ABC-LMPC: Safe Sample-Based Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions

Brijen Thananjeyan*[1], Ashwin Balakrishna*[1], Ugo Rosolia[2],
Joseph E. Gonzalez[1], Aaron Ames[2], and Ken Goldberg[1]
* equal contribution

[1] University of California Berkeley, Berkeley CA 94720, USA,
[2] California Institute of Technology, Pasadena CA 91125, USA
{bthananjeyan, ashwin_balakrishna} @berkeley.edu

Abstract. Sample-based learning model predictive control (LMPC) strategies have recently attracted attention due to their desirable theoretical properties and their good empirical performance on robotic tasks. However, prior analysis of LMPC controllers for stochastic systems has mainly focused on linear systems in the iterative learning control setting. We present a novel LMPC algorithm, Adjustable Boundary Condition LMPC (ABC-LMPC), which enables rapid adaptation to novel start and goal configurations and theoretically show that the resulting controller guarantees iterative improvement in expectation for stochastic nonlinear systems. We present results with a practical instantiation of this algorithm and experimentally demonstrate that the resulting controller adapts to a variety of initial and terminal conditions on 3 stochastic continuous control tasks.

Keywords: model predictive control, control theory, imitation learning

1 Introduction

Model predictive control (MPC) has seen significant success in a variety of robotic tasks [1–3], and there is substantial experimental and theoretical work demonstrating that the resulting closed loop system performs well on challenging tasks in stochastic dynamical systems [1, 4–6]. In this work, we build on the recently proposed learning model predictive control (LMPC) framework [5–7]. We assume a known stochastic dynamics model and focus on designing an iteratively improving MPC-based control strategy by estimating safe sets and value functions from past closed-loop trajectories.

The LMPC framework [5–7] presents a novel class of reference-free control strategies which utilize MPC to iteratively improve upon a suboptimal controller for a goal directed task. LMPC algorithms typically operate in the iterative learning control setting with fixed initial and terminal conditions, and provide robust guarantees on iterative improvement (in terms of task cost) for stochastic linear systems [5, 6] and deterministic nonlinear systems [7] if the MPC objective can be solved exactly. However, while LMPC based control strategies exhibit a variety of desirable theoretical properties [5–7] and have been shown to work well on practical problems on physical robotic systems [1, 8], they have two key limitations: (1) guarantees for stochastic systems are limited to

linear systems while practical systems are often stochastic and nonlinear and (2) start states and goal sets are typically assumed to be identical in each iteration.

We address both of these challenges. First, we extend the results in [5] to show iterative improvement guarantees for stochastic nonlinear systems. Then, we present a practical algorithm for expanding the set of allowed start states and goal sets during learning while maintaining these guarantees by adapting controller boundary conditions using learned safe sets. The contributions of this work are (1) a novel multi-start, multi-goal LMPC algorithm, Adjustable Boundary Condition LMPC (ABC-LMPC), which optimizes expected costs subject to robust constraints, with (2) guarantees on expected performance, robust constraint satisfaction, and convergence to the goal for stochastic nonlinear systems, (3) a practical method for expanding the allowed set of initial states and goal sets during learning, and (4) simulated continuous control experiments demonstrating that the learned controller can adapt to novel start states and goal sets while consistently and efficiently completing tasks during learning.

2 Related Work

Model Predictive Control: There has been a variety of prior work on learning based strategies for model predictive control in the reinforcement learning [1–3] and controls communities [9–14]. Prior work in learning for model predictive control has focused on estimating the following three components used to design MPC policies: *i*) a model of the system [1–3, 8, 10, 12, 13, 15], *ii*) a safe set of states from which the control task can be completed using a known safe policy [16–19] and *iii*) a value function [1, 5, 6, 20], which for a given safe policy, maps each state of the safe set to the closed-loop cost to complete the task. The most closely related works, both by Rosolia et. al. [5, 6], introduce the learning MPC framework for iterative learning control in stochastic linear systems. Here, MPC is used to iteratively improve upon a suboptimal demonstration by estimating a safe set and a value function from past closed loop trajectories. Robust guarantees are provided for iterative controller improvement if the MPC objective can be solved exactly. Furthermore, Thananjeyan et al. [1] propose a practical reinforcement learning algorithm using these strategies to learn policies for nonlinear systems. However, [1, 6] are limited to the iterative learning control setting, and although [5] presents a strategy for controller domain expansion, the method is limited to linear systems and requires the user to precisely specify an expansion direction. In this work, we build on this framework by (1) extending the theoretical results to prove that under similar assumptions, LMPC based controllers yield iterative improvement in expectation under certain restrictions on the task cost function and (2) providing a practical and general algorithm to adapt to novel start states and goal sets while preserving all theoretical guarantees on controller performance.

Reinforcement Learning: There has been a variety of work from the reinforcement learning (RL) community on learning policies which generalize across a variety of initial and terminal conditions. Curriculum learning [21–23] has achieved practical success in RL by initially training agents on easier tasks and reusing this experience to accelerate learning of more difficult tasks. Florensa et al. [21] and Resnick et al. [22] train policies initialized near a desired goal state, and then iteratively increase the distance

to the goal state as learning progresses. While these approaches have achieved practical success on a variety of simulated robotic and navigation tasks, the method used to expand the start state distribution is heuristic-based and requires significant hand-tuning. We build on these ideas by designing an algorithm which expands the start state distribution for an MPC-based policy by reasoning about reachability, similar to Ivanovic et al. [24]. However, [24] provides a curriculum for model free RL algorithms and does not provide feasibility or convergence guarantees, while we present an MPC algorithm which expands the set of allowed start states while preserving controller feasibility and convergence guarantees. There is also recent interest in goal-conditioned RL [25, 26]. The most relevant prior work in this area is hindsight experience replay [27], which trains a goal-conditioned policy using imagined goals from past failures. This strategy efficiently reuses data to transfer to new goal sets in the absence of dense rewards. We use a similar idea to learn goal-conditioned safe sets to adapt to novel goal sets by reusing data from past trajectories corresponding to goal sets reached in prior iterations.
Motion Planning: The domain expansion strategy of the proposed algorithm, ABC-LMPC, bears a clear connection to motion planning in stochastic dynamical systems [28, 29]. Exploring ways to use ABC-LMPC to design motion planning algorithms which can efficiently leverage demonstrations is an exciting avenue for future work, since the receding horizon planning strategy could prevent the exponential scaling in complexity with time horizon characteristic of open-loop algorithms [30].

3 Problem Statement

In this work, we consider nonlinear, stochastic, time-invariant systems of the form:

$$x_{t+1} = f(x_t, u_t, w_t) \tag{3.0.1}$$

where $x_t \in \mathbb{R}^n$ is the state at time t, $u_t \in \mathbb{R}^m$ is the control, $w_t \in \mathbb{R}^k$ is a disturbance input, and x_{t+1} is the next state. The disturbance w_t is sampled i.i.d. from a known distribution over a bounded set $\mathcal{W} \subseteq \mathbb{R}^p$. We denote Cartesian products with exponentiation, e.g. $\mathcal{W}^2 = \mathcal{W} \times \mathcal{W}$. We consider constraints requiring states to fall in the feasible state space $\mathcal{X} \subseteq \mathbb{R}^n$ and controls to fall in $\mathcal{U} \subseteq \mathbb{R}^m$. Let x_t^j, u_t^j, and w_t^j be the state, control input, and disturbance realization sampled at time t of iteration j respectively. Let $\pi^j : \mathbb{R}^n \to \mathbb{R}^m$ be the control policy at iteration j that maps states to controls (i.e. $u_t^j = \pi^j(x_t^j)$).

Unlike [5], in which the goal of the control design is to solve a robust optimal control problem, we instead define the following objective function with the following Bellman equation with cost function $C(\cdot,\cdot)$:

$$J^{\pi^j}(x_0^j) = \mathop{\mathbb{E}}_{w_0^j}\left[C(x_0^j, \pi^j(x_0^j)) + J^{\pi^j}(f(x_0^j, u_0^j, w_0^j))\right] \tag{3.0.2}$$

Note here that instead of optimizing for the worst case noise realization, we consider control policies which optimize the given cost function in expectation over possible noise realizations. However, we would like to only consider policies that are robustly constraint-satisfying for all time. Thus, the goal of the control design is to solve the

following infinite time optimal control problem:

$$
\begin{aligned}
J^{j,*}_{0\to\infty}(x^j_0) = \min_{\pi^j(.)} \; & J^{\pi^j}(x^j_0) \\
\text{s.t. } & x^j_{t+1} = f(x^j_t, u^j_t, w^j_t) \\
& u^j_t = \pi^j(x^j_t) \\
& x^j_t \in \mathcal{X}, u^j_t \in \mathcal{U} \\
& \forall w^j_t \in \mathcal{W}, t \in \{0,1,\ldots\}
\end{aligned}
\tag{3.0.3}
$$

In this paper, we present a strategy to iteratively design a feedback policy $\pi^j(.) : \mathcal{F}^j_{\mathcal{G}} \subseteq \mathcal{X} \to \mathcal{U}$, where $\mathcal{F}^j_{\mathcal{G}}$ is the domain of π^j for goal set $\mathcal{G}$ (and also the set of allowable initial conditions). Conditioned on the goal set $\mathcal{G}$, the controller design provides guarantees for (i) robust satisfaction of state and input constraints, (ii) convergence in probability of the closed-loop system to $\mathcal{G}$, (iii) iterative improvement: for any $x^j_0 = x^l_0$ where $j < l$, expected trajectory cost is non-increasing ($J^{\pi^j}(x^j_0) \geq J^{\pi^{j+1}}(x^{j+1}_0)$), and (iv) exploration: the domain of the control policy does not shrink over iterations ($\mathcal{F}^j_{\mathcal{G}} \subseteq \mathcal{F}^{j+1}_{\mathcal{G}}$ for all goal sets $\mathcal{G}$ sampled up to iteration j). In Section 4.3, we describe how to transfer to a new goal set $\mathcal{H}$ by reusing data from prior iterations while maintaining the same properties.

