Value learning from trajectory optimization and Sobolev descent: A step toward reinforcement learning with superlinear convergence properties

Amit Parag, Sébastien Kleff, Léo Saci, Nicolas Mansard, Olivier Stasse

Value learning from trajectory optimization and Sobolev descent: A step toward reinforcement learning with superlinear convergence properties

Amit Parag[1,2,*], Sébastien Kleff[2,3], Léo Saci[2], Nicolas Mansard[1,2] and Olivier Stasse[1,2]

***Abstract*— The recent successes in deep reinforcement learning largely rely on the capabilities of generating masses of data, which in turn implies the use of a simulator. In particular, current progress in multi body dynamic simulators are underpinning the implementation of reinforcement learning for end-to-end control of robotic systems. Yet simulators are mostly considered as black boxes while we have the knowledge to make them produce a richer information. In this paper, we are proposing to use the derivatives of the simulator to help with the convergence of the learning. For that, we combine model-based trajectory optimization to produce informative trials using 1st- and 2nd-order simulation derivatives. These locally-optimal runs give fair estimates of the value function and its derivatives, that we use to accelerate the convergence of the critics using Sobolev learning. We empirically demonstrate that the algorithm leads to a faster and more accurate estimation of the value function. The resulting value estimate is used in model-predictive controller as a proxy for shortening the preview horizon. We believe that it is also a first step toward superlinear reinforcement learning algorithm using simulation derivatives, that we need for end-to-end legged locomotion.**

I. Introduction

We consider the problem of computing the optimal control policy in the continuous domain, with application to the end-to-end control of polyarticulated robots, like torque-controlled manipulator arms or legged robots.

Since the advent of deep reinforcement learning [1] and its generalization to continuous domain [2], [3], robot control has been recognized as one of the major challenges of the domain [4]. On the one hand, research has targeted advanced robot scenarios and sensor outputs [5], [6], often relying on an underpinning low-level controller (impedance or Cartesian controller). On the other hand, efforts have been made to target more complex robots such as quadruped [7] or biped [8], by focusing on a more accurate modeling of dynamics [9] and a more advanced - stronger, robust, faster - simulation of polyarticulated behavior [10], [11]. In parallel, RL solvers have made great progress and are now more stable [3] and faster to converge [12].

Yet solving end-to-end polyarticulated problem remains a difficult challenge for generic RL solvers. Several appraoches have been proposed to make their work easier. Acknowledging for the improved capabilities of motion planner to discover complex movement, guided policy search has been used to bootstrap the reinforcement loop with locally optimal trajectories obtained from a trajectory optimizer [13]. The trajectories can later be refined using the trained policy to produce better demonstrations by warm-starting the trajectory optimizer [14], or the policy and the demonstrations be simultaneously optimized [15], [16]. These works show the correlation between policy learning and trajectory optimization, as optimal trajectories and optimal policies are two ways of considering the solution of the optimal control (or Markov decision) problem.

Reciprocally, several works have tried to take advantage of this duality to learn a quantity that would help a trajectory optimizer. In [17], a trajectory optimizer is helped with a dedicated dynamic model learned on a specific task. In [18], a dedicated cost model is learned to cope with the prohibitive complexity of optimizing long-horizon trajectories. Shortening the trajectory horizon typically leads to observing the importance of obtaining a good approximation of the value function, both in trajectory optimization and policy learning. It was originally proposed to learn the value as a proxy for the cost corresponding to the end of the horizon [19]. While a fair approximation of the value function can be obtained offline by sampling long locally optimal trajectories, the computation cost might become prohibitive when the state space is large or requires clever sampling solutions [20].

Producing new optimal trajectories while exploiting a previously optimized value approximation in a reinforcement loop can be considered as an actor-critic formulation, where the actor is optimally acting over an horizon: it can then be shown that the prediction reduces the noise in learning the value and speeds up the training [21]. As shown in [22], deep policy gradients methods often fail to adhere to the key primitives of reinforcement learning: the role of the value function estimated by the critic is only marginal since the learned value function does not accurately model the underlying value function.

