IMT School for Advanced Studies, Lucca
Lucca, Italy

Learning optimal control policies from data: a partially model-based actor-only approach

PhD Program in Institutions, Markets and Technologies

Track in Computer Science and System Engineering

XXXII Cycle

By

Laura Ferrarotti

2022

The dissertation of Laura Ferrarotti is approved.

PhD Program Coordinator: Rocco De Nicola, IMT School for Advanced Studies Lucca

Advisor: Prof. Alberto Bemporad, IMT School for Advanced Studies Lucca

The dissertation of Laura Ferrarotti has been reviewed by:

Prof. Simone Formentin, Politecnico di Milano

Prof. Melanie Nicole Zeilinger, ETH Zürich

IMT School for Advanced Studies Lucca
2022

To my granny, unwitting master of optimization,
and to my mom, who never gives up

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to my advisor Prof. Alberto Bemporad for giving me the opportunity of pursuing a PhD and guiding me through the first steps in the research world. I will always remember with deep appreciation the combination of intelligence, expertise, and practical sense that characterized every advice that I received from him. My gratitude goes also to my co-author, role model and friend Valentina Breschi, for all the time she spent helping and encouraging me since the first day of our collaboration. I would like to thank Prof. Mario Zanon and Prof. Dario Piga, that generously took the time to listen to my ramblings and share their academic and personal wisdom with me more than once. Next, I would like to sincerely thank Prof. Simone Formentin and Prof. Melanie Nicole Zeilinger for reviewing this thesis and providing useful feedback.

Thanks to all current and former members of the DYSCO research unit for creating a kind and friendly work environment, that made me feel at home in the many days (and nights) spent together in the study room. In particular, I will always be grateful to Sampath Kumar Mulagaleti for teaching me so much about Control Systems, friendship, life and how to crack inappropriate jokes at any time. A special thanks goes to Nilay Saraf, for being a great travel companion and a truly caring friend, always forgiving me for my sharp tongue and "italian" manners. I thank all staff members at IMT, with a special thanks to Andrea Ceravolo, the kindest IT personnel I ever met.

Personally, I would also like to express gratitude to all my friends at IMT or away. I can't mention them all, but I would like at least to thank Marco Francischello, Giulia Muti, Teresa Guillen Hernandez,

and Niraj Rathod for being great friends. In addition, this aknowledgement would not be complete without thanking Ilaria Zampieri and Stella Simic, my two “Grazie”, that made PhD life an adventure literally since day one and are still there for me every day.

Finally, I can’t express how much I owe to my family, for being there through my journey. I will start by thanking Antonio for all the care, and for all the trips to Lucca and Pisa that I made him do. I would like to thank my brother Dario and his wife Anastasia, for opening their heart, their door and their fridge anytime to a stressed PhD student in the process of writing the thesis. I thank my father for telling me his stories and listening to mine. The biggest thanks, though, is for my mom, without whom I would not be here and I would not be me.

Vita

February 22, 1990	Born, La Spezia, Italy
2016-2022	Ph.D.,IMT School for Advanced Studies, Lucca Research Unit: Dynamical Systems, Control and Optimization Advisor: Prof. Alberto Bemporad
2015	Master Degree in Mathematical Engineering Final mark: 110/110 cum laude Politecnico di Torino, Torino, Italy
2013	Bachelor Degree in Mathematics Final mark: 102/110 University of Pisa, Pisa, Italy

Publications

1. L. Ferrarotti and A. Bemporad, "Synthesis of feedback controllers from data via optimal policy search and stochastic gradient descent," in *Proceedings of the 18th European Control Conference*, 2019.
2. L. Ferrarotti and A. Bemporad, "Learning optimal switching feedback controllers from data," *Proceedings of the 21st International Federation of Automatic Control (IFAC) World Congress*, 2020.
3. V. Breschi, L. Ferrarotti and A. Bemporad, "Cloud-based collaborative learning of optimal feedback controllers," *Proceedings of the 21st International Federation of Automatic Control (IFAC) World Congress*, 2020
4. L. Ferrarotti and A. Bemporad, "Learning nonlinear feedback controllers from data via optimal policy search and stochastic gradient descent," *Proceedings of the 59th IEEE Conference on Decision and Control*, 2020.
5. L. Ferrarotti, V. Breschi and A. Bemporad, "The benefits of sharing: a cloud-aided performance-driven framework to learn optimal feedback policies," *Proceedings of the 3rd Conference on Learning for Decision and Control*, 2021

Abstract

This dissertation presents new algorithms for learning optimal feedback controllers directly from experimental data, considering the plant to be controlled as a black-box source of streaming input and output data. The presented methods fall in the Reinforcement Learning "actor-only" family of algorithms, employing a representation (policy parameterization) of the controller as a function of the feedback values and of a set of parameters to be tuned. The optimization of a policy parameterization corresponds to the search of the set of parameters associated with the best value of a chosen performance index. Such a search is carried on via numerical optimization techniques, such as the Stochastic Gradient Descent algorithm and related techniques. The proposed methods are based on a combination of the data-driven policy search framework with some elements of the model-based scenario, in order to mitigate some of the drawbacks presented by the purely data-driven approach, while retaining a low modeling effort, as compared to the typical identification and model-based control design scenario.

In particular, we initially introduce an algorithm for the search of smooth control policies, considering both the online scenario (when new data are collected from the plant during the iterative policy synthesis, while the plant is also under closed-loop control) and the offline one (i.e. from open-loop data that were previously collected from the plant). The proposed method is then extended to learn non-smooth control policies, in particular hybrid control laws, optimizing both the local controllers and the switching law directly from data. The described methods are then extended in order to be employed in a collaborative learning setup, considering multi-agent systems characterized by heavy similarities, exploiting a cloud-aided scenario to enhance the learning process by sharing information.

Chapter 1

Introduction

This thesis addresses the problem of learning feedback controllers while considering the plant to be controlled as a black-box source of inputs and outputs. New iterative gradient-based methods are presented in the following chapters, considering both the online and offline settings, and relying on a combination of the data-driven policy search framework with the use of simple local models. This chapter introduces the main concepts on which this dissertation is built, namely model-based and data-driven control, Reinforcement Learning-based control and Stochastic Gradient Descent methods. Subsequently, the contributions and the outline of this dissertation are described in more detail.

1.1 Model-based and data-driven control synthesis

In modern control theory, the term "model-based control" indicates the set of control design techniques that are grounded on the explicit knowledge of a model of the object to be controlled, e.g. a state-space model or transfer function. Given that a model of the system is rarely known in advance, model-based methods are characterized by two sequential steps, as shown in Figure 1: a modeling phase and a control synthesis phase (Ogata, 2010). The modeling phase is dedicated to obtaining a model approximation of the underlying system to be controlled, typically derived from first principles, or by system identification techniques (Ogata, 2010), (Ljung, 1999). Once a model describing the system behavior is

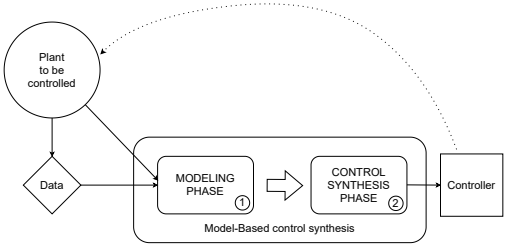

Figure 1: Model-based controller synthesis process.

available, the control synthesis phase exploits it to design a control law, using any of the classical techniques in literature.

The model-based approach has been largely and successfuly applied on many benchmark examples and classic control applications. However, relying on a model presents as well some critical points. For instance, deriving a model of the unknown dynamics that is both simple and reliable for control design is often hard and time-consuming (Formentin, van Heusden, and Karimi, 2014). On a side, a controller synthesized based on a model approximating the underlying plant is not necessarily optimal when connected to the plant itself. Unmodeled dynamics, unavoidable when the model structure associated with the real plant is not completely known, limit the performance of the controller in closed-loop. On the other side, a complex high-order model might require an excessively complex controller design that can be unwieldy in practice (Hou and Wang, 2013). Such situation, thus, usually requires additional effort in finding a reduced order linear model that is a good approximation of the original one at some operating point of interest (Anderson and Liu, 1989).

In order to avoid modeling an accurate high-order model to target high performance for a control system design, and then having to perform a controller order reduction or model simplification, two broad approaches are considered in literature: identification for control (I4C) and direct data-driven control synthesis. I4C pertains methods that minimize a control-related data-fitting cost, in order to tune the model identification towards the control objective for which the model is to be used (Gevers, 1993). Since the identification phase preceeds the synthesis of the controller whose performance should ideally drive

the model optimization, this leads to the use of iterative steps of controller updates and model updates obtained by closed-loop identification (Forssell and Ljung, 1999). The ideas behind identification for control have been recently exploited for data-driven learning, employing Bayesian optimization (Bansal et al., 2017), (Makrygiorgos et al., 2022), model adaptation for Model Predictive Control (MPC) (Piga, Forgione, et al., 2019), and in combination with neural architectures, in order to synthesize a control law stabilizing an unknown nonlinear dynamical system at an equilibrium point (Saha, Egerstedt, and Mukhopadhyay, 2021). I4C does not necessarily find the full model of the plant at hand, but instead can provide a fixed complexity model, optimized with respect to the closed-loop performance of an associated fixed complexity controller designed based on it. In (Hjalmarsson, Gevers, and De Bruyne, 1996) is shown that a full model closed-loop identification with a specific controller in the loop provides an estimated controller that achieves the best possible performance when tested on the actual system. If instead a low-order model and controller are synthesised, the controller does not necessarily converge to a local minimum of the design objective (Hjalmarsson, Gunnarsson, and Gevers, 1995). The identification of a low-order model is characterized by a *bias* and a *variance* component in the prediction error, associated with the simplified model inability of representing the true system and the noise and finiteness of the data, respectively. In order to reduced both the components, I4C methods require to design a control oriented identification experiment in order to obtain both a model and an uncertainty set; then, to design a new controller that achieves closed-loop stability and meets the required performance for all the scenarios in the chosen uncertainty set (Hjalmarsson, 2005). In (De Callafon and Van Den Hof, 1997), for instance, an uncertainty set is estimated from closed-loop experimental data, in terms of model perturbations in the dual Youla parametrization. It might be difficult, though, to model an appropriate uncertainty set, particularly in a control-oriented sense (Gevers, 2005).

In the I4C scheme the model is used specifically as a tool for the control performance objective minimization. Alternatively one can directly optimize the control goal with respect to parameters describing the controller, as done by direct data-driven techniques. Such methods base the design of controllers on knowledge extracted from a stream of input/output data from the controlled

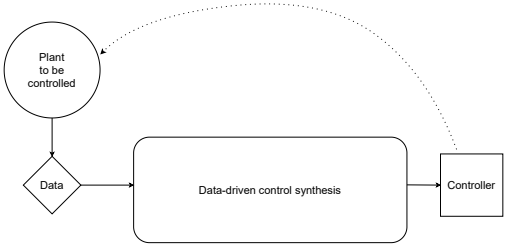

Figure 2: Direct data-driven controller synthesis process.

system, as shown in Figure 2, without using explicit or implicit information of the underlying dynamical system or of a mathematical model of such dynamics (Hou, Gao, and Lewis, 2017). Direct data-driven methods might differ on the learning paradigm: some techniques only use a batch of process data, meaning that learning is performed offline, while other methods learn while performing online experiments, executing sequential decision making steps.

The direct data-driven (also called "model-free") control design paradigm includes many techniques and the literature on the subjects is extensive. Some of the important methods in the field of direct data-driven control are, for instance iterative feedback tuning (IFT), correlation-based tuning, Virtual Reference Feedback Tuning (VRFT), and data-driven predictive control (DDPC). IFT (Hjalmarsson, Gevers, Gunnarsson, et al., 1998) runs a sequence of experiments on the plant in closed-loop with the last updated controller, and employs the collected data to estimate the gradient of the controller performance, that is then used to update the controllers parameters. As it will be introduced in Section 1.4 and detailed in Chapter 2, the policy search method proposed in this dissertation iteratively optimize the controller parameters via gradient-based iterations, as done by IFT. Differently by IFT, though, the proposed iterative approach is designed to include the case in which the controller learning is performed online while the plant is being controlled, avoiding to set up a sequence of closed-loop data-collection experiments. Moreover, while IFT requires two of such closed-loop experiments (the so called *normal* and *gradient* experiments) for each gradient estimation, the approach described in this thesis, by relying on information collected from data and encoded in simple local models for the gradient approximation, reduces the amount of expensive and possibly risky interactions with

the real plant. A similar approach is iterative correlation-based tuning (Karimi, Mišković, and Bonvin, 2004), that considers a user-defined desired closed-loop behavior and is based on minimizing the correlation between the reference signal and the desired and the achieved closed-loop output error. Such signals being uncorrelated implies that the closed-loop output error is mainly composed by the noise component (uncorrelated with the reference signal) and hence that the achieved closed-loop system captures the dynamics of the desired one. The solution of correlation equations involving chosen instrumental variables leads to parameter computation, carried out through iterative numerical steps. In order to avoid repeated experiment and synthesise the controller offline from data, a non-iterative data-driven correlation-based approach is propsed for LTI systems in (Karimi, Heusden, and Bonvin, 2007), convexifying the cost function associated with the model reference control problem optimization and solving the correlation equations in a non iterative data-driven fashion. Such method necessites though of a previously collected stream of input/output data, obtained by injecting a persistently exciting input signal. This characteristic, as well as the use of a user-defined desired model of the closed-loop behavior is shared as well by VRFT (Campi, Lecchini, and Savaresi, 2002). VRFT expresses a set of reference signals as a function of the designed feedback controller parameters by imposing that, if given in feedback to the controller together with the measurements collected in open-loop, such references would result in the designed controller generating the inputs present in the dataset. The parameters are optimized by minimizing the distance between the ouput obtained by feeding the parameterized references to the closed-loop model to be attained, and the collected plant output. Finally, the DDPC paradigm considers a receding horizon MPC scheme in which the model employed for prediction is substituted by the use of information extracted from data collected directly from the plant. For instance, according to the DeePC algorithm introduced in (Coulson, Lygeros, and Dörfler, 2019) the system trajectory is predicted using a finite number of open-loop data samples generated by a persistently exciting input signal. Such data is organized in Hankel matrices that, thanks to behavioral systems theory results, are proved to be a model for the generation of future trajectories of the plant. The method is initially built for controllable LTI systems, and then it is applied as well to control discrete nonlinear noisy systems, combined with regulariza-

tions in order to handle both the noise on data and the unaccounted nonlinearity of the original dynamics. Another example of DDPC is proposed in (Salvador Ortiz et al., 2018), where the model of the MPC scheme is substituted with a linear combination of past system trajectories in which the weights are optimized with respect to both the control performance and the variance of the estimation error.

Data driven techniques are nowadays applied in many classic control fields such as optimal control (Pang, Bian, and Jiang, 2019), (Goncalves da Silva et al., 2019), PID tuning (Fliess and Join, 2013) and nonlinear control (Novara et al., 2016), just to mention few examples. Among such contributions, (Fliess and Join, 2013) presents a technique for the data-driven tuning of intelligent proportional-integral-derivative (PID) controllers. This method is of particular interest with respect to the approach described in this thesis because they both rely on local models for the tuning of the controller parameters, abandoning the idea of estimating any good or global model of the plant to be controlled. (Fliess and Join, 2013) is based on a specific design for both the feedback control policy and the local model. In particular the control parameterization includes two groups of parameters: on one side the classic PID gains K_P, K_I, K_D associated with the tracking error, its integral and its derivative. On the other side two parameters α and F, shared as well with the parameterized "ultra-local" model $y^{(\nu)} = \alpha\, u + F$ (where $y^{(\nu)}$ is the ν-th order derivative of the output dynamics) considered in place of the unknown dynamics of the plant. The control parameterization is designed in a way that when combined with such model, the gains K_P, K_I, K_D are decoupled by the model parameters α and F. A linear differential equation involving the tracking error and its integral and derivative is obtained, exclusively parameterized in K_P, K_I, K_D. Such parameters can then be tuned offline in order to stabilize the equation, and hance obtain asymptotic convergence of the tracking error to 0. The model parameters are tuned in a different way: α is an a-priori fixed value, chosen based on the practictioner knowledge of the plant at hand. F instead is modeled as piecewise-constant, and it is constantly updated online by integrating algebraic formulae involving the inputs and outputs trajectories over short time lapses. Finally, the policy parameterization depends as well from a reference trajectory representing a modeled desired behavior, to be chosen again by the practictioner depending on the set

point to be tracked and on knowledge on the plant to be controlled. Differently from what was just described, the approach proposed in this thesis does not rely on any specific policy parameterization function and, although exploiting local linear models to keep the modeling effort to the minimum, can be effortlessly generalized in order to consider local models of any other shape. The tuning of the controller parameters is carried out by mini-batch stochastic gradient descent steps, without requiring expert knowledge on the considered plant. Moreover, the proposed method does not require a user-defined model of the output trajectory that, as indicated as well in (Fliess and Join, 2013), might be a sensitive tuning decision, particularly when the plant dynamics are fully unknown.

The mentioned literature shows a collection of promising methods for the model-free synthesis of controllers, demonstrating the interest of the community in developing this kind of techniques. At the same time, the absence of a model makes it hard to provide guarantees regarding characteristics of the methods such as stability, robustness and constraints satisfaction. Attempts are made in literature, in order to overcome such issues. Among the others: a data-driven approach to the model reference problem is proposed in (van Heusden, Karimi, and Bonvin, 2011). The method synthesizes linear fixed-order controllers and ensures closed-loop stability in the limit of infinite data, by adding a set of convex constraints to a non-iterative correlation-based scheme. In (Piga, Formentin, and Bemporad, 2018), a hierarchical control architecture that can handle constraints is synthesized directly from data, composed by an inner linear parameter-varying controller, matching a desired closed-loop model, and an outer MPC module, handling the constraints. The idea of enhancing learning systems with an additional MPC component in order to achieve safety is thoroughly surveyed in (Hewing et al., 2020). The survey considers as well the application of learning techniques to MPC controllers, dividing the contributions into model learning and controllers tuning categories. In (De Persis and Tesi, 2020) a parameterization of linear feedback systems is derived, that enables to reduce the stabilization problem to an equivalent data-dependent linear matrix inequality, without identification of the underlying system. However, the problems of stability, robustness and constraints satisfaction in data-driven scenarios continue to remain active subjects of research.

1.2 Reinforcement Learning and feedback control

Reinforcement Learning is a field of Machine Learning inspired by natural learning mechanisms, where animals adjust their actions based on reward and punishment stimuli received from the environment (Sutton and Barto, 1998). The classic Reinforcement Learning setup is composed by an actor (or agent) that interacts with the surrounding environment. The objective of Reinforcement Learning algorithms is to train the actor, in order for it to autonomously learn the best behavior to attain in the environment, given specific problem-dependent goals. Such objective is formulated as an optimization problem that the actor solves, based on stimuli received from the environment in response to its actions. It has been observed that Reinforcement Learning strategies in order to learn properly need to balance between decisions that are optimal with respect to the data observed up to the current time (*exploitative* decisions) and decisions that test new (possibly risky) options, in order to discover rewarding states and trajectories (*explorative* decisions) (Bertsekas and Tsitsiklis, 1996).

From the previous definitions, it is clear that there is a tight relationship between Reinforcement Learning and the concept of data-driven optimal decision making, that is obviously central in control problems where the system to be controlled is "black box". In (Powell, 2019) it is remarked how Reinforcement Learning and Stochastic Optimal Control are two research communities addressing decision problems, both evolved starting from a theoretical background based on Hamilton-Jacobi-Bellman equations. The view of Reinforcement Learning from the feedback control perspective is included as well in (Lewis, Vrabie, and Vamvoudakis, 2012), (Kiumarsi et al., 2018), where the authors observe that Reinforcement Learning-based feedback control design combines features of adaptive control (regarding the control of unknown systems using data measured in real time along the system trajectories) and Optimal Control (for what concerns the formulation of the learning as an optimization problem). Many contributions can be found in literature applying and adapting the most successful Reinforcement Learning schemes to the control design and Optimal Control problems, as for instance(Bradtke, Ydstie, and Barto, 1994), (Vamvoudakis and Lewis, 2009), (Mukherjee, Bai, and Chakrabortty, 2018), (Lillicrap et al., 2019) just to mention few important examples.

In the Reinforcement Learning framework, the key concepts of action (the control input) and cost-to-go are mathematically encoded as policy function and value function, respectively[1]. Reinforcement Learning methods are divided into three main groups, based on how they treat the policy and value functions: *actor-only* methods, *critic-only* methods, and *actor-critic* methods (Konda and Tsitsiklis, 2003), (Grondman, 2015). Critic-only methods rely on value function learning in order to approximate the solution of the Bellman equation (a procedure also indicated as Approximate Dynamic Programming, cf. Chapter 6 of (Bertsekas, 2005)), and then use it to derive actions. Actor-only methods avoid giving an overall estimate of the value function, as they work with a parameterized family of policies and search the optimal policy directly in the parameter space; for this reason they are also called *policy search* methods. Actor-critic methods, finally, combine both the previous classes, performing Approximate Dynamic Programming to estimate a value function, and then employing it to optimize the parameters of a policy function.

Critic-only methods learn an approximated value function which is then used to compute an associated decision policy. Such policy is often designed to select actions in a greedy fashion with respect to the estimated value function. The critic-only methods that follow the *value iteration* scheme at first iteratively approximate the optimal value function and then extract the optimal policy from it. The ones in accordance with the *policy iteration* scheme, instead, alternate updating the value function of the currently employed policy with updates of the policy based on the last value function obtained. Given that such scheme provides iterative updates of the the decision policy, it might seem to belong to the actor-critic set of methods, but this is not the case: for a method to be actor-critic (or actor-only) the decision policy constituting the actor module needs to be modeled directly as a parametric function, not indirectly via a value assessment. Instead in critic-only methods the decision policy is not a representation that can be directly manipulated, but is a consequence of the estimated value function. Following from this, the exploitation of the approximated value function for the

[1]Following the control literature notation, in this dissertation we consider the stimuli received by the agents to be costs instead of the rewards usually employed in Reinforcement Learning. Accordingly, the value function here is intended as a cumulative cost along a trajectory and the considered optimization problems will be formulated as problems of minimization of an expected cost, instead of maximizing a gain. The two points of views are equivalent, considering costs as negative rewards.

action selection entails an additional optimization procedure in every state encountered, that is both difficult due to the common phenomenon of insufficient smoothness in the approximated value function, and computationally inefficient if the action space is large or continuous, as it often happens in control applications. To employ critic-only methods in case of continuous actions space the most common approach is hence to discretize the action space. The discretization of the action space can reduce the computational cost by transforming such optimization into a search procedure on a look-up table, but indeed it forces the choice of possibly sub-optimal actions (Alibekov, Kubalík, and Babuška, 2018).

The use of a parameterized control policy, exploited both by actor-only and actor-critic methods, gives one the advantage of effectively dealing with a continuous set of actions and high dimensional, including continuous, state spaces. Moreover, a policy parameterization can easily incorporate domain knowledge or can be chosen such that it is meaningful for the task at hand, often leading to fewer parameters to be learned, with respect to traditional value function-based approaches (Peters and Schaal, 2008). Finally, while small changes in the value function might lead to abrupt changes in the derived greedy control policy in the critic-only scenario, a slow update of policy parameters (often guided by a learning rate) can provide smoother changes in the control policy, resulting in less risky online operations (Desienroth, Neumann, and Peters, 2011).

Reinforcement Learning methods, being based on value functions and policy functions, represent an advantageous framework, permitting to implicitly incorporate the dynamics identification into the learning method, as underlined by (Hafner and Riedmiller, 2011), that observes as well how the learning design, including the handling of the data sampling, can hence be concentrated on the regions that are important to the control application. However, as recollected in (Recht, 2018), modeling is still built into the assumptions of Reinforcement Learning methods, even in a model-free setup, for what concerns methods doted of a critic module: estimating value functions is based on assumptions on their structure, and hence requires modeling effort. Actor-critic methods optimize the actor parameters with respect to the approximation of the value function carried on by the critic module. This has as a consequence the propagation of eventual error from the critic to the actor (given by underrepresentative model structure or by poorly estimated parameters) (Desienroth, Neumann, and Peters, 2011).

Following such considerations, in order to learn controllers while reducing the modeling effort, we focus on policy search methods.

1.2.1 Policy search methods

The basic idea of policy search methods is based on the concept of episodic cost

$$R(\tau) = \sum_{t=0}^{L-1} r_t(s_t, u_t) + r_L(s_L),$$

evaluating the cost of an episode $\tau = (s_0, u_0, s_1, \ldots, u_{L-1}, s_L)$ [2] , composed by a sequence of input and states generated by the plant to be controlled, and on the policy parameterization π_θ. The function π_θ can be deterministic, i.e. a function $\pi_\theta : \mathcal{S} \longrightarrow \mathcal{A}$ that, given a state s_t in the state space $\mathcal{S}$, generates an input action $u_t = \pi_\theta(s_t) \in \mathcal{A}$. In alternative, stochastic control policies $\pi_\theta : \mathcal{S} \times \mathcal{A} \longrightarrow [0,\, 1]$ are widely considered, parameterizing the probability $\pi_\theta(u_t|s_t)$ that the action u_t is chosen when in state s_t. Using such concepts one can formulate the problem of finding $\theta^\star$ such that

$$\theta^\star = \arg\min_{\tau} \mathbb{E}_{\tau \sim T_\theta} \left[R(\tau)\right], \tag{1.1}$$

where the dependence of the goal function on θ is given by the distribution of a random variable T_θ representing the trajectories τ induced by the policy π_θ. Considering a stochastic policy $\pi_\theta(u_t|s_t)$, the law of distribution of T_θ is

$$p_{\pi_\theta}(\tau) = \mathbb{P}(s_0) \prod_{t=0}^{L-1} \mathbb{P}(s_{t+1}|s_t, u_t)\, \pi_\theta(u_t|s_t),$$

where $\mathbb{P}(s_{t+1}|s_t, u_t)$ is the transition probability of the system-environment dynamics. In case of a deterministic policy $\pi_\theta(s_t)$, instead, we have

$$p_{\pi_\theta}(\tau) = \mathbb{P}(s_0) \prod_{t=0}^{L-1} \mathbb{P}(s_{t+1}|s_t, \pi_\theta(s_t)).$$

[2] In this summary we consider the finite horizon scenario, that is common: many methods often perform episodic policy updates. When an infinite horizon is considered, the employed notation holds, with the addition of a discounting factor $\gamma \in (0,\, 1]$ in the trajectory cost.

Many model-free policy search methods tackle problem (1.1) by iterativly optimizing the policy parameters following the direction of steepest descent of the expected cost, indicated by the gradient of the goal function, motivated by the successful application of gradient methods in optimization. Moreover, gradient methods often allow the derivation of convergence guarantees as for instance in (Fazel et al., 2018), where conditions on the principal gradient updates are given to guarantee convergence to a globally optimal solution, in the case of linearized control problems, considering both sample and computational complexity. Starting from an initial guess θ_0, gradient-based methods compute at every iteration an updated set of parameters θ_{i+1} as

$$\theta_{i+1} = \theta_i - \alpha_i \, \nabla_\theta \mathbb{E}_{\tau \sim T_\theta} \left[R(\tau) \right],$$

in which the gradient $\nabla_\theta \mathbb{E}_{\tau \sim T_\theta} \left[R(\tau) \right]$ is approximated using data collected from controller-plant-environment interactions.

A simple method to approximate the required gradients from data is based on the Least Square-Based Finite Difference (LSFD) scheme (Chang, Hang, and Zhao, 2006). Starting from an initial guess θ_0, at every iteration i the method applys a set of perturbations δ_j to the parameter vector θ_i. Employing the policies induced by the perturbed parameters to generate trajectory from the plant we obtain $\Delta R_j = R(\tau(\theta_i + \delta_j)) - R(\tau(\theta_i))$. An approximation of the gradient of R with respect to θ is then obtained by using a first order Taylor-expansion of $R(\tau(\theta))$ and solving for the gradient in a least-squares sense, i.e., $\nabla_\theta R(\tau(\theta) = (\underline{\delta}' \, \underline{\delta})^{-1} \, \underline{\delta}' \, \Delta R$, where $\underline{\delta}$ is the matrix of the perturbations δ_j and ΔR is a vector containing the ΔR_j. This method is quite expensive in terms of number of interactions with the real-life plant, requiring at every iteration i the execution of a set of policies corresponding to the perturbed parameters $\theta_i + \delta_j$.

A definitely less expensive method is REINFORCE (Williams, 1992), that in order to disentangle the approximation of $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim T_\theta} \left[R(\tau) \right]$ from the use of the transition probabilities $\mathbb{P}(s_{t+1} | s_t, u_t)$ requires to employ stochastic control policies. Thanks to stochastic policies, it is indeed possible to apply the so-called "log-likelihood trick", and the gradient of the cost function $J(\tau)$ can be rewritten as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim T_\theta} \left[R(\tau) \left(\sum_{t=0}^{L-1} \nabla_\theta \log \pi_\theta(u_t | s_t) \right) \right].$$

The algorithm can then update the policy in episodic fashion, by collecting at iteration i a full trajectory $\tau^{(i)} = (s_0^{(i)}, u_0^{(i)}, \dots, s_{L-1}^{(i)}, u_{L-1}^{(i)})$ directly from the plant, computing the cost $R(\tau^{(i)})$ and then proceeding to update the parameters as

$$\theta_{i+1} = \theta_i - \alpha_i \, R(\tau^{(i)}) \Big(\sum_{t=0}^{L-1} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \Big), \tag{1.2}$$

without employing the unknown transition probabilities of the system. It is important to underline that the use of deterministic policies does not allow to employ the "log-likelihood trick" and hence to eliminate the dependency of the optimization direction from the system dynamics. For this reason, the stochasticity of the policy parameterization is a fundamental requisite to employ such methods.

It has been observed that the proposed gradient-based updates experience very high variance (Peters and Schaal, 2008), (Desienroth, Neumann, and Peters, 2011), (Grondman, 2015), (Recht, 2018). This variance arises from a combination of employing stochastic policies (in order to avoid the use of a model), and inherent properties of the approach. As can be seen from the Bellman's equation, though, the optimal policy for Problem (1.1) (the greedy policy associated with the optimal value function) is always deterministic (Recht, 2018), making the choice of a different class of functions counterintuitive.

From optimization theory we know that the convergence speed of stochastic gradient methods will be negatively affected by the variance of the gradient (Bottou, Curtis, and Nocedal, 2018). For this reason efforts can be found in literature to reduce the variance in the gradient estimates, via the introduction of baselines, tuned for this goal (Greensmith, P. Bartlett, and Baxter, 2002), (Hofmann et al., 2016), or via modifications in the algorithm, like in the case of G(PO)MDP (Baxter and P. L. Bartlett, 2001) and the policy gradient theorem algorithm (Sutton, McAllester, et al., 1999), that exploit the intuiton that future actions do not depend on past rewards to distribute the step rewards $r_t(s_t, u_t)$ in (1.2) among the gradients terms, with the rationale that longer trajectories increase the gradient variance.

Methods that rely on some form of model for the optimization of the policy are less affected by the mentioned effects. In literature policy search applications

based on a model rely mostly on Locally Weighted Bayesian Regression (Atkeson, Moore, and Schaal, 1997) and Gaussian Process Regression (Rasmussen, 2003), as for instance (Bagnell and Schneider, 2001), (Ko et al., 2007). The employed models are adaptive global models of the system dynamics, updated online via episodic interaction with the real system. Although they are not obtained through a preliminary modeling or identification phase, but rather adapted online, they suffer of all the issues related to modeling effort, modeling error and robustness discussed in Section 1.1 (Desienroth, Neumann, and Peters, 2011).

In this dissertation policy search methods for the synthesis of deterministic control policies from data are proposed, applicable both offline and online, avoiding the episodic paradigm of learning in favor of a "learning-while-controlling effort". The fundamental idea behind this work (explained in more detail in Section 1.4 and Chapter 2) is to design a data-driven approach for estimating the gradient based on local linear approximations of the system dynamics, updated online and not trained to provide information on the overall (possibly nonlinear) dynamics of the plant. Due to their lack of generalization capabilities such local models are valid and employed only in specific neighborhoods of the visited space, with the sole scope of providing a tool for gradient approximation. By combining the gradient-based framework with some elements of the model-based scenario, the idea is to leverage between the discussed drawbacks of the data-driven and model-based classes of methods. In particular on one side we succeed in mantaining the modeling effort extremely low, employing linear objects for the optimization of controllers on nonlinear plants, and avoiding a preliminar identification phase. On the other side the method is capable of learning (and applying online) policies that are deterministic, reducing both the variance affecting the gradients and the risk of randomic operations online on the plant.

1.3 Stochastic Gradient Descent methods

This section provides a short recap of the numerical optimization methods employed in the dissertation, in particular of the mini-batch gradient-based methods that solve stochastic optimization problems.

Considering a function $F(z, \xi)$, mini-batch gradient methods attempt at computing

$$z^* = \arg\min_z \mathbb{E}_{\xi \sim \Xi} \left[F(z, \xi) \right] \tag{1.3}$$

where the expectation in the goal function is considered with respect to the random variable Ξ.

In particular, in this dissertation we focus on the mini-batch Stochastic Gradient Descent (SGD) algorithm (Robbins and Monoro, 1951) and some of its faster variants, i.e., RMSProp (Tieleman and Hinton, 2012), Adam (Kingma and Ba, 2015) and AMSGrad (Reddi, Kale, and S. Kumar, 2019). Such methods are studied and compared in literature (Ruder, 2017), (Bottou, Curtis, and Nocedal, 2018). The cited literature provides proofs of convengerce (at least to a local minimum) for the aforementioned methods, under opportune assumptions on the smoothness of F, and provided that the sequence of learning rates $\{\alpha_t\}_t$ is suitably chosen. In practice, though, such methods are widely applied, even outside such hypothesis, especially in Machine Learning.

Mini-batch SGD is the most classic of the mini-batch methods; the SGD algorithm starts from an initial guess z_0 as a solution of Problem (1.3) and iteratively updates it, by implementing at every step t an update rule of the form

$$z_t = z_{t-1} - \alpha_t \; D_t(\mathcal{G}_F(z_{t-1})), \tag{1.4}$$

where the optimization direction $D_t(\mathcal{G}_F(z_{t-1}))$ is the averaged gradient in z over the mini-batch $\{ \xi_i^{(t)} \}_{i=1}^{N_b}$ sampled at instant t according to Ξ, that is, $D_t(\mathcal{G}_F(z)) = \mathcal{G}_F(z)$, with

$$\mathcal{G}_F(z) = \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla_z F(z, \; \xi_i^{(t)}).$$

RMSProp is developed from mini-batch SGD by adding adaptive learning rates for each of the parameters components. This permits to achieve more flexibility in the updating of the different components, considering that the components of the gradient might differ in magnitude, and hence require to be updated at a different rate. Hence, the learning rates and optimization direction at instant t

are obtained as

$$\alpha_t = \frac{\alpha}{\sqrt{v_t + \epsilon}}, \qquad D_t(\mathcal{G}_F(z_{t-1})) = \mathcal{G}_F(z_{t-1}).$$

In the previous equation the learning rates are automatically computed by dividing a constant α by the squared root of the weighted sum v_t of the squared mini-batch gradients, i.e.,

$$v_t = \beta\, v_{t-1} + (1-\beta)\, \mathcal{G}_F(z_{t-1})^2,$$

and the squared root and the power are meant as componentwise operations. The weight β is chosen in $[0,\ 1]$, acting as forgetting factor. A small positive constant ϵ is added in the denominator, to avoid dividing by zero. This setup eliminates the need to manually tune the learning rate, that is usually a difficult task in the algorithm design.

If the function F is complex, the gradients will change dramatically in different, but possibly close, areas of the space. Hence, updating the variable z_t according to such gradients would result in almost random updates, making the optimization process slower. On the other side, if a value of z_t is reached such that the corresponding surface of the function F is flat, the associated gradients reduce, making it difficult for the algorithm to move away and find the optimal value. In order to avoid such phenomena some methods introduce the concept of momentum. The momentum, in practice, is a smoothed optimization direction obtained as a weighted average of the past gradient values. In the classic momentum definition, the weight on the past gradient values is an additional hyperparameter, to be chosen in $[0,\ 1]$ in the tuning phase. Adam, instead, combines the adaptive learning rates introduced in RMSProp with the use of an adaptive estimation of the momentum, computed as an exponentially decaying average of past gradients, employed as optimization direction, i.e.,

$$\begin{aligned}
m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, \mathcal{G}_F(z_{t-1}), \\
v_t &= \beta_2\ v_{t-1} + (1-\beta_2)\, \mathcal{G}_F(z_{t-1})^2, \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\
\alpha_t &= \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}}, \qquad D_t(\mathcal{G}_F(z_{t-1})) = \hat{m}_t.
\end{aligned}$$

Finally, AMSGrad is a slightly modified version of Adam that endowes the algorithm with a long-term memory of past gradients, solving some issues given by the essentially fixed sized window of past gradients employed by Adam. The associated learning rate and optimization direction are

$$\alpha_t \quad = \quad \frac{\alpha}{\sqrt{\overline{v}_t + \epsilon}}, \qquad D_t(\mathcal{G}_F(z_{t-1})) = \hat{m}_t,$$

where the learning rate α_t is defined based on $\overline{v}_t = \max(\overline{v}_{t-1}, \hat{v}_t)$.

1.4 Structure of the dissertation

This dissertation presents new algorithms for learning optimal feedback controllers directly from experimental data, considering the plant to be controlled as a black-box source of inputs and outputs. The proposed methods fall under the Reinforcement Learning-based feedback control design umbrella, belonging to the policy search family of algorithms. In this work deterministic policy parameterizations are employed, representing the feedback controllers as a function of the values in feedback, and of a set of parameters to be learned. The optimization of a policy parameterization corresponds to the search of the set of parameters associated with the best value of a chosen performance index. For all the proposed algorithms, both offline and online settings are considered. In the first case the learning process occurs on a dataset previously collected from the plant; in the second case instead, the control law that is being optimized iteratively is directly applied to the plant to perform the desired task and collect the successive data sample. Additionally, although a state observer on the plant can be employed as a source of feedback data, the dissertation focuses on an ARX-formulation, building a measured state composed by a set of past input and (possibly noisy) output measurements, collected from sensors on the plant itself, and hence not requiring any effort in observing the actual state of the plant.

Following the discussion presented in Section 1.1, the methods are designed not to require an initial identification of a model of the process dynamics, with the idea of lowering the effort intrinsic to the modeling challenge. In this sense the proposed control design approach is not model-based. Nonetheless, it is not completely model-free, relying on simple local linear models estimated online from the input/output data stream and valid in specific neighborhoods of the

states visited by the system. The purpose of such local linear models is not to provide information on the overall dynamics of the plant, that could be possibly nonlinear, but rather merely to approximate the gradient of the performance index. The gradient will be employed to drive the optimization of the control law, according to numerical optimization techniques, such as the mini-batch Stochastic Gradient Descent techniques introduced in Section 1.3.

On one side, the effort required to compute the local linear models is indeed lower than the one necessary to obtain a simple but reliable model of an unknown system, based on first principles or on system identification techniques. On the other side, collecting the states visited by the system and the associated local linear models, and using them in mini-batch in order to compute the gradient-based parameter updates results to be a sufficiently accurate data-driven gradient approximation, capable of synthesizing good control policies. Moreover, the use of a deterministic control policy, together with the sampling and averaging of a mini-batch of collected local models at every step can have a positive effect over the variance of the approximated gradients.

The next chapters are organized as follows:

- In Chapter 2, after the required problem formulation, the basic elements characterizing the Optimal Policy Search (OPS) approach are introduced, in particular the designed data-driven approach to stochastic optimization. The online and offline OPS algorithms for the synthesis of smooth feedback controllers are detailed. The content of this chapter is mainly based on (Ferrarotti and Bemporad, 2019), (Ferrarotti and Bemporad, 2020a).

- Chapter 3 is dedicated to the application of the OPS method to the tracking of a-priori unknown reference signals. The tracking problem is formulated as a policy search problem, then numerical examples are included, demonstrating the performance of the proposed approach in handling such problem for both linear and nonlinear plants, synthesizing linear feedback controllers and nonlinear ones (i.e., neural networks). The content of this chapter is mainly based on (Ferrarotti and Bemporad, 2019), (Ferrarotti and Bemporad, 2020a).

- In Chapter 4 the OPS method is extended to the OSPS method for the synthesis of non-smooth (hybrid) controllers directly from data, handling the learning of both the set of local controllers and the switching law. Given that the policy parameterization is not smooth, additional care is given to avoid the computation of the gradients necessary for the OPS steps in areas of the parameters space where the performance index is not continuous and hence not differentiable. Numerical examples of the application of OSPS to hybrid or nonlinear systems for the synthesis of tracking policies are included. The content of this chapter is mainly based on (Ferrarotti and Bemporad, 2020b).

- Chapter 5 incorporates the Optimal Policy Search approach in a collaborative multi-agent framework for the learning of optimal feedback control laws, considering multi-agent systems characterized by structural similarities, exploiting a cloud-aided scenario. Two collaborative learning strategies are designed, based on consensus and on evaluation of levels of trust among the agents respectively. The resulting enhancement of learning performance with respect to the single agent scenario are shown, by applying the extended approach to the output-tracking problem. The content of this chapter is mainly based on (Breschi, Ferrarotti, and Bemporad, 2020), (Ferrarotti, Breschi, and Bemporad, 2021).

Concluding remarks and notes on possible future research are included in Chapter 6.

1.5 List of symbols

This section contains a list of the main mathematical symbols employed in this dissertation:

$\mathbb{R}^n$	set of the vectors with n real elements
$\mathcal{P}(\mathbb{R}^n)$	set of subsets of $\mathbb{R}^n$
$[x]_i$	i-th element of $x \in \mathbb{R}^n$
$\{x_\ell\}_{\ell=0}^{\infty}$	infinite sequence of elements x_ℓ
$\mathbb{R}^{n \times m}$	set of the real-valued matrices with n rows and m columns
$M_{i,j}$	element on the i-th row, j-th column of $M \in \mathbb{R}^{n \times m}$
$\underline{0}_{n \times m}$	$n \times m$ matrix of zeroes
$\underline{1}_{n \times m}$	$n \times m$ matrix of ones
I_n	$n \times n$ identity matrix, $(I_n)_{i,i} = 1$, $(I_n)_{i,j} = 0$ otherwise
$M' \in \mathbb{R}^{m \times n}$	transpose of $M \in \mathbb{R}^{n \times m}$
M^{-1}	inverse of $M \in \mathbb{R}^{n \times n}$, $M M^{-1} = I_n = M^{-1} M$
$M \succeq 0$	semi-definite positive matrix
$M \succ 0$	definite positive matrix
$\|x\|$	absolute value of $x \in \mathbb{R}$
$\|\|x\|\|$	generical norm of $x \in \mathbb{R}^n$
$\|\|x\|\|_2$	Euclidean norm of $x \in \mathbb{R}^n$, $\|\|x\|\|_2 = \sqrt{x' x}$
$\|\|x\|\|_M$	weighted norm of $x \in \mathbb{R}^n$, $\|\|x\|\|_M = \sqrt{x' M x}$, with $M \in \mathbb{R}^{n \times n}$
$[a, b]$	set of $x \in \mathbb{R}$ such that $a \leq x \leq b$
$[a, b)$	set of $x \in \mathbb{R}$ such that $a \leq x < b$
$(a, b]$	set of $x \in \mathbb{R}$ such that $a < x \leq b$
(a, b)	set of $x \in \mathbb{R}$ such that $a < x < b$
$f : D \to C$	function with domain D, taking value in C

$\mathcal{F}(D,\, C)$	set of all the functions with domain D, taking value in C
$f\Big\vert_E : E \to f(E)$	restriction of $f : D \to C$ to $E \subset D$
$\partial f(x)/\partial [x]_i$	partial derivative of f with respect to $[x]_i$
$\nabla_x f(x)$	vector of $\partial f(x)/\partial [x]_i$ for each element $[x]_i$ of $x \in \mathbb{R}^n$
$C^1(\mathbb{R}^n)$	set of differentiable functions over $\mathbb{R}^n$, with continuous derivatives over $\mathbb{R}^n$
$id : D \to D$	identity function, $id(x) = x$
$w \sim W$	w sampled from the random variable W
$\mathbb{E}_w[f(x, w)]$	expected value of $f(x, w)$ with respect to $w \sim W$
$\mathcal{N}(\mu,\, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$\mathcal{U}(A)$	uniform distribution over set A
$\mathring{A}$	interior of set A
$\overline{A}$	closure of set A
∂A	boundary of set A
$\mathbb{P}(M)$	set of permutations of M elements
$\mathcal{S}_{M,L}$	set of L-length sequences of M objects

Chapter 2

Learning smooth feedback controllers from data

This chapter introduces the basic elements composing the Optimal Policy Search approach for the synthesis of feedback controllers. After describing the problem formulation and the approximations necessary to make the problem tractable in practice, the techniques used to deal with the unknown plant dynamics and the algorithms for the online and offline learning will be described in detail.

2.1 The Optimal Policy Search problem

Let P be a strictly causal discrete-time plant, with dynamics

$$\begin{cases} \overline{x}_{t+1} &= f(\overline{x}_t,\ u_t,\ w_t^f\ ), \\ \overline{y}_t &= g(\overline{x}_t,\ w_t^g\ ), \end{cases} \tag{2.1}$$

in terms of input $u_t \in \mathbb{R}^{n_u}$, output $\overline{y}_t \in \mathbb{R}^{n_y}$, exogenous non measurable disturbances $w_t^f \in \mathbb{R}^{n_w^f}$, $w_t^g \in \mathbb{R}^{n_w^g}$ and inner state $\overline{x}_t \in \mathbb{R}^{n_{\overline{x}}}$, as shown in Figure 3.

Our aim is to design a feedback controller generating command inputs u_0, u_1, u_2, ... capable of driving the plant P to perform a given task, without an explicit knowledge of model (f, g) in (2.1). The classic control goals of output regulation and reference tracking can be considered as possible tasks, as well as more general objectives, generally exemplified by economic costs.

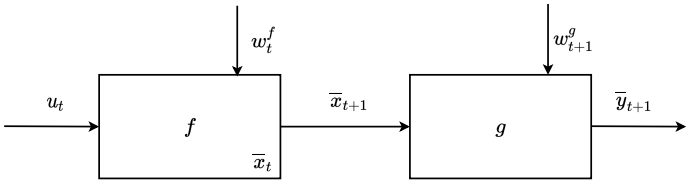

Figure 3: Diagram of the considered strictly causal plant P.

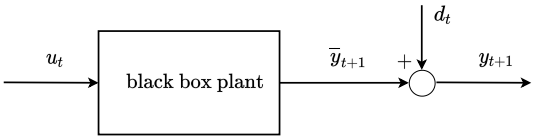

Figure 4: Diagram of plant P under "black box plant" hypothesis.

To formulate this problem, a set $(s_t, p_t) \in \mathbb{R}^{n_s} \times \mathbb{R}^{n_p}$ of decision variables is considered, collecting the signals composing the feedback for the controller. The vector s_t is designed to contain all the Markovian signals relevant to the task to be performed by the plant, while p_t is a vector of non-Markovian exogenous signals, such as measured disturbances, time-varying parameters, and reference signals to track. In particular, s_t contains a set of elements x_t composed by the available information useful to currently describe the plant P to be controlled. Eventually, s_t can also include a set of other variables z_t that are significant for the decision process, representing relevant interactions of P with the environment, like the integral of tracking errors with respect of a reference signal in case of output tracking task, i.e., $s_t = [\, x_t \,,\, z_t \,]'$. Note that when the inner state $\overline{x}_t$ of plant P introduced in (2.1) is observable it may be used to describe the state of P, hence x_t can be set to $\overline{x}_t$, i.e., $x_t = \overline{x}_t$. In the remaining part of this dissertation, though, we will assume not to observe the state $\overline{x}_t$. The control design can be based on an input/output model representation, following the ARX paradigm. A finite collection of the most recent samples of the input and output of the plant is used to build x_t, as

$$x_t = [\, y_t' \;\ldots\; y_{t-n_o+1}' \; u_{t-1}' \;\ldots\; u_{t-n_i}' \,]' \in \mathbb{R}^{n_x}, \tag{2.2}$$

where we consider $n_x = n_y n_o + n_u n_i$.

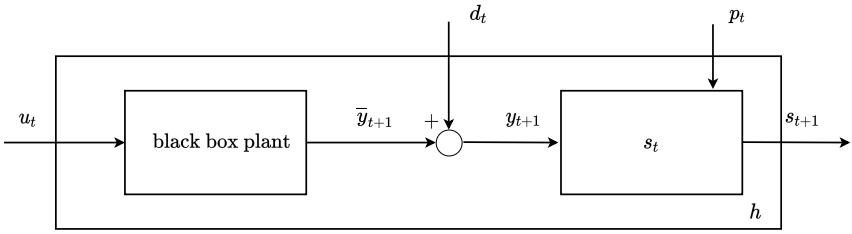

Figure 5: Diagram of the decision variables dynamics h.

Additionally, the model of the plant is assumed to be a "black box": P is used exclusively as generator of input and output (u_t, y_{t+1}), to be measured thanks to (possibly noisy) sensors, as shown in Figure 4, where the vector d_t represents additive noise over the measured output of the system. The dynamics (f, g) in (2.1) are considered to be unknown in the control design phase. Based on this assumption, the temporal evolution of the decision variable s_t

$$s_{t+1} = h(s_t,\, p_t,\, u_t,\, d_t), \tag{2.3}$$

represented in Figure 5, is supposed to be unknown as well, depending on (f, g), and therefore not used for control design purpouses.

Fixed a set of decision variables (s_t, p_t), the problem representation requires to design a differentiable function $\rho : \mathbb{R}^{n_s+n_p+n_u} \to \mathbb{R}$ called the *stage-cost* function. The value $\rho(s_t,\, p_t,\, u_t)$ represents the cost of applying an action u_t to P while in $(s_t,\, p_t)$, thus evaluating the performance of u_t in steering plant P to accomplish an assigned task.

In general, a task is going to require more than one action to be accomplished: we define a trajectory as the sequence of states, controls, exogenous signals and disturbances $(\, s_\ell,\, u_\ell,\, p_\ell,\, d_\ell\,)$ encountered while accomplishing a task. Notice that, fixed a (finite or infinite) sequence of control input $\{u_\ell\}_\ell$, the associated trajectory depends only on the initial state s_0, and on the sequences $\{p_\ell\}_\ell$, $\{d_\ell\}_\ell$ of exogenous signals and disturbances. In case of infinite horizon

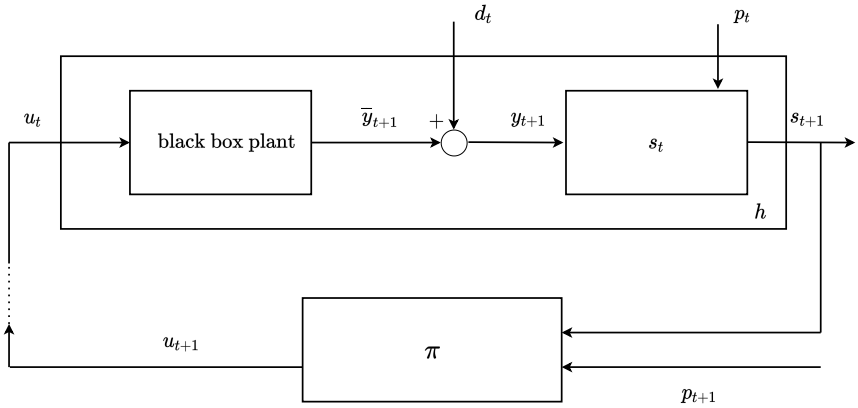

Figure 6: Diagram of dynamics h in closed-loop with the policy π.

tasks, the trajectory cost is the sum of the associated stage-costs

$$J_\infty(\{u_\ell\}_{\ell=0}^{\infty}, w) = \sum_{\ell=0}^{\infty} \rho(s_\ell, p_\ell, u_\ell), \tag{2.4}$$
$$\text{such that} \quad s_{\ell+1} = h(s_\ell, p_\ell, u_\ell, d_\ell), \quad \ell = 0, 1, \ldots$$

where we name $w = (s_0, \{p_\ell, d_\ell\}_{\ell=0}^{\infty})$ the vector collecting the initial state s_0, and the values p_0, d_0, p_1, $d_1, \ldots$ of exogenous signals and disturbances. Considering an episodic task, instead, that is a finite horizon task with horizon of length T, the trajectory cost is defined as

$$J_T(\{u_\ell\}_{\ell=0}^{T}, w) = \sum_{\ell=0}^{T-1} \rho(s_\ell, p_\ell, u_\ell) + \rho_T(s_T, p_T), \tag{2.5}$$
$$\text{such that} \quad s_{\ell+1} = h(s_\ell, p_\ell, u_\ell, d_\ell), \quad \ell = 0, 1, \ldots, T\text{ - }1$$

where $w = (s_0, \{p_\ell\}_{\ell=0}^{T}, \{d_\ell\}_{\ell=0}^{T-1})$ collects the initial state s_0 and a finite sequence of exogenous signals and disturbances. As shown in (2.5), such cost might include a term $\rho_T(s_T, p_T)$, weighting the distance of the final state of the trajectory from a specific goal state for the episodic task.

