

Article

CACTO-BIC: Scalable Actor–Critic Learning via Biased Sampling and GPU-Accelerated Trajectory Optimization

Elisa Alboni ^{1,*} , Pietro Noah Crestaz ^{1,2}, Elias Fontanari ¹ and Andrea Del Prete ¹ 

¹ Department of Industrial Engineering, University of Trento, Via Sommarive, 9, 38123 Trento, Italy; pietronoah.crestaz@unitn.it (P.N.C.); elias.fontanari@unitn.it (E.F.); andrea.delprete@unitn.it (A.D.P.)

² Laboratory for Analysis and Architecture of Systems, Université de Toulouse, CNRS, 7 Av. du Colonel Roche, 31400 Toulouse, France

* Correspondence: elisa.alboni@unitn.it

Abstract

Trajectory Optimization (TO) and Reinforcement Learning (RL) offer complementary strengths for solving optimal control problems. TO efficiently computes locally optimal solutions but can struggle with non-convexity, while RL is more robust to non-convexity at the cost of significantly higher computational demands. The Continuous Actor–Critic with Trajectory Optimization (CACTO) algorithm was introduced to combine these advantages by learning a warm-start policy that guides the TO solver towards low-cost trajectories. However, scalability remains a key limitation, as increasing system complexity significantly raises the computational cost of TO. This work introduces CACTO with Biased Initial Conditions (CACTO-BIC) to address these challenges. CACTO-BIC improves data efficiency by biasing initial-state sampling leveraging a property of the value function associated with locally optimal policies; moreover, it reduces computation time by exploiting GPU acceleration. Empirical evaluations show improved sample efficiency and faster computation compared to CACTO. Comparisons with Proximal Policy Optimization (PPO) demonstrate that our approach can achieve similar solutions in less time. Finally, experiments on the AlienGO quadruped robot demonstrate that CACTO-BIC, deployed as a receding-horizon planner using a reduced 15D robot model, can scale to high-dimensional systems and is suitable for real-time applications.

Keywords: trajectory optimization; reinforcement learning; quadruped locomotion

1. Introduction

Trajectory Optimization (TO) is a widely used and flexible technique for solving robotic control problems. In TO, the high-level task is formulated as a constrained optimal control problem (OCP), where the optimization variables are the system's state and control trajectories. Constraints enforce compliance with system dynamics and kinematics, actuator limits, and task-specific requirements. However, OCPs are typically highly non-convex, making gradient-based solvers prone to converging to poor local minima. While global methods based on the Hamilton–Jacobi–Bellman equation or Dynamic Programming [1] exist, their applicability is limited by the curse of dimensionality.

Deep Reinforcement Learning (RL) has emerged as an alternative framework, particularly for continuous-state and -action spaces. Algorithms such as DDPG [2], SAC [3], and PPO [4] have demonstrated strong performance in robotic control tasks. Due to their exploratory nature, RL methods are generally less sensitive to local minima but they typically suffer from high sample complexity and long training times.



Academic Editor: Sunan Huang

Received: 30 May 2026

Revised: 18 July 2026

Accepted: 28 July 2026

Published: 30 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

To overcome the complementary limitations of TO and RL, hybrid approaches combining the two have recently gained significant research attention. A popular choice is to rely on *TO imitation*: policies are trained to mimic TO or model predictive control (MPC) solutions—through value-based or action-based imitation—to reduce online computational costs and to leverage sensor feedback [5,6]. However, these methods neither improve the TO solution's quality, nor guarantee constraint satisfaction. Accounting for policy approximation errors in the TO problem can lead to better results [7,8], but it inherits the same limitations.

Other methods use learned policies or value functions to warm-start TO or to define terminal costs, thereby accelerating optimization and guiding the solver toward improved solutions [9,10]. While effective for TO, these approaches provide no benefits for RL training.

An increasingly prominent class of approaches embeds TO directly within the RL framework to improve training efficiency. These methods can be classified based on where TO is included: *post-policy*, *pre-policy*, or as a *residual* policy. *Post-policy* methods evaluate TO after the policy. Some methods learn cost or constraint parameters [11,12], leveraging MPC's safety and stability guarantees, but require solving TO online and face convergence challenges. Others use TO as an actor and learn its terminal cost [13,14], accelerating training and improving constraint handling, yet potentially yielding suboptimal solutions or neglecting sensor feedback. Another variant initializes TO with an RL policy [15,16], improving convergence speed and solution quality, yet still does not exploit sensor feedback. In *pre-policy* methods, TO generates reference trajectories or auxiliary information that is input into the RL policy, effectively speeding up training [17]. *Residual* methods instead learn a residual policy to improve TO-generated control inputs with learned corrections [18]. While both *pre-policy* and *residual* methods accelerate training, they require online TO and depend on its ability to find high-quality solutions; moreover, learned policies may violate constraints even when TO does not.

In this work, we extend CACTO [15,19], a *post-policy* algorithm that exploits the interplay between TO and RL to accelerate training. The actor policy generates the initial guess for TO, leveraging the exploratory nature of RL to avoid convergence to poor local minima, while TO guides the learning process of the RL agent.

The primary limitation of CACTO is scalability. As the system complexity increases, TO becomes more expensive, and actor-critic training requires more iterations to converge. We investigate strategies to reduce the computational burden in both the TO and RL phases. Our main contributions are:

- A method to identify state-space regions where an improvement in the actor policy is more likely.
- A new version of the CACTO algorithm, called CACTO-BIC, that exploits biased initial conditions (BIC) and GPU-based computation to achieve improved sample and time efficiency.
- A JAX-based [20] open-source implementation of CACTO-BIC that exploits GPUs to solve TO problems and train neural networks.
- The first validation of CACTO through hardware experiments on a quadruped robot (using a reduced 15D robot model).

2. Background

This section summarizes the latest formulation of CACTO (originally called CACTO-SL [19]), an algorithm for solving finite-horizon discrete-time optimal control problems such as:

$$\begin{aligned} \underset{X, U}{\text{minimize}} \quad & L(X, U) \triangleq \sum_{k=0}^{T-1} l_k(x_k, u_k) + l_T(x_T) \\ \text{subject to} \quad & x_{k+1} = f_k(x_k, u_k), \quad k = 0, \dots, T-1 \\ & |u_k| \leq u_{\max}, \quad k = 0, \dots, T-1 \\ & x_0 = x_{\text{init}} \end{aligned} \quad (1)$$

where $X = \{x_0, \dots, x_T\}$ and $U = \{u_0, \dots, u_{T-1}\}$ are the state and control sequences, with $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$. The cost $L(\cdot)$ combines running costs l_k and a terminal cost l_T , while the constraints enforce system dynamics, control bounds, and initial conditions.

CACTO begins by solving N TO problems from randomly sampled initial states, using standard warm-starting (e.g., setting all states to x_{init} and controls to zero as initial guess). As the OCP horizon is finite, the actor policy and the critic value are time-dependent, so also the initial time is randomized in each TO instance and the time is appended to the state vector, $\tilde{x} = [x, t]$. For each state of each optimized trajectory, CACTO computes the partial K -step cost-to-go $\bar{V}$, where K is a user-defined parameter representing the Temporal Difference lookahead horizon. The value function's gradient $\bar{V}_x$ is also computed, using the backward pass of Differential Dynamic Programming (DDP) [21]. These values $(\bar{V}, \bar{V}_x)$, together with the associated state, control, and state after K steps, are stored in a replay buffer.

Afterwards, the critic and actor neural networks are trained for m iterations using mini-batches of size N_B sampled from the replay buffer. The number of training iterations m increases at each cycle according to a predefined schedule M_{list} . The critic approximates the value function and its gradient by matching $\bar{V}$ and $\bar{V}_x$:

$$\underset{\theta^V}{\text{minimize}} \quad (\bar{V} - V(\tilde{x}|\theta^V))^2 + k_s (\bar{V}_x - S_x V_{\tilde{x}}(\tilde{x}|\theta^V))^2 \quad (2)$$

where $V(\cdot)$, $V_{\tilde{x}}(\cdot)$ and θ^V are the critic network, its gradient (with respect to $\tilde{x}$) and its parameters, k_s is the weight of the gradient term, and S_x is the selection matrix to exclude the partial derivative of V with respect to time. The actor is updated by minimizing the Q-value:

$$\underset{\theta^\mu}{\text{minimize}} \quad l(\tilde{x}, \mu(\tilde{x}|\theta^\mu)) + V_{\theta^V}(f(\tilde{x}, \mu(\tilde{x}|\theta^\mu))) \quad (3)$$

where $\mu(\cdot)$ and θ^μ are the actor network and its parameters. The improved actor policy then generates rollouts—sequences of actions produced by iteratively applying the policy in the environment from the new initial state over the finite horizon—that warm-start the subsequent TO problems, closing the loop between TO and RL.