In this paper, we focus on learning the value function by combining a trajectory optimizer and a supervised learning phase using value approximation and its derivatives. Our objective is to learn the value with a high accuracy, while avoiding the need to extensively sample the state space. A trajectory optimizer is first used to sample optimal trajectories over a short horizon to get a lower bound on the value function. As we are using differential dynamic programming [23]–[25], a particular instance of (direct shooting) trajectory optimizer, we also get, for free, samples of the derivative of the value (see Sec. II). We then train a neural network to approximate the value and its derivative using a Sobolev loss (see Sec. III). This particular loss accelerates

[1]Artificial and Natural Intelligence Toulouse Institute, France
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[3]New York University, USA
*corresponding author: aparag@laas.fr

This work has been supported by the the European project MEMMO (GA-780684) and ANITI (ANR-19-P3IA-0004).

the convergence to a more accurate approximation, while enabling better generalization despite sparse sampling. As a side effect, it also provides an accurate way of evaluating the approximate gradient of the value. As our trajectory optimizer also demands the Hessian of the value, we propose a particular network architecture, that we call Residual network, which provides a fair approximation of the second derivatives. We evaluate of our approach on three classical control problems : inverted pendulum, cartpole and unicycle, that are exhaustively investigated, and on a more demanding demonstrator featuring a 7-dof manipulator arm.

II. Recursive optimality

A. Problem formulation and notations

While optimal control problem are best written with continuous time and integrals, we directly consider here its transcription into a time-discrete system with autonomous dynamics:

$$x_+ = f(x, u, \Omega) \tag{1}$$

where the next state, x_+, and the current state, x, are living in an n-dimensional vector space $x \in \mathcal{X}$, (possibly representing an element of a Lie group), and controls $u \in \mathcal{U}$, with Ω a parametrization of the environment where the system evolves and f, the time independent dynamic function that governs the state transition from the current state and control. The goal in optimal control is to find pairs of trajectories $X : t \to x(t) \in \mathcal{X}$ and $U : t \to u(t) \in \mathcal{U}$ that minimizes some cost functional $L(X, U)$. For a discrete system, the cost functional can be written as :

$$L(X, U) = \sum_{k=0}^{+\infty} l(x_k, u_k) \tag{2}$$

where $l(x, u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost. Therefore given an initial state x_0 and a time horizon T, aim of optimal control is to find the pair of optimal state and control trajectories X^*, U^* that minimizes the cost functional.

We denote by $V : x \to V(x) \in \mathbb{R}$ the value function, i.e. the optimal value of $L(X, U)$ when the trajectory X starts from a given state x, and by $\pi : x \to \pi(x) \in \mathcal{U}$ the optimal policy.

B. Truncating the horizon

While we typically want to control the system during an indefinite (infinite) horizon, finite computational resources implies to shorten the horizon to a finite length T. The infinite sum (2) is then truncated to:

$$L(X, U) = \sum_{k=0}^{T-1} l(x_k, u_k) + l^f(x_T) \tag{3}$$

We define V_k as the cost-to-go, over the horizon $T - k$, from a starting state x as :

$$V_{T-k}(x) = \min_U \sum_{j=k}^{T-1} l(x_j, u_j) + l^f(x_T) \tag{4}$$

Recursive optimality can be obtained if the terminal cost is chosen to be the value function. In that case, the problem writes:

$$V_{T-k}(x) = \min_u l(x, u) + V_{T-k-1}(f(x, u)) \tag{5}$$

The problem then becomes the infinite horizon problem but can be solved with finite resources, given that $V(x)$ can be properly evaluated. In this particular case, the cost-to-go is independent of the timestep and is equal to the value function:

$$V_T(x) = V(x), \quad \forall T > 0$$

C. Differential Dynamic Programming

Differential dynamic programming (DDP) exploits the recursive nature of Bellman Optimality Principle by rewriting the optimal value function, Eq 3 with the terminal condition $V_T(x) = l^f(x)$. It proceeds by iteratively solving the optimal control problem described in Sec II-A to find the local solution by performing a backward pass to approximate a quadratic fit of the value function around the current candidate trajectory and a forward pass to compute a new nominal trajectory based on the value function computed in backward pass.

A consequence of the backward pass is that it reconstructs a quadratic model of the value function along the horizon. The solver then returns an evaluation of the cost-to-go and its first and second order derivatives over the preview horizon.

DDP is a second order algorithm with superlinear convergence rate. It requires at each iteration an evaluation of the dynamic and cost function and their derivatives. In particular, we need to provide the first and second order derivatives of the terminal cost $l^f(x_T)$.