The feedback controller to be designed is represented as a deterministic *policy function* $\pi : \mathbb{R}^{n_s+n_p} \to \mathbb{R}^{n_u}$ that associates to each s_t and p_t an action $u_t = \pi(s_t, p_t)$, automatizing the decision making process, as represented in Figure 6. As discussed in Section 1.2, data-driven policy search methods often require to synthesize stochastic policies, i.e., $\pi : \mathbb{R}^{n_s+n_p} \times \mathbb{R}^{n_u} \to [\,0, 1\,]$, such that $\int_{\mathbb{R}^{n_u}} \pi(s_t, u)\ du = 1$ for each $s_t \in \mathbb{R}^{n_s+n_p}$. In the control field, though, it is quite natural to prefer a deterministic controller over a stochastic one, given the uncertainty a stochastic law may inject into the closed-loop behavior. Moreover, as can be seen from the Bellman's equation, it always exists an optimal policy that is deterministic: such policy is the one that acts greedily with respect to the optimal value function V_*, i.e., $\pi(s_t, p_t) = \arg\min_u \rho(s_t, p_t, u) + V_*(s_{t+1})$, where s_{t+1} depends on u by means of (2.3). In the following we will hence consider deterministic controllers.

We are interested in finding the controller π minimizing the trajectories cost, prioritizing trajectories based on their realization probability. This is expressed by considering the expectation of the trajectory cost with respect to the trajectories realization probability, that is

$$\begin{aligned} \mathbb{E}_W[J](\pi) &= \mathbb{E}_w[J_\infty(\,\{\,\pi(s_\ell,\, p_\ell)\}_{\ell=0}^{\infty},\, w\,)], \\ \text{such that} \quad & s_{\ell+1} = h(s_\ell,\, p_\ell,\, \pi(s_\ell,\, p_\ell),\, d_\ell), \quad \ell = 0, 1, \dots \\ & w \sim W, \end{aligned} \tag{2.6}$$

where J_∞ is defined in (2.4), and

$$\begin{aligned} \mathbb{E}_W[J](\pi) &= \mathbb{E}_w[J_T(\,\{\,\pi(s_\ell,\, p_\ell)\}_{\ell=0}^{\infty},\, w\,)], \\ \text{such that} \quad & s_{\ell+1} = h(s_\ell,\, p_\ell,\, \pi(s_\ell,\, p_\ell),\, d_\ell), \quad \ell = 0, \dots, T\text{ - }1 \\ & w \sim W, \end{aligned} \tag{2.7}$$

with J_T defined in (2.5). In both cases the trajectories realization probability corresponds to the realization probability of the random vector W, containing S_0 (assuming the value of a possible initial state of the trajectory), and a collection of P_ℓ, D_ℓ representing the random values of the signals p_ℓ, d_ℓ at step ℓ. The collection of P_ℓ, D_ℓ in W can have infinite or finite cardinality, depending on the infinite or finite horizon setup we are in, respectively.

Based on (2.6) - (2.7), the optimal feedback controller corresponds to

$$\pi^* = \arg \min_{\pi \in \mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})} \mathbb{E}_W[J](\pi), \tag{2.8}$$

where $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ is the set of functions of $n_s + n_p$ real variables taking values in $\mathbb{R}^{n_u}$. Equation (2.8) represents a general abstract Optimal Policy Search (OPS) problem.

Aiming at approximating the solution of problem (2.8), we consider a Model Predictive Control (MPC)-like iterative optimization routine, that at every instant t of the control procedure solves

$$\begin{aligned}
\pi_t^* = \arg \quad & \min_{\pi} \quad \mathbb{E}\left[\sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, u_\ell) + \rho_L(s_L, p_L) \right], \\
\text{such that} \quad & u_\ell = \pi(s_\ell, p_\ell), \quad \ell = 0, 1, \ldots, L\text{-}1, \\
& s_{\ell+1} = h(s_\ell, p_\ell, u_\ell, d_\ell), \quad \ell = 0, 1, \ldots, L\text{-}1, \\
& s_0 = s_t \\
& d_\ell \sim D_\ell, \quad \ell = 0, 1, \ldots, L\text{-}1, \\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \ldots, L.
\end{aligned}$$

where s_t is the state at instant t and a finite horizon $L \leq T < \infty$ is employed, eventually together with a terminal cost ρ_L. At every iteration t the MPC scheme impose to solve such problem, act on the plant using $u_t = \pi_t^*(s_t, p_t)$, and collect the next state s_{t+1}.

Through this MPC scheme a sequence $\{\pi_t\}_t$ of feedback laws is computed. Each optimization problem is characterized by the initial condition $s_0 = s_t$ and the sliding window optimization finds a specific optimal policy π_t^* for each s_t reached by the plant.

As formulated in the OPS problem (2.8), though, we are interested on a static time-invariant feedback controller, and hence on a policy capable to perform optimally on average from every possible reachable state. With this in mind, we formulate a time-invariant problem where the expected value is taken with respect to the random trajectories of P_ℓ and D_ℓ and the random vector S_P representing all the states reachable by plant P.

In this scenario S_P is used as generator of initial states for the trajectory, substituting the sliding-window mechanism of the MPC-like control strategy, as

shown in the following formulation:

$$\begin{aligned}
\pi^* = \arg \quad & \min_{\pi} \quad \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho(s_\ell,\, p_\ell,\, u_\ell) + \rho_L(s_{_L},\, p_{_L})\Big], \qquad (2.9)\\
\text{such that} \quad & u_\ell = \pi(s_\ell,\, p_\ell), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_{\ell+1} = h(s_\ell,\, p_\ell,\, u_\ell,\, d_\ell), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_0 \sim S_P\\
& d_\ell \sim D_\ell, \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \ldots, L.
\end{aligned}$$

As for the MPC formulation, the shortening of the horizon is necessary in order to make the problem tractable. The terminal cost ρ_L can be designed to approximate the original infinite horizon cost.

Again for tractability, we restrict the generic set $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ in Problem (2.8) to a subset of feedback controllers, by choosing a *policy parameterization*. A policy parameterization is an analytic form that expresses a relationship between u_t and the decision variables $(s_t,\, p_t)$, in function of a vector $H \in \mathbb{R}^{n_h}$ of parameters. We consider policy functions that are differentiable and whose derivatives are continuous with respect to the parameters H, i.e., for each $(s_t,\, p_t) \in \mathbb{R}^{n_s+n_p}$ the policy $\pi_H(s_t,\, p_t) \in C^1(\mathbb{R}^{n_h})$. Denoting a policy parameterization by $\pi_H(s_t, p_t)$, the optimization problem (2.8) is therefore transformed into

$$\begin{aligned}
H^* = \arg \quad & \min_{H} \quad \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho(s_\ell,\, p_\ell,\, u_\ell) + \rho_L(s_{_L},\, p_{_L})\Big], \qquad (2.10)\\
\text{such that} \quad & u_\ell = \pi_H(s_\ell,\, p_\ell), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_{\ell+1} = h(s_\ell,\, p_\ell,\, u_\ell,\, d_\ell), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_0 \sim S_P\\
& d_\ell \sim D_\ell, \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \ldots, L.
\end{aligned}$$

By parameterizing the policy, we simplify Problem (2.9), passing from the optimization over a generical space of functions $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ to optimizing

a vector of parameters $H \in \mathbb{R}^{n_h}$. Moreover, eventual knowledge on efficient feedback controllers that is gathered from the established control theory and applications can be incorporated in the structure of the controller, specifying the relationship between the input generated by the controller and the feedback collected from the plant and the environment.

We underline that problem (2.10) is not necessarily convex with respect to the parameters H. Even if the stage cost ρ are designed to be convex in s_ℓ, u_ℓ, and the policy π_H is convex in H, the composition of such costs and policy with the unknown dynamics h that generate the sequence $\{s_\ell\}_\ell$ could still result in a non-convex optimization problem. Even in the simple case of a linear system and a linear policy parameterization, i.e., $s_{t+1} = A\, s_t + B\, u_t + E\, p_l + D\, d_t$ and $u_t = H_s\, s_t + H_p\, p_t$, if we considering the elements of $\{s_\ell\}_\ell$ in function of the parameters $H = (H_s, H_p)$, we have that

$$s_{\ell+1} = (A + B\, H_s)^\ell\, s_0 + \sum_{j=0}^{\ell-1} (A + B\, H_s)^{j-\ell-1}\, (E + B\, H_p)\, p_j + D d_t.$$

The convexity of such function in H is not always matched, depending on the matrices (A, B, E).

Observing the goal function of Problem (2.10), it corresponds to the expected cost of applying the parameterized policy π_H to the plant P for L steps, following the dynamics h introduced in (2.3). As anticipated in Chapter 1 and previously in this section, we work under the assumption not to know the nominal model of the plant. The dynamics described by (f, g) in (2.1) and hence by h in (2.3) will not be used for the control policy synthesis. Section 2.2.2.2 describes the techniques employed to approximate the solution of Problem (2.10) while considering the system (2.3) as a "black-box", and deriving a serie of local linear models from data, without requiring the identification of the overall dynamics of the system.

2.2 Data-driven approach to the OPS problem

In order to tackle problem (2.10) in a setup that does not provide knowledge on the dynamics of the plant P to be controlled, we design a data-driven approach

that, combined with first order stochastic optimization techniques, carries out the optimization procedure. As introduced in Section 1.3, first order stochastic optimization methods are iterative optimization methods that, once applied to a stochastic problem, update at every iteration the current parameters value by taking a step of length α on a specific direction v, i.e., performing $H_t = H_{t-1} - \alpha\, v$. To individuate the optimal direction v, such methods require the computation of a set of gradients of the goal function. Particularly, applying such methods to problem (2.10) means, at iteration t, to compute the gradient $\nabla_H J_L(\, H_{t-1},\, w_i^t\,)$ for each element w_i^t in a mini-batch $\{w_i^t\}_{i=1}^{N_b}$ obtained by sampling from W_P containing the random variables defined in (2.10), i.e.,

$$W_P = (\, S_P,\, \{P_\ell\}_{\ell=0}^{L}, \{D_\ell\}_{\ell=0}^{L-1}\,). \tag{2.11}$$

However, the shortened-horizon cost $J_L(\cdot,\, w)$ depends on (2.3), and so does its gradient. Since in our setup the dynamics (2.3) are unknown, we design a data-driven approximation procedure, to obtain at every step t a set of approximated gradients $\{\nabla_H \hat{J}_L(\, H_{t-1},\, w_i^t\,)\}_{i=1}^{N_b}$, to be used in place of $\nabla_H J_L(\, H_{t-1},\, w_i^t\,)$ throughout the optimization. In order to make up for the lack of knowledge on h, we employ local linear models, approximating the behavior of s_t only in specific neighborhoods of the state-space. In the following, the gradient approximation at iteration t is described, after presenting the way data are used to substitute knowledge of the dynamics of P and the sampling procedure employed to obtain the mini-batch $\{w_i^t\}_{i=1}^{N_b}$.

2.2.1 Data stream management and local linear models

To approximate the gradients, we rely on the information collected from plant P. Such information is organized in a couple (X, Θ), composed by a dataset X called *states history* of P, and a set Θ. The states history X contains a sequence of states, obtained from input and output data collected from the plant P. We assume to be capable of collecting a sequence

$$\{\ (u_0,\, y_1),\ (u_1,\, y_2),\ (u_2,\, y_3),\ (u_3,\, y_4),\ \dots\ \}$$

of couples (u_i, y_{i+1}) representing interactions with the stricty causal plant P such that u_i is given as input to P and y_{i+1} is the measured (and possibly noisy)

one step ahead output associated to it. Then, we can use them to build the vectors described by (2.2), composed by n_o output and n_i input. Such states get collected in X, i.e.,

$$X = \left\{ x_0 = \begin{bmatrix} y_{n_o} \\ \vdots \\ y_1 \\ u_{n_o-1} \\ \vdots \\ u_{n_o-n_i} \end{bmatrix}, \; x_1 = \begin{bmatrix} y_{n_o+1} \\ \vdots \\ y_2 \\ u_{n_o} \\ \vdots \\ u_{n_o-n_i+1} \end{bmatrix}, \; x_2 = \begin{bmatrix} y_{n_o+2} \\ \vdots \\ y_3 \\ u_{n_o+1} \\ \vdots \\ u_{n_o-n_i+2} \end{bmatrix}, \; \dots \right\}.$$

We set $n_o - n_i \geq 0$ to ensure that the earliest input $u_{n_o-n_i}$ employed belongs to the considered stream. The set Θ instead is a collection of matrices Θ_i, one per each state x_i in X. The matrix Θ_i is a local linear model representing an approximation of the local behavior of plant P in a neighborhood of x_i. The idea is to collect simple local linear models and use them to approximate the required gradients, in order to avoid identifying first a full model of (2.3) from data. It is worthy to notice that none of the Θ_i contained in Θ can be considered as a global model of system (2.3): their function is to be linear approximators of the possibly nonlinear plant P, when restricted to the closely surrounding area of the corresponding states x_i contained in X. For this reason their validity is in general merely local. In the specific case in which the underlying plant is linear and n_i and n_o are chosen to match the order of the system, however, the local models will tend to the nominal model of the plant.

To collect said local linear models, the process is modeled as a linear time-varying system with noise such that

$$\begin{cases} \Theta_i = \Theta_{i-1} + \xi_i \\ y_{i+1} = \Theta_i \begin{bmatrix} x_i \\ u_i \end{bmatrix} + d_i \\ \Theta_0 = \overline{\Theta} \end{cases} \tag{2.12}$$

where ξ_i is a Gaussian white noise with covariance Q_k, d_i is a Gaussian white noise with covariance matrix R_k.

Starting from an initial guess $\overline{\Theta} \in \mathbb{R}^{n_y \times (n_x+n_u)}$, each triplet (u_i, x_i, y_{i+1}) collected from P is used to perform a Kalman filtering (KF) to update Θ_i. Then

x_i and the associated local model Θ_i are stored in X and Θ, respectively. This procedure can be performed using a finite stream of data, previously collected from P, or it can be performed step by step online, while the input u_i are fed to the plant and the outputs are measured. Depending on this, the information at disposal for the controller optimization at iteration t changes, and so does the couple (X, Θ), as furtherly specified in Section 2.3.

2.2.2 Data-driven stochastic optimization

In Section 2.2.1 we saw how the information obtained by interacting with the "black box" plant P is treated and stored. The final part of this section, instead, describes the iterative procedure to apply the stochastic gradient based methods to Problem (2.10) in the presented scenario, by computing the approximation of the necessary mini-batch gradients. In the context of Problem (2.10) all the methods introduced in Section 1.3 iteratively improve the optimization variable H_t, by applying at every step t an update formulated as

$$H_t = H_{t-1} - \alpha_t \; D_t(\mathcal{G}_{J_L}(H_{t-1})), \tag{2.13}$$

where

$$\mathcal{G}_{J_L}(H_{t-1}) = \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla_H J_L(H_{t-1}, \; w_i^t).$$

To this end, at itration t of the optimization procedure, the following steps are performed:

- sampling of a mini-batch $\{w_i^t\}_{i=1}^{N_b}$ from the random vector W_P (2.11);
- computation of the data driven approximation $\hat{\mathcal{G}}_{J_L}(H_{t-1})$ of the mini-batch gradient $\mathcal{G}_{J_L}(H_{t-1})$;
- update of the controller parameters H_t according to the first order stochastic optimization method of choice (see Section 1.3), using the learning rate α_t and the approximated optimization direction $D_t(\hat{\mathcal{G}}_{J_L}(H_{t-1}))$ associated with the method.

The last step corresponds to the computation of Equation (2.13), while the first and second steps are expained in the following subsections.

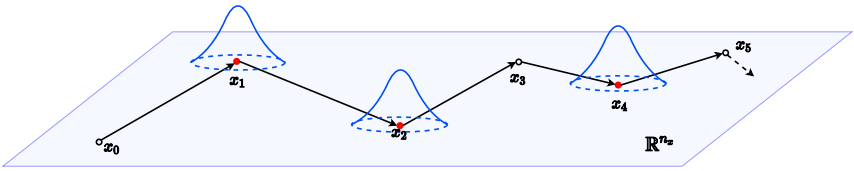

Figure 7: Sampling of initial states from the state history of plant P.

2.2.2.1 Sampling procedure

As previously introduced, every step t of iterative optimization requires a mini-batch $\{w_i^t\}_{i=1}^{N_b}$ to be sampled from the random vector W_P (2.11). According to the structure of W_P, a generical element w of the mini-batch is composed by a reachable state, a sequence of exogenous signals, and one of disturbances, such that $w = (s_0^i,\ \{p_\ell^j\}_{\ell=0}^{L},\ \{d_\ell^k\}_{\ell=0}^{L-1})$.

As per the inital definitions of this chapter, the state s_0^i is composed by a vector x_0^i representing a reachable state of the plant P, and possibly by additional Markovian states that one wants to include in the control policy, denoted by vector z_0^i. The probabilistic distribution $\mathcal{P}_x$ of the reachable states x_0^i over the whole state-space set is unknonw and expensive to estimate. To avoid this time-consuming and resource-consuming task, we rely on the following intuition: if the data in X are collected performing sufficiently explorative tasks, sampling from X would permit an accurate approximation of the unknonwn probability distribution $\mathcal{P}_x$ of the reachable states of P. To obtain the values s_0^i, hence, we sample a set of N_0 states from the history X and add a small random noise v_n from a given set V to explore the neighborhood of the trajectory collected in X. A simple representation of this procedure is proposed in Figure 7, where the state history X contains just five states $x_0, \dots, x_5$. The states (x_1, x_2, x_4), indicated with a red dot, represent an example of possible random sampling of $N_0 = 3$ states. The bel curve on top of each sampled state represents the small Gaussian white noise v_n that is applied to said states to explore the close neighborhood of the trajectory in X. Combining the sampled perturbed states with N_z realizations z_0^m of the chosen additional states, sampled from a probability

distribution $\mathcal{P}_z$, we form a set of $N_s = N_0\, N_z$ initial states

$$s_0^i = s_0^{n,m} = \begin{bmatrix} x_0^n + v_n \\ z_0^m \end{bmatrix}, \tag{2.14}$$

for $n = 1, \ldots, N_0$ and $m = 1, \ldots, N_z$.

A number N_p of exogenous signals sequences is then sampled: each sequence is composed by $L + 1$ values p_ℓ^j, randomly generated according to a probability distribution $\mathcal{P}_p$. Analogously, N_d disturbance trajectories of length L are built by randomly generating instances d_ℓ^k using a chosen probabilty distribution $\mathcal{P}_d$. The sampled states, exogenous signals, and disturbances are combined to form $N_b = N_s\, N_p\, N_d$ elements of the form

$$w = w_{i,j,k} = (\ s_0^i,\ \{p_\ell^j\}_{\ell=0}^{L},\ \{d_\ell^k\}_{\ell=0}^{L-1}\ ),$$

for $i = 1, \ldots, N_s$, $j = 1, \ldots, N_p$, $k = 1, \ldots, N_d$.

The choice of the probability distributions $\mathcal{P}_z$, $\mathcal{P}_p$ and $\mathcal{P}_d$ employed in the previously described sampling procedure is treated as a tuning decision. The design of such distributions might depend on the empyrical knowledge that one has on the plant and on the considered variables. In case no additional knowledge is available, it is anyways always possible to sample additional states, exogenous signals and disturbances uniformly over the set of the possible values they can assume by definition. This practice can still lead to controllers with good performances as demonstrated in all the numerical examples throughout this dissertation. In particular, each element of the sequence $\{p_\ell^j\}_{\ell=0}^{L}$ of exogenous signals can be sampled uniformly from a user defined interval $[p_{\min},\, p_{\max}]$, while the disturbances can be considered as Gaussian white noise with variance σ_d^2 estimated from the known precision charachterizing the sensors employed for the signal measurement. It would be immediate, however, to incorporate any information possibly available on the probability distribution in the chosen formulation, for example in case one can estimate them empirically. That can potentially reduce the numbers N_s, N_p, N_d of considered samples, and so the resulting computation time to construct and solve the stochastic optimization problem (cf. (Bemporad, Gabbriellini, et al., 2010)).

2.2.2.2 Data-driven gradients approximation

After sampling the mini-batch $\{w_i^t\}_{i=1}^{N_b}$, the values $\{\nabla_H J_L(\, H_{t-1},\, w_i^t\,)\}_{i=1}^{N_b}$ of the associated gradients of $J_L(\,\cdot\,,\, w_i^t\,)$ evaluated in the last updated set of parameters H_{t-1} are approximated. As already anticipated, we intend to use the local models collected in Θ to tackle the gradient approximation and overcome the fact that $\nabla_H J_L(H_{t-1}, w_i^t)$ depends on the unknown dynamics (2.3). We observe that $J_L(\,\cdot\,, w_i^t)$ is a function of the parameters H that evaluates how π_H affects the cost of controlling P in the conditions dictated by w_i^t for L steps. The sample w_i^t contains, together with exogenous signals and disturbances, a reachable state s_0 considered as initial state for the trajectory considered by J_L. To approximate the gradient of $J_L(\,\cdot\,, w_i^t)$ we use then a local linear model fitted around s_0, approximating the dynamics (2.3) in a neighborhood of s_0.

Given a sample $w = (s_0^i = [x_0^n + v_n, z_0^m]',\ \{p_\ell^j\}_{\ell=0}^{L},\ \{d_\ell^k\}_{\ell=0}^{L-1})$, the local linear model Θ_n associated with the initial state x_0^n in s_0^i is retrieved from the set of local linear models Θ. Such model is then used to simulate the dynamics of P in the computation of the gradient $\nabla_H \hat{J}_L(H_{t-1}, w)$. One can obviously rely on the analytic computation of the gradient (possibly benefitting from automatic differentiation tools to easen the procedure). Alternatively, it is possible to employ finite differences, substituting the gradient with a vector having as $i-$th element

$$\frac{\hat{J}_L(H_{t-1} + \epsilon\, e_i,\ w) - \hat{J}_L(H_{t-1},\ w).}{\epsilon} \tag{2.15}$$

In both cases, as anticipated in the beginning of Section 2.2, we consider

$$\hat{J}_L(H_{t-1},\, w) = \sum_{\ell=0}^{L-1} \rho(\hat{s}_\ell,\, p_\ell^j,\, \pi_{H_{t-1}}(\hat{s}_\ell,\, p_\ell^j)) + \rho_L(\hat{s}_{_L},\, p_{_L}^j), \tag{2.16}$$

and the sequence of approximated states $\{\hat{s}_\ell\}_{\ell=0}^{L}$ is obtained by applying $\hat{u}_\ell = \pi_{H_{t-1}}(\hat{s}_\ell,\, p_\ell^j)$ to Θ_n, collecting

$$\hat{y}_{\ell+1} = \Theta_n \begin{bmatrix} \hat{x}_\ell \\ \hat{u}_\ell \end{bmatrix} + d_\ell^k$$

and using it to build $\hat{s}_{\ell+1}$, given that $\hat{s}_0 = s_0^i$.

The vector $\hat{s}_{\ell+1}$ is composed by the plant state $\hat{x}_{\ell+1}$ and the additional Markovian state $\hat{z}_{\ell+1}$. Given equation (2.2), it is easy to build $\hat{x}_{\ell+1}$ from $\hat{x}_\ell$ by adding $\hat{u}_\ell$ and $\hat{y}_{\ell+1}$ as most recent input and output and removing the oldest ones.

For the approximation of the dynamics of the additional states z, instead, we have different scenarios:

- if $z_{\ell+1}$ is an exogenous signal that can be derived from x_ℓ, $z_\ell$and p_ℓ^j through some known differentiable dynamics

$$z_{\ell+1} = f_z(x_\ell,\, z_\ell,\, p_\ell^j), \tag{2.17}$$

  then we can just use the predicted value $\hat{y}_\ell$, obtained by using Θ_n, to derive $\hat{x}_\ell$ (2.2) and hence the approximated dynamics $\hat{z}_{\ell+1} = f_z(\hat{x}_\ell, \hat{z}_\ell,\, p_\ell^j)$, given $\hat{z}_0$ specified in the sample w currently considered;

- if instead z_ℓ has non differentiable dynamics, or if z_ℓ measurable, but having completely unknown dynamics, we can enrich the ARX state (2.2) by adding the measurements of the additional states, i.e.,

$$x_t = [\, y_t',\, z_t', \dots\, y_{t-n_o+1}',\, z_{t-n_o+1}',\, u_{t-1}'\, \dots\, u_{t-n_i}'\,]'. \tag{2.18}$$

  In this case we will consider $s_t = x_t$ and the local linear model will be updated according to

$$\begin{cases} \Theta_i = \Theta_{i-1} + \xi_i, \\ \begin{bmatrix} y_{i+1} \\ z_{i+1} \end{bmatrix} = \Theta_i \begin{bmatrix} x_i \\ u_i \end{bmatrix} + d_i. \end{cases} \tag{2.19}$$

A simple representation of the data-driven gradient approximation procedure is given in Figure 8, where the trajectory cost approximation is represented in a simplified setup characterized by just to two samples w_1 and w_2 of the mini-batch, sharing the same initial state $s_0^i = [x_0^n + v_n,\, z_0^m]'$ and different exogenous and disturbances signals, namely $w_1 = (s_0^i,\, \{p_\ell^{j_1}\}_{\ell=0}^{L}\, \{d_\ell^{k_1}\}_{\ell=0}^{L-1})$ and $w_2 = (s_0^i,\, \{p_\ell^{j_2}\}_{\ell=0}^{L}\, \{d_\ell^{k_2}\}_{\ell=0}^{L-1})$. The red dots represent the simulated states $\hat{s}_\ell$, obtained using the local linear model Θ_n, as indicated by the underlying formulas. The cost function $\hat{J}_L(H_{t-1},\, w_1)$ is obtained by summing the stage costs $\rho(\hat{s}_\ell, \hat{u}_\ell, p_\ell^{j_1})$ along the trajectory generated by Θ_n (or by Θ_n and f_z) while receiving $\hat{u}_\ell = \pi_{H_{t-1}}(\hat{s}_\ell,\, p_\ell^{j_1})$ in input. Analogously for $\hat{J}_L(H_{t-1},\, w_2)$.

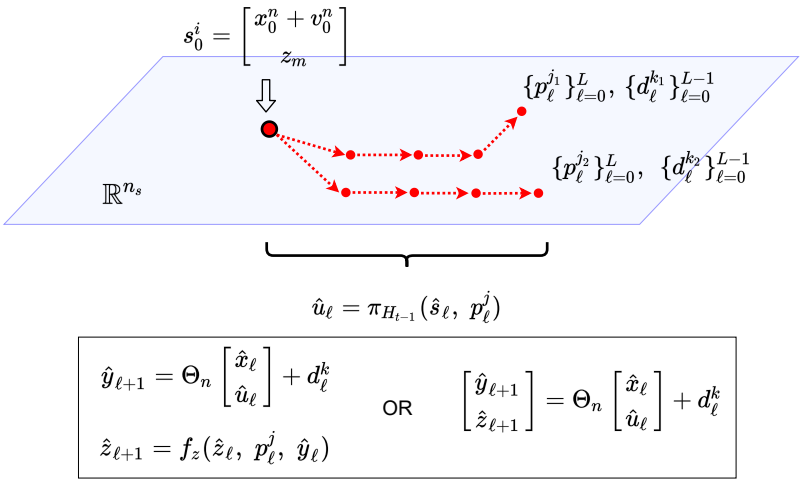

Figure 8: Use of the local linear model to compute the trajectories associated with the cost $\hat{J}_L(H_{t-1},\ w)$ approximating $J_L(H_{t-1},\ w)$.

2.3 Learning smooth control policies via OPS

The previous part of this chapter introduced the general framework characterizing the Optimal Policy Search approach for the synthesis of feedback controllers of black-box plants. In this section we use such approach to build an algorithm for the synthesis of smooth feedback controllers. To represent smooth controllers, as specified before, we consider policy parameterizations such that for each $(s_t,\ p_t) \in \mathbb{R}^{n_s+n_p}$ the policy $\pi_H(s_t,\ p_t) \in C^1(\mathbb{R}^{n_h})$. For the application of the Optimal Policy Search algorithm, two settings are considered: *offline* setting and *online* setting. In the offline setting, the policy π_H is synthesized from open-loop data that were previously collected from the plant. In the online setting, instead, new data are collected from the plant during the iterative policy synthesis, while the plant is also being controlled, and employed in further stochastic optimization iterations. Sections 2.3.1 and 2.3.2 contain the details on the algorithm to be applied offline and online, respectively.

2.3.1 Offline setting

As previously mentioned, in the offline setting the synthesis of smooth controllers is performed with no direct interaction with the plant, but based on a dataset composed by input-output couples (u_t, y_{t+1}) collected in open-loop from the plant at a previous stage. Hence, disposing of a finite sequence of N couples

$$\{\ (u_0,\, y_1),\ (u_1,\, y_2),\ \dots\ (u_{N-1},\, y_N)\}$$

we proceed, analogously with what introduced in Section 2.2.1, by building a finite states history of cardinality $N - (n_0 - 1)$,

$$X = \Bigg\{\ x_0 = \begin{bmatrix} y_{n_o} \\ \vdots \\ y_1 \\ u_{n_o-1} \\ \vdots \\ u_{n_o-n_i} \end{bmatrix},\ x_1 = \begin{bmatrix} y_{n_o+1} \\ \vdots \\ y_2 \\ u_{n_o} \\ \vdots \\ u_{n_o-n_i+1} \end{bmatrix},\ \dots\ x_{N-n_0} = \begin{bmatrix} y_N \\ \vdots \\ y_{N-n_0} \\ u_{N-1} \\ \vdots \\ u_{N-n_i} \end{bmatrix}\ \Bigg\},$$

with $N >> n_0 \geq n_i$.

Based on the information carried by the states history X, three steps of policy optimization procedure are applied in sequence for a number N_{learn} of iterations, as represented in Algorithm 1.

At every iteration t, firstly, we collect a mini-batch $\{w_i^t\}_{i=0}^{N_b}$ by sampling N_b elements according to the sampling procedure described in Section 2.2.2.1. Then, for each element in the mini-batch, the data-driven gradient approximation presented in Section 2.2.2.2 is employed to compute the gradients $\nabla_H \hat{J}_L(H_{t-1}, w_i^t)$, approximating the direction of descent for the cost function of Problem (2.10), i.e.,

$$\hat{\mathcal{G}}_{J_L}(H_{t-1}) = \mathcal{G}_{\hat{J}_L}(H_{t-1}) = \frac{1}{N_b} \sum_{i=1}^{N_b} \nabla_H \hat{J}_L(H_{t-1}, w_i). \tag{2.20}$$

As a third and last step, at every iteration t the vector $\hat{\mathcal{G}}_{J_L}(H_{t-1})$ is employed to update the parameters H_t, according to the rule (2.13).

Algorithm 1 OPS with data-driven gradients - Offline

Input: Initial guess H_{-1}, number N_{learn} of learning steps, state history $X = \{x_0, .., x_{N-n_o}\}$ of P and associated local linear models $\Theta = \{\Theta_0, .., \Theta_{N-n_o}\}$.
Output: Policy parameters H_{OPS}.

for $t = 0, \ldots, N_{\text{learn}} - 1$ **do**
  for $h = 1, 2, ..., N_b$ **do**
    Sample $w_h^t = (\, s_0^i,\ \{p_\ell^j\}_{\ell=0}^{L},\ \{d_\ell^k\}_{\ell=0}^{L-1}\,)$;
    Retrieve model coefficients Θ_i associated with s_0^i;
    Compute $\nabla_H \hat{J}_L(H_{t-1}, w_h^t)$;
  end for
  Mini-batch gradient:
    $\hat{\mathcal{G}}_{J_L}(H_{t-1}) \leftarrow \frac{1}{N_b} \sum_{h=1}^{N_b} \nabla_H \hat{J}_L(H_{t-1}, w_h^t)$;
  Policy update:
    $H_t \leftarrow H_{t-1} - \alpha_t D_t(\hat{\mathcal{G}}_{J_L}(H_{t-1}))$;
end for
$H_{\text{OPS}} \leftarrow H_{N_{\text{learn}}-1}$;
end.

2.3.2 Online setting

In the online setting, the Optimal Policy Search procedure has to take care of both the plant control and the policy optimization, without having at disposal a set of previously collected input-output couples from the plant. The control of the plant requires to provide, at every time instant t, an input u_t to the plant, in order to perform an on-going task.

As summarized in Algorithm 2, the algorithm in the online setup differs from the offline one for the data collection and storage online steps. At every iteration t, the optimization steps are performed after a sequence of operations in which the current controller $\pi_{H_{t-1}}$ interacts with the plant, starting from an initial state $x_0 = [y_0' \ldots y_{-n_o+1}' u_{-1}' \ldots u_{-n_i}']'$ composed by n_i input and n_o output. The set of exogenous signals p_t is first measured. Then, the state x_t described in equation (2.2) is build and saved in the state history.

Following from this, the states history of P evolves in time: at an instant t it corresponds to the set of all the states visited by the plant from the beginning of

the experience up to the current time, i.e.,

$$X(t) = \left\{ x_0 = \begin{bmatrix} y_0 \\ \vdots \\ y_{-n_0+1} \\ u_{-1} \\ \vdots \\ u_{-n_i} \end{bmatrix}, \; x_1 = \begin{bmatrix} y_1 \\ \vdots \\ y_{-n_0+2} \\ u_0 \\ \vdots \\ u_{-n_i+1} \end{bmatrix}, \; \ldots, \; x_t = \begin{bmatrix} y_t \\ \vdots \\ y_{t-n_0+1} \\ u_{t-1} \\ \vdots \\ u_{t-n_i} \end{bmatrix} \right\}.$$

Algorithm 2 OPS with data-driven gradients - Online

Input: Initial guess H_{-1}, number N_{learn} of learning steps. Initial state x_0, and local linear model Θ_{-1}, $X(-1) = \emptyset$, $\Theta(-1) = \emptyset$.

Output: Policy parameters H_{OPS}.

for $t = 0, \ldots, N_{\text{learn}} - 1$ **do**
  Measure signal p_t;
  Build x_t as in (2.2) and store it, i.e., $X(t) \leftarrow X(t-1) \cup \{x_t\}$;
  Compute/measure the additional states z_t; (see Eq. (2.17)-(2.18))
  Build s_t from x_t and z_t;
  Apply $u_t = \pi_{H_{t-1}}(s_t, p_t)$ to the plant and collect y_{t+1};
  Update Θ_t based on $(x_t,\, u_t,\, y_{t+1})$;
  $\Theta(t) \leftarrow \Theta(t-1) \cup \{\Theta_t\}$;
  for $h = 1, 2, ..., N_b$ **do**
    Sample $w_h^t = (\, s_0^i,\, \{p_\ell^j\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1}\,)$;
    Retrieve model coefficients Θ_i associated with s_0^i;
    Compute $\nabla_H \hat{J}_L(H_{t-1}, w_h^t)$;
  end for
  Mini-batch gradient:
    $\hat{\mathcal{G}}_{J_L}(H_{t-1}) \leftarrow \frac{1}{N_b} \sum_{h=1}^{N_b} \nabla_H \hat{J}_L(H_{t-1}, w_h^t)$;
  Policy update:
    $H_t \leftarrow H_{t-1} - \alpha_t D_t(\hat{\mathcal{G}}_{J_L}(H_{t-1}))$;
end for
$H_{\text{OPS}} \leftarrow H_{N_{\text{learn}}-1}$;
end.

After storing x_t in $X(t)$, the action $u_t = \pi_{H_{t-1}}(s_t, p_t)$ generated by the parameterized policy with the current set of parameters is computed and applied to plant P. The associated output y_{t+1} is collected and used, together with x_t and u_t to obtain an updated value of the local linear model Θ_t, as described in Section 2.2.1. The matrix Θ_t is then stored in the local linear models set $\Theta(t)$, that is time-varying as well as $X(t)$. Before performing the optimization steps of sampling, gradient approximation and policy update (that are analogous to the steps described for the offline setup) the values s_t, u_t and y_{t+1} are used to update s_{t+1}.

2.3.2.1 Online synthesis of linear feedback controllers

In this subsection we instantiate the method described in Section 2.3.2 for the online synthesis of feedback controllers expressing a linear relation between parameters and feedback states. Specifically, we consider two cases: linear parameterization of the input u_t and linear parameterization of the input increment $\Delta u_t = u_t - u_{t-1}$. In the first case, the selected parameterization is

$$\pi_K(s_t, p_t) = K \begin{bmatrix} s_t \\ p_t \end{bmatrix} = K^s s_t + K^p p_t. \tag{2.21}$$

while in the second, given that $u_t = u_{t-1} + \Delta u_t$, we have

$$\pi_K(s_t, p_t) = u_{t-1} + K \begin{bmatrix} s_t \\ p_t \end{bmatrix} = u_{t-1} + K^s s_t + K^p p_t. \tag{2.22}$$

Considering that in the online setup the OPS algorithm is applied directly on plant P during the learning phase, it is important to avoid destabilizing the system. In general, to establish with certainty if a controller will destabilize a plant, it is necessary to rely on information about the dynamics of the plant itself. In case of linear feedback controller synthesis, precautions can be taken while remaining faithful to the "black-box" hypothesis over the plant dynamics. In particular, a heuristic can be applied at every iteration, to gain local stability. Such heuristic, at every instant t, builds an approximation of the non-minimal state space realization x_t dynamics, from the last updated local linear model Θ_t.

The approximated dynamics of x_t are extended to describe the local evolution of the whole state s_t. Hence, at every iteration t, a system of the form

$$s_{j+1} \quad = \quad A_t s_j + B_t u_j + E_t p_j + D_t d_j. \tag{2.23}$$

is considered. The procedure to build the matrices (A_t, B_t, E_t, D_t) is described in detail in Appendix A. The heuristic employs such matrices in the following way: after one step of policy optimization, the updated policy K_t is projected onto the space of policies stabilizing (A_t, B_t), so into the vector that is closest to K_t^s and is also asymptotically stabilizing (A_t, B_t). This can be formulated as the solution of the optimization problem

$$\begin{aligned} \min_{K} \quad & \|K - K_t^s\|_2^2 \\ \text{s.t.} \quad & \mid \text{Spec}(A_t + B_t K) \mid \subset [0,\, 1) \end{aligned} \tag{2.24}$$

where $\text{Spec}(A_t + B_t K)$ denotes the set of eigenvalues of the closed-loop dynamics matrix $A_t + B_t K$. Solving problem (2.24) is equivalent to solve the following semidefinite program

$$\begin{aligned} \min_{Y,Q,W} \quad & \|Y - K_t^s Q\|_2^2 \\ \text{s.t.} \quad & \begin{bmatrix} Q & Q & M' \\ Q & W & 0 \\ M & 0 & Q \end{bmatrix} \succeq 0, \\ & \text{with } M = A_t Q + B_t Y, \\ & W \succ 0, \quad Q \succ 0, \end{aligned} \tag{2.25}$$

as proved for instance in Bernardini and Bemporad, 2012. The feedback gain $K_{s,t}^{\circledast} = Y^\star (Q^\star)^{-1}$ obtained from the solution $Y^\star, Q^\star, W^\star$ of (2.25) is then computed. The input $u_t = K_{s,t}^{\circledast} s_t + K_t^p p_t$ is finally applied to the plant.

Up to this point this section described how to apply the local stabilization heuristic combined with the linear policy (2.21). Analogous steps can be performed using (2.22): in this case we will consider the approximated dynamics

$$s_{j+1} = A_t^{\Delta u} s_j + B_t^{\Delta u} \Delta u_j + E_t p_j + D_t d_j,$$

built as shown in Appendix A. Considering $\Delta u_t = K_t^s s_t + K_t^p p_t$, at iteration t the policy K_t is projected onto the space of policies stabilizing $(A_t^{\Delta u}, B_t^{\Delta u})$, by solving a semidefinite problem analogous to (2.25), but built using the matrices $(A_t^{\Delta u}, B_t^{\Delta u})$.

Chapter 3

Output tracking via Optimal Policy Search

In Chapter 2 both the general problem formulation and the online and offline Optimal Policy Search algorithms for the synthesis of smooth policies were presented. In this chapter, instead, we focus on the application of such methods to the output-tracking problem. The beginning of the chapter is dedicated to formulate the tracking problem as an Optimal Policy Search problem. The rest of the chapter provides examples of the performance of the proposed approach in handling such problem for both linear and nonlinear plants, synthesizing linear and nonlinear feedback controllers.

3.1 OPS for output tracking

To formulate a specific problem in the Optimal Policy Search framework, care has to be taken in the choice of the stage cost ρ, the terminal cost ρ_L, the state s_t, and the exogenous signals p_t. The design of such characteristics has to be based on the task to be performed and on the available information.

Hence, to learn a policy that makes the output y_t of the plant P track a

reference signal $r_t \in [r_{\min}, r_{\max}]$, we consider

$$s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix}, \quad p_t = r_t, \tag{3.1}$$

where the state s_t is composed by x_t as in (2.2) and by the tracking error integral formula q_t as additional Markovian state with known dynamics, i.e. $z_t = q_t$ such that $q_{t+1} = q_t + (y_t - r_t)$.

To achieve the output-tracking task we consider the following stage cost and terminal cost

$$\rho(s_t,\, r_t,\, u_t) = \|y_t - r_t\|^2_{Q_y} + \|\Delta u_t\|^2_R + \|q_t\|^2_{Q_q}, \tag{3.2a}$$

$$\rho_L(s_L,\, r_L) = \|y_L - r_L\|^2_{Q_y^L} + \|q_L\|^2_{Q_q^L}, \tag{3.2b}$$

where y_t is a component of x_t, and therefore of s_t, and the input increment $\Delta u_t = u_t - u_{t-1}$ is obtained from u_t and s_t (we assume the number of input n_i contained in x_t to be greater or equal to 1, so u_{t-1} is always included in x_t, and hence in s_t). In the stage cost, $R = R' \succ 0$ penalizes the control effort, while the matrices $Q_y = Q_y' \succeq 0$ and $Q_q = Q_q' \succeq 0$ weight the tracking error and the integral action respectively. Analogous role is covered by Q_y^L and Q_q^L in the terminal cost. A penalty on u_t of the form $\|u_t - u_t^r\|^2_{Q_u}$ might be easily introduced as well in (3.2); however, for proper reference tracking the input set-point u_t^r should be consistent with r_t, therefore requiring a preview of the reference or some knowledge on P, that we do not want to assume available.

We assume that within the L steps of the shortened horizon the reference signal remains constant, i.e., $r_\ell = r$, where r is generated by an opportune random variable R having as set of possible outcomes the interval $[r_{\min}, r_{\max}]$. Considering that we are optimizing the expected trajectory cost with respect to all the possible values assumed by R, starting from every possible initial value $s_0 \sim S_P$, the solution of the optimization problem will still be a parameterized contoller capable of tracking all the (possibly non constant) signals assuming values in $[r_{\min}, r_{\max}]$.

According to the mentioned choices, the OPS Problem (2.10) can be instan-

tiaded for the synthesis of output-tracking controllers as

$$
\begin{aligned}
H^* = \arg \min_{H} \quad & \mathbb{E}\Big[\sum_{\ell=0}^{L-1}\Big(\|y_\ell - r\|^2_{Q_y} + \|\Delta u_\ell\|^2_R + \|q_\ell\|^2_{Q_q}\Big) + \\
& + \|y_L - r\|^2_{Q_y^L} + \|q_L\|^2_{Q_q^L}\Big], \qquad (3.3)\\
\text{such that} \quad & u_\ell = \pi_H(s_\ell,\, r), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_{\ell+1} = h(s_\ell,\, r,\, u_\ell,\, d_\ell), \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& d_\ell \sim D_\ell, \quad \ell = 0, 1, \ldots, L\text{ - }1,\\
& s_0 \sim S_P, \quad r \sim R.
\end{aligned}
$$

3.1.1 Online exploration for the learning of tracking policies

In the context of synthesizing an output-tracking controller online using Optimal Policy Search algorithm, the policy π_{H_t} is optimized while the plant is stirred to track an a-priori unknonw reference $\{r_i\}_{i=0}^{N_{\text{learn}}}$. Designing the reference signal to be used as online task during the learning phase corresponds to choosing an exploratory strategy for the collection of samples in the states history.

As detailed in Chapter 2, Section 2.3.2, the online optimization of H_t requires the approximation of the gradients of the cost over a mini-batch of trajectories. Such trajectories are generated applying the current policy π_{H_t} starting from initial states $s_0 = [\, x_0\,,\, z_0\,]'$. At instant t, x_0 is sampled from the current states history $X(t)$.

While learning control policies for output-tracking online the states history $X(t)$ at instant t is populated by the states x_i for $i = 0, 1, \ldots, i$ visited by the plant while following the first t instances of the online tracking task. Hence, to learn a policy capable of tracking in a certain set of conditions it is important to choose an online tracking task that conducts the plant to visit the states that characterize the realization of such conditions. On the other hand, such choice has to be made taking into account risk aversion criteria as well, considering that references that are hard to track, although providing plenty of exploration, could be dangerous to track online, expecially at an early stage, when the performance of π_{H_t} is still far from optimal.

3.1.2 Online output-tracking: assisted control

As described in the previous paragraphs, the online implementation of the Optimal Policy Search algorithm for output tracking enforces the exploration of the state-space through the reference design. In order for the space exploration not to lead to unsafe scenarios, particularly in the initial phase of the learning, our approach can benefit from combining the use of a behavioral policy π_b, together with π_{H_t}, when it comes to steer the plant output while learning. In general, delegating the control of the plant to a behavioral policy can be safer, if π_b is designed taking into account some safety guarantees or based on previous knowledge, and not for pure exploration purposes. It is still important though to apply the policy π_{H_t} to collect data while learning, in order to populate the state and model history with states and models often encountered using the current controller parameters, in order to optimize the policy H_t using relevant optimization directions.

The contribution of the behavioral policy at every iteration t can be expressed through a procedure of input selection, summarized in Algorithm 3, that substitutes the computation of the current input $u_t = \pi_{H_t}(s_t, p_t)$ in the classic online learning algorithm (see Step 6 of Algorithm 2). The mentioned input selection procedure is composed by three phases: *off-policy* learning, online learning with assisted control, and *on-policy* learning. The behavioral policy π_b is used in the off-policy learning phase, for $t \in [\,0, T_1 - 1\,]$, to generate the input to the plant, to track the reference r_t, and to collect input/output data for the ongoing optimization of H_t. After T_1 steps, online learning with assisted control of the plant is performed, for $t \in [\,T_1, T_2\,]$: every $M \geq 1$ steps, the input $u_t = \pi_{H_t}(s_t, r_t)$ is generated from the current policy π_{H_t}. Then, the result of applying this u_t to the plant is predicted using the last updated local model Θ_t. If the predicted output is sufficiently close to the desired set-point value r_t, we assign the control of the plant for the next M iterations to π_{H_t} while it is being optimized. Otherwise the task is performed by the behavioral policy π_b for M steps. After M steps the test is repeated, in order to assign the control of the plant to π_b or π_{H_t} for other M steps, and so on, till t is smaller than T_2. After T_2 steps, learning is performed completely in *on-policy* setting, with the policy π_{H_t} being employed at every successive iteration.

Algorithm 3 Online learning of output-tracking controller: input selection with assisted control (parameters M, T_1, T_2, ϵ, behavioral policy π_b)

Input: Policy π_{H_t}, state s_t, reference r_t, local model Θ_t, on-policy$(t-1)$.
Output: Input u_t, on-policy(t).

on-policy(t) = False;
if $t > T_2$ **then**
    on-policy(t) = True;
else
  if $t \geq T_1$ **then**
    if rem$(t, M) = 0$ **then**
      $u_H \leftarrow \pi_{H_t}(s_t, r_t)$;
      $y_{\text{pred}} = \Theta_t \left[\begin{smallmatrix} x_t \\ u_H \end{smallmatrix} \right]$;
      if $\|y_{\text{pred}} - r_t\|_\infty \leq \epsilon$ **then**
        on-policy(t) = True;
      end if
    else
      on-policy(t) = on-policy$(t-1)$;
    end if
  end if
end if
if on-policy(t) **then**
    $u_t \leftarrow \pi_{H_t}(s_t, r_t)$;
else
    $u_t \leftarrow \pi_b(s_t, r_t)$;
end if

The choice of parameters T_1 and T_2 such that $0 \leq T_1 \leq T_2 \leq N_{\text{learn}}$ is part of the learning phase design. By choosing $T_1 = 0$, $T_1 = T_2$, or $T_2 = N_{\text{learn}}$ it is possible to eliminate respectively the initial off-policy phase, the online learning with assisted control phase, or the final on-policy phase. The behavioral policy π_b is not necessarily modeled following the parameterization π_{H_t}: any controller can be used as behavioral policy.

3.2 Example 1 - LTI system - linear policy parameterization

Let the system generating the data be the (unknown) single-input single-output (SISO) linear time invariant (LTI) system ($n_y = n_u = 1$)

$$\begin{aligned} \overline{x}_{t+1} &= \begin{bmatrix} -0.669 & 0.378 & 0.233 \\ -0.288 & -0.147 & -0.638 \\ -0.337 & -0.589 & 0.043 \end{bmatrix} \overline{x}_t + \begin{bmatrix} -0.295 \\ 0.325 \\ 0.026 \end{bmatrix} u_t, \\ y_t &= \begin{bmatrix} -1.140 & 0.320 & -0.571 \end{bmatrix} \overline{x}_t \end{aligned} \tag{3.4}$$

that in ARX form corresponds to

$$\begin{aligned} y_{t+1} = &- 0.7738\ y_t + 0.1245\ y_{t-1} + 0.3699\ y_{t-2} + \\ &+ 0.4257\ u_t + 0.0159\ u_{t-1} + 0.0282\ u_{t-2} \end{aligned}$$

The state s_t defined, according to (3.1), as

$$s_t = \begin{bmatrix} x_t \\ q_t \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} y_t \\ y_{t-1} \\ y_{t-2} \\ u_{t-1} \\ u_{t-2} \end{bmatrix} \\ q_t \end{bmatrix} \in \mathbb{R}^{n_x + n_y},$$

evolves with respect to the input increment Δu_t according to

$$s_{t+1} = A_s s_t + B_s \Delta u_t + E_s r_t, \tag{3.5}$$

where

$$A_s = \begin{bmatrix} -0.7738 & 0.1245 & 0.3699 & 0.4416 & 0.0282 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, B_s = \begin{bmatrix} 0.4257 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

$$E_s = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}'.$$

We design the cost function in (3.2) by choosing horizon $L = 10$ and weights $Q_y = Q_y^L = 1$, $R = 0.1$, and $Q_q = Q_q^L = 1$ and we apply the Optimal Policy Search algorithm in the offline and online setting, employing a linear parameterization of the input increment, according to (2.22). By choosing $n_o = 3$, $n_i = 2$, our parameterization is

$$\Delta u_t = K_1\, y_t + K_2\, y_{t-1} + K_3\, y_{t-2} + K_4\, u_{t-1} + K_5\, u_{t-2} + K_6\, q_t + K_7\, r_t.$$

Considering the chosen parametrization, our problem coincides with the Linear Quadratic Regulation (LQR) problem applied to (3.5), having optimal solution

$$K_*' = \left[0.1160,\ -0.2196,\ -0.6526,\ -0.8978,\ -0.0498,\ -1.1411,\ 3.3370\right].$$

The aim of this example is to show that, when applied to solve an LQR problem, the Optimal Policy Search method, if tuned with the appropriate system orders converges to the known optimal solution, both in offline and online setting, in noiseless or noisy scenarios. Regarding the online setting, it will also be shown that the method is capable, while learning, of effectively performing otput tracking of a-priori unknonwn reference signals.

3.2.1 Offline setting

We start by analysing the convergence of the method in the offline case. At first, a dataset of $N_{\text{data}} = 30000$ input-output couples is collected in open-loop from (3.4) by giving in input to the system a random sequence of piecewise constant inputs (represented in the lower plot of Figure 9) and measuring the associated output, with an additive Gaussian white noise with variance σ_y^2. The data are employed to build the states history X as described in Section 2.2.1. The associated local linear models are fitted and stored in Θ, by initializing the parameters as $\Theta_0 = \underline{0}_{1\times 6}$ and $P_0 = (1e+5)\cdot I_6$, and recursively updating them by Kalman Filtering, with covariance matrices $Q_k = (1e-5)\cdot I_6$ and R_k. The Optimal Policy Search offline algorithm is applied: the learning procedure is executed for $N_{\text{learn}} = 30000$ iterations, starting from an initial policy K_0. Regarding the sampling procedure, $N_0 = 1$ state, $N_r = 1$ reference, $N_q = 1$ integral action values and N_d disturbance are sampled at each learning iteration. The references and the integral action values composing the initial states for the cost

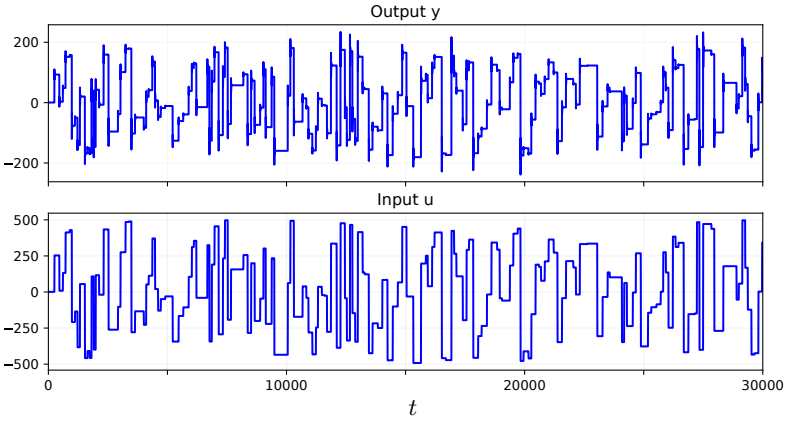

Figure 9: Example 1, offline learning, open-loop noiseless dataset.

simulations are both sampled uniformly in $[-10, 10]$. The disturbances, instead, are sampled according to a Gaussian white random variable, characterized by variance σ_d^2. The small perturbation noise applied to the state sampled from X is chosen as $v_n \sim 0.1 \cdot \mathcal{N}(0, 1)$. To update the controller parameters, AMSGrad is employed, with parameters α, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The following two subsections contain the tuning details that differ between the noiseless case and the noisy one (namely, the values of parameters σ_y^2, N_d, σ_d, R_k and the learnig rate α), together with the results obtained in the two cases.

