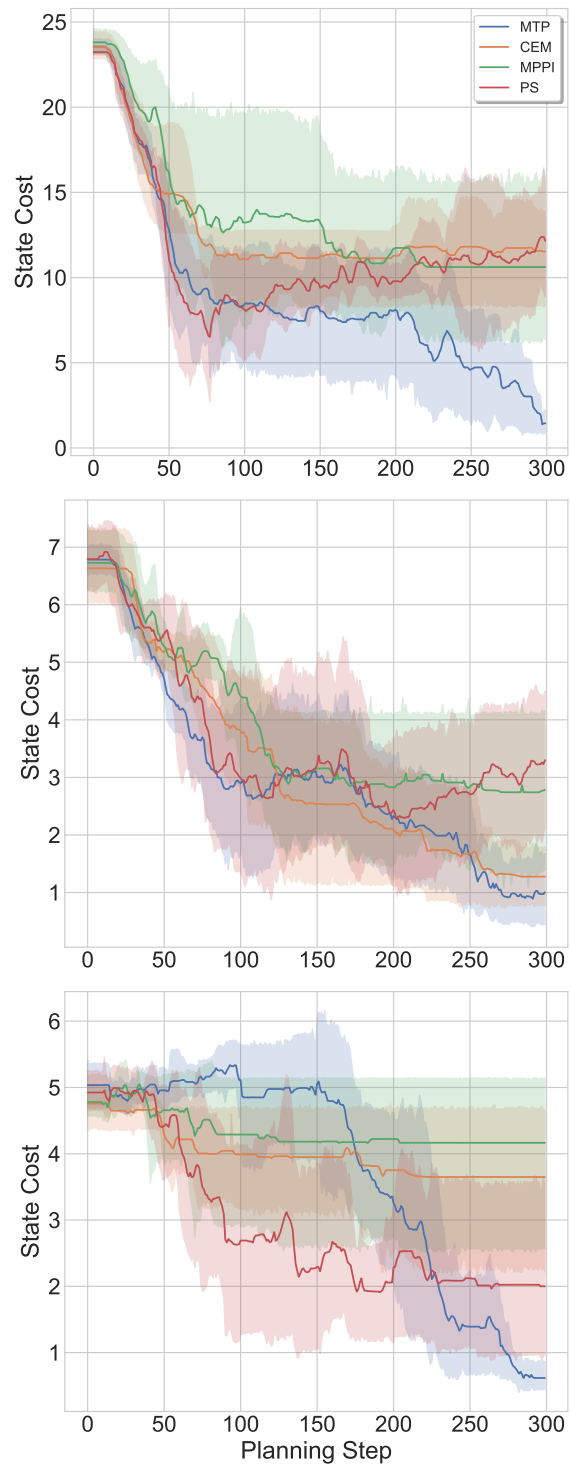
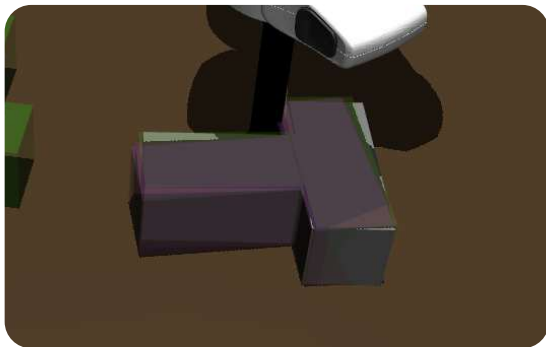


Figure 6.8.: Real-to-sim-to-real results with the 7 DoF T model across 6 seeds with pose/end-effector variations. All methods use 1024 samples at 8 Hz. Hyperparameters are reported in Appendix A.5.

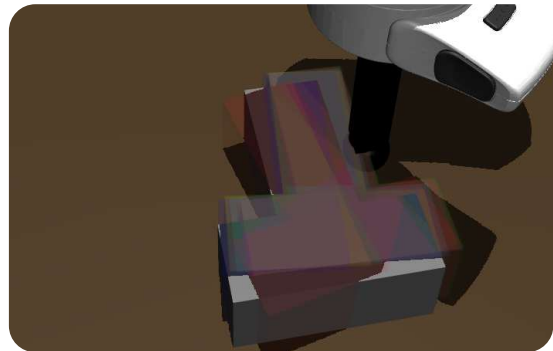
6.5. Domain Randomization

We finally investigate whether online domain randomization (DR) can improve robustness of sampling-based MPC under sim-to-real discrepancies in the Push-T task. The central question is whether the available sample budget can be traded for parallel domains without sacrificing control quality, and whether real-world feedback can be used to adapt the domain distribution online as explained in Section 2.5.1.

In the following, *adaptive* DR refers to updating the domain distribution based on the pose error according to Section 4.2. Push-T typically requires on the order of 150 – 200 samples for reliable performance, which motivates splitting the sample budget into multiple domains. In simulation, however, randomizing global physics parameters such as tangential table friction or the total mass of the T object does not yield a stable or informative feedback signal at replanning time. As visualized in Figure 6.9, discrepancies across domains do not vary smoothly with the randomized parameter values. A qualitative exception appears in the mass randomization plot (Figure 6.9b), where the domain highlighted in red exhibits substantially different behavior. This domain used a total object mass of 50 g, i.e., far below the range we considered plausible for matching the real setup, and masses within reasonable ranges produced rollouts that were difficult to distinguish at replanning time. Consistent with this observation, independently varying top and bottom masses to shift the center of mass within reasonable ranges did not improve the signal.