D. Value at the horizon start and end

In the following, we will use an approximation of the value function as a proxy to represent the truncated horizon end. This implies that we should be able to evaluate an approximation of the value function and its first and second order derivatives at the end of the horizon. The solver will then return a refined approximation of the value function at the beginning of the horizon, and its derivatives. We will now use these two properties to build our algorithm.

III. Differential value programming

A. Algorithm principles

1) Batch of trajectories and cost-to-go learning: We propose an algorithm named Differential Value Programming (DVP) that iteratively builds a better neural approximation of the value function, V_θ^n, where θ are the parameters of the neural network at n^{th} iteration.

The first iteration of DVP simply generates a batch of optimal trajectories of horizon length T without any terminal cost model, i.e. approximating the value function at the end of the horizon by 0. We then learn the value function by supervised learning, as explained with more details below. The result of this first iteration is a neural network approximating the cost-to-go for an horizon of T, denoted by V_θ^0

2) Iterative value learning: DVP then proceeds by iteratively building upon its estimates of value functions. In the subsequent iterations, we replace the terminal cost functional with the approximated value predicted by the neural network. So, after the initial iteration is complete, we replace (3) with

$$V_n(x) = \min_u \sum_{k=0}^{T-1} l(x_k, u_k) + V_\theta^{n-1}(x_T) \tag{6}$$

where n is the n^{th} DVP iteration and V_θ^{n-1} is the deep neural network parameterized by θ representing the value approximated in the previous iteration. Should each iteration result in a perfect training, iteration n would lead to the approximation of the cost-to-go for an horizon of $(n+1).T$ which would tend toward the global V as n increases.

B. Sobolev Regression

Classical regression invariably involves optimizing the parameters of a deep neural network such that the error between the learned function and the *ground truth* function decreases. The premise of Sobolev learning, [26], is to use the derivatives of the function to be approximated, such that the derivatives of the network match the derivatives of the learned function.

The loss function for our regression is then composed of two terms:

$$loss_f = \sum_{s=1}^{S} \lambda \left(m\left(x_s \mid \theta\right), f\left(x_s\right)\right) \tag{7}$$

$$loss_d = \sum_{s=1}^{S} \sum_{d=1}^{D} \lambda_d \left(D_{\mathrm{x}}^d m\left(x_s \mid \theta\right), D_{\mathrm{x}}^d f\left(x_s\right)\right) \tag{8}$$

where $loss_f$ penalizes the differences between the model and the dataset, of S samples, with a norm λ (in our case the L_2 norm), $loss_d$ penalizes the difference between the derivatives of the model and the derivatives of the observations, and D_x^d are the higher order derivatives evaluated at x (we stay at order $d = 1$ in our implementation) for a norm λ_d (also L_2 in our implementation).

Sobolev learning has been shown to lead to better generalization in robotics and reinforcement learning [27], albeit at a higher computation cost [28], as it constrains training by forcing neural networks to fit a target slope. However, encoding the target derivative information in neural networks has been shown, empirically, to increase robustness against noise as proven in [29] and mitigate the problem of increased computation cost by being more data efficient [30].

C. Evaluating second-order derivatives

As mentioned above, Sobolev learning allows us to obtain a more accurate convergence despite a sparser dataset, and also regularizes the derivatives of the model. This second point is important as these derivatives are needed by the DDP, as explained in Sec. II. Yet DDP also requires the Hessian of the value function. While Sobolev would theoretically benefit from second-order information, which is also available in practice from the DDP, the evaluation of the second-order loss (8) and the backpropagation of its gradient are not reasonable to evaluate.

We rather propose to set up a particular network architecture, called Residual network architecture, that we demonstrate to properly regularize the training of the Hessian. Our network model reads:

$$m(x|\theta) = r(x|\theta)^2 \tag{9}$$

where $r(x|\theta)$ is a vector-tailed network, called the residual network, whose output is squared and summed to produce the final value. The gradient of this model is then simply:

$$D_x^1 m(x|\theta) = 2 D_x^1 r(x|\theta)^T r(x|\theta)$$

where $D_x^1 m$ and $D_x^1 r$ are the gradient of m and Jacobian of r respectively. The Hessian of m is approximated as:

$$D_x^2 m(x|\theta) \approx 2 D_x^1 r(x|\theta)^T D_x^1 r(x|\theta)$$

This approximation is known as the Gauss approximation, [31], and leads to the famous superlinear algorithm Gauss-Newton (which is typically implemented in most of DDP frameworks). We can interpret this network as computing the value as the square of a latent vector r, and approximating the second order derivatives of the value as the derivatives of the network tail.