We adopt the following definitions and assumptions:

Assumption 1. ***Costs:*** *We consider costs which are zero at the goal set $\mathcal{G}$ and greater than some $\varepsilon > 0$ otherwise: $\exists \varepsilon > 0$ s.t. $C(x,u) \geq \varepsilon \mathbb{1}_{\mathcal{G}^C}(x)$ where $\mathbb{1}$ is an indicator function and $\mathcal{G}^C$ is the complement of $\mathcal{G}$.*

Definition 1. ***Robust Control Invariant Set:*** *As in Rosolia et al. [5], we define a robust control invariant set $\mathcal{A} \subseteq \mathcal{X}$ with respect to dynamics $f(x,u,w)$ and policy class Π as a set where $\forall x \in \mathcal{A}$, $\exists \pi \in \Pi$ s.t. $f(x,\pi(x),w) \in \mathcal{A}, \pi(x) \in \mathcal{U}$, $\forall w \in \mathcal{W}$.*

Assumption 2. ***Robust Control Invariant Goal Set:*** *$\mathcal{G} \subseteq \mathcal{X}$ is a robust control invariant set with respect to the dynamics and the set of state feedback policies Π.*

4 Preliminaries

Here we formalize the notion of safe sets, value functions, and how they can be conditioned on specific goals. We also review standard definitions and assumptions.

4.1 Safe Set

We first recall the definition of a robust reachable set as in [5]:

Definition 2. ***Robust Reachable Set:*** *The robust reachable set $\mathcal{R}^\pi_t(x^j_0)$ contains the set of states reachable in t-steps by the system* (3.0.1) *in closed loop with π at iteration j:*

$$
\begin{aligned}
\mathcal{R}^\pi_{t+1}(x^j_0) = \Big\{ & x_{t+1} \in \mathbb{R}^n \,|\, \exists w_t \in \mathcal{W},\; x_t \in \mathcal{R}^\pi_t(x^j_0),\; x_{t+1} = f(x_t, \pi(x_t), w_t) \Big\} \\
& \textit{where } \mathcal{R}^\pi_0(x^j_0) = x^j_0.
\end{aligned}
\tag{4.1.1}
$$

We define $\mathcal{R}^\pi_{t+1}$ similarly when the input is a set and for time-varying policies.

Now, we define the safe set at iteration j for the goal set $\mathcal{G}$ as in [5].

Definition 3. ***Safe Set:*** *The safe set $\mathcal{SS}^j_{\mathcal{G}}$ contains the full evolution of the system at iteration j,*

$$\mathcal{SS}^j_{\mathcal{G}} = \left\{ \bigcup_{t=0}^{\infty} \mathcal{R}_t^{\pi^j}(x_0^j) \bigcup \mathcal{G} \right\}. \tag{4.1.2}$$

Note that (4.1.2) is robust control invariant by construction [5]. We could set $\mathcal{SS}^0_{\mathcal{G}} = \mathcal{G}$ or initialize the algorithm with a nominal controller π^0. This enables the algorithm to naturally incorporate demonstrations to speed up training.

Definition 4. ***Expected Cost:*** *The expected cost of π^j from start state x_0^j is defined as*

$$J^{\pi^j}(x_0^j) = \mathop{\mathbb{E}}_{w^j} \left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j)) \right] \tag{4.1.3}$$

4.2 Value Function

Definition 5. ***Value Function:*** *Recursively define the value function of π^j in closed-loop with* (3.0.3) *as:*

$$L^{\pi^j}_{\mathcal{G}}(x) = \begin{cases} \mathop{\mathbb{E}}_{w} \left[C(x, \pi^j(x)) + L^{\pi^j}_{\mathcal{G}}(f(x, \pi^j(x), w)) \right] & x \in \mathcal{SS}^j_{\mathcal{G}} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{G}} \end{cases} \tag{4.2.4}$$

Let $V^{\pi^j}_{\mathcal{G}}(x) = \min\limits_{k \in \{0,\dots j\}} L^{\pi^k}_{\mathcal{G}}(x)$, which is the expected cost-to-go of the best performing prior controller at x.

In the event a nominal controller π^0 is used, we require the following assumption on the initial safe set $\mathcal{SS}^0_{\mathcal{G}}$, which is implicitly a restriction on π^0 for start state x_0^0. Observe that $L^{\pi^j}_{\mathcal{G}}$ is defined only on $\mathcal{SS}^j_{\mathcal{G}}$, and $J^{\pi^j} = L^{\pi^j}_{\mathcal{G}}$ on $\mathcal{SS}^j_{\mathcal{G}}$.

Assumption 3. ***Safe Set Initial Condition:*** *If a nominal controller π^0 is used, then $\forall x \in \mathcal{SS}^0_{\mathcal{G}}$, $L^{\pi^0}_{\mathcal{G}}(x) < \infty$.*

This assumption requires that the nominal controller is able to robustly satisfy constraints and converge in probability to $\mathcal{G}$. If no nominal controller is used, then this assumption is not required. In that case, we let $\mathcal{SS}^0_{\mathcal{G}} = \mathcal{G}$ and $L^{\pi^0}_{\mathcal{G}}(x) = 0 \ \forall x \in \mathcal{SS}^0_{\mathcal{G}}$.

4.3 Transfer to Novel Goal Sets

Here we study how the safe set and value function can be modified to transfer the learned controller at iteration $j+1$ to a new robust control invariant goal set $\mathcal{H}$ and reuse data from the earlier iterations.

Definition 6. ***Goal Conditioned Safe Set:*** *Define the goal conditioned safe set by collecting the prefixes of all robust reachable sets until they robustly fall in $\mathcal{H}$ as follows:*

$$\mathcal{SS}^j_{\mathcal{H}} = \begin{cases} \bigcup_{k=0}^{k^*} \mathcal{R}_k^{\pi^j} \bigcup \mathcal{H} & \max_{k \in \mathbb{N}} \mathbb{1}\{\mathcal{R}_k^{\pi^j}(x_0^j) \subseteq \mathcal{H}\} = 1 \\ \mathcal{H} & \textit{otherwise} \end{cases} \tag{4.3.5}$$

where $k^* = \arg\max_{k \in \mathbb{N}} \mathbb{1}\{\mathcal{R}_k^{\pi^j}(x_0^j) \subseteq \mathcal{H}\}$

We also redefine the value function as follows:

Definition 7. ***Goal Conditioned Value Function:*** *Recursively define the goal-conditioned value function of π^j in closed-loop with* (3.0.3) *as:*

$$L^{\pi^j}_{\mathcal{H}}(x) = \begin{cases} \mathbb{E}_w \left[C(x, \pi^j(x)) + L^{\pi^j}(f(x, \pi^j(x), w)) \right] & x \in \mathcal{SS}^j_{\mathcal{H}} \setminus \mathcal{H} \\ 0 & x \in \mathcal{H} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{H}} \end{cases} \tag{4.3.6}$$

Define $V^{\pi^j}_{\mathcal{H}}(x) = \min_{k \in \{0,\dots j\}} L^{\pi^k}_{\mathcal{H}}(x)$ *as before.*

This new value function is for a policy that executes π^j but switches to a policy that keeps the system in $\mathcal{H}$ upon entry.

5 Controller Design

Here we describe the controller design for optimizing the task cost function (Section 5.1), and discuss how this can be extended to iteratively expand the controller domain (Section 5.2). We consider a fixed goal set $\mathcal{G}$ for clarity, but note that the same formulation holds for other goal sets if the safe set and value function are appropriately defined as in (6) and (7). See Figure 1 for an illustration of the learned controller.

