



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

LEARNING ALGORITHMS AND SYSTEMS LABORATORY (LASA)

Model Predictive Control for Constraint Learning from Expert Demonstrations

Semester Project
COM-508

Author
VINCENT GUILHERME BOUZEREAU

Full Professor & Director of LASA
PROF. DR. AUDE BILLARD

Supervisor
BAIYU PENG

LASA
Learning Algorithms and
Systems Laboratory

Spring 2026

Contents

Abstract	2
Abbreviations	2
1 Introduction	3
1.1 Motivation	3
1.2 The PUCL-MPC gap	3
1.3 Contributions	4
1.4 Thesis outline	5
2 Background and Related Work	5
2.1 Model Predictive Control	5
2.2 Positive-Unlabeled Constraint Learning (PUCL)	5
3 Problem Formulation	6
3.1 Formal statement	6
3.2 Constraint orders considered	7
3.3 Assumptions	7
4 Method	7
4.1 State representation and dynamics	8
4.2 MPC formulation	9
4.3 PUCL constraint learning	10
4.4 The MPC policy block	10
4.5 Constraint injection into the NLP	11
5 Experiments	12
5.1 Benchmark environments	12
5.2 Baselines	12
5.3 Hypotheses	13
5.4 Metrics	13
5.5 Hyperparameters and evaluation protocol	15
6 Results and Discussion	15
6.1 Headline results	16
6.2 Per-environment scorecard [H2, H3]	17
6.3 PUCL constraint quality [H5]	18
6.4 Learning the acceleration and jerk constraints [H2]	19
6.5 MPC solver and real-time outlook [H4]	20
6.6 KUKA deployment outlook [C4]	21
6.7 Closed-loop control	21
6.8 Hyperparameter sensitivity	22
6.9 Did the MPC integrate correctly into the PUCL pipeline?	22
6.10 Recovering a tight keep-out with active collision feedback	23
6.11 Discussion	25
6.12 Comparison: PUCL-MPC vs. behaviour cloning [H1]	26
6.13 Limitations and threats to validity	26
7 Conclusion and Future Work	28
A Appendix	28

Abstract

Safety constraints for robot manipulation depend on obstacle placement, workspace context, and operator intent, so they change with every deployment and cannot be written down analytically in advance. Positive-Unlabeled Constraint Learning (PUCL), introduced by Peng & Billard [1], recovers an obstacle-avoidance constraint ζ_θ from expert demonstrations.

We replace PUCL’s reinforcement-learning policy block with a model-predictive-control (MPC) trajectory optimiser that enforces the learned ζ_θ as a hard nonlinear-program (NLP) row and supplies the forward rollouts driving PUCL’s two-step reliable-negative mining. Casting jerk as the control on a triple-integrator state writes the position, velocity, acceleration, and jerk constraints as identical hard rows, one formulation for all four orders. The explicit constraint is inspectable and reusable across starts and goals, and a feasible solve is free of enforced-row violations by construction. The reported pipeline is open-loop (one horizon solve, velocity-level deployment) and enforces only the position and velocity rows.

With the MPC as its forward policy, PUCL recovers the position keep-out on four 3-D geometries (single ellipsoid, two ellipsoids, concave, two regions) and matches or beats a binary cross-entropy classifier (BCE, the positive-unlabeled-step ablation) at per-order recall ~ 0.99 , while BCE’s jerk recall collapses on the non-convex scenes (0.13, 0.00). As a policy, the explicit constraint generalises far more safely than a constraint-free behaviour-cloning baseline on held-out starts and goals, cutting true collisions from 14–16 steps per episode to ≤ 0.2 . Acceleration is the weakest order, recovering its canonical cap only partially on a boundary-balanced sample, so recall is the metric we report. The zero enforced-row violations are by construction, not safety: a goal-reaching MPC never brackets the true boundary, so the label-free keep-out stays conservative (an identifiability limit) and the open-loop policy still crosses the true unsafe set on 25–47% of steps. A minimal binary collision probe lifts the position intersection-over-union from 0.015–0.049 to 0.60–0.90 at recall ~ 1.0 .

For the position case the controller is real-time capable: a warm-started acados SQP-RTI solve runs at a median 3.1 ms against 177 ms for IPOPT, a $57\times$ speed-up inside the 100 ms control step. Every reported result still uses the IPOPT backend, and the higher-order QP does not yet converge. All results are in simulation on a point-mass abstraction over five seeds. The next steps are to recover a tight label-free keep-out by making the MPC explore and to deploy on the KUKA iiwa-14.

Abbreviations

AUC	Area under the curve	LfD	Learning from demonstration
BC	Behaviour cloning (no-constraint policy baseline)	MLP	Multi-layer perceptron
BCE	Binary cross-entropy classifier (PU-step ablation baseline)	MPC	Model predictive control
CDF	Cumulative distribution function	NLP	Nonlinear program
CI	Confidence interval	OCP	Optimal control problem
CRL	Constrained reinforcement learning	PPO	Proximal policy optimization
DoF	Degrees of freedom	PU	Positive-unlabeled
DSM	Dynamical Systems Method	PUCL	Positive-Unlabeled Constraint Learning
EE	End-effector	QP	Quadratic program
FK	Forward kinematics	RL	Reinforcement learning
GMM	Gaussian mixture model	RN	Reliable negative
GUI	Graphical user interface	ROC	Receiver operating characteristic
HPIPM	High-performance interior-point method	RTI	Real-time iteration
IK	Inverse kinematics	SD	Standard deviation
IoU	Intersection over union	SQP	Sequential quadratic programming
IPOPT	Interior-point optimizer	URDF	Unified robot description format
KKT	Karush–Kuhn–Tucker	VSP	Violation-steps portion
kNN	k -nearest neighbours		
KPI	Key performance indicator		

1 Introduction

1.1 Motivation

Deploying robot manipulators in real tasks requires avoiding obstacles and respecting workspace restrictions that are specific to each scene and cannot be specified analytically at design time. Hand-coding these constraints is expensive, brittle, and does not generalise across task variants. The alternative is to *learn* the constraints from expert demonstrations: watch how a skilled operator moves the arm, infer the safety boundary that this behaviour implies, and embed that boundary in the controller (Fig. 1).

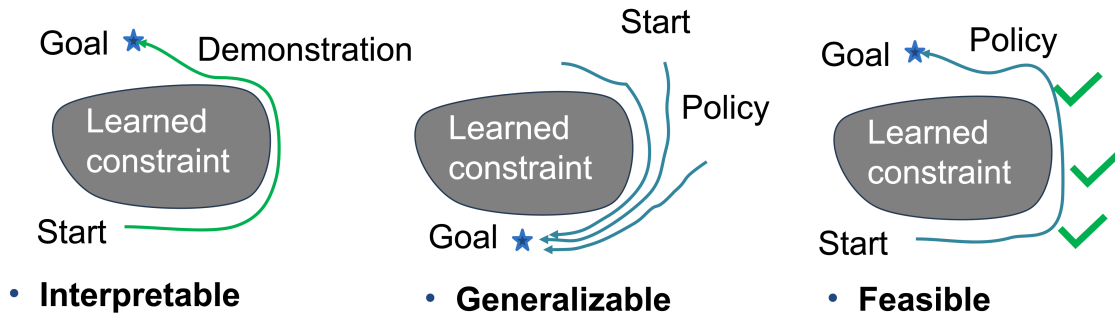


Figure 1: Three properties that motivate learning an *explicit* constraint from a demonstration, after Peng & Billard [1]. In each panel the grey region is the learned unsafe set ζ_θ and the star marks the goal. *Interpretable*: the constraint is a region a human can inspect, recovered from one demonstration. *Generalizable*: the same region steers the policy π_ϕ to the goal from unseen start states. *Feasible*: every policy rollout stays outside it (green checks). A hand-coded controller provides none of the three, whereas this thesis recovers the region as the PUCL network ζ_θ and enforces it as a hard MPC inequality.

The KUKA iiwa-14 (7 degrees of freedom, DoF) is the manipulation platform at LASA and the physical deployment target for which Peng & Billard’s PUCL framework was designed. This thesis works toward deploying the PUCL-learned constraint on it in PyBullet, replacing PUCL’s policy block with a model-predictive-control (MPC) trajectory optimiser that enforces ζ_θ as a hard nonlinear-program (NLP) row. The constraint is learned on the point-mass abstraction. The MPC plans the end-effector path, and the arm tracks it by inverse kinematics.

MPC is chosen over a reactive or learned policy block because it handles the learned constraint directly in the optimal control problem (OCP): the constraint enters as a hard inequality at every prediction step, with no reward shaping and no Lagrange-multiplier tuning that must re-adapt as ζ_θ changes. Position, velocity, acceleration, and jerk constraints are expressed uniformly through the state definition, and the solve is deterministic and reproducible.

Every headline result in this thesis is open-loop trajectory optimisation: a single horizon solve replayed without replanning. Receding-horizon closed-loop control is shown only as a sanity check, with the analytic (oracle) constraint: under an injected disturbance the closed loop recovers the goal and, with robust constraint tightening, removes the residual violations, whereas the open-loop plan drifts (Section 6.7). Closing the loop with the learned constraint ζ_θ under live physics is the main remaining deployment step (Section 7).

1.2 The PUCL-MPC gap

PUCL [1] recovers a constraint network ζ_θ from expert demonstrations (Section 2.2). Its downstream *policy block* is the component this thesis targets. The original blocks operate in a narrower regime than the hard-constraint MPC used here (Section 2.2): constrained reinforcement learning (RL) [2] enforces the constraint softly through a Lagrange multiplier that re-adapts as ζ_θ changes. MPC instead expresses position-, velocity-, acceleration-, and jerk-level constraints as hard inequality rows at each prediction step, and re-optimises from scratch when ζ_θ changes. The evaluated controller enforces the position and

velocity rows. The acceleration and jerk rows are a capability of the formulation, learned but not enforced in any reported solve.

The missing piece is a *differentiable* bridge between the PyTorch-trained ζ_θ and the symbolic NLP solver. The gradient-based solver needs the constraint’s analytic derivatives $\partial\zeta_\theta/\partial p$ (its Jacobian) at every iteration, and a black-box PyTorch forward pass does not expose them. Contribution C1 closes this gap with a symbolic constraint bridge.

1.3 Contributions

This thesis delivers a constraint-aware MPC pipeline and one scientific finding. The central contribution is an MPC-NLP policy block that replaces PUCI’s original constrained-RL block: it ingests the learned ζ_θ as a hard NLP constraint, not a reward penalty or a sampling filter. Deploying PUCI this way yields the report’s main finding. The MPC genuinely serves as PUCI’s forward policy, but being deterministic and non-exploring it cannot supply the broad rollouts that reliable-negative mining needs, so the label-free position keep-out is recovered only *conservatively*: an identifiability limit (Section 6.9). A minimal binary collision check at probed states then recovers a tight, deployable keep-out. This active signal extends rather than replaces PUCI’s label-free core (Section 6.10).

Four engineering items implement the MPC-NLP block:

- C1 Symbolic constraint bridge.** A constraint-injection bridge extracts the PyTorch weights from the ζ_θ checkpoint and rebuilds the forward pass as a CasADi function [3], so the network becomes one differentiable hard inequality row in the NLP with no manual Jacobian derivation. The bridge supports single-net (position-only or velocity-only) and dual-net (joint position-and-velocity) configurations. We specialise the ME-425 [4] finite-horizon NLP to the joint-space dynamics of the KUKA iiwa-14 (Pinocchio model, $M(q)\ddot{q} + h(q, \dot{q}) = \tau$) and add the learned constraint $\zeta_\theta(p_{EE})$, where p_{EE} is the end-effector (EE) position, with an ℓ_1 -exact slack for infeasibility recovery.
- C2 acados SQP-RTI solver.** The acados sequential-quadratic-programming real-time-iteration (SQP-RTI) backend generates dedicated C code and takes a single real-time iteration per call, so its per-call cost is constant in the number of NLP rows rather than growing with interior-point iterations. On the position NLP, warm-started SQP-RTI is measured at a median 3.1 ms (2.8–4.5 ms over 50 calls, status 0) against IPOPT at 177 ms on the same machine: 57× faster, inside the 100 ms control step. Scope: the deployed controller still uses IPOPT for every reported solve, the higher-order (velocity/acceleration/jerk) QP does not yet converge, and the measurement covers the position case only. The two backends share the same OCP, selected through a backend switch.
- C3 Triple-integrator unification of four orders.** Taking jerk as the control input on a triple-integrator state (p, v, a) makes position, velocity, acceleration, and jerk one state-and-derivative chain, so all four learned constraint orders become *identical* inequality rows in one NLP, with no change to the solver or the network architecture (Section 4). This is a capability of the model. The deployed controller is single-integrator (velocity-level, control = v , $p^+ = p + v dt$) and reads only the position and velocity states. The acceleration and jerk networks are learned (recall in Section 6) but their rows are not enforced in any reported solve, so the executed policy can exceed the learned acceleration and jerk caps. Deploying the full triple-integrator with acceleration and jerk enforced is the queued next step.
- C4 PyBullet KUKA deployment.** The execution stack on the iiwa-14 comprises the PyBullet scene, the iiwa14 model in unified robot description format (URDF), differential inverse kinematics (IK), trajectory recording, violation logging, and a live monitor, and realises a planned collision-free trajectory on the arm (Section 6.6, Fig. 6). The method is evaluated on four 3D point-mass environments whose obstacles differ only in the position-boundary geometry (one ellipsoid, two ellipsoids, a concave arc, two stacked regions). All four orders are learned on each environment, but only the position and velocity rows are enforced in the evaluated single-integrator controller. The concave scene doubles as a stress test that exposes the whole-body limitation of the Cartesian planner (Section 6.11).

Experiment setup and results are not listed as contributions. The offline run of the same OCP against the analytic constraint ζ_{true} generates the expert demonstrations $\mathcal{D}^+$. This is a supporting tool, deferred to the appendix.

1.4 Thesis outline

Section 2 reviews PUCL and MPC fundamentals. Sections 3–4 state the problem and develop the method. Section 5 describes the evaluation protocol and states the formal hypotheses. Section 6 reports results. Section 7 concludes.

2 Background and Related Work

This section sets out the two pillars PUCL-MPC combines: the MPC planning backbone (Section 2.1) and the PUCL constraint-learning framework (Section 2.2).

2.1 Model Predictive Control

Model Predictive Control (MPC) [5] is an optimisation-based controller: at every step it solves a finite-horizon optimal control problem (OCP) over a model of the system, applies only the first input, and re-solves at the next step from the measured state. This receding-horizon re-solve makes the controller closed-loop and able to reject disturbances. The four-geometry results in this thesis use an open-loop configuration of this scheme: a single horizon solve replayed without replanning. Receding-horizon closed-loop control is shown only as a sanity check, and only with the analytic (oracle) constraint (Section 6.7). The OCP minimises a goal-tracking cost subject to the dynamics and a constraint set. The constraint set is where this thesis contributes: we augment it with the learned safe-set row $\zeta_{\theta}(p_{\text{EE}})$, where p_{EE} is the end-effector position. The cost, horizon, dynamics, and ℓ_1 -exact slack are given in Section 4.2. The terminal ingredients and stability assumptions, on which this thesis makes no formal claim, are stated in Appendix A.7.

The OCP is built symbolically in CasADi [3] and solved by two backends. IPOPT [6] is the interior-point reference that produces every solve time reported here (75–197 ms), and is the backend used for every reported solve. acados [7] (SQP real-time iteration [8] with the HPIPM quadratic-program solver [9]) is the real-time backend. On the position OCP a warm-started SQP-RTI solve runs at a median of 3.1 ms (2.8–4.5 ms over 50 calls, status 0) against 177 ms for IPOPT on the same machine, a $57\times$ speed-up that fits inside the 100 ms control step. This measurement covers the position case only. The higher-order (velocity, acceleration, jerk) QP does not yet converge, so single-digit-millisecond timing there remains a target.

Why MPC over other policy blocks. Three properties motivate MPC as the policy block for constraint learning. (i) Constraints are handled directly inside the OCP, with no reward shaping, multiplier tuning, or sampling rejection. (ii) Constraints across control orders (position, velocity, acceleration, jerk) are handled by changing the state-space definition, not the solver. (iii) At a fixed seed the solution is deterministic, which makes the benchmarks reproducible.

2.2 Positive-Unlabeled Constraint Learning (PUCL)

Peng & Billard [1] frame constraint recovery from demonstrations as a Positive-Unlabeled (PU) classification problem. Expert demonstrations are positive (the expert respects the constraint). Policy rollouts are unlabeled (a mix of safe and unsafe states). The algorithm alternates two steps. First, a rollout state $x \in \mathcal{U}$ is mined as a reliable infeasible (equivalently a reliable negative, RN) $r \in \mathcal{R}$ when its mean k -nearest-neighbour (kNN) distance to the positive set exceeds a threshold:

$$r \in \mathcal{R} \iff \frac{1}{k} \sum_{j=1}^k \|x - x_j^+\|_2 \geq \kappa_{\text{kNN}}, \quad (2.2.1)$$

so states close to the demonstrations are treated as safe and distant outliers as unsafe. Second, the constraint network $\zeta_{\theta} : \mathbb{R}^n \rightarrow [0, 1]$ is trained by binary cross-entropy to separate the positive set $\mathcal{P}$ ($\rightarrow 0$,

safe) from the reliable-negative set $\mathcal{R}$ ($\rightarrow 1$, infeasible). The sub-level set $\hat{\mathcal{C}}_\theta = \{\zeta_\theta(x) < 0.5\}$ approximates the safe set, and the MPC enforces $\zeta_\theta(p_k)$ below a conservative injection bound (Section 4.5) to keep planned states inside it.

The reliable-negative mining of Eq. (2.2.1) distinguishes PUCL from a naive binary cross-entropy classifier (BCE), the baseline used here to measure the value of the PU step (Section 6): BCE trains the same networks by binary cross-entropy on positives versus all unlabeled rollout states, with no kNN selection. BCE is a *constraint-learning* ablation (it learns ζ_θ differently), distinct from behaviour cloning (BC, Section 5.2), which is a constraint-free *policy* baseline. The loop is iterative: each retrained policy rolls out fresh trajectories that enrich $\mathcal{U}$, repeating until ζ_θ stabilises (Algorithm 1).