Noiseless case

In the noiseless case we assume to be capable of measuring the exact output of (3.4), i.e., we set $\sigma_y^2 = 0$. The upper plot of Figure 9 shows the stream of noiseless output associated to the chosen piecewise constant input signal. We apply the offline OPS method, starting from $K_0 = 0.0001 \cdot \underline{1}_{7\times 1}$, employing the aforementioned parameters, together with the ones contained in Table 1. Figure 10 shows the results in terms of convergence to the optimal policy K_*. We can observe that after 2500 iterations the distance between the synthesized parameters K_t and K_* is much smaller than 0.1.

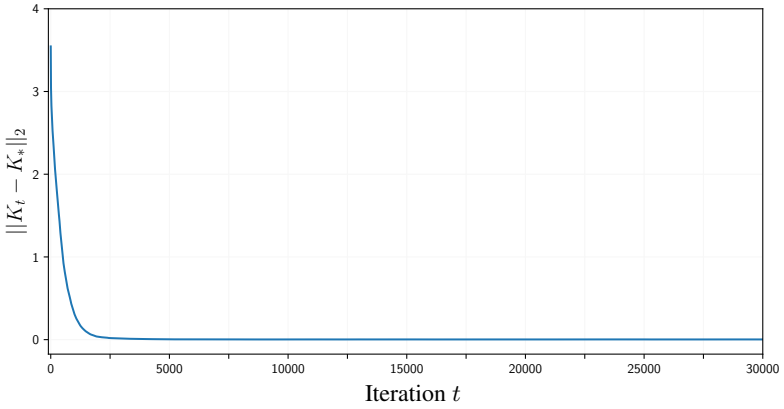

Figure 10: Example 1, offline learning in noiseless conditions, convergence of the linear policy parameters to the optimal value K_*.

N_d	σ_d	R_k	α
1	0	$1e-5$	0.1

Table 1: Example 1, offline learning, noiseless case, parameters

Noisy case

The process of learning from a noisy dataset is then studied, by considering data generated by the same input stream represented in Figure 9 and adding Gaussian white noise to the associated output, i.e., generated sampling from $\mathcal{N}(0, \sigma_y^2)$. The noise on data is introduced to model the precision level of the sensors that, in a real setup, are used to measure the output of the plant. Noise realizations over data are per se not measurable, but it is reasonable to assume the knowledge of the range of precision of the physical sensors mounted over a plant, often provided by the sensors' producer, to establish the expected performances in terms of precision of their product. It is hence possible to use σ_y for the tuning of some of the parameters of the Optimal Policy Search method, specifically the covariance matrix R_k and the random variable used to sample the disturbance trajectories $\{d_\ell^k\}_{\ell=0}^{L-1}$.

N_d	σ_d	R_k	α
50	σ_y	σ_y^2	0.001

Table 2: Example 1, offline learning, noisy case (noise variance σ_y^2), parameters

Following this rationale the method is tested, starting from $K_0 = 0.0001 \cdot \underline{1}_{7\times 1}$ and considering different values $\sigma_y = 0.1, 0.2, 0.3$ for the standard deviation of the noise. The parameters included in the generical offline case description are employed again in this scenario, together with the values shown in Table 2. Figure 11 shows the convergence of the method. As a comparison, the parameters evolution in the noiseless case is also included in Figure 11, indicated as $\sigma_y = 0.0$. From the top plot in Figure 11 we see that in each noisy scenario the policies converge to the optimal policy. The bottom plot focuses on the last 1000 steps of learning, showing that, although the convergence rate is extremely similar for all the scenarios, the noise slightly improves the convergence, probably because it increases the state-space exploration in the data.

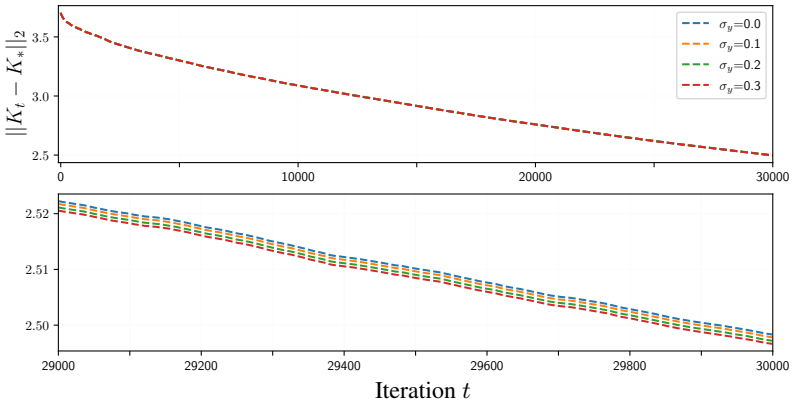

Figure 11: Example 1, offline learning in different noise conditions, convergence of linear policy parameters to the optimal value K_*.

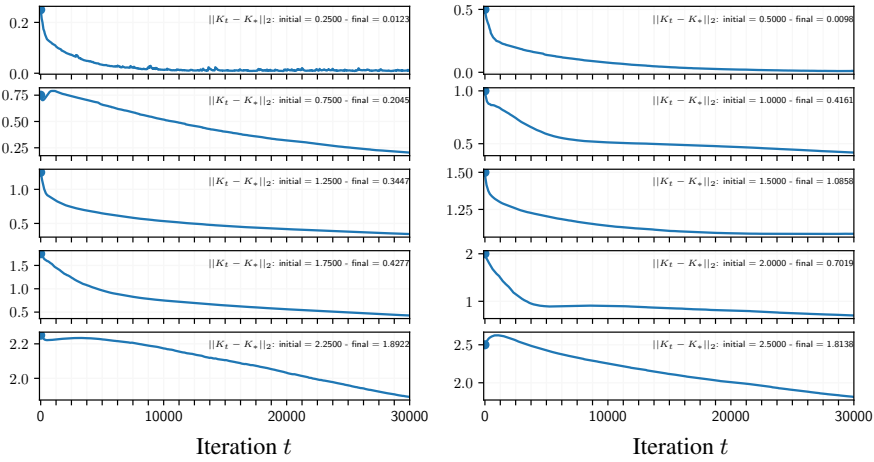

Figure 12: Example 1, offline learning with different initial guesses K_0, convergence of linear policy parameters to the optimal value K_*.

Finally, Figure 12 shows the convergence to K_* when starting the learning process from different initial policies K_0, increasingly distant from K_*. The noise standard deviation σ_y is set to 0.1 and the parameters in Table 2 are tuned accordingly. The remaining parameters are tuned analogously to the previous cases. It is useful to remember that, even if the stage costs ρ introduced in (3.2) are convex, the deriving cost function J_L expressed in (2.9) might not be convex with respect to the policy parameters, making the convergence to the global optimum hard to achieve. In this sense, Figure 12 shows that the policy updates associated with the proposed search method push the policy parameters closer to the optimal ones when the initial guess is chosen in a neighborhood of K_*.

3.2.2 Online setting

The Optimal Policy Search online algorithm is executed for $N_{\text{learn}} = 30000$ iterations, starting from an initial policy $K_0 = 0.0001 \cdot \underline{1}_{7\times 1}$. At each iteration t the current measured state x_t described in (2.2) is sampled as initial state ($N_0 = 1$), together with $N_q = 1$ initial integral action values, $N_r = 1$ reference

trajectories and N_d disturbances. The references and the integral action values are both sampled uniformly in $[-10, 10]$. The disturbances, instead, are sampled according to a Gaussian white random variable, characterized by variance σ_d^2. The sampled state is perturbed with $v_n \sim 0.1 \cdot \mathcal{N}(0, 1)$. For the update of the controller parameters, Adam is employed, with parameters $\alpha = 0.001$, $\beta_1 = \beta_2 = 0.9$. The parameters' optimization is carried on while an online tracking task is performed, starting from the plant being in an initial state x_0 such that $x_0 = [y_{ss}, \ldots, y_{ss}, u_{ss} \ldots, u_{ss}]'$, where (u_{ss}, y_{ss}) is a steady-state input-output couple. Considering as initial guess $\Theta_0 = \underline{0}_{1\times 6}$ and $P_0 = (1e+5) \cdot I_6$, the local models are recursively computed by Kalman Filtering, with noise covariance matrices $Q_k = (1e-8) \cdot I_6$ and R_k. The heuristic for stability described in Section 2.3.2.1 is implemented in this case.

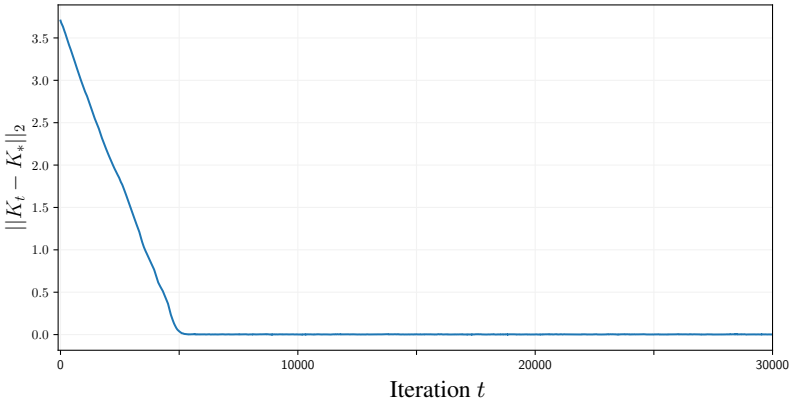

Figure 13: Example 1, online learning in noiseless conditions, convergence of the linear policy parameters to the optimal value K_*.

N_d	σ_d	R_k
1	0	$1e-8$

Table 3: Example 1, online learning, noiseless case, parameters

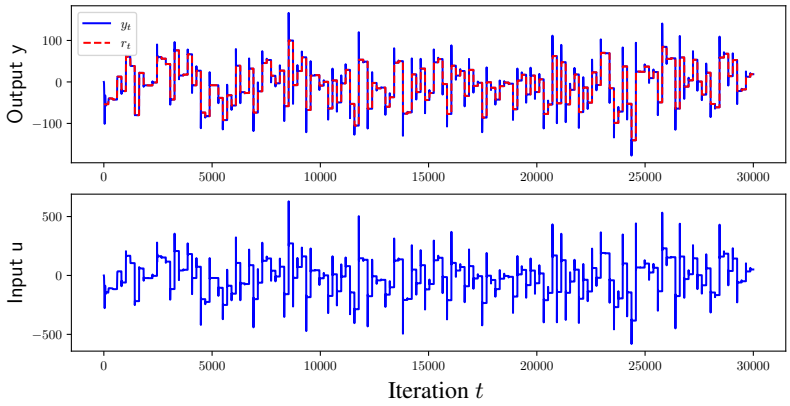

Figure 14: Example 1, online learning in noiseless conditions, online output tracking of an a-priori unknown exploratory reference signal.

The following two subsections contain the tuning details that differ between the noiseless case and the noisy one (that is, the values of parameters σ_y^2, N_d, σ_d, R_k, u_{ss}, y_{ss}), together with the results obtained in the two cases.

Noiseless case

As done in offline, in the online noiseless case we assume to be capable of measuring the exact output of (3.4), i.e., we set $\sigma_y^2 = 0$. We apply the online OPS method, with parameters values coherent with what presented in the online case introduction, together with the ones contained in Table 3. The parameters K_t quickly converge to the optimal policy K_*, as it is shown in Figure 13. While learning, the plant, starting from an initial state $x_0 = \underline{0}_{6\times 1}$ (i.e., $y_{ss} = u_{ss} = 0$) performs online a tracking task: the output y_t generated by the policy π_{K_t} follows the exploratory reference trajectory r_t, as represented in Figure 14.

N_d	σ_d	R_k
100	σ_y	σ_y^2

Table 4: Example 1, online learning, noisy case (noise variance σ_y^2), parameters

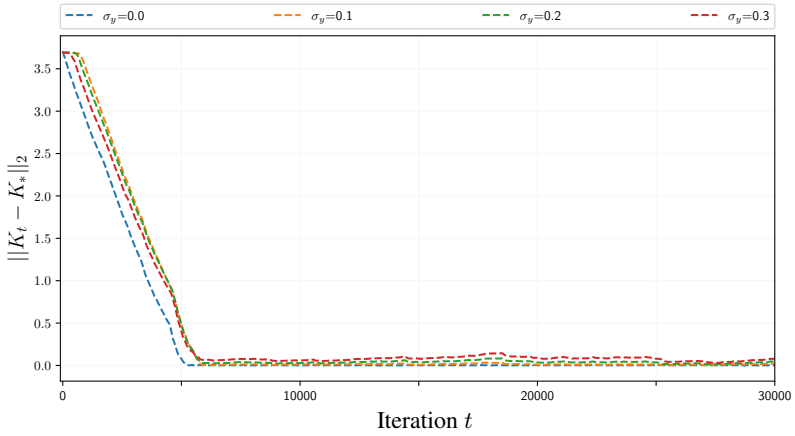

Figure 15: Example 1, online learning with noisy output ($\sigma_y = 0.0, 0.1, 0.2, 0.3$), convergence of the linear policy parameters to the optimal value K_*.

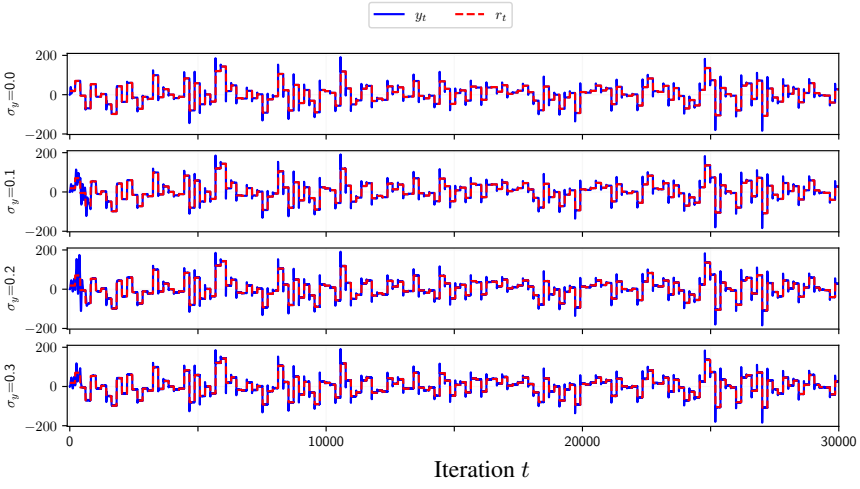

Figure 16: Example 1, online learning with noisy output ($\sigma_y = 0.0, 0.1, 0.2, 0.3$ from top to bottom), online output tracking of an a-priori unknown exploratory reference signal.

Noisy case

The online learning procedure is replcated as well in case of noisy output measurements, generated by corrupting y_t with samples from $\mathcal{N}(0, \sigma_y^2)$. The parameters values are kept equal to the one presented in the online case introduction, together with the ones indicate in Table 4.

We analyse the behavior of the algorithm considering increasing values of the noise standard deviation σ_y, namely $0.1, 0.2, 0.3$. Figure 15 shows the results in terms of convergence of K_t to K_* in the three scenarios. As a comparison, the noiseless case is also included, indicated as $\sigma_y = 0.0$. It can be noticed that the learned parameters converge to the optimal ones in all the four cases, although the evolution of the parameters in the noisy cases is affected by noise, resulting in "wavy" behavior.

The behavior of the controllers, performing online a tracking task while learning, presents few differences, mainly in the initial stages of the learning, as shown in Figure 16.

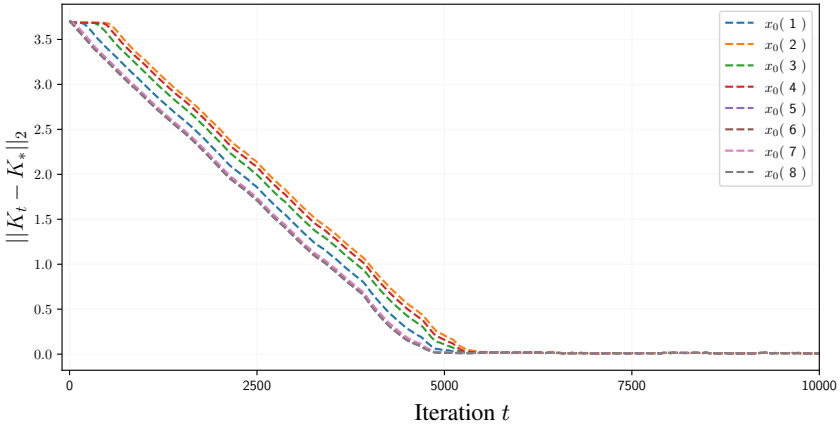

Figure 17: Example 1, online learning from different initial conditions, convergence of the linear policy parameters to the optimal value K_*.

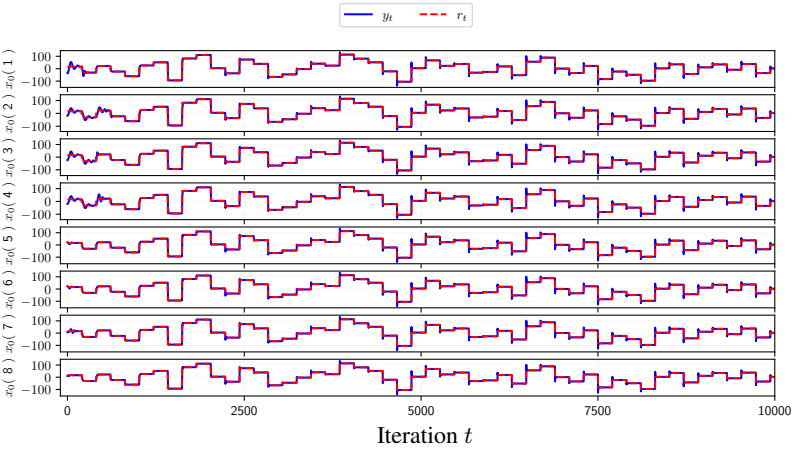

Figure 18: Example 1, online learning from different initial conditions, online output tracking of an a-priori unknown exploratory reference signal.

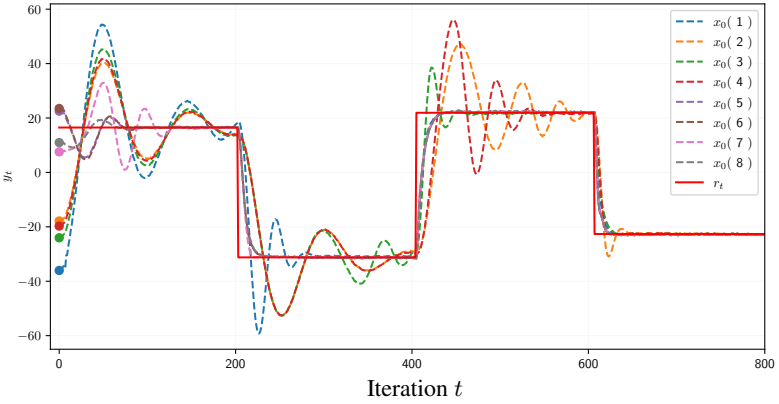

Figure 19: Example 1, online learning from different initial conditions, initial 800 steps: online output tracking of an a-priori unknown exploratory reference signal.

In Figure 17-18-19 convergence and tracking results are achieved from steady-state values obtained by sampling 8 different $u_{\text{ss}} \sim \mathcal{U}([-100,\, 100])$ and considering the associated steady-state output y_{ss}. Such values are used to build $x_0(j) = [y_{ss},\, \ldots,\, y_{ss},\, u_{ss},\, \ldots,\, u_{ss}]'$ as in (2.2) for $j = 1, 2, \ldots 8$. In this set of tests, noisy output are considered, with $\sigma_y = 0.1$, and the learning is carryed on for $N_{\text{learn}} = 10000$ steps. Even in this case the tuning is analogous to what presented in the online case introduction and with the parameters in Table 4.

From the figures we can observe that the influence of the initial conditions both on the learning and on the tracking performances is visible in the initial phase of the learning process, where the initial condition dictates the area of exploration for the algorithm, while it fades away further on.

3.3 Example 2 - LTI system - reduced feedback

The previous example served to show that if the local linear models are analogous in structure to the system dynamics (i.e., when the system is LTI and the numbers n_i and n_o of past input and output are chosen accordingly) and if the policy parameterization is chosen to have the same structure of the (known) optimal solution, the Optimal Policy Search method converges to it, both online and offline, handling noiseless or noisy data. The aim of the example presented in this section, instead, is to demonstrate the performance of the Optimal Policy Search method when synthesizing controllers without employing any knowledge of the structure of a system. In the first part of this section we show that Optimal Policy Search, although exploiting less informative and noisy feedback state, reaches near-to-optimal performance when applied to an LQR problem, by comparison with the known associated optimal policy. To evaluate the online tracking performance while learning it is shown that the closed-loop OPS behavior after an initial learning phase is very simlar to the one achieved by the DeePC method (Coulson, Lygeros, and Dörfler, 2019) in an unconstrained setting. The second part of this section, instead, presents a comparison between the performance achieved by Optimal Policy Search and VRFT (Campi, Lecchini, and Savaresi, 2002) in synthesising a controller from the same set of noisy measured data and exploiting the same policy parameterization.

For all the mentioned comparisons a SISO LTI system $x_{t+1} = A\,x_t + B\,u_t$, $y_t = C\,x_t$ is considered, such that

$$A = \begin{bmatrix} 0.311 & -0.010 & -0.075 & -0.011 & 0.012 & 0.113 \\ -0.006 & 0.373 & 0.130 & -0.218 & -0.076 & -0.133 \\ -0.079 & 0.137 & 0.200 & -0.133 & -0.037 & -0.085 \\ 0.008 & -0.219 & -0.139 & 0.317 & 0.132 & 0.139 \\ 0.012 & -0.083 & -0.032 & 0.127 & 0.428 & 0.168 \\ 0.111 & -0.120 & -0.093 & 0.149 & 0.167 & 0.302 \end{bmatrix},$$

$$B = \begin{bmatrix} -0.027 & -0.122 & 0.098 & 0.310 & 0.550 & 0.177 \end{bmatrix}',$$

$$C = \begin{bmatrix} -1.491 & 0 & -1.061 & 2.350 & 0 & 0.748 \end{bmatrix}.$$

In ARX form such system corresponds to

$$\begin{aligned} y_{t+1} = \;& 1.9312\; y_t \;-\; 1.3395\; y_{t-1} \;+\; 0.4335\; y_{t-2} + \\ & -\; 0.0692\; y_{t-3} \;+\; 0.0053\; y_{t-4} \;-\; 0.0001\; y_{t-5} + \\ & 0.7964\; u_t \;-\; 0.8400\; u_{t-1} \;+\; 0.3184\; u_{t-2} + \\ & -\; 0.0524\; u_{t-3} \;+\; 0.0038\; u_{t-4} \;-\; 9e-5\; u_{t-5}. \end{aligned} \tag{3.6}$$

To synthesize an output-tracking control using OPS we design the cost function in (3.3), choosing weights $Q_y = Q_y^L = 1$, $R = 0.1$, and $Q_q = Q_q^L = 1$ and horizon $L = 10$. As per the previous hypothesis we consider the system dynamics to be unknown, and we ignore as well the structure of the system, i.e., the proper input and output orders. We consider, instead, a feedback vector for the OPS controller composed by one past input and one past measured output, i.e., $n_o = 1$, $n_i = 1$. In this way the OPS algorithm does not employ any knowledge of the order of the system, relying on a reduced state $s_t^R = [x_t^R \;\; q_t]' = [y_t \;\; u_{t-1} \;\; q_t]'$ and $p_t = r_t$. The parameterized controller $\pi_K(s_t^R, r_t)$ is represented in closed-loop with the plant (3.6) in Figure 20. The training of such controller, according to the OPS approach, will involve as usual the use of local linear models of the form $y_{t+1} = \Theta_t \begin{bmatrix} x_t^R \\ u_t \end{bmatrix} + d_t$, hence, not sharing the structure of the nominal model of the plant.

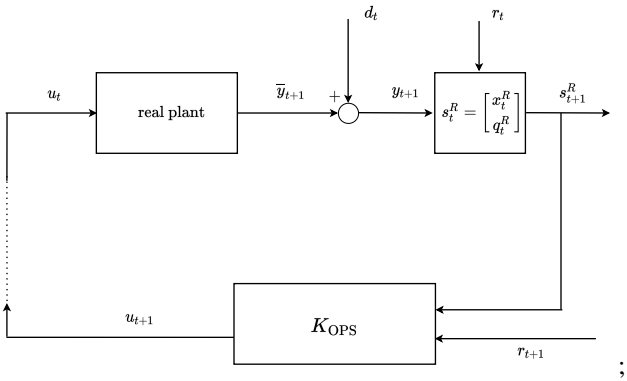

Figure 20: Example 2, closed-loop scheme for the OPS controller K_{OPS}.

3.3.1 Comparison with LQR and DeePC controllers

We choose to parameterize the input increment $\Delta u_t = u_t - u_{t-1}$ according to the linear expression in (2.22) and hence we consider

$$u_t = \pi_K(\, s_t^R,\, r_t\,) = u_{t-1} + K^s \begin{bmatrix} y_t \\ u_{t-1} \\ q_t \end{bmatrix} + K^r r_t.$$

In order to compute an optimal baseline controller to compare with the OPS controller, we consider the full feedback state

$$x_t^F = [\, y_t \quad y_{t-1} \quad \dots \quad y_{t-5} \quad u_{t-1} \quad \dots \quad u_{t-5}]'. \tag{3.7}$$

The evolution of the state $s_t^F = [\, x_t^F,\, q_t]'$, with respect to the input increment Δu_t can be described by an LTI system, built from (3.6), analogously to how (3.5) was built. It is possible to employ such system dynamics (equivalent to the nominal model of the system) together with classic Control Theory methods (the Riccati backward iterations for instance) to compute the optimal linear feedback controller K_{OPT} for the infinte horizon tracking of constant references, i. e.,

$$u_t = u_{t-1} + K_{\mathrm{OPT}} \begin{bmatrix} x_t^F \\ q_t \\ r_t \end{bmatrix}. \tag{3.8}$$

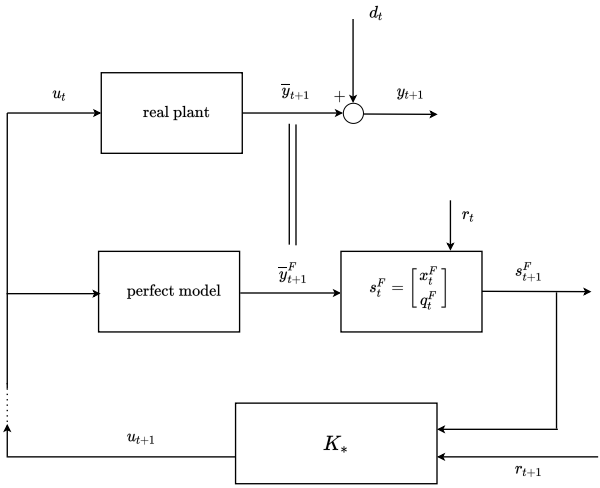

Figure 21: Example 2, closed-loop scheme for the baseline controller K_{OPT}.

Such controller will be used as baseline in our comparisons, considering the closed-loop scheme represented in Figure 21. It is important to underline that the baseline controller K_{OPT} is computed exploiting the exact dynamics of the system, while the OPS algorithm works under the "black-box" hypothesis, not knowing the real dynamics, and under-representing them with local linear models based on the reduced state x_t^R. Moreover, the baseline policy $K_{\mathrm{OPT}} \in \mathbb{R}^{13}$ acts on the filtered state (obtained from the nominal model dynamics), while the OPS-derived policy $K_{\mathrm{OPS}} \in \mathbb{R}^4$ employs inputs and outputs collected from the plant and, hence, possibly noisy. Additionally, as already mentioned, the OPS controller will rely on a reduced feedback vector of just 4 components, while the chosen baseline benefits of a more informative feedback from the plant, built with the knowledge of the appropriate order of the system.

Performance index In order to asses the similarity between the behavior of the policy K_{OPS} and the optimal one K_{OPT}, the following performance index is introduced

$$\Delta(K_{\mathrm{OPS}}) = \frac{|\mathrm{cost}(K_{\mathrm{OPS}}) - \mathrm{cost}(K_{\mathrm{OPT}})|}{\mathrm{cost}(K_{\mathrm{OPT}})}, \tag{3.9}$$

where

$$\mathrm{cost}(K) = \frac{\sum_{i=1}^{N_{\mathrm{task}}} \mathrm{cost}(K,\ \mathrm{task}_i)}{N_{\mathrm{task}}}. \tag{3.10}$$

In the previous equation $\mathrm{cost}(K,\ \mathrm{task}_i)$ is the cost of performing the i-th task of a batch of N_{task} tracking tasks. The i-th task, characterized by a reference signal $\{\, r_l^i \,\}_{l=0}^{N_{\mathrm{st}}}$ and a steady-state initial condition $(u_{-1}^i,\ y_0^i)$, is performed through a policy π_K.

In the noiseless scenario we can consider

$$\mathrm{cost}(K,\ \mathrm{task}_i) = J_{N_{\mathrm{st}}}(\, K,\, s_0^i,\, \{\, r_l^i \,\}_{l=0}^{N_{\mathrm{st}}},\, \{\, 0 \,\}_{l=0}^{N_{\mathrm{st}}-1}\,), \tag{3.11}$$

while in the presence of noise over the output measurements, the performance of π_K is evaluated by means of the average cost with respect to N_{noise} realizations of disturbance trajectory $\{\, d_l^k \,\}_{l=0}^{N_{\mathrm{st}}-1}$, i.e.,

$$\mathrm{cost}(K,\ \mathrm{task}_i) = \frac{\sum_{k=1}^{N_{\mathrm{noise}}} J_{N_{\mathrm{st}}}(\, K,\, s_0^i,\, \{\, r_l^i \,\}_{l=0}^{N_{\mathrm{st}}},\, \{\, d_l^k \,\}_{l=0}^{N_{\mathrm{st}}-1}\,)}{N_{\mathrm{noise}}}. \tag{3.12}$$

In both (3.11) and (3.12) $J_{N_{\mathrm{st}}}$ is defined as in (2.5), employing the stage cost (3.2), and $(u_{-1}^i,\ y_0^i)$ are used to build the i-th initial state s_0^i.

As previously specified the disturbance signal will affect the control inputs generated by K_{OPS}, because it acts on a measured feedback state, while the inputs generated by K_{OPT}, acting on a filtered state, are unaffected by disturbance (see Figure 20-21). Moreover, we assume (exclusively for the performance evaluation of the two controllers) to have access to the original outputs $\overline{y}_t$, not corrupted by noise, and to the associated integral $\overline{q}_{t+1} = \overline{q}_t + (\overline{y}_t - r_t)$. Hence the disturbance signals will affect solely the generation of the OPS inputs and not the computation of the stage costs composing the performance index per se.

3.3.1.1 Offline setting

A set of $N_{\mathrm{data}} = 30000$ input-output couples is collected from the plant, by giving in input to the system a random sequence of piecewise constant input and collecting the associated output. After building the states history X and the associated local linear models history Θ (with $\Theta_0 = \underline{0}_{1\times 3}$, $P_0 = (1e+5)\cdot I_3$, $Q_k = I_3$ and $R_k = 0.01$), the offline learning procedure is carried on for

$N_{\text{learn}} = 10000$ iterations, starting from the initial guess $K_0 = 0.0001 \cdot \underline{1}_{4\times 1}$. At each iteration the sampling procedure described in Chapter 2, Section 2.2.2.1 is employed, considering $N_0 = 10$ initial states, $N_r = 1$ references, $N_q = 10$ integral action values and N_d disturbances. The initial states are sampled uniformly from the states history, with a small perturbation $v_n \sim 0.1 \cdot \mathcal{N}(0, 1)$. References and integral actions are sampled uniformly in $[-100, 100]$ and $[-10, 10]$, respectively. The disturbances, instead, are sampled according to $\mathcal{N}(0, \sigma_d^2)$.

For the update of the controller parameters, AMSGrad is employed, with parameters $\alpha = 0.1$, $\beta_1 = 0.5$, $\beta_2 = 0.9999$. The following is divided in two subsections, containing the values of parameters σ_y^2, N_d, σ_d and the results of the comparison between K_{OPS} and K_{OPT} in the noiseless and noisy case respectively.

Noiseless case

In this case we consider the noise variance σ_y^2 to be equal to zero and we assume that there is no disturbance d_t affecting the local linear models ($N_d = 1$, $\sigma_d = 0$). Figure 22 shows the noiseless dataset employed for the offline noiseless learning. Figure 23 shows the evolution of the policy parameters K_t during the learning phase, till their convergence to a policy that we indicate as K_{OPS}.

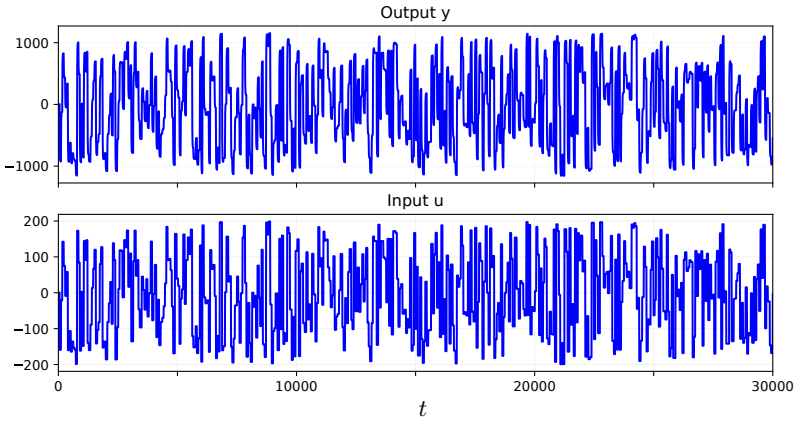

Figure 22: Example 2, offline learning, open-loop noiseless dataset.

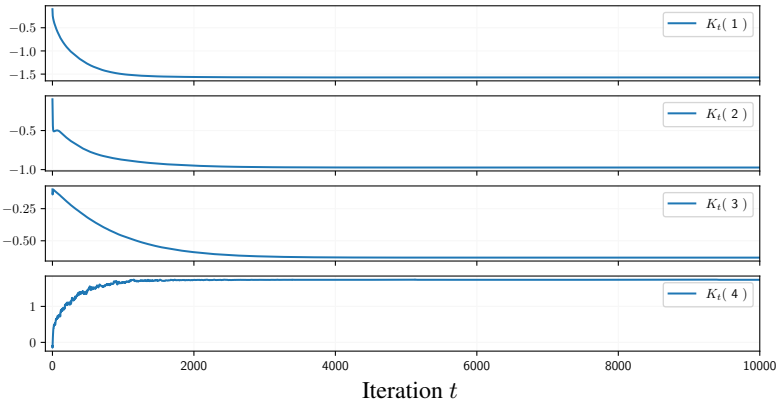

Figure 23: Example 2, offline learning in noiseless conditions, evolution of the linear policy parameters.

The controllers K_{OPT} and K_{OPS} are then applied and compared on two test batches, each of them composed by $N_{\mathrm{task}} = 100$ random tracking tasks of length $N_{\mathrm{st}} = 60$, one characterized by constant reference signals, and the other by piecewise constant references, both combined with randomly generated initial steady-state conditions (u_{-1}, y_0). Figure 24 shows examples of the two controllers' performance over two of the random tasks in the test batches, demonstrating the close similarity of their behavior in terms of y_t, u_t and q_t. This similarity is quite uniform over the two test batches of constant and piecewise constant tracking tasks, as expressed by the performance index (3.9) in Table 5, and by the cost of the two controllers in Table 6. In particular K_{OPT} slightly outperforms K_{OPS} over the constant references tracking, where it is actually known to be the optimal controller. The performance of the two policies are very similar also in tracking piecewise signals, with a slightly better result obtained by K_{OPS}.

	constant r_t	piecewise constant r_t
$\Delta(K_{\mathrm{OPS}})$	$6e-5$	$6e-5$

Table 5: Example 2, offline learning, noiseless case, performance index (3.9).

	constant r_t	piecewise constant r_t
K_{OPT}	13060.9	22053.2
K_{OPS}	13061.7	22051.8

Table 6: Example 2, offline learning, noiseless case, averaged cost over the described test batches.

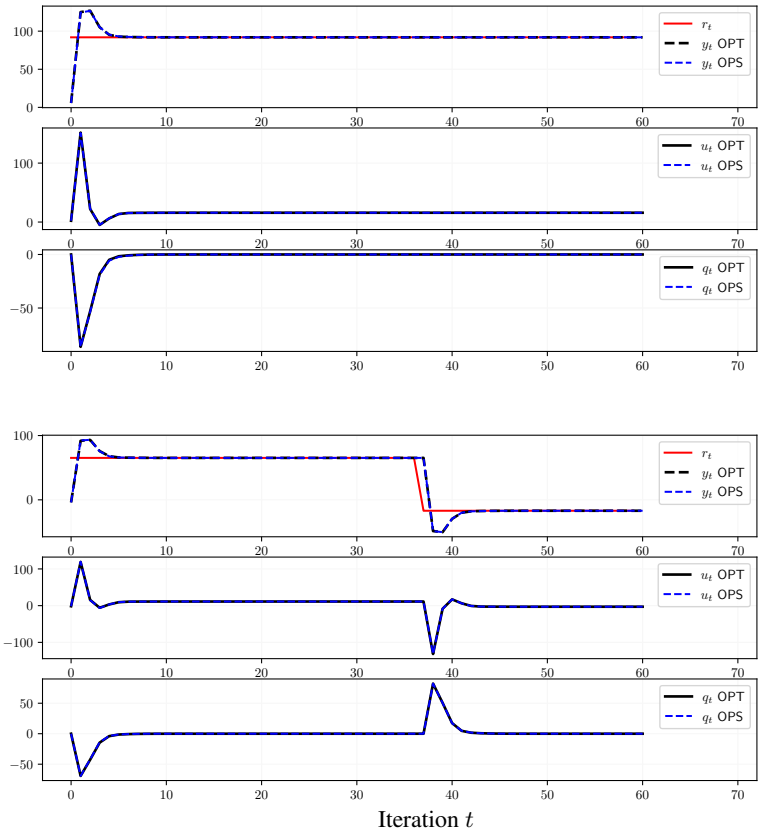

Figure 24: Example 2, offline noiseless case, tracking of a constant reference (top 3 plots), and of a piecewise constant reference (bottom 3 plots)

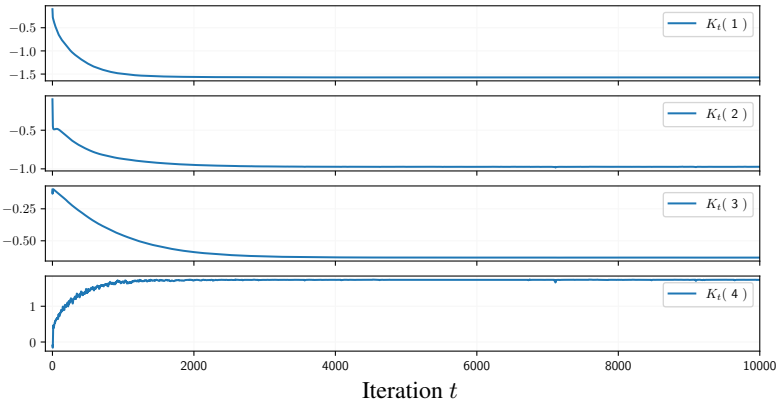

Figure 25: Example 2, offline learning in noisy conditions, evolution of the linear policy parameters.

Noisy case

To work in a noisy scenario, white gaussian noise with standard deviation $\sigma_y = 0.1$ is then added to the stream of output collected in open-loop shown in Figure 22, and the offline learning is performed over this new dataset. The employed parameters related to the sampling of the disturbance trajectories are the following: $N_d = 10$ and $\sigma_d = 0.1$. Figure 25 shows the convergence of the parameters to the final policy K_{OPS} during the learning iterations.

The obtained policy is tested against K_{OPT} in performing two random batches of tracking tasks (constant and piecewise constant references, both batches with randomly generated initial steady-state conditions $(u_{-1},\, y_0)$), each of them containing $N_{\mathrm{task}} = 100$ tasks of length $N_{\mathrm{st}} = 60$. For each task the test is repeated considering $N_{\mathrm{noise}} = 200$ different realizations for the disturbance trajectory.

Table 7 includes the performance index (3.9) and the averaged cost obtained by learning and testing the policy in noisy conditions. Even in this case, from the results shown in the two tables and from the examples of tracking in Figure 26 we can see that the two controllers performance are very similar even in the presence of noise affecting the output.

	constant r_t	piecewise constant r_t
$\Delta(K_{\mathrm{OPS}})$	$2e-3$	$8e-5$
K_{OPT}	9001.1	25335.4
K_{OPS}	9021.3	25337.6

Table 7: Example 2, offline learning, noisy case, performance index (3.9) and averaged cost over the described test batches.

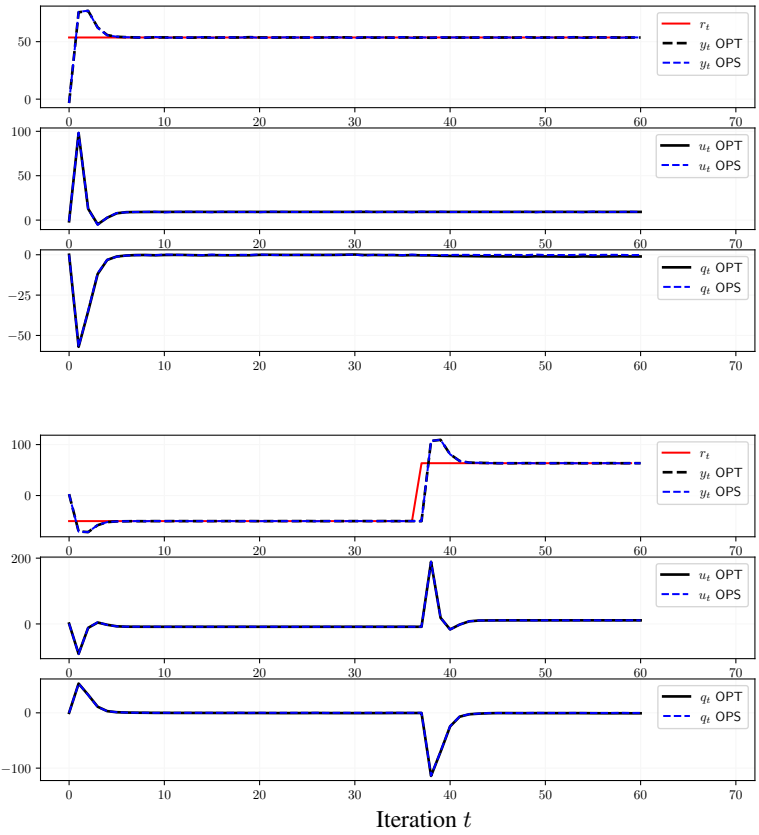

Figure 26: Example 2, offline noisy case, tracking of a constant reference (top 3 plots), and of a piecewise constant reference (bottom 3 plots)

3.3.1.2 Online setting

The OPS online learning procedure is executed for $N_{\text{learn}} = 20000$ iterations, from an initial policy $K_0 = 0.0001 \cdot \underline{1}_{4\times 1}$, and starting from $x_0 = [0\ ,0\ ,0\ ,0]'$. At each iteration t the current measured state x_t^R is sampled ($N_0 = 1$), together with $N_q = 50$ initial integral action values and $N_r = 10$ reference trajectories, sampled uniformly in $[-100, 100]$ and $[10, 10]$, respectively. The sampled state is perturbed with $v_n \sim 0.1 \cdot \mathcal{N}(0, 1)$. A set of N_d disturbance trajectories are sampled from $\mathcal{N}(0, \sigma_d^2)$. Starting from $\Theta_0 = \underline{0}_{1\times 3}$ and $P_0 = (1e + 5) \cdot I_3$, the local models are computed and stored online, recursively updating it by Kalman Filtering (with $Q_k = I_3$ and $R_k = 1$). The controller parameters are updated using Adam, with $\alpha = 0.1$, $\beta_1 = 0.5$, $\beta_2 = 0.9999$. The heuristic for stability described in Section 2.3.2.1 is implemented. The noiseless and noisy online cases are tuned differently only regarding the parameters N_d and σ_d associated with the sampling of the disturbances. The specific values for such parameters in each of the cases will follow in the appropriate subsection, together with the comparison between K_{OPS} and K_{OPT}.

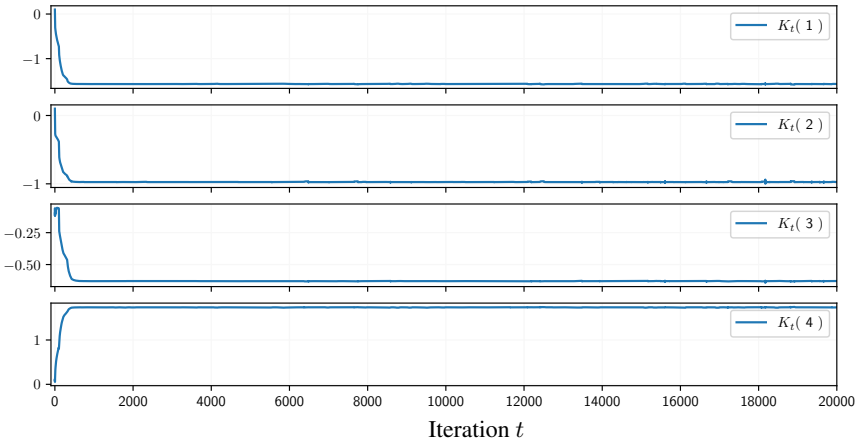

Figure 27: Example 2, online learning in noiseless conditions, evolution of the linear policy parameters.

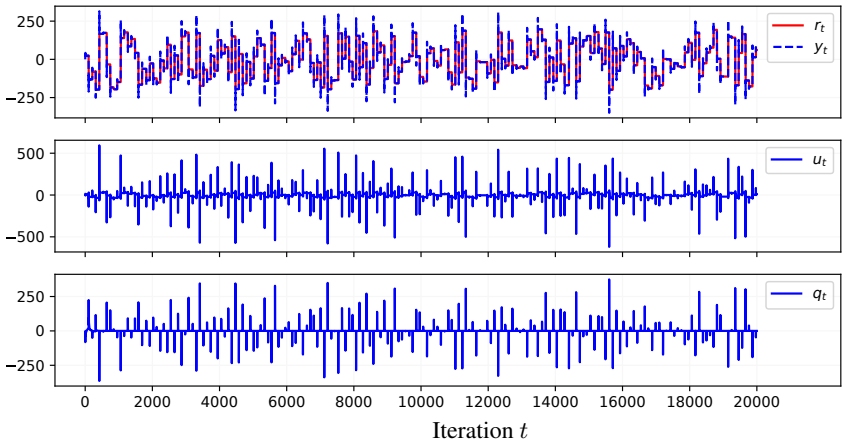

Figure 28: Example 2, online learning in noiseless condition, online output tracking of an a-priori unknown exploratory reference signal.

Noiseless case

As in the noiseless offline case, we initially assume that there is no noise affecting the output measurement ($\sigma_y^2 = 0$) and that there is no disturbance d_t affecting the local linear models ($N_d = 1$, $\sigma_d = 0$).

The parameters K_t quickly converge to a vector indicated as K_{OPS}, as it is shown in Figure 27. While learning, the algorithm performs online a tracking task using the parameterized policy, as represented in Figure 28.

	constant r_t	piecewise constant r_t
$\Delta(K_{\text{OPS}})$	$6e-5$	$6e-5$
K_{OPT}	10008.9	18547.9
K_{OPS}	10009.5	18549.2

Table 8: Example 2, online learning, noiseless case, performance index (3.9) and averaged cost over the described test batches.

In order to be compared, K_{OPT} and K_{OPS} are applied on two test batches, each of them composed by $N_{\mathrm{task}} = 100$ randomly generated tracking tasks of length $N_{\mathrm{st}} = 60$, one characterized by constant reference signals, and the other by piecewise constant references, both combined with randomly generated initial steady-state conditions (u_{-1}, y_0). The results in terms of performance index and cost are shown in Table 8, and the similarity in the behavior of the two controllers can be seen also in the two examples in Figure 29.

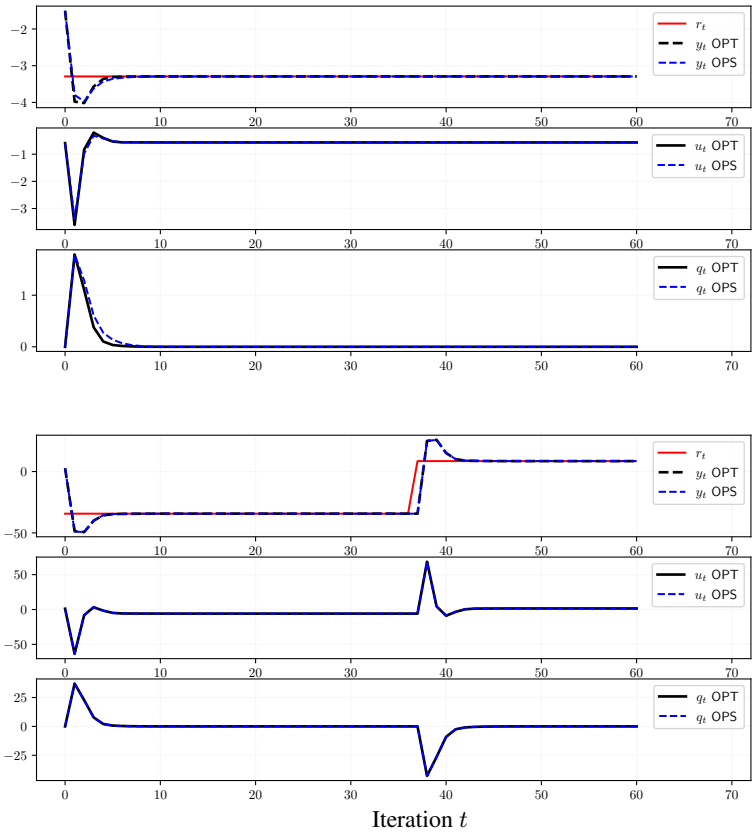

Figure 29: Example 2, online noiseless case, tracking of a constant reference (top 3 plots), and of a piecewise constant reference (bottom 3 plots)

Noisy case

In this subsection are included the results obtained in noisy conditions, so when white gaussian noise with standard deviation $\sigma_y = 0.1$ is added to the measured stream of output. In this case we choose the parameters related to the disturbance sampling as $N_d = 10$ and $\sigma_d = 0.1$.

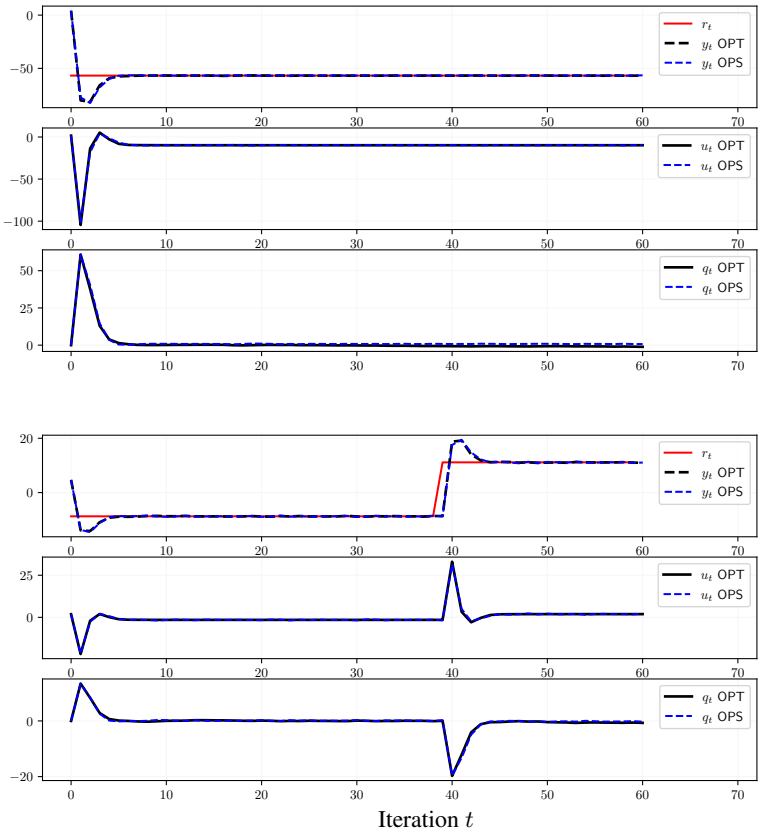

Figure 30: Example 2, online noisy case, tracking of a constant reference (top 3 plots), and of a piecewise constant reference (bottom 3 plots)

The obtained policy is tested against K_{OPT} in performing two random batches of tracking tasks composed by constant and piecewise constant references respectively, both batches with randomly generated initial steady-state conditions $(u_{-1},\, y_0)$. Each of the batch contains $N_{\mathrm{task}} = 100$ tracking tasks of length $N_{\mathrm{st}} = 60$. For each task the test is repeated considering $N_{\mathrm{noise}} = 200$ different realization for the disturbance trajectory. Even in the online noisy case it is possible to observe the similarity in the behavior of the two controllers in performing both constant and piecewise constant tracking tasks. This aspect is underlined by the indices in Table 9, showing the proximity in the cost attained by the two controllers. Additionally the tracking examples in Figure 30 show once more how not only the output tracking is similar, but also the input and the integrals generated by the two controllers are close.