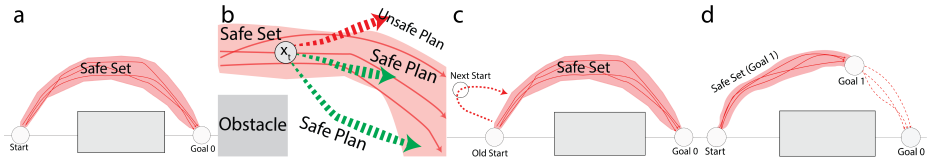


Fig. 1: **Algorithm Overview: (a) Safe Set:** We visualize $\mathcal{SS}^j_{\mathcal{G}_0}$ which is estimated via past closed-loop trajectories collected at iteration j which successfully stabilized to $\mathcal{G}_0$; **(b) Controller Optimization:** Receding-horizon trajectories are chosen to minimize the task cost function while remaining in the feasible state space and terminating in $\mathcal{SS}^j_{\mathcal{G}_0}$; **(c) Start State Expansion:** Trajectories are optimized to explore in desired directions outside of $\mathcal{SS}^j_{\mathcal{G}_0}$ while still terminating in $\mathcal{SS}^j_{\mathcal{G}_0}$, facilitating iterative domain expansion (5.2.4); **(d) Goal Set Transfer:** Trajectories to goal $\mathcal{G}_0$ can be reused to estimate a new safe set for a new goal $\mathcal{G}_1$ ($\mathcal{SS}^j_{\mathcal{G}_1}$).

5.1 Controller Design: Task Driven Optimization

At time t of iteration j with goal set $\mathcal{G}$, the controller solves the following receding-horizon trajectory optimization problem with planning horizon $H > 0$:

$$\begin{aligned} J^j_{t\to t+H}(x^j_t) = \min_{\pi_{t:t+H-1|t}\in\Pi^H} \quad & \mathbb{E}_{w^j_{t:t+H-1}}\left[\sum_{i=0}^{H-1} C(x^j_{t+i|t}, \pi_{t+i|t}(x^j_{t+i|t})) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right] \\ \text{s.t.} \quad & x^j_{t+i+1|t} = f(x^j_{t+i|t}, \pi_{t+i|t}(x^j_{t+i}), w_{t+i}) \ \forall i \in \{0,\dots,H-1\} \\ & x^j_{t+H|t} \in \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}}, \ \forall w^j_{t:t+H-1} \in \mathcal{W}^H \\ & x^j_{t:t+H|t} \in \mathcal{X}^{H+1}, \ \forall w^j_{t:t+H-1} \in \mathcal{W}^H \end{aligned} \tag{5.1.1}$$

where $\pi_{t+i|t}$ is the i-th policy in the planning horizon conditioned on x^j_t and $\pi_{t:t+H-1|t} = \{\pi_{t|t},\dots,\pi_{t+H-1|t}\}$ (likewise for other optimization variables). Let the minimizer of (5.1.1) be $\pi^{*,j}_{t:t+H-1|t}$. Then, execute the first policy at x^j_t:

$$u^j_t = \pi^j(x^j_t) = \pi^{*,j}_{t|t}(x^j_t) \tag{5.1.2}$$

5.2 Controller Design: Start State Expansion

We now describe the control strategy for expanding the controller domain. If there exists a policy π for which the H-step robust reachable set for the start states sampled at iteration j is contained within the current safe set for goal set $\mathcal{G}$, then we can define the feasible set/domain for the controller at iteration j, π^j, for $\mathcal{G}$, by collecting the set of all states for which there exists a sequence of policies which robustly keep the system in $\bigcup_{k=0}^{j-1}\mathcal{SS}^k_{\mathcal{G}}$. Precisely, we define the controller domain as follows:

$$\mathcal{F}^j_{\mathcal{G}} = \{x \mid \exists \pi_{0:H-1} \in \Pi^H \text{ s.t. } \mathcal{R}^{\pi_{0:H-1}}_H(x) \subseteq \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}}\} \tag{5.2.3}$$

This set denotes the points from which the system can robustly plan back to $\bigcup_{k=0}^{j-1}\mathcal{SS}^k_{\mathcal{G}}$. Note that the domain of the policy is a function of the goal set $\mathcal{G}$. While any start state sampled from $\mathcal{F}^j_{\mathcal{G}}$ will ensure feasibility and convergence for goal set $\mathcal{G}$ (proven in Section 6), we present a method to compute states from $\mathcal{F}^j_{\mathcal{G}} \setminus \bigcup_{k=0}^{j-1}\mathcal{SS}^k_{\mathcal{G}}$ to expand $\mathcal{F}^j_{\mathcal{G}}$ towards a desired start state, which may not be added to the domain through task-directed exploration. Computing this set is intractable for general nonlinear stochastic systems, so we introduce a method to approximate this.

At the end of iteration j, we sample a start state $x^j_S \in \bigcup_{k=0}^{j}\mathcal{SS}^k_{\mathcal{G}}$ and seek to execute a sequence of H' exploration policies $\pi^j_{E,0:H'-1}$ which carry the system outside of $\bigcup_{k=0}^{j}\mathcal{SS}^k_{\mathcal{G}}$ and then robustly back into $\bigcup_{k=0}^{j}\mathcal{SS}^k_{\mathcal{G}}$, for all noise realizations

$w_{0:H'-2} \in \mathcal{W}^{H'-1}$ where $H' \geq 0$. The sequence of policies $\pi^j_{E,0:H'-1}$ is computed by solving an H'-step optimization problem with a cost function $C^j_E(x,u)$ that encourages exploration outside of $\bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}}$ while enforcing that the controller terminates in some state $x^j_{H'} \in \bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}}$. In Section 7, we discuss some possibilities for $C^j_E(x,u)$, implement one instantiation, and demonstrate that it enables adaptation to novel start states while maintaining controller feasibility. The sequence of controllers can computed by solving the following 1-step trajectory optimization problem:

$$\begin{aligned} \pi^j_{E,0:H'-1} = \underset{\pi_{0:H'-1} \in \Pi^{H'}}{\operatorname{argmin}} \quad & \mathop{\mathbb{E}}_{w^j_{0:H'-2}} \left[\sum_{i=0}^{H'-1} C^j_E(x^j_i, \pi_i(x^j_i)) \right] \\ \text{s.t.} \quad & x^j_{i+1} = f(x^j_i, \pi_i(x^j_i), w_i),\ \forall i \in \{0,\ldots,H'-1\} \\ & x^j_{H'} \in \bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\ & x^j_{0:H'} \in \mathcal{X}^{H'+1},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \end{aligned} \tag{5.2.4}$$

States from the last H elements of $\left(\mathcal{R}_i^{\pi^j_{E,0:H'-1}}(x^j_S) \right)_{i=0}^{H'}$ can be robustly guided to $\bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}}$ in H steps. This means that $\mathcal{P} = \bigcup_{i=0}^{H} \mathcal{R}_{H'-i}^{\pi^j_{E,0:H'-1}}(x^j_S) \subseteq \mathcal{F}^{j+1}_{\mathcal{G}}$, since $\bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}}$ can be robustly reached within H steps. At iteration $j+1$, feasible start states can be sampled from $\mathcal{P}$ to guide the policy's domain toward a desired target start state. An MPC policy π^j_E could be executed instead to generate these future start states. We could also use the exploration policy to explicitly augment the value function $L^{\pi^j}_{\mathcal{G}}$ and safe set $\mathcal{SS}^j_{\mathcal{G}}$ and thus $\mathcal{F}^j_{\mathcal{G}}$ (Appendix A.4). This could be used for general domain expansion instead of directed expansion towards a particular target start state as presented above.

6 Properties of ABC-LMPC

In this section, we study the properties of the controller constructed in Section 5. For analysis, we will assume a fixed goal set $\mathcal{G}$, but note that if the goal set is changed at some iteration, the same properties still apply to the new goal set $\mathcal{H}$ by the same proofs, because all of the same assumptions hold for $\mathcal{H}$. See Appendix A.1 for all proofs.

Lemma 1. *Recursive Feasibility. Consider the closed-loop system* (5.1.1) *and* (5.1.2)*. Let the safe set $\mathcal{SS}^j_{\mathcal{G}}$ be defined as in* (4.1.2)*. If assumptions 1-3 hold and $x^j_0 \in \mathcal{F}^j_{\mathcal{G}}$, then the controller induced by optimizing* (5.1.1) *and* (5.1.2) *is feasible a.s. for $t \geq 0$ and $j \geq 0$. Equivalently stated, $\mathop{\mathbb{E}}_{w^j_{0:H-1}} [J^j_{t \to t+H}(x^j_t)] < \infty$, $\forall t, j \geq 0$.*

Lemma 1 shows that the controller is guaranteed to satisfy state-space constraints for all timesteps t in all iterations j given the definitions and assumptions presented above. Equivalently, the expected planning cost of the controller is guaranteed to be finite. The following lemma establishes convergence in probability to the goal set given initialization within the controller domain.

