Leveraging Proximal Optimization for Differentiating Optimal Control Solvers

Oumayma Bounou
oumayma.bounou@inria.fr [*,1, 2]

Jean Ponce
jean.ponce@ens.fr [2, 3]

Justin Carpentier
justin.carpentier@inria.fr [1, 2]

Abstract— **Over the past few years, differentiable optimization has gained in maturity and attractivity within both machine learning and robotics communities. It consists in computing the derivatives of a given optimization problem which can then be used by learning algorithms, and enables to generically plug computational blocks reflecting the solving of generic mathematical programming problems into a learning pipeline. