# CACTO-SL: Using Sobolev Learning to improve Continuous Actor-Critic with Trajectory Optimization

Elisa Alboni[1], Gianluigi Grandesso[1], Gastone P. Rosati Papini[1], Justin Carpentier[2], and Andrea Del Prete [1]

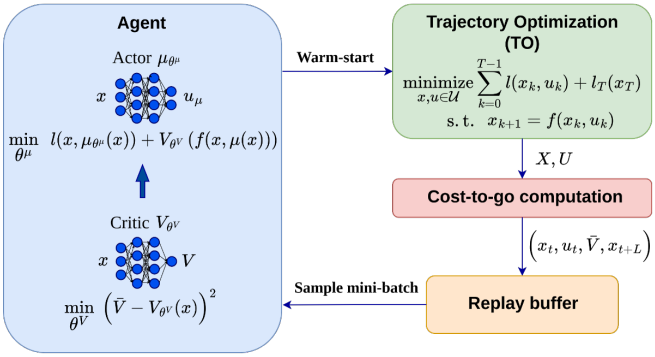
***Abstract*— Trajectory Optimization (TO) and Reinforcement Learning (RL) are two powerful and complementary tools to solve optimal control problems. On the one hand, TO can efficiently compute locally-optimal solutions, but it tends to get stuck in local minima if the problem is highly not convex. On the other hand, RL is less sensitive to non-convexity, but it requires a much higher computational effort. Recently, we have proposed CACTO (Continuous Actor-Critic with Trajectory Optimization), and algorithm that uses TO to guide the exploration of an actor-critic RL algorithm. In turns, the policy encoded by the actor is used to warm-start TO, closing the loop between TO and RL. In this work, we present an extension of CACTO exploiting the idea of Sobolev learning. To make the training of the critic network faster and more data efficient, we enrich it with the gradient of the Value function, computed via a backward pass of the Differential Dynamic Programming algorithm. Our preliminary results show that the new algorithm is more efficient than the original CACTO, both in terms of computation time and of data points, while also providing more consistent results across runs with different random seeds.**

## I. Introduction

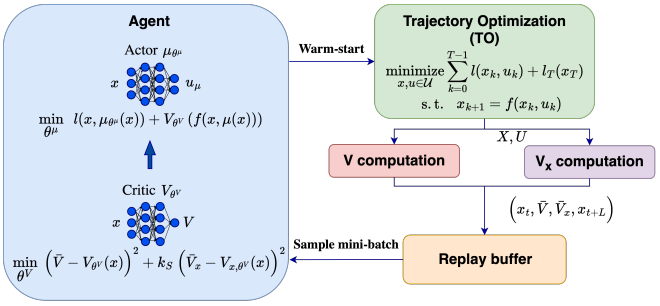
Robotic control challenges have long been addressed through Trajectory Optimization (TO). The high-level desired task is encoded in the cost function of a constrained Optimal Control Problem (OCP), which is minimised by acting on the OCP decision variables, the trajectories of state and control. When tackling complex problems, the OCP may feature a highly non-convex cost function and/or highly nonlinear dynamics. Therefore, gradient-based solvers frequently encounter local minima and are unable to find the globally optimal solution.

With the emergence of deep Reinforcement Learning (RL) and its application to the continuous domain, this machine learning tool is applied more and more widely to robot control problems showing impressive results on continuous state and control spaces [1], [2], [3], [4]. RL algorithms are less prone to converge to local minima due to their exploratory nature. Yet, there are still several challenges related to the application of RL to robot control, such as the necessity for extensive exploration.

As a potential solution to overcome the limitations of RL and TO, we have presented CACTO [5]. CACTO iteratively leverages the explorative nature of RL to initialize TO to escape local minima, while exploiting TO to guide the RL exploration. Thanks to the interplay of TO and RL,




(a) Scheme of the original CACTO formulation




(b) Scheme of CACTO with Sobolev Learning

Fig. 1: Comparison between the schematics of the original CACTO formulation and the CACTO formulation enhanced by Sobolev Learning.

CACTO's policy provides TO with initial guesses that allow it to obtain better trajectories than with other initialization techniques. While the use of TO demonstrated to efficiently accelerate the algorithm convergence, the computational burden associated with solving TO episodes has posed some limitations. In particular, as system complexity increases, this issue becomes more relevant, hindering the scalability of the algorithm. The ability of TO to produce the Value function derivatives with limited computational cost presents an opportunity to enhance the algorithm's efficiency by increasing the information extracted from each TO problem. By leveraging Sobolev Learning, the gradient information can be incorporated in the training, enhancing the performance of the algorithm.