Lemma 2. *Convergence in Probability. Consider the closed-loop system defined by* (5.1.1) *and* (5.1.2)*. Let the sampled safe set $\mathcal{SS}^j_{\mathcal{G}}$ be defined as in* (4.1.2)*. Let assumptions 1-3 hold and $x^j_0 \in \mathcal{F}^j_{\mathcal{G}}$. If the closed-loop system converges in probability to $\mathcal{G}$ at iteration* 0*, then it converges in probability at all subsequent iterations. Stated precisely, at iteration j: $\lim_{t\to\infty} P(x^j_t \notin \mathcal{G}) = 0$.*

Theorem 1. *Iterative Improvement. Consider system* (3.0.1) *in closed-loop with* (5.1.1) *and* (5.1.2)*. Let the sampled safe set $\mathcal{SS}^j$ be defined as in* (4.1.2)*. Let assumptions 1- 3 hold, then the expected cost-to-go* (4.1.3) *associated with the control policy* (5.1.2) *is non-increasing in iterations for a fixed start state. More formally:*

$$\forall j \in \mathbb{N},\, x^j_0 \in \mathcal{F}^j_{\mathcal{G}},\, x^{j+1}_0 \in \mathcal{F}^{j+1}_{\mathcal{G}} \implies J^{\pi^j}(x^j_0) \geq J^{\pi^{j+1}}(x^{j+1}_0)$$

Furthermore, $\{J^{\pi^j}(x^j_0)\}_{j=0}^{\infty}$ is a convergent sequence.

Theorem 1 extends prior results [5], which guarantee robust iterative improvement for stochastic linear systems with convex costs. Here we show iterative improvement in *expectation* for ABC-LMPC for stochastic nonlinear systems with costs as in Assumption 1. The following result implies that the policy domain is non-decreasing.

Lemma 3. *Policy domain expansion. The domain of π^j is an non-decreasing sequence of sets: $\mathcal{F}^j_{\mathcal{G}} \subseteq \mathcal{F}^{j+1}_{\mathcal{G}}$.*

7 Practical Implementation

ABC-LMPC alternates between two phases at each iteration: the first phase performs the task by executing π^j and the second phase runs the exploration policy $\pi^j_{E,0:H'-1}$. Only data from π^j is added to an approximation of $\mathcal{SS}^j_{\mathcal{G}}$, on which the value function L^{π^j} is fit, but in principle, data from $\pi^j_{E,0:H'-1}$ can also be used. This procedure provides a simple algorithm to expand the policy's domain $\mathcal{F}^j_{\mathcal{G}}$ while approximately maintaining theoretical properties. Here, we describe how each component in the controller design is implemented and how optimization is performed in practice. See Appendix A.2 for further implementation details.

7.1 Sampled-Based Safe Set

In practice, as in [5], we approximate the safe set $\mathcal{SS}^j_{\mathcal{G}}$ using samples from the closed loop system defined by (5.1.1) and (5.1.2). To do this, we collect R closed-loop trajectories at iteration j, each of length T as in [5] where T is the task horizon.

Thus, given the ith disturbance realization sequence collected at iteration j, given by $\mathbf{w}^j_i = [w^j_{0,i}, \dots w^j_{T,i}]$, we define the closed loop trajectory associated with this sequence as in [5]: $\mathbf{x}^j(\mathbf{w}^j_i) = \left[x^j_0(\mathbf{w}^j_i), \dots, x^j_T(\mathbf{w}^j_i)\right]$. As in [5], we note that $x^j_k(\mathbf{w}^j_i) \in \mathcal{R}^{\pi^j}_k(x^j_0)$, so R rollouts from the closed-loop system provides a sampled-based approximation to $\mathcal{R}^{\pi^j}_k(x^j_0)$ as follows: $\tilde{\mathcal{R}}^{\pi^j}_k(x^j_0) = \bigcup_{i=1}^R x^j_k(\mathbf{w}^j_i) \subseteq \mathcal{R}^{\pi^j}_k(x^j_0)$. Similarly, we can define a sampled-based approximation to the safe set as follows: $\tilde{\mathcal{SS}}^j_{\mathcal{G}} = \left\{\bigcup_{k=0}^{\infty} \tilde{\mathcal{R}}^{\pi^j}_k(x^j_0) \bigcup \mathcal{G}\right\}$.

While $\tilde{\mathcal{SS}}^j_{\mathcal{G}}$ is not robust control invariant, with sufficiently many trajectory samples (i.e. R sufficiently big), this approximation becomes more accurate in practice [5]. To obtain a continuous approximation of the safe set for planning, we use the same technique as [1], and fit density model $\rho^{\mathcal{G}}_{\alpha}$ to $\bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ and instead of enforcing the terminal constraint by checking if $x_{t+H} \in \bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$, ABC-LMPC instead enforces that $\rho^{\mathcal{G}}_{\alpha}(x_{t+H}) > \delta$, where α is a kernel width parameter. We implement a tophat kernel density model using a nearest neighbors classifier with tuned kernel width α and use $\delta = 0$ for all experiments. Thus, all states within Euclidean distance α from the closest state in $\bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ are considered safe under $\rho^{\mathcal{G}}_{\alpha}$.

7.2 Start State Expansion Strategy

To provide a sample-based approximation to the procedure from Section 5.2, we sample states x^j_S from $\bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^k$ and execute $\pi^j_{E,0:H'-1}$ for R trajectories of length H'. These trajectories approximate $\bigcup_{k=0}^{H'} \mathcal{R}_k^{\pi^j_{E,0:H'-1}}(x^j_S)$. We repeat this process until an x^j_S is found such that all R sampled trajectories satisfy the terminal state constraint that $x^j_{H'} \in \bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ (Section 5.2). Once such a state is found, a state is sampled from the last H steps of the corresponding trajectories to serve as the start state for the next iteration which approximates sampling from $\bigcup_{i=0}^{H} \mathcal{R}_{H'-i}^{\pi^j_{E,0:H'-1}}(x^j_S)$. We utilize a cost function which encourages expansion of the policy domain towards a specific desired start state x^*, although in general any cost function can be used. This cost function is interesting because it enables adaptation of a learning MPC controller to desired specifications while maintain controller feasibility. Precisely, we optimize a cost function which simply measures the discrepancy between a given state in a sampled trajectory and x^*, ie. $C^j_E(x,u) = D(x,x^*)$. This distance measure can be tuned on a task-specific basis based on the appropriate distance measures for the domain (Section 8.3). However, we remark that this technique requires: (1) design of an appropriate distance function $D(\cdot,\cdot)$, (2) a reverse path from the goal to the start state, that may differ from the optimal forward path, along which the goal is robustly reachable.

7.3 Goal Set Transfer

We practically implement the goal set transfer strategy in Section 4.3 by fitting a new density model $\rho^{\mathcal{H}}_{\alpha}$ on the prefixes of prior trajectories that intersect some new user-specified goal set $\mathcal{H}$. If $\mathcal{H}$ is chosen such that $\tilde{\mathcal{SS}}^j_{\mathcal{H}}$ contains many points, the controller can seamlessly transfer to $\mathcal{H}$. If this is not the case, the controller domain for $\mathcal{H}$ must be expanded from $\mathcal{H}$ until it intersects many trajectories in the original domain.

7.4 ABC-LMPC Optimization Procedure

As in prior work on MPC for nonlinear control [1, 2], we solve the MPC optimization problem in (5.1.1) over sequences of controls using the cross entropy method (CEM) [31]. In practice, we implement the terminal safe set constraints and state-space constraints in (5.1.1) and (5.2.4) by imposing a large cost on sampled action sequences which violate constraints when performing CEM. We use a probabilistic ensemble of 5 neural networks to approximate $L^{\pi^j}_{\mathcal{G}}(x)$ as in [1]. In contrast to [1], a separate $L^{\pi^j}_{\mathcal{G}}(x)$

is fit using data from each iteration instead of fitting a single function approximator on data from all iterations. We utilize Monte Carlo estimates of the cost-to-go values when fitting $L_{\mathcal{G}}^{\pi^j}(x)$. Each element of the ensemble outputs the parameters of a conditional axis-aligned Gaussian distribution and are trained on bootstrapped samples from the training dataset using a maximum likelihood objective as in [2].

8 Experiments

We evaluate whether ABC-LMPC can enable (1) iterative improvement in expected performance for stochastic nonlinear systems, (2) adaptation to new start states and (3) transfer to new goal sets on 3 simulated continuous control domains. In Section 8.1 we describe the experimental domains, in Section 8.2, we keep the start state and goal sets fixed, in Section 8.3, we expand the policy domain iteratively toward a desired start state far from the goal set, in Section 8.4, we switch the goal set during learning, and finally in Section 8.5 we utilize both start state expansion and the goal set transfer technique to control a pendulum to an upright position. In all experiments, we use $C(x,u) = \mathbb{1}\{x \notin \mathcal{G}\}$ as in [1]. Note that for this cost function, the maximum trajectory cost is the task horizon T, and the resulting objective corresponds to minimum time optimal control. We include comparisons to the minimum trajectory cost achieved by the state-of-the-art demonstration augmented model-based reinforcement learning algorithm, SAVED [1] after 10 iterations of training to evaluate the quality of the learned controller. For all experiments, we use $R = 5$ closed-loop trajectories from the current controller to estimate $\tilde{\mathcal{SS}}_{\mathcal{G}}^{j}$ and perform start state expansion. Experimental domains are designed to have comparable stochasticity to those in [5]. See Appendix A.3 for further details about experimental, optimization, and environment parameters.