In summary, it is possible to observe that in both noiseless and noisy scenarios, offline and online, the controllers obtained by Optimal Policy Search perform similarly to the LQR controller obtained using the Riccati iterations. This result is achieved even though Optimal Policy Search does not employ knowledge of the underlying dynamical system, exploits a reduced feedback state x_t^R instead of the full state x_t^F over which K_{OPT} acts, and works over measured input and output streams, while K_{OPT} utilizes the real model of the plant to filter out the noise from the feedback state. This consideration is based on the performance achieved by both the controllers when tested in the aforementioned scenarios over two batches of tracking tasks, composed by constant and piecewise constant references, respectively . The result obtained over the constant references batch is particularly important, given that the Riccati-derived LQR controller considered as competitor is known from theory to be the optimal controller for the tracking of such signals.

	constant r_t	piecewise constant r_t
$\Delta(K_{\mathrm{OPS}})$	$8e-3$	$5e-4$
K_{OPT}	10403.6	23490.1
K_{OPS}	10496.5	23477.0

Table 9: Example 2, online learning, noisy case, performance index (3.9) and averaged cost over the described test batches.

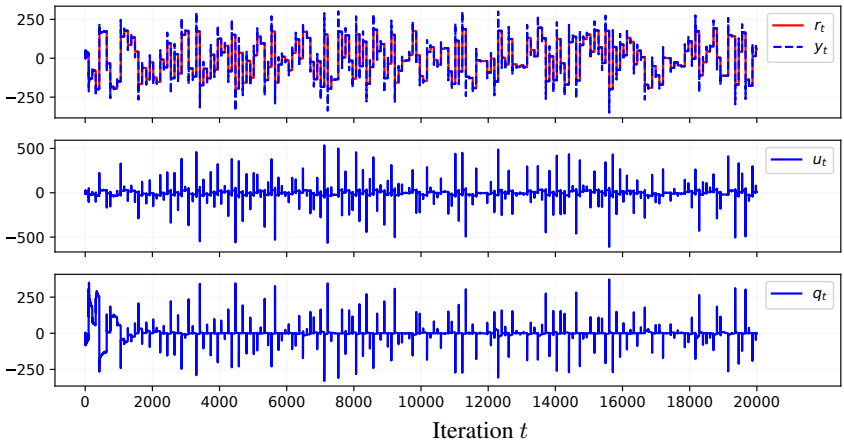

Figure 31: Example 2, online learning in noisy condition, online output tracking of an a-priori unknown exploratory reference signal.

We now analyze the quality of the online tracking performance achieved by Optimal Policy Search while learning from noisy data, shown in Figure 31. In order to do so we compare it with the performance obtained over the same reference signal by a data-driven controller synthesized using the DeePC method (Coulson, Lygeros, and Dörfler, 2019).

As already introduced in Section 1.1 DeePC employs the receding-horizon MPC paradigm, substituting a more complicated model of the plant dynamics with the fitting of a linear combination of previously collected trajectories, generated by an exciting input stream.

DeePC inherits from the MPC frameworks many advantageous characteristics. For instance, it permits to easily incorporate input and output constraints, while additional work still need to be done in order to extend Optimal Policy Search and include constraint satisfaction. DeePC generates at every instant a locally optimal action, by optimizing a performance index over a chosen horizon, with initial condition based on the current state of the plant. Moreover applications demonstrate that the method is quite data efficient, relying on rela-

tively small amounts of previously collected input and output samples. Instead, OPS relies on a policy parameterization, its goal being to globally optimize the policy parameters in order to handle all the possible scenarios. As a consequence of this DeePC is expected to provide better performance, at the expense of solving a Quadratic Programming (QP) problem at every iteration.

The performance of OPS could be limited by the choice of a sub-optimal policy parameterization or by the global nature of the policy parameters. However, after convergence, only a simple function evaluation is necessary to compute the control inputs in OPS.

Another difference of the two methods is that, although small, a previously collected dataset is required in order to build the Hankel matrices constituting the data-driven behavioral model exploited by DeePC. OPS instead might require a longer training session, but it is possible to conjugate training and plant control, as shown in this section, or to use off-policy learning (eventually combining it with on-policy phases, as discussed in Section 3.1.2), in order to learn in safety conditions.

Following the notation in (Coulson, Lygeros, and Dörfler, 2019) we design a DeePC controller in the following way: a dataset composed by the first $N_d = 50$ couples $\{(u_t^d,\, y_{t+1}^d)\}_{t=0}^{N_d-1}$ of inputs and noisy outputs generated by OPS during the online learning in Figure 31 is used to build the Hankel matrices $H_{T+N}(u^d)$, $H_{T+N}(y^d)$ that will substitute a designed mathematical model for the predictive control scheme. It has to be underlined that, although the full set of $N_{\text{learn}} = 20000$ inputs and outpus constituting the trajectory in Figure 31 could be used to build such matrix, this would have been resulted in large QP problems to be solved at each iteration, and hence in a considerable increase of the required computational time. Moreover, the first samples collected by OPS while learning tend to be quite explorative, being generated by almost untrained policy parameters. Each column of $H_{T+N}(u^d)$ contains a trajectory of $T + N$ sequential input values, divided into T "past" input and N "future" inputs. The same structure is replicated in $H_{T+N}(y^d)$ with the associated output values.

Based on the Fundamental Lemma (Markovsky and Rapisarda, 2008), DeePC uses the Hankel matrices to predict each new input-output trajectory as a linear combination of the previous ones, collected in the Haenkel matrices. In particu-

lar, in order to predict the future N input and output u_f, y_f of a trajectory τ that is already composed by T input and output u_p, y_p, the method fits a parameter vector g such that

$$\begin{bmatrix} U_p \\ Y_p \\ U_f \\ Y_f \end{bmatrix} g = \begin{bmatrix} u_p \\ y_p \\ u_f \\ y_f \end{bmatrix}, \tag{3.13}$$

where U_p consists of the first T block rows of $H_{T+N}(u)$, and U_f of the last N ones (analogously for Y_p, Y_f and $H_{T+N}(y)$).

At every iteration, DeePC optimizes the quadratic cost function of the tracking problem (2.5) with stage costs (3.2), with respect to the future N inputs and outputs u_f, y_f, and the parameters g, considering as initial conditions the last T measured inputs and noisy outputs u_p, y_p and the current reference signal r_t to be tracked, that will be considered as constant over the future horizon N (as it is done in the OPS tracking problem formulation (3.3)). In order to have a fair comparison with OPS and optimize the same cost the behavioral model (3.13) enforced as constraint in the predictive scheme is augmented with the known dynamics of the integral action $q_{t+1} = q_t + y_t - r_t$. The same weights Q_y, Q_q, R, Q_y^L and Q_q^L considered for the OPS optimization are employed here as well. The first of the predicted future output, contained in u_f, is then applied to the plant. The same predictive horizon employed for OPS is considered, i.e. $N = L = 10$. Regarding the length of the past trajectories, given that the OPS method relies on local linear models based on the reduced state $x_t^R = [y_t,\, u_{t-1}]$, we consider as past values one previous input and output couple, i.e. $T = 1$.

Given that the Fundamental Lemma does not hold for nonlinear systems or noisy systems, in such cases DeePC relies on regularization terms $||g||_1$ and $||Y_p g - y_p||_1$ to be added to the tracking cost over the fixed horizon, weighted by tuning parameters λ_1 and λ_2. We tune such parameters as $\lambda_1 = \lambda_2 = 300$.

The DeePC control routine is runned considering the a-priori unknown piecewise constant reference signal r_t in Figure 31. The measured output employed by DeePC are altered with additive noise, considering the same noise trajectory realization used as well during the OPS learning phase.

As expected DeePC, relying on the built Hankel matrix and on the local optimization, is capable of successfully tracking the given reference signal, achieving a lower global cost over the $N_{\text{learn}} = 20000$ steps horizon, if compared with the one obtained by OPS while learning the parameters.

In Figure 32-33 the DeePC and OPS behaviors are compared, considering the last 1000 steps of online learning and tracking. The figures show the tracking performance achieved by both methods, and the associated input streams. It is noticeable observing both the plots that towards the end of the learning Optimal Policy Search converges to a policy attaining a closed-loop behavior that is quite similar to the one of DeePC.

The mentioned figures can help in qualitatively assessing the similarity of the two behaviors. The same evaluation can be derived quantitatively from the tracking costs achieved by the two methods in the last 1000 steps of the tracking task, presented in Table 11. We can observe that indeed the two online costs are quite close in value, as it can be expected given the shown similarity in the close-loop performance.

In conclusion, we can observe that despite not computing an optimal local choice at every instant, the OPS controller after a training phase is capable of reaching online performance that is close to the ones of DeePC. The similarity in behavior of OPS and DeePC encourages the learning of OPS controllers on-policy or off-policy, followed by the application of the obtained controller with the optimized parameters, a choice that is computationally cheaper than solving a QP problem at every instant of time.

	tracking cost
OPS online	839015.20
DeePC	842841.45

Table 10: Example 2, OPS vs DeePC online tracking cost comparison - last 1000 steps.

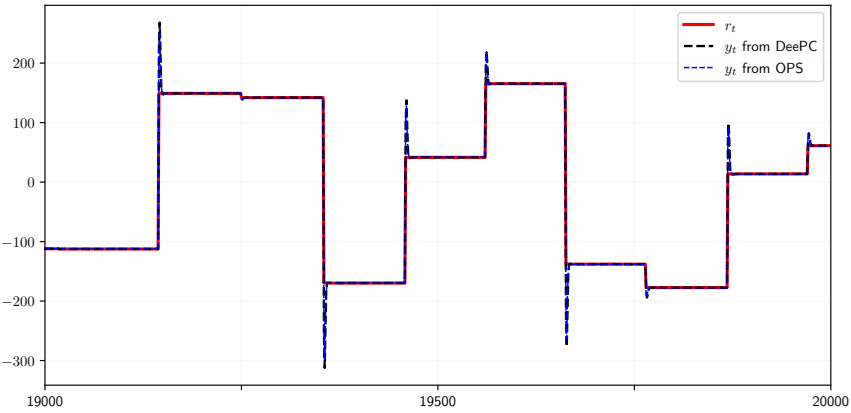

Figure 32: Example 2, noisy case, online comparison OPS vs DeePC, output tracking of an a-priori unknown exploratory reference signal, last 1000 steps.

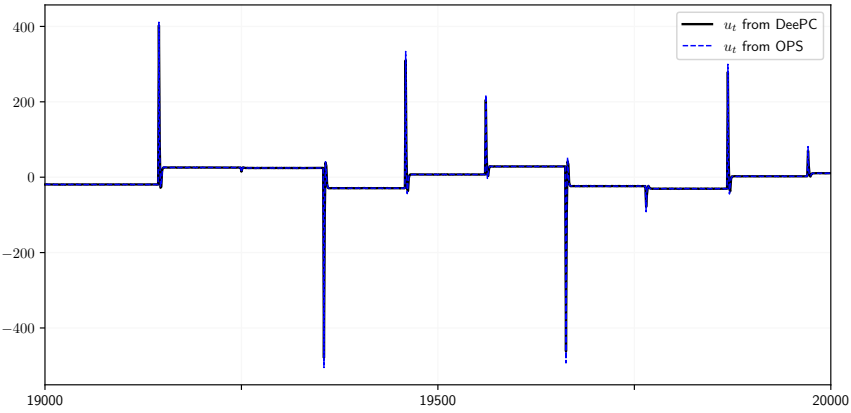

Figure 33: Example 2, noisy case, online output tracking of an a-priori unknown exploratory reference signal, last 1000 steps, comparison OPS vs DeePC, series of inputs fed to the plant online.

3.3.2 Comparison with VRFT controllers

As introduced in Section 1.1, VRFT (Campi, Lecchini, and Savaresi, 2002) is a data-driven method, synthesizing the parameters of a controller based on a stream of input/output data previously collected in open-loop, and on a user-defined reference model of the closed-loop behavior to be attained. The most important difference between VRFT and Optimal Policy Search is hence that the first one requires a reference model, while the second can be applied directly, exploiting only the collected data, or even in an online setup, controlling the plant and collecting the data while optimizing the policy parameterization. On the other hand, while Optimal Policy Search is an iterative gradient-based method, VRFT is non-iterative and solves a linear least-squares problem, hence it does not have local minimum and does not experience initialization problems. Both methods rely on the quality of the collected data, in terms of information carried on the unknown plant dynamics. VRFT, in particular is known to directly search for the optimal solution relative to the information contained in the given batch of data, hence resulting on poorly performing controllers in case of lack of excitation in the data (Campi and Savaresi, 2006). Although the importance on explorative data for the synthesis of an OPS-derived controller is intuitive, the details on the effect of low informative data on the presented method are still under investigation.

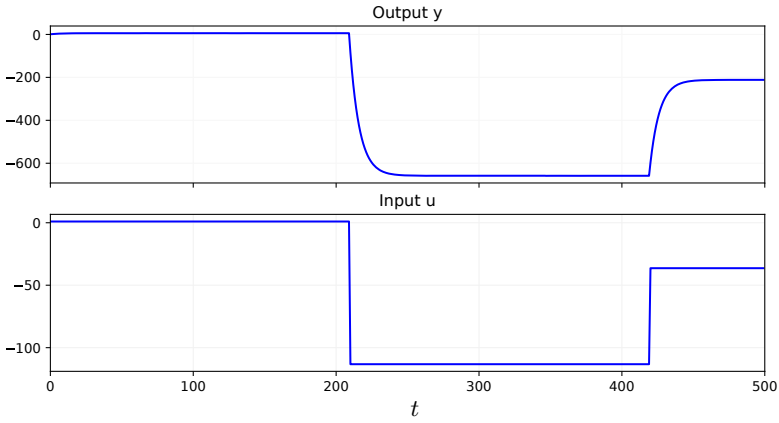

Figure 34: Example 2, OPS vs VRFT comparison, open-loop dataset.

The idea behind this example is to demonstrate that the OPS method is capable of exploiting the information carried by a small dataset in a more efficient way with respect to VRFT, resulting in a better performing controller. In order to fairly compare OPS and VRFT we consider the offline version of the OPS method and we design the controllers for both the methods using the same parameterization. In particular, we model the input u_t as

$$u_t = \pi_K(\, s_t^R = \begin{bmatrix} y_t \\ u_{t-1} \\ q_t \end{bmatrix}, \, r_t \,) = K^e(r_t - y_t) + K^q q_t.$$

The same small set of $N_{\text{data}} = 500$ input-output couples represented in Figure 34 is employed by both OPS and VRFT. White noise with standard deviation $\sigma_y = 0.1$ is added to the output y_t contained in such dataset.

As anticipated, to design the VRFT competitor we have to choose a reference model for the desired closed-loop behavior. In this section we consider two different VRFT-derived controllers, associated with two different reference models. The first reference model, indicated as M_*, is based on full knowledge of the true dynamics (3.6) of the plant, making the associated VRFT controller parameters K^*_{VRFT} not completely data-driven. The second one, denoted as M_{d}, is instead built from the available stream of data, without employing knowledge on the system dynamics or on its order.

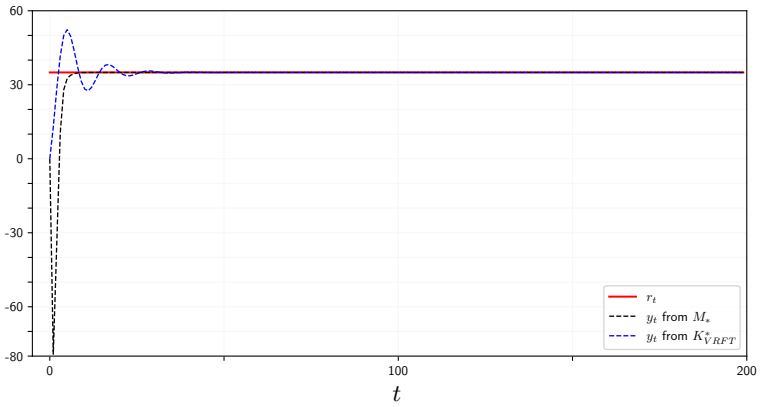

Figure 35: Example 2, OPS vs VRFT comparison, example of tracking behavior of M_*, and of K^*_{VRFT} in closed-loop with the plant.

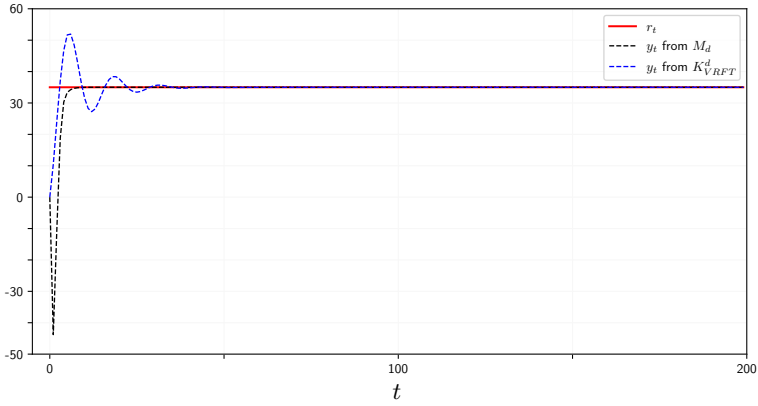

Figure 36: Example 2, OPS vs VRFT comparison, example of tracking behavior of M_d, and of K^d_{VRFT} in closed-loop with the plant.

Hence the VRFT controller parameters K^d_{VRFT} derived from it are properly data-driven. To obtain M_* we consider the true dynamics of the full state $s^F_t = [\, x^F_t,\, q_t]'$ with x^F_t defined in (3.7), in closed-loop with the optimal linear feedback controller K_{OPT} (3.8) for the infinte horizon tracking of constant references. Such controller is obtained by solving the LQR problem associated with the weights $Q_y = 1$, $R = 0.1$, and $Q_q = 1$ employed as well for the OPS optimization. In order to obtain M_d, instead, we use the dataset in Figure 34 to fit a linear model Θ_d mapping the ouputs y_{t+1} and the regressors $x^R_t = [\, y_t,\, u_{t-1}\,]$. The model Θ_d has the same reduced order and the same regressors of the local linear models exploited by OPS. We use Θ_d, augmenting it in order to approximate the dynamics of the reduced state $s^R_t = [\, x^R_t,\, q_t]'$. Even in this case we consider as reference model M_d the approximated dynamics of s^R_t in closed-loop with the reduced order optimal linear feedback controller K^R_{OPT} for the infinte horizon tracking of constant references associated with the same weights optimized by OPS. Both the VRFT controllers are synthesised using the MATLAB VRFT ToolBox (Carè et al., 2019). An example of the tracking behavior of K^*_{VRFT} and K^d_{VRFT}, together with the associated reference model M_* and M_d can be found in Figure 35 - 36. Based on Figure 36 we can observe that,

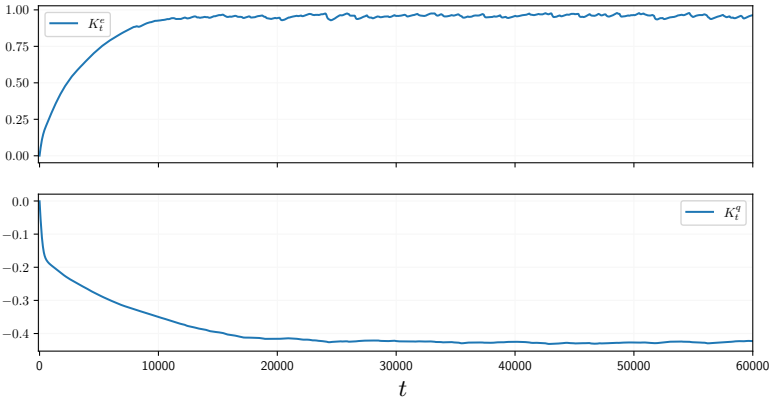

Figure 37: Example 2, OPS vs VRFT comparison, evolution of the linear OPS policy parameters.

although the dataset in Figure 34 is small and generated by a modestly exciting piecewise constant input, it is informative enough to permit the identification of a model Θ_d and then derive a controller K^R_{OPT} such that their behavior in closed-loop M_d is similar to the optimal closed-loop M_* shown in Figure 35.

The OPS synthesis is performed as follows: after building the states history X and the associated local linear models history Θ (with $\Theta_0 = \underline{0}_{1\times 3}$, $P_0 = (1e+5)\cdot I_3$, $Q_k = I_3$ and $R_k = 0.01$), the offline learning procedure is carried on for $N_{\text{learn}} = 60000$ iterations, starting from the initial guess $K_0 = 0.0001\cdot\underline{1}_{2\times 1}$. At each iteration the sampling procedure described in Chapter 2, Section 2.2.2.1 is employed, considering $N_0 = 1$ initial states, $N_r = 1$ references, $N_q = 1$ integral action values and $N_d = 10$ disturbances. The initial states are sampled uniformly from the states history, with a small perturbation $v_n \sim 0.1\cdot\mathcal{N}(0,1)$. References and integral actions are sampled uniformly in $[-100,\ 100]$ and $[-10,\ 10]$, respectively. The disturbances, instead, are sampled according to $\mathcal{N}(0,\sigma_d^2)$, with $\sigma_d^2 = 0.01$. For the update of the controller parameters, AMSGrad is employed, with parameters $\alpha = 0.001$, $\beta_1 = 0.99$, $\beta_2 = 0.9999$. Figure 37 shows the evolution of the policy parameters learned by the OPS method. The obtained controller is indicated as K_{OPS}.

To run the comparison we use K^*_{VRFT}, K^d_{VRFT} and K_{OPS} in closed-loop with the original plant, performing a serie of tracking tasks, and we consider as performance index the cost defined in (3.10) - (3.12). Such cost is the averaged tracking cost with horizon N_{st} over a batch of N_{task} tracking tasks, each of them averaged as well with respect to N_{noise} disturbance signal realizations. In particular, in order to compare K^*_{VRFT}, K^d_{VRFT} and K_{OPS} we consider three different task batches, each of them characterized by $N_{\text{task}} = 100$, $N_{\text{noise}} = 100$, and $N_{\text{st}} = 200$. The tracking tasks constituting each batch are randomly generated: the first one is characterized by constant reference signals, the second by piecewise constant references, and the third by piecewise sinusoidal references. Each referenced signal constituting a task is paired with randomly generated initial steady-state conditions $(u_{-1},\, y_0)$.

Examples of the three controllers' tracking performance are shown in Figure 38 - 39 - 40. The mentioned performance index associated with the three described test batches (constant references, piecewise constant references and piecewise sinusoidal references) is included in Table 11. We can see that K_{OPS} outperforms both the VRFT-derived controllers in each category.

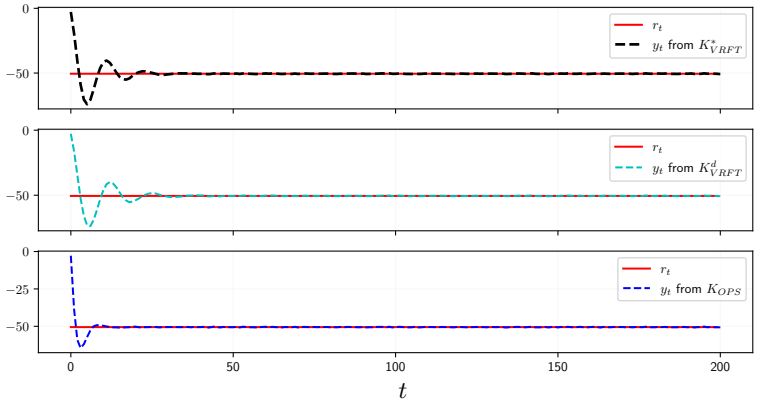

Figure 38: Example 2, OPS vs VRFT comparison, example of tracking of a-priori unknonw constant reference.

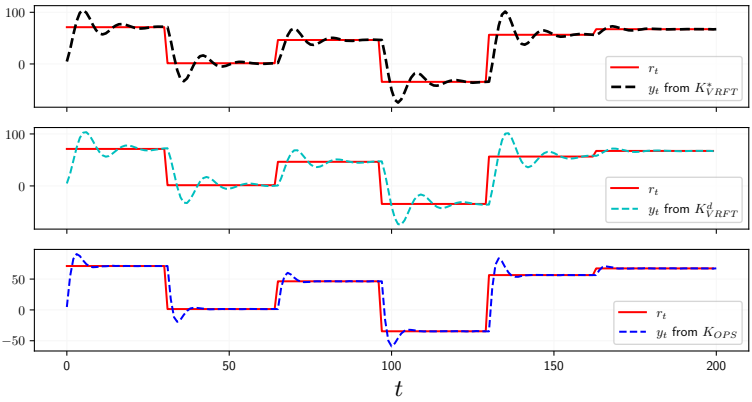

Figure 39: Example 2, OPS vs VRFT comparison, example of tracking of a-priori unknonw piecewise constant reference.

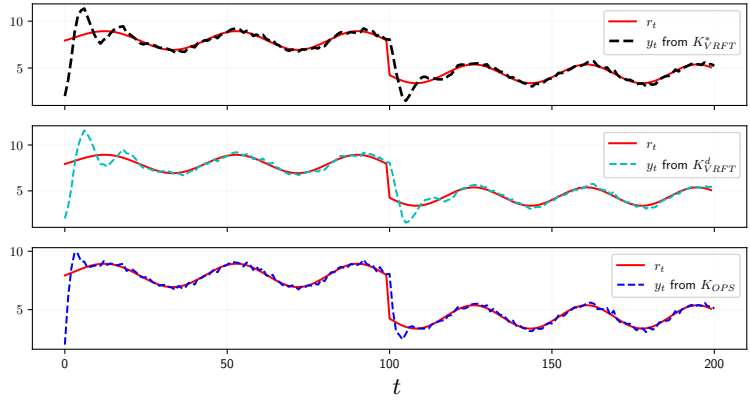

Figure 40: Example 2, OPS vs VRFT comparison, example of tracking of a-priori unknonw piecewise sinusoidal reference.

	constant r_t	piecewise constant r_t	piecewise sinusoidal r_t
K_{OPS}	134748.9	241430.5	1423.6
K^*_{VRFT}	246446.0	525981.4	2650.0
K^d_{VRFT}	342042.4	699744.3	3661.6

Table 11: Example 2, OPS vs VRFT comparison, averaged cost over the described test batches.

The three controllers share the same stucture, the same feedback vector, and are synthesised from the same set of open-loop inputs and outputs, demonstrating that the advantage achieved by the OPS method is derived by its capability of exploiting the information in the data in a more efficient way.

3.4 Example 3 - Continuous Stirred Tank Reactor

In the previous examples the OPS method was employed over LTI systems. First, we demonstrated the method convergence to the optimal policy, in case the policy and local linear model parameterizations are chosen equal to the nominal ones. Then, we showed that, even with policy and local linear models structured differently from the optimal controller and the nominal model (relying on a reduced state representation), the method achieves tracking performance that is close to the optimal one. In this example, instead, our aim is to illustrate the capabilities of the OPS method in synthesizing online and offline controllers, to perform output tracking over a nonlinear system with unknown dynamics.

Table 12: CSTR parameters

symbol	description	value
F	volumetric flowrate	1 m^3/hr
V	volume	1 m^3
ρC_p	volumetric heat capacity A-B	500 kcal/(m^3 C)
H	reaction heat	5960 kcal/kgmol
E	activation energy	11843 kcal/kgmol
R	Bolzmann constant	1.98589 kcal/(kgmol K)
k_0	CSTR coefficient	$9703 * 3600$ 1/hr
u_A	overal heat transfer coefficient	150 kcal/(m^3 C hr)
T^f	feed temperature	298.15 K
C_A^f	feed concentration	10 kg mol/m^3

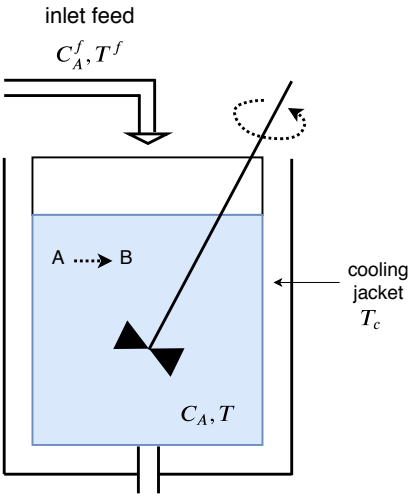

Figure 41: Continuously Stirred Tank Reactor (CSTR).

To this end we consider the Continuous Stirred Tank Reactor (CSTR) problem (cf. Seborg, Edgar, and Mellichamp, 2004). The plant, represented in Figure 41, is characterized by the following dynamics

$$\frac{\partial C_A}{\partial t} = \frac{F}{V}(C_A^f - C_A) - k_0 e^{-\frac{E}{RT}} C_A \tag{3.14a}$$

$$\frac{\partial T}{\partial t} = \frac{F}{V}(T^f - T) + \frac{u_A}{\rho C_p V}(T_c - T) + \frac{H}{\rho C_p} k_0 e^{-\frac{E}{RT}} C_A, \tag{3.14b}$$

and by the parameters in Table 12. We denote as C_A and T the concentration of reactant A and the temperature inside the tank, respectively. The signal T_c is the temperature of the cooling jacket. In the following, we will consider as underlying plant for the learning experiments the numerical system obtained by discretizing the signals with sampling time $T_s = 6$ s.

The control objective is to optimally make the concentration of reactant A track a desired set point taking values in the interval $[2,\ 9]$, by manipulating the temperature of the cooling jacket. We consider the scaled signals

$$y_t^C = \frac{\bar{C}_A^t - c_m}{c_M - c_m} = \frac{C_A^t - c_m}{c_M - c_m} + \frac{d_t}{c_M - c_m}, \qquad u_t = \frac{T_c^t - u_m}{u_M - u_m}, \tag{3.15}$$

as input/output data, where $[c_m,\, c_M] = [5,\, 10]$, $[u_m,\, u_M] = [300,\, 360]$. The quantities $\bar{C}_A^t = C_A^t + d_t$ represent the noisy measurements of C_A^t, considering an unmeasurable Gaussian white noise d_t with standard deviation $\sigma_y = 0.1$.

The stage-cost (3.2) is used, with horizon $L = 10$. The weights $Q_y = Q_y^L = 1$, $R = 0.1$ are chosen, indicating that our goal is to steer y_t^C into tracking $r_t = (\bar{r}_t - c_m)/(c_M - c_m)$. The weight $Q_q = Q_q^L = 0.0001$ over the integral action is used as a regularization term, secundary with respect to the main tracking goal.

In this example we consider two additional states: one is the integral q_t, having known dynamics $q_{t+1} = q_t + (y_t^C - r_t)$. The second one is the scaled temperature inside the tank

$$y_t^T = \frac{\bar{T}_t - t_m}{t_M - t_m}$$

with $[t_m,\, t_M] = [350,\, 400]$ and $\bar{T}_t$ representing the noisy measurements of T_t, subject to unmeasurable white Gaussian noise with standard deviation $\sigma_y = 0.1$. Differently from the integral action q_t, the dynamics of T_t are unknown, hence past realizations of T_t are included in the state x_t and modeled by local linear models, as described in (2.18). To build the feedback vector $n_o = 3$ past measurements of y_t^C and y_T^T and $n_i = 1$ past input u_t are employed, i.e.,

$$x_t = [\, y_t^C,\, y_t^T,\, y_{t-1}^C,\, y_{t-1}^T,\, y_{t-2}^C,\, y_{t-2}^T,\, u_{t-1}\,]'.$$

Adding coefficients of the integral q_t and of the reference r_t, the feedback vector provided to the contollers is composed by 9 elements.

Performance index In both the online and offline settings, the tracking performance of the synthesized policies is evaluated by means of the trajectory cost (2.5), with stage costs

$$\rho(s_t,\, r_t,\, u_t) = \|y_t - r_t\|_{Q_y}^2 + \|\Delta u_t\|_R^2, \quad \rho_L(s_L,\, r_L) = \|y_L - r_L\|_{Q_y^L}^2.$$

To test the capabilities of the OPS method, the policy trained using such algorithm are employed over a batch of tracking tasks. Each task is an a priori unknonw set point signal to be tracked. The average tracking performance over such tasks batch is taken as performance index.

In the following, hence, after synthesizing a policy using the OPS method, a tasks batch is built, composed by N_{tasks} reference signals of length N_L assuming values in $[r_{\min}, r_{\max}]$. Such tracking tasks are executed by the plant, controlled by the OPS-synthesized policy, starting from steady-state conditions $C_A^{ss} = 8.5695$, $T^{ss} = 311.2669$, $T_C^{ss} = 298.15$. Each batch is composed by:

- a subset R_{const} containing N_{const} constant signals $2,\ 2.5,\ 3,\ 3.5,\ \ldots,\ 9$;
- a subset R^1_{pwc} containing N^1_{pwc} random piecewise constant signals, each of them with initial value r_0 sampled according to $\mathcal{U}([\,i,\ i+0.5\,])$ for $i = 2, 2.5, \ldots, 8.5$. The following values r_t are sampled from $\mathcal{U}([\,2,\ 9\,])$;
- a subset R^2_{pwc} containing N^2_{pwc} of random piecewise constant signals, with initial value $r_0 = 8$ close to the initial steady state conditions, while following values r_t sampled from $\mathcal{U}([\,2,\ 9\,])$.

Table 13 contains the parameters characterising such tasks batches.

N_{tasks}	N_{const}	N^1_{pwc}	N^2_{pwc}	$r_{\min}$	$r_{\max}$	N_L
49	15	14	20	2	9	10000

Table 13: Example 3, tasks batch characteristics.

3.4.1 Tuning details

This section includes the tuning choices common to all the scenarios that are going to be presented in the following (namely, synthesis of linear policy offline, synthesis of linear policy online, synthesis of nonlinear policy offline, and synthesis of nonlinear policy online). The choices that differ from one to the other case will be included in the separate sections dedicated to the specific scenarios.

In particular, in all the four mentioned scenarios the mini-batches described in Section 2.2.2.1 are composed by $N_0 = 50$ states x_t as in (2.2), sampled from the states history X, and by $N_r = 7$ reference values. Each of the 7 reference values is sampled in a different sub-interval of the chosen space of tasks $[2,\ 9]$. Specifically each of them is sampled according to one of the uniform random variables $\mathcal{U}([\,i,\ i+1\,])$ for $i = 2, 3, \ldots, 8$. The small perturbation applied to the states sampled from the states history is chosen as $v_n \sim 0.01 \cdot \mathcal{N}(0, 1)$.

The disturbance trajectories are sampled as Gaussian white noise having standard deviation $\sigma_d = 0.1/(c_M - c_m)$. The local linear models Θ_t are fitted by initializing the parameters as $\Theta_0 = \underline{0}_{2\times 8}$ and $P_0 = (1e+5) \cdot I_8$, and recursively updating it by Kalman Filtering, with covariance matrices $Q_k = I_8$ and $R_k = 0.01/(c_M - c_m)$.

3.4.2 CSTR - linear policy parameterization

Initially, to match the control goal we synthesize controllers, parametrized using a linear policy parameterization of the control increment $\Delta u_t = u_t - u_{t-1} = (T_c^t - T_c^{t-1})/(u_M - u_m)$, as in (2.22), i.e.,

$$\pi_K(s_t, r_t) = u_{t-1} + K \begin{bmatrix} s_t \\ r_t \end{bmatrix} = u_{t-1} + K^s s_t + K^r r_t.$$

The synthesis process in this case consists hence in learning a total of 9 linear coefficients. Both in online and offline setting we start the learning from a generical initial guess $K_0 = 0.0001 \cdot \underline{1}_{9\times 1}$.

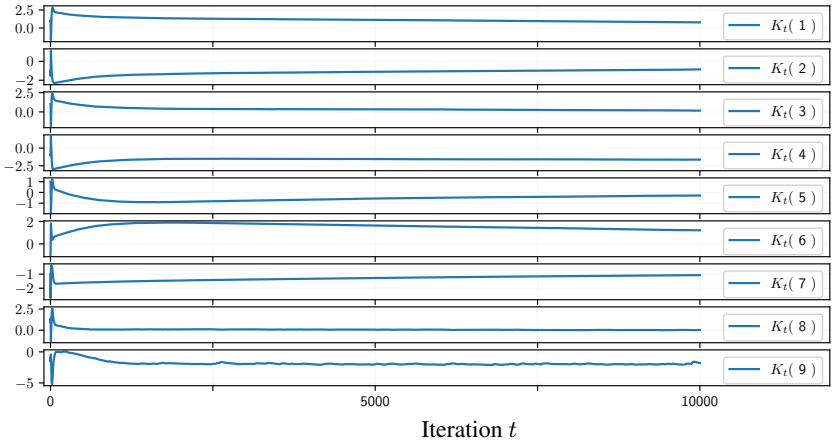

Figure 42: Example 3, offline learning, evolution of the linear policy parameters.

3.4.2.1 Offline setting

To perform the offline learning of the control policy, $N_{\text{data}} = 30000$ input-output couples are collected in open-loop from the plant, excited with a random piece-wise constant input sequence taking values in $[240, 360]$. The collected data are shifted according to (3.15) and stored in the states history X. The associated local linear models are then fitted and stored in the set Θ.

The Optimal Policy Search offline algorithm is executed for $N_{\text{learn}} = 10000$ iterations. The sampled mini-batch is built combining states, reference values and $N_d = 10$ disturbance trajectories (all sampled according to Section 3.4.1), with $N_q = 5$ initial integral action values, sampled uniformly in $[-3, 3]$. For the update of the controller parameters, AMSGrad is employed, with parameters $\alpha = 1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. The evolution of the 9 policy parameters till convergence is shown in Figure 42. As usual, we indicate as K_{OPS} the final set of parameters obtained. Examples of the behavior obtained by K_{OPS} in performing a batch of tracking tasks built according to Table 13 are shown in Figure 43 - 44.

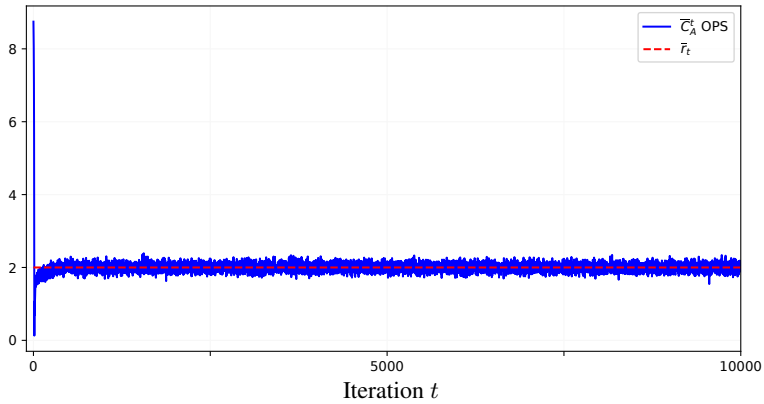

Figure 43: Example 3, offline case, example of tracking of an a priori unknonwn constant reference using K_{OPS}.

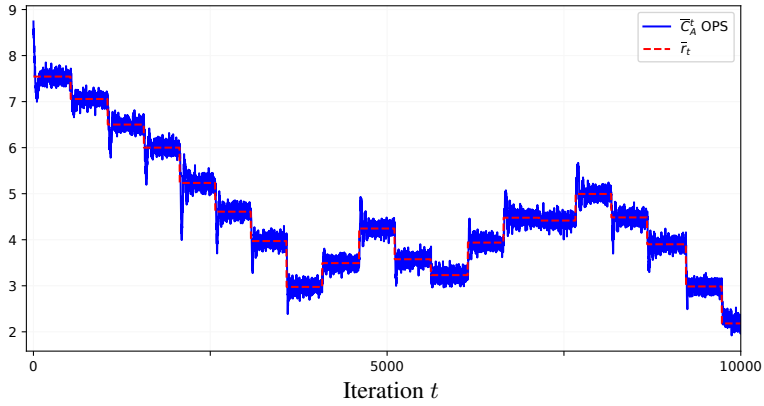

Figure 44: Example 3, offline case, example of tracking of an a priori unknonwn piecewise constant reference using K_{OPS}.

To better analyze the performance achieved by K_{OPS}, we compare it with the behavior of a competitor, synthesized from the same dataset employed for the offline OPS procedure. To this end, Linear Regression is employed on the data used to build the state history for OPS, to fit a linear model approximating the dynamics of y_t^C and y_t^T. Such model is then augmented to obtain the dynamics of $s_t = [\, x_t,\, q_t \,]'$, as it was done in (3.5). The approximated dynamics of s_t are additionally augmented by adding $r_{t+1} = r_t$, coherently with the fact that K_{OPS} is optimized over mini-batches of constant references, as formulated in (3.3). Finally, the backwards Riccati iterations are employed over an horizon of 300 steps, and the associated linear controller, denoted as K_*, is considered as competitor. Although K_* is not the best controller possible for the CSTR plant, it is the one that grants a fair comparison with K_{OPS}, from several points of view. Both controllers do not employ the real dynamics of the plant, but rely only on the information stored in the same set of data. Moreover, both feedback controllers express a linear relationship and explot the same feedback. Finally, both controllers are static.

Table 14 shows the results of the comparison, expressing the number of fail-

	K_{OPS}	K_*
average cost	13.1	54.9
failure occurrence	0	6

Table 14: Example 3, offline case, comparison between K_{OPS} and K_*.

ures encountered by the controllers over the 49 tests and the average cost of the two controllers, considering only the tests over which both controllers do not experience failure (43 over 49 tests). One important observation arising from the results is that, while K_{OPS} is capable of tracking set points taking values in the whole interval $[2,\ 9]$, K_* encounters failure on specific tasks: performing the tests we could notice that the failures of K_* occurred when dealing with set points taking values in $[2,\ 4]$. This is probably caused by the linear model fitted from data and used to synthesize K_*, describing better the dynamics in $[4,\ 9]$, probably linearizable, and not being capable of representing the inherent nonlinearity of the system. On the other hand, the local models employed in the Optimal Policy Search offline synthesis, although linear, seem capable of handling the optimization of nonlinear unknonwn dynamics by being used in combination. Moreover, the average cost in Table 14 shows that, even for tracking tasks with values in $[4,\ 9]$ and hence that can possibly be handled by both controllers, K_{OPS} performs better than K_*.

3.4.2.2 Online setting

The online learning is executed for $N_{\mathrm{learn}} = 50000$ iterations, starting from steady-state conditions $C_A^{ss} = 8.5695$, $T^{ss} = 311.2669$, $T_C^{ss} = 298.15$. At each iteration t a mini-batch is sampled, using $N_d = 1$ disturbance trajectory (sampled as described in Section 3.4.1), $N_0 = 50$ states (the last collected state x_t and 49 states uniformly chosen from the whole history) and $N_q = 20$ initial integral action values (the last value q_t, obtained from the online tracking performance and 19 values sampled using a Gaussian distribuion centered in q_t, with variance 1). For the update of the controller parameters, Adam is employed, with parameters $\alpha = 0.01$, $\beta_1 = 0.8$, $\beta_2 = 0.999$. The heuristic for stability described in Section 2.3.2.1 is implemented in this case.

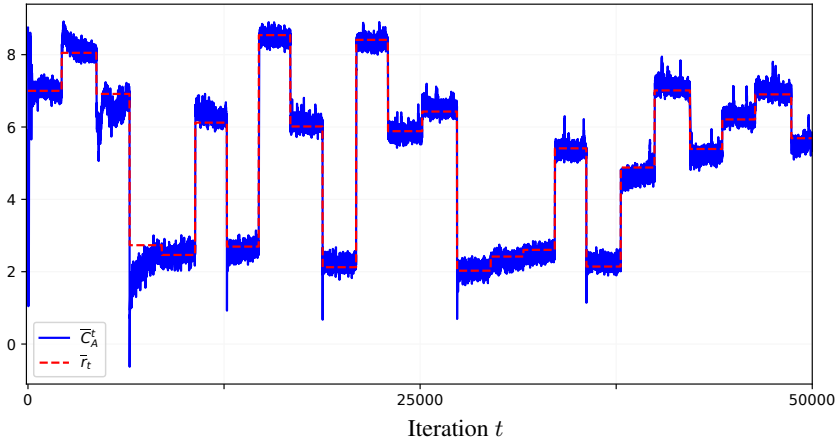

Figure 45: Example 3, online learning, online output tracking of an a-priori unknown exploratory reference signal.

Figure 45 shows the tracking performance achieved by OPS during the learning, while optimizing the policy parameters. The learned policy K_{OPS} is then tested over a batch of tracking tasks built according to Table 13. Examples of the behavior of K_{OPT} while tracking a constant and piecewise constant reference are shown in Figure 46 - 47, respectively. The data collected during the learning phase are employed to compute a competitor K_*, following the same procedure applied in the offline case. The competitor is employed over the same tasks batch used to test K_{OPS}. The average costs achieved by both policies over the tasks batch are included in Table 15.

From such costs and from the number of failures encountered by the competitor policy, it is clear that the data collected during the learning phase do not permit the fitting of a global linear model capable of summarizing the nonlinear dynamics of the plant at a sufficient level. On the other hand K_{OPS}, learned by exploiting the same data, is capable of tracking all the considered tasks and achieving a cost of the same order of the one obtained in the offline case.

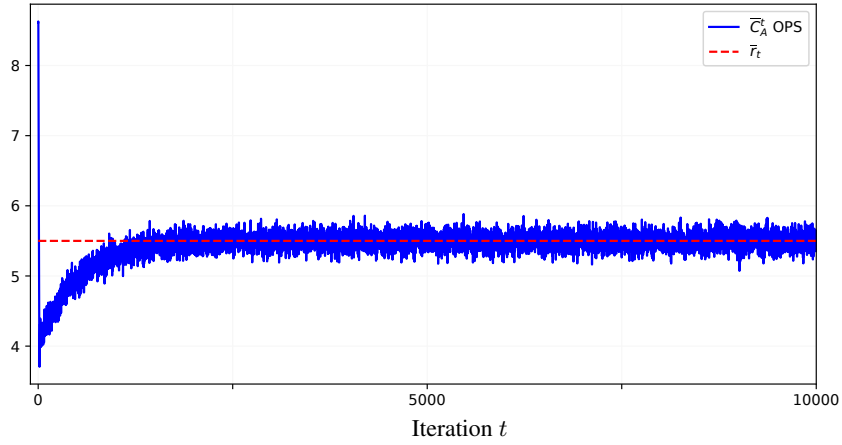

Figure 46: Example 3, online case, example of tracking of an a-priori unknonwn constant reference using K_{OPS}.

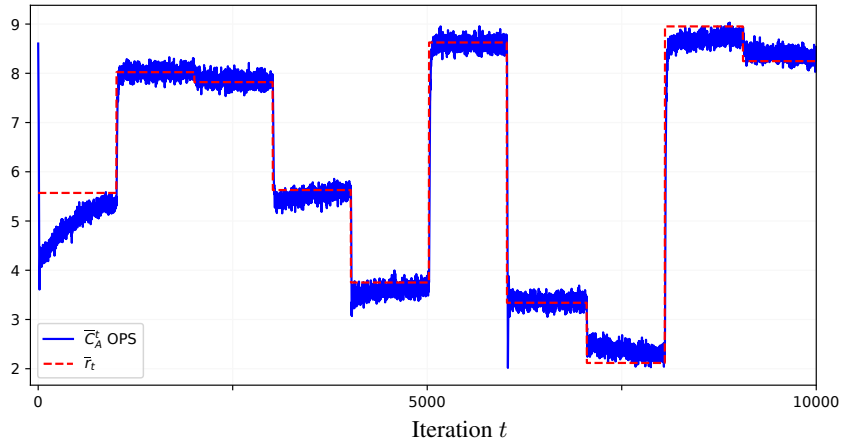

Figure 47: Example 3 online case, example of tracking of an a-priori unknonwn piecewise constant reference using K_{OPS}.

	K_{OPS}	K_*
average cost	28.8	1205.0
failure occurrence	0	33

Table 15: Example 3, online case, comparison between K_{OPS} and K_*.

3.4.3 CSTR - nonlinear policy parameterization

In the last part of this chapter the OPS method is employed to optimize a nonlinear policy parameterization, showing the performance of the method in handling the synthesis of nonlinear controllers. To do so, we substitute the linear parameterization of the input increment Δu_t employed up to now with a neural network. Neural networks are a powerful nonlinear model, composed by a sequence of layers of so called "neurons", each of them providing a composition of linear functions, associated with a matrix of weights W (and eventually a bias vector b), and a nonlinear activation function σ. Hence, each layer i performs the application of a nonlinear function $f_i(x) = \sigma(W_i\, x + b_i)$. A neural network with N layers is structured so that, taken an input value x_0, the layers functions f_i are applied in cascade, and the output of the neural network is the composition $y = f_{N-1}(f_{N-2}(\dots f_0(x_0)))$.

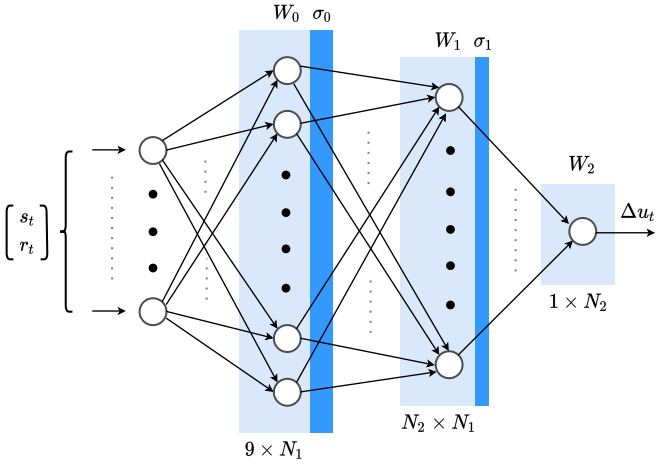

Figure 48: Example 3, nonlinear parameterization, neural architecture.

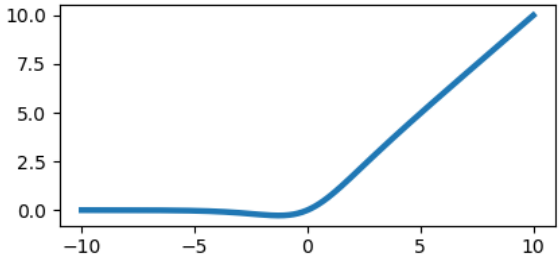

Figure 49: Example 3, nonlinear parameterization, activation function.

The neural parameterization designed to meet our control goal is $u_t = u_{t-1} + \mathcal{NN}(s_t, p_t)$, where $\mathcal{NN}(s_t, p_t)$ is a 3-layered neural network,

$$\mathcal{NN}(s_t, p_t) = W_2 \cdot \sigma_1(W_1 \cdot \sigma_0(W_0 \cdot \left[\begin{smallmatrix} s_t \\ p_t \end{smallmatrix}\right])), \tag{3.16}$$

taking in input the decision variables $(s_t, r_t) \in \mathbb{R}^9$ and restituting the input increment $\Delta u_t \in \mathbb{R}$, as represented in Figure 48. Aside the input and output layers, of dimensions fixed by the characteristics of the problem, the first and the second inner layers contain $N_1 = 15$ and $N_2 = 10$ nodes, respectively.

As activation functions for the inner layers we consider both σ_0 and σ_1 equal to the swish function

$$\sigma(x) = x \cdot \text{sigmoid}(x) = \frac{x}{1 + e^{-x}}, \tag{3.17}$$

a smooth function represented in Figure 49, similar to the most commonly used (but non differentiable) ReLu activation function.

We design the network not to include the bias terms in each layer, hence the set of parameters to be optimized is composed by the weights matrices $W_0 \in \mathbb{R}^{15\times 9}$, $W_1 \in \mathbb{R}^{10\times 15}$, and $W_2 \in \mathbb{R}^{1\times 10}$, i.e., $H = \{W_0, W_1, W_2\}$.

In both offline and online cases, we start the learning from random initial weights $W_i^0 \sim 0.0001 \cdot \mathcal{N}(0, 1)$ for $i = 0, 1, 2$.

3.4.3.1 Offline setting

The offline learning of the neural control policy is performed over the same dataset employed for the offline synthesis of the linear policy parameterization, in Section 3.4.2.

The Optimal Policy Search algorithm is executed for $N_{\text{learn}} = 10000$ iterations. The sampling procedure is performed according to Section 3.4.1, and considering $N_q = 1$ initial integral action values and $N_d = 1$ disturbance trajectories. The integral action values are sampled uniformly in $[-10, 10]$. For the update of the controller parameters, AMSGrad is employed, with parameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.99$. Figure 50 shows the evolution of the neural network weights, each subplot corresponding to the evolution of the Frobenious norm of one of the three neural network layers, i.e., for $W \in \mathbb{R}^{n \times m}$, the value

$$||W||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} w_{i,j}^2}.$$

Examples of the behavior obtained by the resulting neural policy $\mathcal{NN}_{\text{OPS}}$ in performing a batch of tracking tasks built according to Table 13 are shown in Figure 51 - 52.

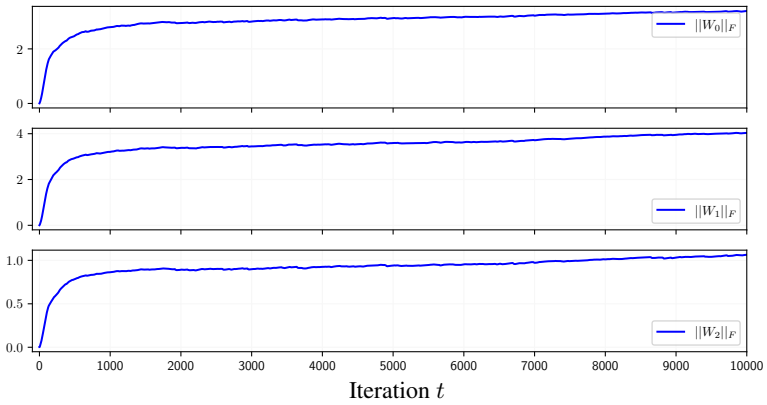

Figure 50: Example 3, offline learning, neural parameterization, evolution of the Frobenious norm of the neural policy parameters.