We will show empirically that this particular model, combined with first-order Sobolev learning, leads to an accurate, robust and efficient representation of the value function, which perfectly suits to the requirements of the DDP algorithm.

IV. Empirical Evaluation

In this section, we report an empirical analysis of the performance of DVP on simple toy problems.

A. Experimental setup

We propose to benchmark our approach on 3 systems by using as a baseline (ground truth) the arbitrarily good approximations of the value function that can be obtained by sampling large datasets of long optimal trajectories in various configurations. We used the following problems:

a) Unicycle: The unicycle [32] features a kinematic model of evolving on the $2D$ horizontal plane either driving forward or turning on the spot. The 3 dimensional configuration vector $q = (x, y, \theta)^\mathsf{T}$ and the control $u = (v, \omega)$ includes the unconstrained longitudinal and angular velocities.

The task is to reach the goal position $q = (0, 0, 0)^\mathsf{T}$ while minimizing the residual sum of errors:

$$L = ||w_1 * q||^2 + ||w_2 * u||^2 \tag{10}$$

b) Cartpole: A cartpole[1] is a dynamical system where an underactuated pole is attached on top of a 1D actuated cart. The task is to balance the pole around its unstable unstable equilibrium (upper position) by controlling the

[1]We use the Open Ai gym, [33], implementation of the dynamical model.

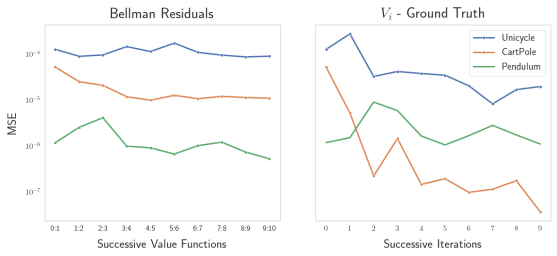


Fig. 1: Convergence of DVP using relative criteria i.e Bellman residuals (left) and absolute criteria i.e difference between value function at each iteration and ground truth (right)

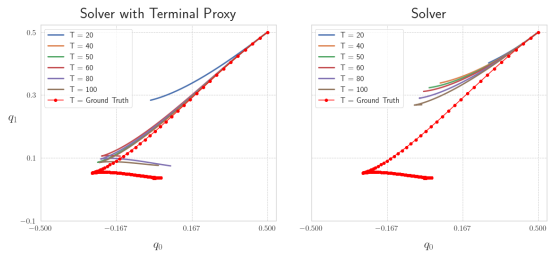


Fig. 2: Importance of horizon Length. (left) State trajectories computed by the solver with the neural value proxy V_θ^N, for T = 20 to 100. The goal position for the unicycle, with the configuration vector $q = [q_0 = x, q_1 = y, q_2 = \theta]$, is the origin, $[0, 0, 0]$, in cartesian space. The trajectory in red is the reference ground truth state trajectory computed by the solver at horizon 1000. (right) State trajectories computed by the solver without the terminal value proxy, for different horizons

horizontal forces acting on the cart [34]. The cost functional to be minimized is:

$$L = w_1||x||^2 + w_2||u||^2 \tag{11}$$

where $x = (q, v)$ is the state and the control u is force exerted on the cart.

c) Simple Pendulum: The inverted pendulum[1] swing-up problem consists in bringing the pendulum from a random position to its upper equilibrium and maintaining it upright. The cost functional we use is identical to the cartpole cost functional, with u representing the torque applied about the rotation axis of the pendulum.

In our experiments, we empirically establish baselines of DVP. Concretely, we aim to quantify the convergence properties of DVP and the influence of Sobolev training. In order to characterize the convergence properties of our algorithm, we use a validation dataset as a substitute for the value function of the infinite horizon problem. This can be done, albeit at a high computational cost, by solving the optimal control problem over extremely long horizons sufficient for reaching steady state from any starting point.

B. Convergence of DVP

1) Overall convergence: The difference between the value function at each iteration and *ground truth* with respect to DVP iterations is depicted in Fig. 1. For the pendulum case, just 1 iteration is sufficient for DVP to achieve convergence. For cartpole, DVP takes a few more iterations to converge to a good enough approximation of the global value function. For systems with regions of local minima like unicycle, achieving convergence requires relatively more iterations.