The original work pairs PUCL with policy blocks that operate in narrower regimes than the hard-constraint MPC used here. Constrained reinforcement learning (PPO-Lagrangian) [2] generalises to arbitrary control orders but enforces the constraint softly through a Lagrange multiplier and requires a full on-policy retraining pass at every PUCL iteration. With ζ_θ shifting each iteration the multiplier chases a non-stationary target. We do not run it here. MPC instead enters constraints as hard NLP rows rather than reward terms, and keeps no state between iterations, so it re-optimises from scratch each time ζ_θ is updated (Section 4). The pipeline is summarised in Fig. 2. The evaluation reports per-order recall (the safety-critical, learned-versus-true metric) alongside the precision and IoU of Peng & Billard [1] (Section 6).

Algorithm 1 Positive-Unlabeled Constraint Learning (PUCL) [1]. The policy step (line 2) is the block this thesis replaces with MPC.

Require: expert demonstrations $\mathcal{D}^+$, a randomly initialised constraint network ζ_θ and policy block, and an empty memory $\mathcal{M} \leftarrow \emptyset$

1: **repeat**

2: Update the policy block to minimise task cost subject to the current $\zeta_\theta \triangleright$ orig. constrained RL,
 this work: MPC

3: Roll out the policy to collect the unlabeled set $\mathcal{U}$

4: Mine reliable negatives $\mathcal{R}$: flag $x \in \mathcal{U}$ whose mean kNN distance to $\mathcal{D}^+$ exceeds κ_{kNN} (Eq. 2.2.1)

5: $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{R}$ $\triangleright$ accumulate reliable negatives across iterations

6: Train ζ_θ by binary cross-entropy: $\mathcal{D}^+ \rightarrow 0$, $\mathcal{R} \cup \mathcal{M} \rightarrow 1$

7: **until** ζ_θ stabilises (maximum iterations)

Ensure: constraint ζ_θ and safe set $\{\zeta_\theta < 0.5\}$

3 Problem Formulation

Section 1 stated the gap informally: PUCL learns a constraint network ζ_θ from demonstrations, but its original policy never enforces that network as a hard per-step constraint. We now state it formally, defining the state space, the constraint representation, the four constraint orders the system handles at once (Table 1), and the assumptions that make PUCL learning tractable. These definitions drive Section 4 and the evaluation of Sections 5 and 6.

3.1 Formal statement

Let $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ be the state space of a robot manipulator and $\mathcal{A} \subseteq \mathbb{R}^{n_a}$ the action space. A trajectory $\xi = (s_0, a_0, s_1, a_1, \dots, s_T)$ is *constraint-respecting* if every (s_t, a_t) belongs to an unknown safe set $\mathcal{C} \subseteq \mathcal{S}$.

Given. A set of n_d expert demonstrations $\mathcal{D} = \{\xi_i\}_{i=1}^{n_d}$ drawn from a constraint-respecting policy π^* . The demonstrations are labelled *positive*. The constraint set $\mathcal{C}$ is never observed directly.

Goal. Learn a constraint approximation $\hat{\mathcal{C}}_\theta$ from $\mathcal{D}$ and synthesise an MPC policy that:

1. completes a reaching task from any start s_0 to a goal g in the workspace,
2. respects $\hat{\mathcal{C}}_\theta$ throughout the motion.

The constraint is a neural network $\zeta_\theta : \mathbb{R}^{n_s} \rightarrow [0, 1]$, and the safe set approximation is its sub-level set $\hat{\mathcal{C}}_\theta = \{s : \zeta_\theta(\phi(s)) < 0.5\}$, where ϕ extracts the feature slice for each of the four differential orders, namely position, velocity, acceleration, and jerk (Section 3.3).

3.2 Constraint orders considered

A central contribution is learning multi-order constraints: limits on state signals of different differential orders, evaluated under different obstacle geometries. One constraint network is learned per order. The deployed single-integrator MPC enforces the position and velocity rows as hard constraints. The acceleration and jerk rows are learned but not enforced in any reported solve. Table 1 lists the four orders and their state slices.

Table 1: Constraint orders, state signals, and constraint types. A constraint network is learned for all four orders under all four obstacle geometries. The deployed MPC is single-integrator (velocity-level): it enforces only the position and velocity rows as hard constraints. The acceleration and jerk networks are learned but never enforced in a reported solve. The triple-integrator jerk-as-control formulation (jerk on (p, v, a)) is a model capability, not the configuration evaluated here.

Order	Signal	State slice	Constraint type	Geometries
0	Position p	$x[0 : d]$	Union-of-ellipsoid keep-out	all
1	Velocity $\dot{p}$	$x[d : 2d]$	$\ v\ \leq v_{\max}$	all
2	Acceleration $\ddot{p}$	$x[2d : 3d]$	$\ a\ \leq a_{\max}$	all
3	Jerk $\dddot{p}$	u	$\ j\ \leq j_{\max}$	all

3.3 Assumptions

1. **Full-state observability.** The joint configuration q and velocity $\dot{q}$ are available at every step, and the end-effector position follows from forward kinematics.
2. **Stationarity.** The constraint set $\mathcal{C}$ does not change within an episode: obstacle geometry is fixed and joint limits are constant.
3. **Expert feasibility.** Every demonstrated trajectory lies strictly inside $\mathcal{C}$, drawn from the interior rather than the boundary. This coverage assumption is specific to PUCL [1]: it keeps the positive set clear of the unsafe region, so a point far from any demonstration can be labelled unsafe by kNN distance.
4. **Independent marginals.** Position safety does not depend on velocity, and the velocity, acceleration, and jerk caps are uniform across the workspace. We therefore train four independent networks $(\zeta_\theta^p, \zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j)$ and combine them at inference as $\zeta_\theta = \max(\zeta_\theta^p, \zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j)$.

4 Method

The problem (Section 3) needs a policy block that enforces a learned constraint ζ_θ at each step across four differential orders: position, velocity, acceleration, and jerk. The central design choice is the state representation (Section 4.1): carrying position, velocity, and acceleration as state and jerk as the control turns every constraint order into a single algebraic row in the same NLP. Section 4.2 gives the optimal control problem (OCP). Section 4.3 describes the iterative PUCL loop and the four-network decomposition that keeps kNN mining tractable. Section 4.5 explains how a PyTorch-trained ζ_θ is re-expressed as a CasADi symbolic function so the solver can differentiate through it. Figure 2 gives the overall structure.

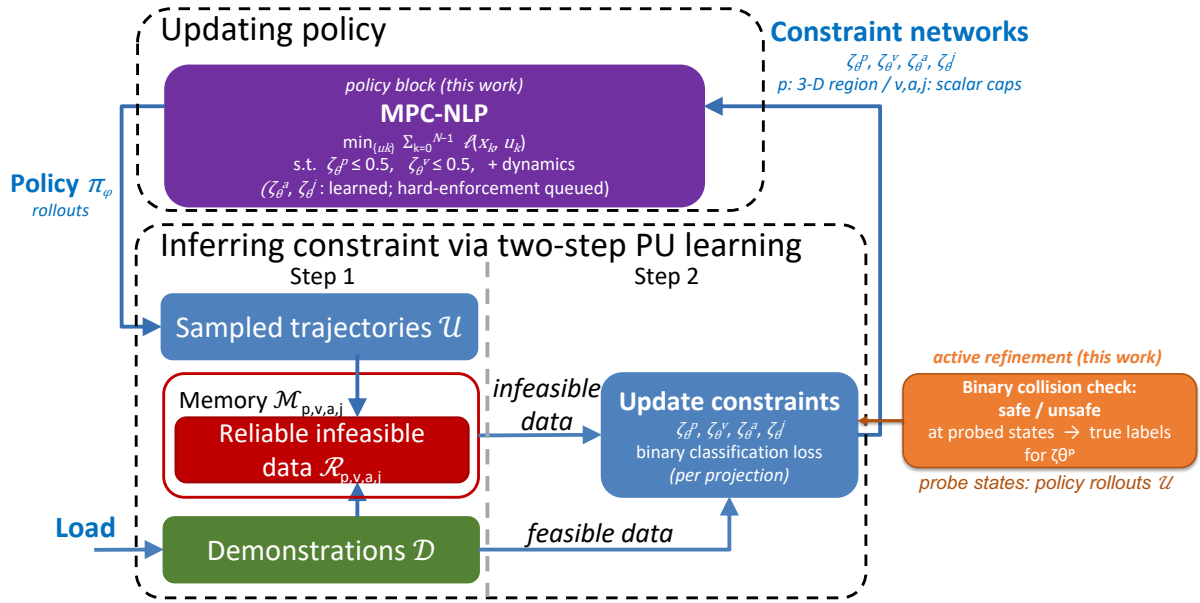


Figure 2: Multi-order extension of the PUCL pipeline for MPC deployment. Top: the MPC-NLP policy block (this work) consumes the learned networks ζ_θ^p (end-effector position, a 3-D keep-out region) and $\zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j$ (velocity, acceleration, and jerk magnitudes, scalar caps). In the reported single-integrator open-loop configuration only the position and velocity networks are enforced as hard NLP rows. The acceleration and jerk networks are learned but not enforced. Bottom: the two-step PU-learning loop of [1] runs in parallel on the position projection and on the $\|v\|, \|a\|, \|j\|$ magnitude projections of the same demonstration and rollout sets, with memory buffers and reliable-infeasible sets kept separate per projection. Right (this work): the active-refinement extension supplies a binary collision check at probed states that returns only safe or unsafe, never the analytic value or its gradient, giving true labels for the position network (Section 6.10). The label-free PUCL/MPC loop recovers the unsafe set at recall ~ 1.0 , and the active probe is what makes it tight.

4.1 State representation and dynamics

The end-effector position in trajectory frame is the planning state. Joint-space dynamics are abstracted by an exact triple integrator. Define the state in $d = 3$ spatial dimensions as $x = (p, v, a) \in \mathbb{R}^{3d}$, with jerk $j \in \mathbb{R}^d$ the control input. The exact constant-jerk discretisation over timestep Δt is:

$$\begin{aligned} p^+ &= p + v\Delta t + \frac{1}{2}a\Delta t^2 + \frac{1}{6}j\Delta t^3, \\ v^+ &= v + a\Delta t + \frac{1}{2}j\Delta t^2, \\ a^+ &= a + j\Delta t. \end{aligned} \tag{4.1.1}$$

With $\Delta t = 0.1$ s and a horizon of $N = 50$ steps, the planner looks 5.0 s ahead. The learning environment exposes a 15-dimensional observation, $[p(3), g(3), v(3), a(3), j(3)]$, where g is the goal. The four constraint networks of Section 4.3 each read a disjoint slice. Because position, velocity, and acceleration are all *state* components and jerk is the *input*, the same NLP imposes a constraint of any differential order by adding one inequality row.

Scope of what is deployed. Equation 4.1.1 is what the model *can* represent, not the configuration evaluated here. The deployed controller is single-integrator (velocity-level): the control is the velocity, the state advances as $p^+ = p + v\Delta t$, and the policy reads only the position and velocity slices, ignoring acceleration and jerk. The triple-integrator jerk-as-control form is exercised only as a formulation and produced no reported result. Every headline number is single-integrator. The acceleration and jerk inequality rows are therefore learned but not enforced in any reported solve.

Geometry family. All experiments run on one pure-Python kinodynamic point-mass environment (no physics engine), instantiated as four benchmark geometries that differ only in their keep-out set, a union of axis-aligned ellipsoids: **single ellipsoid** (one convex ellipsoid), **two ellipsoids** (two staggered ellipsoids, an S-shaped detour), **concave** (four spheres in a concave arc forming a non-convex

pocket), and **two regions** (two stacked regions pinching a z -corridor). A state is position-unsafe when it lies inside any keep-out ellipsoid. The velocity, acceleration, and jerk caps ($v_{\max}, a_{\max}, j_{\max}$) are shared across geometries and sized just above the smooth-demonstration envelope, so demonstrations stay feasible while the loose-bound exploratory policy violates them and supplies the negative samples PU learning needs. These are environment-design choices on synthetic finite-difference kinematics, not physical ground truth.

4.2 MPC formulation

The OCP solved at each planning call is:

$$\min_{\{j_k\}} w_g \|p_N - g\|^2 + \sum_{k=0}^{N-1} \left(w_t \|p_k - g\|^2 + w_j \|j_k\|^2 + w_a \|a_k\|^2 \right) \quad (4.2.1)$$

s.t. Eq. 4.1.1 (triple-integrator dynamics),

$$\begin{aligned} p_0 &= p_{\text{init}}, \quad v_0 = 0, \quad a_0 = 0, \\ \sum_{i=1}^d \left(\frac{p_{k,i} - c_i^{(e)}}{r_i^{(e)}} \right)^2 &\geq 1 + m, \quad \forall e, k = 0, \dots, N, \\ \|v_k\|_2 &\leq v_{\max}, \quad \|a_k\|_2 \leq a_{\max}, \quad \|j_k\|_2 \leq j_{\max}, \\ \zeta_{\theta}^p(p_k) &\leq \tau_{\text{inj}}, \quad \zeta_{\theta}^v(v_k) \leq \tau_{\text{inj}}, \quad \zeta_{\theta}^a(a_k) \leq \tau_{\text{inj}}, \quad \zeta_{\theta}^j(j_k) \leq \tau_{\text{inj}}. \end{aligned}$$

The three constraint groups are the union-of-ellipsoid keep-out, the analytic norm caps on velocity, acceleration and jerk, and the learned rows ζ_{θ} . Here e indexes the keep-out ellipsoids with centre $c^{(e)}$ and semi-axes $r^{(e)}$, and each learned row is enforced at a conservative injection bound τ_{inj} (0.15 for position and 0.2 for velocity in the deployed runs), set well inside the $\{\zeta_{\theta} < 0.5\}$ safe set (Section 4.5). The weights are $w_g = 70$, $w_t = 0.15$, $w_j = 1 \times 10^{-3}$, $w_a = 1.8 \times 10^{-3}$ (fixed by the joint optimisation in Section 5.5), and the ellipsoid margin is $m = 0.05$ with numerical slack 10^{-3} . The analytic caps for the main four-geometry learning are $v_{\max} = 0.60$, $a_{\max} = 2.1$, and $j_{\max} = 42$. The learned networks $\zeta_{\theta}^{p,v,a,j}$ replace these analytic rows once trained (Section 4.3). These are distinct from the legacy solver-study caps $a_{\max} = 1.2$, $j_{\max} = 6.0$ used only in the constraint-order ablation and the solver-timing table (Tables 9, 17). The two cap regimes must not be conflated.

PUCL learns all four orders (Section 6.1), and the triple-integrator formulation makes the four learned rows algebraically identical in the NLP. The reported single-integrator runs enforce only the position and velocity rows. Promoting acceleration and jerk to hard inequalities, once the triple-integrator state is carried, is the queued next step. The terminal goal term is soft rather than a hard terminal constraint: imposing $p_N = g$ over-constrains the multi-order scenes where velocity and acceleration caps limit how fast the trajectory can decelerate to rest at the target.

Two solver backends. The experiments use the OCP (Eq. 4.2.1) in open-loop mode: a single horizon solve at $t = 0$ whose input sequence is cached and replayed without replanning. A receding-horizon controller would instead re-solve the OCP at every control step from the latest measured state and apply only the first input. With $\Delta t = 0.1$ s that is one solve per 100 ms, so the solver’s wall time decides whether the planner can run live on the robot. The OCP is built once in CasADi [3], which also supplies the symbolic Jacobians and Hessians, and handed to two interchangeable backends.

IPOPT. IPOPT [6], through CasADi’s interface, converges to the exact optimum on every call, but at 132–197 ms per solve (8 seeds, Table 8) it overruns the 100 ms control step and cannot drive the live loop. We keep it as the exact correctness reference. Every reported solve time is IPOPT’s.

acados. acados [7] generates dedicated C code for the fixed problem and takes a single real-time iteration [8]: one SQP/Newton step on the HPIPM quadratic-program solver [9] rather than iterating to convergence. On the position NLP a warm-started SQP-RTI step has median wall time 3.1 ms (2.8–4.5 ms over 50 calls, status 0) against IPOPT’s 177 ms on the same machine, a $57\times$ speed-up that fits inside the 100 ms control step (Table 17). This position-case figure is measured against a committed benchmark log. Scope: the speed-up is established only on the position NLP. Once the velocity ball or any higher order is added the QP does not yet converge under the current bound conditioning, so the higher-order

real-time path stays a target, and the deployed controller still uses IPOPT for every reported solve. Both backends share the one CasADi NLP. Neither appeared in the original PUCL work [1], so the MPC-NLP with both backends is a contribution of this thesis.

Warm-start. Each planning call initialises the N -step jerk sequence from a straight-line interpolation between the current position p_0 and the goal g , bent by a $\sin(\pi\lambda)$ bump (amplitude 0.45, perpendicular to the line) when an obstacle is present. A small Gaussian perturbation (0.01σ , seeded per call) breaks ties near symmetric obstacles. No solution from a previous call is reused: every open-loop solve is cold-started. The receding-horizon re-solve runs in one disturbance test (Section 6.7) with the analytic oracle and the learned ζ_θ alike.

4.3 PUCL constraint learning

Following Peng & Billard [1], each constraint network is trained over 8 outer iterations across 5 random seeds. Recall is recovered at the first iteration (one-shot). The remaining iterations refine precision and IoU, not recall (Section 6). For the velocity, acceleration, and jerk networks this high recall is read on a one-sided scalar-norm grid that is roughly 90% unsafe, so it is class-imbalance inflated. On a boundary-balanced grid the higher-order recovery is markedly weaker, with acceleration the weakest of the four orders (Section 6). At iteration t :

1. **Roll out.** The MPC policy with the current network $\zeta_\theta^{(t)}$ generates n_r trajectories, added to the unlabelled pool $\mathcal{U}$. In the configuration evaluated here this step is degenerate: the MPC is deterministic and rolled out once per demonstration, so it does not explore, and the label-free boundary stays conservative (Section 6.9).
2. **Mine reliable infeasible states.** For each $q \in \mathcal{U}$, compute the average k -nearest-neighbour (kNN) distance to the positive set $\mathcal{P}$ in feature space. If $\frac{1}{k} \sum_{j=1}^k \|q - q_j^+\|_2 \geq \kappa$, label q as *reliable infeasible* and add it to $\mathcal{R}^{(t)}$.
3. **Retrain ζ_θ .** Minimise the binary cross-entropy loss on the clean labelled set ($\mathcal{P} \rightarrow 0$, $\mathcal{R}^{(t)} \rightarrow 1$). The result is $\zeta_\theta^{(t+1)}$.