(a) Domain randomization over tangential table friction.



(b) Domain randomization over the total mass of the T object (red: 50 g).

Figure 6.9.: Simulation rollouts under domain randomization. Differences across domains are largely driven by contact initiation and constraint resolution rather than by a smooth dependence on global physics parameters.

Given the weak per-step signal, adaptive reweighting exhibits a characteristic failure mode. Figure 6.10 shows the evolution of domain weights for mass randomization: the distribution tends to drift toward heavier masses over time. This is consistent with an asymmetric error signal under short horizons: overly light objects overshoot and generate large pose errors, whereas overly heavy objects move less and can appear spuriously consistent with the observation, leading the update to prefer high-mass domains. Overall, this behavior makes feedback-based identification of global physics parameters unreliable at replanning time, and we did not observe stable improvements from adaptive strategies that resample domain parameters based on this signal.

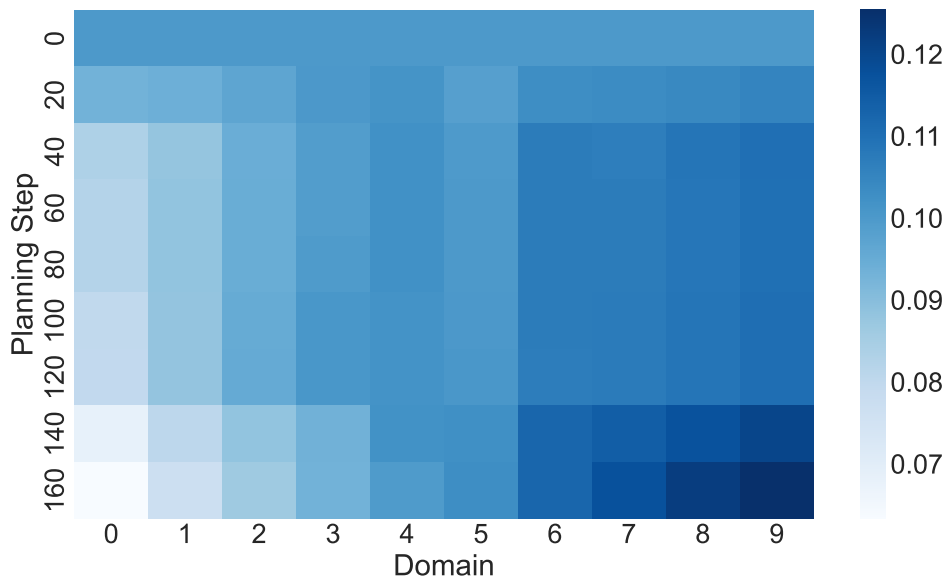


Figure 6.10.: Evolution of domain weights under adaptive domain randomization for mass parameters.

Randomizing Contact Initiation

While global physics parameters produced weak feedback, we found that informative signals can be obtained primarily for parameters that directly affect *contact initiation*. Based on the simulation observations, we therefore focus on randomizing contact margins on the real system, specifically the end-effector. To test this, we roll out all algorithms without the additional manual calibration step outlined in Section 5.4.4. In this setting, domain randomization improves robustness compared to a single fixed model by reducing sensitivity to small frame-alignment and contact-timing mismatches.

Figure 6.11 shows improved final pose quality under margin randomization: both randomized variants achieve lower final error, whereas the non-randomized baseline tends to increase its error toward the end due to repeated overshooting. Across multiple seeds, both uniform DR and adaptive reweighting outperform the non-randomized baseline, indicating that allocating part of the sample budget to multiple domains can be beneficial when the randomized parameter controls the discrete event of making (or missing) contact. However, within this regime it remains unclear whether adaptive reweighting is consistently superior to uniform weighting, as the observed margin is small and requires evaluation over a larger number of seeds to obtain more robust statistics.

To better understand the behavior of the adaptive variant, Figure 6.12 visualizes the evolution of domain weights over a single rollout for margin randomization. Compared to the mass-randomization case (Figure 6.10), the weights show more structured and interpretable shifts over time, which is consistent with the fact that margins directly modulate contact activation and therefore affect the observed pose error more reliably. In particular, we observe that during turning phases of the T object (approximately steps 18–37 and 55–74), the distribution shifts toward positive margins, whereas during predominantly straight pushing phases (approximately steps 37–55 and 74–93) the weights shift away from positive margins. This supports the qualitative hypothesis that different interaction modes benefit from different effective contact buffers, and that contact-initiation parameters yield a feedback signal that is informative enough to track such mode-dependent preferences online.