8.1 Experimental Domains

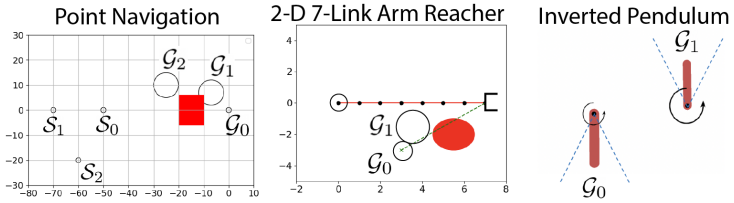


Fig. 2: **Experimental Domains:** We evaluate ABC-LMPC on three stochastic domains: a navigation domain with an obstacle, a 2D 7-link arm reacher domain with an obstacle, and an inverted pendulum domain. In the first two domains, suboptimal demonstrations are provided, while no demonstrations are provided for the inverted pendulum task.

Point Mass Navigation: We consider a 4-dimensional (x, y, v_x, v_y) navigation task as in [1], in which a point unit mass is navigating to a goal set (a unit ball centered at the origin unless otherwise specified). The agent exerts force (f_x, f_y), $\|(f_x, f_y)\| \leq 1$, in each cardinal direction and experiences drag coefficient ψ. We introduce truncated Gaussian process noise $z_t \sim \mathcal{N}(0, \sigma^2 I)$ in the dynamics with domain $[-\sigma, \sigma]$. We include a large obstacle in the center of the environment that the robot must navigate around to reach

the goal. While this task has linear dynamics, the algorithm must consider non-convex state space constraints and stochasticity.
7-Link Arm Reacher: Here, we consider a 2D kinematic chain with 7 joints where the agent provides commands in delta joint angles. We introduce truncated Gaussian process noise $z_t \sim \mathcal{N}(0, \sigma^2 I)$ in the dynamics with domain $[-\sigma, \sigma]$ and build on the implementation from [32]. The goal is to control the end effector position to a 0.5 radius circle in $\mathbb{R}^2$ centered at $(3, -3)$. We do not model self-collisions but include a circular obstacle of radius 1 in the environment which the kinematic chain must avoid.
Inverted Pendulum: This environment is a noisy inverted pendulum task adapted from OpenAI Gym [33]. We introduce truncated Gaussian process noise in the dynamics.

8.2 Fixed Start and Goal Conditions

We first evaluate ABC-LMPC on the navigation and reacher environments with a fixed start state and goal set, with no start state or goal set adaptation. In the navigation domain, the robot must navigate from $\mathcal{S}_\prime = (-50, 0, 0, 0)$ to the origin ($\mathcal{G}_0$) while in the reacher domain, the agent must navigate from a joint configuration with the end effector at $(7, 0)$ to one with the end effector at $(3, -3)$ ($\mathcal{G}_1$). For optimization parameters and other experimental details, see Appendix A.3. The controller rapidly and significantly improves upon demonstrations for both domains (Figure 3). The controller achieves comparable cost to SAVED for both tasks and never violates constraints during learning.

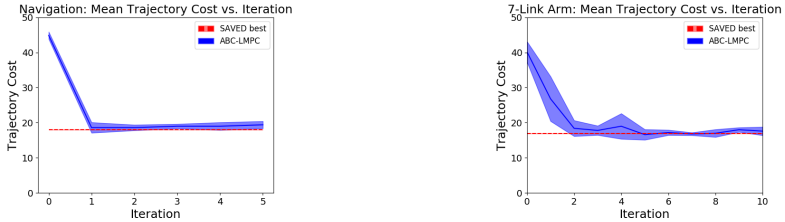


Fig. 3: **Fixed Start, Single Goal Set Experiments:** Learning curves for ABC-LMPC averaged over $R = 5$ rollouts per iteration on simulated continuous control domains when the start state and goal set is held fixed during learning. Performance of the demonstrations is shown at iteration 0, and the controller performance is shown thereafter. **Point Mass Navigation:** The controller immediately improves significantly upon the demonstration performance within 1 iteration, achieving a mean trajectory cost of around 20 while demonstrations have mean trajectory cost of 42.58. **7-Link Arm Reacher:** The controller significantly improves upon the demonstrations, achieving a final trajectory cost of around 18 while demonstrations achieve a mean trajectory cost of 37.77. In all experiments, the controller quickly converges to the best cost produced by SAVED.

8.3 Start State Expansion

ABC-LMPC is now additionally provided a target start state which is initially outside its domain and learns to iteratively expand its domain toward the desired start state. We report the sequence of achieved start states over iterations in addition to the mean and standard deviation trajectory cost. ABC-LMPC is able to maintain feasibility throughout learning and achieve comparable performance to SAVED at the final start state when SAVED is supplied with 100 demonstrations from the desired state. To ensure that results are meaningful, we specifically pick desired start states such that given 100

demonstrations from the original start state, ABC-LMPC is never able to accomplish the task after 30 iterations of learning. A different $C_E(x,u)$ is used for start state expansion based on an appropriate distance metric for each domain.

We first consider a navigation task where 100 suboptimal demonstrations are supplied from $(-25,0,0,0)$ with average trajectory cost of 44.76. The goal is to expand the policy domain in order to navigate from start states $\mathcal{S}_1 = (-70,0,0,0)$ and $\mathcal{S}_2 = (-60,-20,0,0)$. $C_E(x,u)$ measures the Euclidean distance between the positions of x and the desired start state. After 20 iterations, the controller reaches the desired start state while consistently maintaining feasibility during learning for both tasks (Table 1).

We then consider a similar Reacher task using the same suboptimal demonstrations from Section 8.2. The desired start end effector position is $(-1,0)$, and $C_E(x,u)$ measures the Euclidean distance between the end effector position of states x in optimized trajectories and that of the desired start state. Within 16 iterations of learning, the controller is able to start at the desired start state while maintaining feasibility during learning (Table 1). On both domains, the controller achieves comparable performance to SAVED when trained with demonstrations from that start state and the controller successfully expands its domain while rapidly achieving good performance at the new states. Constraints are never violated during learning for all experiments.

Table 1: **Start State Expansion Experiments: Pointmass Navigation:** Start State Expansion towards position $(-70,0)$ (left) and $(-60,-20)$ (center). Here we see that ABC-LMPC is able to reach the desired start state in both cases while consistently maintaining controller feasibility throughout learning. Furthermore, the controller achieves competitive performance with SAVED, which achieves a minimum trajectory cost of 21 from $(-70,0)$ and 23 from $(-60,-20)$; **7-link Arm Reacher:** Here we expand the start state from that corresponding to an end effector position of $(7,0)$ to that corresponding to an end effector position of $(-1,0)$ (right). Again, we see that the controller consistently maintains feasibility during learning and achieves trajectory costs comparable to SAVED, which achieves a minimum trajectory cost of 24. The trajectory costs are presented in format: mean $\pm$ standard deviation over $R = 5$ rollouts.

Point Navigation (-70, 0)

Iteration	Start Pos (x, y)	Trajectory Cost
4	$(-42.3, 1.33)$	23.0 ± 0.89
8	$(-54.1, 0.08)$	22.8 ± 1.67
12	$(-61.2, 2.70)$	25.0 ± 2.37
16	$(-70.3, -0.26)$	32.6 ± 5.08
20	$(-70.4, 0.12)$	29.4 ± 2.33

Point Navigation (-60, -20)

Iteration	Start Pos (x, y)	Trajectory Cost
4	$(-42.6, -8.76)$	19.6 ± 4.22
8	$(-54.6, -14.2)$	25.6 ± 5.23
12	$(-58.8, -20.3)$	27.2 ± 12.0
16	$(-60.6, -20.2)$	21.0 ± 0.63
20	$(-60.5, -19.6)$	22.4 ± 1.85

7-Link Reacher

Iteration	Start EE Position	Trajectory Cost
4	$(-1.28, -0.309)$	31.6 ± 8.04
8	$(-0.85, -0.067)$	30.8 ± 15.7
12	$(-0.95, -0.014)$	20.2 ± 1.83
16	$(-1.02, -0.023)$	19.4 ± 4.03

8.4 Goal Set Transfer

ABC-LMPC is trained as in Section 8.2, but after a few iterations, the goal set is changed to a new goal set that is in the policy domain. In the navigation domain, the robot is supplied a new goal set centered at $\mathcal{G}_1 = (-25,10,0,0)$ or $\mathcal{G}_2 = (-7,7,0,0)$ with radius 7 after 2 iterations of learning on the original goal set. We increase the radius so more prior trajectories can be reused for the new goal-conditioned value function. Results are shown in Figure 4 for both goal set transfer experiments. We also perform a goal set transfer experiment on the 7-link Reacher Task in which the robot is supplied a new goal set centered at $\mathcal{G}_1 = (4,0.2)$ with radius 1 after 2 iterations of training. Results are shown in Figure 4. In both domains, ABC-LMPC seamlessly transfers to the new goal by leveraging prior experience to train a new set of value functions.