We used the idea of *Bellman residual*, [35], to establish the convergence criteria of DVP. It is easy to see that as the value function estimates come closer to the optimal value function, the difference in successive estimates decreases. Residuals between two successive value functions can be a good indication of prediction. Upon or near convergence, the higher iterations of DVP should not show much difference between their behaviors.

2) Influence of the horizon length: We discuss two results regarding the impact of the initial horizon length T. Short horizons lead to important differences between the cost-to-go and the value, hence to a poor approximation of the value in early iterations of DVP. This clearly appears in Fig. 2,

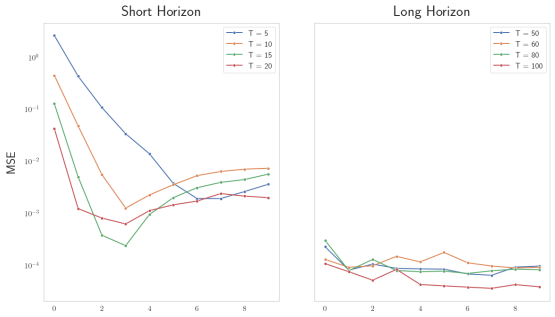


Fig. 3: Evolution of the difference between ground truth value function and different iterations, for short horizons (left) and long horizons (right), shown here for unicycle.

where the truncation to small horizon leads to trajectories far from the optimum. Once the value is properly estimated, the bundle of trajectories converges closer to the optimum (on the unicycle, the convergence is not perfect despite an accurate convergence to the value, due to nonholonomy).

Consequently, DVP converges faster when T increases. For the considered system, the typical duration of an episode (until system steady state) is 150, and DVP shows proper convergences for $T \geq 40$, see Fig. 3. For smaller T, the convergence is slower or even fails to reach a global optimum. This is not surprising and can be mitigated by increasing the dataset size.

3) Robust convergence: We empirically establish the stability and robustness of our algorithm by forcing DVP to learn a dataset with artificially biased terminal cost $l^f x_T$ in the first iteration. We find that DVP requires only a few iterations to learn the ground truth. Fig. 4 shows the convergence of DVP under various levels of initialization noise.

C. Influence of Sobolev learning

1) Importance of the Sobolev loss: Our experiments with Sobolev learning corroborate the generalization capabilities and confirm that Sobolev regression requires fewer training epochs than classical regression. Sobolev training requires only 50 samples to achieve a 10^{-3} accuracy on a validation

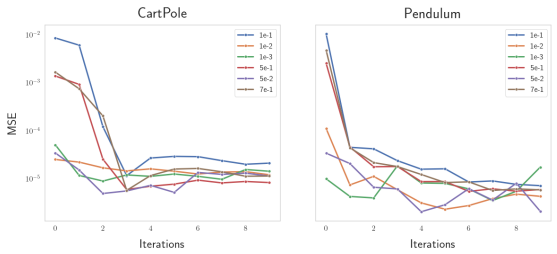


Fig. 4: Evolution of the algorithm under incorrect initialization. The y-axis measures the mean squared difference between each iteration and ground truth value function.

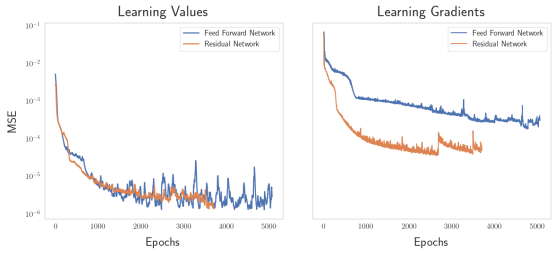


Fig. 5: Sobolev training Loss comparison for Residual Network and Feed Forward Network. The training loss in predicting value is shown in left. The training loss in gradients for feedforward and residual network is shown on the right.

dataset.

2) Residual network: For our experiments, we use a 3 layered Residual network with hyperbolic tangent as an activation function and 64 units in each hidden layers. The final residual layer contains 3 units. Empirically, we find that the advantage of modeling the value function as a squared residual leads to faster convergence during Sobolev training. The gradients of residual network are also more accurate than those of feed forward network as shown on Fig. 6.