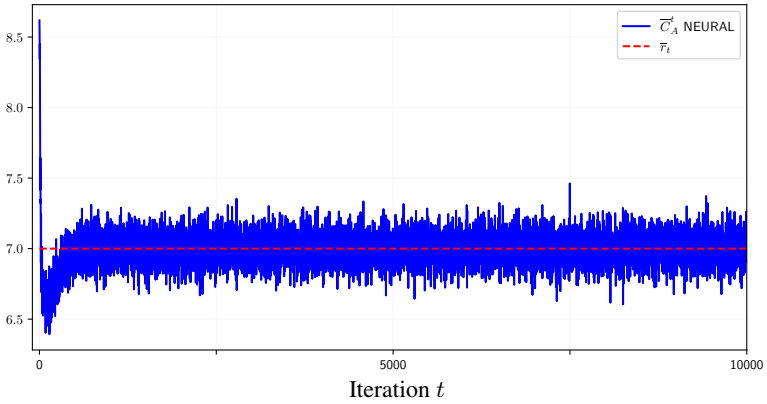

Figure 51: Example 3, offline case, example of tracking of an a priori unknonwn constant reference using $\mathcal{NN}_{\mathrm{OPS}}$.

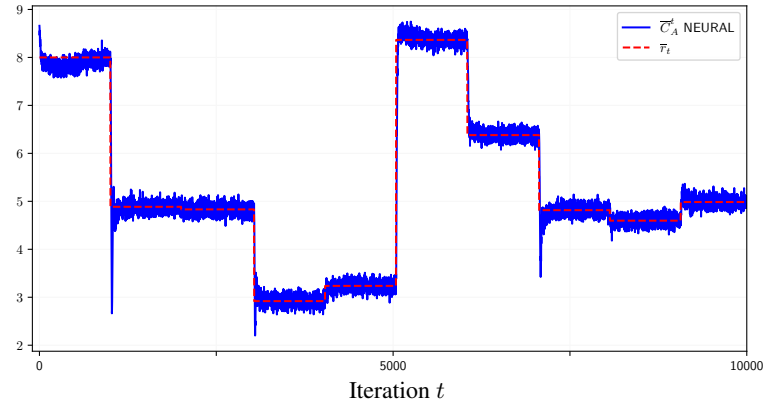

Figure 52: Example 3, offline case, example of tracking of an a priori unknonwn piecewise constant reference using $\mathcal{NN}_{\mathrm{OPS}}$.

references	$\mathcal{NN}_{\rm OPS}$	$K_{\rm OPS}$
whole tasks batch	30.3	38.6
$R_{\rm const}$	12.0	13.5
$R^1_{\rm pwc}$	45.1	57.5
$R^2_{\rm pwc}$	33.6	44.2

Table 16: Example 3, offline case, neural parameterization, averaged tracking cost.

Table 16 includes the average tracking cost achieved by $\mathcal{NN}_{\rm OPS}$ over such batch of tasks, with details on the average tracking cost achieved over constant references $R_{\rm const}$, piece-wise constant references with different initial value $R^1_{\rm pwc}$, and piece-wise constant references with initial value close to the initial state of the plant $R^2_{\rm pwc}$. As a reference, the average tracking cost achieved over the same batch by the linear policy $K_{\rm OPS}$ synthesized offline in Section 3.4.2 is included. We can observe that the neural network performance, in general, achieves lower costs than the linear controller, as expected. The difference among the costs is small over the tracking of constant references, while it becomes more significant considering more difficult piecewise constant tasks.

3.4.3.2 Online setting

The online learning is executed for $N_{\rm learn} = 50000$ iterations, starting from steady-state conditions $C_A^{ss} = 8.5695$, $T^{ss} = 311.2669$, $T_C^{ss} = 298.15$. During the learning, the input selection procedure described in Algorithm 3 is employed, with parameters $T_1 = 5000$, $T_2 = 20000$ $M = 100$ and $\epsilon = 0.05$. We employ as behavioral policy π_b for the input selection procedure the linear feedback controller $K_{\rm OPS}$ synthesized offline in Section 3.4.2.

At each iteration t a mini-batch is sampled, using $N_d = 1$ disturbance trajectory (sampled as described in Section 3.4.1), $N_0 = 50$ states (the last collected state x_t and 49 states uniformly chosen from the whole history) and $N_q = 5$ initial integral action values (sampled uniformly in $[\,-20,\,20\,]$). For the update of the controller parameters, RMSProp is employed, with parameters $\alpha_t = 0.0001$ for $t < 20000$, $\alpha_t = \alpha_{t-1} \cdot (9/10)$ for $t > 20000$, $t\,\%\,5000 = 0$, $\alpha_t = \alpha_{t-1}$ otherwise, $\beta = 0.9999$.

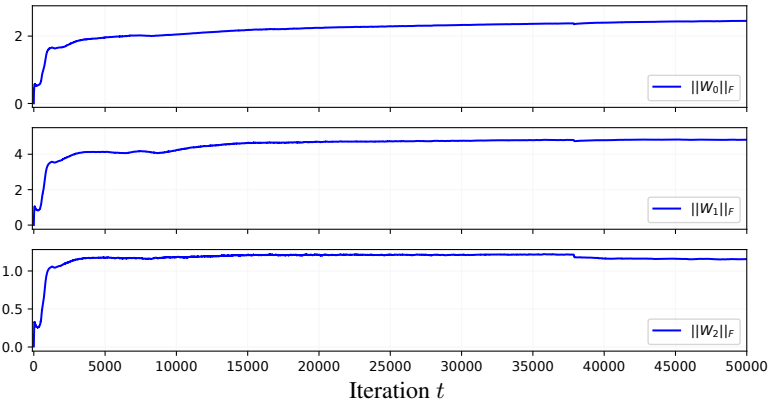

Figure 53: Example 3, online learning, neural parameterization, evolution of the Frobenious norm of the neural policy parameters.

The evolution of the neural network weights W_0, W_1, W_2 is shown in Figure 53 in terms of Frobenious norm of the sequence of matrices generated by the algorithm updates.

Figure 54, instead, shows the tracking performance achieved by OPS during the learning, starting from steady-state conditions $C_A^{ss} = 8.5695$, $T^{ss} = 311.2669$, $T_C^{ss} = 298.15$, in the top plot. The bottom one instead, represents which of the two controllers at disposal ($\pi_{H_t} = \mathcal{NN}_{\text{OPS}}^t$ and $\pi_b = K_{\text{OPS}}$) are employed by the input selection procedure to generate the input to be fed to the plant.

cost	total	off-policy	assisted phase	on-policy
$\mathcal{NN}_{\text{OPS}}$ in learning	116.5	5.8	52.8	57.9
K_{OPS}	124.7	5.8	51.3	67.6

Table 17: Example 3, online learning, neural parameterization, tracking cost.

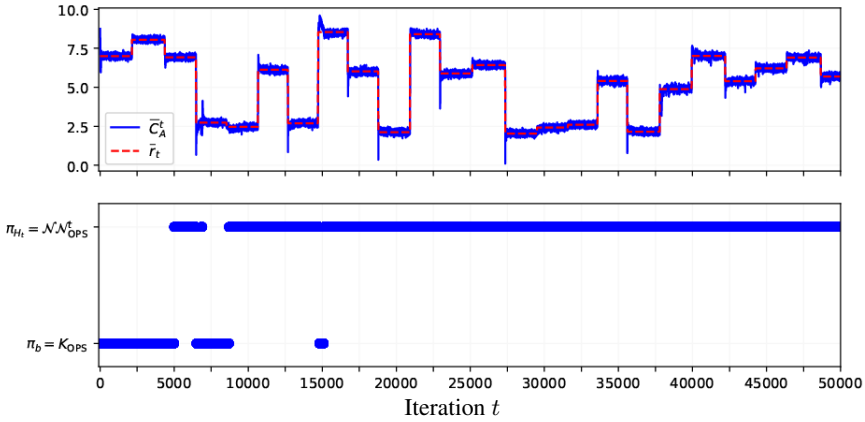

Figure 54: Example 3, online learning, online output tracking of an a-priori unknown exploratory reference signal.

Table 17 presents the online tracking cost achieved by the OPS algorithm together with the assisted control procedure while learning the neural policy $\mathcal{NN}_{\text{OPS}}$. As a comparison, the same table includes the cost obtained by tracking the same exploratory reference employing directly the already trained linear controller K_{OPS}.

To better analyze the obtained results the table includes the detail of the cost achieved in the three phases dictated by the assisted control procedure, that is, the cost associated with the off-policy learning phase (iterations $t \in [\,0,\, T_1-1\,]$), the cost of the assisted control phase (iterations $t \in [\,T_1,\, T_2\,]$), and the cost of the pure on-policy learning phase (from $t = T_2 + 1$ on). We can see that the input selection procedure applied during the assisted phase has a similar (slightly higher) cost than simply using π_b, but that it finally pays off in the sense of learning, permitting to refine the initial policy and obtain a better performance in the long run.

Chapter 4

Learning hybrid controllers from data

This chapter extends the Optimal Policy Search algorithm for the synthesis of hybrid feedback controllers under "black box" hypothesis. As previously done in Chapter 2 for smooth controllers, we introduce the problem formulation and the online and offline algorithms for the hybrid case, detailing how the controllers, the modes and the switching law that characterize a hybrid control law are learned from data. The chapter includes a section of examples, in which such algorithms are applied to learn hybrid controllers for the output-tracking problem.

4.1 The Optimal Switching Policy Search problem

To formulate the Optimal Switching Policy Search problem we consider a plant P as defined in (2.1) and we rely on the structure described in Chapter 2, Section 2.1, working with the system as presented in (2.3), i.e., in form $s_{t+1} = h(s_t, p_t, u_t, d_t)$. The "black box" assumption that makes the dynamics (f, g) and h of the plant unknown at design phase is mantained.

Our aim is to design an algorithm for the synthesis of a hybrid feedback controller from data. A hybrid feedback controller is a control law that is composed by a set of controllers, a set of modes, and a switching law. The switching law

selects, based on the current feedback, one of the modes of the control law, and hence which of the controllers has to be employed to generate the next input u_t to be fed to the plant. A hybrid controller hence allows to diversify the input computation process in different areas of the feedback space. Each of the controllers composing the hybrid law can be specialized, in order to generate the best input for the plant in specific conditions, dictated both by the environment and by the local dynamics of the plant. Not being constrained by a uniform mechanism of input computation, a hybrid controller is hence capable of achieving overall better performance, when the behavior of the plant and its interaction with the environment are varying. Our goal is to synthesize such controllers from data, without any knowledge of the plant dynamics. We also avoid to identify the dynamics, which can be costly and time consuming in case of nonlinear/non-smooth plant behavior. To do so we require to design a method capable not only of optimizing the controllers characterizing the hybrid law, but also to learn the regions of application of such controllers directly from a stream of measured data, with the rationale of optimizing the overall control performance.

As will be discussed in more detail in the following sections, a hybrid controller is not everywhere differentiable over the feedback space, and involves discrete variables, that characterize the switching law. It is hence not possible to directly apply the Optimal Policy Search approach introduced in Chapter 2 for its synthesis. In this chapter we will extend such approach, by designing an alternative data-driven optimization procedure for the discrete components of the control law, and resolving eventual issues of non-differentiability.

A hybrid controller with M modes is representable as a switching policy function, i.e.,

$$\pi(s_t,\, p_t) = \begin{cases} \pi_1(s_t,\, p_t), & \text{if } (s_t,\, p_t) \in R_1, \\ \pi_2(s_t,\, p_t), & \text{if } (s_t,\, p_t) \in R_2, \\ \quad\vdots & \quad\vdots \\ \pi_M(s_t,\, p_t), & \text{if } (s_t,\, p_t) \in R_M, \end{cases} \tag{4.1}$$

where $(\pi_1,\, \ldots,\, \pi_M) \in \mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})^M$ is the set of M controllers characterizing the control law and $(R_1,\, \ldots,\, R_M) \in \mathcal{P}^M$ is the associated set of regions of $\mathbb{R}^{n_s+n_p}$ where each policy π_m should be applied.

As in Chapter 2, we denote as $\mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})$ the set of functions of $n_s + n_p$ real variables taking values in $\mathbb{R}^{n_u}$. The set $\mathcal{P}^M$ instead indicates the set of all the M disjoint subsets of $\mathbb{R}^{n_s+n_p}$ such that their union corresponds to the domain $\mathbb{R}^{n_s+n_p}$ of the controller, i.e.,

$$\mathcal{P}^M = \Big\{ (R_1, \ldots, R_M) \subseteq \mathcal{P}(\mathbb{R}^{n_s+n_p})^M \mid R_1 \cup \ldots \cup R_M = \mathbb{R}^{n_s+n_p}, \\ R_i \cap R_j = \emptyset, \\ \forall\, i, j \in \{1, \ldots, M\},\, i \neq j \Big\}.$$

According to (4.1) the switching law associates the use of the m-th controller π_m with (s_t, p_t) belonging to the m-th region R_m and hence with the m-th mode. According to this representation, a hybrid controller π is then univocally determined by the set $\{\pi_m, R_m\}_{m=1}^M$ of its sub-controllers and regions.

Combining the cost functions (2.6) - (2.7) with the switching policy function definition, we formulate the abstract Optimal Switching Policy Search (OSPS) problem

$$\begin{aligned} &\min\ \mathbb{E}_W[J](\{\pi_m, R_m\}_{m=1}^M), \\ &\text{such that}\ \ \{\pi_m\}_{m=1}^M \in \mathcal{F}(\mathbb{R}^{n_s+n_p}, \mathbb{R}^{n_u})^M, \\ &\qquad\qquad \{R_m\}_{m=1}^M \in \mathcal{P}^M \end{aligned} \tag{4.2}$$

In the same spirit of what was done in Chapter 2, to make problem 4.2 tractable, we parameterize the controller $\{\pi_m, R_m\}_{m=1}^M$. We consider the parameters $H = (K, c)$, where $K = \{K_1, \ldots, K_M\}$ is a set of vectors associated to the M local controllers and c shapes the associated regions. Hence, a switching policy parameterization can be written as

$$\pi_K^c(s_t, p_t) = \begin{cases} \pi_{K_1}(s_t, p_t), & \text{if } (s_t, p_t) \in R_1(c), \\ \pi_{K_2}(s_t, p_t), & \text{if } (s_t, p_t) \in R_2(c), \\ \quad\vdots & \quad\vdots \\ \pi_{K_M}(s_t, p_t), & \text{if } (s_t, p_t) \in R_M(c). \end{cases} \tag{4.3}$$

In (4.3), each subdomain $R_m(c)$ is a function $R_m : \mathbb{R}^{n_c} \to \mathcal{P}(\mathbb{R}^{n_s+n_p})$, mapping the values of the parameters c into a subset of $\mathbb{R}^{n_s+n_p}$.

We consider parameterizations such that each $R_m(c)$ is connected in $\mathbb{R}^{n_s+n_p}$. The parameterized regions $\{ R_m(c) \}_{m=1}^{M}$ have to be designed to belong to $\mathcal{P}^M$. Regarding the M local subcontrollers parameterization, instead, we consider parametric subfunctions π_{K_m} that are differentiable, with continuous partial derivatives, with respect to the parameters $K_m \in \mathbb{R}^{n_{k_m}}$ everywhere in the interior $\mathring{R}_m(c)$ of $R_m(c)$, i.e., for each $m \in \{1, \dots, M\}$ $\pi_{K_m}(s_t, p_t) \in C^1(\mathbb{R}^{n_{k_m}})$ for each $(s_t, p_t) \in \mathring{R}_m(c)$. The value M is here considered as an hyperparameter, chosen at design phase as well as the functional structures $\pi_{K_m}(s, p)$ and $R_m(c)$, and not as a parameter to be learned. Following the construction presented in Section 2.1 to pass from problem 2.8 to 2.10, problem 4.2 can be approximated as

$$
\begin{aligned}
K^*,\, c^* = \arg \quad & \min_{K,\, c} \quad \mathbb{E}\left[\sum_{\ell=0}^{L-1} \rho(s_\ell,\, p_\ell,\, u_\ell) + \rho_L(s_{_L},\, p_{_L}) \right], \qquad (4.4) \\
\text{such that} \quad & u_\ell = \pi_K^c(s_\ell,\, p_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_{\ell+1} = h(s_\ell,\, p_\ell,\, u_\ell,\, d_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_0 \sim S_P \\
& d_\ell \sim D_\ell, \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \dots, L.
\end{aligned}
$$

We indicate this problem as the Optimal Parameterized Switching Policy Search (OPSPS) problem.

Literature on data-driven hybrid control synthesis

Before describing the proposed approach to the OPSPS problem a few literature references on data-driven hybrid control synthesis are mentioned, with particular attention to wether the considered methods provide a data-driven estimation of the switching law or if it requires previous knowledge on the system dynamics.

For instance, in the case of switched linear multiple-input multiple-output (MIMO) systems, the method introduced in (Dai and Sznaier, 2018) synthesizes a robust switching controller from experimental data, without explicitly identifying a model of the open-loop process. Nonetheless, the method requires the knowledge of the model structure, and therefore of the switching law.

In robotics literature, contributions can be found related to the synthesis of switching controllers exploiting experience gathered from process/environment interactions, in the way that is typical of Reinforcement Learning. These methods are mainly related to motion tasks. For instance, in (Grudic, V. R. Kumar, and Ungar, 2003) the control synthesis starts from an existing controller and uses policy gradient techniques to synthesize a switching controller online with improved performance, while (Nagayoshi, Murao, and Tamaki, 2010) uses two control laws, one based on Q-learning and the other on actor-critic, to mimic gross and fine motor skills respectively. However, in both (Grudic, V. R. Kumar, and Ungar, 2003) and (Nagayoshi, Murao, and Tamaki, 2010) the switching law is fixed and known a priori. In (Breschi and Formentin, 2020), instead, a data-driven method is proposed, based on VRFT. Such method estimates a piecewise-affine controller together with the switching law. The method is completely model-free, requiring the tuning at design phase of the reference model for the desired closed-loop behavior, as it is typical for VRFT techniques.

4.2 Extending the OPS method to learn non-smooth controllers

This section describes a new approach to learn both the set of controllers and the modes directly from a stream of inputs and outputs collected from the plant, in an offline or online setting, in the same setup already described in Chapter 2.

Note that it is not possible to directly employ the gradient-based strategy introduced in Chapter 2, Section 2.2 to problem 4.4. Preliminarly, let us analyze this aspect. To this end, let us define a function to represent the switching law associated to parameters c. We define the function $\sigma_c : \mathbb{R}^{n_s+n_p} \to \{1, \dots, M\}$ that assigns to each state and exogenous signal the corresponding region index, i.e., $\sigma_c(s,\, p) = j$ if $(s,\, p) \in R_j(c)$. Using σ_c it is possible to rewrite the policy parameterization (4.3) as $\pi_K^c(s_t,\, p_t) = \pi_{K_{\sigma_c(s_t,\, p_t)}}(s_t,\, p_t)$. From this expression we can observe how the parameterization is a composition of functions of the parameters $K_1, \dots, K_M$ assuming continuous values (the subcontrollers π_{K_m}) with the switching law σ_c, parameterized using c and taking discrete values. Is it then impossible to directly apply the gradient-based numerical optimization methods to learn K and c.

4.2.1 Optimization strategy

In order to extend the Optimal Policy Search method a coordinate descent strategy is designed, based on the idea of alternating the optimization of the local controllers with respect to the current switching law and the optimization of the switching law, given the last updated local controllers. In terms of search in the parameters space, this translates into an iterative procedure such that, starting from an initial guess K_0, c_0, at every iteration $t = 1, \dots, N_{\text{learn}}$ we attempt to solve, in sequence

$$
\begin{aligned}
K^t = \arg \min_{K} \quad & \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, u_\ell) + \rho_L(s_{_L}, p_{_L}) \Big], && \text{(4.5a)} \\
\text{such that} \quad & u_\ell = \pi_K^{c^{t-1}}(s_\ell, p_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_{\ell+1} = h(s_\ell, p_\ell, u_\ell, d_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_0 \sim S_P, \quad d_\ell \sim D_\ell, \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \dots, L.
\end{aligned}
$$

$$
\begin{aligned}
c^t = \arg \min_{c} \quad & \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho(s_\ell, p_\ell, u_\ell) + \rho_L(s_{_L}, p_{_L}) \Big], && \text{(4.5b)} \\
\text{such that} \quad & u_\ell = \pi_{K^t}^{c}(s_\ell, p_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_{\ell+1} = h(s_\ell, p_\ell, u_\ell, d_\ell), \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& s_0 \sim S_P, \quad d_\ell \sim D_\ell, \quad \ell = 0, 1, \dots, L\text{ - }1, \\
& p_\ell \sim P_\ell, \quad \ell = 0, 1, \dots, L.
\end{aligned}
$$

Note that the alteration of the coordinate descent steps in (4.5) can indifferently start with the optimization of the sub-controllers, or with the regions update. The next sections of this chapter are dedicated to describe the Optimal Switching Policy Search method, that implements a data-driven unsupervised learning approach to deal with the optimization of the discrete valued switching law, and an extension of the Optimal Policy Search method to deal with the sub-controller's optimization, following the introduced coordinate descent strategy. The remaining part of this section instead shows why the Optimal Policy Search method cannot directly be applied to tackle problem (4.5a), requiring instead to be adapted.

4.2.1.1 Multi-modal policy cost: differentiability

In this section we discuss the possible issues arising in the application of the OPS method to problem (4.5a). In particular, we analyse the differentiability with respect to K of the approximated cost function exploited for the data-driven gradient approximation by the OPS approach, when combined with a switching policy parameterization.

As already explained, the mentioned approximated cost is obtained substituting the unknown nominal dynamics of the plant with local linear models in specific neighborhoods of states, sampled from the states history. Hence, for a fixed sample $\overline{w} = (\,\overline{s}_0,\, \{\overline{p}_\ell\}_{\ell=0}^{L}, \{\overline{d}_\ell\}_{\ell=0}^{L-1}\,)$, and local model $\overline{\Theta}_0$ associated to the initial state $\overline{s}_0$, we study the differentiability of

$$J_{\overline{w}}^{\overline{c}}(K) = \hat{J}_L(K,\, \overline{c},\, \overline{w}) = \sum_{\ell=0}^{L-1} \rho(s_\ell,\, \overline{p}_\ell,\, \pi_K^{\overline{c}}(s_\ell,\, \overline{p}_\ell)) + \rho_L(s_{_L},\, \overline{p}_{_L}), \quad (4.6)$$

with respect to $K = \{\, K_1,\, \dots\, , K_M\,\}$, given fixed parameters $\overline{c}$.

In the case of a single mode controller ($M = 1$), π_K^c is characterized by $K = \{\, K_1\,\}$ and $R_1(c) = \{\, \mathbb{R}^{n_s+n_p}\,\}$, by definition of $\mathcal{P}^M$. Given that π_{K_1} is assumed to be differentiable (and hence continuous) in K_1 for each $(s_t,\, p_t)$ in $\mathring{R}_1(c) = \mathbb{R}^{n_s+n_p}$, the policy parameterization π_K^c corresponds to the smooth policy parameterization π_H considered in Chapter 2, and problem 4.4 coincides exactly with problem 2.10. The function composition of $\pi_K^c = \pi_{K_1}$ with the cost function of problem 2.10 is then everywhere differentiable with respect to $H = K_1$ and the OPS method can be employed.

When the desired controller has more than one mode ($M > 1$), instead, the goal function in (4.6) is possibly not continuous (and hence not differentiable) with respect to K. Indeed, $J_{\overline{c}}^{\overline{w}}(K)$ is the composition of:

- the cost $J_L^{\overline{w}}(\{s_\ell\}_{\ell=1}^{L}, \{u_\ell\}_{\ell=0}^{L-1}) = \sum_{\ell=0}^{L-1} \rho(s_\ell, \overline{p}_\ell, u_\ell) + \rho_L(s_{_L}, \overline{p}_{_L})$, such that $\{\overline{p}_\ell\}_{\ell=0}^{L}$ and $s_0 = \overline{s}_0$ are specified in $\overline{w}$. This function is continuous and differentiable with respect to the variables $\{s_\ell\}_{\ell=1}^{L}$ and $\{u_\ell\}_{\ell=0}^{L-1}$, by design of ρ and ρ_L.

- The functions $s_\ell^{\overline{w}} : (\mathbb{R}^{n_u})^\ell \longrightarrow SR_\ell(\overline{w}) \subseteq \mathbb{R}^{n_s}$ such that

$$s_\ell^{\overline{w}}(\{u_j\}_{j=0}^{\ell-1}) = A_0^\ell\, \overline{s}_0 + \sum_{j=0}^{\ell-1} A_0^{\ell-1-j}(B_0 u_j + E_0 \overline{p}_j + D_0 \overline{d}_j) \qquad (4.7)$$

  for $\ell = 1, \ldots, L$. The codomain $SR_\ell(\overline{w})$ of $s_\ell^{\overline{w}}$ is the set of states reachable by $(A_0,\ B_0,\ D_0,\ E_0)$ in ℓ steps, based on the sample $\overline{w} = (\,\overline{s}_0,\ \{\overline{p}_\ell\}_{\ell=0}^{L}, \{\overline{d}_\ell\}_{\ell=0}^{L-1})$. The construction of $(A_0,\ B_0,\ E_0,\ D_0)$ from the local model $\overline{\Theta}_0$ associated with $\overline{s}_0$ is described in Appendix A. For each $\ell = 1, \ldots, L,\ s_\ell^{\overline{w}}$ is continuos and differentiable with respect to $\{u_j\}_{j=0}^{\ell-1}$.

- The policy function $u_\ell = \pi_K^{\overline{c}}(s, p)$. Based on the hypotesis on the policy function (4.3), we can grant continuity and differentiability only if $(s,\ p)$ belongs to the interior of $R_m(\overline{c})$ for some m, i.e., $(s,\ p) \in \bigcup_{m=1}^{M} \mathring{R}_m(\overline{c})$.

We recursively define the function $s_\ell^{\overline{w}}(\cdot,\ \overline{c}) : \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \longrightarrow SR_\ell(\overline{w})$ as the composition of (4.7) and $\pi_K^{\overline{c}}(s,\ p)$, i.e.,

$$s_\ell^{\overline{w}}(K, \overline{c}) = s_\ell^{\overline{w}}(\ \{u_j\}_{j=0}^{\ell-1}\ ) \quad \text{with} \quad u_j = \pi_K^{\overline{c}}(s_j^{\overline{w}}(\ \{\,\pi_K^{\overline{c}}(\ldots, \overline{p}_k)\,\}_{k=0}^{j-1}\ ),\ \overline{p}_j).$$

for each $\ell = 1, \ldots, L$. Such composition is granted to be continuous and differentiable in K only if at every step of the composition $\pi_k^{\overline{c}}(s, \overline{p}_j)$ is applied on $s_j^{\overline{w}} \in \bigcup_{m=1}^{M} \mathring{R}_m(\overline{c})$ for each j. Following from this, function (4.6) is granted to be continuous and differentiable in the subset of parameters K such that at every step ℓ of the considered trajectory of lenght L, the state-exogenous signal couple $(s_\ell^{\overline{w}}(K, \overline{c}),\ \overline{p}_\ell)$ lays in the interior of $R_m(\overline{c})$ for some m.

Fixed $\overline{c}$ and $\overline{w}$, hence, the set of parameters K such that (4.6) is granted to be continuous and differentiable is

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \Big\{ K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \Big| \forall\ \ell \in \{1, \ldots, L\} \quad \exists\ m \in \{1, \ldots, M\} \quad (s_\ell^{\overline{w}}(K, \overline{c}),\ \overline{p}_\ell) \in \mathring{R}_m(\overline{c}) \Big\}. \qquad (4.8)$$

The composition of $J_L^{\overline{w}}$, $s_\ell^{\overline{w}}$ and $\pi_K^{\overline{c}}$, restricted to $\mathcal{S}_{\overline{c}}^{\overline{w}}$ guarantees to apply $\pi_K^{\overline{c}}$ only on couples $(s_\ell^{\overline{w}}(K,\ \overline{c}),\ \overline{p}_\ell)$ such that $\pi_K^{\overline{c}}(s_\ell^{\overline{w}}(K,\ \overline{c}),\ \overline{p}_\ell)$ is differentiable.

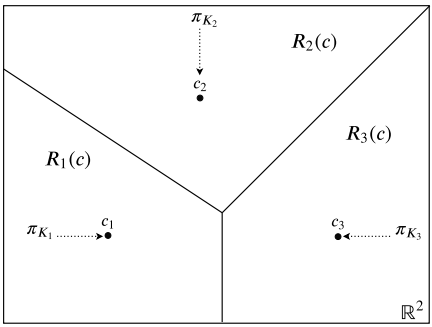

Figure 55: Voronoi partition $R_1(c), R_2(c), R_3(c)$ of $\mathbb{R}^2$, generated by a set of centroids $c = \{c_1, c_2, c_3\}$, associated to the local policies $\pi_{K_1}, \pi_{K_2}, \pi_{K_3}$.

The eventual lack of differentiability outside $\mathcal{S}_c^{\overline{w}}$ would make the naïve use of gradient methods ineffective, possibly requiring to compute the gradients of the cost functions in places where they are not even defined. The Optimal Policy Search approach presented in Chapter 2, dealing with smooth policies and hence smooth costs, is not strightforwardly applicable to tackle problem (4.5a), thus needing to be adapted.

4.3 The OSPS method

The Optimal Policy Search method is extended here to learn switching policies with polyhedral switching laws, that is switching controllers such that the considered sub-controllers are associated with polyhedral areas of application. We choose the following polyhedral parameterization for $R_1(c), \dots, R_M(c) \in \mathcal{P}(\mathbb{R}^{n_s+n_p})$ in (4.3): such regions are designed to be polyhedra obtained from the Voronoi partition of $\mathbb{R}^{n_s+n_p}$ associated with M centroids $c = \{c_1, \dots, c_M\}$, using an opportune distance or semi-distance d over $\mathbb{R}^{n_s+n_p}$, i.e.

$$\begin{aligned} R_j(c) = \{x \;\mid\; & (d(x, c_j) = d(x, c_h) \text{ and } j < h) \text{ or} \\ & d(x, c_j) < d(x, c_h) \quad \forall h \neq j,\ h = 1, .., M \,\} \end{aligned} \tag{4.9}$$

as represented in Figure 55. The parameters of said switching law are then the centroids of the Voronoi partition.

This parameterization respects the assumptions introduced in Section 4.1 on the controllers mode: the regions $R_j(c)$ in (4.9) are two-by-two disjoint, they are a partition of $\mathbb{R}^{n_s+n_p}$, henceforth their union is equal to $\mathbb{R}^{n_s+n_p}$, and they are connected. The policy parameterization (4.3) combined with (4.9) is well defined as a function over $\mathbb{R}^{n_s+n_p}$.

In this context, the parameterized switching law σ_c introduced in Section 4.2.1.1 to assign to each couple (s, p) the associated local controller can be defined as

$$\sigma_c(s,p) = \min\Big\{\arg\min_{i\in\{1,\dots,M\}} d(\begin{bmatrix} s \\ p \end{bmatrix},\ c_i)\Big\}. \tag{4.10}$$

The extended approach, that we denote as the Optimal Switching Policy Search (OSPS) method, is again based on input-output data and can be applied either offline or online, by following the steps in Algorithm 4 and Algorithm 5, respectively.

Both the algorithms at every iteration t call in sequence two sub-algorithms, one for the optimization of the local controllers (addressing problem 4.5a), and one for the update of the centroids (according to problem 4.5b). Such sub-algorithms are presented in Section 4.3.1 (detailing an adapted version of the OPS method taking into account of the discussed differentiability issues) and in Section 4.3.2, respectively.

Algorithm 4 OSPS - Offline setting

Input: Initial guess $H_{-1} = \{K^{-1}, c^{-1}\}$, number N_{learn} of learning steps. State history $X = \{x_0, .., x_{N-n_o}\}$ of P and associated local linear models $\Theta = \{\Theta_0, .., \Theta_{N-n_o}\}$.
Output: Policy parameters H_{OSPS}.

1: **for** $t = 0, \dots, N_{\text{learn}} - 1$ **do**
2: $K^t \leftarrow$ controllers update $(c^{t-1}, K^{t-1}, X, \Theta)$; (cf. Algorithm 6)
3: $c^t \leftarrow$ centroids update $(c^{t-1}, K^t, X, \Theta)$; (cf. Algorithm 8)
4: **end for**
5: $H_{\text{OSPS}} \leftarrow H_{N_{\text{learn}}-1} = \{K^{N_{\text{learn}}-1}, c^{N_{\text{learn}}-1}\}$;
6: **end**.

Algorithm 5 OSPS - Online setting

Input: Initial guess $H_{-1} = \{K^{-1},\, c^{-1}\}$, number N_{learn} of learning steps. Initial state x_0 and input u_0. Initial guess for the local linear model Θ_{-1}. $X(0) = \{x_0\},\ \Theta(-1) = \emptyset$.

Output: Policy parameters H_{OSPS}.

1: **for** $t = 0, \ldots, N_{\text{learn}} - 1$ **do**
2: Apply u_t to the plant and collect y_{t+1} from the plant;
3: Update Θ_t based on $(x_t,\, u_t,\, y_{t+1})$;
4: $\Theta(t) \leftarrow \Theta(t-1) \cup \{\Theta_t\}$;
5: $K^t \leftarrow$ controllers update $(c^{t-1},\, K^{t-1},\, X(t),\, \Theta(t))$; (cf. Algorithm 6)
6: $c^t \leftarrow$ centroids update $(c^{t-1},\, K^t,\, X(t),\, \Theta(t))$; (cf. Algorithm 8)
7: Measure signal p_{t+1};
8: Build x_{t+1} as in (2.2) and store it, i.e., $X(t+1) \leftarrow X(t) \cup \{x_{t+1}\}$;
9: Measure/compute the additional states z_{t+1}; (see Eq. (2.17)-(2.18))
10: Build s_{t+1} from x_{t+1} and z_{t+1};
11: $u_{t+1} \leftarrow \pi^{c^t}_{K^t}(s_{t+1}, p_{t+1})$;
12: **end for**
13: $H_{\text{OSPS}} \leftarrow H_{N_{\text{learn}}-1} = \{K^{N_{\text{learn}}-1},\, c^{N_{\text{learn}}-1}\}$;
14: **end**.

4.3.1 Local controllers update

To update the local controllers parameters $K = \{K_1,\, \ldots,\, K_M\}$ at time t, the steps presented in Algorithm 6 are performed, implementing the sampling phase, the gradient approximation and the update procedure.

Steps $1 - 5$ are dedicated to sampling a minibatch $\{w_h\}_{h=1}^{N_b}$, as described in Section 2.2.2.1. In this phase, each w_h is assigned to one of the regions $\{R_m(c^{t-1}) \mid m = 1, \ldots, M\}$, based on the distance of $(s_0^h,\, r_0^h)$ from the centroids, as defined in (4.9). In this way, we obtain M non overlapping sub-batches R_m^t such that $|R_1^t| + \cdots + |R_M^t| = N_b$. For each $m \in \{1, \ldots, M\}$ the samples in the sub-batch R_m^t are used to compute the gradients of the approximated cost function $\hat{J}_L$ with respect to the vector of parameters K_m in order to update it.

Algorithm 6 Local controllers update at iteration t

Input: Parameters values K^{t-1}, c^{t-1}. States and models history $X(t), \Theta(t)$.
Output: Updated parameters K^t of the local controllers.

for $h = 1, 2, ..., N_b$ **do**
  sample $w_h = (s_0^h, \{p_l^h\}_{l=0}^{L}, \{d_l^h\}_{l=0}^{L-1})$ as described in Section 2.2.2.1;
  compute $j^h \leftarrow \min\left(\arg\min_j d([s_0^h, p_0^h]', c_j^{t-1})\right)$;
  add w_h to $R_{j^h}^t$;
end for
select a random permutation $\xi \in \mathbb{P}(M)$;
$K^{(0)} \leftarrow K^{t-1}$;
for $m = 1, 2, ..., M$ **do**
  $n \leftarrow 0$;
  $\mathcal{G} \leftarrow \underline{0}$;
  for $j = 1, 2, ..., |R_{\xi(m)}^t|$ **do**
    if $K^{(m-1)} \in \mathcal{S}_{c^{t-1}}^{w_j}$ **then**
      compute $\mathcal{G} \leftarrow \mathcal{G} + \nabla_{K_{\xi(m)}} \hat{J}_L(K^{(m-1)}, c^{t-1}, w_j)$;
      $n \leftarrow n + 1$;
    end if
  end for
  if $n > 0$ **then**
    $\hat{\mathcal{G}}_{J_L}(K_{\xi(m)}^{(m\text{-}1)}) \leftarrow \mathcal{G}/n$;
    $K_{\xi(m)}^{(m)} \leftarrow K_{\xi(m)}^{(m-1)} - \alpha_t \; D_t(\hat{\mathcal{G}}_{J_L}(K_{\xi(m)}^{(m\text{-}1)}))$;
  else
    $K_{\xi(m)}^{(m)} \leftarrow K_{\xi(m)}^{(m-1)}$;
  end if
end for
$K^t \leftarrow K^{(M)}$;
end.

The gradients computation and the parameters update are not performed in parallel, but sequentially, the order being established by a permutation ξ, randomly selected at every iteration from the set $\mathbb{P}(M)$ of all the possible permutations of M elements (at line 6 of Algorithm 6). The local linear models are used to compute the data-driven gradient approximations $\nabla_{K_{\xi(m)}} \hat{J}_L$, as described in Section 2.2.2.2. Steps $9 - 16$ are dedicated to compute the approximated gradient of $\hat{J}_L$ with respect to $K_{\xi(m)}$. In particular, at Step 11, the algorithm performs a sub-batch reduction over $R^t_{\xi(m)}$, selecting for the computation of the gradients only the samples $w_j \in R^t_{\xi(m)}$ such that $K^{(m-1)} \in \mathcal{S}^{w_j}_{c^{t-1}}$: such countermeasure, as discussed in Section 4.2.1.1, corresponds to discarding the samples such that the gradient with respect of $K_{\xi(m)}$ of $\hat{J}_L(\cdot, c^{t-1}, w_k)$ is not defined in the current value $K^{(m-1)}$. This is done by computing $\hat{J}_L(K^{(m-1)}, c^{t-1}, w_j)$ and keeping track along the associated trajectory of wether any of the couples $\{s^{w_j}_\ell(K^{(m-1)}, c^{t-1})\}_{\ell=1}^L$ belongs to some boundary $\partial R_m(\overline{c})$ (i.e. if any of such element is equidistant between two different centroids in c^{t-1}).

It is possible that no gradient with respect to a certain $K_{\xi(m)}$ is computed, if for instance the sub-batch $R^t_{\xi(m)}$ is empty (in case the originally sampled mini-batch does not contain elements belonging to the $\xi(m)$-th region) or if no element in $R^t_{\xi(m)}$ belongs to $\mathcal{S}^{w_j}_{c^{t-1}}$: in that case the associated controller $K_{\xi(m)}$ will not be updated at the current iteration t. Otherwise, the computed gradients are employed to perform the parameter updates, as expressed by Steps $17 - 22$.

The next part of this subsection taylors the described sub-batch reduction to the specific case of gradient approximation employing finite differences of precision ϵ: the measures to obtain a well defined gradient approximation are described, considering the possible lack of continuity of (4.6) outside $\mathcal{S}^{w_j}_{c^{t-1}}$.

4.3.1.1 Gradient approximation via finite differences

Probably the simplest way of approximating $\nabla_{K_{\xi(m)}} J^{c^{t-1}}_{w_j}(K^{(m-1)})$, is via finite differences with fixed precision ϵ. To this end, we define for each $m = 1 \ldots, M$ and $i = 1, \ldots, n_{\xi(m)}$

$$K_i^{\xi(m)}(\epsilon) = (K_1^{(m-1)}, \ldots, K_{\xi(m)}^{(m-1)} + \epsilon\, e_i, \ldots, K_M^{(m-1)})$$

as the set of parameter vectors obtained from $K^{(m-1)}$, varying of a quantity ϵ the i-th element of the $\xi(m)$-th vector. For each $i = 1, \ldots, n_{\xi(m)}$ the finite differences approximation of the i-th gradient element is

$$\left[\nabla_{K_{\xi(m)}} J^{c^{t-1}}_{w_j}(K^{(m-1)})\right]_i \approx \frac{J^{c^{t-1}}_{w_j}(K^{\xi(m)}_i(\epsilon)) - J^{c^{t-1}}_{w_j}(K^{(m-1)})}{\epsilon} \tag{4.11}$$

where, as anticipated, we consider the samples $w_j \in R^t_{\xi(m)}$ such that $K^{(m-1)} \in \mathcal{S}^{w_j}_{c^{t-1}}$, to ensure differentiability.

Such approximation, though, could be ill defined for a fixed precision ϵ, even if $K^{(m-1)} \in \mathcal{S}^{w_j}_{c^{t-1}}$. We analyse this aspect for a generic fixed sample $\overline{w} = (\,\overline{s}_0,\, \{\overline{p}_\ell\}_{\ell=0}^{L}, \{\overline{d}_\ell\}_{\ell=0}^{L-1}\,)$, local model $\overline{\Theta}_0$ associated to $\overline{s}_0$, and set of centroids $\overline{c}$. In order to do so, we define

$$\begin{aligned} \Sigma^{\overline{w}}_{\overline{c}}(K) \;=\; & \Big(\, \sigma_{\overline{c}}(s^{\overline{w}}_1,\, \overline{p}_\ell\,),\; \ldots,\; \sigma_{\overline{c}}(s^{\overline{w}}_L,\, \overline{p}_\ell)\,\Big) \\ & \text{such that} \quad s^{\overline{w}}_\ell \;\text{ defined in (4.7) for }\; \ell = 1, \ldots, L, \\ & \qquad\qquad u_\ell = \pi^{\overline{c}}_K(s^{\overline{w}}_\ell,\, \overline{p}_\ell) \;\text{ for }\; \ell = 0, \ldots, L-1. \end{aligned} \tag{4.12}$$

as the sequence of the indices of the regions $R_1(\overline{c}), \ldots, R_M(\overline{c})$ visited by plant P following a trajectory of length L induced by the policy $\pi^{\overline{c}}_K$. We indicate as $\mathcal{S}_{M,L} = \{\chi_1, \ldots, \chi_n\}$ the set of all the possible sequences of indices $\{1, \ldots, M\}$ of length L, having finite cardinality $n = M^L$. We will indicate as $\chi_j(\ell)$ the ℓ-th element of the j-th sequence χ_j. The sequence $\Sigma^{\overline{w}}_{\overline{c}}(K)$ defined in (4.12) is an element of $\mathcal{S}_{M,L}$.

Using the switching law $\sigma_{\overline{c}}$ we can rewrite (4.6) as

$$J^{\overline{w}}_{\overline{c}}(K) = \sum_{\ell=0}^{L-1} \rho(\, s_\ell,\, \overline{p}_\ell,\, \pi_{K_{\sigma_{\overline{c}}(s_\ell, \overline{p}_\ell)}}(s_\ell,\, \overline{p}_\ell)\,) + \rho_L(\, s_L,\, \overline{p}_L\,).$$

Based on this equation and on the definition of $\Sigma^{\overline{w}}_{\overline{c}}(K)$, we have that

$$J^{\overline{w}}_{\overline{c}}(K) = \begin{cases} \sum_{\ell=0}^{L-1} \rho(s_\ell, \overline{p}_\ell, \pi_{K_{\chi_1(\ell)}}(s_\ell, \overline{p}_\ell)) + \rho_L(s_L, \overline{p}_L), & \text{if } K \in R^{\overline{w},\overline{c}}_1, \\ \qquad\vdots \qquad\qquad\vdots \qquad\qquad\vdots \qquad\qquad\vdots & \\ \sum_{\ell=0}^{L-1} \rho(s_\ell, \overline{p}_\ell, \pi_{K_{\chi_n(\ell)}}(s_\ell, \overline{p}_\ell)) + \rho_L(s_L, \overline{p}_L), & \text{if } K \in R^{\overline{w},\overline{c}}_n, \end{cases} \tag{4.13}$$

where for each $j \in \{1, \ldots, n\}$ the region $R_j^{\overline{w},\overline{c}}$ contains the parameters K such that $\pi_K^{\overline{c}}$ visits the regions $\{R_m(\overline{c})\}_m$ following the sequence $\chi_j \in \mathcal{S}_{M,L}$, i.e.,

$$R_j^{\overline{w},\overline{c}} = \{K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \mid \Sigma_{\overline{c}}^{\overline{w}}(K) = \chi_j\}. \tag{4.14}$$

A set $R_j^{\overline{w},\overline{c}}$ is possibly empty for some j, in case no set of parameters K is characterized by a sequence χ_j of regions for the given $\overline{w}$ and $\overline{c}$.

We consider the following result, proved in Appendix B:

Theorem 1. *Given a couple* $(\overline{c}, \overline{w})$*, the set* $\mathcal{S}_{\overline{c}}^{\overline{w}}$ *defined in* (4.8) *is such that*

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \bigcup_{j=1}^{n} \mathring{R}_j^{\overline{w},\overline{c}}$$

where $R_j^{\overline{w},\overline{c}}$ *is defined in* (4.14)*.*

Theorem 1 characterizes for a fixed couple $(\overline{c},\ \overline{w})$ the set $\mathcal{S}_{\overline{c}}^{\overline{w}}$ of the parameters K such that $J_{\overline{c}}^{\overline{w}}(K) = \hat{J}_L(K,\ \overline{c},\ \overline{w})$ is differentiable, and hence continuous. It underlines how such function is continuous on the interior of the regions $\{R_j^{\overline{c},\overline{w}}\}_{j=1}^n$ defined in (4.14), while the boundaries of such regions can correspond to points of discontinuity and hence non differentiability.

Following from Theorem 1, $J_{\overline{c}}^{\overline{w}}(K)$ as it is written in Eq. (4.13) is a piecewise differentiable (piecewise continuous) function. Hence if $K \in \mathcal{S}_{\overline{c}}^{\overline{w}}$ then $J_{\overline{c}}^{\overline{w}}(K)$ is differentiable in K, and it exists a $j \in \{1, \ldots n\}$ such that $K \in \mathring{R}_j^{\overline{w},\overline{c}}$. The gradient of $J_{\overline{c}}^{\overline{w}}$ in K can be approximated successfully using finite differences with fixed precision ϵ only if for each $i = 1, \ldots, n_{k_i}$, the parameters $K_i^{\xi(m)}(\epsilon)$ still belong to the same region $R_j^{\overline{w},\overline{c}}$ of K. In case this is not true, the incremental ratio expressed by (4.11), considering the difference between two values $J_{w_j}^{c^{t-1}}(K_i^{\xi(m)}(\epsilon))$ and $J_{w_j}^{c^{t-1}}(K^{(m-1)})$, can be separated by a discontinuity in the function, not resulting hence close in value to its limit for ϵ that tends to 0, that is to the desired partial derivative $[\nabla_{K_{\xi(m)}} J_{w_j}^{c^{t-1}}(K^{(m-1)})]_i$.

To handle such occurrence, we taylor the sub-batch reduction procedure in case of finite differences approximation, substituting Steps $9 - 18$ of Algorithm 6 with Algorithm 7. At iteration t, following Algorithm 7, for each $m = 1, \ldots, M$ we discard the $w_j \in R_{\xi(m)}^t$ such that $K^{(m\text{-}1)} \in \mathring{R}_k^{w_j, c^{t-1}}$ and

$K_i^{\xi(m)}(\epsilon) \notin R_k^{w_j,c^{t-1}}$ for some i. This simply means to discard $w_j \in R^t_{\xi(m)}$ such that $K^{(m\text{-}1)}$ is associated with a certain sequence $\chi_k \in \mathcal{S}_{M,L}$ of visited regions, that is not the same for the $K_i^{\xi(m)}(\epsilon)$, i.e., $\Sigma^{w_j}_{c^{t-1}}(K^{(m\text{-}1)}) = \chi_k$ and $\Sigma^{w_j}_{c^{t-1}}(K_i^{\xi(m)}(\epsilon)) \neq \chi_k$ for at least one $i \in \{1, \ldots, n_{\xi(m)}\}$.

Algorithm 7 OSPS - Gradient approximation (finite differences) with batch $R^t_{\xi(m)}$ reduction

Input: Last updated centroids c^{t-1}, value of the local controller $K^{(m-1)}$ after iteration $m-1$, finite differences precision ϵ

Output: Approximated gradient $\hat{\mathcal{G}}_{J_L}(K^{(m\text{-}1)}_{\xi(m)})$

$n \leftarrow 0, \quad \mathcal{G} \leftarrow \underline{0}$;
for $j = 1, 2, ..., |R^t_{\xi(m)}|$ **do**
  compute $J^{c^{t-1}}_{w_j}(K^{(m-1)})$ and the associated $\chi_k \leftarrow \Sigma^{w_j}_{c^{t-1}}(K^{(m-1)})$;
  if $K^{(m-1)} \in \mathcal{S}^{w_j}_{c^{t-1}}$ **then**
    $i \leftarrow 1, \quad$ excluded $\leftarrow$ False, $\quad g \leftarrow \underline{0}$;
    while (**not** excluded) **and** $(i \leq n_{\xi(m)})$ **do**
      compute $J^{c^{t-1}}_{w_j}(K_i^{\xi(m)}(\epsilon))$ and the associated $\Sigma^{w_j}_{c^{t-1}}(K_i^{\xi(m)}(\epsilon))$;
      if $\Sigma^{w_j}_{c^{t-1}}(K_i^{\xi(m)}(\epsilon))) \neq \chi_k$ **then**
        excluded $\leftarrow$ True;
      else
        $[\,g\,]_i \leftarrow (\ J^{c^{t-1}}_{w_j}(K_i^{\xi(m)}(\epsilon)) \ - \ J^{c^{t-1}}_{w_j}(K^{(m-1)})\ )/\epsilon$;
      end if
    end while
    if not excluded **then**
      $n \leftarrow n + 1$;
      $\mathcal{G} \leftarrow \mathcal{G} + g$;
    end if
  end if
end for
if $n > 0$ **then**
  $\hat{\mathcal{G}}_{J_L}(K^{(m\text{-}1)}_{\xi(m)}) \leftarrow \mathcal{G}/n$;
end if
end.

A different choice could have been to iteratively reduce ϵ, searching for a precision value $\bar{\epsilon} < \epsilon$ such that $K_i^{\xi(m)}(\bar{\epsilon}) \in R_j^{w_j, c^{t-1}}$. Such value necessarily exists, being K in the inner part of $R_j^{w_j, c^{t-1}}$. This procedure, though, is computationally heavy, and can lead to numerical problems due to computations of finite differences. For this reason, we choose to use only the gradients related to those samples w_j such that $K^{(m-1)}$ and $K_i^{\xi(m)}(\epsilon)$ belong to the same region for $i = 1, \ldots, n_{\xi(m)}$, for a given fixed ϵ.

4.3.2 Switching law update

This section presents an unsupervised learning approach to update the parameters c^{t-1} characterizing problem (4.5b). Following the considered parameterization, such parameters represent the centroids of the polyhedral regions composing the Voronoi partition of the space $\mathbb{R}^{n_s+n_p}$ of the decision variables $(s,\, p)$. The parameterized centroids are employed to define the switching law (4.10) of the parametric hybrid control law.

The centroids are periodically updated every n_K steps, as shown in Algorithm 8. First, we obtain a mini-batch $\{w_k\}_{k=1}^{N_b}$ by performing the sampling procedure (see Section 2.2.2.1). Then, we divide the sampled elements w_k into M subsets $\{W_m\}_{m=1}^{M}$. The assignment of $w_k = (\, s_0^k,\, \{p_\ell^k\}_{\ell=0}^{L},\, \{d_\ell^k\}_{\ell=0}^{L-1}\,)$ to one of the subsets requires computing for $m = 1, \ldots, M$

$$
\begin{aligned}
F(m,\ w_k, K^t,\, c^{t-1}) = \\
= \rho(\, s_0^k,\, p_0^k,\, \pi_{K_m^t}(s_0^k,\, p_0^k)\,) + \hat{J}_{L-1}(\, K^t,\, c^{t-1},\, w_*^m\,).
\end{aligned}
\tag{4.15}
$$

with $w_*^m = (\, s_1^m,\, \{p_\ell^k\}_{\ell=1}^{L},\, \{d_\ell^k\}_{\ell=1}^{L-1}\,)$. Each term $F(m,\ w_k,\, K^t,\, c^{t-1})$ is composed by the stage-cost associated with using the m-th policy $\pi_{K_m^t}$ while being in state $(s_0^k,\, p_0^k)$, plus the approximated cost-to-go associated with the residual trajectory. The expression is approximated using the local linear model fitted in the neighborhood of s_0^k, instead of the unknown system dynamics (2.3). The initial state s_1^m of the cost-to-go is the approximated result of the application of $\pi_{K_m^t}(s_0^k,\, p_0^k)$, while the following ones are obtained employing the sub-controllers in K^t according to the last updated switching law c^{t-1}.

After calculating (4.15) for all $m = 1, \ldots, M$, we assign w_k to the m^*-th subset W_{m^*}, with m^* being the smallest index associated with the most "convenient" subcontroller $\pi_{K^t_{m^*}}$ to be applied in w_k. In this context we consider as more "convenient" the less expensive controller to be applied at the first step, evaluated in terms of the approximated cost F, i.e.,

$$m^* = \min(\arg\min_m F(m,\, w_k,\, K^t,\, c^{t-1})).$$

After the above procedure is executed, we calculate $c^{(1)} = \{c_1^{(1)}, .., c_M^{(1)}\}$, with $c_m^{(1)}$ barycenter of $\{\, (s_0^{(k)}, p_0^{(k)}) \,|\, w_n \in W_m\}$ for each m.