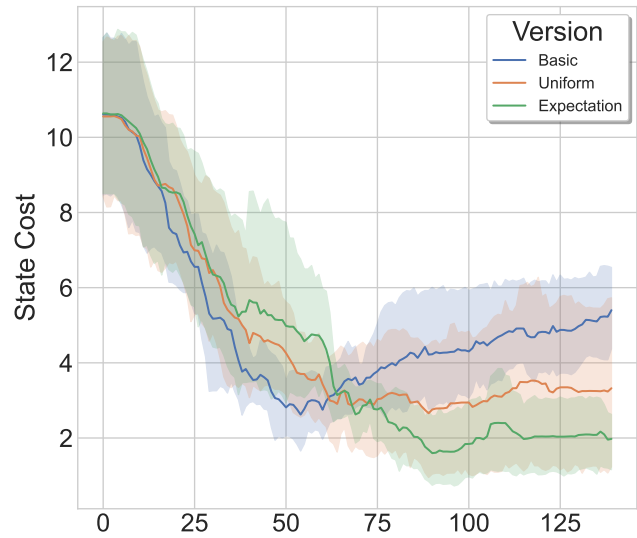
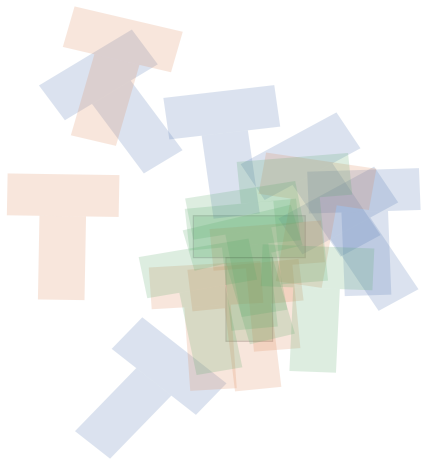


Figure 6.11.: Real-system performance over 6 seeds with 160 samples each for 8 domains at 5 Hz. Left: final object pose relative to the goal pose; right: state cost over planning steps.

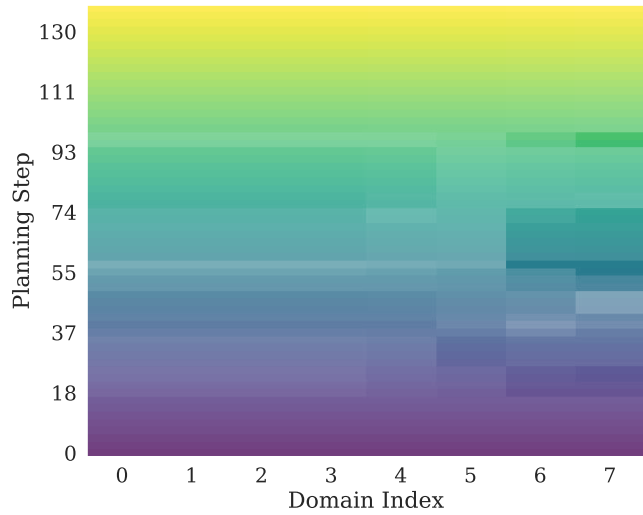


Figure 6.12.: Exemplary evolution of domain weights for the domain-randomized Push-T task under margin randomization. Time is encoded by color and weight magnitude by transparency. Margins are randomized within ± 1 cm around the end-effector.

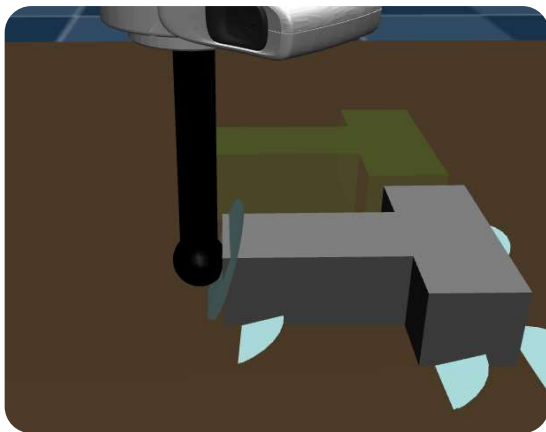
Failure Modes

The limited effectiveness of adaptive DR can be explained by several intertwined factors.

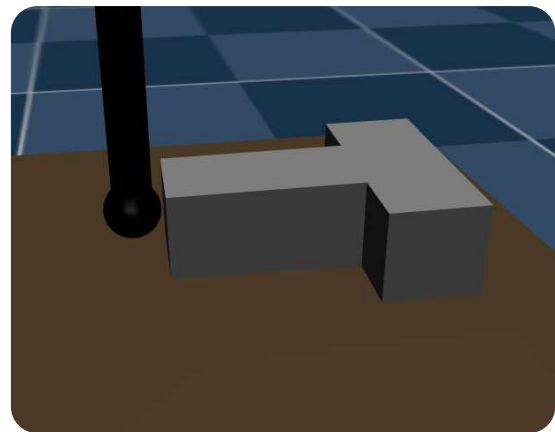
First, the MPC setting restricts feedback to replanning times: since the executed control sequence changes after each plan, comparisons are necessarily short-term, and small changes in continuous-time physics parameters (e.g., friction, mass) often do not produce a separable pose error over a single replanning interval. In contrast, contact initiation parameters can change the qualitative mode of the dynamics (contact vs. no-contact), which yields a stronger signal. Lower frequencies are possible in theory, since they would enlarge the time window over which predicted and observed outcomes can be compared; however, empirical tests revealed that frequencies below 5 Hz resulted in jerky rollouts and degraded performance, and even 5 Hz tended to perform worse than the 10 Hz used in our non-DR experiments. As a result, we cannot simply “wait longer” between replanning steps to obtain a cleaner identification signal without compromising closed-loop control quality.