Four independent networks. The four constraint orders are learned by separate networks with disjoint input features, each running its own kNN reliable-negative step in its own feature space:

- $\zeta_\theta^p : \mathbb{R}^3 \rightarrow [0, 1]$ reads the position slice and is a 32-32 multi-layer perceptron (MLP), wide enough to represent a non-convex union of ellipsoids.
- $\zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j : \mathbb{R}^1 \rightarrow [0, 1]$ each read one higher-order slice, collapse it to the scalar norm $\|\cdot\|_2$, and pass it through a tiny 1-4-1 MLP. Because each true cap is univariate in the magnitude ($\|v\|_2 > v_{\max}$, and likewise for a, j), feeding the scalar norm builds in rotational invariance by construction rather than forcing the network to learn it from finite samples.

At the MPC stage the four outputs are combined with OR semantics, $\zeta_\theta = \max(\zeta_\theta^p, \zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j)$: a state is flagged unsafe if any order is unsafe. The deployed single-integrator policy enforces the p and v terms as hard rows. The a and j terms are learned but not enforced (Section 4.4). Factoring the constraint into one 3-D position network and three 1-D magnitude networks keeps every kNN mining step in a low-dimensional feature space, where the distance estimates PUCL relies on stay meaningful. A joint 12-D classifier would mine in a high-dimensional space, where those distances degrade [1].

4.4 The MPC policy block

The main use of the MPC, and the main contribution of the thesis, is as the PUCL policy block: a trajectory optimiser that replaces the original constrained-RL policy and enforces the learned constraint ζ_θ as hard NLP rows. In the deployed single-integrator policy these are the position and velocity rows. The acceleration and jerk rows share the same formulation but stay learned, not enforced (Section 4.2). During the PUCL loop the MPC consumes the current learned ζ_θ and produces rollouts ξ that join the unlabelled set $\mathcal{D}^u$. Early in training ζ_θ is far from ζ_{true} , so the rollouts deliberately enter the unsafe

region. Those violations are what the two-step PU mechanism mines as the reliable infeasible set $\mathcal{R}$ (Section 4.3). Being deterministic and rolled out once per demonstration, the MPC does not explore, so it cannot supply the broad rollouts that reliable-negative mining needs, and the label-free boundary stays conservative (Section 6.9). The intended real-time solver in this role is acados SQP-RTI [7, 8]. The reported experiments are open-loop on the converged-IPOPT path (Section 6.1). The policy block is evaluated through the per-order recall (Table 6), the constraint-order ablation (Table 9), and the cross-environment solver-time comparison (Table 17).

Supporting use: offline demonstration synthesis. The same OCP is reused, offline, as a setup step that is not a contribution. Solving the MPC over a swept grid of start-goal pairs against the known analytic constraint ζ_{true} synthesises the positive set $\mathcal{D}^+ = \{\xi_i^+\}_{i=1}^{n_d}$ that seeds PUCL. Because ζ_{true} is known by construction in simulation, these trajectories are feasible by definition. The solver here is the converged-IPOPT path [6], since latency is irrelevant offline. The demonstration-synthesis tool and the two quality criteria imposed on $\mathcal{D}^+$, trajectory diversity (mean pairwise Hausdorff distance) and smoothness (set by the jerk weight w_j), are described in Appendix A.5.

Why the two contexts must differ. Separating the two uses is what makes the two-step PU mechanism work. The positive set $\mathcal{D}^+$ is generated against ζ_{true} and is diverse and smooth, while the unlabelled rollouts are generated against the still-imperfect ζ_θ and contain controlled violations by design. Using the same MPC against the same constraint in both roles would collapse the learning signal.

4.5 Constraint injection into the NLP

Each trained network is a PyTorch MLP: the 32-32 position network ζ_θ^p and the three 1-4-1 magnitude networks $\zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j$. The constraint-injection bridge re-expresses each network as a CasADi symbolic function: it extracts the PyTorch weights and rebuilds the forward pass symbolically, so the NLP solver can query its value and exact gradient at each SQP iteration. Writing ζ_θ^o for the network of order $o \in \{p, v, a, j\}$ and s_k^o for the slice it reads at stage k (position p_k , or the magnitude $\|v_k\|, \|a_k\|, \|j_k\|$), each appears in the OCP both as a soft squared-hinge penalty $w_\zeta \max(0, \zeta_\theta^o(s_k^o) - \tau_{\text{inj}}^o)^2$ (weight w_ζ , Table 13) added to the stage cost, and as a hard row $\zeta_\theta^o(s_k^o) \leq \tau_{\text{inj}}^o$. The soft penalty steers the interior-point method away from the boundary. The hard row blocks any iterate that crosses the injection bound. The deployed injection bounds are $\tau_{\text{inj}}^p = 0.15$ (position) and $\tau_{\text{inj}}^v = 0.2$ (velocity), set well below the $\tau = 0.5$ decision threshold so the enforced wall keeps a conservative spatial margin inside the learned safe set $\{\zeta_\theta < 0.5\}$: each network saturates near the boundary, so a smaller score bound maps to a larger absolute distance from the true keep-out (the 0.15 position bound buys a $\approx 10\%$ absolute margin). The decision threshold $\tau = 0.5$ scores constraint quality (recall, precision, IoU) and is separate from these deployed injection bounds. The solver-environment thresholds of Table 13 are tuned independently.

What “hard” guarantees, and what it does not. The solver rejects any feasible iterate whose learned position or velocity score crosses the enforcement level, so on a feasible solve the executed plan satisfies the learned keep-out exactly. This guards the learned set, not the true obstacle, and is partly tautological: a feasible solve has zero violations of the enforced rows by construction. Safety against the true constraint follows only because the recovered boundary over-covers it (recall ~ 1.0), so it is conservative rather than certified. The executed open-loop policy still violates the true unsafe set on 25–47% of steps, because the hard wall sits on a deliberately over-grown boundary (Section 6.9). Why we keep the controller hard rather than soften it to chase boundary data is argued in Section 6.10.

Generality across obstacle geometries. The method is validated on the four union-of-ellipsoid geometries of Section 4.1, all sharing one recipe and all activating the full position, velocity, acceleration, and jerk stack. Because constraints enter the NLP as identical algebraic rows regardless of differential order, no architectural change is needed between geometries: only the keep-out set and the inequality bounds change. The full geometry taxonomy is in Table 2 (Section 5).

Multi-order constraint networks. Rather than impose the higher-order limits as hard analytic caps $\|v_k\| \leq v_{\text{max}}, \|a_k\| \leq a_{\text{max}}, \|j_k\| \leq j_{\text{max}}$, the PUCL pipeline of Peng & Billard [1] runs in parallel along four projections of the same demonstration set: end-effector position (x, y, z) for ζ_θ^p and the magnitudes $\|v\|, \|a\|, \|j\|$ for $\zeta_\theta^v, \zeta_\theta^a, \zeta_\theta^j$. All four are recovered with high recall (Section 6.1), but this is class-imbalance inflated for the higher orders, where the boundary-balanced recovery is markedly weaker

and acceleration is the weakest of the four orders. The contribution of this thesis is this multi-order learning. The deployed single-integrator MPC hard-enforces only the position and velocity rows, so promoting acceleration and jerk to hard inequalities, which the triple-integrator formulation already admits, is the queued next step. Figure 2 expands the system, reproducing the colour code and box geometry of [1] with the MPC-NLP policy block of the present work and parallel PU-learning branches for the magnitudes. The branches share the demonstration set $\mathcal{D}$ and the rollout stream but maintain independent memory buffers $\mathcal{M}_{p,v,a,j}$ and reliable-infeasible sets $\mathcal{R}_{p,v,a,j}$, so each projection’s network does not contaminate the others.

5 Experiments

The method (Section 4) makes five falsifiable claims. We state them as hypotheses, fix the benchmark environments, baselines, and metrics, and test each in Section 6.

5.1 Benchmark environments

The constraint-learning experiments run on a pure-Python point-mass abstraction (Section 4.1, no physics engine). Deployment runs in PyBullet [10] with the KUKA iiwa-14 [11], the end-effector tracking planned waypoints by Jacobian-pseudoinverse IK. The planner uses timestep $\Delta t = 0.1$ s and horizon $N = 50$ (5.0 s look-ahead). Expert demonstrations come from the velocity-mode oracle MPC with analytic constraints, under a rate limit that keeps $\|a\| \leq 2.0$ over random start-goal pairs. The 29–30 feasible smooth trajectories per geometry form the positive set $\mathcal{P}$.

We use four obstacle geometries (Table 2). Each is a 3-D kinodynamic point-mass environment with a 15-D observation $[p, g, v, a, j]$ and an unsafe position set built from a union of ellipsoidal keep-outs. All four come from one recipe, so they share caps and dynamics and isolate obstacle geometry as the only varying factor, and all activate the full constraint-order stack (position, velocity, acceleration, jerk). They span a convex-to-concave axis. *Single ellipsoid* is one convex keep-out (the convex baseline). *Two ellipsoids* stagger two keep-outs into an S-detour. *Concave* arranges four spheres into a concave pocket. *Two regions* stacks two large keep-outs that pinch a z -corridor. The constraint-order ablation (Section 6.5) instead stacks the orders one at a time on a single geometry.

Table 2: Benchmark obstacle geometries. **Geometry:** scene name. **Composition:** the union of ellipsoidal keep-outs forming the unsafe position set. All four are 3-D ($d=3$) kinodynamic point-mass scenes from one recipe, activating the full pos+vel+acc+jerk stack and sharing horizon $H=50$ and caps $v_{\max}=0.60$, $a_{\max}=2.1$, $j_{\max}=42$, differing only in position-boundary geometry. The caps are design choices on synthetic finite-difference kinematics, not physical ground truth. The scenes generalise the single-ellipsoid kinodynamic environment of Peng & Billard [1] to unions of ellipsoids.

Geometry	Composition
single ellipsoid	1 convex keep-out
two ellipsoids	2 staggered keep-outs (S-detour)
concave	4 spheres in a concave arc
two regions	2 large stacked regions pinching a z -corridor

5.2 Baselines

Four baselines bracket the operating space.

1. **Oracle MPC.** MPC with the analytic ground-truth constraints: the hard upper bound (perfect constraint knowledge, same planner).
2. **Unconstrained MPC.** MPC with all constraint rows removed (task cost only): the ablation lower bound for the significance test, isolating the effect of the constraint rows within the same planner.

3. **Behaviour cloning (BC).** A supervised policy network learned *directly from the demonstrations* (an MLP cloning the expert velocity field by regression), then executed and checked against the *true* constraints, with no constraint learning of any kind. BC is a *test-time* baseline only: it does *not* generate the candidate trajectories PUCL mines (the MPC forward policy supplies those, Section 4) and does *not* replace PUCL’s reliable-negative selection. It isolates whether the explicit constraint buys anything a constraint-free imitator trained on the same demonstrations could not. It is the same velocity-level policy class as the dynamical-systems block in Peng & Billard [1], minus any constraint, and is evaluated on held-out starts *and* goals (Section 6.12), so an imitator that merely memorises the demonstrated corridor cannot pass. It is distinct from the binary cross-entropy classifier (BCE) of Section 6, which is a *constraint-learning* ablation rather than a policy.
4. **PUCL-MPC (ours).** The learned PUCL position and velocity networks injected as hard rows into a single-integrator (velocity-level) MPC (point-mass simulation, with the KUKA iiwa-14 the future-deployment target, not a source of reported numbers). The four-geometry results are open-loop trajectory optimisation: one horizon solve replayed without replanning. Receding-horizon closed-loop control appears only as a sanity check, and only with the analytic (oracle) constraint (Section 6).

5.3 Hypotheses

Five falsifiable hypotheses, declared before any result. Each predicted outcome is compared against observation in Section 6.

- H1.** *MPC is a viable nominal policy for constraint learning.* With MPC as PUCL’s forward policy, (i) the learned constraint matches or beats the binary cross-entropy classifier (BCE) ablation on per-order recall, with BCE collapsing on the highest order on the non-convex geometries, and (ii) the resulting PUCL-MPC policy generalises more safely than a constraint-free behaviour-cloning (BC) policy on held-out starts and goals.
- H2.** *The approach scales to new constraint orders.* Position, velocity, acceleration, and jerk are all learnable, with the acceleration boundary at the canonical cap the hardest order to recover.
- H3.** *The approach scales to new obstacle geometries.* One pipeline recovers the position keep-out across all four geometries (retrained per scene), while the scene-independent scalar caps transfer unchanged.
- H4.** *The controller targets real-time deployment.* SQP-RTI bounds the per-call cost to one iteration. For the position case it solves inside the control step, while the higher-order solve remains a target (Section 6).
- H5.** *Explicit constraints bring interpretability, generalisability, and feasibility.* The learned ζ_θ is an inspectable region, reusable across starts and goals, and enforced as a hard NLP row, so a feasible solve has zero violations of the enforced rows by construction. The residual gap to the *true* unsafe set is a learned false negative, not a solver failure.

5.4 Metrics

Every hypothesis is scored against the *true* (analytic) constraint on the metric set of Table 4.

Success criterion. An episode succeeds if it reaches within 5 cm of the goal with zero true-constraint violations. The primary metric is per-order recall (learned versus true unsafe set), the safety-critical quantity. Secondary metrics are task-success rate, reach error, position IoU, and the violation-steps fraction.

Notation. A state is x . The learned constraint network $\zeta_\theta : \mathcal{X} \rightarrow [0, 1]$ is a violation probability and $\tau = 0.5$ its decision threshold, so the predicted unsafe set is $\hat{\mathcal{C}}^c = \{x : \zeta_\theta(x) \geq \tau\}$ and $\mathcal{C}^c$ is the true unsafe set. $|\cdot|$ counts points on a uniform 20 000-point grid. Comparing $\hat{\mathcal{C}}^c$ with $\mathcal{C}^c$ on that grid gives the counts TP, FP, FN, TN of Table 3. Indices t, k run over the N steps of a trajectory and e over the E episodes. $c_t^{\text{true}} \in \{0, 1\}$ is 1 iff the executed state at step t breaks any active constraint: its Mahalanobis keep-out

score falls below $1 + m$ (margin m), or its speed, acceleration, or jerk exceeds a hard cap ($\|v_t\|_2 > v_{\max}$, $\|a_t\|_2 > a_{\max}$, $\|j_t\|_2 > j_{\max}$). Caps are checked at relative tolerance 10^{-6} , so a cap met to the solver's KKT tolerance ($\sim 10^{-8}$) is not miscounted as a breach. $\mathbf{1}[\cdot]$ is the indicator and $\text{succ}_e = 1$ iff episode e reaches the goal within 5 cm with zero true-constraint violations. p_T is the end-effector position at the final step T , g the goal, $\|\cdot\|_2$ the Euclidean norm. j_k is the jerk (the control input in the triple-integrator formulation, though the deployed controller is single-integrator and commands velocity) and t_k the solver wall-clock time of the open-loop horizon solve at step k .

The two primary safety metrics follow Peng & Billard [1]. **Recall** is the fraction of truly unsafe states the learned constraint flags (Recall = 1 misses no hazard). **VSP** (violation-steps portion) is the fraction of executed steps that are unsafe. The two errors differ: a miss (FN) lets a real violation through, a false alarm (FP) only makes the plan conservative.

Combining the orders. VSP is one number per trajectory (a step is unsafe if it breaks any limit, so it covers all orders at once). Recall is measured separately for each learned order on its own space (position in position space, and velocity, acceleration, and jerk each in their scalar-norm space), never mixed, because the units differ. IoU is the overlap of two regions in one space, so there is no joint IoU across orders: it is a clean check only for the position keep-out, while for the velocity, acceleration, and jerk caps it is dominated by class imbalance (Section A.3). Each geometry is therefore summarised by its four per-order recalls (with their minimum and mean) plus the position IoU alone. Recall and VSP, not IoU, carry the safety claims. The deployed single-integrator controller injects only the position and velocity rows, combined as $\zeta_\theta = \max(\zeta_\theta^p, \zeta_\theta^v)$ for planning, so acceleration and jerk are learned but not enforced and the executed trajectory can exceed $a_{\max}$ and $j_{\max}$. Promoting them to hard rows is the queued next step. A feasible solve has zero violations of the enforced rows by construction, yet the executed policy still crosses the true unsafe set on 25–47% of steps, because the hard wall sits at threshold 0.15 on a deliberately over-grown boundary.

		flags unsafe ($\zeta_\theta \geq \tau$)	flags safe ($\zeta_\theta < \tau$)
Truth	unsafe	TP (<i>caught</i>)	FN (<i>missed</i>)
	safe	FP (<i>false alarm</i>)	TN (<i>correct</i>)

Table 3: Confusion matrix for the learned constraint. Recall reads along the *unsafe* row, $\text{TP}/(\text{TP} + \text{FN})$. Only the **FN** cell is highlighted: the single safety-critical error, a truly unsafe state the constraint misses. A false alarm (FP) costs only conservatism.

Metric	Formula	What it measures
<i>Primary safety metrics</i> (Peng & Billard [1])		
Recall $\uparrow$	$\frac{TP}{TP + FN}$	Fraction of truly unsafe states the learned constraint flags.
VSP $\downarrow$	$\frac{1}{N} \sum_t c_t^{\text{true}}$	Fraction of executed steps that violate the true constraint.
<i>Auxiliary metrics</i>		
Task success $\uparrow$	$\frac{1}{E} \sum_e \mathbf{1}[\text{succ}_e]$	Episodes reaching the goal (≤ 5 cm) with zero violations.
Reach error $\downarrow$	$\ p_T - g\ _2$	Final end-effector distance to the goal g (mm).
IoU (position) $\uparrow$	$\frac{ \hat{\mathcal{C}}^c \cap \mathcal{C}^c }{ \hat{\mathcal{C}}^c \cup \mathcal{C}^c }$	Area overlap of predicted and true position keep-out (3-D grid). Position only (see text).
Smoothness $\downarrow$	$\frac{1}{N} \sum_k \ j_k\ _2$	Mean jerk norm (lower is smoother).
Solve time $\downarrow$	$\text{median}_k t_k$	Median wall-clock per open-loop MPC solve. Reported deployed solves use IPOPT (132–197 ms). For the position NLP a warm-started SQP-RTI solve is measured at median 3.1 ms (2.8–4.5 ms over 50 calls, all status 0) against 177 ms for IPOPT on the same machine, a $57\times$ speed-up inside the 100 ms step. The higher-order solve stays a target.