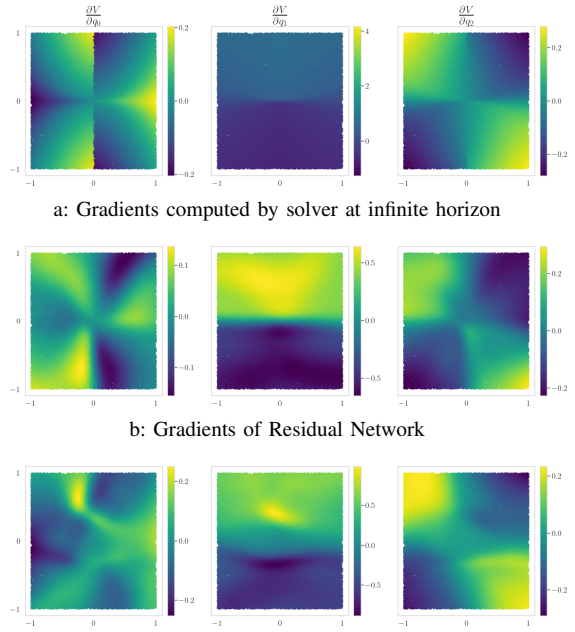
D. Comparison with PPO

We compare the accuracy and rate of convergence of DVP against a classical actor-critic RL algorithm off the shelf, which approximates the value as a side quantity when computing the policy (critic network). To cope with the requirement of the RL solver, we have consider for both DVP and PPO a discounted version of the optimal control problem considered until now.

Fig. 7 shows the qualitative comparison of the value functions predicted for pendulum, cartpole and unicycle by DDP, DVP and PPO. PPO properly captures the overall shape and the spread of the topology, but overestimates it. This is to be expected, since policy gradient methods often fail to guarantee anything about value function as empirically established in [22].

Fig. 8 shows the state trajectories computed by DVP and PPO. We observe that the trajectories computed by DVP are smoother than the trajectories output by PPO.

From our experience, PPO was also more sensitive to small changes, either to the environment parameters (e.g discount factor) and algorithm hyperparameters which limited the experiments we have been able to carry out.



a: Gradients computed by solver at infinite horizon

b: Gradients of Residual Network

c: Gradients of Feed forward Network

Fig. 6: Evaluation of the gradient approximation for the unicycle. (top) true gradients of the underlying function computed by the solver. (mid) gradients of Residual network. (bottom) gradients when using a plain feedforward network (no Gauss approximation). The two neural networks were trained for 50000 epochs on 100 samples generated by solver at horizon = 50

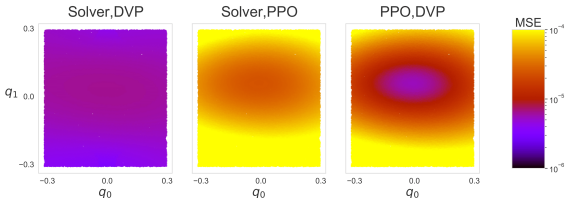
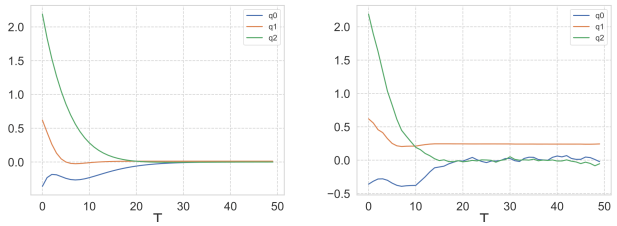


Fig. 7: Comparison between the values estimated by DVP, PPO and the ground truth for the unicycle. Differences between (left) ground truth and DVP, (mid) ground truth and PPO and (right) DVP and PPO. PPO produces a value approximation which is further to the ground truth than DVP.



a: DVP b: PPO

Fig. 8: The state trajectories predicted by DVP and PPO for unicycle. The x-axis denotes the time horizon in knots, while the y-axis shows the evolution of the configuration vector, q, of unicycle. The goal is to reach the origin $[0, 0, 0]$

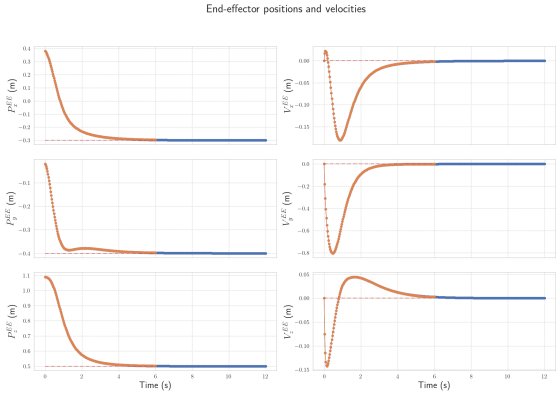


Fig. 9: End Effector position, pEE, and velocity vEE in cartesian coordinates for DVP (orange) and ground truth (blue)