Second, the observed behavior in simulation is frequently dominated by constraint resolution and accurate contact initiation, which reduces the relative influence of the randomized parameters and makes the feedback effectively non-identifiable. Improving numerical stability would require smaller timesteps and/or more solver iterations, but this directly reduces the compute budget available for domains and horizon length, making large-scale DR impractical.

Finally, the real-to-sim pipeline is highly sensitive to mismatches that affect contact timing. In particular, intermittent contact and complete contact loss can occur during execution; in the latter case, the dominant physics change qualitatively from frictional pushing to near free-body motion, which invalidates the assumptions behind state-based feedback and can destabilize adaptive updates. A kinematics issue and small table/OptiTrack frame mismatches caused significant challenges in this regard, as they can induce systematic end-effector drift relative to the table frame and thereby shift contact conditions across the workspace (Figure 6.13). To isolate the source of these effects, we compared kinematics computed from the URDF ([59]) against the MuJoCo XML model ([61]), but did not observe a meaningful difference, suggesting that residual errors are likely caused by calibration and frame-alignment.



(a) Stable contact activates frictional effects, full table interaction.



(b) End-effector drift and table/OptiTrack mismatch can lead to contact loss.

Figure 6.13.: Effect of kinematic and frame-alignment errors: for the same nominal setting, different workspace regions can exhibit qualitatively different contact behavior.

Result: Domain Randomization

Domain randomization over global physics parameters like mass and friction yields weak adaptation signals at high MPC replanning frequencies. In contrast, contact-initiation parameters such as collision margins produce more interpretable, mode-dependent weight evolution and improve robustness to kinematic misalignment in real-world deployment. While adaptive reweighting shows marginal benefit over uniform averages, active resampling of the domain distribution did not prove feasible under the strict real-time computational constraints of the pipeline.

7. Conclusion and Future work

7.1. Conclusion

This thesis developed and validated a complete real-to-sim-to-real pipeline that couples Model Tensor Planning (MTP) and baseline SMPC algorithms with GPU-accelerated MuJoCo MJX simulation and a Franka Research 3 robot on the Push-T manipulation benchmark. Across complementary evaluations—Bugtrap Escape, sim-to-sim Push-T, and real-world Push-T—MTP consistently demonstrated stronger performance than their baselines, particularly in scenarios that require mode switching and robust behavior in contact-rich dynamics.

On the algorithmic side, Bugtrap Escape verified that MTP’s mixed sampling strategy provides reliable exploration in an extreme local-minimum setting where standard baselines frequently fail. In Push-T, MTP retained this advantage in both simulation and hardware, achieving the best overall success/robustness while operating under practical planning-frequency constraints. Beyond benchmarking, the thesis contributed a set of engineering and systems insights on how solver settings, horizon length, and cost design interact with Monte Carlo sampling efficiency in contact-rich SMPC.

Finally, the domain randomization study clarified when DR provides useful robustness signals in SMPC and when it does not. Global physics parameters, such as mass and friction, often yield weak or ambiguous feedback at MPC time scales due to the dominance of contact initiation and constraint resolution effects. Conversely, contact-initiation parameters (e.g., collision margins) produce more interpretable, mode-dependent weight evolution and improve robustness in sim-to-real deployment. However, even with these improved signals, the feedback proved too noisy to support stable adaptive resampling of the domain distribution. Overall, the results support MTP as a practical sampling-based MPC method for contact-rich manipulation, while highlighting fundamental challenges for adaptive DR under short horizons and tight real-time budgets.

7.2. Future Work

This section briefly outlines follow-up directions motivated by the main bottlenecks and failure modes observed in this thesis, with an emphasis on improving real-world robustness and reducing brittleness under tight compute budgets.

Sim-to-Real Gap

A clear next step is to revisit the simulation backend used for large-scale rollouts. MuJoCo Warp (MJWarp) [10] provides a GPU-optimized MuJoCo implementation built on NVIDIA Warp, and it is explicitly motivated by limitations encountered with MJX in more complex and contact-heavy settings. Even when raw step-time

improvements are modest, MJWarp’s different implementation stack and contact/constraint handling may reduce practical pain points that showed up in this thesis, such as sensitivity to branching-heavy contact patterns and the need for extensive solver and numerical tuning to obtain stable behavior. Concretely, a valuable follow-up would be a controlled A/B evaluation across CPU MuJoCo, MJX, and MJWarp.

Locality of MPC

Independent of simulator choice, sampling-based MPC remains fundamentally limited by its receding-horizon nature: in practice, real-time constraints force short horizons and weaken long-horizon credit assignment, especially in contact-rich manipulation where progress can be sparse and mode changes are essential. A promising direction is therefore to augment planning with learned long-horizon structure, e.g., via learned terminal costs/value surrogates or generative predictive models that bias sampling toward globally meaningful modes while retaining MPC’s reactivity; flow-matching based approaches are one concrete example of how prior simulated experience (and potentially demonstrations) can be injected into the planning distribution. Alternatively, demonstration-regularized objectives (trajectory similarity or goal-conditioned priors) could provide a global guidance signal that reduces indecision and oscillations near mode switches without requiring substantially longer horizons.