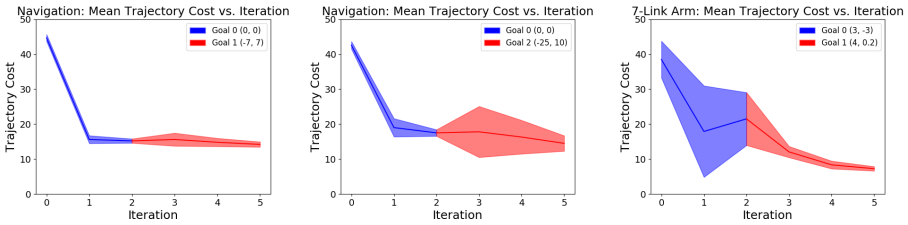


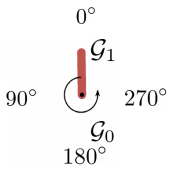
Fig. 4: **Goal Set Transfer Learning:** In this experiment, the goal set is switched to to a new goal set at iteration 3 and we show a learning curve which indicates performance on both the first goal (blue) and new goal (red). The controller is re-trained as in Section 7.3 to stabilize to the new goal. The controller immediately is able to perform the new task and never hits the obstacle. Results are plotted over $R = 5$ controller rollouts per iteration.

8.5 Inverted Pendulum Swing-Up Task

In this experiment, we incorporate both the start state optimization procedure and goal set transfer strategies to balance the pendulum in the upright position, but without any demonstrations. We initialize the pendulum in the downward orientation ($\mathcal{G}_0$), and the goal of the task is to eventually stabilize the system to the upright orientation ($\mathcal{G}_1$). We iteratively expand the policy domain using the start state expansion strategy with initial goal $\mathcal{G}_0$ until the pendulum has swung up sufficiently close to the upright orientation. Once this is the case, we switch the goal set to $\mathcal{G}_1$ to stabilize to the upright position. The controller seamlessly transitions between the two goal sets, immediately transitioning to $\mathcal{G}_1$ while completing the task (convergence to either $\mathcal{G}_0$ or $\mathcal{G}_1$ within the task horizon) on all iterations (Table 2). $C_E(x,u)$ measures the distance between the pendulum's orientation and the desired start state's orientation.

Table 2: **Pendulum Swing Up Experiment:** We iteratively expand the policy domain outward from a goal set centered around the downward orientation ($\mathcal{G}_0$) towards the upward orientation until the policy domain includes a goal set centered around the upward orientation ($\mathcal{G}_1$). Then, the goal set is switched to $\mathcal{G}_1$. The resulting controller maintains feasibility throughout and seamlessly transitions to $\mathcal{G}_1$. The trajectory costs are presented as: mean $\pm$ standard deviation over $R = 5$ rollouts. The upward orientation corresponds to a pendulum angle of 0° and the angle (degrees) increases counterclockwise from this position until 360°.

Iteration	Start Angle	Goal Set	Trajectory Cost
3	200.3	$\mathcal{G}_0$	30.2 ± 1.47
6	74.3	$\mathcal{G}_0$	35.0 ± 0.00
9	53.9	$\mathcal{G}_0$	34.4 ± 0.49
12	328.1	$\mathcal{G}_1$	36.0 ± 0.63
15	345.1	$\mathcal{G}_1$	13.8 ± 7.03
18	0.6	$\mathcal{G}_1$	0.00 ± 0.00



9 Discussion and Future Work

We present a new algorithm for iteratively expanding the set of feasible start states and goal sets for an LMPC-based controller and provide theoretical guarantees on iterative improvement in expectation for non-linear systems under certain conditions on the cost function and demonstrate its performance on stochastic linear and nonlinear continuous control tasks. In future work, we will explore synergies with sample based motion planning to efficiently generate asymptotically optimal plans. We will also integrate the

reachability-based domain expansion strategies of ABC-LMPC with model-based RL to learn safe and efficient controllers when dynamics are learned from experience.

Acknowledgements: This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, Berkeley Deep Drive (BDD), the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS "People and Robots" (CPAR) Initiative. Authors were also supported by the Scalable Collaborative Human-Robot Learning (SCHooL) Project, a NSF National Robotics Initiative Award 1734633, and in part by donations from Google and Toyota Research Institute. Ashwin Balakrishna is supported by an NSF GRFP. This article solely reflects the opinions and conclusions of its authors and do not reflect the views of the sponsors. We thank our colleagues who provided feedback and suggestions, in particular Michael Danielczuk, Daniel Brown and Suraj Nair for their helpful input.

References

1. Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E. Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. "Safety Augmented Value Estimation from Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks". In: *IEEE Robotics & Automation Letters*. 2020.
2. Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Proc. Advances in Neural Information Processing Systems*. 2018.
3. Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. "Deep Dynamics Models for Learning Dexterous Manipulation". In: *Conf. on Robot Learning (CoRL)*. 2019.
4. Ashwin Balakrishna, Brijen Thananjeyan, Jonathan Lee, Arsh Zahed, Felix Li, Joseph E. Gonzalez, and Ken Goldberg. "On-Policy Robot Imitation Learning from a Converging Supervisor". In: *Conf. on Robot Learning (CoRL)*. 2019.
5. Ugo Rosolia and Francesco Borrelli. "Sample-Based Learning Model Predictive Control for Linear Uncertain Systems". In: *CoRR* (2019). arXiv: 1904.06432.
6. U. Rosolia, X. Zhang, and F. Borrelli. "Robust Learning Model Predictive Control for Iterative Tasks: Learning from Experience". In: *Annual Conference on Decision and Control (CDC)*. 2017.
7. Ugo Rosolia and Francesco Borrelli. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework". In: *IEEE Transactions on Automatic Control* (2018).
8. Ugo Rosolia and Francesco Borrelli. "Learning how to autonomously race a car: a predictive control approach". In: IEEE, 2019.
9. Anil Aswani, Humberto Gonzalez, Shankar Sastry, and Claire Tomlin. "Provably Safe and Robust Learning-Based Model Predictive Control". In: *Automatica* 49 (2011).
10. Jus Kocijan, Roderick Murray-Smith, C.E. Rasmussen, and A. Girard. "Gaussian process model based predictive control". In: 2004.
11. Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. "Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning". In: 2018.
12. Lukas Hewing, Alexander Liniger, and Melanie Zeilinger. "Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars". In: 2018.
13. Enrico Terzi, Lorenzo Fagiano, Marcello Farina, and Riccardo Scattolini. "Learning-based predictive control for linear systems: A unitary approach". In: *Automatica* 108 (2019).
14. Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. "Cautious model predictive control using Gaussian process regression". In: *IEEE Transactions on Control Systems Technology* (2019).

15. Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. "Gaussian process model based predictive control". In: *Proceedings of the 2004 American Control Conference*.
16. Marko Bacic, Mark Cannon, Young Il Lee, and Basil Kouvaritakis. "General interpolation in MPC and its advantages". In: *IEEE Transactions on Automatic Control* 48 (2003).
17. Florian D Brunner, Mircea Lazar, and Frank Allgöwer. "Stabilizing linear model predictive control: On the enlargement of the terminal set". In: *2013 European Control Conference (ECC)*. 2013.
18. Kim P Wabersich and Melanie N Zeilinger. "Linear model predictive safety certification for learning-based control". In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018.
19. Franco Blanchini and Felice Andrea Pellegrino. "Relatively optimal control and its linear implementation". In: *IEEE Transactions on Automatic Control* 48 (2003).
20. Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *Proc. Int. Conf. on Machine Learning*. 2019.
21. Carlos Florensa, David Held, Markus Wulfmeier, and Pieter Abbeel. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conf. on Robot Learning (CoRL)*. 2017.
22. Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alex Peysakhovich, Kyunghyun Cho, and Joan Bruna. "Backplay: "Man muss immer umkehren"". In: *CoRR* (2018). arXiv: 1807.06919.
23. Sanmit Narvekar and Peter Stone. "Learning Curriculum Policies for Reinforcement Learning". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019.
24. Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. "BaRC: Backward Reachability Curriculum for Robotic Reinforcement Learning". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2019.
25. Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. "Visual Reinforcement Learning with Imagined Goals". In: *Proc. Advances in Neural Information Processing Systems*. 2018.
26. Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. "Universal Value Function Approximators". In: *Proc. Int. Conf. on Machine Learning*. 2015.
27. Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. 2017.
28. Jur van den Berg, Pieter Abbeel, and Kenneth Y. Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information." In: *I. J. Robotics Res.* 30.7 (2011), pp. 895–913.
29. Alex Pui-wai Lee, Sachin Patil, John Schulman, Zoe McCarthy, Jur van den Berg, Ken Goldberg, and Pieter Abbeel. "Gaussian Belief Space Planning for Imprecise Articulated Robots". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2013.
30. Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. "Motion planning under uncertainty for robotic tasks with long time horizons". In: *Int. Journal of Robotics Research (IJRR)* 30.3 (2011), pp. 308–323.
31. Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Faculty Of Industrial Engineering. *The Cross-Entropy Method for Optimization*. 2013.
32. Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. "PythonRobotics: a Python code collection of robotics algorithms". In: (2018).
33. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