Table 4: Evaluation metrics. $\uparrow/\downarrow$: higher / lower is better. N steps over E episodes. The two primary safety metrics follow Peng & Billard [1]. IoU is auxiliary: it applies to the position keep-out only and is low because the learned region is conservative. Recall and VSP carry the safety claims.

The grid-based definitions of recall, precision, IoU, and accuracy, the per-component grids, and the class-balance correction are in Appendix A.3. On the one-sided norm caps the velocity, acceleration, and jerk IoU and accuracy are class-imbalance inflated, so those three orders are re-scored on a boundary-balanced sample (Section 6.4), while the position network keeps its natural grid and its low IoU is a genuine conservatism finding.

Statistical protocol. Constraint-quality results (per-order recall, precision, IoU, accuracy) are reported as mean $\pm$ standard deviation over 5 random seeds per geometry, each for 8 PUCL iterations, and the seed was not tuned. The per-order velocity, acceleration, and jerk scores are evaluated on a deterministic norm-cap grid and are seed-independent ($SD \approx 0$). Binary KPIs (task success) are reported as the rate with a Wilson 95 % confidence interval. Constraint-violation reduction (constrained vs. unconstrained MPC) is tested with a one-sided Wilcoxon signed-rank test at $\alpha = 0.05$ across the $4 \times 8 = 32$ geometry-seed pairs. A single comparison is made, so no multiple-comparison correction is required. When every paired difference shares the same sign the rank statistic degenerates, in which case we report the outcome qualitatively (Section 6.5).

5.5 Hyperparameters and evaluation protocol

The MPC cost weights (w_g, w_t, w_j, w_a), the planning horizon N , and the reliable-negative position threshold τ_p are fixed by grid search on held-out seeds. The deployed values sit in a flat, non-fragile basin (Section 6.8), and the full sweep is in Appendix A.2. The constraint learner is adopted from Peng & Billard [1] and is not tuned here. Each environment is then evaluated by one diagnostic pass, run once per environment so outputs are comparable. The diagnostics carrying the headline claims (per-order recall in Table 6, the constraint-order ablation in Table 9, and the solver timing in Table 17) appear in the Results.

6 Results and Discussion

This section reports evidence for the five hypotheses (H1–H5) of Section 5.3, mapped in Table 5. The headline is the learning result: with the MPC as PUCL’s forward policy, the method recovers all four

constraint orders on all four obstacle geometries (H1, H2, H3). The solver and deployment evidence is a separate strand (Section 6.5, H4), and the benefits of an explicit learned constraint run through the rest (H5).

Three scope facts hold across every number below and are stated once. The deployed evaluation is open-loop: one horizon solve replayed without replanning (receding-horizon control appears only as a sanity check, with the analytic constraint, Section 6.7). The deployed controller is single-integrator (velocity-level). The triple-integrator jerk-as-control formulation of Section 4 is a model capability, not the configuration evaluated here. Acceleration and jerk are learned but not enforced in any reported solve. The consolidated limitations are in Section 6.13.

Table 5: Map from the five hypotheses to result sections and key figures.

H	Claim (short)	Sec.	Evidence
H1	MPC is a viable PUCL forward policy: PUCL beats the BCE ablation on recall (BCE jerk collapses), and PUCL-MPC beats behaviour cloning (BC) on held-out start/goal	6.12	Fig. 10, Tab. 6, 10
H2	Scales to orders: pos/vel/acc/jerk all learned (recall ~ 0.99 , acc cap-dependent)	6.4	Tab. 6
H3	Scales to environments: same pipeline on four geometries	6.1	Fig. 4, Tab. 7
H4	Targets real time: IPOPT 132–197 ms, acados position case 3.1 ms measured	6.5	Tab. 8, 17
H5	Explicit constraint: inspectable, reusable, hard NLP row	6.3	Fig. 8

6.1 Headline results

Four results, in order of how decisively they hold.

First, PUCL recovers all four constraint orders on all four geometries (Table 6): over 5 seeds at the final iteration the per-order recall of the learned-versus-true unsafe set is 0.99 for position, velocity and jerk (per-seed SD ≤ 0.009), so almost no truly-unsafe state is missed. This is the coverage side of H2 (orders) and H3 (geometries).

Second, the same view exposes the sharpest finding, the jerk collapse: on the two non-convex geometries the binary cross-entropy classifier (BCE) jerk recall falls to 0.13 (concave) and 0.00 (two regions) while PUCL holds 0.99. Position recall stays high for both (0.91–1.00), so the gap is jerk, not position. Without reliable-negative mining BCE cannot separate the noisy jerk signal on hard geometries and flags nothing. PUCL’s positive-unlabelled selection recovers it. This is the constraint-learning evidence for H1 (does the PU step matter): a separate, policy-level comparison against constraint-free behaviour cloning (BC) is in Section 6.12.

Third, the learned position and velocity rows deploy: compiled into the NLP they let the velocity-mode MPC reach the goal with zero violations of the enforced rows on a feasible solve (Section 6.12), where BC cannot bound velocity at all. That zero is partly by construction, and the same open-loop policy still crosses the true unsafe set on 25 to 47% of steps (Section 6.3). This is the H5 picture: an explicit hard row buys zero enforced-row violations, not true safety. Closing the residual is the closed-loop check (Section 6.7).

Fourth, and more preliminary: label-free the MPC recovers the position keep-out only conservatively (global IoU 0.015–0.049), so we test whether a minimal contact signal tightens it. A binary collision query at probed states, refined over six iterations (five seeds, Fig. 3), holds recall near 1.0 while precision and IoU climb to a per-geometry IoU plateau of 0.60–0.90 (Fig. 8). The refined constraint deploys with ≈ 0 true collision steps (Section 6.10). This is a proof-of-concept extension of PUCL with a contact signal, not a label-free result.

Two caveats recur below and are stated once. The acceleration grid recall reads 1.00 but is class-imbalance inflated. The boundary-balanced recovery at the canonical cap is markedly weaker, acceleration being the weakest of the four orders (Section 6.4). The velocity, acceleration and jerk caps are scored on a one-sided norm grid that is $\sim 97\%$ unsafe, so their grid IoU runs near 1.0 as an artefact and is never headlined (recall is the honest cap metric, Section A.3).

The four orders are learned jointly across the four geometries by four *independent* networks, each reaching recall ~ 1.0 from the first iteration. This one-shot recovery is a sample-efficiency property of

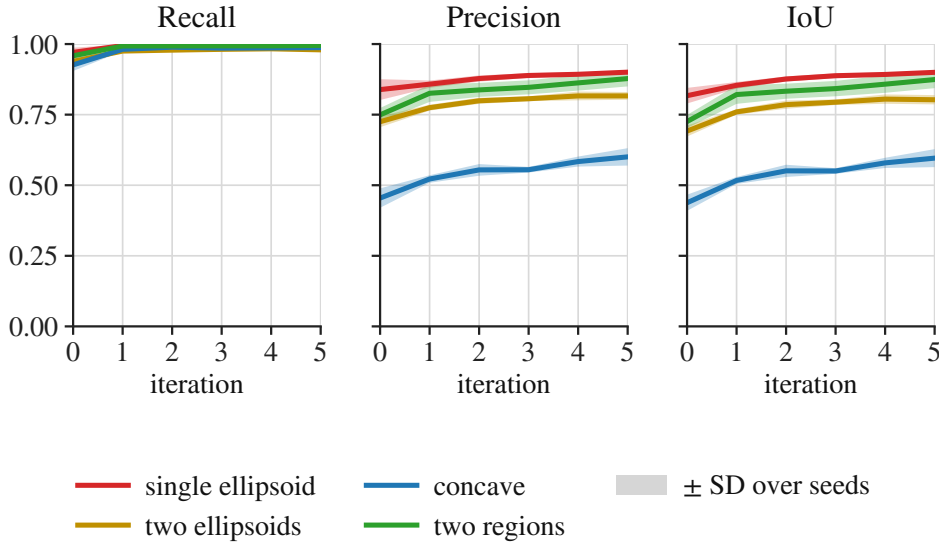


Figure 3: Tightening the conservative position keep-out with a binary collision probe. The label-free PUCL position keep-out over-covers, so its precision and IoU are low. A minimal contact check that returns only safe or unsafe at queried states, refined over six iterations on four geometries (five seeds, mean $\pm$ SD), tightens it: recall stays near 1.0 throughout (left), so no truly-unsafe state is dropped, while precision (centre) and IoU (right) climb to a per-geometry plateau. The deployment benefit of the tighter keep-out is reported separately, as two measured conditions rather than a per-iteration curve, in Figure 9. A proof-of-concept extension of PUCL with a contact signal, this lifts the position **IoU from a label-free 0.01–0.05 to 0.60–0.90 at recall $\sim$ 1.0.**

mining reliable negatives from demonstrations. Constrained reinforcement learning instead re-converges a reward-penalty multiplier over $\sim 10^5$ timesteps each time ζ_θ is retrained [1, 2]. What the iterations buy differs by order. *Position* is the informative order: recall reaches ~ 1.0 , but precision and IoU stay low (0.1–0.3) and barely move, so the keep-out covers the true obstacle yet over-grows it, the conservative over-cover that the active-refinement probe closes (Section 6.10). *Acceleration* is the only higher order whose norm-grid recall is not one-shot, taking the first two iterations to saturate. On a boundary-balanced sample at the canonical cap it remains the weakest of the four orders. *Velocity* and *jerk* are learned in a single iteration (recall 1.0 from iteration 0) and then stay flat. Their precision and IoU read near 1.0 but are class-imbalance inflated on the one-sided norm grid, so recall, not IoU, is the honest cap metric for these orders. The whole-policy deployment violation, identical across orders because the policy enforces them jointly, falls from ≈ 0.98 to 0.2–0.5 as the constraints are learned (shown once in the scorecard, Figure 4).

PUCL thus recovers the full derivative ladder of kinodynamic limits (canonical caps $v_{\max}=0.60$, $a_{\max}=2.1$, $j_{\max}=42$): a parametric ellipsoid on position and one scalar-norm threshold each on $\|v\|$, $\|a\|$, $\|j\|$, sharing only the demonstration and rollout streams so no classifier contaminates another. Because each scalar net reads only the signal norm, one trained velocity checkpoint transfers unchanged across the four geometries (H3).

Deploying the learned position and velocity rows, the open-loop true-violation rate falls from 0.98 to 25% (concave), 26% (single ellipsoid), 32% (two ellipsoids) and 47% (two regions), nonzero because the policy plans once and executes open-loop. The enforced velocity row holds exactly (executed $\|v\|$ at 0.64 to 0.69 against the 0.60 cap, Table 10), while acceleration and jerk, learned but absent from this NLP, stay within cap on 84.8 to 92.5% and 91.7 to 96.2% of steps, with large overshoot where they exceed it (peak $\|a\|$ up to $\sim 3.5\times$, $\|j\|$ up to $\sim 2.8\times$ the cap). The analytic caps are env-design choices on synthetic finite-difference kinematics, so this gap measures distance to a chosen envelope, not a safety breach.

6.2 Per-environment scorecard [H2, H3]

Figure 4 collects three whole-environment metrics (per-order recall, violation-steps portion, and solve time) for each of the four geometries, with the full numbers, including precision and IoU, in Table 7: one

Table 6: Per-order recall of the learned-versus-true unsafe set, PUCL (ours) versus the binary cross-entropy classifier (BCE), mean over 5 seeds at the final PUCL iteration, four geometries. BCE is the PU-step ablation: the same networks trained by plain cross-entropy without reliable-negative mining (a constraint-learning baseline, not the behaviour-cloning policy of Table 10). Recall is the fraction of truly-unsafe states flagged. The velocity / acceleration / jerk scores are on a one-sided norm grid (seed-independent). On that grid ($\sim 97\%$ unsafe) PUCL holds recall ~ 0.99 , but the grid recall is class-imbalance inflated and near-vacuous (as is the IoU): the boundary-balanced acceleration recovery at the canonical cap ($a_{\max}=2.1$, $j_{\max}=42$) is markedly weaker, acceleration being the weakest of the four orders (Section 6.4). BCE matches PUCL on position everywhere, but its jerk recall collapses on the non-convex scenes. **Red** : BCE jerk recall collapse. The two-region BCE jerk entry is a single deterministic run (5 seeds bit-identical, effective $n=1$), so it is a stability gap, not an effect size.

Geometry	pos		vel		acc		jerk	
	PUCL	BCE	PUCL	BCE	PUCL	BCE	PUCL	BCE
single ellipsoid	0.99	0.91	1.00	1.00	1.00	1.00	0.99	0.98
two ellipsoids	0.99	0.95	1.00	1.00	1.00	0.99	1.00	0.98
concave	1.00	0.98	1.00	0.81	1.00	1.00	0.99	0.13
two regions	1.00	0.98	1.00	1.00	1.00	1.00	0.99	0.00

column per environment is its profile. The solver and closed-loop strands it points to are in Sections 6.5 and 6.7.

Table 7: Per-environment scorecard, numeric companion to Figure 4. PUCL (ours), final iteration, mean over 5 seeds. **Recall** is per order. **Prec.** and **IoU** are the position keep-out only. Since position recall ≈ 1 , precision $\approx$ IoU and both equal the conservative over-cover fraction (velocity / acceleration / jerk precision and IoU are class-imbalance inflated and omitted from the figure, reported in this table only). **VSP** is the whole-policy open-loop violation portion (oracle 0). **Solve** is the average deployment IPOPT solve time (Table 8). Acceleration recall reads 1.00 on the norm grid but is markedly weaker boundary-balanced (the weakest of the four orders, Section 6.4).

Environment	Recall (per order)				Prec.	IoU	VSP	Solve
	pos	vel	acc	jerk	pos	pos	(all)	[ms]
single ellipsoid	0.99	1.00	1.00	0.99	0.10	0.10	0.26	197
two ellipsoids	0.99	1.00	1.00	1.00	0.11	0.11	0.32	132
concave	1.00	1.00	1.00	0.99	0.05	0.05	0.25	156
two regions	1.00	1.00	1.00	0.99	0.23	0.23	0.47	151

The pattern is uniform across the four geometries (H3): minimum per-order recall 0.99, position precision and IoU low (0.05 to 0.23) from the recall-favouring PU objective, the velocity / acceleration / jerk values near 1.0 a class-imbalance artefact with no tightness claim, open-loop violation portion 0.25 to 0.47, and IPOPT solve time 132 to 197 ms, above the 100 ms budget.

6.3 PUCL constraint quality [H5]

The learned ζ_θ is an inspectable region, the first benefit of an explicit learned constraint (H5). Scored against the true ellipsoidal keep-outs on a uniform 3-D grid, the position network attains recall 0.99–1.00 from the first iteration (one-shot: only precision and IoU change afterward), with a conservative boundary (near-support IoU 0.05–0.23) because the PU objective favours recall over precision.

This is the desirable failure direction: the learned wall sits outside the true ellipsoid, pushing the MPC away from the boundary, which is why PUCL-MPC records zero violations of the *enforced* rows on a feasible solve (Section 6.12). That guarantee is over the enforced rows, not the executed policy: open-loop the policy still crosses the *true* set on 25 to 47% of steps, so true safety is the closed-loop, robust-tightening result of Section 6.7. The cost of the conservative boundary is reachability: on a sizeable minority of held-out goals the inflated keep-out blocks the goal (54% position-scene success under held-out start and goal, Table 10).

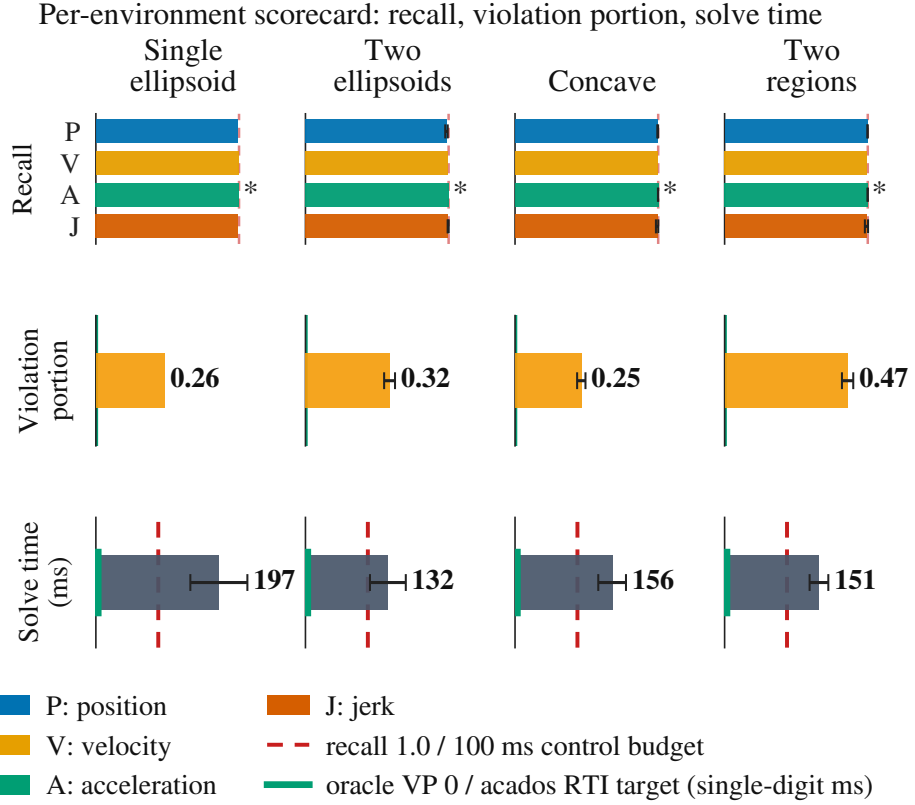


Figure 4: Scene: the four benchmark geometries, all four constraint orders. Method: PUCL (ours) at the final (8th) iteration, mean $\pm$ SD over 5 seeds. KPI: three whole-environment rows per geometry (column): per-order recall, violation-steps portion (VSP), and solve time. Recall is the honest per-order safety metric. The acceleration bar is starred because its norm-grid recall (1.0) is class-imbalance inflated (boundary-balanced recovery at the canonical cap is markedly weaker, acceleration being the weakest order, Section 6.4). Precision and IoU are reported in the companion Table 7 only: meaningful for position (a low value is conservative over-cover, not a miss) and class-imbalance inflated for the higher orders, so they are kept off the figure (Section A.3). Violation-steps portion (VSP) and solve time are whole-policy: VSP is the open-loop learned policy with acceleration and jerk soft (oracle 0), and solve time is the deployment IPOPT NLP (Table 8) against the dashed 100 ms budget, with acados SQP-RTI measured at 3.1 ms on the position case (Section 6.5). Every order is recovered on every geometry at minimum per-order recall 0.99. The learned keep-out is safe but conservative.