Camera-based Pose Estimation

A natural extension of the real-world pipeline is to replace the OptiTrack motion-capture system with an RGB-D camera setup and estimate the T-object pose using, e.g., FoundationPose [52]. Compared to OptiTrack, this is expected to introduce higher measurement noise, occasional outliers, and additional latency, but it better reflects the sensing conditions of many practical manipulation systems and would therefore provide a more realistic stress test for closed-loop planning under partial observability. In this regime, domain randomization may become even more beneficial: without motion-capture-based ground truth and manual frame-alignment calibration, robustness must come from explicitly modeling perception- and calibration-induced uncertainty and planning in a way that tolerates these mismatches.

Bibliography

- [1] Ian Abraham et al. “Model-Based Generalization Under Parameter Uncertainty Using Path Integral Control”. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 2864–2871. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.2972836. arXiv: 2006.03106 [cs]. (Visited on 07/25/2025).
- [2] Muhammad Ali et al. “A New Approach to Empirical Mode Decomposition Based on Akima Spline Interpolation Technique”. In: *IEEE Access* 11 (2023), pp. 67370–67384. DOI: 10.1109/ACCESS.2023.3253279.
- [3] Juan Alvarez-Padilla et al. *Real-Time Whole-Body Control of Legged Robots with Model-Predictive Path Integral Control*. Sept. 2024. DOI: 10.48550/arXiv.2409.10469. arXiv: 2409.10469 [cs]. (Visited on 01/05/2026).
- [4] Isin M. Balci et al. *Constrained Covariance Steering Based Tube-MPPI*. Apr. 2022. DOI: 10.48550/arXiv.2110.07744. arXiv: 2110.07744 [math]. (Visited on 07/25/2025).
- [5] Mohak Bhardwaj et al. *STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation*. Sept. 2021. DOI: 10.48550/arXiv.2104.13542. arXiv: 2104.13542 [cs]. (Visited on 08/07/2025).
- [6] James Bradbury et al. *JAX: Composable transformations of Python+NumPy programs*. Version 0.3.13. Software repository. 2018. URL: <https://github.com/google/jax> (visited on 03/04/2026).
- [7] Lara Bruder Müller et al. “Generative Models from and for Sampling-Based MPC: A Bootstrapped Approach for Adaptive Contact-Rich Manipulation”. In: *IEEE Robotics and Automation Letters* (2026).
- [8] Kanti Datta. “Linear system theory and design, by Chi-Tsong Chen, Oxford University Press, New York, 1999, 334 pages, ISBN 0-19-511777-8”. In: *International Journal of Robust and Nonlinear Control* 10 (Dec. 2000). DOI: 10.1002/1099-1239(20001230)10:15<1360::AID-RNC539>3.0.CO;2-C.
- [9] DeepMind. *MuJoCo Documentation: Computation*. 2025. URL: <https://mujoco.readthedocs.io/en/stable/computation/index.html#geintegration> (visited on 10/05/2025).
- [10] DeepMind. *MuJoCo Documentation: MuJoCo Warp (MJWarp)*. 2025. URL: <https://mujoco.readthedocs.io/en/latest/mjwarp/> (visited on 03/03/2026).
- [11] DeepMind. *MuJoCo Documentation: MuJoCo XLA (MJX)*. 2025. URL: <https://mujoco.readthedocs.io/en/stable/mjx.html> (visited on 10/05/2025).
- [12] Stephane Doncieux et al. “Evolutionary Robotics: What, Why, and Where To”. In: *Frontiers in Robotics and AI Volume 2 - 2015* (2015). ISSN: 2296-9144. DOI: 10.3389/frobt.2015.00004.
- [13] Franka Emika GmbH. *Product Manual Franka Research 3*. 1.0. Franka Emika GmbH. 2022. URL: https://franka.de/hubfs/Product%20Manual%20Franka%20Research%203_R02210_1.0_EN.pdf.
- [14] Gene Franklin, J.D. Powell, and M.L. Workman. *Digital Control of Dynamic Systems-Third Edition*. Nov. 2022. ISBN: 0-9791226-3-5 or 978-0-9791226-3-7.