In this work, we incorporate Sobolev Learning in CACTO to improve its efficacy and scalability. We empirically compare the new algorithm with its original version, showing a remarkable improvement in terms of both median perfor-

This work was supported by PRIN project DOCEAT (CUP n. E63C22000410001).

[1] Dept. of Industrial Engineering, University of Trento, Italy [name.surname]@unitn.it

[2] INRIA, Paris, France justin.carpentier@inria.fr

mance and its consistency across different runs.

## II. Method

In [5] we presented CACTO, an algorithm that leverages the complementarity between TO and RL for finding quasi-optimal control policies through TO-guided RL policy search. To increase its computational efficiency, we propose to use Sobolev Learning (SL) for learning the Value function. This method exploits the derivative information about the target function in the training process, including an additional term in the regression loss function that encourages the derivatives of the network to match the target function's derivatives. We developed a new formulation of CACTO, which involves the computation of the derivatives of the Value function at the end of each episode and employs them for training the critic. The schematics of the original CACTO formulation and the CACTO-SL formulation are compared in Fig. 1.

### A. Original CACTO Formulation

This section presents the former formulation of CACTO [5], an optimization algorithm designed to address discrete-time optimal control problems with a finite time horizon, characterized by the following structure:

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

where the decision variables are the state and control sequences denoted as $X = x_{0\dots T}$ and $U = u_{0\dots T-1}$, with $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$. The cost function $L(\cdot)$ is defined as the sum of the running costs $l_k(x_k,u_k)$ and the terminal cost $l_T(x_T)$. The dynamics, control limits and initial conditions are represented by (1b), (1c) and (1d).

The algorithm begins by solving $N$ TO problems from random initial states using a classic warm-starting technique, such as random values, or the initial condition for the state and zero for the control. Time is included as the last component of the state vector to address the time dependency of the Value function. Therefore, for each TO episode the initial time is randomized as the other state elements. For each problem, CACTO computes the partial $L$-step cost-to-go associated to each optimal state, where $L$ is the number of lookahead steps used for Temporal Difference learning, and stores them in a replay buffer along with the relative transition (i.e. state, control, and state after $L$ steps). Notice that by setting $L$ equal to $T$, the cost-to-go is computed with Monte-Carlo. After $N$ episodes the networks are updated. For $M$ times, a batch of transitions is sampled from the replay buffer and used to update the neural networks of the critic and the actor. In particular, the parameters of the critic are updated so that the mean squared error between the critic's output and the reference cost-to-go is minimized, while the parameters of the actor are updated according to the gradient of the Q function. Finally, a rollout of the actor's policy is used to warm-start the TO problems in the next episodes, closing the loop.

The algorithm has been proven to converge to the globally optimal solution in a discrete-space version, providing valuable insights into its theoretical principles.

### B. CACTO with Sobolev Learning

In order to exploit Sobolev Learning we need to provide the agent with the derivative of the value function with respect to the state: $V_x$. To analytically compute $V_x$, we use the backward-pass of *Differential Dynamic Programming* (DDP) [6], an optimal control method for unconstrained nonlinear problems. At the end of each episode we compute a local quadratic approximation of the action-value function $Q$, about each state-action pair of the locally-optimal TO trajectory $(\bar{x}_i, \bar{u}_i)$:

$$\begin{aligned}
Q_i(\bar{x}_i+\delta x_i, \bar{u}_i+\delta u_i) \approx\ & Q_i(\bar{x}_i,\bar{u}_i) + \begin{bmatrix} Q_{x,i}^\top & Q_{u,i}^\top \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix} + \\
& \frac{1}{2} \begin{bmatrix} \delta x_i^\top & \delta u_i^\top \end{bmatrix} \begin{bmatrix} Q_{xx,i} & Q_{xu,i} \\ Q_{ux,i} & Q_{uu,i} \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix}
\end{aligned} \tag{2}$$

where $Q_x, Q_u, Q_{xx}, Q_{xu}$, and $Q_{uu}$ are the partial first and second derivatives of $Q$ with respect to $x$ and $u$. These derivatives can be expressed as a function of the derivatives of the cost $l$, the dynamics $f$, and the Value function $V$:

$$\begin{aligned}
Q_{x,i} &\triangleq l_{x,i} + f_x^\top V_{x,i+1} \\
Q_{u,i} &\triangleq l_{u,i} + f_u^\top V_{x,i+1} \\
Q_{xx,i} &\triangleq l_{xx,i} + f_x^\top V_{xx,i+1} f_x \\
Q_{uu,i} &\triangleq l_{uu,i} + f_u^\top V_{xx,i+1} f_u \\
Q_{xu,i} &\triangleq l_{xu,i} + f_x^\top V_{xx,i+1} f_u
\end{aligned} \tag{3}$$