Algorithm 8 Centroids update at iteration t

Input: Parameters $K^t, c^{t-1}, X(t), \Theta(t)$.
Output: Parameters c^t.

if $\text{rem}(t,\ n_K) = 0$ **then**
  $c^{(0)} = c^{t-1}$
  for $i = 1, 2, ..., n_c$ **do**
    $C_m = 0 \quad m = 1, \ldots, M;$
    $\bar{N}_i^m = 0 \quad m = 1, \ldots, M;$
    for $k = 1, 2, ..., N_b$ **do**
      sample $w_k = (s_0^k,\, \{p_l^k,\ d_l^k\}_{l=0}^{L-1})$ as described in Section 2.2.2.1;
      $m^k = \min(\text{argmin}_m F(m, w_k, K^t, c^{(i-1)});$
      $\bar{N}_i^{m^k} = \bar{N}_i^{m^k} + 1;$
      $C_{m^k} = C_{m^k} + \begin{bmatrix} s_0^k \\ p_0^k \end{bmatrix};$
    end for
    for $m = 1, 2, ..., M$ **do**
      $c_m^{(i)} = C_m / \bar{N}_i^m;$
    end for
  end for
  $c^t = (1 - \alpha_t)\, c^{t-1} + \alpha_t\, c^{(n_c)};$
else
  $c^t = c^{t-1};$
end if
end.

The above process is iterated n_c times: at every iteration i the most updated version of the mini-batch barycenters $c^{(i-1)}$ is employed in (4.15) to evaluate the cost-to-go. In this way, we generate a sequence $c^{(0)} = c^{t-1}, c^{(1)}, \dots, c^{(n_c)}$ that refines the barycenters of the regions on the sampled mini-batch. Finally we update the barycenters estimation as

$$c^t = (1 - \alpha_t)\, c^{t-1} + \alpha_t\, c^{(n_c)},$$

with $\alpha_t \in [\,0,\, 1\,]^M$. One possible choice for α_t is to consider

$$N_m^t = N_m^{t-1} + \bar{N}_{n_c}^m,$$
$$\alpha_t^m = \frac{\bar{N}_{n_c}^m}{N_m^t},$$

where N_m^t and $\bar{N}_k^m$ are the cardinalities of the set of states averaged to estimate c_m^t from the beginning up to time t and in the current sampled mini-batch at iteration k of the described procedure, respectively.

4.4 Examples of OSPS for output-tracking

In the following we use the Optimal Switching Policy Search method to synthesize controllers for the tracking of a priori unknonw output reference signals. To do so we employ the definitions of states and exogenous signals (3.1) designed for the output tracking problem in Section 3.1, with the sole difference of a delay in the integral action dynamics, that will be defined as $q_{t+1} = q_t + (y_{t+1} - r_t)$. The same delay is added as well in the considered stage costs at instant t, i.e.,

$$\rho(s_{t+1},\, r_t,\, u_t) = \|y_{t+1} - r_t\|^2_{Q_y} + \|u_t\|^2_R + \|q_{t+1}\|^2_{Q_q}. \tag{4.16}$$

Considering that we work on strictly causal plants, the effect of the input u_t, decided at instant t as a reaction with respect to the feedback s_t, r_t, is perceived at output level on y_{t+1} and q_{t+1}. Hence, the stage costs (4.16) represent the cost of the decision u_t made at instant t, weighted by a matrix $R = R^{'} \succ 0$ and the cost of the direct effects of such decision in comparison with the requested task (the reference r_t that was indicated as signal to be tracked by the feedback vector), weighted by $Q_y = Q_y^{'} \succeq 0$, $Q_q = Q_q^{'} \succeq 0$. No terminal cost is considered, i.e., $Q_y^L = Q_q^L = 0$.

The results of the OSPS method on two numerical examples are shown. In Section 4.4.1 we consider a piecewise LTI plant, while Section 4.4.2 analyzes a nonlinear example. In both cases we test the method in an offline and online setting. Together with the performance of the OSPS method, alternating $n_K = 10$ steps of gain update with a step containing $n_c = 10$ iterations of centroid optimization, we present as a comparison the behavior of the method in case we just optimize the gains, maintaining the centroids unaltered ($n_c = 0, n_K = N_{learn}+1$). Both cases (fixed and optimized centroids) are compared to learning a single controller, synthesized using the OPS method described in Section 2.3 (corresponding to setting $M = 1$), either on the same dataset, if in the offline setting, or while performing the same online learning task.

Performance index

In both the online and offline settings, the tracking performance of the synthesized policies is evaluated considering as performance index the trajectory cost (2.5), considering as stage costs $\rho(s_{t+1}, r_t, u_t) = \|y_{t+1} - r_t\|^2_{Q_y} + \|u_t\|^2_R$ or $\rho(s_{t+1}, r_t, \Delta u_t) = \|y_{t+1} - r_t\|^2_{Q_y} + \|\Delta u_t\|^2_R$, depending on the control variable of choice.

4.4.1 Example 1 - piecewise LTI system

Let the plant P in (2.3) be the (unknown) SISO piecewise LTI system

$$x_{t+1} = 0.8\, A_{\alpha_t}\, x_t + \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right] u_t, \qquad y_t = \left[\begin{smallmatrix} 0 & 1 \end{smallmatrix}\right] x_t \tag{4.17}$$

with

$$A_{\alpha(t)} = \begin{bmatrix} \cos\alpha_t & -\sin\alpha_t \\ \sin\alpha_t & \cos\alpha_t \end{bmatrix}, \quad \text{and} \quad \alpha_t = \begin{cases} \frac{\pi}{3} & \text{if } x_{t_1} > 0 \\ -\frac{\pi}{3} & \text{otherwise} \end{cases}$$

as in (Bemporad and Morari, 1999, Example 4.1).

We assume that there is no disturbance affecting the output measurements of P, and we measure only the one dimensional output y_t, consisting in the second component x_{t_2} of x_t, while we do not measure x_{t_1}.

The chosen feedback state s_t is built employing $n_o = 2$ past output $n_i = 1$ past input, and the integral formula q_t, i.e. $s_t \in \mathbb{R}^4$.

We consider as control variable the action increment $\Delta u_t = u_t - u_{t-1}$ and the employed stage cost (4.16) is

$$\rho(s_{t+1},\, r_t,\, \Delta u_t) = \|y_{t+1} - r_t\|^2_{Q_y} + \|\Delta u_t\|^2_R + \|q_{t+1}\|^2_{Q_q},$$

with stage-cost weights $Q_y = 1$, $R = Q_q = 0.01$. The optimization horizon is set to $L = 10$.

We design the sub-controllers characterizing the hybrid parameterization to be linear, i.e., $\Delta u_t = K_m \left[\begin{smallmatrix} s_t \\ r_t \end{smallmatrix}\right]$, with $\sigma_c(s_t,\, r_t) = m$, where $K_m \in \mathbb{R}^5$ for $m = 1, \ldots, M$. For the centroids optimization, it is noticeable that in this plant the switching law is based on the first state x_{t_1}, a piece of information that we assume not to know.

We consider regions in $\mathbb{R}^{n_s+n_p} = \mathbb{R}^5$ defined using the semidistance $d(x, y) = \|x_1 - y_1\|_2$. Based on the given definition of s_t, this means that we base the controller switching law on the output $y_t = x_{t_2}$ of the plant P.

Tuning details

The sampling parameters are chosen as follows: reference signals are sampled uniformly in the interval $[\, r_{\min},\, r_{\max}\,] = [\, -10,\, 10\,]$ while the integral action values are generated by a Gaussian white random variable, having standard deviation $\sigma_q = 10$. The initial states are sampled uniformly from the states history and perturbed by a Gaussian white signal, having standard deviation $\sigma_v = 0.01$. The local linear models (2.12) are designed neglecting the dependence from the disturbances d_t (i.e., $n_d = 1$, $\sigma_d = 0$), by Kalman Filtering with matrices $Q_k = 10 \cdot I_4$, $R_k = 0.01$, initialized as $\Theta_0 = \underline{0}_{1\times 4}$ and $P_0 = (1e + 5) \cdot I_4$.

We synthesize a switching controller for reference tracking having $M = 2$ modes. The sub-controllers parameters are initialized with K_1^0 and K_2^0 randomly generated from a normally distributed vector with mean zero and standard deviation 0.001. The initial centroids c_0 are sampled uniformly in $[\, r_{\min},\, r_{\max}\,]$. We update the gains using the AMSGrad algorithm (Reddi, Kale, and S. Kumar, 2019), with $\alpha = 0.1$, $\beta_1 = 0.5$, and $\beta_2 = 0.6$.

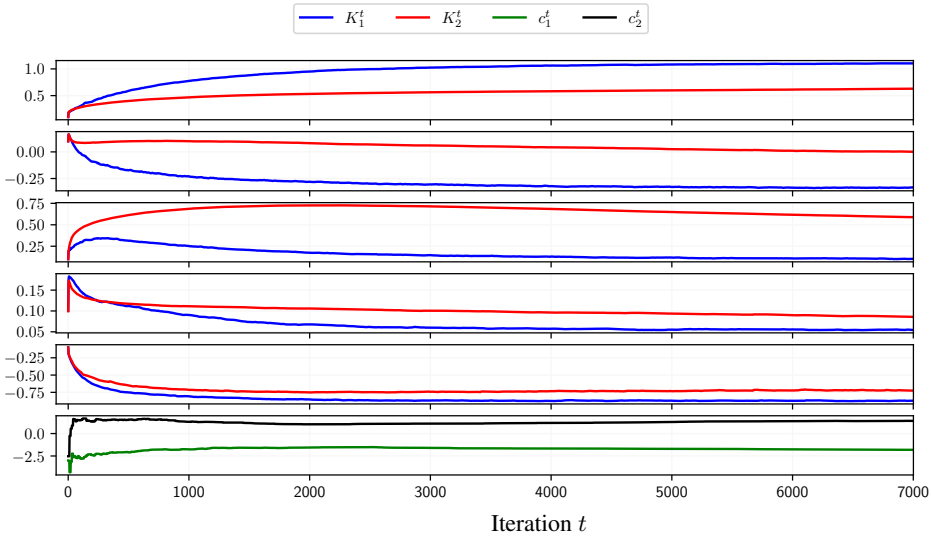

Figure 56: Example 1, offline learning, convergence of the switching policy parameters: K_1^t, c_1^t (blue and green lines) and K_2^t, c_2^t (red and black lines).

4.4.1.1 Offline setting

In case of offline learning, the learning procedure is executed for $N_{\text{learn}} = 7000$ iterations over a dataset of cardinality $N_{\text{data}} = 5000$ collected in open-loop from the plant. At every iteration $n_s = 50$ states are used, together with $n_r = 10$ references and $n_q = 5$ integral action values, to form the mini-batches necessary for the sub-controllers optimization. The mini-batch necessary for the centroids optimization is sampled analogously. Figure 56 shows the evolution of the policy parameters $\{K_1^t,\ c_1^t\}$ during the learning phase, till convergence to a policy that we indicate as $\{K_{\text{OPS}}^{\text{off}},\ c_{\text{OPS}}^{\text{off}}\}$.

Table 18: Example 1, offline learning, tracking cost.

	centroids	task A	task B
$M = 1$		11679.5	3065.0
$M = 2$	fixed	179.5	80.4
$M = 2$	optimized	155.8	71.9

The behavior of $\{K_{\mathrm{OPS}}^{\mathrm{off}}, c_{\mathrm{OPS}}^{\mathrm{off}}\}$ while performing two different tracking tasks (indicated as task A and task B) is shown in Figure 57. The cost of both tasks is compared in Table 18 with the ones achieved by a mono-modal control law, and by a bi-modal controller synthesized with fixed centroids at $[-4,\ 0.5]$.

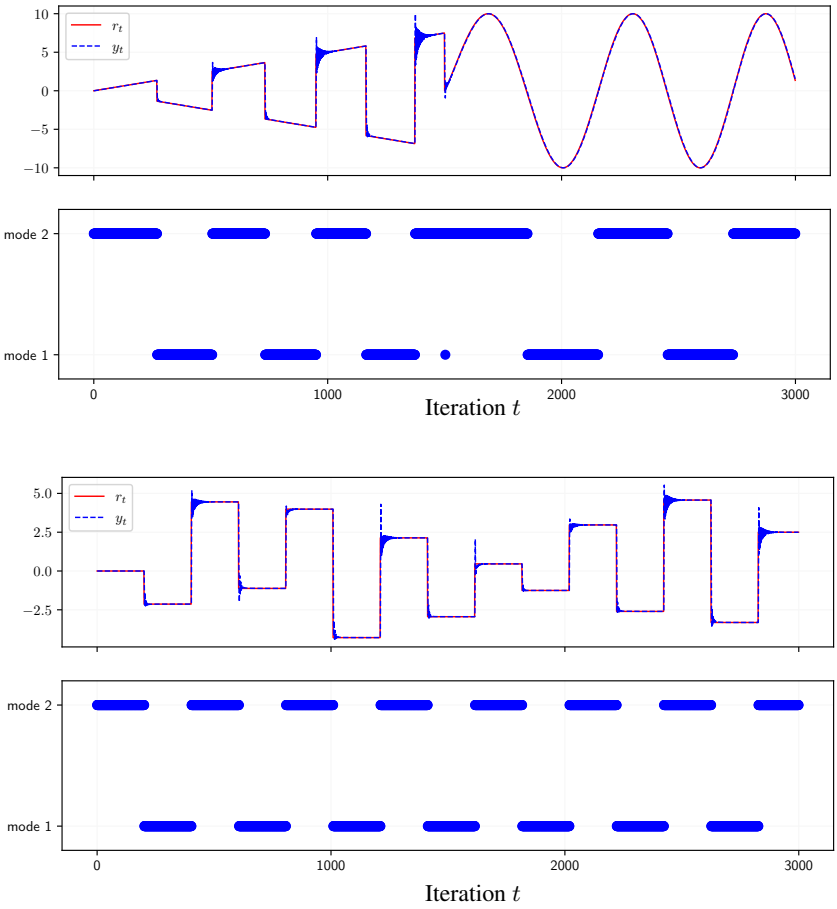

Figure 57: Example 1, offline case, tracking of a priori unknown reference signals: task A and associated modes (upper two plots); task B and associated modes (lower two plots).

Table 19: Example 1, offline learning, averaged cost over 1000 tracking tasks.

	$M = 1$	$M = 2$	$M = 2$
centroids	//	fixed	optimized
averaged cost	2365.4	91.4	80.8

It is observable that this policy is more competitive on the two tasks at hand than the mono-modal controller, and than the bi-modal controller with fixed centroids. To investigate the capabilities of the learned policy more in detail, we tested it, together with the competitors, over a batch of 1000 randomly generated piecewise constant reference signals. The averaged tracking cost obtained from this test mirrors the results obtained over tasks A and B, as seen in Table 19.

4.4.1.2 Online setting

In the online case the learning procedure is executed while performing a tracking task for $N_{\text{learn}} = 3000$ steps, based on the piecewise constant exploration reference shown in Figure 58. At every iteration $n_s = 20$ states, $n_r = 2$ refer-

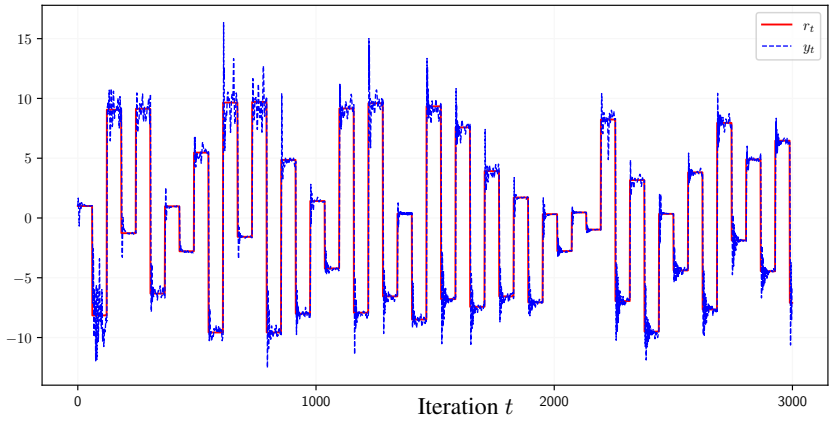

Figure 58: Example 1, online learning, online output tracking of an a-priori unknown exploratory reference signal.

ences and $n_q = 5$ integral actions are sampled and used to build the mini-batches used for the parameters optimization. Such mini-batches are then augmented by adding an element $[\, x_t,\, q_t,\, r_t \,]$ containing the current state, integral action and reference. Figure 59 shows the tracking performance of the synthesized policy $\{\, K_{\mathrm{OPS}}^{\mathrm{on}},\, c_{\mathrm{OPS}}^{\mathrm{on}} \,\}$ on the tasks A and B analyzed in the offline case.

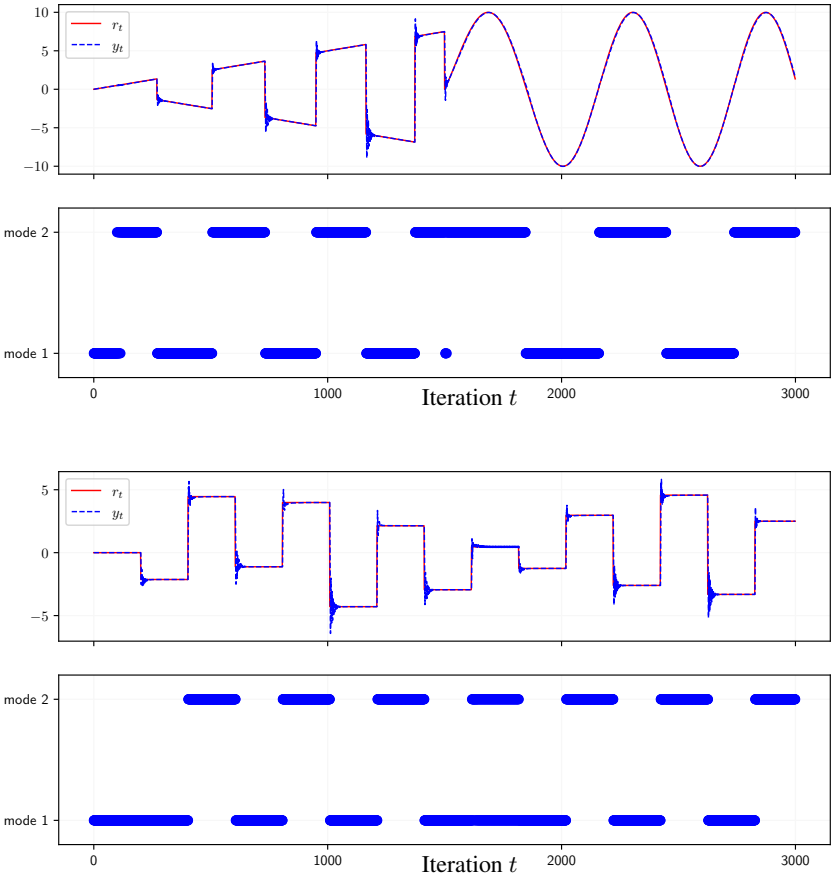

Figure 59: Example 1, online case, tracking of a priori unknown references and associated modes: task A (upper two plots); task B (lower two plots).

Table 20: Example 1, online case, tracking cost.

	centroids	online	validation	
			task A	task B
$M = 1$		4694.5	321.9	271.8
$M = 2$	fixed	3924.5	280.5	230.4
$M = 2$	optimized	3046.4	193.1	169.7

The costs over task A and B can be found in Table 20, together with the online tracking cost, associated with the performance achieved in Figure 58, while learning. Comparing Table 18 and Table 20, we observe that all the switching controllers synthesized offline perform better on tasks A and B than their online counterparts. This could be attributed in part to the higher number of offline learning steps, but mostly to the higher amount of information stored in the offline dataset, compared to the data collected during the online learning. This explaination is supported by the fact that the single-mode controller obtained offline is the most expensive of all, with a cost that is two orders of magnitude higher than the other controllers trained on the same dataset, even though it was synthesized with the same number of learning steps. Such large difference is not observed instead in the online setting. This reinforces the idea that the increased amount of information provided by the offline data helps the switching controllers to specialize, while disturbing a single-mode controller.

The averaged cost obtained applying $\{ K_{\mathrm{OPS}}^{\mathrm{on}}, c_{\mathrm{OPS}}^{\mathrm{on}} \}$ over 1000 randomly generated piecewise constant tracking tasks, included in Table 21, confirms again the considerations on the performance of the three controllers.

	$M = 1$	$M = 2$	$M = 2$
centroids	//	fixed	optimized
averaged cost	228.1	196.0	138.5

Table 21: Example 1, online case, averaged cost over 1000 tracking tasks.

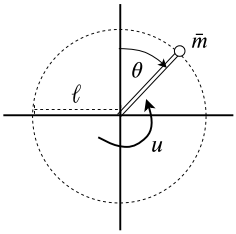

Figure 60: Inverted pendulum.

4.4.2 Example 2 - Inverted pendulum

To test our approach on a simple nonlinear system, we consider the inverted pendulum depicted in Figure 60, consisting of a mass $\bar{m} = 1$ kg rotating by an angle θ at a fixed distance $\ell = 0.5$ m from the central joint, subject to earth gravity $g = 9.81$ m/s^2 and experiencing a viscous friction governed by the viscosity coefficient $\beta = 0.5$ Nms. The physical model of the inverted pendulum dynamics, when subject to the action of the torque u, is the nonlinear ordinary differential equation (ODE)

$$\ell^2 \bar{m}\ddot{\theta} = \bar{m}\, g\, \ell\, \sin\theta - \beta\dot{\theta} + u.$$

We simulate the inverted pendulum using the ODE solver `ode45` from MATLAB ODE Toolbox with sampling time $T_s = 0.05$ seconds. The control goal consists into steering the angular position θ_t of the mass $\bar{m}$ at instant t, following an a priori unknonwn set point r_t.

Assuming not to know the dynamics of the system, we consider as output the angular position θ_t, available through measurements subject to Gaussian white noise with standard deviation 0.01. Moreover, we consider two additional states: one is the integral q_t, having known dynamics $q_{t+1} = q_t + (\theta_{t+1} - r_t)$. The second one is the angular velocity $\dot{\theta}_t$ with unknown dynamics, but measurable and subject to white Gaussian noise with standard deviation 0.01. Past realizations of $\dot{\theta}_t$ are included in the state x_t and modeled by local models, as described in (2.18). Using $n_o = 1$ past masurement of θ_t, $\dot{\theta}_t$, and u_t together with q_t, we build the state representation $s_t \in \mathbb{R}^4$. Considering the stage-costs (4.16), the associated weights are chosen as $Q_y = 1$, $R = Q_q = 0.01$ and the cost horizon is set to $L = 10$.

The torque is parameterized with a switching policy: the sub-controllers are designed as affine policy parameterizations, i.e.,

$$u_t = K_m \begin{bmatrix} s_t \\ r_t \\ 1 \end{bmatrix} = K_m^s \, s_t \, + \, K_m^r \, r_t \, + \, K_m^{\text{bias}} \quad \text{if} \quad \sigma_c(s_t, \, r_t) = m,$$

where $K_m \in \mathbb{R}^6$ for $m = 1, \ldots, M$.

Tuning details The local models are designed to be affine in this case, i.e.,

$$y_{t+1} = \Theta_t \begin{bmatrix} x_t \\ u_t \\ 1 \end{bmatrix} + d_t = \Theta_t^x \, x_t \, + \, \Theta_t^u \, u_t \, + \, \Theta_t^{\text{bias}} \, + \, d_t.$$

At every iteration Θ_t is updated by Kalman Filtering. $\Theta_0 = \underline{0}_{1\times 4}$ and $P_0 = (1e+5) \cdot I_4$ are chosen as initial guesses for the kalman Filter. The sampling parameters are chosen as follows: $N_r = 5$ reference signals are sampled uniformly in the interval $[\, -\pi, \, \pi \,]$ while $N_q = 2$ integral action values are generated by a Gaussian white random variable, having standard deviation $\sigma_q = 10$. A set of N_0 initial states are sampled from the states history and perturbed by a Gaussian white signal, having standard deviation $\sigma_v = 0.1$. N_0 is chosen as $N_0 = 50$ in the offline case, and $N_0 = 20$ in the online one. The dependence from the disturbances d_t is neglected in the sampling of the mini-batches for the parameters optimization (i.e., $n_d = 1$, $\sigma_d = 0$). The sub-controllers parameters are initialized with vectors K_m^0 whose elements are uniformly sampled from $[\, -0.01, \, 0.01 \,]$. We update the gains using the AMSGrad algorithm (Reddi, Kale, and S. Kumar, 2019), with $\alpha = 0.1$, $\beta_2 = 0.6$ and $\beta_1 = 0.8$ in the offline case, while $\beta_1 = 0.5$ in the online one.

4.4.2.1 Offline setting

We start the analysis of the OSPS method in the offline setting by synthesizing a switching policy with $M = 3$ control modes. We perform $N_{\text{learn}} = 7000$ iterations of the proposed method, using a dataset of cardinality $N_{\text{data}} = 5900$, collected in open-loop from the plant.

Local affine models are fitted tuning the Kalman Filter with covariance matrices $Q_k = 10 \cdot I_4$, $R_k = 0.01$. The centroids optimization is realized with

centroids	fixed	optimized
$M = 1$	1097.62	//
$M = 2$	988.18	966.80
$M = 3$	1003.88	978.22

Table 22: Example 2, offline case, tracking cost.

respect to the semidistance

$$d(x, y) = \left\| \begin{bmatrix} \cos(x_1) \\ \sin(x_1) \end{bmatrix} - \begin{bmatrix} \cos(y_1) \\ \sin(y_1) \end{bmatrix} \right\|_2,$$

i.e., we define the switching law as a function of the angular position θ_t. The centroids are initialized as $c_0 = [\,(-\pi + 0.1),\ 0,\ (\pi - 0.1)\,]$. The same values are used as constant centroids in the fixed-regions case. The tracking costs on a piecewise constant tracking task assuming values in $[\,-\pi,\ \pi\,]$ is presented in Table 22.

Both the switching controllers with $M = 3$ modes outperform the single controller; in particular, the policy $\{\, K^{\text{off}}_{(3)},\ c^{\text{off}}_{(3)} \,\}$, obtained by the procedure with centroids optimization, results to be the best of the three, showing the importance of learning the partition. The optimized centroids $c^{\text{off}}_{(3)} = [\,0.648,\ 0.267,\ 0.647\,]$ obtained through the learning procedure indicate that the algorithm, even if programmed to divide the domain is $M = 3$ regions, is generating two large regions $R_1(c)$ and $R_2(c)$, while the remaining region $R_3(c)$ results to be comparatively small. These results might suggest that the optimal choice for the parameter M is 2, and is individuating the boundary between the two regions approximatively at 0.457. This result is coherent with the physics of the problem, considering that the plant has two equilibrium points. Following these results we synthesize the controller with $M = 2$ modes, using the same dataset and the same number of iterations N_{learn}. The centroids are initialized as $c_0 = [\,0,\ \pi\,]$ and again the same values are used as centroids in the fixed-centroid case. As anticipated and visible in Table 22, the cost of the bi-modal policy $\{\, K^{\text{off}}_{(2)},\ c^{\text{off}}_{(2)} \,\}$ decreases. In particular, the best cost is achieved in the case of optimized centroids, where the final centroids are $c^{\text{off}}_{(2)} = [\,-0.201,\ 1.157\,]$.

It can be noticed that the boundary between the two regions generated by the centroids $c^{\text{off}}_{(2)}$ is 0.478, and hence close to the one suggested by the $M = 3$ case.

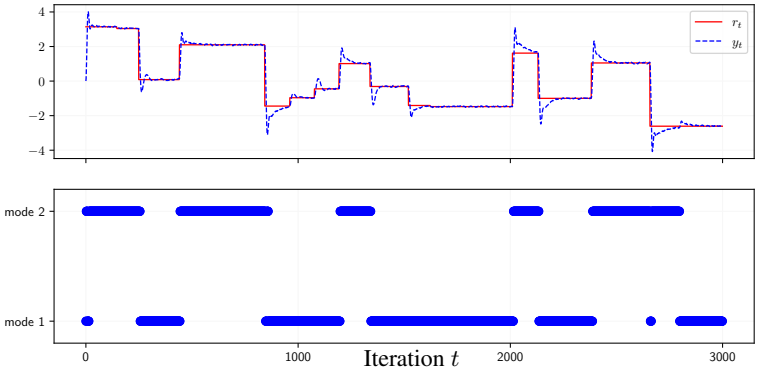

Figure 61: Example 2, offline case, tracking of a priori unknown reference signal (top plot) and associated controller's modes (bottom plot).

	$M = 1$	$M = 2$	$M = 2$
centroids	//	fixed	optimized
averaged cost	1016.8	961.8	932.7

Table 23: Example 2, offline case, averaged cost over 1000 tracking tasks.

Figure 61 shows the tracking behavior of the synthesized controller with $M = 2$ modes and optimized centroids. The costs of the bi-modal and single-modal controllers over a batch of 1000 piecewise constant reference signals taking values in $[-\pi, \pi]$, included in Table 23, are in line with what observed.

4.4.2.2 Online setting

We synthesize online a switching controller with $M = 2$ modes while performing the tracking task online shown in Figure 62, for $N_{\text{learn}} = 10000$ steps. In this case we test the OSPS algorithm, assigning the decision variables (s_t, r_t) to either of the two modes according to (4.10), considering as semi-distance for

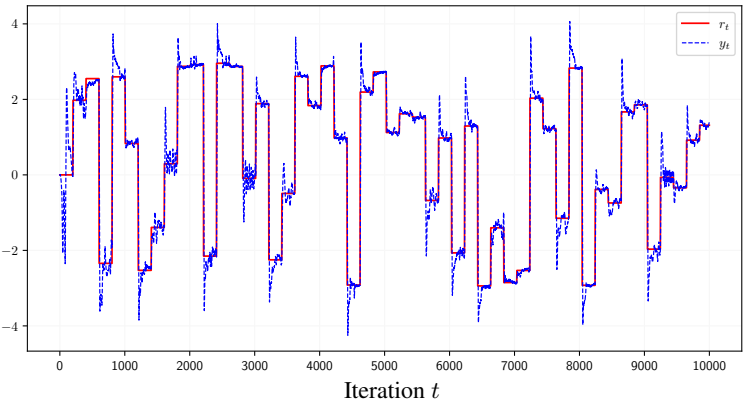

Figure 62: Example 2, online learning, online output tracking of an a-priori unknown exploratory reference signal.

the clusters assignment $d([\begin{smallmatrix} s_1 \\ r_1 \end{smallmatrix}], [\begin{smallmatrix} s_2 \\ r_2 \end{smallmatrix}]) = \left\| f([\begin{smallmatrix} s_1 \\ r_1 \end{smallmatrix}]) - f([\begin{smallmatrix} s_2 \\ r_2 \end{smallmatrix}]) \right\|_2$, with

$$f([\begin{smallmatrix} s \\ r \end{smallmatrix}]) = [\cos(\theta),\ \sin(\theta),\ \dot{\theta},\ u,\ q,\ \cos(r),\ \sin(r)]'$$

for each $s = [\theta,\ \dot{\theta},\ u,\ q]' \in \mathbb{R}^4$.

The values $c_1^0 = [0,\ 0,\ 0,\ 0,\ 0]'$ and $c_2^0 = [\pi,\ 0,\ 0,\ 0,\ \pi]'$ are used as centroids initial guess for the synthesis of controller with optimized centroids, while they are kept as centroids throughout the learning of the switching controller with fixed centroids. Local affine models are fitted tuning the Kalman Filter with covariance matrices $Q_k = I_4$, $R_k = 1$. At every iteration t the mini-batch sampled according to the described tuning parameters is augmented by adding an element $[x_t,\ q_t,\ r_t]'$ containing the current state, integral action and reference. Table 24 contains the costs achieved online while learning the controllers, and after the learning, while tracking the same same reference employed as a test in the offline case. Figure 63 shows the tracking behavior of the synthesized $M = 2$ modes controller with optimized centroids on said tracking test. From Table 24, the switching controllers are preferable than the single mode one.

In particular, learning online the controller with optimized centroids is a little

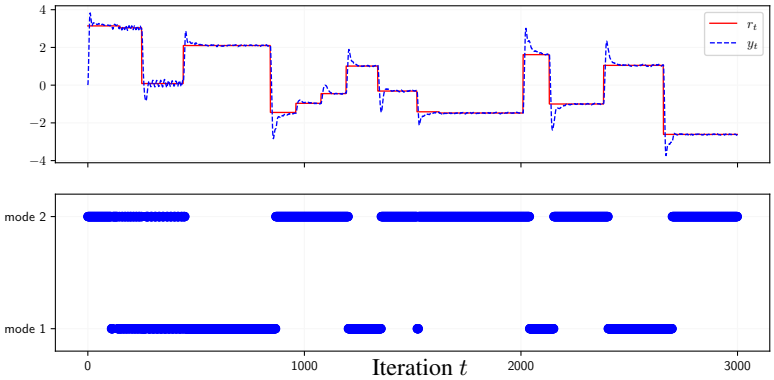

Figure 63: Example 2, online case, tracking of a priori unknown reference signal (top plot) and associated controller's modes (bottom plot).

	centroids	online	validation
$M = 1$		4066.9	1082.9
$M = 2$	fixed	3644.1	1010.4
$M = 2$	optimized	3830.3	944.2

Table 24: Example 2, online case, tracking cost.

more expensive than what is required for the one with fixed centroids, but it pays off in the successive tracking performance. This can be observed as well from the averaged cost obtained applying the controllers over 1000 randomly generated piecewise constant tracking tasks, as demonstarted in Table 25.

	$M = 1$	$M = 2$	$M = 2$
centroids	//	fixed	optimized
averaged cost	1019.2	973.1	920.8

Table 25: Example 2, online case, averaged cost over 1000 tracking tasks.

Chapter 5

Collaborative cloud-aided learning of optimal controllers

This chapter incorporates the Optimal Policy Search algorithm in a collaborative multi-agent framework for the learning of optimal feedback control laws, considering multi-agent systems characterized by structural similarities, exploiting a cloud-aided scenario. At first, the multi-agent scenario is described and the collaborative problem is formulated. Two collaborative learning strategies are then designed, and the resulting enhancement of learning performance with respect to the single agent scenario is shown, by applying the approaches to the output-tracking problem.

5.1 Multi-agent cloud-aided setup

In this chapter we consider a multi-agent scenario, characterized by a set of systems that share similar nominal dynamics. This is quite common, particularly for industrial systems, such as automobiles, aerial vehicles, industrial robots, or chemical process units, that are deployed in mass production. These plants (here indicated as *agents*) are often clones of each other, designed, constructed, and calibrated by the manufacturer in the same way.

Mass produced devices are likely designed to be employed on specific sets of tasks, involving the same functionalities, and hence sharing similar general objectives, while possibly operating within different environments. Thinking of mass produced learning agents, this translates directly to them being designed to learn a shared set of capabilities. Nonetheless, the individual learning of such capabilities is usually sought while each agent is employed in performing a local task, often different from the one of other agents. An example of this is embodied by a set of autonomous vehicles: each vehicle should acquire the capability of handling many different kinds of road and many traffic configurations. Each agent per se, once undergoing a training phase for instance, is likely to be indeed employed in a different environment, on a specific driving task (i.e. on a circuit with many curves, or on a straight road, and so on).

It is known that the design of policies for a single plant usually requires rather long and expensive experimental campaigns to obtain informative data. Thanks to recent advances in cloud computing, though, designers are allowed to overcome this limitation by leveraging on the increasing connectivity between devices, to gather more information when searching for control policies for systems that share similar dynamics. Not only it is now technically and economically feasible to collect and store information gathered from different plants, for example in a large volume production setting, but mass-produced systems are nowadays often connected to the cloud. For instance modern vehicles maintain a constant V2C (Vehicle to Cloud) bi-directional connection with decent data bandwidth for different monitoring, maintenance, and updating services.

The exploitation of such connectivity can be greatly beneficial for agents with similar dynamics when asked to self-adapt their control laws. In this context, the advances in cloud computing allow designers to gather more information that can be exploited to improve exploration, ultimately leading to better individual policies. In fact, each plant may explore different regions of the state and action spaces than others, so that the union of such explorations can provide a wide coverage of the operating space. Thus, sharing information during closed-loop operation can dramatically help each system to adapt its control laws so to attain its own goals, in particular when optimal performance is sought.

The connection with the cloud has other advantages. For example, in case of agents having limited embedded computing capabilities but access to resources

on the cloud, we can even assume that each agent locally performs simple operations only. Together with exploiting the agents connectivity to improve their learning capabilities, we are also interested in granting that the agents retain their states, actions, and rewards, that could be sensitive for privacy reasons. This will be achieved by allowing them to share surrogate of their experiences instead, properly chosen to carry on information able to enhance the learning procedure.

Literature on cooperative policy search methods

Before introducing the proposed collaborative learning approaches, few contributions on cooperative policy search are mentioned in the following paragraph, with particular focus on underling the differences in the considered setup. Sharing-based principles have already been exploited for policy search purposes. In literature it is possible to find contributions that consider scenarios where all agents are cooperating in the same environment towards the achievement of the same goal, as for instance in (Dimakopoulou, I., and Van Roy, 2018), or to steer all local policies to the same value, under the assumption that they lie in the same space, as in (Nair et al., 2015), (Khan et al., 2018). Differently from (Dimakopoulou, I., and Van Roy, 2018), in the collaborative approach presented in this chapter the agents are not required to share the same goal and to operate within the same environment. Moreover, unlike (Khan et al., 2018), the collaborative approaches proposed in this chapter do not require the agents to share their private states, but surrogate of their experiences only, as already pointed out while presenting the considered multi-agent cloud-aided setup.

5.2 Consensus-based collaborative learning

Consider N dynamical agents, and let $s_t^n \in \mathbb{R}^{n_s}$ be a Markovian signal specifying the behavior of the n-th system. Assume that the latter evolves over time according to

$$s_{t+1}^n = h(s_t^n, p_t^n, u_t^n, d_t^n), \tag{5.1}$$

where, as in (2.3), the signal $p_t^n \in \mathbb{R}^{n_p}$ in (5.1) is a vector of measurable exogenous signals, and $d_t^n \in \mathbb{R}^{n_d}$ is a vector of unmeasured disturbances.

As previously specified, we consider the dynamics h to be unknown, but in this case we assume it to be common to all agents. To represent the shared set of capabilities to be learned by each agent, we consider a stage-cost function ρ_n for the n-th agent, such that $\rho_n : \mathbb{R}^{n_s} \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}$. The functions ρ_n are assumed to be somehow similar, in the sense that, although they could assign more or less penalty to specific behaviors, each ρ_n should be in line with the common set of capabilities to be learned by all the agents. In general, one agent could be requested to "specialize" more over tasks of a specific nature, but as mentioned, we consider group of agents with similar purpose and learning goals. Said stage cost is employed to define the local costs J^n_∞ associated with an agent n, according to (2.4) (or J^n_T as in (2.5), in case of episodic tasks). Following from that, we proceed by considering, for each agent, the expected trajectory cost $\mathbb{E}_W[J^n](\pi)$ introduced in (2.6) - (2.7) as goal function.

In this setup we aim at finding N optimal deterministic control policies $\pi_n : \mathbb{R}^{n_s} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_u}$, so that the input fed to the n-th system at time t is given by $u^n_t = \pi_n(s^n_t, p^n_t)$. In case of agents separately learning such policy, the n-th learning problem could be formulated as an optimization problem and tackled following the Optimal Policy Search method, as previously shown. By doing so, though, all the notions of similarity expressed in the previous section are disregarded and each agent would exploit exclusively its own experience, gathered from the direct interaction with the environment induced by its local task. Our current aim is instead to design a collaborative learning strategy capable of exploiting the similarity of the agents, the similarity of their goals, and their capability of exchanging information, in order to improve their learning process. In order to incorporate such elements, we formulate a generic multi-agent collaborative learning problem, i.e.,

$$\begin{aligned} \{\pi^*_n\}_{n=1}^N,\ \pi^* = \arg \quad & \min_{\substack{\{\pi_n\}_n, \\ \pi}} \sum_{n=1}^{N} \mathbb{E}_W[J^n](\pi_n), \\ & \text{such that} \quad \phi(\pi_n) = \pi, \qquad n = 1, \dots, N, \end{aligned} \tag{5.2}$$

where the (known) function ϕ describes the relation between the local policies $\{\pi_n\}_{n=1}^N$ and a global control law $\pi : \mathbb{R}^{n_s} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_u}$. Such global law embeds the characteristics that should be shared by all the local policies, with the rationale that plants sharing the same dynamics and having similar optimization

goals should share similar optimal policies as well. In case of $\rho_n = \rho$ for each n, given that the goal of each agent is exactly the same, the optimal policies π_n^* should coincide, hence we consider $\phi = id$.

To simplify the search domain, we parameterize problem (5.2), substituting each agent's policy π_n with a parameterized one $\pi_{H_n}^n$ with parameters $H_n \in \mathbb{R}^{n_{h_n}}$. To represent the similarity among optimal policies we design

$$\pi_{H_n}^n(s_t,\, p_t) = \pi_{K_n}(s_t,\, p_t) + \psi_{J_n}^n(s_t,\, p_t), \tag{5.3}$$

with $K_n \in \mathbb{R}^{n_k}$ and $J_n \in \mathbb{R}^{n_{h_n} - n_k}$. In this case, the local controllers employ by design a partially common structure $\pi_\bullet$, together with a local component, embodied for agent n by $\psi_\bullet^n$. The idea behind this is that, on one hand, π_{K_n} would be dedicated to the goals shared by all agents, encoding the similarity among local controllers. On the other hand $\psi_{J_n}^n$ would specialize on the local goals specific to agent n. The global policy π, representing the similarities existing among local optimal controllers, is as well parameterized employing the shared structure $\pi_\bullet$, and we indicate its parameterization as $\pi_K \in \mathbb{R}^{n_k}$. One can note that the use of Dirac functions in the definition of π_{K_n} and $\psi_{J_n}^n$ can separate the decision variables domain $\mathbb{R}^{n_s+n_p}$, if necessary, making it so that the π_{K_n} and $\psi_{J_n}^n$ are alternatively used to generate the inputs u_t fed to the n-th agent. Otherwise the two policies can be used in combination to generate control inputs at every instant.

Following the construction presented in Section 2.1 to build the approximated problem (2.10) from problem (2.8), we approximate problem (5.2) as

$$\{H_n^*\}_{n=1}^N, K^* = \underset{\substack{\{H_n\}_n,\\ K}}{\arg\min} \sum_{n=1}^{N} \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho_n(s_\ell^n,\, p_\ell^n,\, u_\ell^n) + \rho_L^n(s_L^n,\, p_L^n)\Big], \tag{5.4}$$

$$\text{such that} \quad u_\ell^n = \pi_{H_n}^n(s_\ell^n,\, p_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \tag{5.5}$$

$$s_{\ell+1}^n = h(s_\ell^n,\, p_\ell^n,\, u_\ell^n,\, d_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \tag{5.6}$$

$$d_\ell^n \sim D_\ell^n, \quad \ell = 0, 1, \ldots, L-1, \tag{5.7}$$

$$p_\ell^n \sim P_\ell^n, \quad \ell = 0, 1, \ldots, L, \tag{5.8}$$

$$s_0^n \sim S_P^n, \tag{5.9}$$

$$K_n - K = 0, \quad n = 1, \ldots, N, \tag{5.10}$$

where we can see that the relation between π_n and π introduced in the constraint of problem (5.2) is encoded in (5.10). In particular, the function ϕ describing the relation between the local policies and the global one is the projection of H_n on the parameters characterizing the shared part π_{K_n} of the policy structure, i.e. $\phi(H_n) = K_n$. Problem (5.4) formulates the collaborative learning problem as a consensus problem: the agents are called to learn their policy parameters $H_n = (K_n, J_n)$ and to combine their understandings to agree on part of them and, specifically, on the part dedicated to the global goals of the group, represented by the vector K_n^t.

5.3 Consensus-based collaborative OPS

To approach problem (5.4) we design an iterative collaborative policy search scheme, that combines updates of the local policies H_n via the Optimal Policy Search approach, and the computation of the global gain through the local information shared by the agents, exploiting the Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011). We consider the augmented Lagrangian associated to (5.4), defined as

$$\mathcal{L}(\{H_n\}_{n=1}^N, \lambda_n, K) = \sum_{n=1}^{N} \mathcal{L}_n(H_n, \lambda_n, K), \tag{5.11}$$

$$\mathcal{L}_n(H_n, \lambda_n, K) = \mathbb{E}_W[J_L^n](H_n) + \lambda_n'(K_n - K) + \frac{\beta}{2}\|K_n - K\|_2^2, \tag{5.12}$$

with $\beta > 0$ being a tuning parameter and $K_n = \phi(H_n^t)$. In (5.12) $\mathbb{E}_W[J_L^n]$ and λ_n are the goal function of agent n and the Lagrange multiplier associated with the constraint $K_n - K = 0$, respectively. By running a new instance of ADMM at each iteration t, the local policies, the global policy, and the Lagrange multipliers are computed iterating the following steps:

$$H_n^t(i+1) = \arg\min_{H_n} \mathcal{L}_n(H_n, \lambda_n^t(i), K^t(i)) = \tag{5.13a}$$

$$= \arg\min_{H_n} \mathbb{E}_W[J_L^n](H_n) + (\lambda_n^t(i))'(K_n - K^t(i)) +$$

$$+ \frac{\beta}{2}\| K_n - K^t(i) \|_2^2, \quad n = 1, \ldots, N$$

$$K^t(i+1) = \frac{1}{N}\sum_{n=1}^{N}\left[K_n^t(i+1) + \frac{1}{\beta}\lambda_n^t(i)\right], \tag{5.13b}$$

$$\lambda_n^t(i+1) = \lambda_n^t(i) + \beta\,(K_n^t(i+1) - K^t(i+1)), \tag{5.13c}$$

$$n = 1, \ldots, N$$

where $i \in \mathbb{N}$ is a counter of the ADMM iterations, and the values $H_n^t(0)$, $K^t(0)$, and $\lambda_n^t(0)$ are initialized with the values computed at iteration $t-1$. The steps in (5.13) are carried out until a predefined stopping criterion is satisfied.

According to (5.13b), the global estimate is the average of the local polices and Lagrange multipliers, thus relying on information collected from all agents. The local information is aggregated into K^t, that is then used for the optimization of the local controllers at the next iteration. By doing so, each local controller can access a surrogate of the experience of the other controllers only, without being privy to their states, actions, exogenous signals or current policies, that can hence be kept private. The privacy requirements stated in Section 5.1 are then satisfied, thanks to the introduction of the global law π_{K^t}.

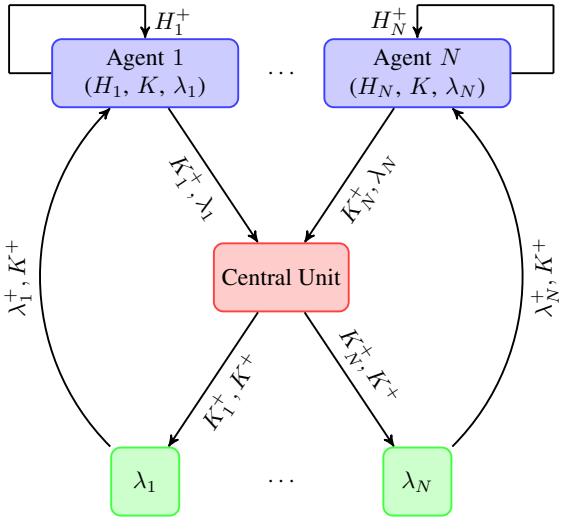

Figure 64: Transmission scheme over an ADMM iteration.

By looking at the steps in (5.13), it is also clear that at each ADMM iteration the global policy can be updated on a central processing unit, provided the updated local estimates $\{H_n^t(i+1)\}_{n=1}^N$, while the local policies and the local Lagrange multipliers can be updated by using parallel dedicated resources on the cloud, as represented in Figure 64 (where the step and iteration indices t and i are substituted by a simpler notation for clarity).

The iterations to be carried out at a given time instant t are summarized in Algorithms 9 - 10. In particular, steps 1-14 of Algorithm 9 are computed by the resources on the cloud dedicated to the separate agents: after updating the states and models histories, the n-th local policy H_n is updated by solving (5.13a).

Algorithm 9 Consensus-based collaborative OPS - Cloud actions at step t

Input: Last measurements $\{y_{t+1}^n,\, x_t^n,\, u_t^n\}_{n=1}^N$ for each agent.
States/models histories $\{X_{(n)}(t-1),\, \Theta_{(n)}(t-1)\}_{n=1}^N$.
Previous values $\{H_n^{t-1}\}_{n=1}^N$, K^{t-1}, $\{\lambda_n^{t-1}\}_{n=1}^N$.
Output: Updated parameters $\{H_n^t\}_{n=1}^N$, K^t, $\{\lambda_n^t\}_{n=1}^N$.

1: **for** $n = 1, \ldots, N$ **do**
2: $X_{(n)}(t) = X_{(n)}(t-1) \cup \{x_t^n\}$;
3: update Θ_n^t based on $(y_{t+1}^n, x_t^n, u_t^n)$;
4: $\Theta_{(n)}(t) = \Theta_{(n)}(t-1) \cup \{\Theta_n^t\}$;
5: set $H_n^t(0) = H_n^{t-1}$, $K^t(0) = K^{t-1}$, $\lambda_n^t(0) = \lambda_n^{t-1}$;
6: **end for**
7: **while** $(i = 0\,;\ i < i_{max}$ or **not**(termination-cryterion)$\,;\ i++)$ **do**
8: **for** $n = 1, \ldots, N$ **do**
9: set $H_n^0 = H_n^t(i)$
10: **for** $m = 1, \ldots, M$ **do**
11: OPS/OSPS iteration applied to pb. (5.13a) $\rightarrow$ H_n^m ;
12: **end for**
13: $H_n^t(i+1) = H_n^M$;
14: **end for**
15: compute $K^t(i+1)$ as in (5.13b);
16: **for** $n = 1, \ldots, N$ **do**
17: compute $\lambda_n^t(i+1)$ as in (5.13c);
18: **end for**
19: evaluate termination-cryterion;
20: **end while**
21: $H_n^t = H_n^t(\text{last})$, $K(t) = K^t(\text{last})$ and $\lambda_n(t) = \lambda_n^t(\text{last})$

Algorithm 10 Consensus-based collaborative OPS - online actions agent n at step t

Input: State and action $(s_t^n,\, u_t^n)$.
Output: State and action $(s_{t+1}^n,\, u_{t+1}^n)$.

1: apply u_t^n to the n-th agent and collect y_{t+1}^n;
2: measure or compute z_{t+1}^n (see Eq. (2.17)-(2.18));
3: transmit $(y_{t+1}^n, x_t^n, u_t^n)$ and eventually z_{t+1}^n to the cloud;
4: retrieve H_n^t from the cloud (see Algorithm (9));
5: measure p_{t+1}^n;
6: build x_{t+1}^n as in (2.2) and $s_{t+1}^n = [\, x_{t+1}^n,\, z_{t+1}^n]$;
7: compute $u_{t+1}^n = \pi_{H_n^t}(s_{t+1}^n,\, p_{t+1}^n)$;

The problem is tackled via M iterations of the OPS method described in Algorithm 1, performing hence sampling, gradient approximation and parameters update as described in Chapter 2 if the policy parameterization in (5.3) is smooth. Alternatively the local controllers update described for the OSPS method in Algorithm 6 can be employed if the policy is non-smooth with respect to H_n. Step 15 of Algorithm 9 represents the computation of the global law by the central unit, and steps 16-18 represent the parallel update of the Lagrange multipliers.

The combination of Algorithms 9 and 10 compose the online ADMM-based OPS method. The described algorithm can be performed offline as well, avoiding to perform the steps in Algorithm 10 related to the real-time interaction of the single agents with the environment, instead relying on a previously collected stream of input-output data to carry on the optimization in the cloud, iterating the steps in Algorithm 9.

The online setup, involving a real-time implementation, is described assuming synchronous back and forth communications between the agents and the central processing unit, which might be unfeasible, especially for fast sampling systems. In such cases, then, lags in the communication among the separate units might be experienced.

Since in the considered setting we search for the optimal time-invariant policy parameters, after some initial steps we expect the lags in communication not to substantially deteriorate the performance of the method.

5.4 Collaborative learning of output-tracking controllers

We now analyze the advantages given by the proposed multi-agent strategy for the collaborative learning of controllers, considering as a control goal the tracking of a-priori unknonw output reference signals. To do so, we employ for each of the N plants the definitions of states s_t^n and exogenous signals r_t^n in (3.1) designed for the output tracking problem in Section 3.1, as well as the associated stage costs ρ_n, ρ_L^n in (3.2).

We select equal stage cost functions among the agents (i.e., $\rho_n = \rho$ for each n), considering the goal of each agent to coincide exactly with the global goal of the whole group, that is learning a static controller capable of tracking every possible a-priori unknown reference signal taking values in a known interval $[\, r_{\min},\, r_{\max}]$. Following from this, all the policy parameters are included on the consensus process (i.e., $\phi = id$), and we parameterize the local policies and the global one employing the same functional structure, disregarding the local part of the controller $\psi_{J_n}^n$.

We choose a linear policy parameterization

$$\begin{aligned} u_t^n = \pi_{H_n}^n(s_t^n, r_t^n) = u_{t-1}^n + \pi_{K_n}(s_t^n, p_t^n) = \\ = u_{t-1}^n + K_n \begin{bmatrix} s_t^n \\ r_t^n \end{bmatrix} = u_{t-1}^n + K_n^s s_t^n + K_n^r r_t^n. \\ \pi_K(s_t, r_t) = u_{t-1} + K \begin{bmatrix} s_t \\ r_t \end{bmatrix} = u_{t-1} + K^s s_t + K^r r_t. \end{aligned}$$

In this setup, the policy parameterization is thus smooth, so that the OPS method will be employed for the local parameters update.