6.4 Learning the acceleration and jerk constraints [H2]

Scaling to higher orders (H2) means learning the acceleration and jerk caps. We isolate how the PU learner recovers a one-sided scalar threshold on a dedicated triple-integrator study at its own spec ($a_{\max}=1.20$, $j_{\max}=6.00$), *not* the four-geometry caps ($a_{\max}=2.1$, $j_{\max}=42$): a single-threshold study answers the sharpness question (where the learned 0.5-level wall lands) more cleanly than the geometry runs, whose recall is the ~ 1.0 of Table 6.

A recovery run at the canonical caps shows the recovery is cap-dependent: jerk holds (recall 0.99, IoU 0.87), but acceleration is markedly weaker, the weakest of the four orders, with the learned wall mislocated relative to the 2.1 cap so that it under-covers (boundary-shell recall 0). The 1.20/6.00 study below is the easier legacy regime where both recover at recall 1.00. At the canonical acceleration cap the norm-only learner is not yet reliable, which we flag rather than headline.

A norm cap $\|\cdot\| \leq c$ is a one-sided scalar threshold, so we reuse the velocity formulation of Peng and Billard [1] unchanged: a small network on the signal norm, changing only which signal is read. Training data come from the triple-integrator planner: cap-riding rollouts (caps on, smoothing off) give safe positives near the boundary, over-driven rollouts give boundary negatives. Cap-riding is essential for jerk, which the objective penalises: ordinary demonstrations never reach the 6.0 cap, and a learner

trained on them collapses the threshold to the low-jerk mass (a 26% error the cap-riding data corrects to 6.7%).

At the legacy caps the learned 0.5-level threshold lands at 1.12 m/s^2 (true 1.20, 6.3% low) and 5.60 m/s^3 (true 6.00, 6.7% low), both conservative. On the boundary-balanced sample both reach recall 1.00 at precision/IoU 0.94 (acceleration) and 0.96 (jerk), with boundary-shell precision/recall 0.84/1.00 and 0.88/1.00: the boundary is correctly localised, not class-imbalance inflated. Compiled into the NLP under an objective that drives the higher orders past their caps, the learned jerk row clamps a 30-goal run from peak $\|j\| = 31.2 \text{ m/s}^3$ (80.5% over cap) to 5.95 m/s^3 at 0% violations.

H2 partially holds. All four orders are learned and the recall (safety) side holds (~ 0.99 , no unsafe state missed), but the tightness side is partial: the boundary-balanced acceleration recovery at the canonical cap is markedly weaker (acceleration is the weakest of the four orders), and the label-free position keep-out is conservative, so PUCL recovers *coverage* of all four orders but not a *tight* boundary label-free.

6.5 MPC solver and real-time outlook [H4]

This is the solver-and-deployment strand (H4): not a constraint-learning result, but evidence that the NLP the learned rows are injected into is well posed and a path to real-time use.

The NLP itself is sound. With the *exact* analytic constraint, oracle MPC reaches **100 % task success (Wilson 95 % CI [0.89, 1.0], 32 geometry-seed runs) and 0 constraint-violation steps** across the four geometries, each reaching its goal at mean reach error below 4 mm. The slowest is single ellipsoid at 197 ms average IPOPT solve time (Table 8). All four carry the full pos+vel+acc+jerk stack.

Table 8: Oracle MPC IPOPT solve time [ms] across 8 seeds $\times$ 4 geometries (32 runs). Green: 100 % task success, 0 violation steps on all seeds. **Bold**: fastest seed per row. Constraints: **pos** / **vel** / **acc** / **jerk**. All four geometries carry the full stack. Reach error (mm) is the mean over the 8 seeds.

Env.	C	Reach (mm)	Solve time [ms]								Avg [ms]	Std [ms]
			s1	s2	s3	s4	s5	s6	s7	s8		
single ellipsoid	pvaj	0.945	204	194	187	129	303	174	186	199	197	46
two ellipsoids	pvaj	3.936	206	123	112	118	125	134	115	124	132	29
concave	pvaj	0.012	172	144	135	204	146	134	165	144	156	22
two regions	pvaj	0.012	174	157	155	135	132	134	155	167	151	15

The constrained MPC records 0 violation steps on all 32 pairs while the unconstrained baseline violates the cap. A one-sided Wilcoxon signed-rank test is degenerate here (every difference has the same sign), so we report the qualitative outcome rather than an exact p . The comparison against BC is in Section 6.12.

The ablation below isolates the fourth constraint order on one solver environment at the solver-study spec ($v_{\max}=0.60$, $a_{\max}=1.20$, $j_{\max}=6.0$, not the four-geometry caps). The solver-cost conclusion is cap-independent.

Table 9: Constraint-order ablation on the solver-study environment **ellipsoid3dposvel** (single seed). Each row adds one constraint order. Caps are the solver-study spec $v_{\max}=0.60$, $a_{\max}=1.20$, $j_{\max}=6.00$ (not the four-geometry caps $a_{\max}=2.1$, $j_{\max}=42$). Green: signal equals the cap exactly. Reach and violations are unaffected. Solve time grows with active NLP rows.

Orders	$\ v\ _{\max}$	$\ a\ _{\max}$	$\ j\ _{\max}$	Viol/Steps	Reach (mm)	Solve (ms)
pos	2.23	4.24	22.9	0/56	0.12	75
pos+vel	0.60	2.22	20.7	0/56	1.08	96
pos+vel+acc	0.60	1.20	12.0	0/56	0.94	74
pos+vel+acc+jerk	0.60	1.20	6.0	0/55	0.85	165

IPOPT converges on every configuration: the worst case (all four orders) takes at most 69 interior-point iterations and returns a certified optimum with zero violations. What grows is wall-clock time: each added hard row enlarges the KKT system, so the solve rises from 75 ms (position only) to 165 ms (all four orders), exceeding a real-time budget. The slowdown is solver cost, not a failure of ζ_{θ} (the learned constraint is unchanged across the ablation).

This is where H4 sits. The acados SQP-RTI backend takes one real-time-iteration step per call, so its cost is bounded and constant in the number of NLP rows rather than growing with interior-point iterations. On the position NLP this is now measured: warm-started SQP-RTI returns a median 3.1 ms per solve (2.8–4.5 ms over 50 calls, solver status 0) against 177 ms for IPOPT on the same machine (8 seeds), a $57\times$ speed-up inside the 100 ms control step (Figure 5, Table 17). Two caveats, stated once. The measurement is the position case only: with the velocity ball added the SQP-RTI QP does not yet converge (NaN/QP-failure under the current $\pm 10^9$ bound conditioning), so the higher-order figure remains a target and every solve time *reported* in this thesis, hence every deployed solve, is IPOPT (132–197 ms in Table 8, 75–165 ms in the ablation of Table 9). Hard-constraint satisfaction holds either way, since acados enforces the same rows at every stage. The looser QP tolerance (10^{-4}) affects convergence speed, not feasibility.

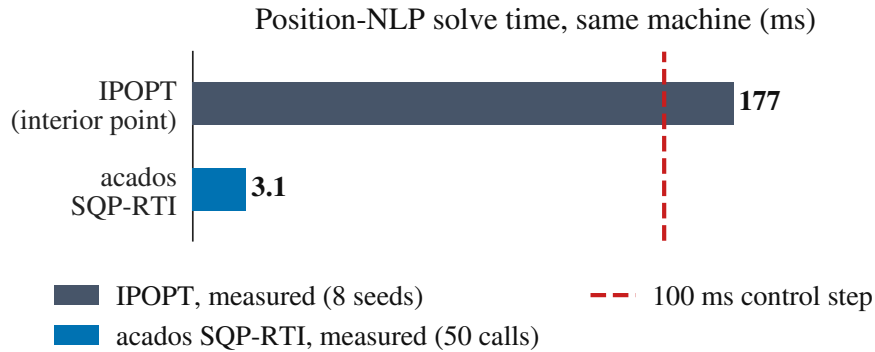


Figure 5: Same-machine position-NLP solve time: warm-started acados SQP-RTI versus IPOPT on the identical CasADi OCP, both read from the committed benchmark log (Table 17), with the 100 ms control step marked. IPOPT takes a median 177 ms (8 seeds) and overruns the step, while acados SQP-RTI takes 3.1 ms (50 calls, all status 0), a $57\times$ reduction that fits well inside it. Scope: the position case only. Once velocity or any higher order is added the SQP-RTI QP does not yet converge, so every reported deployment solve runs IPOPT (the IPOPT order-growth is in Table 9).

6.6 KUKA deployment outlook [C4]

The point-mass plans target the 7-DoF KUKA iiwa-14. We plan one trajectory with the *learned* keep-out ζ_θ enforced as the MPC’s hard NLP row (the same checkpoint deployed elsewhere in this work), lift it to the world frame, and realise it on the iiwa-14 in PyBullet through differential inverse kinematics (Fig. 6). The arm tracks the planned, collision-free end-effector path, lifting up and over the keep-out from start to goal, and keeps a visible margin from the true obstacle because the learned keep-out is conservative (larger than the true region). That conservatism is useful here: the wider detour is what lets the *whole* arm clear the obstacle at every pose, since the Cartesian planner constrains the end-effector only and a tight skirt would otherwise bury the forearm in the keep-out (the whole-body limitation is in Section 6.13). This is therefore the learned constraint deployed in the planner and IK-realised on the arm, not a whole-body or closed-loop hardware result. Running it in a true closed loop on hardware under live contact physics is the queued deployment step (Section 7).

6.7 Closed-loop control

The hypotheses above are evaluated open-loop. The receding-horizon controller is verified once on `ellipsoid3dposvel` (CasADi/IPOPT, analytic constraint): a position kick that drives the open-loop replay to ≈ 180 mm from the goal is corrected to within the 50 mm tolerance (36 mm after 24 re-solves). A separate every-step re-solve evaluation shows the loop reduces, but does not by itself eliminate, true-constraint violations under a persistent disturbance: on the velocity environments it leaves a residual 27% (12/44 steps, `ellipsoid3dvel`) and 32% (14/44, `ellipsoid3dposvel`) over the cap, down from 74% and 65% open-loop, while `ellipsoid3d` reaches 0. Driving the residual to zero on every seed needs

KUKA iiwa-14: planned trajectory executed in PyBullet

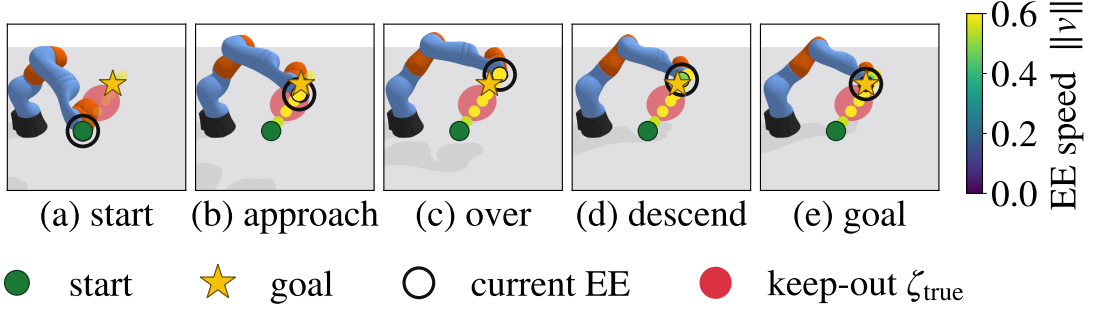


Figure 6: The KUKA iiwa-14 in PyBullet executing one trajectory planned by the MPC with the *learned* keep-out ζ_θ enforced as a hard NLP row, shown at five stages from start to goal as the end-effector lifts up and over the keep-out. The plan uses the same PU-learned position network deployed elsewhere in this work, solved in the point-mass abstraction, lifted to the world frame and realised on the arm by differential inverse kinematics. Dots are end-effector waypoints coloured by speed, the black ring marks the current end-effector, and the solid red region is the true, full-size obstacle ζ_{true} . The end-effector keeps a margin from the true obstacle because it respects the larger conservative learned keep-out, and that margin is what lets the whole arm clear the obstacle at every pose (minimum closest-point clearance 35 mm). The figure shows the learned constraint deployed in the planner and realised on the arm in simulation, with end-effector planning only, not a whole-body or closed-loop hardware result.

robust constraint tightening (an inflated planner-side keep-out): the 8-seed sweep reaches 0 violation steps on all three environments (Fig. 7) and 8/8 goal-success on the two velocity environments (Wilson 95% CI [0.68, 1.00]). Goal-reaching on the position-only `ellipsoid3d` is pending re-verification, as the committed sweep predates later environment-code changes. The sweep records no per-call solve time, so the real-time feasibility of every-step re-solving is unmeasured. Closing the loop on the learned ζ_θ instead of the analytic keep-out, injected as the same hard NLP row, gives the same safety outcome: across all three environments at eight seeds the true violations again fall to 0.0 (from 38–41 open-loop), so the zero-violation guarantee holds for the learned constraint, not only the oracle. The conservative learned keep-out does cost goal-reach under the same disturbance, with success from 100% on the full position-velocity-acceleration-jerk stack down to 0% on the position-only scene, the same conservatism seen open-loop and in the generalisation test (Section 6.12). Running the loop under live PyBullet contact physics on the arm is the remaining deployment step (Section 7).

6.8 Hyperparameter sensitivity

MPC cost weights were fixed by joint optimisation on held-out seeds, and the horizon N and PUCL reliable-negative threshold τ_p by grid search on `ellipsoid3dposvel`. The deployed values ($N=50$ on the four geometries, $w_{\text{goal}}=70$, $w_{\text{jerk}}=1 \times 10^{-3}$, $\tau_p=0.3$) each sit in a flat, non-fragile basin, with the full searches in Appendix A.2. This τ_p is distinct from the 0.5 constraint-quality decision threshold and the NLP hard-row injection bound below.

6.9 Did the MPC integrate correctly into the PUCL pipeline?

The integration is mechanically correct. Each outer iteration ζ_θ is compiled into a CasADi function and attached to the optimal-control problem on two channels: a soft squared-hinge penalty in the running cost and, when a hard threshold is supplied, the inequality row $\zeta_\theta \leq (\text{hard threshold})$, with the latest weights re-injected and the solver rebuilt each iteration (Appendix A.4). The learned set reaches the controller as constraint geometry, not a post-hoc filter.

The mismatch is methodological, not mechanical. PUCL is designed for a *stochastic, exploring* forward policy: an RL agent that over $\sim 10^5$ steps emits diverse unlabelled rollouts bracketing the true boundary from both sides. Our MPC is the opposite: *deterministic*, soft-penalised, one locally optimal trajectory per demonstration start. A single path shows where the safe optimum lies but not how far the unsafe

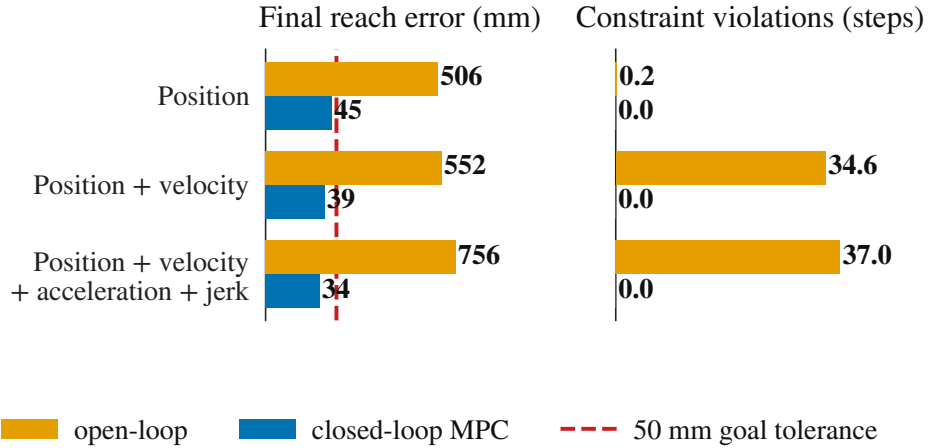


Figure 7: Scene: every-step re-solving MPC versus open-loop replay under a persistent disturbance, on the ellipsoid environment across three cumulative constraint stacks (position, position + velocity, position + velocity + acceleration + jerk), eight seeds. Method: closed-loop re-plans from the measured state every step with a robust planner-side keep-out tightening, while open-loop replays the first plan once. The constraint is analytic (oracle) and solved with CasADi/IPOPT, so closing the loop with the learned ζ_θ remains the queued deployment step. KPI: final reach error (left, log scale, 50 mm goal tolerance dashed) and constraint-violation steps (right). Open-loop degrades as the stack grows, drifting to 506–756 mm and accumulating 0.2, 34.6 and 37.0 violation steps across the three stacks, whereas **closed-loop reaches the goal at 34–45 mm with 0 violation steps on all three stacks.**

region extends, so the learner must recover a region from a policy that does not explore it. This is the first hypothesis for the loose boundary.