In [15], CACTO was shown to outperform other RL algorithms such as DDPG [2] and PPO [4] as a warm-start provider in terms of training time.

3. Biased Initial Sampling

Identifying regions of the state space with high potential for policy improvement is a challenging problem.

Exploration strategies in RL can be broadly categorized into undirected and directed approaches. Undirected exploration relies on stochastic action selection, such as ϵ -greedy policies, to explore the state space without explicitly accounting for uncertainty or novelty. Directed exploration, in contrast, uses signals derived from the agent's learning process or auxiliary models to guide behavior toward less familiar or more uncertain regions. A common class of directed methods employs intrinsic rewards or exploration bonuses, including count-based or approximate count methods [22], uncertainty metrics [13], and curiosity-driven approaches such as prediction-error bonuses and random network distillation [23,24]. Several works have also explored initial-state or restart-based exploration. These approaches often assign scores to states to prioritize which ones to explore further. The criteria for scoring include the system's sensitivity [25], the familiarity [26], the uncertainty [27], or the TD error [28]. Ref. [29] proposes a structured replay buffer that groups states by task relevance and prioritizes sampling from unmastered sub-tasks. When available, prior or expert knowledge can also be leveraged to further guide exploration.

We address the exploration problem by leveraging the insight that the value function $\bar{V}$ associated with a locally optimal policy, i.e., a policy that cannot be improved by locally changing the value of its parameters, is generally *piecewise continuous*.

3.1. Motivating Example: Discontinuous Value Function

Let us illustrate how the structure of the value function associated with a locally optimal policy can help us tackle the exploration problem. We focus on a toy problem with a 1D state, single integrator dynamics, and a cost with two local minima (see Figure 1). Solving TO problems with a naive initial guess highlights the presence of two basins of attraction, $\mathcal{R}_1$, and $\mathcal{R}_2$, each corresponding to a different local minimum (see Figure 1).

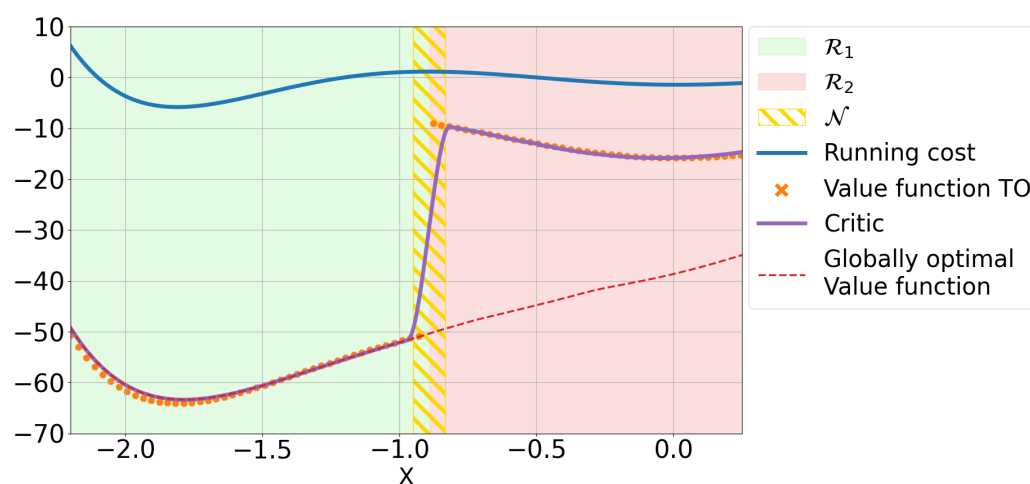


Figure 1. Cost and value obtained with TO using a naive initial guess. The critic smooths the value's discontinuities. The globally optimal value function is added for reference.

Within each basin, the real value $\bar{V}$ is continuous, but it is discontinuous at the shared boundary $\mathcal{R}_1 \cap \mathcal{R}_2 \triangleq \partial\mathcal{R}$. In the neighborhood of $\partial\mathcal{R}$, $\bar{V}$ is lower (i.e., better) in $\mathcal{R}_1$ than in $\mathcal{R}_2$.

After training the critic with the first batch of TO episodes, the network smooths out the discontinuity (see Figure 1) in a region $\mathcal{N}$ around $\partial\mathcal{R}$, where the critic either underestimates or overestimates the true value $\bar{V}$:

$$V(\bar{x}|\theta_V) > \bar{V}(\bar{x}) \quad \forall \bar{x} \in \mathcal{N} \cap \mathcal{R}_1 \quad (4)$$

$$V(\bar{x}|\theta_V) < \bar{V}(\bar{x}) \quad \forall \bar{x} \in \mathcal{N} \cap \mathcal{R}_2 \quad (5)$$

Therefore, in $\mathcal{N}$ the critic's gradient will point towards $\mathcal{R}_2$. Due to this gradient, during the policy improvement phase, the actor can improve in $\mathcal{N} \cap \mathcal{R}_2$, learning to steer the state toward $\mathcal{R}_1$ rather than $\mathcal{R}_2$ (Figure 2). This example reveals that the regions near the value function's discontinuities have great potential for policy improvement.

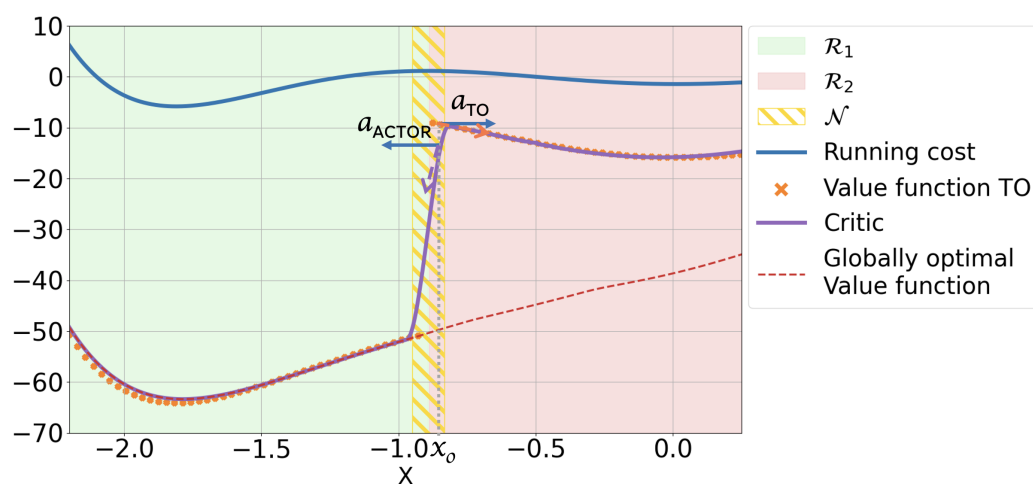


Figure 2. Dashed arrows indicate directions of decreasing value according to the critic and TO value function. Solid arrows represent the selected control actions by the TO solver and the trained actor at x_0 , in the region $\mathcal{N}$. The actor's action points toward the better minimum.

3.2. General Case: Discontinuous Value Function

The phenomenon shown in Section 3.1 frequently occurs in problems with multiple local minima. Each local minimum defines a basin of attraction where the value function is continuous, while discontinuities typically appear at the boundaries between basins. These discontinuities highlight regions with great potential for policy improvement, where the actor can learn a better policy and provide an effective warm-start that guides the TO solver toward higher-quality solutions. In contrast, sampling initial states far from such discontinuities likely leads the actor to simply imitate TO.

3.3. Detecting Informative Regions via Critic Uncertainty

Approximating the value function close to the discontinuities is particularly challenging for the critic network, which cannot accurately represent abrupt changes due to its continuous activation functions. As a result, the critic tends to make larger errors near these boundaries. We suggest leveraging the uncertainty estimation of the critic's output to identify the value's discontinuities. An additional neural network, referred to as *std-critic*, is introduced to predict the standard deviation of the critic: the network is encouraged to output small values when the critic accurately estimates the value function and to increase its output when the critic exhibits large prediction errors. Consequently, states associated with high uncertainty are characterized by higher *std-critic*'s values, which enables their identification and prioritization during subsequent training.