We can then minimize the quadratic approximated action-value function w.r.t. $\delta u_i$, obtaining the locally-optimal control $\delta u_i^* = -Q_{uu,i}^{-1}(Q_{u,i} + Q_{ux,i}\delta x_i)$. We can then substitute $\delta u_i^*$ in (2) to compute our quadratic approximation of $V_i$, which gives us:

$$\begin{aligned}
V_{x,i} &= Q_{x,i} - Q_{xu,i} Q_{uu,i}^{-1} Q_{u,i} \\
V_{xx,i} &= Q_{xx,i} - Q_{xu,i} Q_{uu,i}^{-1} Q_{ux,i}
\end{aligned} \tag{4}$$

The algorithm is initialized with the value of $V$ at time step $T$, which corresponds to the terminal cost: $V_{x,T} = l_{x,T}$, $V_{xx,T} = l_{xx,T}$. For more details on DDP, we refer the reader to [6].

The derivative of the Value function is then stored in the replay buffer along with the associated transition. Finally, during the update-phase, the critic parameters $\theta^V$ are updated to approximate both the target values and their derivatives:

$$\underset{\theta^V}{\text{minimize}} \quad \frac{1}{S}\sum_{i=1}^{S} \left(\bar{V}_i - V(x_i|\theta^V)\right)^2 + k_S \left(\bar{V}_{x,i} - V_x(x_i|\theta^V)\right)^2 \tag{5a}$$

The relative importance is assigned to the two loss component by mean of the coefficient $k_S$.

Since the state in CACTO is augmented with the time variable, we could also benefit from the derivative of $V$

w.r.t. $t$. However, since time in DDP is discrete, we cannot compute it.

It is important to remark that our computation of $V_x$ corresponds to a Monte-Carlo approach rather than a TD approach. Potentially, a customized backward-pass could be designed to compute $V_x$ with a TD approach, but we leave this extension to future work.

### C. Implementation Details

This section discusses some implementation details, with a focus on the differences w.r.t. the original CACTO.

In CACTO-SL, for the critic network we have switched from ReLU to ELU activation functions because they are smooth and therefore help to match the Value's gradient.

Sobolev learning leads to a more accurate approximation of the Value's gradient, used in the actor's update. Therefore, we could increase the *learning rate* for the actor's training.

While CACTO relied on *Pyomo* [7] for implementing the TO problems, CACTO-SL switched to *CasADi* [8], an open-source Automatic Differentiation framework for numeric optimization. By leveraging its symbolic framework and automatic differentiation capabilities, CasADi enables efficient computation of the cost function's derivatives, required for implementing Sobolev Learning. Moreover, *CasADi* is compatible with Pinocchio [9], a versatile rigid body dynamics library, which freed us from the burden to hand-code the system dynamics. Each TO problem is transcribed using collocation and then solved with the nonlinear optimization solver *IpOpt* [10]. Finally, to speed up the code, we parallelized the generation of the warm-start trajectories, the TO problems, and the simulations.

## III. Results

We tested the proposed algorithm CACTO-SL, with the goal to understand whether it performs better than the original CACTO. We analyse one of the scenarios presented in [5]. The task consists in reaching in the shortest time a desired position with the robot's end-effector, while avoiding three ellipse-shaped obstacles, and minimizing the control effort. The task is described by the following running cost:

$$l(x,u) = l_1(x) + l_2(x) + l_3(x) + l_4(u), \tag{6}$$

$$l_1(x) = w_d ||p_{ee} - p_g||^2 \tag{7}$$

$$l_2(x) = -\frac{w_p}{\alpha_1} \ln(e^{-\alpha_1\left(\sqrt{(x_{ee}-x_g)^2+c_2}+\sqrt{(y_{ee}-y_g)^2+c_3}+c_4\right)} + 1) \tag{8}$$

$$l_3(x) = \frac{w_{ob}}{\alpha_2} \sum_{i=1}^{3} \ln(e^{-\alpha_2\left(\frac{(x_{ee}-x_{ob,i})^2}{(a_i/2)^2}+\frac{(y_{ee}-y_{ob,i})^2}{(b_i/2)^2}-1\right)} + 1) \tag{9}$$