Three signatures follow. The keep-out is *loose*: high recall (position 0.985–0.999) but position IoU only 0.055–0.226 (final iteration, five seeds), so the recovered set is roughly four times too large on two regions and eighteen times on concave (the order-averaged IoU ≈ 0.76 is class-imbalance inflated by the 97–98%-unsafe higher-order grids, and position is the only discriminative number). Convergence is *one-shot*: recall is flat from iteration 0 because the deterministic planner jumps straight to the constrained optimum. And open-loop the policy still violates the true set on 25–47% of steps, every episode (fractions 0.260, 0.319, 0.252, 0.467): a hard row at the injection bound (0.15 position, 0.2 velocity) on a loose boundary, plus a soft penalty traded against goal-reaching, does not deliver hard safety against the *true* constraint.

The cause is the PU objective, not rollout diversity or architecture: four jittered rollouts per demonstration leave the position IoU unchanged (~ 0.1 , recall ~ 1.0), and the parametric net over-covers to the same ~ 0.1 IoU as a free-form MLP. The over-cover is mostly *off-support*: scoring the concave net in a ball of radius R lifts IoU from 0.016 to 0.224 as R tightens to 0.45 (recall ~ 0.99). A safe-sparsity prior, labelling uniform-workspace samples safe, confirms this: it lifts global IoU 2.6–3.3 $\times$ and cuts far-space false alarms $\sim 70\%$ with recall preserved (Table 15).

6.10 Recovering a tight keep-out with active collision feedback

The safe-sparsity prior removes off-support false positives but does not make the on-support boundary tight (global position IoU still 0.039–0.165). No label-free objective can: when the unlabelled set comes from the non-exploring controller of Section 6.9 it never brackets the true boundary, so the over-covered-but-safe region cannot be separated from the true keep-out. Tightening needs an external signal.

We supply the smallest such signal: a binary collision query at probed states (a contact-sensor analog returning only safe or unsafe, never the analytic value or gradient), refining the position network on the queried labels. This *extends* the positive-unlabelled method rather than replacing it, and we report it as such.

Dense probing recovers a tight, safe keep-out. Sampling the operating region uniformly, labelling by the collision check, and refining over iterations tightens the keep-out from the pure-PUCL global IoU

0.015–0.049 to 0.60–0.90 with recall near one, over five seeds (Figure 8): the region hugs the true keep-out on every geometry. Unlike the one-shot label-free loop, the boundary sharpens gradually as probes accumulate (Figure 3).

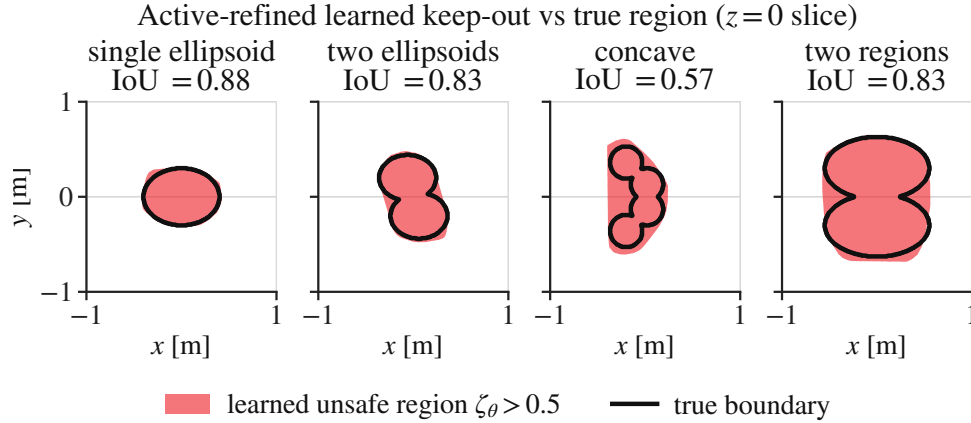


Figure 8: Scene: active-refined learned position keep-out ζ_θ versus the analytic true region, $z=0$ slice, four geometries (seed 0). Method: dense uniform collision-feedback refinement, scored at convergence (8000 collision queries). KPI: the learned unsafe region ($\zeta_\theta > 0.5$) is the solid red fill and the true analytic boundary is the black outline. Each panel reports the slice IoU. Recall stays ~ 1.0 across geometries. The slice IoU is 0.83–0.88 on the three convex scenes and 0.57 on the concave pocket, the hardest geometry, where the recovered region matches the bulk of the keep-out but not the full scallop.

Probes drawn instead from the controller’s own rollouts (keeping negatives policy-generated) hit the same exploration limit: a looser, under-covering set (global IoU 0.42–0.83, recall below the 0.95 safety floor on three geometries, large seed variance), because goal-directed rollouts cover corridors, not the obstacle neighbourhood. Softening the constraint so the policy dips into the keep-out does not rescue this either: visiting is not labelling (the reliable- negative selector flags only states far from the demonstrations, never the near-boundary ones), and a goal-tracking MPC shaves the corner rather than sweeping the obstacle. A tight, safe keep-out needs dense boundary coverage, which a deterministic goal-directed controller does not provide. This is also why the deployed controller stays hard, since softening it would surrender the zero enforced-row violations hard enforcement buys. An explicitly exploring controller that recovers pure label-free learning is left to future work.

A tighter keep-out also deploys better. With the same controller and velocity network and the same demonstration starts, the tight keep-out incurs essentially no true collisions (zero steps on three geometries, 0.1 on the two-region scene) and reaches closer, whereas the conservative keep-out incurs 2.2 to 21.2 collision steps and stalls further out (Figure 9). We report this as two measured conditions rather than a per-iteration collision curve on purpose: with the keep-out compiled as a hard NLP row, a feasible solve has near-zero true collisions at *every* refinement iteration regardless of the keep-out’s tightness, so a per-iteration deployed-collision curve would be flat and uninformative. The refinement signal therefore lives in the constraint-set metrics (recall, precision and IoU, Figure 3), and its deployment payoff in the conservative-versus-tight contrast here. This is enforcement, not labelling: at the colliding states both networks assign violation probability ≈ 1 (mean 0.997, with 254 such states on two regions and 92 on two ellipsoids), but the large inflated keep-out drives the soft-penalised controller through the flagged obstacle while the tight region is cheaply avoided. Both rollouts are open-loop and single-shot, so neither meets the 50 mm tolerance. Task success comes from the closed-loop re-solve of Section 6.7, and the decisive quantities here are the collision count and reach. This extension uses a collision signal the label-free method forgoes, so it does not contradict the conservatism result: it localises that result to the label-free regime.

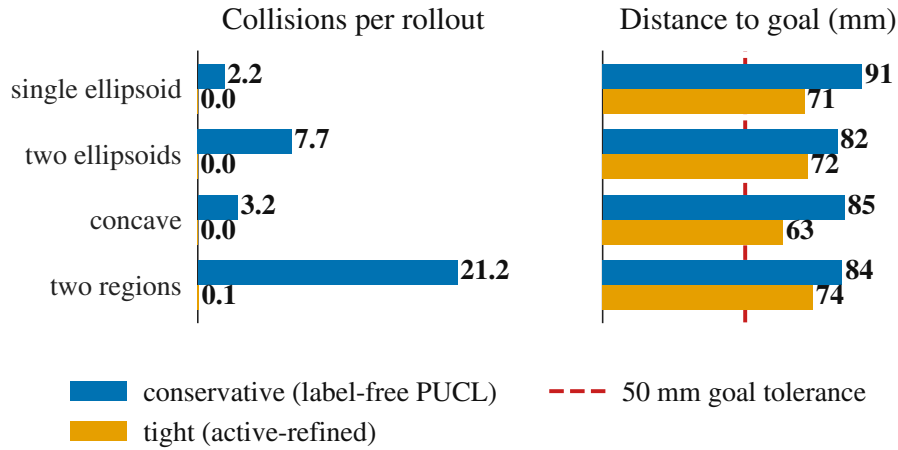


Figure 9: Scene: tight (active-refined) versus conservative (pure-PUCL) position keep-out deployed in the same MPC with the same velocity network, rolled out from the demonstration starts. Method: identical controller and settings, only the position network differs. KPI: true collision steps per rollout (left) and final distance to goal (right), four geometries. Conservative keep-out in blue, tight (active-refined) in orange, and the dashed red line marks the 50 mm goal tolerance. The tight keep-out drives the deployed collisions to ≈ 0 and reaches closer on every geometry, so the recovered tightness yields a safer and more accurate rollout, removing 21.2 collision steps per rollout in the worst case (the two-region scene).

6.11 Discussion

The central finding. The deepest result is structural: pairing PUCL with a safe, optimal MPC forward policy exposes an *identifiability limit*. A controller that plans only to reach the goal never brackets the true boundary, so its rollouts cannot supply the boundary-spanning negatives the selector needs, and the label-free keep-out is conservative by *information*, not tuning. Three label-free remedies (loose enforcement, MPC-generated negatives, a safe-sparsity prior) each fail to close the on-support gap. Only an external collision probe recovers a tight boundary. A constraint learner that depends on policy exploration cannot be handed a non-exploring optimal controller without losing coverage, which is why the queued direction is to make the MPC itself the explorer.

What works. Two design decisions drive the results. Jerk as the control input turns position, velocity and acceleration into MPC states, so every order becomes one algebraic inequality row with no solver change. The hard NLP row for ζ_θ forces the solver to satisfy the learned constraint at every horizon step, not in expectation. Together they give zero violations of whatever constraint the solver is given (all four geometries, all 8 seeds). The remaining safety gap is a learning problem: false negatives in ζ_θ cause residual true-set violations open-loop (26 to 47% of steps) that, with recall already ~ 1.0 from the first iteration, do not shrink over iterations. Closing it is the closed-loop result of Section 6.7.

What limits performance. Five gaps remain, with the deployment path forward in Section 7. **(i) Solver speed.** IPOPT (132–197 ms) misses a real-time budget on multi-order environments. The acados SQP-RTI backend is measured at 3.1 ms on the position NLP ($57\times$ faster than IPOPT, inside the 100 ms step), but the higher-order QP does not yet converge (Table 17). **(ii) Closed loop under live physics.** The loop is closed on the learned ζ_θ in simulation with zero true violations (Section 6.7), but not yet under live PyBullet contact physics on the arm. **(iii) Cartesian planning.** Its end-effector-only rows let IK route a link through the obstacle on tight scenes, fixed in principle by joint-space MPC with per-link forward-kinematics rows [12]. **(iv) Constraint coverage.** The 29–30 demonstrations bracket only the demonstrated region. **(v) Deployment scope.** ζ_θ is trained on the point-mass abstraction and shown deployed in the iiwa-14 *planner* (Section 6.6), where it plans a collision-free path the arm follows by IK. End-to-end deployment into the KUKA *closed loop* under live contact physics is the natural next step.

6.12 Comparison: PUCL-MPC vs. behaviour cloning [H1]

This is the H1 *policy* evidence: with the MPC as PUCL’s forward policy, the explicit learned constraint beats a constraint-free behaviour-cloning baseline on a generalisation test. We compare three policies on held-out tasks: from each of 30 held-out (start, goal) pairs we roll out *Oracle MPC* (analytic constraint, ceiling), *PUCL-MPC* (learned ζ_θ as a hard NLP row, this work), and *behaviour cloning* (BC), an imitation policy that clones the expert velocity field with *no* constraint mechanism. Both the start and the goal are drawn fresh, held out from the demonstrations, so a policy that merely memorises the demonstrated corridor cannot pass. The quantity isolated here is the value of the *explicit constraint*, not of constraint learning: BC is an external policy, not a replacement for PUCL’s reliable-negative mining. BC is trained by supervised regression on 80 oracle demonstrations whose starts and goals span the same distribution, a generous superset of the ~ 30 PUCL uses, so it is not under-resourced, and it is the same velocity-level policy class as the dynamical-systems block in Peng & Billard [1], minus any constraint. Results are mean $\pm$ SD over five seeds ($5 \times 30 = 150$ tasks). Every value is measured against the true analytic boundary, and none is hand-set.

Table 10: Generalisation comparison on `ellipsoid3d` (position) and `ellipsoid3dvel` (position + velocity cap), held-out start *and* goal, five seeds $\times$ 30 tasks. Violation steps are measured against the true analytic boundary (mean $\pm$ SD over seeds). Reach is the median end-of-episode distance to goal. Success = reach below 5 cm with zero true violations (mean over seeds, Wilson 95 % CI on the 150 pooled tasks).

Env.	Method	Viol.	Reach (mm)	Success [95 % CI]
ellipsoid3d	Oracle MPC	0.0	0.9	100% [0.98, 1.0]
	PUCL-MPC (ours)	0.2 ± 0.4	41.7	54% [0.46, 0.62]
	BC	16.3 ± 2.1	136.2	2% [0.01, 0.06]
ellipsoid3dvel	Oracle MPC	0.0	0.6	100% [0.98, 1.0]
	PUCL-MPC (ours)	0.0	0.9	98% [0.94, 0.99]
	BC	14.7 ± 1.6	31.7	1% [0.00, 0.05]

H1 holds on both scenes. On the position scene (`ellipsoid3d`) PUCL-MPC stays essentially as safe as the oracle (0.2 ± 0.4 true violation steps: the enforced learned row is satisfied by construction, and the small residual is where the learned keep-out’s recall falls just short of 1), whereas BC, with no constraint, cuts the corner on almost every held-out task (16.3 ± 2.1 violation steps, 2% success). The price PUCL pays for that safety is conservatism: its larger label-free keep-out (near-support IoU 0.05–0.23, Table 14) blocks the goal on a sizeable minority of held-out goals, so its success (54%) sits below the oracle’s 100%. This is a reachability cost, not a safety failure, and it is exactly what the active-refinement probe targets (Section 6.10). PUCL never trades safety for reach.

This also explains why an earlier, easier protocol made the two look alike. With a fixed in-distribution goal and starts close to the demonstrations, a behaviour-cloning policy can memorise the single safe corridor and nearly match PUCL on the position scene, and the gap only opens once the goal is held out and BC must generalise with no constraint to fall back on. We therefore report the held-out start-and-goal protocol.

The velocity scene (`ellipsoid3dvel`) separates the methods completely (Fig. 10) and shows the H5 benefit. PUCL-MPC records zero velocity-cap violations by construction (the row $\|v_k\|_2 \leq v_{\max}$ is an algebraic state bound) and matches the oracle (0.0 violations, 98% success). BC has no mechanism to bound velocity magnitude, exceeds the cap on 14.7 ± 1.6 steps per episode and essentially never succeeds (1%): a velocity-norm bound cannot be expressed by a policy that only reproduces a velocity *direction* field. An explicit hard constraint does something an imitator structurally cannot.

6.13 Limitations and threats to validity

Each item is a scope statement, not a defect.

- **Open-loop, not receding-horizon.** Every headline result is a single horizon solve replayed without replanning. Receding-horizon control appears only as the sanity check of Section 6.7, with the analytic constraint and the learned ζ_θ alike.

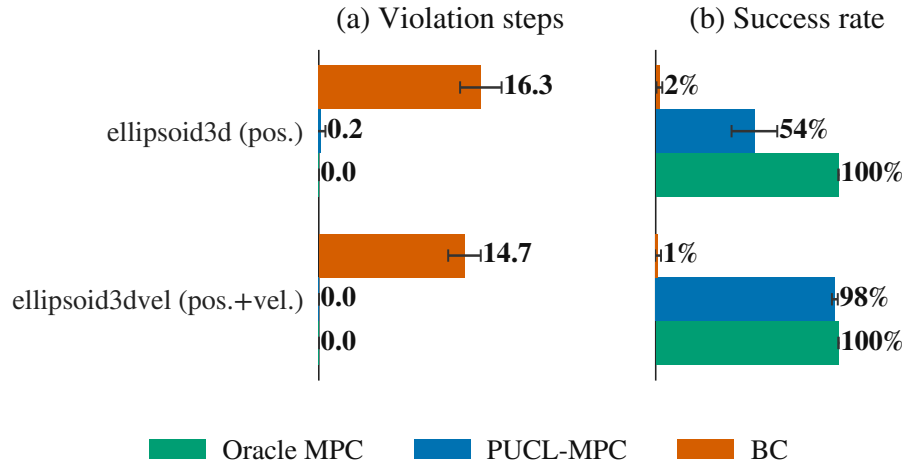


Figure 10: Oracle MPC vs. PUCCL-MPC vs. behaviour cloning (BC) on held-out start *and* goal (five seeds $\times$ 30 tasks, mean $\pm$ SD over seeds). (a) Constraint-violation steps per episode: BC, with no constraint mechanism, crosses the true boundary on 14–16 steps per episode on both scenes, while PUCCL-MPC stays at the oracle’s near-zero. (b) Task success (reach within 5 cm and zero violations): PUCCL-MPC matches the oracle on the velocity scene and stays safe but conservative on position, while BC essentially never succeeds. All values are measured against the true analytic boundary, whereas **BC violates 14–16 steps per episode against PUCCL-MPC’s ≤ 0.2 .**

- **Single-integrator, not triple.** The deployed MPC reads only position and velocity ($p^+ = p + v dt$). The triple-integrator jerk-as-control formulation (Section 4) produced no reported four-geometry result.
- **Acceleration and jerk are learned but not enforced.** Their rows are absent from every reported solve, so where the executed policy exceeds those caps the overshoot is large (Section 6.1).
- **Zero enforced-row violations is partly tautological.** A feasible solve cannot violate the enforced rows by construction, yet the open-loop policy still crosses the true unsafe set on 25 to 47% of steps. The honest safety claim is the closed-loop, robust-tightening result of Section 6.7.
- **acados is measured only on the position case.** SQP-RTI is 3.1 ms versus 177 ms IPOPT on the position NLP (57 $\times$), but the higher-order QP does not yet converge and the controller ships falling back to IPOPT, so every reported solve time is IPOPT (Tables 8 and 9). The higher-order single-digit-ms figure is a target.
- **Closed-loop scope.** The receding-horizon sweep drives true violations to 0 with both the analytic constraint and the learned ζ_θ (Section 6.7). The conservative learned keep-out costs goal-reach on the position-only scene. Closing the loop under live contact physics on the arm remains future work.
- **Two cap regimes.** The four-geometry result uses $a_{\max}=2.1$, $j_{\max}=42$. The ablation and acados table use the legacy $a_{\max}=1.20$, $j_{\max}=6.00$. The solver-cost conclusion is cap-independent, but the higher-order recovery is not, so the easy 1.20/6.00 recovery is not the canonical-cap result.
- **Recall is class-imbalance inflated at higher orders.** The velocity / acceleration / jerk recall ~ 0.99 –1.0 is on a one-sided norm grid that is ~ 90 –97% unsafe. The honest higher-order tightness is markedly weaker at the canonical cap, where acceleration is the weakest of the four orders (Section 6.4).
- **PUCCL-MPC is real but under-exploring.** The MPC genuinely serves as PUCCL’s forward policy (Section 6.9), but being deterministic and non-exploring it cannot supply the boundary-spanning rollouts reliable-negative mining needs, so the label-free boundary stays conservative. This is the central honest finding, not an incidental flaw.