Following [30], this network is trained at the end of each actor–critic update phase by minimizing the negative log-likelihood of a normal distribution. In particular, even though

V is a deterministic function in our settings, we let the network interpret the target value $V(\tilde{x}|\theta^V)$ as a Gaussian variable with mean $\bar{V}$ and standard deviation $V^{\text{std}}(\tilde{x}|\theta^{\text{std}})$:

$$p(V(\tilde{x}|\theta^V) | \bar{V}) = \frac{1}{\sqrt{2\pi} V^{\text{std}}(\tilde{x}|\theta^{\text{std}})} \exp\left(-\frac{(\bar{V} - V(\tilde{x}|\theta^V))^2}{2 V^{\text{std}}(\tilde{x}|\theta^{\text{std}})^2}\right) \quad (6)$$

where $V^{\text{std}}(\cdot)$ and θ^{std} are the *std-critic* network and its parameters. Taking the negative logarithm of the likelihood (6) and neglecting constant terms yields the following objective:

$$\underset{\theta^{\text{std}}}{\text{minimize}} \quad \ln(V^{\text{std}}(\tilde{x}|\theta^{\text{std}})) + \frac{1}{2} \frac{(\bar{V} - V(\tilde{x}|\theta^V))^2}{V^{\text{std}}(\tilde{x}|\theta^{\text{std}})^2}. \quad (7)$$

The first term of this loss pushes V^{std} to be small everywhere, while the second term pushes V^{std} to increase when the critic's error is large.

The initial states with the highest potential for policy improvement are then selected according to their predicted uncertainty $V^{\text{std}}(\tilde{x}|\theta^{\text{std}})$.

Even though biased sampling could theoretically compromise the convergence of the algorithm because it could prevent complete coverage of the state space, in practice we have seen that it leads to better results.

Remark 1. Critic uncertainty does not necessarily indicate a discontinuity. High uncertainty may simply reflect regions where the critic has not yet learned accurately, or even out-of-distribution states. In these cases, focusing sampling in such regions is still beneficial, as it drives learning toward poorly modeled areas. If the observed uncertainty is due to insufficient data rather than a true discontinuity, it will progressively decrease as additional samples are collected and incorporated into training.

3.4. Algorithm Overview

An overview of the CACTO-BIC (Biased Initial Conditions) algorithm is illustrated in Figure 3, while a more detailed description is provided in Algorithm 1.

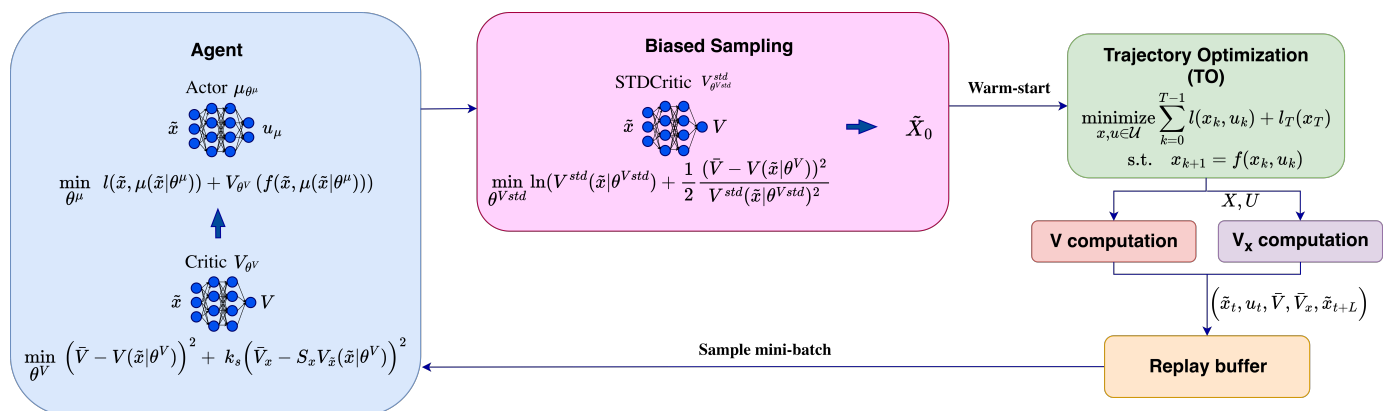


Figure 3. Overview of CACTO-BIC with biased initial-state sampling.

During the first iteration, as in the original CACTO framework [15], initial states for the TO problems are sampled uniformly at random. From the second iteration onward, a set of $10N$ candidate initial states is sampled and ranked based on V^{std} . The top N samples are then selected for the next TO batch. Moreover, since trajectories starting from these initial states provide more information, we can reduce the number of problems solved from the second iteration (see Section 5.1 for details).

Algorithm 1: CACTO-BIC

Input: dynamics $f(\cdot, \cdot)$, running cost $l_k(\cdot, \cdot)$, terminal cost $l_T(\cdot)$, T , $N_{episode}$, N_B , K , τ , N , M_{list} , k_S

Output: trained control policy $\mu(\tilde{x})$

$\theta^V \leftarrow \text{random}$, $\theta^\mu \leftarrow \text{random}$, $\theta^{\text{std}} \leftarrow \text{random}$, $\theta^{V'} \leftarrow \theta^V$ $cycle_count \leftarrow 0$

Initialize replay buffer R

for $episode \leftarrow 1$ **to** $N_{episode}$ **do**

if $episode < N$ **then**

$\tilde{x}_0 \leftarrow \text{random}$

else

 Sample $\{\tilde{x}_0^{(i)}\}_{i=1}^{10}$

$\tilde{x}_0 \leftarrow \arg \max_{i \in \{1, \dots, 10\}} V_{\text{std}}(\tilde{x}_0^{(i)} | \theta^{\text{std}})$

if $episode < N$ **then**

for $t \leftarrow \tilde{x}_0[n+1]$ **to** $T-1$ **do** (ICS warm-start)

$u_t \leftarrow 0$

$\tilde{x}_{t+1} \leftarrow \tilde{x}_0$

else

for $t \leftarrow \tilde{x}_0[n+1]$ **to** $T-1$ **do** (policy rollout)

$u_t \leftarrow \mu(\tilde{x}_t | \theta^\mu)$

$\tilde{x}_{t+1} \leftarrow [f(\tilde{x}_t[:n], u_t), t+1]$

$x_0 \leftarrow \tilde{x}_0[:n]$, $x \leftarrow \tilde{x}[:, :n]$

$x, u, l \leftarrow \text{SolveTO}(x_0, x, u)$

$V_x = \text{BackwardPass}(x, u)$ (Compute value function's gradient)

$\tilde{x}[:, :n] \leftarrow x$

for $t \leftarrow \tilde{x}_0[n+1]$ **to** T **do**

 Compute partial cost-to-go: $V_t = \sum_{j=t}^{\min(t+K, T)} l_j$

 Store the transition in the replay buffer: $R \leftarrow (\tilde{x}_t, V_t, \bar{V}_{x,t}, \tilde{x}_{\min(t+K, T)})$

if $episode \% N = 0$ **then**

for $m \leftarrow 1$ **to** $M_{list}[cycle_count]$ **do** (critic & actor update)

 Sample minibatch of N_B transitions: $(\tilde{x}_i, V_i, \bar{V}_{x,i}, \tilde{x}_{\min(i+K, T)})$

 Compute cost-to-go: $\bar{V}_i = \begin{cases} V_i & \text{if } i+K > T \\ V_i + V'(\tilde{x}_{i+K}) & \text{otherwise} \end{cases}$

 Update critic by minimizing the loss over θ^V :

$L_c = \frac{1}{N_B} \sum_{i=1}^{N_B} (\bar{V}_i - V(\tilde{x}_i | \theta^V))^2 + k_S (\text{logsym}(\bar{V}_{x,i}) - \text{logsym}(S_x V_{\tilde{x}}(\tilde{x}_i | \theta^V)))^2$

 Update actor by minimizing the loss over θ^μ :

$L_a = \frac{1}{N_B} \sum_{i=1}^{N_B} l(x_i, \mu(\tilde{x}_i | \theta^\mu)) + V(f(x_i, \mu(\tilde{x}_i | \theta^\mu)) | \theta^V)$

 Update target critic: $\theta^{V'} \leftarrow \tau \theta^V + (1 - \tau) \theta^{V'}$

for $m \leftarrow 1$ **to** $M_{list}[cycle_count]$ **do** (std-critic update)

 Sample minibatch of N_B transitions: $(\tilde{x}_i, V_i, \bar{V}_{x,i}, \tilde{x}_{\min(i+K, T)})$

 Compute cost-to-go: $\bar{V}_i = \begin{cases} V_i & \text{if } i+K > T \\ V_i + V'(\tilde{x}_{i+K}) & \text{otherwise} \end{cases}$

 Update std-critic by minimizing the loss over θ^{std} :

$L_{stdc} = \ln(V^{\text{std}}(\tilde{x} | \theta^{\text{std}})) + \frac{1}{2} \frac{(\bar{V} - V(\tilde{x} | \theta^V))^2}{V^{\text{std}}(\tilde{x} | \theta^{\text{std}})^2}$

$cycle_count = cycle_count + 1$

4. GPU-Based Computation

To reduce the computation time of the algorithm by leveraging GPUs, the entire framework was migrated to JAX (version 0.4.20), a Python library for accelerator-oriented array computation and program transformation [20]. The migration is beneficial in two ways: first, migrating the neural network training to a GPU—accounting for about 90%

of CACTO's total computation time—yields significant speedups; second, performing TO directly on a GPU eliminates any CPU-GPU data-transfer overhead, in addition to reducing TO's computation time, which significantly grows with the complexity of the system.