Performance index To evaluate the advantage in learning given by applying the ADMM-based collaborative OPS method, we test it on problems having a known optimal linear controller, characterized by parameters K_{OPT}. In this setting, we want to quantitatively assess how the convergence of the local and global controllers K_n^t, K^t to K_{OPT} improves thanks to the proposed tecnique. Hence, given a generical series of evolving parameters $\{K_t\}_t$, we define the

index

$$T_\epsilon(\{K_t\}_t) \doteq \min\{t \in [1,\, N_{\text{learn}}] \mid \|K_t - K_{\text{OPT}}\|_2 < \epsilon\}, \tag{5.14}$$

representing the first iteration such that K_t is in the ϵ-neighborhood of K_{OPT}. The quality of the convergence of $\{K_t\}_t$ to K_{OPT} once the ϵ-neighborhood has been reached is evaluated, considering the following index:

$$\mathrm{D}_\epsilon(\{K_t\}_t) \doteq \frac{1}{N_{\text{learn}} - T_\epsilon(\{K_t\}_t) + 1} \sum_{t=T_\epsilon(\{K_t\}_t)}^{N_{\text{learn}}} \|K_t - K_{\text{OPT}}\|_2, \tag{5.15}$$

representing the distance from K_{OPT} mantained by K_t, after $T_\epsilon(\{K_t\}_t)$.

The indices (5.14) and (5.15) can be computed considering the serie $\{K^t\}$ of evolving parameters associated with the global law or the N series of local laws parameters $\{K_n^t\}$. In the latter case, the performance over the group of agents can be summarized by considering the averaged indices

$$\overline{\mathrm{T}}_\epsilon \doteq \frac{1}{N} \sum_{n=1}^{N} \mathrm{T}_\epsilon(\{K_n^t\}_t), \tag{5.16}$$

$$\overline{\mathrm{D}}_\epsilon \doteq \frac{1}{N} \sum_{n=1}^{N} \mathrm{D}_\epsilon(\{K_n^t\}_t). \tag{5.17}$$

We assess the benefits of exploiting shared experiences by comparing the results obtained by a group of N agents that learn through the consensus-based collaborative OPS to the ones obtained for the same N agents individually learning through the non-collaborative baseline OPS approach.

To assess the improvement achieved by collaboration in an online setting, the costs of online tracking are also considered, looking at the sum of the individual stage costs, averaged with respect to the N agents.

5.5 Example: consensus-based collaborative learning

We consider $N = 4$ data-generating systems described by the (unknown) SISO LTI model in Example 1 of Chapter 2, equation (3.4), employing the same states

definition s_t^n, with $n_y = 3$ measured output and $n_u = 2$ measured input, i.e.,

$$s_t^n = \begin{bmatrix} x_t^n \\ q_t^n \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} y_t^n \\ y_{t-1}^n \\ y_{t-2}^n \\ u_{t-1}^n \\ u_{t-2}^n \end{bmatrix} \\ q_t^n \end{bmatrix}, \quad q_{t+1}^n = q_t^n + (y_t^n - r_t^n).$$

The employed stage costs and terminal costs are tuned with the same weights for all agents, and the considered trajectory foresees over the same horizon $L = 10$. All the considerations related to the tracking problem expressed in Section 3.2 hold also for the considered multi-agent scenario, in particular the value of the known optimal linear feedback K_* to be retrieved.

In the following, we are going to study the effect of Algorithm 9-10 on the convergence of the local policies $\{K_n^t\}_n$ and K^t to K_* over a learning process of length $N_{\text{learn}} = 500$ steps within an ideal setting, with no latencies in communications. The agents' policies are initialized with $K_n^0 = \underline{1}_{7\times 1}$, the initial global law is set as $K^0 = \underline{0}_{7\times 1}$, and the initial Lagrange multipliers are chosen as $\lambda_n^0 = 1e-3 \cdot \underline{1}_{7\times 1}$. The online learning phase is conducted starting from an initial steady state $x_0 = \underline{0}_{6\times 1}$ (i.e., $y_{ss} = u_{ss} = 0$). Each agent learns while performing online a different local task, consisting in an a-priori unknown piecewise constant exploratory reference signal, randomly sampled and taking values in $[-100,\, 100]$.

At instant t, we perform 5 ADMM iterations, with $\beta = 1$. The ADMM steps include the update of the local policies K_n^t, performed by approximating the solution of problem 5.13a with $M = 1$ iteration of the Optimal Policy Search method. Each OPS iteration m employs the current measured state x_t described in (2.2) as sample of the initial state ($N_0 = 1$), together with $N_q = 1$ initial integral action values, $N_r = 1$ reference trajectories and N_d disturbances. The references and the integral action values are sampled uniformly in $[-100, 100]$ and $[-10, 10]$ respectively. The disturbances, instead, are sampled according to a Gaussian white random variable, characterized by variance σ_d^2. The sampled state is perturbed with $v_n \sim 0.1 \cdot \mathcal{N}(0, 1)$. For the update of the controller parameters, AMSGrad (Reddi, Kale, and S. Kumar, 2019) is employed, with parameters $\alpha = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Considering as initial guess $\Theta_0 = \underline{0}_{1\times 6}$ and $P_0 = (1e+5)\cdot I_6$, the local models are recursively computed by Kalman Filtering, with noise covariance matrices $Q_k = 0.01 \cdot I_6$ and $R_k = 0.01$. Having chosen a linear parameterization, at every iteration we combine the described algorithm with the heuristic for stability described in Section 2.3.2.1. The following two subsections contain the tuning details that differ between the noiseless case and the noisy one (that is, the values N_d, σ_d), together with the results obtained in the two cases. The same parameters are employed for the individual learning of the 4 agents through the (non-collaborative) OPS approach, that is used as baseline case, to show the achieved enhancement of the learning.

5.5.1 Noiseless case

In the noiseless case, we can measure the exact output of (3.4), i.e., we set $\sigma_y^2 = 0$. We consider no disturbance d_t affecting the local linear models ($N_d = 1$, $\sigma_d = 0$). From Figure 65 it is observable that the agents that learn in collaboration converge faster to the optimal policy than the ones that individually optimize their policy parameters.

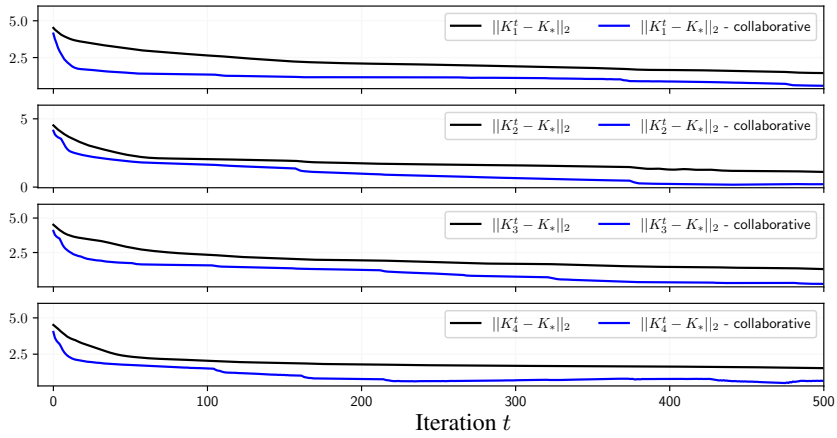

Figure 65: Consensus-based collaborative learning in noiseless conditions, $N = 4$ agents, convergence of the local parameters $\{K_1^t\}_t$, $\{K_2^t\}_t$, $\{K_3^t\}_t$, $\{K_4^t\}_t$ to the optimal value K_*.

learning	average online cost	$\overline{\mathrm{T}}_2$	$\overline{\mathrm{D}}_2$
individual	$1.43 \cdot 1e+6$	164.5	1.65
collaborative	$5.77 \cdot 1e+5$	25.25	0.96

Table 26: Consensus-based collaborative learning in noiseless conditions, $N = 4$ agents, performance indices

This evaluation is confirmed by the results in Table 26, where the indices (5.17) with $\epsilon = 2$ are included, together with the online tracking cost achieved on average by the agents in collaboration, while tracking different online tasks as shown in Figure 66. All such indicators are compared with the same ones, associated with the agents learning individually. In particular, $\overline{\mathrm{T}}_2$ inidicates that on average the four agents, when collaborating, reach the 2-distance neighborhood of K_* more than a hundred steps before than when the learning is separate. Moreover, the average distance $\overline{\mathrm{D}}_2$ mantained by the collaborative agents is smaller, indicating that, after reaching the 2-distance neighborhood of K_* the first time, the agents tend not to leave it, but to proceed towards K_*. The average tracking cost in collaboration is reduced of one order of magnitude, showing how the increased convergence speed positively affects the performance.

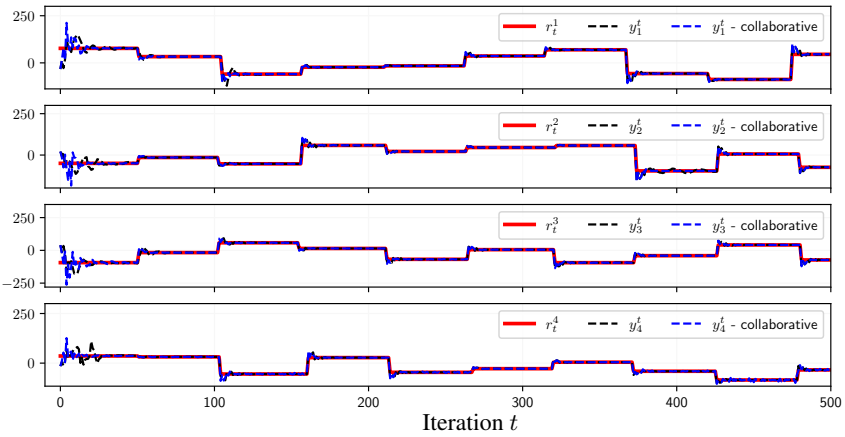

Figure 66: Consensus-based collaborative learning in noiseless conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal.

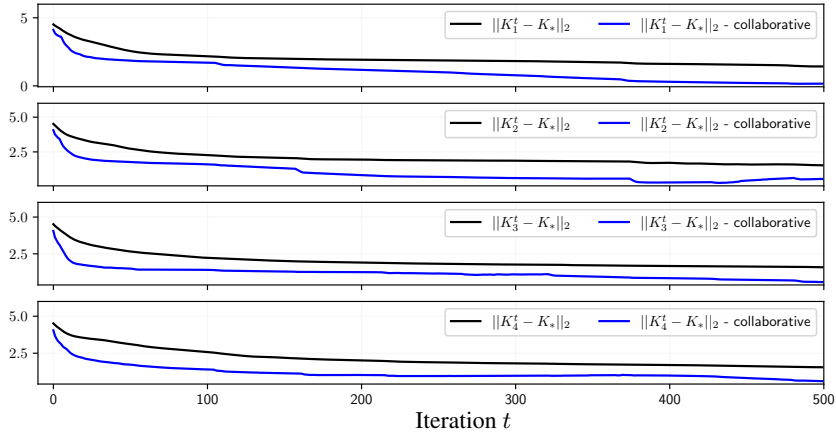

Figure 67: Consensus-based collaborative learning in noisy conditions, $N = 4$ agents, convergence of $\{K_1^t\}_t$, $\{K_2^t\}_t$, $\{K_3^t\}_t$, $\{K_4^t\}_t$ to the optimal value K_*.

5.5.2 Noisy case

Now, we consider a noisy output of (3.4), i.e., we set $\sigma_y = 0.1$. At every iteration, hence, we sample $N_d = 10$ disturbance trajectories, from a white Gaussian random variable with variance $\sigma_d^2 = 0.01$. Regarding the learning of the single agents, the same considerations expressed in the noiseless case are valid when performing the same test in the noisy case, as shown by the convergence of the local laws $\{K_1^t\}_t$, $\{K_2^t\}_t$, $\{K_3^t\}_t$, $\{K_4^t\}_t$ in Figure 67, and by the online tracking performed by the four agents in Figure 68. A quantitative evaluation is given by the performance indices (5.17) and average online tracking costs in Table 27.

learning	average online cost	$\overline{\mathrm{T}}_2$	$\overline{\mathrm{D}}_2$
individual	$1.85 \cdot 1e+6$	168.7	1.76
collaborative	$5.64 \cdot 1e+5$	24.5	1.01

Table 27: Consensus-based collaborative learning in noisy conditions, $N = 4$ agents, performance indices

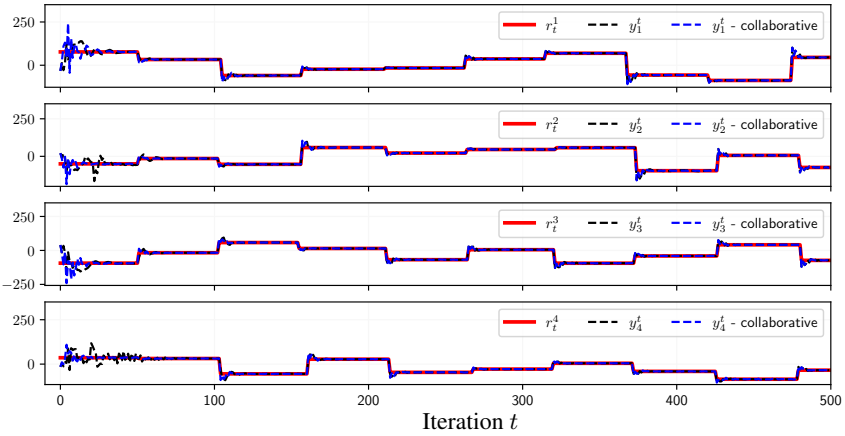

Figure 68: Consensus-based collaborative learning in noisy conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal.

We consider now a different setup, in order to underline the advantages of the consensus-based multi-agent collaborative strategy on the quality of both the synthesized global law, and the local law of a specific agent, that we will indicate as Agent 1, while increasing the number of agents collaborating.

Each agent performs a different online tracking task while learning. The tasks are designed so that different agents are employed on different parts of the space of reference signals. In order to do so, we repeat the learning experiment in the described conditions and employing the described tuning, while considering a varying number of agent $N = 1, 2, 10$. Agent 1 is present in all the three learning tests ($N = 1, 2, 10$). In each test, it is associated with the same tracking task T_1 and when $N = 1$ it performs the non-collaborative Optimal Policy Search method. In the second test (i.e., when $N = 2$) Agent 1 and Agent 2 learn in collaboration, while performing two different tasks. Agent 1 performs the same task T_1 as in the previous test, while Agent 2 performs a different task T_2. Finally, the third test involves $N = 10$ agents collaborating. Each of them performing a different task. Among them, Agent 1 and 2 perform task T_1 and T_2, respectively, as in the previous tests.

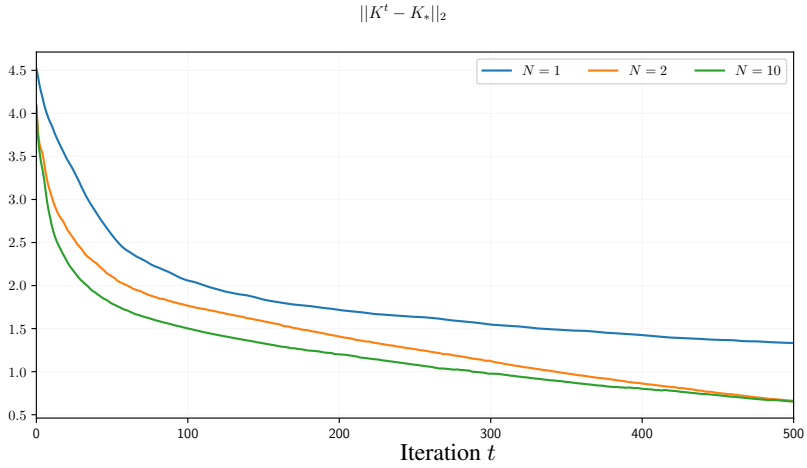

Figure 69: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, convergence of the global parameters $\{K^t\}_t$ to K_*.

Figure 69 shows the effect of increasing the number of collaborators on the associated global law K^t. Given that the non-collaborative Optimal Policy Search method does not provide any global law and that only one agent is involved, in case of $N = 1$, we consider for the comparison the only available law $K^t = K_1^t$.

From Figure 69 it is visible that both the global laws obtained in collaborative mode contain more information with respect to the optimal policy K_* than the policy K_1^t, individually synthesized by Agent 1 in the same conditions. Hence, Agent 1 (as well as all the other agents) could benefit from exploiting such information, in order to achieve faster convergence. Moreover, in case of 10 agents collaborating, the convergence of K^t is additionally improved with respect to only 2 agents sharing information, in the sense that, although the distance between K_* and the collaborative global policies for $N = 2, 10$ reached after $N_{\text{learn}} = 500$ steps is similar, as shown by Figure 69, the global policies synthesized by 10 agents in collaboration reaches the 2-distance neighborhood centered in K_* significantly faster than the global policies synthesized by 2 agents, and it proceeds towards convergence while being closer to K_*.

learning	N	$\mathrm{T}_2(K^t)$	$\mathrm{D}_2(K^t)$
individual	1	113	1.57
collaborative	2	62	1.21
collaborative	10	35	1.10

Table 28: Consensus-based collaborative learning in noisy conditions, varying number of agents, performance indices related to the global law convergence.

Hence, sharing such global law with the agents, drives them earlier and better towards K_*. This is quantitatively observable as well from the performance indices (5.14) - (5.15) associated with K^t, included in Table 28.

The effect of the sharing such information on Agent 1 is underlined by Figure 70 - 71 and Table 29. Such figures and performance indices show how both the convergence and performance of Agent 1 are improved by the exploitation of a better global law.

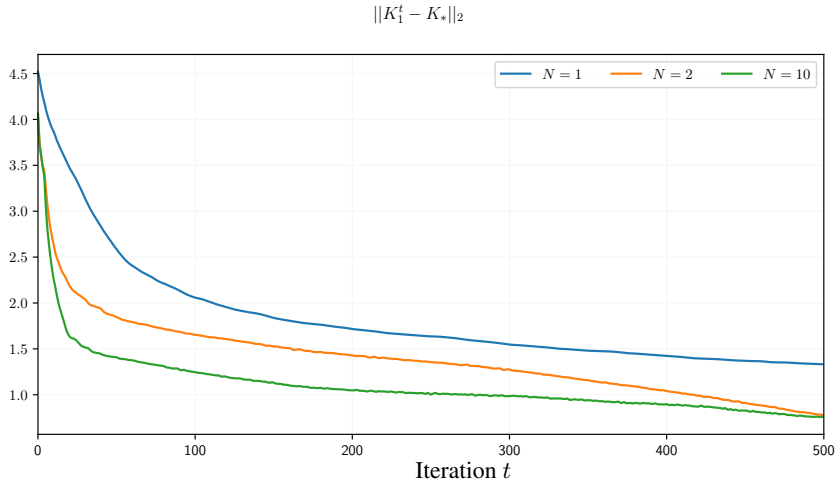

Figure 70: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, convergence of the local parameters $\{K_1^t\}_t$ of Agent 1 to the optimal value K_*.

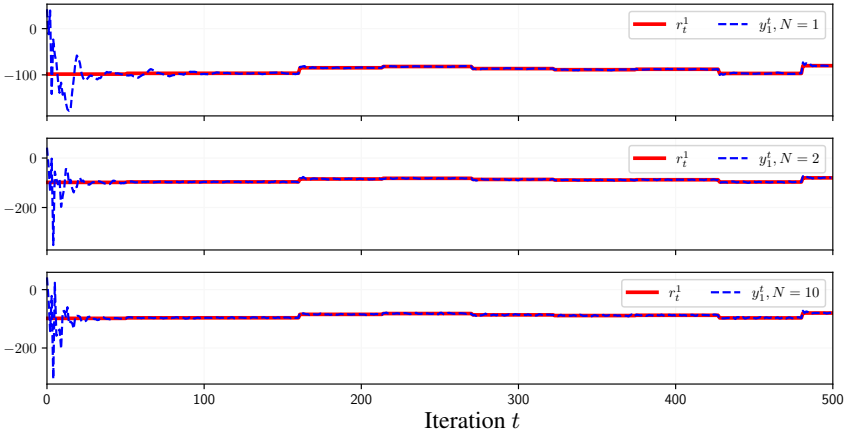

Figure 71: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, online output tracking of an a-priori unknown exploratory reference signal r_t^1 by Agent 1.

learning	N	$\mathrm{T}_2(K_1^t)$	$\mathrm{D}_2(K_1^t)$	tracking cost Agent 1
individual	1	113	1.57	$3.58 \cdot 1e+6$
collaborative	2	33	1.32	$7.97 \cdot 1e+5$
collaborative	10	15	1.06	$7.19 \cdot 1e+5$

Table 29: Consensus-based collaborative learning in noisy conditions, varying number of agents, performance indices related to the local law K_1^t convergence and to Agent 1 online performance.

Analogous results are achieved by repeating the three learning experiments ($N = 1, 2, 10$), but assigning to each agent a different initial guess K_n^0 and a different initial steady-state condition x_{ss}^n. For each $n = 1, \ldots, N$, the initial guess K_n^0 is obtained by uniformly sampling its component in the hypercube $[-0.1,\, 0.1]^7$, while the initial condition $x_{ss}^n = [y_{ss}^n,\, \ldots,\, y_{ss}^n,\, u_{ss}^n,\, \ldots,\, u_{ss}^n]'$ is built as in (2.2), by sampling each different $u_{\mathrm{ss}}^n \sim \mathcal{U}([-100,\, 100])$ and considering the associated steady-state output y_{ss}^n. Regarding the global law convergence, the improvement is shown by Figure 72 and Table 30, while observations

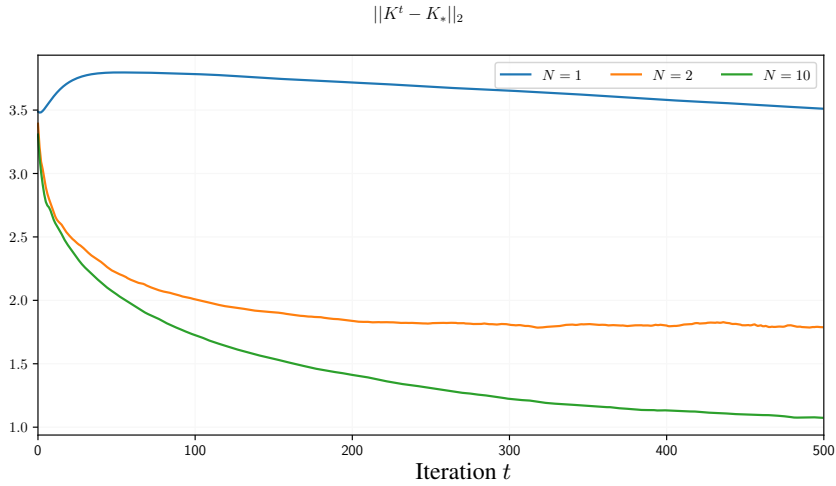

Figure 72: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, different initial conditions and initial guesses $\{x_{ss}^n, K_n^0\}_{n=1}^N$, convergence of the global parameters $\{K^t\}_t$ to K_*.

learning	N	$\mathrm{T}_2(K^t)$	$\mathrm{D}_2(K^t)$
individual	1	> 500	–
collaborative	2	104	1.83
collaborative	10	57	1.34

Table 30: Consensus-based collaborative learning in noisy conditions, varying number of agents, different initial conditions and initial guesses $\{x_{ss}^n, K_n^0\}_{n=1}^N$, performance indices related to the global law convergence.

on the convergence and online behavior of K_1^t can be derived from Figure 73-74 and Table 31. Once more, the collaborative approach strongly benefits from the joint information. The different agents are trained following different reference signal, from different initial states, starting from different initial guesses. Hence, each agent's contribution consists in information gathered from different exploration paths. Separately, the agents do not have access to enough information in order to speed up the search for the optimal parameters.

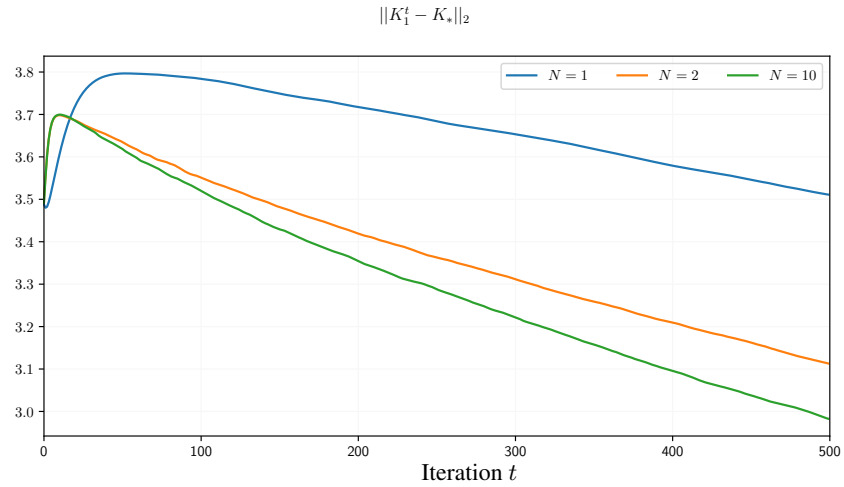

Figure 73: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, different initial conditions and initial guesses $\{x_{ss}^n, K_n^0\}_{n=1}^N$, convergence of the local parameters $\{K_1^t\}_t$ of Agent 1 to the optimal value K_*.

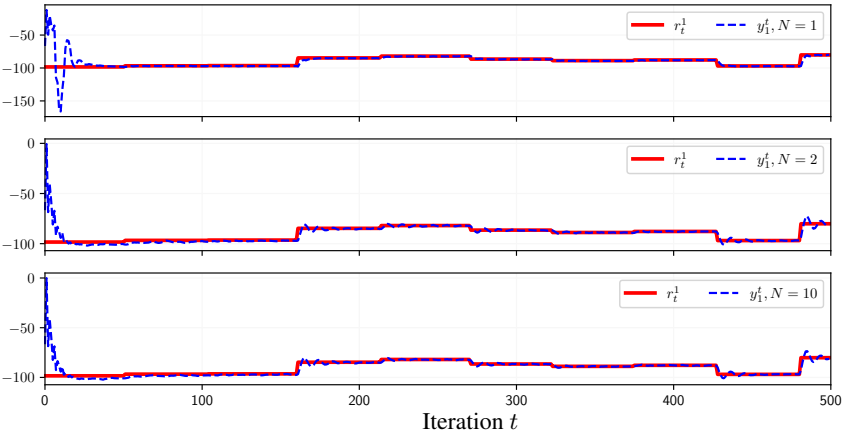

Figure 74: Consensus-based collaborative learning in noisy conditions, $N = 1, 2, 10$ agents, different initial conditions and initial guesses $\{x_{ss}^n, K_n^0\}_{n=1}^N$, online output tracking of an a-priori unknown exploratory reference signal r_t^1 by Agent 1.

learning	N	$\mathrm{T}_{3.2}(K_1^t)$	$\mathrm{D}_{3.2}(K_1^t)$	tracking cost Agent 1
individual	1	> 500	$-$	$7.83 \cdot 1e+7$
collaborative	2	411	3.16	$2.16 \cdot 1e+7$
collaborative	10	318	3.09	$1.98 \cdot 1e+7$

Table 31: Consensus-based collaborative learning in noisy conditions, varying number of agents, different initial conditions and initial guesses $\{x_{ss}^n, K_n^0\}_{n=1}^N$, performance indices related to the local law K_1^t convergence and to Agent 1 online performance.

5.6 Trust-based collaborative learning

The formulation proposed in the previous sections allows us to explicitly exploit the analogies between the processes and their connection to the cloud, in order to optimize the control policies directly from data. Considering the same setup, we now present a second method to achieve the same result, based on a different definition of global policy. Instead of considering as global policy the one obtained through consensus among the local ones, here we rely on a set of weights, based on local performances or other factors of interest, in order to individuate how "trustworthy" are the local policies and improve the quality of the shared information. This second method, indicated as trust-based collaborative Optimal Policy Search, embeds again the OPS approach within a scheme based on ADMM (Boyd et al., 2011), to allow the agents to share only the surrogate policy, while retaining their states, actions, and rewards. Differently from the consensus-based collaborative Optimal Policy Search, though, the trust-based method does not impose hard constraints on the policy of each agent. Rather, it softly steers the agents according to the following formulation, i.e.,

$$\min_{\{\pi_n\}_n} \sum_{n=1}^{N} \Big(\mathbb{E}_W[J^n](\pi_n) + \frac{\gamma_n}{2} \|\phi(\pi_n) - \sum_{m=1}^{N} w_m \phi(\pi_m)\|_2^2 \Big). \tag{5.18}$$

As in (5.2), the (known) function ϕ selects the characteristics that should be shared by all the local policies, considering the correspondences among them, again with the rationale that plants sharing the same dynamics and having similar optimization goals should share similar optimal policies as well.

Problem 5.18 represents a collaborative learning problem among N agents, where for agent n the cost $\mathbb{E}_W[J^n](\pi_n)$ of the local policy π_n is augmented by an additional term, weighted by a tunable parameter $\gamma_n > 0$. Such term can be interpreted as a softened version of a constraint, steering each local law π_n to the weighted average over the policies of the N agents. It is worth remarking that γ_n can be equal for all systems, i.e. $\gamma_n = \gamma$ for each n.

The weights $\{w_n\}_{n=1}^N$ are customizable by the user, and chosen to be approximate indicators of the "level of trust" on the policy of each agent. In general we consider sets of weights such that $w_n \geq 0$, $\sum_{n=1}^N w_n = 1$. Given that the policies are updated in time, and possibly at each time step, time-varying weights $\{w_n^t\}_{n=1}^N$ can be chosen, to account for up-to-date information on the local policy and the parameters γ_n can vary in time as well, (i.e., we can consider γ_n^t at iteration t).

The proposed formulation, like the one introduced in (5.2), allows us to explicitly account for the similarities between the N agents in the computation of the local policies, while enabling one to explicitly account for the performance of each agent through the weights $\{w_n\}_{n=1}^N$, so as to steer the fleet of systems towards the behavior of the better ones.

By considering the policy parameterization in (5.3) and once again the construction in Section 2.1, we approximate problem (5.18) as

$$
\begin{aligned}
&\min_{\{H_n\}_n} \sum_{n=1}^N \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho_n(s_\ell^n,\, p_\ell^n,\, u_\ell^n) + \rho_L^n(s_L^n,\, p_L^n) \Big] + \frac{\gamma_n}{2} \|K_n - \sum_{m=1}^N w_m K_m\|_2^2, \\
&\text{such that} \quad u_\ell^n = \pi_{H_n}^n(s_\ell^n,\, p_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \\
&\qquad\qquad s_{\ell+1}^n = h(s_\ell^n,\, p_\ell^n,\, u_\ell^n,\, d_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \\
&\qquad\qquad d_\ell^n \sim D_\ell^n, \quad \ell = 0, 1, \ldots, L-1, \\
&\qquad\qquad p_\ell^n \sim P_\ell^n, \quad \ell = 0, 1, \ldots, L, \\
&\qquad\qquad s_0^n \sim S_P^n, \quad n = 1, \ldots, N,
\end{aligned}
\tag{5.19}
$$

where, as previously done, we indicate $\phi(H_n) = K_n$ for each $n = 1, \ldots, N$.

5.7 Trust-based collaborative OPS

In order to tackle problem (5.19) in practice, we propose an ADMM-based scheme to learn the local policies, that allows us to fully exploit the computational power of the cloud and the resources available either on board of the single agents or individually allocated on the cloud for each system, bypassing the lack of separability of the problem cost over the agents.

To this end, problem (5.19) is reformulated as

$$\begin{aligned}
\min_{\{H_n, Z_n\}_n} \sum_{n=1}^{N} \mathbb{E}\Big[\sum_{\ell=0}^{L-1} \rho_n(s_\ell^n,\, p_\ell^n,\, u_\ell^n) + \rho_L^n(s_L^n,\, p_L^n) \Big] + \frac{\gamma_n}{2} \|K_n - \sum_{m=1}^{N} Z_m\|_2^2, \\
\text{such that} \quad u_\ell^n = \pi_{H_n}^n(s_\ell^n,\, p_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \\
s_{\ell+1}^n = h(s_\ell^n,\, p_\ell^n,\, u_\ell^n,\, d_\ell^n), \quad \ell = 0, 1, \ldots, L-1, \\
Z_n = w_n K_n, \quad n = 1, \ldots, N, \\
d_\ell^n \sim D_\ell^n, \quad \ell = 0, 1, \ldots, L-1, \\
p_\ell^n \sim P_\ell^n, \quad \ell = 0, 1, \ldots, L, \\
s_0^n \sim S_P^n, \quad n = 1, \ldots, N,
\end{aligned} \tag{5.20}$$

where the auxiliary variables $\{Z_n\}_{n=1}^N$ are introduced to decouple the original problem. The associated Lagrangian in scaled form is given by:

$$\mathcal{L}(\{H_n, Z_n, \upsilon_n\}_{n=1}^N) \;=\; \sum_{n=1}^{N} \mathcal{L}_n(H_n,\, \{Z_m\}_m,\, \upsilon_n), \tag{5.21}$$

where $\mathcal{L}_n$ indicates the equation

$$\begin{aligned}
\mathcal{L}_n(H_n,\, \{Z_m\}_m,\, \upsilon_n) \;=\; \mathbb{E}_W[J_L^n](H_n) \;+ \\
+\; \frac{\gamma_n}{2} \|K_n - \sum_{m=1}^{N} Z_m\|_2^2 \;+\; \frac{\beta}{2} \|w_n K_n - Z_n \;+\; \upsilon_n\|_2^2.
\end{aligned} \tag{5.22}$$

In the Lagrangian, $\beta > 0$ is a tunable penalty parameter, $\phi(H_n) = K_n$ for each n, and $\{\upsilon_n\}_{n=1}^N$ are the normalized Lagrange multipliers associated with (5.20).

Accordingly, the ADMM steps needed to solve the considered policy search problem at time step t are:

$$H_n^t(i+1) = \arg\min_{H_n} \; \mathcal{L}_n(H_n, \{Z_m^t(i)\}_m, \upsilon_n^t(i)) = \tag{5.23a}$$

$$= \arg\min_{H_n} \; \mathbb{E}_W[J_L^n](H_n) \; + \; \frac{\gamma_n}{2}\|K_n - \sum_{m=1}^{N} Z_m^t(i)\|_2^2 \; +$$

$$+ \; \frac{\beta}{2}\|w_n K_n - Z_n^t(i) \; + \; \upsilon_n^t(i)\|_2^2, \qquad n = 1, \ldots, N$$

$$\{Z_m^t(i+1)\}_{m=1}^{N} = \arg\min_{\{Z_m\}_{m=1}^{N}} \sum_{n=1}^{N} \Big[\frac{\gamma_n}{2}\|K_n^t(i+1) - \sum_{m=1}^{N} Z_m\|_2^2 + \tag{5.23b}$$

$$+ \frac{\beta}{2}\|w_n K_n^t(i+1) - Z_n + \upsilon_n^t(i)\|_2^2, \Big]$$

$$\upsilon_n^t(i+1) = \upsilon_n^t(i) + \left(w_n K_n^t(i+1) - Z_n^t(i+1)\right), \quad n = 1, .., N, \tag{5.23c}$$

where $i \in \mathbb{N}$ denotes the ADMM iteration.

As it can be seen by looking at (5.23a), the local parameters can be computed separately. Instead, the auxiliary variables in (5.23b) have all to be retrieved at once. In order to handle this, let us then focus on the second ADMM step. Problem (5.23b) can be recast as:

$$\min_{\substack{\{Z_m\}_{m=1}^{N} \\ \bar{Z}}} \sum_{n=1}^{N} \Big[\frac{\gamma_n}{2}\|K_n^t(i+1) - \bar{Z}\|_2^2 + \tag{5.24}$$

$$+ \; \frac{\beta}{2}\|w_n K_n^t(i+1) - Z_n + \upsilon_n^t(i)\|_2^2 \Big],$$

$$\text{such that} \quad \bar{Z} = \sum_{m=1}^{N} Z_m.$$

As in (Boyd et al., 2011), by fixing $\bar{Z}$ it can be easily proven that the local auxiliary variables are equal to $Z_n = w_n K_n^t(i+1) + \upsilon_n^t(i)$, for $n = 1, \ldots, N$.

This further implies that

$$\bar{Z} = \sum_{n=1}^{N} Z_n = \bar{K}_w^t(i+1) + N\bar{v}^t(i) \tag{5.25}$$

where we indicate

$$\begin{aligned} \bar{K}_w^t(i+1) &\doteq \sum_{n=1}^{N} w_n K_n^t(i+1), \\ \bar{v}^t(i) &\doteq \frac{1}{N}\sum_{n=1}^{N} v_n^t(i). \end{aligned} \tag{5.26}$$

By exploiting (5.25) - (5.26) , the constrained problem in (5.24) is equivalent to the following unconstrained one on the variable $\bar{Z}$:

$$\begin{aligned} \min_{\bar{Z}} \sum_{n=1}^{N} \frac{\gamma_n}{2} \| K_n^t(i\ +1) - \bar{Z} \|_2^2 + \\ + \frac{\beta}{2} \| \frac{1}{N} \left(\bar{K}_w^t(i+1) - \bar{Z} \right) + \bar{v}^t(i) \|_2^2. \end{aligned} \tag{5.27}$$

We can thus reduce the number of auxiliary variables, directly searching for $\bar{Z}$ instead. By shifting towards the optimization of $\bar{Z}$, once $\bar{Z}^t(i+1)$ has been computed via (5.27), we also need to change the update of the Lagrange multipliers, using the equalities in (5.26), as

$$v_n^t(i+1) = \bar{v}^t(i) + \frac{1}{N} \left(\bar{K}_w^t(i+1) - \bar{Z}^t(i+1) \right)$$

for each $n = 1, \ldots, N$. From such update is observable that $v_n^t(i+1)$ are invariant with respect to n, coinciding with the average $\bar{v}^t(i+1)$ defined in (5.26).

The original ADMM-based scheme (5.23) reduces to

$$H_n^t(i+1) = \arg\min_{H_n} \ \mathbb{E}_W[J_L^n](H_n) \ + \ \frac{\gamma_n}{2}\|K_n - \bar{Z}^t(i)\|_2^2 + \tag{5.28a}$$
$$+ \frac{\beta}{2}\|w_n(K_n - K_n^t(i)) + \frac{1}{N}(\bar{Z}^t(i) - \bar{K}_w^t(i)) + \bar{v}^t(i)\|_2^2,$$
$$n = 1, .., N$$

$$\bar{K}_w^t(i+1) = \sum_{n=1}^{N} w_n K_n^t(i+1) \tag{5.28b}$$

$$\bar{Z}^t(i+1) = \arg\min_{\bar{Z}} \ \sum_{n=1}^{N} \frac{\gamma_n}{2}\|K_n^t(i+1) - \bar{Z}\|_2^2 + \tag{5.28c}$$
$$+ \frac{\beta}{2}\|\frac{1}{N}\left(\bar{K}_w^t(i+1) - \bar{Z}\right) + \bar{v}^t(i)\|_2^2,$$

$$\bar{v}^t(i+1) = \bar{v}^t(i) + \frac{1}{N}\left(\bar{Z}^t(i+1) - \bar{K}_w^t(i+1)\right). \tag{5.28d}$$

The problem in (5.28c) can be explicitly solved, with the closed-form expression for $\bar{Z}$ given by:

$$\bar{Z}^t(i+1) = \frac{N\left[\left(\sum_{n=1}^{N}\gamma_n K_n^t(i+1)\right) + \frac{\beta}{N}\bar{K}_w(i+1) + \beta\bar{v}^t(i)\right]}{N(\sum_{n=1}^{N}\gamma_n) + \beta}. \tag{5.29}$$

The ADMM steps are carried on till a user-defined stopping criterion is met, i.e., when $\| \bar{Z}^t(i+1) - \bar{K}_w^t(i+1) \|_2^2$ is smaller than a set treshold, or the number of steps exceeds the maximum number of iterations permitted. As it can be noticed, the parameters of the local policies can be computed separately according to (5.28a). This operation can be either performed on board of each agent, if the computational power locally available is sufficient, or on resources allocated for each agent on the cloud. Then, the agents share with a central unit their parameters and local weights to update the auxiliary variable as in (5.28c) and the weighted average (5.28b). Such values are used to update the Lagrange multipliers (5.28d). The updated variables are then broadcast back to the agents, as

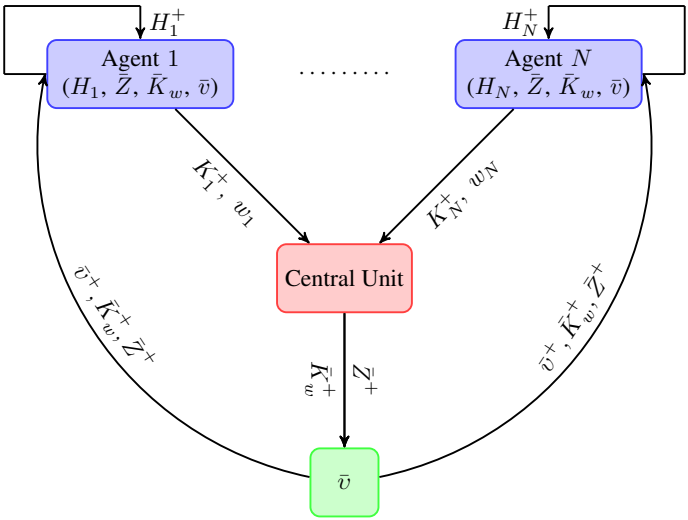

Figure 75: Transmission scheme over an ADMM iteration.

summarized in Figure 75 (where t and i are omitted for simplicity). The Algorithm flow follows the phases of the consensus-based collaborative OPS method, described in Section 5.3 by Algorithms 9-10: in this case, though, Algorithm 9 is substituted by Algorithm 11, containing the steps to be performed in the cloud associated with the trust-based collaborative OPS method.

5.8 Example: trust-based collaborative learning

The described trust-based collaborative Optimal Policy Search approach is now applied on the LTI example described in Section 5.5, considering the noisy case characterized by noisy output of (3.4), i.e., $\sigma_y = 0.1$. We tune the trust-based OPS by setting $\beta = 1$ and $\gamma_n = \gamma = 1/N$, and we employ the same parameters described in Section 5.5. We define time-varying weights w_n^t in order to assess the level of trust on each agent, based on the different local performance of each agent with respect to its tracking task and on the distance of each local policy from the global parameters $\bar{Z}^t$.

Algorithm 11 Trust-based collaborative OPS - Cloud actions at step t

Input: Last measurements $\{y_{t+1}^n,\, x_t^n,\, u_t^n\}_{n=1}^N$ for each agent.
States and models histories $\{X_{(n)}(t-1),\, \Theta_{(n)}(t-1)\}_{n=1}^N$.
Previous values $\{H_n^{t-1}\}_{n=1}^N$, $\bar{Z}^{t-1}$, $\bar{v}^{t-1}$.

Output: Updated parameters $\{H_n^t\}_{n=1}^N$, $\bar{Z}^t$, $\bar{v}^t$.

for $n = 1, \ldots, N$ **do**
  $X_{(n)}(t) = X_{(n)}(t-1) \cup \{x_t^n\}$;
  update Θ_n^t based on $(y_{t+1}^n, x_t^n, u_t^n)$;
  compute w_n^t;
  $\Theta_{(n)}(t) = \Theta_{(n)}(t-1) \cup \{\Theta_n^t\}$;
  set $H_n^t(0) = H_n^{t-1}$, $K^t(0) = K^{t-1}$, $\lambda_n^t(0) = \lambda_n^{t-1}$;
end for
while $(i = 0\,;\ i < i_{max}$ or **not**(termination-cryterion)$\,;\ i++)$ **do**
  for $n = 1, \ldots, N$ **do**
    set $H_n^0 = H_n^t(i)$
    for $m = 1, \ldots, M$ **do**
      OPS/OSPS iteration applied to pb. (5.28a) $\rightarrow$ H_n^m ;
    end for
    $H_n^t(i+1) = H_n^M$;
  end for
  compute $\bar{K}_w^t(i+1) = \sum_{n=1}^N w_n K_n^t(i+1)$
  compute $\bar{Z}^t(i+1)$ as in (5.29);
  compute $\bar{v}^t(i+1)$ as in (5.28d);
  evaluate termination-cryterion;
end while
$H_n^t = H_n^t(\text{last})$, $\bar{Z}^t = \bar{Z}^t(\text{last})$ and $\bar{v}^t = \bar{v}^t(\text{last})$

We encode such definition of trust as

$$w_n^t = 0.5 \cdot w_n^p(t) + 0.5 \cdot w_n^d(t), \tag{5.30}$$

that is, the weights w_n^t defined in (5.30) are the average of the following weights:

$$w_n^p(t) = P_n^t / \Sigma_n P_m^t, \quad \text{such that} \quad P_n^t = 1 - \frac{p_n^t}{\Sigma_m p_m^t}, \tag{5.31}$$

$$w_n^d(t) = D_n^t / \Sigma_n D_m^t, \quad \text{such that} \quad D_n^t = 1 - \frac{d_n^t}{\Sigma_m d_m^t}. \tag{5.32}$$

In particular the weights $w_n^p(t)$ in (5.31) are based on the index p_n^t, evaluating the performance of agent n with respect to the others as

$$p_n^t = \frac{c_n^t}{\Sigma_m c_m^t} / \frac{t_n^t}{\Sigma_m t_m^t},$$

where $c_n^t$is the sum of the last $n_w = 10$ stage costs, and on t_n^t, defined as

$$t_n^t = \| r_n(t - n_w + 1) - y_n(t - n_w + 1) \|_2^2 + \sum_{j=t-n_w+2}^{t} \| r_n(j) - r_n(j-1) \|_2^2 .$$

The indices c_n^t and t_n^t describe the quality of the local performance of the n-th agent and the complexity of the n-th local task, respectively, over a time window of n_w steps. On the other hand, the weights $w_n^d(t)$ in (5.32) are based on the distance $d_n^t = \| K_n^t - \bar{Z}^t \|_2^2$.

The definition (5.30) expresses mathematically the intuition that the higher the trust on agent n is, the lower is the cost of its performance, and that we should consider each performance in relation to the difficulty to the assigned task. The weights (5.30) are easy to interpret, and can be compactly rewritten as

$$w_n^t = \frac{1}{2(N-1)} \left(2 - \frac{p_n^t}{\Sigma_m p_m^t} - \frac{d_n^t}{\Sigma_m d_m^t} \right). \tag{5.33}$$

In the current setup p_n^t, t_n^t and d_n^t are computed locally and transmitted to the central unit where $\{w_n^t\}_n$ is finally obtained, avoiding the agents to share their performance index. As additional precaution, we assign weight zero to agents whose local policy is unstable over the last n_w steps and to the agents that are "outliers" with respect to the majority, i.e., agents such that $d_n^t \geq 0.5 \sum_m d_m^t$ or $p_n^t \geq 0.5 \sum_m p_m^t$. Such agents are also disregarded in the normalization of c_n^t, t_n^t, and d_n^t.

To test the capabilities of the approach, we train $N = 4$ agents while making them perform 4 different online tracking tasks: Agent 1 is trained while tracking r_t^1 taking values in $[-100, -50]$, Agent 2 is trained while tracking r_t^2 taking values in $[-50, 0]$, Agent 3 is trained while tracking r_t^3 taking values in $[0, 50]$, and Agent 4 is trained while tracking r_t^4 taking values in $[50, 100]$.

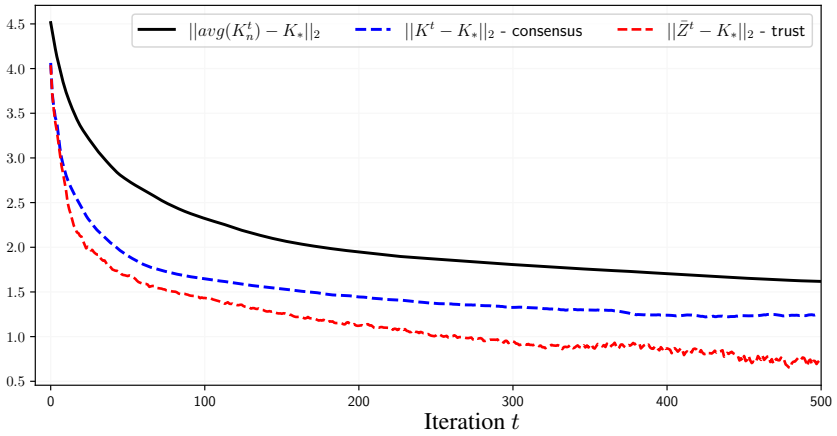

Figure 76: Trust-based collaborative OPS in noisy conditions, convergence of the global parameters $\bar{Z}^t$ to the optimal value K_*. Comparison with the convergence of the consensus-based collaborative OPS global parameters K^t and the average of the local parameters obtained by learning individually through OPS.

learning		T_2	D_2
individual		177	1.78
collaborative	consensus	44	1.41
collaborative	trust	24	1.10

Table 32: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, performance indices related to the global laws.

We compare the local policies, global policies and performances obtained via the proposed sharing strategy with the ones resulting from the application of OPS for the individual learning and the ones obtained by applying the multi-agent consensus method introduced in Section 5.3.

Figure 76 shows that the global law $\bar{Z}^t$ obtained in the trust-based case results to be more informative than the consensus-based one, indicated as K^t. This is mirrored as well by the performance indices in Table 32.

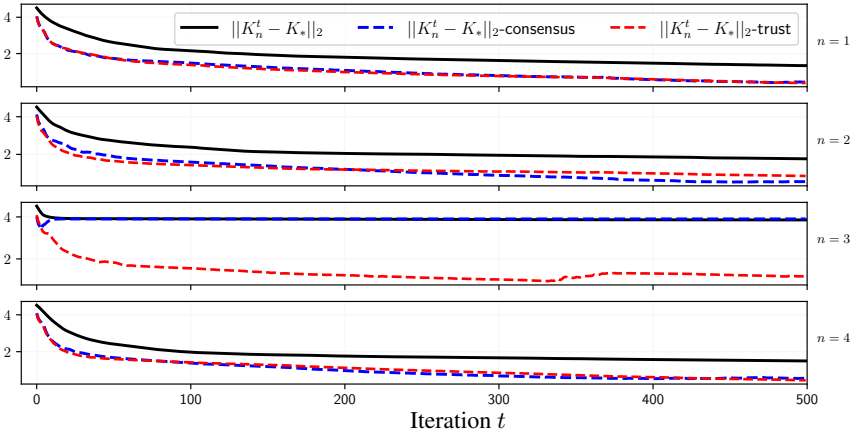

Figure 77: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, convergence of the local parameters K_n^t to the optimal value K_*. Comparison with the consensus-based collaborative local parameters convergence and the average of the local parameters obtained by learning individually through OPS.

Observing both Figure 76-77, we can infer the beneficial effect of incorporating the concept of trustworthy agents on the quality of the global laws, and of the use of a better global law on the convergence of the local ones. Indeed, considering the convergence of Agent 3 in Figure 77, we can observe that both the individual OPS method and the consensus-based collaborative OPS seem to get trapped into a local minimum, differently from the other agents.

Knowing that the consensus-based global law K^t is obtained by constraining all the different agents to "agree" on a shared consensus policy, it is as well clear that Agent 3 contributes to K^t by slowing down its convergence to the optimal value, and hence interfering with the quality of information exploited by the group of agents. Instead, the trust-based method drives the local parameters of Agent 3 towards the global optimum, by exploiting the global knowledge collected in $\bar{Z}^t$. As a chain effect, the local parameters K_3^t avoid to divert the convergence of $\bar{Z}^t$, making it so that the whole group of agents exploits a better global law than the one available in the consensus-based scenario.

learning		$T_2(K_3^t)$	$D_2(K_3^t)$	tracking cost Agent 3
individual		> 500	$-$	$1.12 \cdot 1e+7$
collaborative	consensus	> 500	$-$	$1.45 \cdot 1e+9$
collaborative	trust	35	1.28	$7.19 \cdot 1e+4$

Table 33: Trust-based collaborative learning in noisy conditions, $N = 4$ agents, performance indices associated with Agent 3.

The performance indices $T_2(K_3^t)$ and $D_2(K_3^t)$ associated with Agent 3 are included in Table 33 and mirror the previous observations about the convergence of the local policy K_3^t. The same Table contains as well the online tracking cost achieved by Agent 3 while learning, showing how relevant it is in terms of performance to enhance the convergence and avoiding local minima.

Figures 78 - 79 - 80 - 81 show the improvement in the online tracking of the four agents, when trained using the trust-based collaborative OPS, in comparison with the performance obtained by the consensus-based collaborative OPS and the one obtained by learning individually through OPS. We can observe that exploiting $\bar{Z}^t$ is really more advantageous than exploiting K^t from the performance point of view and for all the agents (not exclusively for Agent 3), as confirmed as well by the average costs in Table 34.

In summary, the trust-based Optimal Policy Search method seems to ulteriorly improve the convergence speed in the described multi-agent scenario, and to be better suited than the consensus-based scheme in handling the collaborative learning of tracking policies in case complex non-convex optimization functions are considered.

learning		average tracking cost
individual		$3.74 \cdot 1e+6$
collaborative	consensus	$3.63 \cdot 1e+8$
collaborative	trust	$2.23 \cdot 1e+5$

Table 34: Trust-based collaborative learning in noisy conditions, $N = 4$ agents, average online tracking cost.