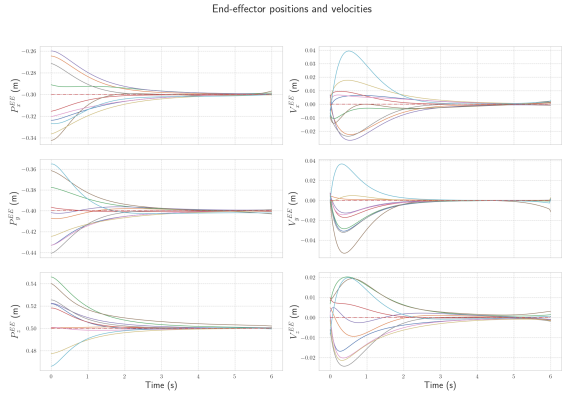


Fig. 10: EE trajectories for multiple starting configurations computed by solver with terminal neural cost functional.

V. Application to a Robot Manipulator

In this section, we show the scaling of our algorithm. We consider a 7-dof manipulator, controlled in torque, where the robot state $x = (q, \dot{q})$ concatenates at the joint configuration and velocity and $u = Z_q$ are the joint torques. The dynamics are computed using Pinocchio, [36], while policy trials are validated with Bullet [37].

We formulate the optimal control problem as a static End Effector (EE) pose reaching ocp task. We use a quadratic cost on the EE translation and state limits. Additionally, we regularize the state and torque controls.

The primary feature of DVP is that as iterations increase, the approximated value function asymtotes to global time independent value function. So, when used as a proxy for terminal cost functional, DVP tends to drive the locally optimal solver toward globally optimal solution. This immediately constrains the corresponding trajectories to satisfy the Hamilton-Jacobi-Bellman criteria of optimal substructures: sub-solutions of an indefinite horizon optimal control problem should also be optimal solutions to the corresponding definite horizon subproblems. We can see this quite easily in Fig 9 for the EE trajectory computed by the solver with approximated value function as terminal proxy computed at horizon 200 (in orange) and the ground truth EE trajectory computed at horizon 1000 (shown in blue). The EE trajectories for truncated horizon problem are co-incident with the infinite horizon trajectory.

The trajectories computed with DVP and solver, for truncated horizon, also maintain the recursive optimality and stability when used online in simulation. The DVP terminal cost can also serve as a highly stable anchor that allows for quick re-planning online under external disturbances. Fig 11 shows the evolution of mean squared errors between EE trajectories computed by solver-DVP and solver at infinite horizon, when external perturbations are injected in the system.

Finally, in Fig 10 we show the generalizability of our algorithm to compute optimal trajectories for different starting configurations. We approximate the terminal value function with the 10^{th} iteration of DVP.

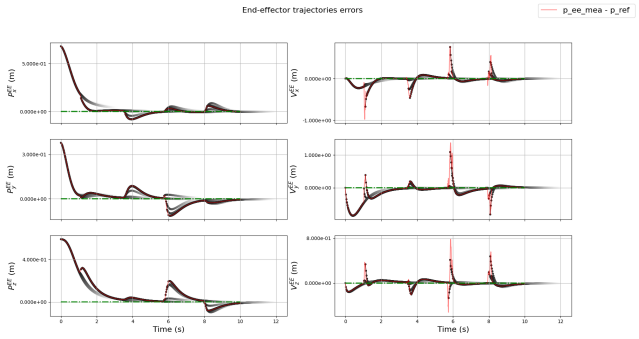


Fig. 11: Evolution of mean squared errors when disturbances are externally injected in Bullet.

A more comprehensive rendering of PyBullet simulation can be found at `https://gepettoweb.laas.fr/articles/amit_icra_22.html`

VI. Conclusion

In this paper, we have proposed an algorithm to accurately learn the value function of an optimal control problem. Our contribution relies on a trajectory optimizer which produces good estimates of the cost-to-go over a finite preview horizon. We then leverage on supervised learning using (i) a Sobolev loss and (ii) a particular network architecture that we named Residual network, to learn an accurate approximation of the value function and its derivatives. We can then use our value model to evaluate an approximation of the value at the end of the horizon, and its derivatives. By alternating between production of new optimal trajectories and refinement of the value and its derivatives, we set up a reinforcement loop which leads to a quick convergence to an accurate value approximation. We have proposed a complete evaluation of our method, on three typical classic control and a more demanding system. We see our algorithm as a first step toward building a complete reinforcement learning algorithm for the continuous domain, able to fully exploits the derivatives of the simulator to reach superlinear convergence rate.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[2] 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," in *International Conference on Learning Representations*. ICLR, 2015.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*. ICML, 2015.