GPU acceleration is not required during online deployment. This is because the learned policy has a lightweight architecture and can be executed with low latency on CPU hardware. Consequently, while GPU acceleration is highly beneficial for reducing the computational burden of the learning pipeline, it would have a negligible impact on the execution time of the deployed controller.

Migrating to GPU-based computation required some adaptations to meet the constraints of parallel processing: GPUs require fixed-size arrays and uniform computation across batches to enable efficient vectorization, leading to the following two key challenges.

4.1. Fixed Number of Iterations

Because GPUs execute batched computations in parallel, all TO problems in a batch must undergo the same number of optimization iterations. Consequently, the maximum number of iterations, max_iter , becomes a crucial hyperparameter. Setting this value too low may lead to premature termination, and hence to the collection of too few or insufficiently informative samples. In contrast, setting it too large can cause longer runtimes, since hard-to-converge problems dominate the batch's overall computation time, slowing the entire pipeline. Moreover, the ideal number of iterations may vary as the policy improves, as better initial guesses generally lead to faster convergence.

To determine the maximum number of iterations, we suggest solving a large set of problems using a naive warm-start and a high iteration limit ($max_iter = 1000$), generating a dataset of the iterations needed to converge. The maximum number of iterations can then be set to the 99th percentile of iteration counts for the first iteration of CACTO (where the naive warm-start is used) and to the 50th percentile for subsequent iterations (where CACTO's actor provides the warm-start).

4.2. Regularization for Matrix Inversion

In iLQR [21], it is crucial to regularize the Hessian of the value function so that it is positive definite.

The standard regularization consists in increasing the local control-cost Hessian with a diagonal term: $\tilde{Q}_{uu} = Q_{uu} + \mu I_m$, where μ plays the role of a Levenberg–Marquardt parameter. During the backward pass, if the line search fails, μ is increased and the backward pass is retried; otherwise, μ is decreased.

This approach is computationally efficient when solving problems sequentially on a CPU. However, it becomes impractical when solving multiple problems in parallel on a GPU, where such iterative tuning can significantly slow down the batched computation.

This issue is addressed through regularization based on the eigenvalue decomposition. The eigenvalues S and corresponding eigenvectors W of the state-cost Hessian and the control-cost Hessian are computed, the eigenvalues are clipped from below by a user-defined constant $\epsilon > 0$, $S' = \max(S, \epsilon)$, and the matrices are reconstructed:

$$Q_+ = W \text{diag}(S') W^T, \quad Q_{psd} = \frac{1}{2}(Q_+ + Q_+^T) \quad (8)$$

This procedure handles poorly conditioned Hessians, while avoiding additional iterative steps. As a result, it is better suited for large-batch processing on GPUs.

4.3. Implementation Details

The system dynamics and cost functions, implemented using the CasADi library [31] in CACTO-SL [19], were converted into JAX-compatible functions through the Jaxadi

library [32]. For TO, we replaced the previous solver with an iLQR implementation from the Trajax library [33], which provides the additional benefit of computing the gradient of the value function (used for training the critic network) while solving the optimal control problem. The neural networks were implemented using the Flax library [34], allowing the entire pipeline to remain on the GPU. Our new implementation is open-source and available on the GitHub page of the project: https://github.com/idra-lab/cacto/tree/main_jax (accessed on 20 July 2026).

5. Results

This section presents our evaluation of CACTO-BIC. First, we assess the impact of the biased initial-state sampling on data efficiency (Section 5.1). Second, we analyze the computational benefits of the GPU-based implementation (Section 5.2). Third, we compare CACTO-BIC with a state-of-the-art RL algorithm (Section 5.3). Finally, we demonstrate the scalability of the approach through experiments on a high-dimensional quadruped robot, AlienGO [35] (Section 5.4).

5.1. Biased Initial State Sampling

We evaluate the proposed exploration method on the same benchmark scenarios used in [15,19]. The task consists in minimizing the distance between the robot's end-effector and a target, while avoiding three elliptical obstacles (encoded with large penalties) and minimizing control effort. An additional reward is provided in the neighborhood of the target, as shown in Figure 4.

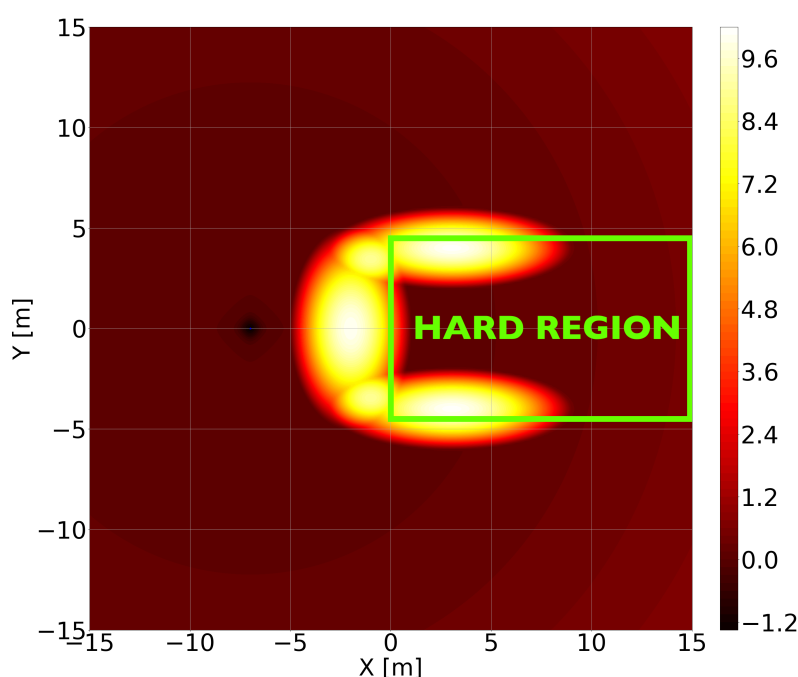


Figure 4. Cost function excluding the control effort term, with target set at $[-7, 0]$, adapted from ref. [19].

When the system starts from the *Hard Region* (highlighted in Figure 4), it becomes challenging for a gradient-based solver to converge to the globally optimal solution.

We consider three systems of increasing complexity: a point mass with state $(x, y, v_x, v_y, t) \in \mathbb{R}^5$ and control $(a_x, a_y) \in \mathbb{R}^2$, a jerk-controlled version of the *Dubins car model* [36] with state $(x, y, \theta, v, a, t) \in \mathbb{R}^6$ and control $(\omega, j) \in \mathbb{R}^2$, and a three-degree-of-freedom (DoF) planar manipulator with a seven-dimensional state space and a three-dimensional control input. We compare three algorithms:

1. CACTO as presented in [19];
2. CACTO-BIC: CACTO with biased initial-state sampling and a reduced number of TO episodes (25%) from the second iteration onward;
3. CACTO with reduced TO episodes (as CACTO-BIC), but without biased initial-state sampling.

All algorithms run for the same number of learning iterations (i.e., updates of the neural networks). Figure 5 shows the median (across five runs) of the mean cost (across initial conditions) as a function of the number of TO episodes. The results show that CACTO-BIC achieves comparable performance using 30–40% of the number of TO episodes. However, this improvement comes with increased training time due to the additional std-critic network. As a result, with CACTO-BIC, the percentage of computation time devoted to training the networks rises to 93–94%, up from 80–82%, due to both the additional training overhead and the reduced number of TO problems solved, which shortens the TO phase. This motivates moving to the GPU.