-
- [15] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML]. URL: <https://arxiv.org/abs/1807.02811>.
- [16] C. Daniel Freeman et al. *Brax: A Differentiable Physics Engine for Large Scale Rigid Body Simulation*. Version 0.13.0. Software repository. 2021. URL: <https://github.com/google/brax> (visited on 10/25/2025).
- [17] Michael Görner et al. “MoveIt! Task Constructor for Task-Level Motion Planning”. In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 190–196. DOI: 10.1109/ICRA.2019.8793898.
- [18] Lars Grüne and Jürgen Pannek. “Nonlinear Model Predictive Control”. In: Apr. 2011, pp. 43–66. ISBN: 978-0-85729-500-2. DOI: 10.1007/978-0-85729-501-9_3.
- [19] Adrian Hess et al. *Sampling-Based Model Predictive Control for Dexterous Manipulation on a Biomimetic Tendon-Driven Hand*. Aug. 2025. DOI: 10.48550/arXiv.2411.06183. arXiv: 2411.06183 [cs]. (Visited on 08/14/2025).
- [20] Hannes Homburger et al. *Optimality and Suboptimality of MPPI Control in Stochastic and Deterministic Settings*. Feb. 2025. DOI: 10.48550/arXiv.2502.20953. arXiv: 2502.20953 [eess]. (Visited on 06/10/2025).
- [21] Radu Horaud and Fadi Dornaika. “Hand-Eye Calibration”. In: *The International Journal of Robotics Research* 14.3 (June 1995), pp. 195–210. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/027836499501400301. arXiv: 2311.12655 [cs]. (Visited on 08/29/2025).
- [22] Taylor Howell et al. *Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo*. Dec. 2022. DOI: 10.48550/arXiv.2212.00541. arXiv: 2212.00541 [cs]. (Visited on 11/08/2025).
- [23] Vince Kurtz. *Hydrax*. Software repository. 2024. URL: <https://github.com/vincekurtz/hydrax> (visited on 03/04/2026).
- [24] Vince Kurtz and Joel W. Burdick. *Generative Predictive Control: Flow Matching Policies for Dynamic and Difficult-to-Demonstrate Tasks*. May 2025. DOI: 10.48550/arXiv.2502.13406. arXiv: 2502.13406 [cs]. (Visited on 05/30/2025).
- [25] Alexander Lambert et al. *Stein Variational Model Predictive Control*. Apr. 2021. DOI: 10.48550/arXiv.2011.07641. arXiv: 2011.07641 [cs]. (Visited on 06/17/2025).
- [26] W. Langson et al. “Robust Model Predictive Control Using Tubes”. In: *Automatica* 40.1 (2004), pp. 125–133. DOI: 10.1016/j.automatica.2003.08.009. URL: <https://www.sciencedirect.com/science/article/pii/S0005109803002838>.
- [27] Kody Law, A. Stuart, and K. Zygalakis. *Data Assimilation: A Mathematical Introduction*. Vol. 62. Oct. 2015. ISBN: 978-3-319-20324-9. DOI: 10.1007/978-3-319-20325-6.
- [28] An T. Le et al. *Model Tensor Planning*. May 2025. DOI: 10.48550/arXiv.2505.01059. arXiv: 2505.01059. (Visited on 05/22/2025).
- [29] N. Lehtomaki, N. Sandell, and M. Athans. “Robustness results in linear-quadratic Gaussian based multivariable control designs”. In: *IEEE Transactions on Automatic Control* 26.1 (1981), pp. 75–93. DOI: 10.1109/TAC.1981.1102565.
- [30] Jacky Liang et al. *GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning*. 2018. arXiv: 1810.05762 [cs.RO]. URL: <https://arxiv.org/abs/1810.05762>.
- [31] Anders Lindquist. “On Feedback Control of Linear Stochastic Systems”. In: *SIAM Journal on Control* 11.2 (1973), pp. 323–343. DOI: 10.1137/0311025. URL: <https://doi.org/10.1137/0311025>.
- [32] Steven Macenski et al. “Robot Operating System 2: Design, Architecture, and Uses in the Wild”. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.

-
- [33] Mayank Mittal et al. “Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning”. In: *arXiv preprint arXiv:2511.04831* (2025). URL: <https://arxiv.org/abs/2511.04831>.
- [34] Fabio Muratore et al. *Robot Learning from Randomized Simulations: A Review*. Jan. 2022. DOI: 10.48550/arXiv.2111.00956. arXiv: 2111.00956 [cs]. (Visited on 06/05/2025).
- [35] NaturalPoint Inc. *Flex 3 - Specs*. URL: <https://optitrack.com/cameras/flex-3/specs> (visited on 01/30/2026).
- [36] NVIDIA Corporation. *GeForce RTX 5090 Graphics Card*. URL: <https://www.nvidia.com/en-us/geforce/graphics-cards/50-series/rtx-5090/> (visited on 01/30/2026).
- [37] Andrej Orsula et al. *pymoveit2*. Version 4.2.0. Software repository. 2026. URL: <https://github.com/AndrejOrsula/pymoveit2> (visited on 01/30/2026).
- [38] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. 5th ed. San Francisco, CA: Morgan Kaufmann, 2013. ISBN: 0124077269.
- [39] Corrado Pezzato et al. *Sampling-Based Model Predictive Control Leveraging Parallelizable Physics Simulations*. Jan. 2025. DOI: 10.48550/arXiv.2307.09105. arXiv: 2307.09105 [cs]. (Visited on 05/28/2025).
- [40] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 2012.
- [41] Cristina Pinneri et al. *Sample-Efficient Cross-Entropy Method for Real-time Planning*. Aug. 2020. DOI: 10.48550/arXiv.2008.06389. arXiv: 2008.06389 [cs]. (Visited on 05/28/2025).
- [42] J. Rawlings, D.Q. Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*. Jan. 2017.
- [43] Abraham Savitzky and Marcel J. E. Golay. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures”. In: *Analytical Chemistry* 36.8 (1964), pp. 1627–1639.
- [44] Max Schwenzer et al. “Review on model predictive control: An engineering perspective”. In: *The international journal of advanced manufacturing technology* 117.5 (2021), pp. 1327–1349.
- [45] R.S. Sutton and A.G. Barto. “Reinforcement Learning: An Introduction”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 1054–1054. DOI: 10.1109/TNN.1998.712192.
- [46] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A Physics Engine for Model-Based Control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.
- [47] Elia Trevisan and Javier Alonso-Mora. “Biased-MPPI: Informing Sampling-Based Model Predictive Control by Fusing Ancillary Controllers”. In: *IEEE Robotics and Automation Letters* 9.6 (June 2024), pp. 5871–5878. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2024.3397083. arXiv: 2401.09241 [cs]. (Visited on 07/09/2025).
- [48] R.Y. Tsai and R.K. Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration”. In: *IEEE Transactions on Robotics and Automation* 5.3 (1989), pp. 345–358. DOI: 10.1109/70.34770.
- [49] Bogdan Vlahov et al. “Low Frequency Sampling in Model Predictive Path Integral Control”. In: *IEEE Robotics and Automation Letters* 9.5 (May 2024), pp. 4543–4550. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2024.3382530. arXiv: 2404.03094 [cs]. (Visited on 06/17/2025).
- [50] Bogdan Vlahov et al. *MPPI-Generic: A CUDA Library for Stochastic Trajectory Optimization*. Mar. 2025. DOI: 10.48550/arXiv.2409.07563. arXiv: 2409.07563 [cs]. (Visited on 06/13/2025).
- [51] Ziyi Wang et al. *Variational Inference MPC Using Tsallis Divergence*. Apr. 2021. DOI: 10.48550/arXiv.2104.00241. arXiv: 2104.00241 [cs]. (Visited on 06/17/2025).