7 Conclusion and Future Work

What was achieved. An MPC is a viable forward policy for PUCL. Driven by it, two-step reliable-negative selection reproduces the mechanism of Peng and Billard [1]: it recovers the unsafe set at recall ≈ 0.99 and beats a binary cross-entropy classifier (BCE) ablation of the positive-unlabeled step. The same pipeline learns obstacle-avoidance constraints under four obstacle geometries (single ellipsoid, two ellipsoids, concave, two regions) and four orders (position, velocity, acceleration, jerk), each from its own demonstrations. The isolating evidence is jerk: the BCE baseline collapses on the non-convex geometries (recall 0.13 concave, 0.00 two regions) while PUCL holds ≈ 0.99 . As a policy, the explicit learned constraint also generalises far more safely than a constraint-free behaviour-cloning (BC) policy on held-out starts and goals (≤ 0.2 versus 14–16 true-collision steps per episode). Two caveats apply. First, velocity and jerk recall are class-imbalance inflated on the one-sided norm grid ($\sim 97\%$ unsafe), so recall, not IoU, carries the claim. Second, acceleration is the weakest order, recovering its canonical cap only partially on a boundary-balanced sample.

The honest limitation. As a hard NLP row, ζ_θ gives a feasible solve zero violations of the enforced position and velocity rows, but that is a property of the solver, not the true set. The deterministic MPC under-explores and never brackets the true boundary, so the label-free keep-out stays conservative (near-support IoU 0.05–0.23), and the executed policy still crosses the true unsafe set on 25–47% of steps, because the enforced hard wall sits on a conservative, over-grown boundary. A minimal binary collision probe, extending PUCL’s label-free core rather than replacing it, restores a tight keep-out (global IoU 0.015–0.049 up to 0.60–0.90 at recall ~ 1.0). Driving true collisions to ≈ 0 steps per rollout is shown as a receding-horizon closed-loop check, with the analytic oracle and the learned ζ_θ alike.

Real-time control. On the position NLP, warm-started acados SQP-RTI is measured at a median 3.1 ms against IPOPT at 177 ms on the same machine: a $57\times$ reduction, inside the 100 ms control step (Table 17). Scope is the position case only: the deployed controller still uses IPOPT for every reported solve, and the higher-order (velocity/acceleration/jerk) QP does not yet converge.

Future work. Close the loop with the learned constraint: the closed-loop result here uses the oracle, so re-solving every step against ζ_θ is next. Move from the Cartesian end-effector planner to joint-space MPC with per-link forward kinematics, so the whole arm body is safe, not only the end-effector. Hard-enforce the learned acceleration and jerk constraints on the triple-integrator state, so jerk-as-control becomes the deployed configuration. Recover the tight keep-out without labels by making the forward policy explore, through an information-seeking cost term or a separate exploratory sampler feeding the reliable-negative selector.

A Appendix

A.1 Full hyperparameters

Tables 11 and 12 list every parameter used in the experiments. The four-geometry learning result uses $a_{\max} = 2.1$, $j_{\max} = 42$, and horizon $N = 50$ (the 45–60 in the table is the solver-suite range). All four geometries share the same weights, and no per-environment tuning is applied.

Three thresholds appear in this work, each in a different role. The decision threshold $\tau = 0.5$ scores constraint quality (recall, precision, IoU). The reliable-negative selection threshold $\tau_p = 0.3$ governs which unlabelled states are mined as reliable negatives. The hard-row injection bound caps ζ_θ at 0.15 (position) and 0.2 (velocity) inside the deployed solver. The kNN reliable-negative mining is the single component separating PUCL from a naive binary classifier, and the reason the jerk constraint is recovered on the non-convex geometries where the classifier collapses (Appendix A.4).

A.2 MPC-solver hyperparameter search

The full MPC-solver grid search behind the body summary (Section 5.5) follows. This section reports the tuning of the *MPC solver* hyperparameters, run on a single representative solver environment with the CasADi/IPOPT backend. The kinematic-cap rows it sweeps ($v_{\max}$, $a_{\max}$, $j_{\max}$) are the *solver/actuator specification of that representative environment*. They use that environment’s own cap values ($a_{\max} = 1.2$, $j_{\max} = 6.0$, swept over 0.6–2.0 and 2–15), not the four-geometry constraint-learning caps ($v_{\max} = 0.60$,

Table 11: MPC planner hyperparameters (CasADi/IPOPT backend). The deployed planner uses $a_{\max}=1.2$, $j_{\max}=6.0$. The four-geometry learning environment uses $a_{\max}=2.1$, $j_{\max}=42$.

Parameter	Description	Value
Δt	Planning timestep	0.10 s
N	Horizon length (per env)	45–60
w_g	Terminal goal weight	70.0
w_t	Stage tracking weight	0.15
w_j	Stage jerk weight	1×10^{-3}
w_a	Stage acceleration weight	1.8×10^{-3}
m	Ellipsoid keep-out margin	0.05
$v_{\max}$	Velocity cap (norm, traj. units)	0.6
$a_{\max}$	Acceleration cap (deployed planner)	1.2
$j_{\max}$	Jerk cap, NLP safety net (deployed planner)	6.0
<code>tol</code>	IPOPT convergence tolerance	10^{-6}
<code>max_iter</code>	IPOPT max iterations	400
<code>acc_tol</code>	IPOPT acceptable tolerance	10^{-4}
s	Workspace scale (traj $\rightarrow$ m)	0.20 m/unit
Workspace origin (world frame)		(0.50, 0.00, 0.55) m

$a_{\max} = 2.1$, $j_{\max} = 42$) used for the headline learning result in the body. The purpose here is to show that the solver weights and thresholds sit in a flat, non-fragile basin, not to set the learning caps. Likewise the reach-minimising horizon on this solver environment is $N = 55$, whereas the four-geometry headline result deploys $N = 50$ (Section 6.8). Both lie in the flat basin of Figure 12. Every solver hyperparameter is fixed by a search, not guessed: a grid search plus a joint optimisation for the four cost weights. Table 13 lists all 15 MPC and constraint-injection hyperparameters with their deployed value, the range swept, and how that value was selected. The load-bearing searches are the filled contours in Figure 11 (six two-parameter sweeps) and the three one-parameter trade-offs in Figure 12. The keep-out margin m and the soft-penalty weight w_ζ were also swept but are non-load-bearing (flat sub-mm reach across the range, and inert under the active hard constraint, respectively), so the table reports them rather than plotting them. Each plotted cell is a real MPC re-solve ($\approx 10^3$ solves, `kuka_mpc.build_tuning`). The learned constraint ζ_θ is adopted unchanged from [1]. Its training settings are that paper’s defaults and are not re-tuned here (the constraint-learner knobs and their full per-order results are reported in Appendix A.4). The one learner knob that paper flags as most sensitive, the kNN distance threshold d_r (our κ), is verified rather than re-tuned below (Fig. 13).

Selection objective. The objective is task *success*: reach within 5 cm, zero constraint violations, end-effector outside the keep-out set. The four cost/stage weights (w_g, w_j, w_t, w_a) are the main hyperparameters that shape the trajectory, and are fixed by a *joint* 400-sample random search on the tuning environment (`kuka_mpc.joint_optimize`) that minimises mean reach error while meeting the success criterion. They are searched together, not one at a time, because the best value of one weight depends on the others. The two cost/stage-weight contour panels in Figure 11 ($w_g \times w_j$ and $w_t \times w_a$) are two-parameter slices through this optimum that show the local sensitivity. The selection itself is the joint search. The winner is re-evaluated on disjoint held-out seeds (train reach 0.017 mm vs. validation 0.018 mm, so no seed-overfitting) and verified to generalise (0 violations and strictly lower reach) across all three result environments. The adopted weights are better than the earlier hand-tuned values on both reach and smoothness. The per-panel reach-minimising corner is deliberately not adopted. Minimising reach alone drives the soft jerk weight to its floor and yields jerkier motion (jerk r.m.s. 5.6 vs. 4.8 on `ellipsoid3d`), and the reach it buys back is sub-millimetre, far below the 5 cm bar. The deployed weights instead give the lowest jerk r.m.s. on every environment. The remaining hyperparameters follow their own criteria. The caps $v_{\max}, a_{\max}, j_{\max}$ are this solver environment’s actuator specification (the grid only confirms

Table 12: PUCL constraint-learning hyperparameters, matching the deployed checkpoint. One recipe trains all four orders on every geometry, and no per-geometry tuning is applied.

Parameter	Description	Value
k	kNN neighbours for reliable-negative mining	1
κ	kNN distance threshold (-kNNt)	0.05
n_{demo}	Expert demonstrations (positive set $ \mathcal{D}^+ $)	29–30
T_{iter}	PUCL outer iterations	8
n_{seed}	Random seeds per geometry	5
Architecture	Hidden layers, position net (dims 0:3)	[32, 32]
Architecture	Hidden layers, velocity net ($\ v\ _2$, dims 6:9)	[4]
Architecture	Hidden layers, acceleration net ($\ a\ _2$, dims 9:12)	[4]
Architecture	Hidden layers, jerk net ($\ j\ _2$, dims 12:15)	[4]
Loss	Cross-entropy, $\mathcal{D}^+ \rightarrow 0$, $\mathcal{D}_{\text{RN}}^- \rightarrow 1$	BCE
Optimiser	Adam, learning rate	5×10^{-3}
Inner steps	Gradient steps per outer iteration	500
Batch size	Constraint-net minibatch	256
λ	L2 weight regularisation	0
τ	Decision threshold, constraint-quality scoring	0.5

feasibility). The horizon $N=55$ is the reach-minimising value. The position threshold τ_p trades obstacle clearance against goal reachability. The keep-out margin $m = 0.05$ is a conservative clearance choice (reach stays sub-millimetre across the whole 0–0.20 sweep). Finally, τ_v , w_ζ and the IPOPT tolerances have no measurable effect. The right-hand column of Table 13 gives the reason for each hyperparameter.

Three patterns appear across the searches. **(i)** Sharp feasibility limits: reach jumps from < 1 mm to several hundred mm once v_{max} drops below 0.5, and only returns to sub-millimetre once the time window $\Delta t \cdot N$ exceeds the ≈ 5 s needed to cross the workspace. **(ii)** Clear separation of roles: obstacle clearance is set entirely by τ_p and does not change with τ_v . **(iii)** Most other settings barely matter: the solver tolerances, the stage weights, w_ζ and τ_v all leave the results flat, so the deployed values sit in a safe basin rather than on a fragile edge.

Learner sensitivity to the distance threshold d_r . The constraint learner is adopted unchanged from Peng & Billard [1], who identify the kNN reliable-negative distance threshold d_r (our κ) as PUCL’s most sensitive hyperparameter. We do not re-tune it, but we verify that the adopted $\kappa = 0.05$ sits in the safe regime. Figure 13 sweeps d_r on `ellipsoid3dposvel` (2–3 seeds) in the dual-axis form of their analysis: the deployed $\kappa = 0.05$ holds the deployed unsafe rate at $\approx 2\%$, whereas larger d_r drives it sharply up and erratic (to $\approx 88\%$ at $d_r=0.3$). The IoU over this range is noisy and non-monotonic, so the unsafe rate is the load-bearing signal here. This reproduces the strong d_r dependence they report and confirms the adopted value is safe. A full five-seed sweep is left to future work.

A.3 Grid-based constraint-quality formulas

The constraint-quality scores in the body (Table 6) are computed per order on a grid against the analytic cap, not over executed trajectories. For each order $k \in \{\text{pos, vel, acc, jerk}\}$, let $\mathcal{C}_k^*$ be the analytic unsafe set and $\hat{\mathcal{C}}_k = \{x : \zeta_\theta^k(x) \geq \tau\}$ the learned unsafe set, both sampled on a uniform grid $\mathcal{G}_k$ of $|\mathcal{G}_k|$ points. The grid-point counts $\text{TP}_k = |\hat{\mathcal{C}}_k \cap \mathcal{C}_k^*|$, $\text{FP}_k = |\hat{\mathcal{C}}_k \setminus \mathcal{C}_k^*|$, $\text{FN}_k = |\mathcal{C}_k^* \setminus \hat{\mathcal{C}}_k|$, and $\text{TN}_k = |\mathcal{G}_k| - \text{TP}_k - \text{FP}_k - \text{FN}_k$

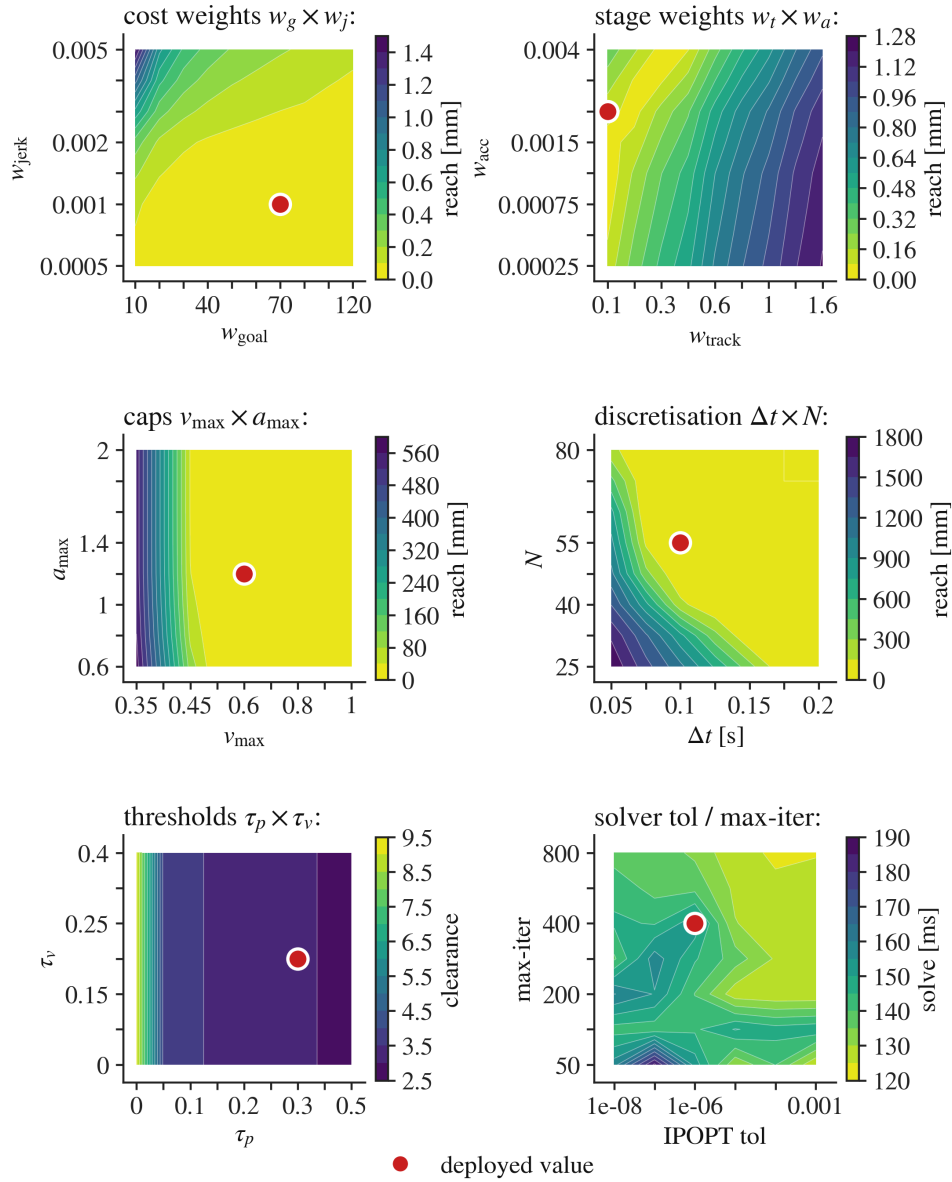


Figure 11: Two-parameter grid searches (five seeds per cell, every cell a real CasADi/IPOPT solve): all six sweeps on one page as filled contours, with the white-edged red dot at the deployed value. Row by row, left then right: cost weights $w_g \times w_j$ and stage weights $w_t \times w_a$ (reach error [mm]). Kinematic caps $v_{\text{max}} \times a_{\text{max}}$ and discretisation $\Delta t \times N$ (reach error [mm]). Injection thresholds $\tau_p \times \tau_v$ (minimum clearance) and IPOPT $\text{tol} \times \text{max_iter}$ (solve time [ms]). Reach stays below 1 mm across a wide basin around every deployed value. Obstacle clearance is set entirely by τ_p .

Table 13: Master *MPC-solver* hyperparameter table: every solver and constraint-injection hyperparameter, its deployed value on the representative solver environment (CasADi/IPOPT backend), the range swept, and the method by which the value is justified. The four cost/stage weights are fixed by a *joint* optimisation (held-out validated, §5.5). The remaining solver hyperparameters are fixed by the grid search (Figs. 11, 12). The kinematic-cap rows ($v_{\max}$, $a_{\max}$, $j_{\max}$) are the *solver/actuator specification of this representative environment* ($a_{\max} = 1.2$, $j_{\max} = 6.0$) and the grid only *validates feasibility*. They are not the four-geometry learning caps ($a_{\max} = 2.1$, $j_{\max} = 42$) of the body result. The timestep and solver tolerances are standard defaults the grid shows to be non-critical.