$$l_4(u) = w_u ||u_k||_2^2 \tag{10}$$

where $l_1$ penalizes the distance between $p_{ee} = (x_{ee}, y_{ee})$ (the x-y coordinates of the end-effector) and $p_g$ (the goal position to be reached); $l_2$ encodes a cost valley in the neighborhood of the goal; $l_3$ penalizes collision with the three obstacles centered in $p_{ob,i} = (x_{ob,i}, y_{ob,i})$ with axes $a_i$, and $b_i$; $l_4$ penalizes control efforts. The $w$'s are user-defined weights; $c_2$, $c_3$, $c_4$, $\alpha_1$, and $\alpha_2$ are the parameters of the softmax functions. Fig. 2 depicts the cost function, neglecting the control-effort term. The terminal cost is equal to the running cost, except for $l_4$.

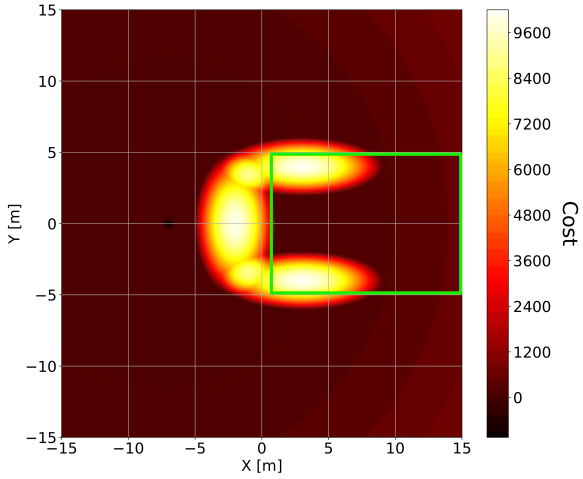



Fig. 2: Cost function without the control effort term (10), considering a target at $[-7,0]$ with weights $w_d = 100$, $w_p = 5 \cdot 10^5$ and $w_{ob} = 1 \cdot 10^6$. The green rectangle delimits the *Hard Region*.

The task is designed to ensure the presence of many local minima. In particular, if the system starts from the *hard region*, highlighted by a green rectangle in Fig. 2, it is hard for the solver to find a globally-optimal solution.

The comparison between CACTO-SL and CACTO is conducted with a 2D point mass considering a double integrator dynamics. The analysed XY space is divided in a grid with mesh size equal to 1 m. The points belonging to the obstacles are considered unfeasible initial state and so neglected. We compare the average cost of the trajectories starting from feasible points, obtained by initializing the TO solver with a rollout of CACTO-SL's policy and CACTO's policy. Each test was repeated 10 times with different random seeds, and so with a different initialization of the neural networks. The same seeds are used for the two sets of tests to minimize the effect of the network initialization. For the training of CACTO's neural networks, we employed the same iperparameters used in [5] while for CACTO-SL we adopted a new set of iperparameters based on the considerations presented in II-C. In both cases, the training is stopped when $5 \times 10^4$ updates are performed.

The state vector is $[x, y, v_x, v_y, t] \in \mathbb{R}^5$ and the control vector is $[a_x, a_y] \in \mathbb{R}^2$, both bounded in [-2, 2] m/s$^2$. The maximum episode length is 10 s, while the target point is $p_g = (-7, 0)$. We used 50-step TD learning, and we set $k_S = 1$ in the critic's loss function.

The analysis focuses on the *hard region* ($x \in [0, 15]$ m and $y \in [-5, 5]$ m). For an easier interpretation of the results, the initial velocity of the system is set to 0. The tests are performed on a Workstation CELSIUS M770 equipped with

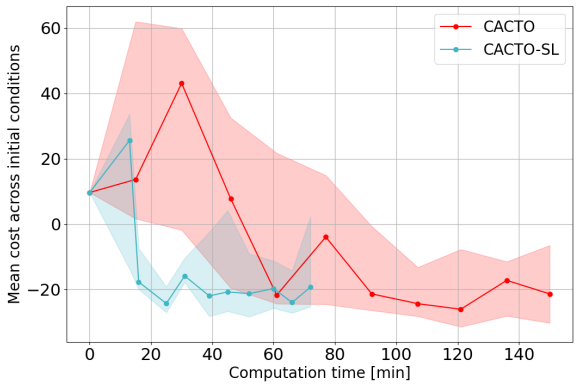



Fig. 3: Median (across runs) of the mean cost (across initial conditions) starting from the *hard* region with zero initial velocity.