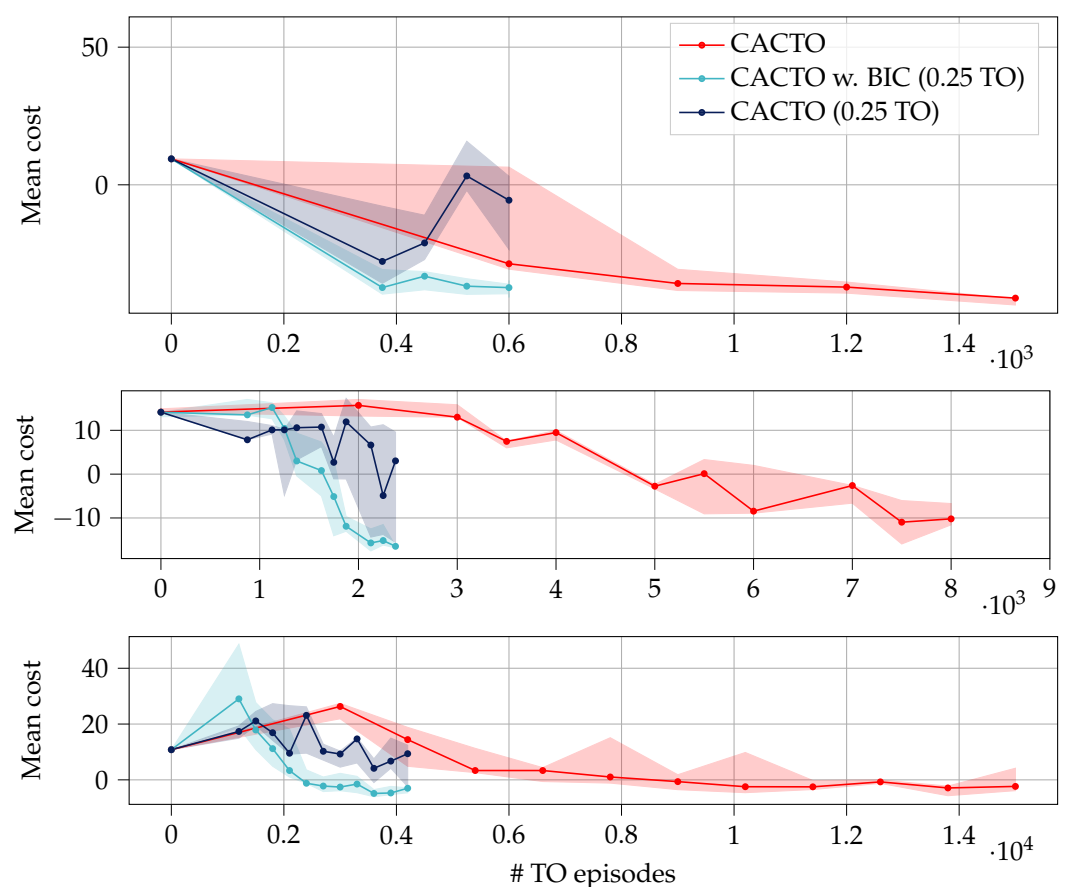


Figure 5. Median (across 5 runs) of the mean cost (across initial conditions) starting from the Hard Region for the point mass (**top**), the *Dubins car* (**center**), and the manipulator (**bottom**). Shaded areas represent first and third quartiles. Data are sampled every n updates. When multiple measurements occur at the same TO episode count, the last one is plotted.

Figure 6 reports representative trajectories obtained for the point mass and Dubins car environments. The trajectories generated using the CACTO-BIC policy as warm-starts are qualitatively comparable to those obtained using the original CACTO pipeline. This is consistent with the quantitative results, where both approaches achieve similar final costs after training convergence. These results indicate that the proposed method preserves the solution quality of the original approach while significantly reducing the computational

effort required to obtain effective initializations for TO, enabling the application of the framework to more complex robotic systems and scenarios.

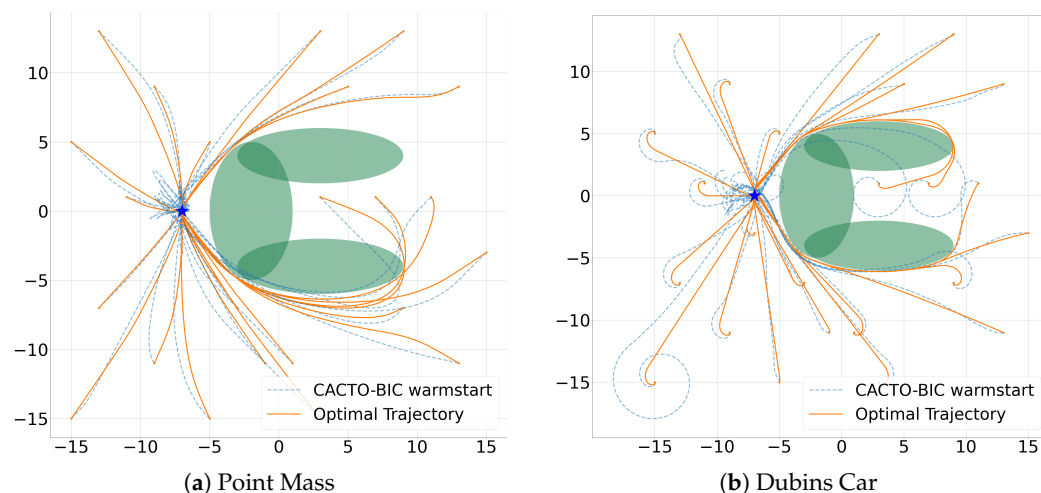


Figure 6. Optimal trajectories obtained using CACTO-BIC warm-starts. The dashed lines represent the CACTO-BIC policy rollouts. The target is indicated by a blue star, while the green areas represent the obstacles.

Additional qualitative results, including trajectory animations for the Dubins car and 3-DOF manipulator tasks, are provided in Supplementary Video S1: https://www.youtube.com/watch?v=3_e7sVgtJAI (accessed on 20 July 2026).

To further assess the sample-efficiency benefits of BIC, we compared BIC against existing exploration and prioritized sampling strategies, including a TD-error-based sampling method, Prioritized Restart (PR) [28], which prioritizes revisiting regions where the value function exhibits larger prediction errors; a density estimation-based exploration approach, Density Estimation-based State-Space Coverage Acceleration (DESSCA) [26]; and an approach that selects promising start points based on co-states, which quantify the sensitivity of the final cost to state deviations [37].

As shown in Figure 7, BIC consistently achieves lower median costs across the evaluated tasks while exhibiting a smaller interquartile range compared with the baseline methods at convergence. The superior average performance and the improved consistency suggest that the learned V_{std} provides a more effective estimation of informative regions in the state space, allowing the algorithm to allocate additional samples to regions where further exploration is expected to yield the greatest improvement. This advantage arises from the fact that the other exploration strategies do not explicitly account for the fact that improvements in warm-start quality arise only where the actor policy outperforms the TO solution. When using a locally optimal policy, this occurs in specific state-space regions that can be effectively identified through value function uncertainty estimation.

5.2. Computational Efficiency of GPU Computation

First, we assessed the speedup achievable when solving a batch of TO problems on a GPU for different batch sizes: [10, 50, 100, 250, 500, 1000, 5000, 10,000].

Figure 8 shows that while the CPU computation time scales approximately linearly with batch size, the GPU benefits from parallelization: the time per problem decreases as the batch size increases, eventually saturating, demonstrating the advantage of GPU-based TO.

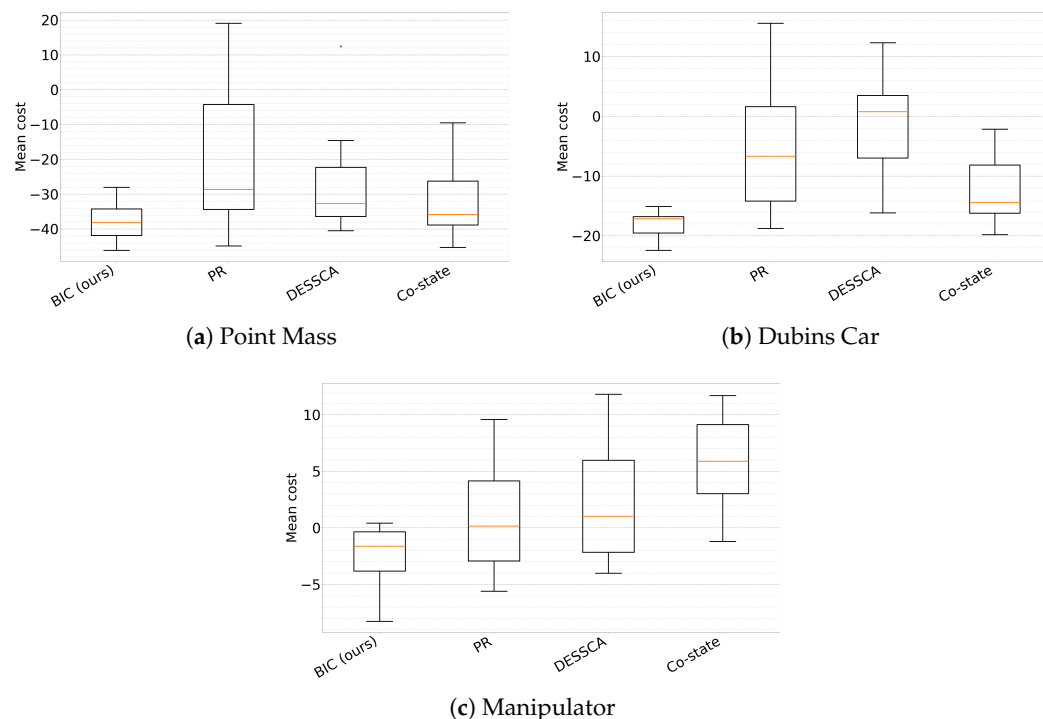


Figure 7. Comparison of BIC with existing exploration and sample-efficiency methods (PR, DESSCA, and Co-state) on the point mass, Dubins car and manipulator environments. The results are computed over 10 independent runs and reported as the distribution of average costs across initial conditions at convergence. The boxes indicate the interquartile range, and the centre lines the median.