-
- [52] Bowen Wen et al. “FoundationPose: Unified 6D Pose Estimation and Tracking of Novel Objects”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024, pp. 17868–17879.
- [53] Daan Wierstra et al. “Natural evolution strategies”. In: *J. Mach. Learn. Res.* 15 (2011), pp. 949–980. URL: <https://api.semanticscholar.org/CorpusID:223633388>.
- [54] Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. “Model Predictive Path Integral Control: From Theory to Parallel Computation”. In: *Journal of Guidance, Control, and Dynamics* 40.2 (Feb. 2017), pp. 344–357. ISSN: 0731-5090, 1533-3884. DOI: 10.2514/1.G001921. (Visited on 05/28/2025).
- [55] Grady Williams et al. *Information Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving*. July 2017. DOI: 10.48550/arXiv.1707.02342. arXiv: 1707.02342 [cs]. (Visited on 06/10/2025).
- [56] Grady Williams et al. “Information Theoretic MPC for Model-Based Reinforcement Learning”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore: IEEE, May 2017, pp. 1714–1721. DOI: 10.1109/icra.2017.7989202. (Visited on 07/26/2025).
- [57] Haoru Xue et al. *DIAL-MPC: Diffusion-Inspired Annealing For Legged MPC*. URL: <https://github.com/LeCAR-Lab/dial-mpc/tree/main> (visited on 12/28/2025).
- [58] Haoru Xue et al. *Full-Order Sampling-Based MPC for Torque-Level Locomotion Control via Diffusion-Style Annealing*. Sept. 2024. DOI: 10.48550/arXiv.2409.15610. arXiv: 2409.15610 [cs]. (Visited on 05/30/2025).
- [59] Baris Yazici, Andreas Kuehner, et al. *franka_ros2*. Version 3.1.1. Git branch: jazzy; software repository. 2026. URL: https://github.com/frankarobotics/franka_ros2 (visited on 01/30/2026).
- [60] Zeji Yi et al. *CoVO-MPC: Theoretical Analysis of Sampling-based MPC and Optimal Covariance Design*. Jan. 2024. DOI: 10.48550/arXiv.2401.07369. arXiv: 2401.07369 [cs]. (Visited on 08/18/2025).
- [61] Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. *MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo*. 2022. URL: http://github.com/google-deepmind/mujoco_menagerie.
- [62] Yuezhe Zhang et al. “Multi-Modal MPPI and Active Inference for Reactive Task and Motion Planning”. In: *IEEE Robotics and Automation Letters* 9.9 (Sept. 2024), pp. 7461–7468. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2024.3426183. arXiv: 2312.02328 [cs]. (Visited on 06/17/2025).

A. Appendix

A.1. Experimental Parameters

A.1.1. Sim-to-sim

Table A.1.: Hyperparameters for the BugTrap escape task comparing MTP, CEM, MPPI, and Predictive Sampling.

Hyperparameter	MTP	CEM	MPPI	PS
Total Samples (N)	128	128	128	128
Planning Horizon (T)	24	24	24	24
Sim Steps per Control	3	3	3	3
Planning Frequency (Hz)	25	25	25	25
Inverse Temperature (λ)	0.01	–	0.01	–
Number of Elites (E)	8	8	–	–
Noise Level / σ_{start}	0.2	0.2	0.2	0.2
Mixing Rate (β)	0.5	–	–	–
Graph Layers (M)	4	–	–	–
Graph Nodes (N_{nodes})	16	–	–	–
Alpha (α)	0.0	0.0	0.0	0.0

Table A.2.: Key MuJoCo/MJX simulation settings for the Push-T scenes (free 6-DoF object vs. jointed 3-DoF object) used in **sim-to-sim** testing.