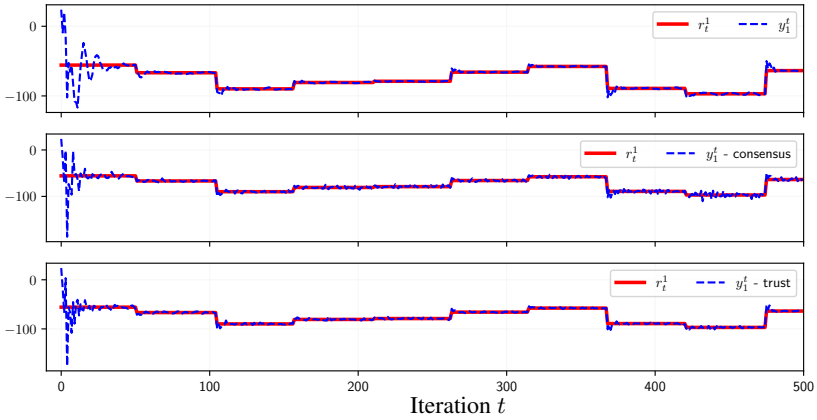

Figure 78: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal r_t^1 by Agent 1.

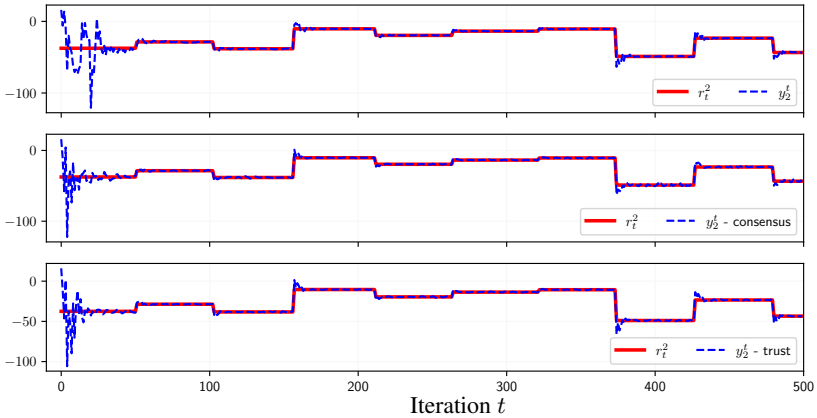

Figure 79: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal r_t^2 by Agent 2.

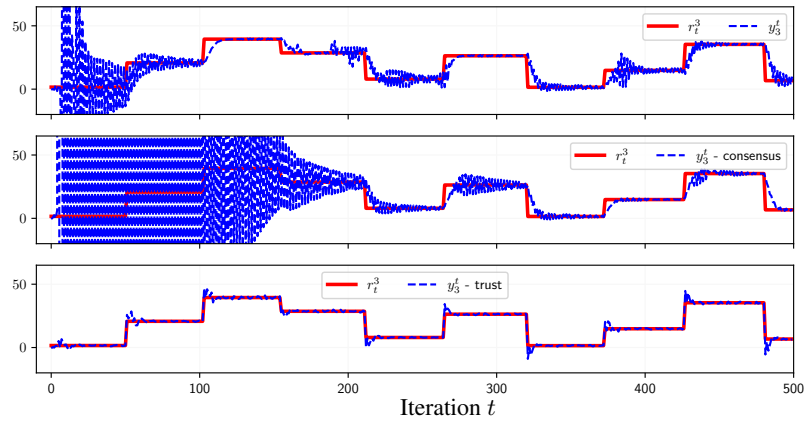

Figure 80: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal r_t^3 by Agent 3.

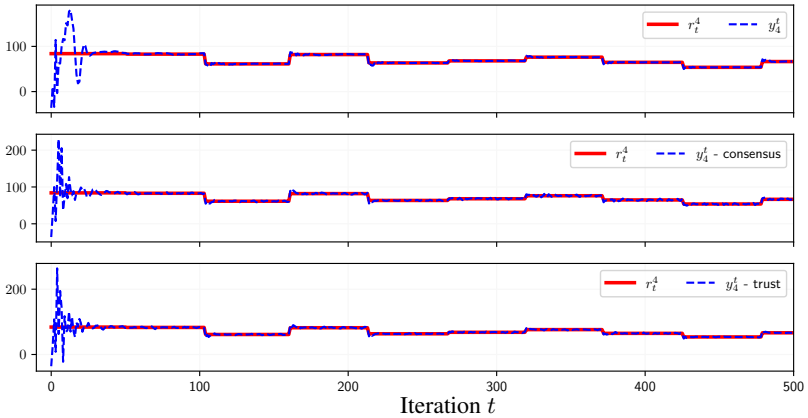

Figure 81: Trust-based collaborative OPS in noisy conditions, $N = 4$ agents, online output tracking of an a-priori unknown exploratory reference signal r_t^4 by Agent 4.

Chapter 6

Conclusions

This thesis addressed the problem of learning feedback controllers from experimental data, considering the plant to be controlled as a black-box source of input and output streams. New policy search methods for the learning of deterministic control policies using Stochastic Gradient Descent were presented, applicable both offline and online. The proposed methods are based on a combination of the data-driven policy search framework with some elements of the model-based scenario. The introduction of a cheap model-based scheme was motivated by the desire to mitigate some of the drawbacks presented by the purely data-driven approach. On one side the method requires a low modeling effort, as compared to the typical identification and model-based control design scenario. On the other side, it reduces risky explorative operations on the plant and mitigates the variance on the gradient approximations, that are characteristic of pure data-driven policy search methods, due to the use of stochastic policies and random sampling of trajectories.

In Chapter 2 a data-driven gradient approximation approach based on states and local linear models history was introduced. Based on this approach, a policy search algorithm for the learning of smooth feedback controllers (indicated as OPS method) was detailed, both in the offline and online scenario. In the online scenario, a local stabilization heuristic was proposed for linear feedback controller synthesis, while remaining faithful to the "black-box" hypothesis over the plant dynamics.

In Chapter 3 the results of the application of the OPS method to learn policies for the tracking of a-priori unknown time-varying setpoints were included. An assisted control strategy for the online learning of tracking policies was introduced in order to leverage between exploration and safety. The OPS method is tested both online and offline on four different scenarios:

1. on an LTI system, learning a linear feedback controller, with local linear models tuned with the real system orders (noiseless and noisy case).

2. On an LTI system, learning linear feedback controllers, with local linear models tuned with reduced system orders (noiseless and noisy case).

3. On a nonlinear system, learning a linear feedback controller (noisy case).

4. On a nonlinear system, learning a nonlinear feedback controller (noisy case).

When the system generating the data is linear and the local linear models belong to the class of the real plant (scenario 1) the results obtained demonstrate how the OPS method converges to the known optimal linear feedback law. The tests were performed considering noise at different intensities, different initial conditions for the plant and starting the learning process from different initial guesses for the policy parameters. If the system dynamics does not belong to the same class of the local linear models (scenarios 2 - 3 - 4) the OPS method was shown to be capable of synthesizing good controllers, by converging to policies capable of successfully tracking a-priori unknown references. In particular, two different linear parameterizations of the feedback controller were considered in scenario 2, both exploiting a reduced feedback state. The first one was shown to achieve tracking performance very close to the one of the known optimal controller both when trained offline and online. The second parameterization was synthesized offline in order to be fairly compared with two VRFT-derived controllers (Campi, Lecchini, and Savaresi, 2002), using as reference model the optimal behavior (based on the true dynamics of the system) and a data-driven derived behavior, respectively. The OPS method was shown to outperform both the VRFT controllers, although synthesized from the same dataset, demonstrating a more efficient use of poorly informative data. Regarding the online setting,

in every scenario it was shown that the method is capable of effectively performing output tracking of a-priori unknown reference signals while learning. In particular, the first controller synthesized online in scenario 2 was shown to achieve an online tracking behavior comparable to the one obtained by DeePC (Coulson, Lygeros, and Dörfler, 2019) in an unconstrained setting on the same task, after an initial phase of online learning.

In Chapter 4 the OSPS algorithm for the learning of non-smooth (hybrid) controllers with a user-defined number of modes was described. The presented approach learns both the set of control laws and a polyhedral switching law, in an online or offline setup. Given that the policy parameterization is not smooth, additional care was taken to avoid the computation of the gradients necessary for the local controllers update in areas of the parameters space where the performance index is not continuous and hence not differentiable. Numerical examples demonstrate that the approach performs well when applied to control processes characterized by hybrid or nonlinear dynamics, outperforming control laws that are single-mode (no switching) or multi-mode but with the switching law defined a priori.

Finally, in Chapter 5 two collaborative learning algorithms (consensus-based and trust-based) were described, incorporating the proposed OPS method with knowledge-sharing strategies in a cloud-aided setup, in order to improve the learning capabilities of a multi-agent system with structural similarities. In both the approaches, the agents are allowed to share surrogate of their experiences, while privately retaining their states, actions, and rewards, that could be sensitive for privacy reasons. To verify the benefits of sharing information on the learning performance, the two methods were tested on the LQR problem. Both methods improve the convergence rate of the different agents to the known optimal policy in comparison with the individual learning via OPS. In particular, the trust-based approach was shown to be faster in convergence than the consensus-based one. Moreover, the trust-based method appeared to be more resilient against the loss of convergence that might result from the randomness of sampling, the differences in initial guesses and the non-convexity of the cost function with respect to the policy parameters.

6.1 Future directions

This work was focused on the design of the proposed algorithms and on the analysis of their performance in different scenarios. Future research will aim at

- providing the proposed approach with a theoretical background and additional analytic insight on its properties. In particular, we will aim at obtaining theoretical conditions on the gradient error, in order to guarantee convergence to a (possibly not globally) optimal policy, seeking to link the precision of the local models with the convergence rate of the method. Additionally, future investigations will include a statistical analysis on the approximated gradients variance: in the current work the reduction in variance is evidenced by the good rate of convergence achieved in the numerical examples, but a formal study, in relation with the increase of the stochasticity and complexity in the plant dynamics, is indeed necessary to understand the limitations of the approach. For the same reasons, an analysis of the classes of systems on which the method can be successfully applied will be important. In order to establish the efficacy of the method, we will as well compare it with other data-driven and model-based approaches. In particular the performance of the approach will be compared with the policy search methods based on a Locally Weighted Bayesian Regression model (Atkeson, Moore, and Schaal, 1997), considered as the closest model-based methods with respect to the one proposed in this thesis.

- Additional effort will be spent in improving the local models, while remaining in the linear paradigm in order to keep the modeling effort low. In particular, a first idea to be tested is to enrich the linear models with linear multi-step forward estimations, to be employed in the gradient computations at different stages of the cumulative cost. Another idea is to improve the precision of the local model by employing the one-step ahead Kalman Filter innovation error measured over the stream of data considered during the learning. The error could be used to give a local estimate of the variance of the disturbances to be sampled in the mini-batch creation, so that they can be employed as an alias for the model error.

- The stochastic gradient based optimization procedure will be enhanced through the use of second order approximations, like the ones provided by the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm and its stochastic variants, as for instance (Moritz, Nishihara, and Jordan, 2016), investigating both the classic second order approximation method used in L-BFGS and a stochastic approximation based on the local linear models.

- The method will be tailored to solve constrained problems by considering the use of specific policy parameterizations. An interesting approach is to learn through OPS an MPC policy parameterization, which is inherently designed to enforce constraints satisfaction (cf. (Gros and Zanon, 2021)). Moreover, recently developed methods employing safety-enforcing control barrier functions (i.e., (Cheng et al., 2019)) can constitute a useful starting point for the design of a safe-learning architecture, while remaining faithful to the local model approach.

- The OSPS method for the synthesis of hybrid controllers will be extended in order to be capable of synthesizing switching policies characterized by non-polyhedral areas of application. The idea that I am currently developing is based on the substitution of the Voronoi diagrams, used in the previous works as switching law parameterization, with the Hyperbolic Voronoi Diagrams.

- Finally, additional research on the multi-agent collaborative learning scenario is currently being conducted. On one side we are testing the benefits of the sharing approach in more challenging scenarios, as for instance by considering nonlinear dynamics or by employing nonlinear and not completely shared policies. On the other side, we are extending the proposed schemes in order to permit the agents to share information as well on their personal environment representation. Moreover, the handling of asynchronous communication schemes between the agents and the central units will be tackled.

Appendix A

Building non minimal state-space dynamics from the ARX model

This section is dedicated to describe how the approximation of the non-minimal state space dynamics is built from the ARX model

$$y_{t+1} = \Theta_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + d_t$$

introduced in (2.12). First, Θ_t is splitted into $(\Theta_t^x, \Theta_t^u) \in \mathbb{R}^{n_y \times n_x} \times \mathbb{R}^{n_y \times n_u}$ such that

$$y_{t+1} = \Theta_t^x x_t + \Theta_t^u u_t + d_t.$$

This permits us to obtain an approximation of the dynamics of state (2.2), i.e.,

$$x_t = [\, y_t' \ \dots \ y_{t-n_o+1}' \ u_{t-1}' \ \dots \ u_{t-n_i}' \,]' \in \mathbb{R}^{n_x},$$

as $x_{t+1} = A_x\, x_t + B_x\, u_t + D_x\, d_t$ where

$$A_x = \begin{bmatrix} \Theta_t^x \\ A_y \\ \underline{0}_{n_u \times n_x} \\ A_u \end{bmatrix} \in \mathbb{R}^{n_x \times n_x}, B_x = \begin{bmatrix} \Theta_t^u \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{n_x}, D_x = \begin{bmatrix} I_{n_y} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{n_x \times n_y}. \tag{A.1}$$

The matrices

$$A_y = \begin{bmatrix} I_{n_y n_o - n_y} & \underline{0}_{(n_y n_o - n_y) \times n_y} & \underline{0}_{(n_y n_o - n_y) \times n_u n_i} \end{bmatrix} \in \mathbb{R}^{(n_y n_o - n_y) \times n_x}$$

and

$$A_u = \begin{bmatrix} \underline{0}_{(n_u n_i - n_u) \times n_y n_o} & I_{n_u n_i - n_u} & \underline{0}_{(n_u n_i - n_u) \times n_u} \end{bmatrix} \in \mathbb{R}^{(n_u n_i - n_u) \times n_x}$$

in (A.1) take care of selecting the lastly occured $n_i - 1$ input and $n_o - 1$ output from vector x_t, removing the oldest output and input, respectively.

We previously defined the state s_t as a vector composed by the state x_t and a set of additional states z_t. In case the dynamics of z_t are given by $z^+ = f_z(x,\, z,\, p,\,)$, where f_z is known and differentiable, as considerd in (2.17), then they can be linearized around the current values (x_t, z_t, p_t), obtaining

$$z^+ \approx M_x^z \cdot x + M_z^z \cdot z + M_p^z \cdot p,$$

with opportune M_x, M_z and M_p.
The following system is considered

$$s_+ \quad = \quad A_t s + B_t u + E_t p + D_t d, \tag{A.2}$$

where

$$A_t = \begin{bmatrix} A_x & \underline{0} \\ M_x^z & M_z^z \end{bmatrix},\ B_t = \begin{bmatrix} B_x \\ \underline{0} \end{bmatrix},\ E = \begin{bmatrix} \underline{0} \\ M_p^z \end{bmatrix},\ D = \begin{bmatrix} D_x \\ \underline{0} \end{bmatrix}$$

.
In case instead the dynamics of z_t are unkonown and z_t is measured, then the state x_t itself contains z_t, as shown in (2.18), and the local linear model Θ_t is updated according to (2.19): then, it is possible to build matrices that are analogous to the ones in (A.1), and are also containing the approximated dynamics of z_t. In this case $A_t = A_x$, $B_t = B_x$, $E_t = \underline{0}$, $D_t = I$.

The matrix construction described in this section can be similarly reproduced to build a system such as

$$s_+ \quad = \quad A_t s + B_t \Delta u + E_t p + D_t d, \tag{A.3}$$

by using the equality $u_t = u_{t-1} + \Delta u_t$ to split Θ_t into $(\Theta_t^x, \Theta_t^{\Delta u})$ such that $y_{t+1} = \Theta_t^x x_t + \Theta_t^{\Delta u} \Delta u_t + d_t$. The matrix construction (A.1) - (A.2) is then followed.

Appendix B

Proof of Theorem 1

In Chapter 4 we consider controllers represented by a hybrid policy parameterization $\pi_k^{\overline{c}}$ with poligonal switching laws, expressed by (4.3)-(4.9), and we show that, for a fixed sample $\overline{w} = (\,\overline{s}_0,\, \{\overline{p}_\ell\}_{\ell=0}^{L}, \{\overline{d}_\ell\}_{\ell=0}^{L-1}\,)$, and local model $\overline{\Theta}_0$, and a fixed set of centroids $\overline{c}$, the approximated cost function in closed-loop with such parameterized controller (4.6), i.e.,

$$J_{\overline{c}}^{\overline{w}}(K) = \hat{J}_L(K,\, \overline{c},\, \overline{w}) = \sum_{\ell=0}^{L-1} \rho(s_\ell,\, \overline{p}_\ell,\, \pi_K^{\overline{c}}(s_\ell,\, \overline{p}_\ell)) + \rho_L(s_{_L},\, \overline{p}_{_L}),$$

is granted to be differentiable (and hence continuous) over

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \Big\{ K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \Big| \forall\ \ell\ \in \{1,\, \ldots,\, L\} \\ \exists\ m\ \in \{1,\, \ldots,\, M\} \quad (s_\ell^{\overline{w}}(K, \overline{c}),\ \overline{p}_\ell) \in \mathring{R}_m(\overline{c}) \Big\}.$$

Considering that $(s_\ell^{\overline{w}}(K, \overline{c}),\ \overline{p}_\ell) \in R_m(\overline{c})$ is equivalent to

$$s_\ell^{\overline{w}}(K, \overline{c}) \in S_m^\ell(\overline{c}, \overline{w}) \doteq \{s \in SR_\ell(\overline{w}) \mid (s,\, \overline{p}_\ell) \in R_m(\overline{c})\},$$

the set $\mathcal{S}_{\overline{c}}^{\overline{w}}$ can be rewritten as

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \Big\{ K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \Big| \forall\ \ell\ \in \{1,\, \ldots,\, L\} \\ \exists\ m\ \in \{1,\, \ldots,\, M\} \quad s_\ell^{\overline{w}}(K, \overline{c}) \in \mathring{S}_m^\ell(\overline{c}, \overline{w}) \Big\}. \tag{B.1}$$

We observe that, for each ℓ, the regions $\{S^{\ell}_m(\overline{c},\overline{w})\}_m$ are disjoint: if $s \in S^{\ell}_m(\overline{c},\overline{w}) \cap S^{\ell}_n(\overline{c},\overline{w})$, then $(s,\, \overline{p}_\ell) \in \mathring{R}_m(\overline{c}) \cap \mathring{R}_n(\overline{c})$, but the regions $\{R_m(\overline{c})\}_m$ are assumed to be disjoint.

Initially, we prove two lemmas that will be useful for the derivation of the proof of Theorem 1. The first lemma that we are going to prove shows that the regions $\{R^{\overline{c},\overline{w}}_j\}_{j=1}^n$ are a partition of the parameters space.

Lemma 2. *The family of sets* $\{R^{\overline{w},\overline{c}}_j\}_{j=1}^n$ *defined as*

$$R^{\overline{w},\overline{c}}_j = \{K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \mid \Sigma^{\overline{w}}_{\overline{c}}(K) = \chi_j\}$$

with

$$\begin{aligned}\Sigma^{\overline{w}}_{\overline{c}}(K) \;=\; \Big(\, \sigma_{\overline{c}}(s^{\overline{w}}_1,\; \overline{p}_\ell),\; \ldots,\; \sigma_{\overline{c}}(s^{\overline{w}}_L,\; \overline{p}_\ell)\, \Big) \in \mathcal{S}_{M,L}\\ \textit{such that} \quad s^{\overline{w}}_\ell \;\; \textit{defined in (4.7) for} \;\; \ell = 1,\ldots,L,\\ u_\ell = \pi^{\overline{c}}_K(s^{\overline{w}}_\ell,\, \overline{p}_\ell) \;\; \textit{for} \;\; \ell = 0,\ldots,L-1,\end{aligned}$$

is a partition of the parameters space $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M}$.

Proof. To prove this result we start by observing that the sets $R^{\overline{w},\overline{c}}_j$ are two-by-two disjoint, given the univocal definition of the sequence $\Sigma^{\overline{w}}_{\overline{c}}(K)$ associated to K. No set of parameters K can simultaneously be associated to two distinct sequences $\chi_i \in \mathcal{S}_{M,L}$ and $\chi_j \in \mathcal{S}_{M,L}$ such that $\chi_i \neq \chi_j$ for the visit of regions $\{R_m(\overline{c})\}_m$, given the couple $(\overline{c},\, \overline{w})$.
To complete the proof we need to verify that the union of the sets $\{R^{\overline{w},\overline{c}}_j\}_{j=1}^n$ coincides with the whole space $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M}$. In order to do so we consider a generic $K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M}$. K induces a trajectory of states $s^{\overline{w}}_\ell(K,\overline{c})$ for $\ell = 1,\ldots,L$: each of them is obtained by applying $u_k = \pi^{\overline{c}}_K(s^{\overline{w}}_k(K,\overline{c}),\, \overline{p}_k)$ for $k = 0,\ldots,\ell-1$, in composition with (4.7). The regions $R_m(\overline{c})$ are designed to belong to $\mathcal{P}^M$ defined in (4.1), so $R_1(\overline{c}) \cup \ldots \cup R_M(\overline{c}) = \mathbb{R}^{n_s+n_p}$. Each couple $(\, s^{\overline{w}}_\ell(K,\overline{c}),\, \overline{p}_\ell\,)$ hence belongs to one of said regions, that we indicate as $R_{m(\ell)}(\overline{c})$. Based on this construction, it is possible to build a sequence χ such that $\chi(\ell) = m(\ell)$ for $\ell = 1,\ldots,L$. Such χ belongs by construction to the set $\mathcal{S}_{M,L} = \{\chi_1,\ldots,\chi_n\}$, hence corresponding to $\chi = \chi_i$ for some $i \in \{1,\ldots,n\}$. We proved in this way that $K \in R^{\overline{w},\overline{c}}_i$, and hence that $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \subset \cup_{j=1}^n R^{\overline{w},\overline{c}}_j$, proving that the union of $R^{\overline{w},\overline{c}}_j$ corresponds to $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M}$. This proves that $\{R^{\overline{w},\overline{c}}_j\}_{j=1}^n$ is a partition of $\mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M}$. □

The second lemma that we are going to prove employs the partition $\{R_j^{\overline{c},\overline{w}}\}_{j=1}^n$ to reformulate the set $\mathcal{S}_{\overline{c}}^{\overline{w}}$ as a disjoint union of the subsets

$$\mathcal{S}_j = \Big\{ K \in R_j^{\overline{w},\overline{c}} \;\mid\; \forall\, \ell \in \{1,\ldots,L\} \quad s_\ell^{\overline{w}}(K,\overline{c}) \in \mathring{S}^\ell_{\chi_j(\ell)}(\overline{c},\overline{w}) \Big\}. \quad \text{(B.2)}$$

Lemma 3. *The set* $\mathcal{S}_{\overline{c}}^{\overline{w}}$ *defined in* (B.1) *corresponds to*

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \bigcup_{j=1}^{n} \mathcal{S}_j$$

with $\mathcal{S}_j$ *defined in* (B.2).

Proof. The partition $\{R_j^{\overline{w},\overline{c}}\}_{j=1}^n$ is used to build the following disjoint union,

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \bigcup_{j=1}^{n} \mathcal{S}_{\overline{c}}^{\overline{w}} \cap R_j^{\overline{w},\overline{c}}.$$

Based on (B.1) and (4.14),

$$\begin{aligned}
&\mathcal{S}_{\overline{c}}^{\overline{w}} \cap R_j^{\overline{w},\overline{c}} = \\
&= \{K \in R_j^{\overline{w},\overline{c}} \mid \forall\, \ell \in \{1,..,L\} \;\; \exists\, m \in \{1,..,M\} \quad s_\ell^{\overline{w}}(K,\overline{c}) \in \mathring{S}^\ell_m(\overline{c},\overline{w})\} = \\
&= \{K \in R_j^{\overline{w},\overline{c}} \mid \forall\, \ell \in \{1,..,L\} \quad s_\ell^{\overline{w}}(K,\overline{c}) \in \mathring{S}^\ell_{\chi_j(\ell)}(\overline{c},\overline{w})\} = \\
&= \mathcal{S}_j,
\end{aligned}$$

where the second equality holds because on one side $K \in R_j^{\overline{w},\overline{c}}$ implies that for each $\ell = 1,\ldots,L$ the states $s_\ell^{\overline{w}}(K,\overline{c}) \in S^\ell_{\chi_j(\ell)}(\overline{c},\overline{w})$. On the other side, if it exists an $m = 1,\ldots,M$ such that $s_\ell^{\overline{w}}(K,\overline{c}) \in \mathring{S}^\ell_m(\overline{c},\overline{w})$ then $m = \chi_j(\ell)$ because for each ℓ the regions $\{S^\ell_m(\overline{c},\overline{w})\}_m$ are disjoint. This concludes the proof of the lemma. □

The previous lemma is employed to prove Theorem 1, that characterizes the subsets of the parameters space such that the cost function is differentiable for a given couple $(\overline{c},\overline{w})$.

Theorem 1. *Given a couple* $(\overline{c},\overline{w})$*, the set* $\mathcal{S}_{\overline{c}}^{\overline{w}}$ *defined in* (B.1) *is such that*

$$\mathcal{S}_{\overline{c}}^{\overline{w}} = \bigcup_{j=1}^{n} \mathring{R}_j^{\overline{w},\overline{c}}$$

with $R_j^{\overline{w},\overline{c}} = \{K \in \mathbb{R}^{n_1} \times \ldots \times \mathbb{R}^{n_M} \mid \Sigma_{\overline{c}}^{\overline{w}}(K) = \chi_j\}$.

Proof. Lemma 3 ensures that $\mathcal{S}_{\overline{c}}^{\overline{w}} = \bigcup_{j=1}^{n} \mathcal{S}_j$, where

$$\mathcal{S}_j = \Big\{ K \in R_j^{\overline{w},\overline{c}} \;\mid\; \forall\, \ell \in \{1, \ldots, L\} \quad s_\ell^{\overline{w}}(K, \overline{c}) \in \mathring{S}_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w}) \Big\}.$$

In order to prove the enunciated result we start by showing that $\mathcal{S}_j \subseteq \mathring{R}_j^{\overline{w},\overline{c}}$.

Consider $K \in \mathcal{S}_j$. By definition this implies that for all $\ell \in \{1, \ldots, L\}$ it exists an $\epsilon_\ell > 0$ such that $B_\ell = B(s_\ell^{\overline{w}}(K, \overline{c}),\, \epsilon_\ell) \subseteq \mathring{S}_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w})$.
For each $\ell = 1, \ldots, L$ the restriction of $s_\ell^{\overline{w}}(K, \overline{c})$ to $\mathcal{S}_j$ (indicated as $s_\ell^{\overline{w}}\Big|_{\mathcal{S}_j}(K, \overline{c})$) is such that the image $s_\ell^{\overline{w}}\Big|_{\mathcal{S}_j}(\mathcal{S}_j) = \mathring{S}_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w})$ by definition of $\mathcal{S}_j$, and it is continuous because $\pi_k^{\overline{c}}$ is continuous over $\mathcal{S}_{\overline{c}}^{\overline{w}}$ and hence over $\mathcal{S}_j$. We define then

$$C = \bigcap_{\ell=1}^{L} (s_\ell^{\overline{w}}\Big|_{\mathcal{S}_j})^{-1}(B_\ell) \subseteq \mathcal{S}_j.$$

The set C is an open set in $\mathcal{S}_j$, being the intersection of the countersets of the open balls B_ℓ with respect to continuous functions. It contains K by construction, i.e., $K \subseteq C \subseteq \mathcal{S}_j$. Hence $K \in \mathring{R}_j^{\overline{w},\overline{c}}$.

In the previous part of the proof we showed that $\mathcal{S}_j \subseteq \mathring{R}_j^{\overline{w},\overline{c}} \subseteq R_j^{\overline{w},\overline{c}}$. Now we take a generic $K \in R_j^{\overline{w},\overline{c}}$ such that $K \notin \mathcal{S}_j$, and we show that K necessarily belongs to $\partial R_j^{\overline{w},\overline{c}}$, hence proving that $\mathcal{S}_j = \mathring{R}_j^{\overline{w},\overline{c}}$. From $K \notin \mathcal{S}_j$ we known that it exists an $\ell \in \{1, \ldots, L\}$ such that $s_\ell^{\overline{w}}(K, \overline{c}) \notin \mathring{S}_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w})$. $K \in R_j^{\overline{w},\overline{c}}$ implies, though, that $s_\ell^{\overline{w}}(K, \overline{c}) \in S_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w})$. Hence, it exists an $\ell \in \{1, \ldots, L\}$ such that $s_\ell^{\overline{w}}(K, \overline{c}) \in \partial S_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w})$. We define $\overline{\ell}$ as the minimum $\ell \in \{1, \ldots, L\}$ that verifies this condition, i.e.,

$$\begin{aligned} & s_\ell^{\overline{w}}(K, \overline{c}) \in \mathring{S}_{\chi_j(\ell)}^{\ell}(\overline{c}, \overline{w}) \quad \forall \ell \in \{1, \ldots, \overline{\ell} - 1\}, \\ & s_{\overline{\ell}}^{\overline{w}}(K, \overline{c}) \in \partial S_{\chi_j(\overline{\ell})}^{\overline{\ell}}(\overline{c}, \overline{w}). \end{aligned}$$

From this follows that $\forall\, \epsilon > 0$ it exists

$$y_{\overline{\ell}}(\epsilon) \in B(s_{\overline{\ell}}^{\overline{w}}(K, \overline{c}),\, \epsilon) \cap [S_{\chi_j(\overline{\ell})}^{\overline{\ell}}(\overline{c}, \overline{w})]^c = B(s_{\overline{\ell}}^{\overline{w}}(K, \overline{c}),\, \epsilon) \cap S_{m_\epsilon}^{\overline{\ell}}(\overline{c}, \overline{w}),$$

for $m_\epsilon \neq \chi_j(\overline{\ell})$, considering that $\{R_m(\overline{c})\}_m$ is a partition of $\mathbb{R}^{n_s+n_p}$ and hence $\{S_m^{\overline{\ell}}(\overline{c}, \overline{w})\}_m$ is a partition of $SR_{\overline{\ell}}(\overline{w})$. Considering that for each $\epsilon > 0$ it

exists $y_{\overline{\ell}}(\epsilon) \in S^{\overline{\ell}}_{m_\epsilon}(\overline{c},\overline{w})$ with $m_\epsilon \neq \chi_j(\overline{\ell})$ means that $y_{\overline{\ell}}(\epsilon) \in SR_{\overline{\ell}}(\overline{w})$, this is equivalent to

$$\forall\ \epsilon > 0 \quad \exists\ K_\epsilon \in (\ R_j^{\overline{c},\overline{w}}\ )^c \quad \text{such that} \quad y_{\overline{\ell}}(\epsilon) = s^{\overline{w}}_{\overline{\ell}}(K_\epsilon,\overline{c}).$$

Choosing $\epsilon_n = \frac{1}{n}$, the series $\{\ y_{\overline{\ell}}(\epsilon_n)\ \}_n$ by construction converges to $s^{\overline{w}}_{\overline{\ell}}(K,\overline{c})$ for $n \to \infty$. Considering the set $\mathcal{S}^{\overline{\ell}-1} = \bigcup_{k=1}^n \mathcal{S}_k^{\overline{\ell}-1}$ with

$$\mathcal{S}_k^{\overline{\ell}-1} = \Big\{ K \in R_k^{\overline{w},\overline{c}}\ \mid\ \forall\ \ell\ \in \{1,..,\overline{\ell}-1\}\ \ s^{\overline{w}}_{\ell}(K,\overline{c}) \in \mathring{S}^{\ell}_{\chi_k(\ell)}(\overline{c},\overline{w}) \Big\}$$

we notice that $K \in \mathcal{S}_j^{\overline{\ell}-1} \subseteq \mathcal{S}^{\overline{\ell}-1}$. The function $s^{\overline{w}}_{\overline{\ell}}\Big|_{\mathcal{S}^{\overline{\ell}-1}} : \mathcal{S}^{\overline{\ell}-1} \longrightarrow SR_{\overline{\ell}}(\overline{w})$,

$$s^{\overline{w}}_{\overline{\ell}}(K,\,\overline{c}) = A_0^{\overline{\ell}}\,\overline{s}_0 + \sum_{\ell=0}^{\overline{\ell}-1} A_0^{\overline{\ell}-1-\ell}(B_0\,\pi^{\overline{c}}_K(s^{\overline{w}}_{\ell}(K,\,\overline{c}),\,\overline{p}_\ell) + E_0\,\overline{p}_\ell + D\,\overline{d}_\ell),$$

is continuous, being the composition of (4.7) and of the policy $\pi^{\overline{c}}_K$ applied on $s^{\overline{w}}_{\ell}(K,\overline{c}) \in \bigcup_{k=1}^n \mathring{S}^{\ell}_{\chi_k(\ell)}(\overline{c},\overline{w})$ for $\ell \in \{1,\ldots,\overline{\ell}-1\}$, and hence where it is continuous.

For the continuity of $s^{\overline{w}}_{\overline{\ell}}\Big|_{\mathcal{S}^{\overline{\ell}-1}}$, K_ϵ should tend to $K \in R_j^{\overline{c},\overline{w}}$ but for each ϵ $K_\epsilon \notin R_j^{\overline{c},\overline{w}}$, proving that K belongs to the boundary $\partial R_j^{\overline{c},\overline{w}}$. □

Bibliography

Alibekov, Eduard, Jiří Kubalík, and Robert Babuška (2018). "Policy derivation methods for critic-only reinforcement learning in continuous spaces". In: *Engineering Applications of Artificial Intelligence* 69, pp. 178–187. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2017.12.004. URL: https://www.sciencedirect.com/science/article/pii/S0952197617302993.

Anderson, B.D.O. and Y. Liu (1989). "Controller reduction: concepts and approaches". In: *IEEE Transactions on Automatic Control* 34.8, pp. 802–812. DOI: 10.1109/9.29422.

Atkeson, Christopher G., Andrew W. Moore, and Stefan Schaal (Feb. 1997). "Locally Weighted Learning". In: *Artificial Intelligence Review* 11 (1), pp. 11–73. ISSN: 1573-7462. DOI: 10.1023/A:1006559212014. URL: https://doi.org/10.1023/A:1006559212014.

Bagnell, James Andrew and Jeff G. Schneider (2001). "Autonomous helicopter control using reinforcement learning policy search methods". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2, pp. 1615–1620. DOI: 10.1109/ROBOT.2001.932842.

Bansal, Somil et al. (2017). "Goal-Driven Dynamics L,earning via Bayesian Optimization". In: *CoRR* abs/1703.09260. URL: http://arxiv.org/abs/1703.09260.

Baxter, Jonathan and Peter L. Bartlett (Nov. 2001). "Infinite-Horizon Policy-Gradient Estimation". In: *Journal of Artificial Intelligence Research* 15.1, pp. 319–350.

ISSN: 1076-9757. DOI: 10.1613/jair.806.
URL: http://dx.doi.org/10.1613/jair.806.
Bemporad, Alberto, Tommaso Gabbriellini, et al. (2010). "Scenario-based stochastic model predictive control for dynamic option hedging". In: *49th IEEE Conference on Decision and Control, CDC, December 15th -17th, 2010*, pp. 6089–6094.
Bemporad, Alberto and Manfred Morari (Mar. 1999). "Control of Systems Integrating Logic, Dynamics, and Constraints". In: *Automatica* 35.3, pp. 407–427.
Bernardini, Daniele and Alberto Bemporad (June 2012). "Stabilizing Model Predictive Control of Stochastic Constrained Linear Systems". In: *IEEE Transactions on Automatic Control* 57.6, pp. 1468–1480.
Bertsekas, Dimitri P. (2005). *Dynamic Programming and Optimal Control*. 3rd. Vol. 1. Belmont, MA, USA: Athena Scientific.
Bertsekas, Dimitri P. and John N. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts.
Bottou, Léon, Frank E. Curtis, and Jorge Nocedal (2018). *Optimization Methods for Large-Scale Machine Learning*. arXiv: 1606.04838.
Boyd, Stephen et al. (Jan. 2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3.1, pp. 1–122. URL: https://doi.org/10.1561/2200000016.
Bradtke, Steven J., B. Erik Ydstie, and Andrew G. Barto (1994). "Adaptive linear quadratic control using policy iteration". In: *Proceedings of 1994 American Control Conference - ACC '94*. Vol. 3, pp. 3475–3479. DOI: 10.1109/ACC.1994.735224.
Breschi, Valentina, Laura Ferrarotti, and Alberto Bemporad (Jan. 2020). "Cloud-based collaborative learning of optimal feedback controllers". In: *IFAC - PapersOnLine* 53, pp. 2660–2665. DOI: 10.1016/j.ifacol.2020.12.381.
Breschi, Valentina and Simone Formentin (2020). "Direct data-driven design of switching controllers". In: *International Journal of Robust and Nonlinear Control*. DOI: 10.1002/rnc.4821.
Campi, Marco C., Andrea Lecchini, and Sergio M. Savaresi (2002). "Virtual reference feedback tuning: a direct method for the design of feedback controllers". In: *Automatica* 38, pp. 337–1346. ISSN: 0005-1098. DOI: 10.1016/S0005-1098(02)00032-8. URL: https://doi.org/10.1016/S0005-1098(02)00032-8.

Campi, Marco C. and Sergio M. Savaresi (2006). "Direct nonlinear control design: The virtual reference feedback tuning (VRFT) approach". In: *IEEE Transactions on Automatic Control* 51.1, pp. 14–27.

Carè, Algo et al. (2019). "A Toolbox for Virtual Reference Feedback Tuning (VRFT)". In: *2019 18th European Control Conference (ECC)*, pp. 4252–4257. DOI: 10.23919/ECC.2019.8795811.

Chang, Shu, Ding Hang, and N. Zhao (Apr. 2006). "Numerical comparison of least square-based finite-difference (LSFD) and radial basis function-based finite-difference (RBFFD) methods". In: *Computers and Mathematics with Applications* 51.8, pp. 1297–1310. DOI: 10.1016/j.camwa.2006.04.015.

Cheng, Richard et al. (2019). *End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks*. arXiv: 1903.08792.

Coulson, Jeremy, John Lygeros, and Florian Dörfler (2019). "Data-enabled predictive control: In the shallows of the DeePC". In: *Proceedings of the 18th European Control Conference, ECC, 2019, Naples, Italy, June 25th - 28th*. IEEE, pp. 307–312. eprint: 1811.05890.

Dai, Tianyu and Mario Sznaier (Dec. 2018). "A Moments Based Approach to Designing MIMO Data Driven Controllers for Switched Systems". In: *Proceedings of the IEEE Conference on Decision and Control, CDC, 2018*, pp. 5652–5657. DOI: 10.1109/CDC.2018.8619361.

De Callafon, Raymond A. and Paul M. J. Van Den Hof (1997). "Suboptimal feedback control by a scheme of iterative identification and control design". In: *Mathematical Modelling of Systems* 3.1, pp. 77–101. DOI: 10.1080/13873959708837050. eprint: https://doi.org/10.1080/13873959708837050. URL: https://doi.org/10.1080/13873959708837050.

De Persis, Claudio and Pietro Tesi (2020). "Formulas for Data-Driven Control: Stabilization, Optimality, and Robustness". In: *IEEE Transactions on Automatic Control* 65.3, pp. 909–924. DOI: 10.1109/TAC.2019.2959924.

Desienroth, Marc P., Gerhard Neumann, and Jan Peters (2011). "A Survey on Policy Search for Robotics". In: *Foundations and Trends in Robotics* 2.1-2, pp. 1–142. DOI: 10.1561/2300000021.

Dimakopoulou, M., Osband I., and B. Van Roy (2018). "Scalable Coordinated Exploration in Concurrent Reinforcement Learning". In: *Proceedings of the*

32nd International Conference on Neural Information Processing Systems. NIPS'18, pp. 4223–4232.

Fazel, Maryam et al. (2018). *Global Convergence of Policy Gradient Methods for Linearized Control Problems*. arXiv: 1801.05039.

Ferrarotti, Laura and Alberto Bemporad (2019). "Synthesis of Optimal Feedback Controllers from Data via Stochastic Gradient Descent". In: *Proceedings of the 18th European Control Conference, ECC, 2019, Naples, Italy, June 25th - 28th*. Institute of Electrical and Electronics Engineers (IEEE), pp. 2486–2491. DOI: 10.23919/ECC.2019.8796130.

— (2020a). "Learning nonlinear feedback controllers from data via optimal policy search and stochastic gradient descent". In: *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 4961–4966. DOI: 10.1109/CDC42340.2020.9304052.

— (2020b). "Learning optimal switching feedback controllers from data". In: *IFAC-PapersOnLine* 53.2. 21st IFAC World Congress, pp. 1602–1607. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.2205. URL: https://www.sciencedirect.com/science/article/pii/S2405896320328597.

Ferrarotti, Laura, Valentina Breschi, and Alberto Bemporad (June 2021). "The benefits of sharing: a cloud-aided performance-driven framework to learn optimal feedback policies". In: *Proceedings of the 3rd Conference on Learning for Dynamics and Control*. Ed. by Ali Jadbabaie et al. Vol. 144. Proceedings of Machine Learning Research. PMLR, pp. 87–98. URL: https://proceedings.mlr.press/v144/ferrarotti21a.html.

Fliess, Michel and Cédric Join (Dec. 2013). "Model-free control". In: *International Journal of Control* 86.12, pp. 2228–2252. DOI: 10.1080/00207179.2013.810345. URL: https://doi.org/10.1080/00207179.2013.810345.

Formentin, Simone, Klaske van Heusden, and Alireza Karimi (2014). "A comparison of model-based and data-driven controller tuning". In: *International Journal of Adaptive Control and Signal Processing* 28.10, pp. 882–897. URL: https://doi.org/10.1002/acs.2415.

Forssell, Urban and Lennart Ljung (1999). "Closed-loop identification revisited". In: *Automatica* 35.7, pp. 1215–1241. ISSN: 0005-1098. DOI: https://doi.org/10.1016/S0005-1098(99)00022-9.

URL: https://www.sciencedirect.com/science/article/pii/S0005109899000229.

Gevers, Michel (1993).
"Towards a Joint Design of Identification and Control ?" In:
Essays on Control: Perspectives in the Theory and its Applications.
Ed. by H. L. Trentelman and J. C. Willems.
Boston, MA: Birkhäuser Boston, pp. 111–151. ISBN: 978-1-4612-0313-1.
DOI: 10.1007/978-1-4612-0313-1_5.
URL: https://doi.org/10.1007/978-1-4612-0313-1_5.

— (2005). "Identification for Control: From the Early Achievements to the Revival of Experiment Design".
In: *European Journal of Control* 11.4, pp. 335–352. ISSN: 0947-3580.
DOI: https://doi.org/10.3166/ejc.11.335-352.
URL: https://www.sciencedirect.com/science/article/pii/S0947358005710414.

Goncalves da Silva, Gustavo R. et al. (Jan. 2019).
"Data-Driven LQR Control Design".
In: *IEEE Control Systems Letters* 1, pp. 180–185.
DOI: 10.1109/LCSYS.2018.2868183.

Greensmith, Evan, Peter Bartlett, and Jonathan Baxter (2002). "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning".
In: *Advances in Neural Information Processing Systems*.
Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press.

Grondman, Ivo (2015).
"Online Model Learning Algorithms for Actor-Critic Control".
PhD dissertation. Delft Center for Systems and Control.

Gros, Sébastien and Mario Zanon (2021). "Reinforcement Learning based on MPC and the Stochastic Policy Gradient Method".
In: *2021 American Control Conference (ACC)*, pp. 1947–1952.
DOI: 10.23919/ACC50511.2021.9482765.

Grudic, Gregory Z., Vijay R. Kumar, and Lyle H. Ungar (Nov. 2003). "Using policy gradient reinforcement learning on autonomous robot controllers".
In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Las Vegas, Nevada, USA.
DOI: 10.1109/IROS.2003.1250662.

Hafner, Roland and Martin Riedmiller (July 2011).
"Reinforcement learning in feedback control".
In: *Machine Learning* 84.1, pp. 137–169.
DOI: 10.1007/s10994-011-5235-x.

Hewing, Lukas et al. (2020). “Learning-Based Model Predictive Control: Toward Safe Learning in Control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3.1, pp. 269–296. DOI: 10.1146/annurev-control-090419-075625. URL: https://doi.org/10.1146/annurev-control-090419-075625.

Hjalmarsson, Håkan (2005). “From experiment design to closed-loop control”. In: *Automatica* 41.3. Data-Based Modelling and System Identification, pp. 393–438. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2004.11.021. URL: https://www.sciencedirect.com/science/article/pii/S0005109804003346.

Hjalmarsson, Håkan, Michel Gevers, and Franky De Bruyne (1996). “For Model-based Control Design, Closed-loop Identification Gives Better Performance”. In: *Automatica* 32.12, pp. 1659–1673. DOI: https://doi.org/10.1016/S0005-1098(96)00116-1.

Hjalmarsson, Håkan, Michel Gevers, Svante Gunnarsson, et al. (1998). “Iterative feedback tuning: theory and applications”. In: *IEEE Control Systems Magazine* 18.4, pp. 26–41. DOI: 10.1109/37.710876.

Hjalmarsson, Håkan, Svante Gunnarsson, and Michel Gevers (1995). “Optimality and sub-optimality of iterative identification and control design schemes”. In: *Proceedings of 1995 American Control Conference - ACC’95*. Vol. 4, 2559–2563 vol.4. DOI: 10.1109/ACC.1995.532309.

Hofmann, Thomas et al. (2016). *Variance Reduced Stochastic Gradient Descent with Neighbors*. arXiv: 1506.03662.

Hou, Zhong-Sheng, Huijun Gao, and Frank L. Lewis (2017). “Data-Driven Control and Learning Systems”. In: *IEEE Transactions on Industrial Electronics* 64.5, pp. 4070–4075. DOI: 10.1109/TIE.2017.2653767.

Hou, Zhong-Sheng and Zhuo Wang (2013). “From Model-Based Control to Data-Driven Control: Survey, Classification and Perspective”. In: *Information Sciences* 235, pp. 3–35. URL: https://doi.org/10.1016/j.ins.2012.07.014.

Karimi, Alireza, Klaske Heusden, and Dominique Bonvin (Jan. 2007). “Noniterative Data-driven Controller Tuning Using the Correlation Approach”. In: *2007 European Control Conference, ECC 2007*.

Karimi, Alireza, Ljubisa Mišković, and Dominique Bonvin (2004). "Iterative correlation-based controller tuning". In: *International Journal of Adaptive Control and Signal Processing* 18.8, pp. 645–664. URL: https://doi.org/10.1002/acs.825.

Khan, Arbaaz et al. (2018). "Scalable Centralized Deep Multi-Agent Reinforcement Learning via Policy Gradients". In: *CoRR* abs/1805.08776. arXiv: 1805.08776. URL: http://arxiv.org/abs/1805.08776.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7th-9th, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Kiumarsi, Bahare et al. (June 2018). "Optimal and autonomous control using Reinforcement Learning: a survey". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.6, pp. 2042–2062. DOI: 10.1109/TNNLS.2017.2773458.

Ko, Jonathan et al. (2007). "Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 742–747. DOI: 10.1109/ROBOT.2007.363075.

Konda, Vijay R. and John N. Tsitsiklis (2003). "On actor-critic algorithms". In: *SIAM Journal of Optimal Control* 42, pp. 1143–1166.

Lewis, Frank L., Draguna Vrabie, and Kyriakos G. Vamvoudakis (Dec. 2012). "Reinforcement Learning and Feedback Control: Using Natural Decision Methods to Design Optimal Adaptive Controllers". In: *IEEE Control Systems Magazine* 32.6, pp. 76–105. DOI: 10.1109/MCS.2012.2214134.

Lillicrap, Timothy P. et al. (2019). *Continuous control with deep reinforcement learning*. arXiv: 1509.02971.

Ljung, Lennart (1999). *System Identification: theory for the user*. 2nd. USA: Prentice Hall PTR.

Makrygiorgos, Georgios et al. (2022). "Performance-Oriented Model Learning for Control via Multi-Objective Bayesian Optimization". In: *Computers & Chemical Engineering*, p. 107770.

Markovsky, Ivan and Paolo Rapisarda (2008). "Data-driven simulation and control". In: *International Journal of Control* 81.12, pp. 1946–1959.

DOI: 10.1080/00207170801942170.
URL: https://doi.org/10.1080/00207170801942170.
Moritz, Philipp, Robert Nishihara, and Michael I. Jordan (2016).
A Linearly-Convergent Stochastic L-BFGS Algorithm.
arXiv: 1508.02087.
Mukherjee, Sayak, He Bai, and Aranya Chakrabortty (2018).
"On Model-Free Reinforcement Learning of Reduced-Order Optimal Control for Singularly Perturbed Systems".
In: *2018 IEEE Conference on Decision and Control (CDC)*, pp. 5288–5293.
DOI: 10.1109/CDC.2018.8619022.
Nagayoshi, Masato, Hajime Murao, and Hisashi Tamaki (Aug. 2010).
"A Reinforcement Learning with switching controllers for a continuous action space". In: *Artificial Life and Robotics* 15.1, pp. 97–100.
DOI: 10.1007/s10015-010-0772-0.
Nair, A. et al. (2015).
Massively Parallel Methods for Deep Reinforcement Learning.
arXiv: 1507.04296 [cs.LG].
Novara, Carlo et al. (2016). "Data-driven design of two degree-of-freedom nonlinear controllers: the D2-IBC approach". In: *Automatica* 72, pp. 19–27.
ISSN: 0005-1098. DOI:
https://doi.org/10.1016/j.automatica.2016.05.010.
URL: https://www.sciencedirect.com/science/article/pii/S0005109816301959.
Ogata, Katsuhiko (2010). *Modern Control Engineering*. 5th.
USA: Prentice Hall PTR.
Pang, Bo, Tao Bian, and Zhong-Ping Jiang (Jan. 2019).
"Data-driven Finite-horizon Optimal Control for Linear Time-varying Discrete-time Systems". In: *Proceedings of the IEEE Conference on Decision and Control, CDC, 2018*.
Institute of Electrical and Electronics Engineers (IEEE), pp. 861–866.
DOI: 10.1109/CDC.2018.8619347.
Peters, Jan and Stefan Schaal (May 2008).
"Reinforcement Learning of Motor Skills with Policy Gradients".
In: *Neural Networks* 21.4, pp. 682–697.
DOI: 10.1016/j.neunet.2008.02.003.
Piga, Dario, Marco Forgione, et al. (2019).
"Performance-oriented model learning for data-driven MPC design".
In: *IEEE control systems letters* 3.3, pp. 577–582.
Piga, Dario, Simone Formentin, and Alberto Bemporad (2018).
"Direct Data-Driven Control of Constrained Systems".

In: *IEEE Transactions on Control Systems Technology* 26.4, pp. 1422–1429. DOI: 10.1109/TCST.2017.2702118.

Powell, Warren B. (2019). *From Reinforcement Learning to Optimal Control: A unified framework for sequential decisions*. arXiv: 1912.03513.

Rasmussen, Carl Edward (2003). "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer, pp. 63–71. URL: https://doi.org/10.1007/978-3-540-28650-9_4.

Recht, Benjamin (2018). *A Tour of Reinforcement Learning: The View from Continuous Control*. eprint: arXiv:1806.09460.

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2019). "On the Convergence of Adam and Beyond". In: *Proceedings of the 6th International Conference on Learning Representations, ICLR, Vancouver, Canada, April 30th-May 3rd, 2018*. arXiv: 1904.09237.

Robbins, Herbert and Sutton Monoro (1951). "A stochastic approximation method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. URL: https://doi.org/10.1214/aoms/1177729586.

Ruder, Sebastian (2017). *An overview of gradient descent optimization algorithms*. arXiv: 1609.04747.

Saha, Priyabrata, Magnus Egerstedt, and Saibal Mukhopadhyay (2021). "Neural Identification for Control". In: *IEEE Robotics and Automation Letters* 6.3, pp. 4648–4655. DOI: 10.1109/LRA.2021.3068099.

Salvador Ortiz, José R. et al. (Jan. 2018). "Data-based predictive control via direct weight optimization". In: vol. 51, pp. 356–361. DOI: 10.1016/j.ifacol.2018.11.059.

Seborg, Dale E., Thomas F. Edgar, and Duncan A. Mellichamp (2004). *Process Dynamics and Control, 2nd ed.* Wiley.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, Richard S., David McAllester, et al. (1999). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, pp. 1057–1063. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Tieleman, Tijmen and Geoffrey Hinton (2012). *Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.

Vamvoudakis, Kyriakos G. and Frank L. Lewis (2009). "Online actor critic algorithm to solve the continuous-time infinite horizon optimal control problem". In: *2009 International Joint Conference on Neural Networks*, pp. 3180–3187. DOI: 10.1109/IJCNN.2009.5178586.

van Heusden, Klaske, Alireza Karimi, and Dominique Bonvin (2011). "Data-driven model reference control with asymptotically guaranteed stability". In: *International Journal of Adaptive Control and Signal Processing* 25.4, pp. 331–351. DOI: 10.1002/acs.1212. URL: https://doi.org/10.1002/acs.1212.

Williams, Ronald J. (May 1992). "Simple statistical gradient-following algorithms for connectionist Reinforcement Learning". In: *Machine Learning* 8.3, pp. 229–256. URL: https://doi.org/10.1007/BF00992696.