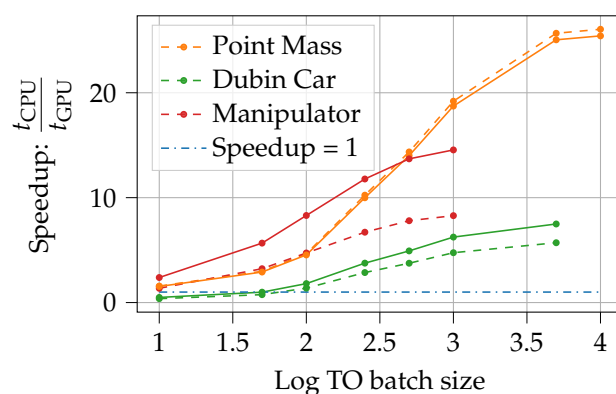


Figure 8. Speedups for TO on GPU (w.r.t. CPU) for the point mass, the *Dubins car*, and the manipulator. Dashed lines represent speedups with random initial states and the maximum number of iterations, while solid lines represent speedups with initial conditions selected using CACTO-BIC and the maximum number of iterations set as in the two versions.

We now analyze the speedup achieved through the new GPU-accelerated implementation of CACTO-BIC. The number of TO problems solved in parallel was 300, 500, and 550 in the first iteration for the point mass, Dubins car and manipulator, respectively. From the second iteration onward, the batch size was reduced to 25%. The initial time is set to 0 in each TO instance. Table 1 reports the computation times required to perform the same updates using three versions of CACTO-BIC: a single-thread CPU version, a multi-thread CPU version, and the novel GPU version.

This test was performed on a workstation equipped with an AMD Ryzen 9 7950X CPU with 192 GB of RAM, and an NVIDIA RTX 6000 GPU with 48 GB VRAM, running on Ubuntu 22.04.

The results show that using 10 cores to parallelize the TO problems has little effect on the total computation time as most time is spent training the neural networks. The novel GPU version of CACTO-BIC instead achieves remarkable speedups ($\approx 30\times$ for the point mass and Dubins car, and $\approx 56\times$ for the manipulator).

Table 1. Comparison between CPU and GPU versions of CACTO-BIC. GPU runtimes do not include the warm-up phase required for the JIT compilation of the TO solver.

System (nb. of Updates)	CPU		GPU
	1 Core	10 Cores	
Point Mass (30,000)	23 min	22 min	44 s
Dubins Car (200,000)	3 h 44 min	3 h 34 min	7 min
Manipulator (330,000)	5 h 35 min	5 h 15 min	6 min

Although GPUs have far more cores than CPUs, the resulting speedups are smaller than this hardware difference suggests. Several factors contribute to this limitation. First, the TO problems are solved in batches on the GPU. Therefore a few hard instances may dominate the total computation time and cap the achievable speedup. Second, computations rely on reduced numerical precision. Although this improves raw throughput, it can introduce additional numerical errors that affect the convergence behavior of the trajectory optimization solver. In our experiments, this effect mainly manifests as an increase in the number of iterations required to reach convergence, reducing the effective speedup obtained from reduced-precision computations. However, once convergence is achieved, we do not observe significant degradation in the quality of the resulting trajectories. It should be noted that the GPU implementation relies on a different solver than the full-precision implementation; therefore, the observed behavior cannot be attributed exclusively to reduced numerical precision. Complete ablation isolating the impact of precision would require a full-precision GPU implementation with identical solver settings, which was beyond the scope of this work.

By analyzing the time spent in the TO phase (creating the warm-start and solving TO problems) and in the RL phase (training networks), we observe that their relative contribution varies across systems, but remains comparatively balanced. Specifically, the TO and RL phases account for approximately 42% and 53% of the total time in the point mass, 61% and 37% in the Dubins car, and 22% and 75% in the manipulator. It follows that the speedup comes mainly from training the networks on the GPU, which is $51 - 85\times$ faster than on the CPU in our tests. In contrast, the speedup in the TO phase is modest ($3 - 5\times$) for the Dubins car and point mass, but it becomes significant ($19\times$) for the most complex system.

5.3. Comparison with PPO

To evaluate CACTO-BIC w.r.t. state-of-the-art RL, we selected Proximal Policy Optimization (PPO) [4] as a benchmark. For a fair comparison, we employed the fully GPU-based implementation of PPO provided by BRAX [38]. Our analysis focuses on two aspects: first, learning a warm-start policy for a problem that exhibits local minima; second, learning a control policy for a classic benchmark problem, a customized version of the *Reacher* environment. We evaluate policy quality as a function of computation time rather than the more common RL metric based on environment interactions. This is because CACTO relies on TO-generated trajectories to train the networks, which are more computationally expensive than standard RL samples. Consequently, using interaction counts would lead to an unfair comparison, favoring CACTO. Figure 9 reports the mean cost obtained using

CACTO-BIC's and PPO's policies as warm-starts for TO in the point mass and manipulator environments. CACTO-BIC converged in one-third of the time in the first environment and in just 7% in the second one.

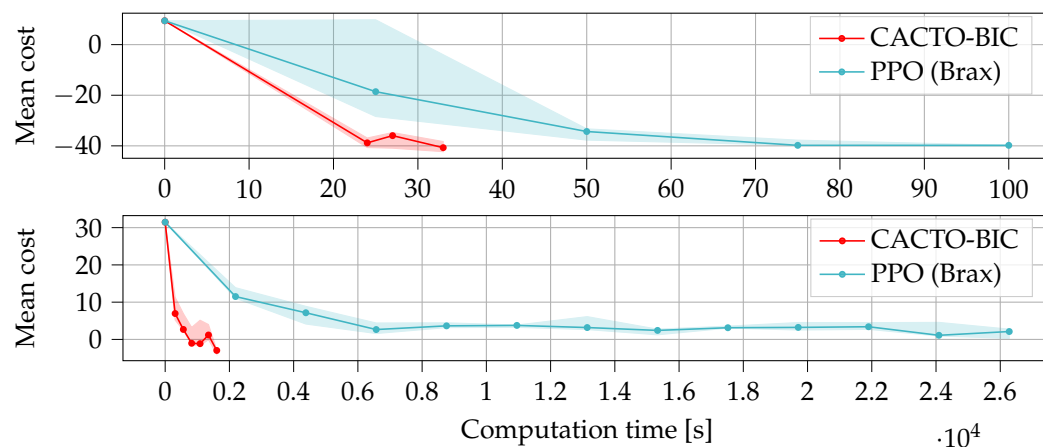


Figure 9. Warm-start provider comparison: Median (across 5 runs) of the mean cost (across initial conditions) starting from the Hard Region for the point mass (**top**), and the manipulator (**bottom**). Shaded areas represent first and third quartiles.

In Figure 10 we compare CACTO-BIC and PPO in the Reacher Environment. Also in this case, CACTO-BIC achieves a similar cost in $\approx 10\%$ of the computation time.

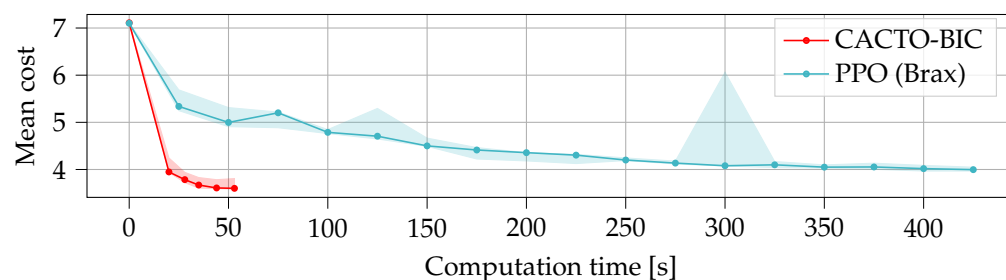


Figure 10. Policy comparison: Median (across 5 runs) of the mean cost (across initial conditions) for a customized *Reacher* environment. Shaded areas represent first and third quartiles.