Setting	Free object (6 DoF)	Jointed object (3 DoF)
Object model	Free joint (type="free")	2× slide + 1× hinge
Timestep [s]	0.025	0.025
Solver / Jacobian	Newton / dense	Newton / dense
Integrator	implicitfast	implicitfast
Friction cone	elliptic	elliptic
Solver iterations / LS iters	7 / 10	5 / 7
impratio	1.5	1.0
Contact budget (nconmax)	64	64
Max joints (njmax)	1200	1024
Custom limits	max_contact_points=32, maxhullverts=32, max_geom_pairs=128	max_contact_points=32, maxhullverts=32, max_geom_pairs=128
T geom friction	(0.3, 0.001, 0.01)	0.1
T geom solref	(0.05, 1.1)	(0.025, 1.2)
T geom solimp	(0.8, 0.85, 0.002, 0.5, 2)	(0.7, 0.9, 0.002, 0.5, 2)
T geom condim	3	3
Ground geom friction	0.3	0.4
Ground geom solref	(0.05, 1.5)	(0.025, 1.2)
Ground geom solimp	(0.5, 0.85, 0.002, 0.5, 2)	(0.1, 0.2, 0.003, 0.5, 2)
Ground geom condim	3	3
3-DoF joint dissipation	–	slides: frictionloss=0.1 damping=0.2; hinge: frictionloss=0.001, damping=0.001

Table A.3.: FR3 Push-T **sim-to-sim** hyperparameters for the 3 DoF vs 7 DoF settings.

Hyperparameter	MTP		CEM		MPPI		AnMTP	
	Joint	Free	Joint	Free	Joint	Free	Joint	Free
Total Samples (N)	256	256	256	256	256	256	256	256
Domains (D)	1	1	1	1	1	1	1	1
Horizon Steps (T)	15	15	15	15	15	15	15	15
Sim Steps per Control	1	1	1	1	1	1	1	1
Planning Frequency	20	20	20	20	20	20	20	20
Velocity Limits (m/s)	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35
Inverse Temperature (λ)	0.1	0.1	–	–	0.1	0.1	0.1	0.1
Number of Elites (E)	48	48	48	48	–	–	48	48
Keep Elites	1	1	–	–	–	–	1	1
Fixed Variance (σ)	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Mixing Rate (β)	0.4	0.4	–	–	–	–	0.4	0.4
Graph Layers (M)	3	3	–	–	–	–	3	3
Graph Samples (N)	64	64	–	–	–	–	64	64
Alpha (α)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Simulation timestep [s]	0.025	0.025	0.025	0.025	0.025	0.025	0.025	0.025

A.1.2. Sim-to-real

Table A.4.: Key MuJoCo/MJX simulation settings for the Push-T scenes (free 6-DoF object vs. jointed 3-DoF object) used in **real-to-sim-to-real** testing.

Setting	Free object (6 DoF)	Jointed object (3 DoF)
Object model	Free joint (type="free")	2× slide + 1× hinge
Timestep [s]	0.025	0.015
Solver / Jacobian	Newton / dense	Newton / dense
Integrator	implicitfast	implicitfast
Friction cone	elliptic	pyramidal
Solver iterations / LS iters	7 / 10	5 / 7
impratio	1.5	1.0
Contact budget (nconmax)	64	64
Max joints (njmax)	1200	1024
Custom limits	max_contact_points=32, maxhullverts=32, max_geom_pairs=128	max_contact_points=32, maxhullverts=32, max_geom_pairs=128
T geom friction	(0.3, 0.001, 0.01)	0.1
T geom solref	(0.05, 1.1)	(0.025, 1.2)
T geom solimp	(0.8, 0.85, 0.002, 0.5, 2)	(0.7, 0.9, 0.002, 0.5, 2)
T geom condim	3	3
Ground geom friction	0.3	0.4
Ground geom solref	(0.05, 1.5)	(0.025, 1.2)
Ground geom solimp	(0.5, 0.85, 0.002, 0.5, 2)	(0.1, 0.2, 0.003, 0.5, 2)
Ground geom condim	3	3
3-DoF joint dissipation	–	slides: frictionloss=0.1 damping=0.2; hinge: frictionloss=0.001, damping=0.001
	–	

Table A.5.: FR3 Push-T **real-to-sim-to-real** hyperparameters for the 3 DoF vs 7 DoF settings.

Hyperparameter	MTP		CEM		MPPI		PS	
	3 DoF	7 DoF	3 DoF	7 DoF	3 DoF	7 DoF	3 DoF	7 DoF
Total samples (N)	1024	1024	1024	1024	1024	1024	1024	1024
Domains (D)	1	1	1	1	1	1	1	1
Horizon steps (T)	13	10	13	10	13	10	13	10
Sim steps per control step	2	2	2	2	2	2	2	2
Horizon time [s]	0.39	0.5	0.39	0.5	0.39	0.5	0.39	0.5
Velocity limits	0.35	0.3	0.35	0.3	0.35	0.3	0.35	0.3
Inverse temperature (λ)	0.1	0.1	–	–	0.1	0.1	–	–
Number of elites (E)	72	72	72	72	–	–	–	–
Fixed variance (σ)	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Mixing rate (β)	0.35	0.25	–	–	–	–	–	–
Graph layers (M)	3	3	–	–	–	–	–	–
Graph samples	64	64	–	–	–	–	–	–
Alpha (α)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Simulation timestep [s]	0.015	0.025	0.015	0.025	0.015	0.025	0.015	0.025