A Appendix

A.1 Proofs of Controller Properties

Proof of Lemma 1 We proceed by induction. By assumption 3, $J^0_{0\to H}(x^j_0) < \infty$. By the definition of $V^{\pi^j}_{\mathcal{G}}$ and $\mathcal{F}^j_{\mathcal{G}}$, $J^j_{0\to H}(x^j_0) < \infty$. Let $J^j_{t\to t+H}(x^j_t) < \infty$ for some $t \in \mathbb{N}$. In the following expressions, we do not explicitly write the MPC problem constraints for clarity. Conditioning on the random variable x^j_t:

$$J^j_{t\to t+H}(x^j_t) = \mathop{\mathbb{E}}_{w^j_{t:t+H-1}}\left[\sum_{k=0}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right] \tag{A.1.1}$$

$$= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t)) + \mathbb{E}_{w^j_{t:t+H-1}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right] \tag{A.1.2}$$

$$\begin{aligned} &= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t)) \\ &+ \mathop{\mathbb{E}}_{w^j_{t:t+H}}\Bigg[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + C(x^j_{t+H|t}, \pi^l(x^j_{t+H|t})) \\ &+ V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H+1|t})\Bigg],\ l \in [j-1] \end{aligned} \tag{A.1.3}$$

$$\begin{aligned} &\geq C(x^j_t, \pi^{*,j}_{t|t}(x^j_t)) \\ &+ \mathop{\mathbb{E}}_{w^j_t}\Bigg[\min_{\pi_{t+1:t+H|t+1}} \mathop{\mathbb{E}}_{w^j_{t+1:t+H}}\Bigg[\sum_{k=1}^{H-1} C(x^j_{t+k|t+1}, \pi_{t+k|t+1}(x^j_{t+k|t+1})) \\ &+ C(x^j_{t+H|t+1}, \pi_{t+H|t+1}(x^j_{t+H|t+1})) \\ &+ V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H+1|t+1})\Bigg]\Bigg] \end{aligned} \tag{A.1.4}$$

$$= C(x^j_t, \pi^j(x^j_t)) + \mathop{\mathbb{E}}_{w^j_t}\left[J^j_{t+1\to t+H+1}(x^j_{t+1})|x^j_t\right] \tag{A.1.5}$$

Equation A.1.1 follows from the definition in 5.1.1, equation A.1.3 follows from the definition of $V^{\pi_{j-1}}_{\mathcal{G}}$, which is defined as a point-wise minimum over $\left(L^{\pi^l}_{\mathcal{G}}\right)_{l=0}^{j-1}$. We take a function $L^{\pi^l}_{\mathcal{G}}$ that is active at $x^j_{t+H|t}$ and apply its definition to expand it and then replace $L^{\pi^l}_{\mathcal{G}}$ with $V^{\pi^{j-1}}_{\mathcal{G}}$ in the expansion. The inner expectation in equation A.1.4 conditions on the random variable x^j_{t+1}, and the outer expectation integrates it out. The inequality in A.1.4 follows from the fact that $[\pi^{*,j}_{t+1|t}, \ldots, \pi^{*,j}_{t+H-1|t}, \pi^{j-1}]$ is a possible solution to (A.1.4). Equation A.1.5 follows from the definition in equation 5.1.1.

We have shown that $J^j_{t\to t+H}(x^j_t) < \infty \implies \mathop{\mathbb{E}}_{w^j_t}\left[J^j_{t+1\to t+H+1}(x^j_{t+1|t})\right] < \infty$. So:

$$\mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x^j_t)\right] < \infty \implies \mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[\mathop{\mathbb{E}}_{w^j_t}\left[J^j_{t+1\to t+H+1}(x^j_{t+1|t})\right]\right] \tag{A.1.6}$$

$$= \mathop{\mathbb{E}}_{w^j_{0:t}}\left[J^j_{t+1\to t+H+1}(x^j_{t+1})\right] < \infty \tag{A.1.7}$$

By induction, $\mathop{\mathbb{E}}_{w^j_{0:t-1}}[J^j_{t\to t+H}(x^j_t)] < \infty \ \forall t \in \mathbb{N}$. Therefore, the controller is feasible at iteration j. □

Proof of Lemma 2 By Lemma 1 and Assumption 1, $\forall L \in \mathbb{N}$,

$$\mathop{\mathbb{E}}_{w^j_{1:L-1}}\left[\sum_{k=0}^{L-1} C(x^j_k, \pi^j(x^j_k)) + J^j_{L\to L+H}(x^j_L)\right] \leq J^j_{0\to H}(x^j_0) \tag{A.1.8}$$

$$\implies \mathop{\mathbb{E}}_{w^j_{1:L-1}}\left[J^j_{L\to L+H}(x^j_L)\right] \leq J^j_{0\to H}(x^j_0) - \mathop{\mathbb{E}}_{w^j_{1:L-1}}\left[\sum_{k=0}^{L-1} C(x^j_k, \pi^j(x^j_k))\right] \tag{A.1.9}$$

$$\leq J^j_{0\to H}(x^j_0) - \varepsilon \sum_{k=0}^{L-1} P(x^j_k \notin \mathcal{G}) \tag{A.1.10}$$

Line A.1.10 follows from rearranging A.1.8 and applying assumption 1. Because $\mathcal{G}$ is robust control invariant by assumption 2, $x_t \in \mathcal{G} \implies x_{t+k} \in \mathcal{G}\ \forall k \geq 0$. Now, assume $\lim_{k\to\infty} P(x^j_k \notin \mathcal{G})$ does not exist or is nonzero. This implies that $P(x^j_k \notin \mathcal{G}) \geq \delta > 0$ infinitely many times. By the Archimedean principle, the RHS of A.1.10 can be driven arbitrarily negative, which is impossible. By contradiction, $\lim_{k\to\infty} P(x^j_k \notin \mathcal{G}) = 0$. □

Proof of Theorem 1 Let $j \in \mathbb{N}$

$$J^j_{0\to H}(x_0) \geq C(x_0, u_0) + \mathop{\mathbb{E}}_{w^j_0}\left[J^j_{1\to H+1}(x^j_1)\right] \tag{A.1.11}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x^j_t, \pi^j(x^j_t))\right] + \lim_{t\to\infty} \mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x^j_t)\right] \tag{A.1.12}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x^j_t, \pi^j(x^j_t))\right] + \lim_{t\to\infty} \mathop{\mathbb{E}}_{\mathbb{1}\{x^j_t \notin \mathcal{G}\}}\left[\mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x_t)|\mathbb{1}\{x^j_t \notin \mathcal{G}\}\right]\right] \tag{A.1.13}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x^j_t, \pi^j(x^j_t))\right] + \lim_{t\to\infty} \mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x^j_t)|x^j_t \notin \mathcal{G}\right] P(x^j_t \notin \mathcal{G}) \tag{A.1.14}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty} \varepsilon P(x_t^j \notin \mathcal{G}) \tag{A.1.15}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] = J^{\pi^j}(x_0) \tag{A.1.16}$$

Equations A.1.11 and A.1.12 follow from repeated application of Lemma 1 (A.1.5). Equation A.1.13 follows from iterated expectation, equation A.1.14 follows from the cost function assumption 1. Equation A.1.15 follows again from assumption 1 (incur a cost of at least ε for not being at the goal at time t). Then, Equation A.1.16 follows from Lemma 2. Using the above inequality with the definition of $J^{\pi^j}(x_0)$,

$$J^j_{0\to H}(x_0) \geq J^{\pi^j}(x_0) = \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x_t^j, \pi^j(x_t)) + V_{\mathcal{G}}^{\pi^j}(x_H^j)\right] \tag{A.1.17}$$

$$\geq \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x_t^j, \pi^{*,j}_{t|0}(x_{t|0})) + V_{\mathcal{G}}^{\pi^j}(x_{H|0})\right] = J^{j+1}_{0\to H}(x_0) \tag{A.1.18}$$

$$\geq J^{\pi^{j+1}}(x_0) \tag{A.1.19}$$

Equation A.1.17 follows from equation A.1.16, equation A.1.18 follows from taking the minimum over all possible H-length sequences of policies in the policy class Π. Equation A.1.19 follows from equation A.1.16. By induction, this proves the theorem.