[4] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning*. CoRL, 2020.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, 2016.

[6] R. Strudel, A. Pashevich, I. Kalevatykh, I. Laptev, J. Sivic, and C. Schmid, "Learning to combine primitive skills: A step towards versatile robotic manipulation," in *IEEE International Conference on Robotics and Automation*. ICRA, 2020.

[7] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," in *Robotics: Science and Systems XV*. RSS, 2019.

[8] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning*. CoRL, 2020.

[9] M. Hutter, C. Gehring, A. Lauber, F. Gunther, C. D. Bellicoso, V. Tsounis, P. Fankhauser, R. Diethelm, S. Bachmann, M. Blösch *et al.*, "Anymal-toward legged robots for harsh environments," *Advanced Robotics*, vol. 31, no. 17, 2017.

[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE International Conference on Intelligent Robots and Systems*. IROS, 2012.

[11] J. Hwangbo, J. Lee, and M. Hutter, in *Per-Contact Iteration Method for Solving Contact Dynamics*. RAL, 2018.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*. ICML, 2016.

[13] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*. ICML, 2013.

[14] N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, and O. Stasse, "Using a memory of motion to efficiently warm-start a nonlinear predictive controller," in *IEEE International Conference on Robotics and Automation*. ICRA, 2018.

[15] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization." in *Robotics: Science and Systems*, vol. 4. RSS, 2014.

[16] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel, "Combining model-based policy search with online model learning for control of physical humanoids," in *IEEE International Conference on Robotics and Automation*. ICRA, 2016.

[17] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *Robotics: Science and Systems*. RSS, 2015.

[18] A. Tamar, G. Thomas, T. Zhang, S. Levine, and P. Abbeel, "Learning from the hindsight plan—episodic mpc improvement," in *IEEE International Conference on Robotics and Automation*. ICRA, 2017.

[19] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2013.

[20] B. Landry, H. Dai, and M. Pavone, "Seagul: Sample efficient adversarially guided learning of value functions," in *Learning for Dynamics and Control*. PMLR, 2021.

[21] D. Hoeller, F. Farshidian, and M. Hutter, "Deep value model predictive control," in *Conference on Robot Learning*. CoRL, 2020.

[22] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A closer look at deep policy gradients," in *International Conference on Learning Representations*. ICLR, 2020.

[23] D. Q. Mayne, "Differential dynamic programming–a unified approach to the optimization of dynamic systems," in *Control and Dynamic Systems*. Elsevier, 1973, vol. 10.

[24] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the American Control Conference*, 2005.

[25] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *International Conference on Robotics and Automation*. ICRA, 2020.

[26] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu, "Sobolev training for neural networks," in *Advances in Neural Information Processing Systems*. Neural IPS, 2017.

[27] T. M. Mitchell, S. B. Thrun *et al.*, "Explanation-based neural network learning for robot control," in *Advances in Neural Information Processing Systems*. Neural IPS, 1993.

[28] R. Masuoka, "Noise robustness of ebnn learning," in *International Conference on Neural Networks*, vol. 2, 1993.

[29] J.-W. Lee and J.-H. Oh, "Hybrid learning of mapping and its jacobian in multilayer neural networks," *Neural computation*, vol. 9, no. 5, 1997.

[30] J. B. Witkoskie and D. J. Doren, "Neural network models of potential energy surfaces: Prototypical examples," *Journal of chemical theory and computation*, vol. 1, no. 1, 2005.

[31] S. Wright, J. Nocedal *et al.*, "Numerical optimization," *Springer Science*, vol. 35, 1999.

[32] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila, "Primitives for smoothing mobile robot trajectories," *Transactions on Tobotics and Automation*, vol. 11, no. 3, 1995.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[34] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," *Center for Cognitive and Neural Studies*, 2007.

[35] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, 1966.

[36] 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 Integration (SII)*. SICE, 2019.

[37] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.