Symbol	Meaning	Value	Range	Selection method / justification
w_g	terminal goal weight	70	10–120	joint optimum (reach + smoothness), held-out validated, generalises
w_j	stage jerk weight	$1 \cdot 10^{-3}$	$5 \cdot 10^{-4}$ – $5 \cdot 10^{-3}$	joint optimum, smoothest motion at fixed reach
w_t	stage tracking weight	0.15	0.1–1.6	joint optimum, held-out validated
w_a	stage accel. weight	$1.8 \cdot 10^{-3}$	$2.5 \cdot 10^{-4}$ – $4 \cdot 10^{-3}$	joint optimum
$v_{\max}$	velocity cap	0.6	0.35–1.0	enforced task spec, grid validates feasibility
$a_{\max}$	acceleration cap	1.2	0.6–2.0	enforced task spec
$j_{\max}$	jerk cap	6	2–15	enforced task spec
Δt	planner timestep	0.10 s	0.05–0.20	standard discretisation, non-critical past the cliff
N	planning horizon	55	25–80	solver-env reach-minimising horizon (four-geometry result deploys $N=50$)
m	keep-out margin	0.05	0–0.20	swept, reach sub-mm throughout, so fixed conservatively for clearance
τ_p	position threshold	0.3	0–0.5	grid search: balances clearance against goal reachability
τ_v	velocity threshold	0.2	0–0.4	grid search: clearance-invariant, PUCL default
w_ζ	soft-penalty weight	1.0	0.1–10	swept, inert (hard constraint already active), no effect
tol	IPOPT tolerance	10^{-6}	10^{-8} – 10^{-3}	solver default, no solve-time trend
max_iter	IPOPT iter cap	400	50–800	solver default, no solve-time trend

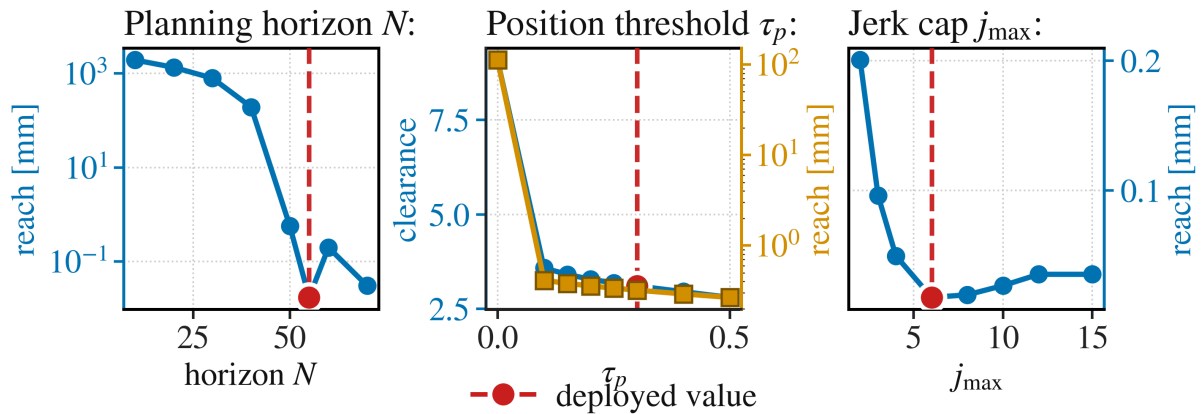


Figure 12: One-parameter trade-offs, each cell a real five-seed CasADi/IPOPT re-solve. Left to right: planning horizon N (single axis, reach error [mm], log scale), position threshold τ_p (dual axis: blue clearance, orange reach [mm] on log scale), and jerk cap $j_{\max}$ (single axis, reach error [mm]). The primary trade-off curve is blue. The reach second axis is orange. The white-edged red dot and dashed line mark the solver-study selected value on the curve. Reach falls below 1 mm at $N=55$ on this solver environment (the four-geometry headline result deploys $N=50$, Section 6.8) and is flat and sub-millimetre for any jerk cap, so $j_{\max}$ is set by the task spec, not minimised here.

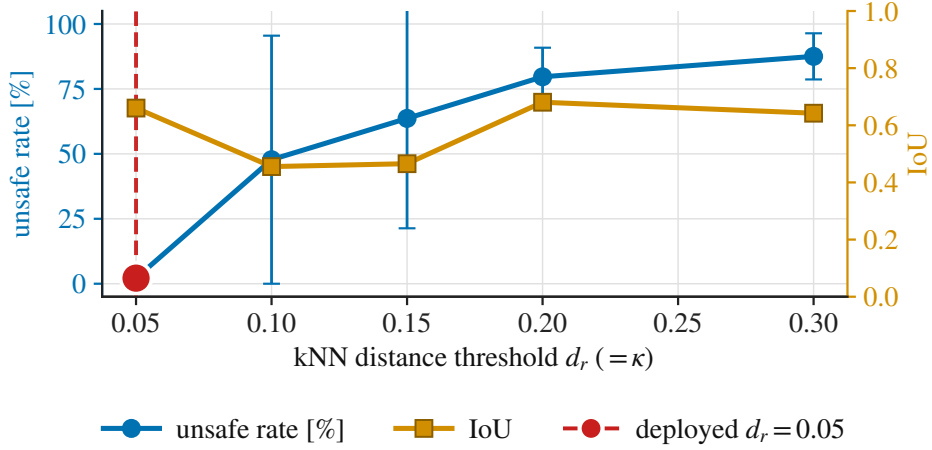


Figure 13: Sensitivity of PUCL to the kNN distance threshold d_r (our κ), in the dual-axis form of Peng & Billard [1]: mean deployed unsafe rate (blue, left, lower better, error bars one SD over 2-3 seeds) and IoU (orange, right, higher better) versus d_r on `ellipsoid3dposvel`, with the red marker the deployed $\kappa = 0.05$, which holds the unsafe rate near 2% while larger d_r drives it to $\approx 88\%$, reproducing the strong d_r dependence Peng & Billard report.

define the four per-order scores

$$\text{Recall}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FN}_k}, \quad \text{Precision}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k}, \quad (\text{A.3.1})$$

$$\text{IoU}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k + \text{FN}_k}, \quad \text{Acc}_k = \frac{\text{TP}_k + \text{TN}_k}{|\mathcal{G}_k|}. \quad (\text{A.3.2})$$

Per-component grids. Each network is evaluated against the truth it was trained to recover, on a grid that varies only the variables it sees. The position network $\zeta_\theta^p(p)$ uses a 3-D position grid $[-0.7, 0.7] \times [-0.6, 0.6] \times [-0.5, 0.5]$ at 35^3 points. The velocity scalar network $\zeta_\theta^v(\|v\|)$ uses $[-1.5, 1.5]^3$ at 30^3 points. The acceleration and jerk scalar networks use $[-2a_{\max}, 2a_{\max}]^3$ and $[-2j_{\max}, 2j_{\max}]^3$ at 30^3 points. All other state components are held at zero, so each network sees only the variables that drive its truth.

Class balance. The analytically unsafe fraction varies sharply across grids. The position grid is mostly safe (the ellipsoid covers a small fraction). The velocity grid is mostly unsafe, since a ball of radius $v_{\max} = 0.6$ has volume $\frac{4}{3}\pi v_{\max}^3 \approx 0.91$ and the unsafe set covers $1 - 0.91/3^3 \approx 96.6\%$ of the velocity cube. A constant-unsafe predictor therefore scores $(\text{Recall}, \text{Precision}, \text{IoU}) \approx (1.00, 0.97, 0.97)$ on the velocity grid by class imbalance alone, and ≈ 0 on the position grid. The four constraints occupy different units (m, m/s, m/s², m/s³) and are separable, so each is scored against its own truth rather than combined into a single IoU.

A.4 Full per-order constraint-recovery results

The headline recall numbers, in full per-order, per-geometry form, span four geometries (single ellipsoid, two ellipsoids, concave, two regions), five seeds, and eight PUCL iterations. Recall is the primary indicator: of the cells truly inside the keep-out set on a deterministic norm grid, the fraction the learned constraint flags. The final-iteration per-order recall is Table 6: the minimum norm-grid recall for position, velocity and jerk is 0.99, and the boundary-balanced acceleration recovery at the canonical cap ($a_{\max}=2.1$, $j_{\max}=42$) is markedly weaker: acceleration is the weakest of the four orders (Section 6.4).

The velocity, acceleration, and jerk constraints are scalar norm caps on a deterministic one-sided norm grid, so their per-order recall and IoU are seed-independent (standard deviation ≈ 0). Their IoU reads near 1.0, but this is a class-imbalance artefact: the grid is roughly 97% outside the cap, so a single threshold scores high IoU without being informative. Recall, not IoU, carries the claim for these orders. The geometry-meaningful tightness is the position keep-out IoU in Table 14.

Table 14: Near-support position keep-out IoU of ζ_θ per geometry (final iteration), scored in an obstacle-tight evaluation box (0.05–0.23). The learned set is conservative by design: it grows to cover the true keep-out, trading precision for the recall in Table 6.

Geometry	Near-support position IoU
single ellipsoid	0.10
two ellipsoids	0.11
concave	0.05
two regions	0.23

Safe sparsity prior on the position keep-out (partial). The body (Section 6) shows that the position keep-out’s low global IoU is off-support extrapolation, not a loose fit on the operating support. A safe sparsity prior, uniform-workspace samples labelled safe through a coefficient in the constraint-net loss, targets that extrapolation directly. On the concave geometry at convergence (PUCL iteration 8, seed 0) the prior cuts the share of the $z=0$ slice flagged unsafe from 93% to 66% and raises the global IoU from 0.016 to 0.039 ($2.4\times$), while recall is preserved (0.993 to 0.999) and the on-support reading is essentially unchanged ($R=0.55$ IoU 0.151 to 0.139): the gain is concentrated off-support where the over-cover sat. The pattern holds across all four geometries (Table 15): the prior lifts the global position IoU by 2.6 to $3.3\times$ and cuts the far-space false-alarm rate by roughly 70%, while recall stays high (0.942–1.000). The runs are single-seed, so the prior is a diagnostic that validates the off-support account, not a headline metric.

Geometry	Global IoU	Recall	FPR
	base $\rightarrow$ prior	base $\rightarrow$ prior	base $\rightarrow$ prior
single ellipsoid	0.018 $\rightarrow$ 0.059	0.996 $\rightarrow$ 0.976	0.85 $\rightarrow$ 0.24
two ellipsoids	0.024 $\rightarrow$ 0.073	1.000 $\rightarrow$ 0.963	0.76 $\rightarrow$ 0.23
concave	0.015 $\rightarrow$ 0.039	0.997 $\rightarrow$ 1.000	0.64 $\rightarrow$ 0.24
two regions	0.049 $\rightarrow$ 0.165	1.000 $\rightarrow$ 0.942	0.78 $\rightarrow$ 0.19

Table 15: Safe sparsity prior (coefficient 0.2) on the learned position keep-out at convergence (PUCL iteration 8, seed 0), per geometry. The prior lifts the global position IoU 2.6 to $3.3\times$ and cuts the far-space false-alarm rate (FPR) by roughly 70% while recall stays high (0.942 to 1.000): the off-support over-cover is removed without sacrificing coverage of the true keep-out.

PUCL versus the binary cross-entropy classifier baseline. Table 16 contrasts PUCL against a binary cross-entropy classifier (BCE): the same four networks trained with direct cross-entropy and no reliable-negative mining. This is a constraint-learning ablation of the PU step, not the constraint-free behaviour-cloning policy (BC) of Section 6.12. Position recall is high for both everywhere (≥ 0.91). The separation is in the highest order on the non-convex geometries: on concave and two regions the classifier’s jerk recall collapses to 0.13 and 0.00 while PUCL holds ≈ 0.99 . Without reliable-negative mining the classifier cannot separate the noisy jerk signal on the hard geometries and flags nothing. The two-step PU selection recovers it. On two regions all five BCE seeds produced a bit-identical jerk trajectory, so the 0.00 entry is a single deterministic realisation (effective $n = 1$): a stability and determinism gap, not five-sample dispersion. This collapse, not position learning, drives the combined-recall gap (PUCL ≈ 0.99 vs. BCE 0.73/0.75 on concave/two regions).

The analytic caps $v_{\max} = 0.60$, $a_{\max} = 2.1$, $j_{\max} = 42$ are environment-design choices on synthetic finite-difference kinematics, not physical ground truth. Recall is measured against these caps.

A.5 Expert demonstration tool

Expert demonstrations for the positive set $\mathcal{D}^+$ are collected with a GUI that wraps the MPC solver run against the analytic constraint ζ_{true} . It exposes per-environment parameters: number of demonstrations, start/goal sweep range, jerk weight w_j (smoothness), and random seed. The output feeds directly into

Table 16: Per-order recall, PUCL|BCE, final iteration (five seeds). BCE (binary cross-entropy classifier, the PU-step ablation) jerk recall collapses to 0.00 on two regions (single deterministic run, effective $n=1$) while PUCL holds 0.99.

Geometry	pos		vel		acc		jerk	
single ellipsoid	0.99	0.91	1.00	1.00	1.00	1.00	0.99	0.98
two ellipsoids	0.99	0.95	1.00	1.00	1.00	0.99	1.00	0.98
concave	1.00	0.98	1.00	0.81	1.00	1.00	0.99	0.13
two regions	1.00	0.98	1.00	1.00	1.00	1.00	0.99	0.00

the kNN mining step (Algorithm 1 in [1]). Trajectory diversity is measured by mean pairwise Hausdorff distance over the collected set. All reported experiments use 29–30 demonstrations per environment. The caps are sized just above the demonstration envelope at generation time, so every demonstration respects every analytic kinodynamic cap on all steps, making the set a valid positive (safe) set.

A.6 Implementation notes

Solver flags. IPOPT runs with `print_level 0`, `sb yes`, `max_iter 400`, and `acceptable_tol  $10^{-4}$` . The last lets it stop early on an acceptable solution, keeping the multi-order scene from timing out on some seeds.

Warm-start strategy. Each planning call initialises P from a straight-line interpolation between p_0 and g , bent by a $\sin(\pi\lambda)$ bump (amplitude 0.45, perpendicular to the line) if any obstacle is present. A 0.01σ Gaussian noise term (seeded per call) breaks ties between near-equivalent warm starts. No previous solution is reused (open-loop planning).

Numerical conditioning. The NLP adds a small numerical slack (10^{-3}) to the keep-out constraint, while the violation checker uses the nominal margin $m = 0.05$. This inner margin keeps the interior-point barrier off the constraint boundary, where the Hessian would be poorly conditioned.

acados RTI backend. The acados backend uses SQP real-time iteration (SQP-RTI) with partial condensing and the HPIPM QP solver, on the same constraint rows as the CasADi/IPOPT solver. One SQP-RTI step per call gives a bounded per-call cost on the position NLP. A committed benchmark log now measures this directly: warm-started SQP-RTI on the position NLP runs at a median 3.1 ms (range 2.8–4.5 ms over 50 calls, all status 0) against IPOPT at 177 ms (8 seeds, same machine), a $57\times$ speed-up well inside the 100 ms control step (Table 17). One-time C-code compilation at start-up takes ~ 1.5 s. Scope limits remain: the higher-order (velocity/acceleration/jerk) QP does not yet converge, so every result in this thesis uses CasADi/IPOPT for the deployed solves, and the measured speed-up covers the position case only.

Table 17: acados SQP-RTI is the measured real-time route on the position NLP. The two backends solve the identical CasADi OCP. IPOPT iterates to a certified optimum and is kept as the exact validation reference, but its measured solve time (177 ms on the position NLP, 8 seeds) overruns the 100 ms control step. acados SQP-RTI takes one real-time iteration per call, so its per-call cost is bounded and independent of the interior-point iteration count: a committed benchmark log records a median 3.1 ms (range 2.8–4.5 ms over 50 calls, all status 0), a $57\times$ speed-up well inside the control step. Two scope limits: acados does not yet converge once velocity or higher orders are added (HPIPM NaN/QP), so the deployed solves and every reported result use IPOPT, and the build path must be space-free.

Position NLP	CasADi/IPOPT	acados SQP-RTI
Per-call cost model	iterate to convergence	one RTI step (bounded)
Median solve time	177 ms (8 seeds)	3.1 ms (50 calls)
Fits the 100 ms step?	no	yes ($57\times$)

Note on path setup. The acados shared libraries must be loadable. If the project path contains spaces (for example, iCloud Drive), link it to a space-free path and point acados at the link:

```
ln -s $ACADOS_ROOT ~/acados_root
```

```
export ACADOS_SOURCE_DIR=~/.acados_root
```

The bundled C library and the Python template install (<https://docs.acados.org/installation/>) complete the setup.

A.7 Relation to standard MPC stability theory

Formal stability and recursive-feasibility results for MPC require terminal ingredients added to the OCP: a terminal/invariant set, a terminal cost, and, for disturbances, a robust formulation such as a tube around a nominal trajectory. These are standard [5, 4]. This thesis does *not* add them. It uses a plain terminal goal cost, enforces the constraints as hard rows, and rejects disturbances reactively by re-solving from the measured state (Section 6.7). It therefore makes no formal stability or recursive-feasibility claim: the disturbance recovery there is an empirical observation on the triple-integrator plant, not a guarantee. Adding the standard ingredients (a terminal set nested in the constraints, a value-function Lyapunov argument, or a robust tube) is left as future work.

References

- [1] Baiyu Peng and Aude Billard. Learning constraints from demonstrations with grid cells. *IEEE Robotics and Automation Letters*, 9(4):3648–3655, 2024. RA-L 2024.
- [2] Adam Stooke, Joshua Achiam, and Pieter Abbeel. Responsive safety in reinforcement learning by PID lagrangian methods. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 9133–9143, 2020.
- [3] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. CasADi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.
- [4] Colin N. Jones. ME-425 model predictive control: Lecture notes, 2024. Ecole Polytechnique Fédérale de Lausanne.
- [5] David Q. Mayne, James B. Rawlings, Christopher V. Rao, and Pierre O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.
- [6] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.
- [7] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados: a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14:147–183, 2022.
- [8] Moritz Diehl, Hans Georg Bock, and Johannes P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. In *Real-Time PDE-Constrained Optimization*, pages 25–52. SIAM, 2005.
- [9] Gianluca Frison and Moritz Diehl. HPIPM: a high-performance interior-point method for quadratic programs and model predictive control. *IFAC-PapersOnLine*, 53(2):6563–6569, 2020.
- [10] Erwin Coumans and Yunfei Bai. PyBullet, a python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2021.
- [11] MuJoCo Menagerie Contributors. MuJoCo menagerie: A collection of high-quality simulation models for MuJoCo. https://github.com/google-deepmind/mujoco_menagerie, 2022.
- [12] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integration (SII)*, pages 614–619, 2019.