Note that this also implies convergence of $(J^{\pi^j}(x_0))_{j=0}^{\infty}$ by the Monotone Convergence Theorem. □

Proof of Lemma 3 The proof is identical to [5]. Because $\mathcal{SS}^j_{\mathcal{G}}$ is an increasing sequence of sets, $\mathcal{F}^j_{\mathcal{G}}$ is also an increasing sequence of sets by definition. □

A.2 Adjustable Boundary Condition LMPC Implementation Details

Solving the MPC Objective As in [1], we sample a fixed population size of action sequences at each iteration of CEM from a truncated Gaussian. These action sequences are simulated over a known model of the system dynamics and then the sampling distribution for the next iteration is updated based on the lowest cost sampled trajectories. For the cross entropy method we build off of the implementation in [2]. Precisely, at each timestep in a trajectory, a conditional Gaussian is initialized with the mean based on the final solution for the previous timestep and some fixed variance. Then, at each iteration of CEM, pop_size action sequences of plan_hor length are sampled from the conditional Gaussian, simulated over a model of the system dynamics, and then the num_elites samples with the lowest sum cost are used to refit the mean and variance of the conditional Gaussian distribution for the next iteration of CEM. This process is repeated num_iters times. The sum cost of an action sequence is computed by summing up the task cost function at each transition in the resulting simulated trajectory and then adding a large penalty for each constraint violating state in the simulated trajectory and

an additional penalty if the terminal state in the simulated trajectory does not have sufficient density under $\rho_{\mathcal{G}}$. For all experiments, we add a $1e6$ penalty for violating terminal state constraints and a $1e8$ penalty for violating task constraints. In practice to accelerate domain expansion to x^*, when selecting initial states x_S^j from $\bigcup_{k=0}^{j} \tilde{\mathcal{S}}\tilde{\mathcal{S}}^k$, we sort states in the safeset under $C_E^j(x)$ and use this to choose x_S^j close to x^* under $C_E^j(x)$. Note that this choice does not impact any of the theoretical guarantees.

Value Function We represent each member of the probabilistic ensemble of neural networks used to approximate $L^{\pi^j}(x)$ with a neural network with 3 hidden layers, each with 500 hidden units. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001. We use 10 epochs to learn the weights for $L^{\pi^j}(x)$.

Start State Expansion We again perform trajectory optimization using the cross entropy method and for each experiment use the same pop_size, num_elites, num_iters parameters as for solving the MPC objective. Costs for action sequences are computed by summing up $C_E^j(x)$ evaluated at each state x in the corresponding simulated trajectory, and the same mechanism is used for enforcing the terminal state constraint and task constraints as for solving the MPC objective.

A.3 Experiment Specific Parameters

Pointmass Navigation

Environment Details: We use $\psi = 0.2$ and $\sigma = 0.05$ in all experiments in this domain. Demonstration trajectories are generated by guiding the robot past the obstacle along a very suboptimal hand-tuned trajectory for the first half of the trajectory before running LQR with clipped actions on a quadratic approximation of the true cost. Gaussian noise is added to the demonstrator controller. The task horizon is set to $T = 50$.

MPC Objective Parameters: For the single start, single goal set case we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15. For all start state expansion experiments, we utilize the same popsize, num_elites, and cem_iters but utilize plan_hor = 20. For experiments we utilize $\alpha = 2$ for the kernel width parameter for density model $\rho_\alpha^{\mathcal{G}}$.

Start State Expansion Parameters: We utilize $H' = H - 5$ for all experiments (trajectory optimization horizon for exploration policy).

SAVED Baseline Experimental Parameters: We supply SAVED with 100 demonstrations generated by the same demonstration policy as for ABC-LMPC. We utilize $\alpha = 3$ and utilize the implementation from [1]. Both the value function and dynamics for SAVED are represented with a probabilistic ensemble of 5 neural networks with 3 hidden layers of 500 hidden units each. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001.

7-Link Reacher Arm

Environment Details: We use $\sigma = 0.03$ for all experiments. The state space consists of the 7 joint angles. Each link is of 1 unit in length and the goal is to control the end effector position to a 0.5 radius circle in $\mathbb{R}^2$ centered at $(3,-3)$. We do not model self-collisions but also include a circular obstacle of radius 1 in the environment which the kinematic chain must navigate around. Collisions with the obstacle are checked by computing the minimum distance between each link in the kinematic chain and the center of the circular obstacle and determining whether any link has a minimum distance from the center of the obstacle that is less than the radius of the obstacle. The task horizon is set to $T = 50$. We build on the implementation provided through [32].

MPC Objective Parameters: For the single start, single goal set case we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15. For all start state expansion experiments, we utilize the same popsize, num_elites, and cem_iters but utilize plan_hor = 20. For experiments we utilize $\alpha = 0.5$ for the kernel width parameter for density model $\rho_{\alpha}^{\mathcal{G}}$.

Start State Expansion Parameters: We utilize $H' = H - 5$ for all experiments (trajectory optimization horizon for exploration policy).

SAVED Baseline Experimental Parameters: We supply SAVED with 100 demonstrations generated by the same demonstration policy as for ABC-LMPC. We utilize $\alpha = 0.5$ and utilize the implementation from [1]. Both the value function and dynamics for SAVED are represented with a probabilistic ensemble of 5 neural networks with 3 hidden layers of 500 hidden units each. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001.

Inverted Pendulum

Environment Details: We use $\sigma = 0.5$ for all experiments. The robot consists of a single link and can exert a torque to rotate it. The state space consists of the angle and angular velocity of the pendulum. Note that there are only stable orientations, the upright orientation and downward orientation for this task, and thus for a goal set to be robust control invariant, it will likely need to be defined around the neighborhood of these orientations. The task horizon is set to $T = 40$. We define $\mathcal{G}_1$ as the goal set centered around the downward orientation and $\mathcal{G}_2$ as the goal set centered around the upright orientation. Precisely, inclusion in $\mathcal{G}_1$ is determined by determining whether the orientation of the pendulum is within 45 degrees of the downward orientation. Similarly, inclusion in $\mathcal{G}_2$ is determined by determining whether the orientation of the pendulum is within 45 degrees of the upward orientation.

MPC Objective Parameters: We utilize popsize = 600, num_elites = 40, cem_iters = 5, and plan_hor = 15. For experiments we utilize $\alpha = 2$ for the kernel width parameter for density model $\rho_{\alpha}^{\mathcal{G}}$.

Start State Expansion Parameters: We utilize $H' = H$ for all experiments (trajectory optimization horizon for exploration policy).

A.4 Controller Domain Expansion Strategy

Here we discuss how the controller domain can be expanded when the safe set and value function are updated based on samples from the exploration policy. To approximately expand $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^{k}$, we can again solve the following 1-step trajectory optimization problem:

$$\begin{aligned} \pi_{E,0:H'-1}^{j} = \underset{\pi_{0:H'-1} \in \Pi^{H'}}{\operatorname{argmin}} \quad & \mathop{\mathbb{E}}_{w_{0:H'-2}^{j}} \left[\sum_{i=0}^{H'-1} C_E^j(x_i^j, \pi_i(x_i^j)) \right] \\ \text{s.t.} \quad & x_{i+1}^j = f(x_i^j, \pi_i(x_i^j), w_i) \ \forall i \in \{0, \ldots, H'-1\} \\ & x_{H'}^j \in \bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^{k}, \ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\ & x_{0:H'}^j \in \mathcal{X}^{H'+1}, \ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \end{aligned} \tag{A.4.20}$$

For all $x_S^j \in \mathcal{SS}_{\mathcal{G}}^{j-1}$, the states $\bigcup_{k=0}^{H'} \mathcal{R}_k^{\pi_{E,0:H'-1}^{j}}(x_S^j) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi_{E,0:H'-1}^{j}}(x_S^j))$ are added to $\mathcal{SS}_{\mathcal{G}}^{j}$. The second union is included to define the value function for the composition of π^j and $\pi_{E,0:H'-1}^{j}$. This is analogous to running the exploration policy followed by running the task-directed policy π^j. Denoting the safe set where π^j is executed as $\mathcal{SS}_{\mathcal{G}}^{\pi^j} = \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi_{E,0:H'-1}^{j}}(\mathcal{SS}_{\mathcal{G}}^{j-1})) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{SS}_{\mathcal{G}}^{j-1})$, we redefine $L_{\mathcal{G}}^{\pi^j}$ as:

$$L_{\mathcal{G}}^{\pi^j}(x) = \begin{cases} \mathop{\mathbb{E}}_{w} \left[C(x, \pi^j(x)) + L_{\mathcal{G}}^{\pi^j}(f(x, \pi^j(x), w)) \right] & x \in \mathcal{SS}_{\mathcal{G}}^{\pi^j} \setminus \mathcal{G} \\ \mathop{\mathbb{E}}_{w} \left[C(x, \pi_{E,0:H'-1}^{j}(x)) + L_{\mathcal{G}}^{\pi^j}(f(x, \pi_{E,0:H'-1}^{j}(x), w)) \right] & x \in \mathcal{SS}_{\mathcal{G}}^{j} \setminus \mathcal{SS}_{\mathcal{G}}^{\pi^j} \\ 0 & x \in \mathcal{G} \\ +\infty & x \notin \mathcal{SS}_{\mathcal{G}}^{j} \end{cases} \tag{A.4.21}$$

This means that trajectories from the exploration policy can spend more time outside of the safe set. In either case, the safe set remains robust control invariant.

Thus, each iteration j is split into two phases. In the first phase, π^j is executed and in the second phase, $\pi_{E,0:H'-1}^{j}$ is executed. This procedure provides a simple algorithm to expand the policy's domain $\mathcal{F}_{\mathcal{G}}^{j}$ while still maintaining its theoretical properties.