5.4. Hardware Experiments

To evaluate CACTO's scalability and its potential for real-time applications we tested it on AlienGO, a quadrupedal robot featuring 12 degrees of freedom [35]. We address the problem of navigation in confined environments with even terrain and the presence of a moving obstacle. The robot must reach a moving target (see Figure 11) while avoiding collisions with both the moving obstacle (a sphere of radius 0.5 m) and the walls of the room (a rectangle with sides ranging from 2 m to 10 m). We consider a fixed trotting gait with alternating diagonal leg pairs making contact with the ground.

Since legged robots have non-differentiable dynamics [39], which are not compatible with gradient-based TO, we adopt a hierarchical approach. We combine a high-level policy relying on a simplified differentiable model, with a low-level policy using the complete robot dynamics. The high-level policy is trained with CACTO-BIC, while the low-level policy is trained with the RL algorithm CAT [40], which can handle non-differentiable dynamics. In practice, CACTO-BIC's actor is used as a standalone policy to provide reference trajectories to the low-level policy. The two policies are trained sequentially and independently. First, CACTO-BIC learns a high-level policy for generating reference trajec-

tories. After convergence, the high-level policy parameters are frozen, and the generated references are used to train the low-level CAT controller.

5.4.1. High-Level Policy

To train this policy, we employed a nonlinear version of the Linear Inverted Pendulum Model [41]. The state $x \in \mathbb{R}^{15}$ comprises the 2D offsets of the front and rear support feet relative to the associated shoulders Δp_f and Δp_r ; the Center of Mass (CoM) position $c \in \mathbb{R}^2$ and velocity $\dot{c} \in \mathbb{R}^2$ on the horizontal plane; the step index $s_{idx} \in \mathbb{N}$, which encodes both the current contact phase and time; the obstacle position $c_{obs} \in \mathbb{R}^2$; and the location of the four walls, expressed as offsets with respect to the global reference frame $\Delta_{walls} \in \mathbb{R}^4$.

$$x \triangleq (\Delta p_f, \Delta p_r, c, \dot{c}, s_{idx}, c_{obs}, \Delta_{walls}) \in \mathbb{R}^{15}. \quad (9)$$

The target position is assumed to be the origin.

Assuming a trotting gait and constant Center of Pressure (CoP) during each contact phase, the control vector $u \in \mathbb{R}^6$ includes the offsets of front and rear support feet w.r.t. the associated shoulder in the next contact phase, $\Delta p_f^+, \Delta p_r^+ \in \mathbb{R}^2$, a scalar $\alpha \in [0, 1]$ that expresses the CoP as a convex combination of the support foot positions, and the contact phase duration δt :

$$u \triangleq (\Delta p_f^+, \Delta p_r^+, \alpha, \delta t) \in \mathbb{R}^6. \quad (10)$$

The cost function penalizes the CoM-target distance and the CoM velocities, and a barrier-like cost penalizes CoM velocities beyond prescribed bounds. A smooth logarithmic reward is used to encourage reaching a narrow region around the target. Control regularization discourages deviations from nominal values: zero foot displacements, centered CoP ($\alpha = 0.5$), and nominal contact-phase duration ($\delta t = 0.375$ s). Obstacle avoidance is enforced through smooth logarithmic penalties. The algorithm took ≈ 109 s to converge. In this scenario, leveraging the GPU for solving the TO problems yields a substantial speedup of roughly $345\times$.

5.4.2. Low-Level Policy

The low-level policy is trained using a constrained RL algorithm [40], which builds upon PPO [4], employing Isaac Gym for GPU simulation. The goal of this policy is to track the references generated by the high-level policy. During training, the high-level policy is rolled out in open loop, and updated only at the beginning of each new contact phase to enhance stability and robustness [42].

The low-level policy receives the following as observations: the base state (position and orientation, linear and angular velocity, and projected gravity), the target base position and linear velocity, yaw orientation error, foot placement errors, a binary flag indicating the active diagonal contact pair, the remaining time in the current gait phase, joint positions, velocities and the previous action.

The reward function encourages accurate tracking of base position and orientation, linear and angular velocity, and the desired foot contact locations. Regularization terms are included to improve smoothness. All safety and style-related constraints are divided into soft constraints (which define a termination probability as a function of the constraint violation) and hard constraints (which cause immediate termination of the episode).

5.4.3. Aliengo Simulation and Hardware Results

The experiments are conducted in an indoor area of $4 \text{ m} \times 4 \text{ m}$ (see Figure 11). The obstacle and the target are either stationary or manually actuated, depending on the experiment.

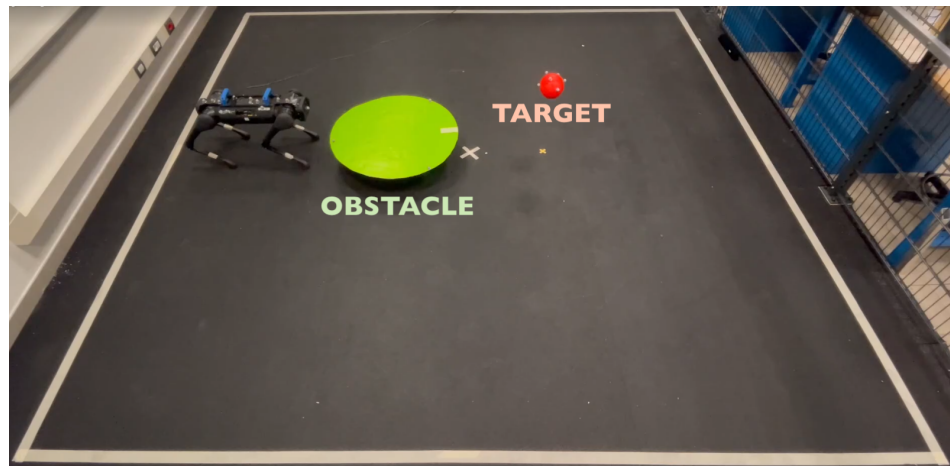


Figure 11. Experimental setup: The lines define a 4 m × 4 m operational area.

State feedback is obtained from a motion capture system, which measures position and orientation of the robot, the obstacle and the target. Velocities are estimated by finite differencing and low-pass filtering.

The results are shown in the accompanying video: https://www.youtube.com/watch?v=3_e7sVgtjAI (accessed on 20 July 2026).

6. Conclusions

This paper presented CACTO-BIC, an extension of CACTO designed to improve scalability and computational efficiency in a combined TO-RL framework by biasing the initial-state sampling using value function properties and leveraging GPU acceleration.

The experimental results demonstrate that the proposed sampling strategy effectively identifies state-space regions where actor policy improvement is more likely, leading to a 2.5–3.5× increase in sample efficiency. In addition, GPU-based computation achieves 30–250× speedup compared to CACTO, with increasing benefits as system complexity grows. When compared to PPO, CACTO-BIC achieves similar final costs requiring only 7–30% of the training time. Finally, experiments on the AlienGO quadruped robot show that CACTO-BIC scales to high-dimensional (15D) robotic systems and can be effectively used for real robot control, by means of receding-horizon planning with a reduced quadruped model.

Future works will focus on extending the proposed approach to handle constraints using augmented Lagrangian formulations [43]. In addition, we will explore the integration of sampling-based optimization techniques, such as MPPI [44], to tackle non-differentiable dynamics. Finally, applying domain randomization may improve robustness to uncertain dynamics enhancing generalization to real-world systems [45].

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/robotics15080144/s1>, Video S1: Simulation and Hardware Experiment Results.

Author Contributions: Conceptualization, A.D.P. and E.A.; methodology, A.D.P. and E.A.; software, E.A.; validation, E.A. and E.F. and P.N.C.; writing—original draft preparation, A.D.P. and E.A.; writing—review and editing, E.A., E.F. and P.N.C.; supervision, A.D.P.; funding acquisition, A.D.P.

Funding: This research was financed by the European Union—Next Generation EU, Mission 4, Component 2—CUP E53D23001130006 (STARLIT).

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The code used to generate the datasets and reproduce the experiments presented in this study is publicly available at the project repository (GitHub page of the project): https://github.com/idra-lab/cacto/tree/main_jax (accessed on 20 July 2026).