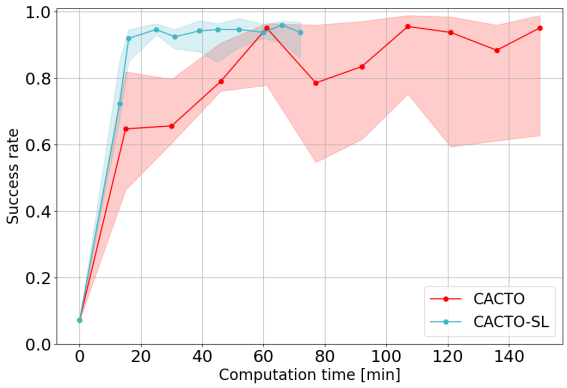



Fig. 4: Median success rate (across runs) starting from the *hard* region with zero initial velocity.

an INTEL XEON W2145 / Octa Core processor. Fig. 3 shows the median (across different runs of each algorithm) of the average cost (across different initial states), as a function of the computation time. Fig. 4 reports instead the success rate, defined as the percentage of times that the system is able to reach the target without colliding with an obstacle. In both figures the shaded areas represent the first and third quartiles, while the data reported at time 0 corresponds to using TO with the initial conditions as initial guess.

At convergence, both CACTO and CACTO-SL were able to achieve good performance in terms of average cost and success rate. However, for the same computation time, the median performance of CACTO-SL was consistently better than the median performance of CACTO. Also in terms of consistency across different runs, measured by the first and third quartiles, CACTO-SL outperformed CACTO, demonstrating to be less sensitive to the initialization of the neural networks.

Fig. 5 shows some trajectories obtained by warm-starting TO with CACTO-SL's rollouts, after $10^4$ updates (corresponding to $\approx$16 minutes of computation). All the reported

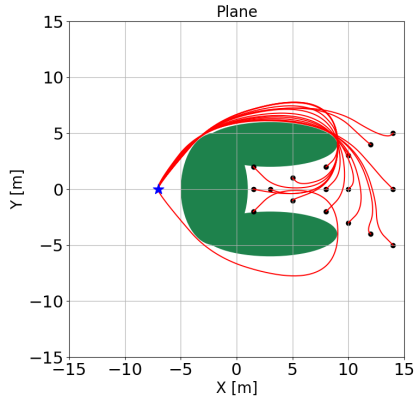



Fig. 5: Optimal trajectories obtained warm-starting TO with CACTO-SL's rollouts after $10^4$ updates.

trajectories reach the target without colliding with the obstacles after few updates.

It should be mentioned that the computation time for each iteration is slightly increased in CACTO-SL due to the computation of the network's derivative and the DDP backward pass. Overall, the computation time increase is 14%, i.e. it takes 72 min to perform $5 \cdot 10^4$ updates, instead of 63 minutes. However, this extra computations seem to be largely compensated for by the improved performance and stability of the algorithm as it more efficiently exploits each TO trajectory, and so it requires less TO episodes to converge.

## IV. Conclusions

We have presented an extension of the CACTO algorithm that exploits the gradient of the Value function in the training of the critic network. Our preliminary results show that this helps the algorithm to converge faster to high-quality solutions, while producing more consistent results across runs using different random seeds.

In the future we plan to benchmark CACTO-SL on a large number of problems to better evaluate its performance. Moreover, we plan to explore different strategies to bias the sampling of the initial states at the beginning of each episode, so as to maximize the amount of information collected from them.

## References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[2] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[3] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[5] G. Grandesso, E. Alboni, G. P. R. Papini, P. M. Wensing, and A. Del Prete, "Cacto: Continuous actor-critic with trajectory optimization—towards global optimality," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3318–3325, 2023.

[6] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2012, pp. 4906–4913. [Online]. Available: https://dada.cs.washington.edu/homes/todorov/papers/MPCGetUp.pdf

[7] B. Nicholson, J. D. Siirola, J. P. Watson, V. M. Zavala, and L. T. Biegler, "Pyomo.Dae: a Modeling and Automatic Discretization Framework for Optimization With Differential and Algebraic Equations," *Mathematical Programming Computation*, vol. 10, no. 2, pp. 187–223, 2018. [Online]. Available: https://doi.org/10.1007/s12532-017-0127-0

[8] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[9] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[10] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 2006.