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Bellman, R. The theory of dynamic programming. *Bull. Am. Math. Soc.* **1954**, *60*, 503–515. [CrossRef]
2. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
3. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2018.
4. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
5. Carius, J.; Farshidian, F.; Hutter, M. Mpc-net: A first principles guided policy search. *IEEE Robot. Autom. Lett.* **2020**, *5*, 2897–2904. [CrossRef]
6. Ghezzi, A.; Hoffman, J.; Frey, J.; Boedecker, J.; Diehl, M. Imitation learning from nonlinear mpc via the exact q-loss and its gauss-newton approximation. In *Proceedings of the 2023 62nd IEEE Conference on Decision and Control (CDC)*; IEEE: New York, NY, USA, 2023.
7. Levine, S.; Koltun, V. Guided policy search. In *Proceedings of the International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2013.
8. Lidec, Q.L.; Jallet, W.; Laptev, I.; Schmid, C.; Carpentier, J. Enforcing the consensus between Trajectory Optimization and Policy Learning for precise robot control. *arXiv* **2022**, arXiv:2209.09006.
9. Reiter, R.; Ghezzi, A.; Baumgärtner, K.; Hoffmann, J.; McAllister, R.D.; Diehl, M. Ac4mpc: Actor-critic reinforcement learning for nonlinear model predictive control. *arXiv* **2024**, arXiv:2406.03995.
10. Ceder, K.; Zhang, Z.; Burman, A.; Kuangaliyev, I.; Mattsson, K.; Nyman, G.; Petersén, A.; Wisell, L.; Åkesson, K. Bird’s-eye-view trajectory planning of multiple robots using continuous deep reinforcement learning and model predictive control. In *Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; IEEE: New York, NY, USA, 2024.
11. Romero, A.; Song, Y.; Scaramuzza, D. Actor-critic model predictive control. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2024.
12. Zarrouki, B.; Spanakakis, M.; Betz, J. A safe reinforcement learning driven weights-varying model predictive control for autonomous vehicle motion control. In *Proceedings of the 2024 IEEE Intelligent Vehicles Symposium (IV)*; IEEE: New York, NY, USA, 2024.
13. Lowrey, K.; Rajeswaran, A.; Kakade, S.; Todorov, E.; Mordatch, I. Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv* **2018**, arXiv:1811.01848.
14. Jordana, A.; Kleff, S.; Haffemayer, A.; Ortiz-Haro, J.; Carpentier, J.; Mansard, N.; Righetti, L. Infinite-Horizon Value Function Approximation for Model Predictive Control. *IEEE Robot. Autom. Lett.* **2025**, *10*, 7563–7570. [CrossRef]
15. Grandesso, G.; Alboni, E.; Papini, G.P.R.; Wensing, P.M.; Del Prete, A. CACTO: Continuous Actor-Critic With Trajectory Optimization—Towards Global Optimality. *IEEE Robot. Autom. Lett.* **2023**, *8*, 3318–3325. [CrossRef]
16. Morgan, A.S.; Nandha, D.; Chalvatzaki, G.; D’Eramo, C.; Dollar, A.M.; Peters, J. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. In *Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2021.
17. Jenelten, F.; He, J.; Farshidian, F.; Hutter, M. Dtc: Deep tracking control. *Sci. Robot.* **2024**, *9*, eadh5401. [CrossRef] [PubMed]
18. Silver, T.; Allen, K.; Tenenbaum, J.; Kaelbling, L. Residual policy learning. *arXiv* **2018**, arXiv:1812.06298.
19. Alboni, E.; Grandesso, G.; Papini, G.P.R.; Carpentier, J.; Del Prete, A. CACTO-SL: Using Sobolev learning to improve continuous actor-critic with trajectory optimization. In *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*; PMLR: Cambridge, MA, USA, 2024.
20. Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M.J.; Katariya, Y.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; Zhang, Q. JAX: Composable Transformations of Python+NumPy Programs. 2018. Available online: <http://github.com/jax-ml/jax> (accessed on 20 July 2026).
21. Jacobson, D.H.; Mayne, D.Q. *Differential Dynamic Programming*; Modern Analytic and Computational Methods in Science and Mathematics; American Elsevier Publishing Company: New York, NY, USA, 1970.

22. Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; Munos, R. Unifying count-based exploration and intrinsic motivation. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 1479–1487.
23. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on Machine Learning*; PMLR: Cambridge, MA, USA, 2017.
24. Burda, Y.; Edwards, H.; Storkey, A.; Klimov, O. Exploration by random network distillation. *arXiv* **2018**, arXiv:1810.12894.
25. Parsa, P.; Moayedi, R.Z.; Bornosi, M.; Bejani, M.M. Where2Start: Leveraging initial States for Robust and Sample-Efficient Reinforcement Learning. *arXiv* **2023**, arXiv:2311.15089.
26. Schenke, M.; Wallscheid, O. Improved exploring starts by kernel density estimation-based state-space coverage acceleration in reinforcement learning. *arXiv* **2021**, arXiv:2105.08990.
27. Yin, D.; Thiagarajan, S.; Lazic, N.; Rajaraman, N.; Hao, B.; Szepesvari, C. Sample efficient deep reinforcement learning via local planning. *arXiv* **2023**, arXiv:2301.12579.
28. Tavakoli, A.; Levdi, V.; Islam, R.; Smith, C.M.; Kormushev, P. Exploring restart distributions. *arXiv* **2018**, arXiv:1811.11298.
29. Messikommer, N.; Song, Y.; Scaramuzza, D. Contrastive initial state buffer for reinforcement learning. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2024.
30. Sluijterman, L.; Cator, E.; Heskes, T. Optimal training of mean variance estimation neural networks. *Neurocomputing* **2024**, *597*, 127929. [\[CrossRef\]](#)
31. Andersson, J.A.E.; Gillis, J.; Horn, G.; Rawlings, J.B.; Diehl, M. CasADi—A software framework for nonlinear optimization and optimal control. *Math. Program. Comput.* **2019**, *11*, 1–36. [\[CrossRef\]](#)
32. Alentev, I.; Kozlov, L.; Nedelchev, S. JaxADi: Bridging CasADi and JAX for Efficient Numerical Computing. 2024. Available online: <https://github.com/based-robotics/jaxadi> (accessed on 20 July 2026).
33. Frostig, R.; Sindhvani, V.; Singh, S.; Tu, S. trajax: Differentiable Optimal Control on Accelerators. 2021. Available online: <http://github.com/google/trjax> (accessed on 20 July 2026).
34. Heek, J.; Levskaya, A.; Oliver, A.; Ritter, M.; Rondepierre, B.; Steiner, A.; van Zee, M. Flax: A Neural Network Library and Ecosystem for JAX, version 0.12.8. 2024. Available online: <http://github.com/google/flax> (accessed on 20 July 2026).
35. Unitree Robotics. AlienGO. Available online: <https://www.unitree.com/cn/products/aliengo> (accessed on 20 July 2026).
36. Dubins, L.E. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *Am. J. Math.* **1957**, *79*, 497–516. [\[CrossRef\]](#)
37. Zelch, C.; Peters, J.; von Stryk, O. Learning control policies from optimal trajectories. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2020; pp. 2529–2535.
38. Freeman, C.D.; Frey, E.; Raichuk, A.; Girgin, S.; Mordatch, I.; Bachem, O. Brax—A Differentiable Physics Engine for Large Scale Rigid Body Simulation. *arXiv* **2021**, arXiv:2106.13281.
39. Wensing, P.M.; Posa, M.; Hu, Y.; Escande, A.; Mansard, N.; Del Prete, A. Optimization-based control for dynamic legged robots. *IEEE Trans. Robot.* **2023**, *40*, 43–63. [\[CrossRef\]](#)
40. Chane-Sane, E.; Leziart, P.A.; Flayols, T.; Stasse, O.; Souères, P.; Mansard, N. Cat: Constraints as terminations for legged locomotion reinforcement learning. In *Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; IEEE: New York, NY, USA, 2024.
41. Kajita, S.; Kanehiro, F.; Kaneko, K.; Fujiwara, K.; Harada, K.; Yokoi, K.; Hirukawa, H. Biped walking pattern generation by using preview control of zero-moment point. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*; IEEE: New York, NY, USA, 2003; Volume 2.
42. Villa, N.A.; Wieber, P.B. Model predictive control of biped walking with bounded uncertainties. In *Proceedings of the 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*; IEEE: New York, NY, USA, 2017.
43. Li, J.; Fridovich-Keil, D.; Sojoudi, S.; Tomlin, C.J. Augmented Lagrangian Method for Instantaneously Constrained Reinforcement Learning Problems. In *Proceedings of the 2021 60th IEEE Conference on Decision and Control (CDC)*; IEEE: New York, NY, USA, 2021.
44. Williams, G.; Aldrich, A.; Theodorou, E. Model predictive path integral control using covariance variable importance sampling. *arXiv* **2015**, arXiv:1509.01149.
45. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*; IEEE: New York, NY, USA, 2018; pp. 3803–3810.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.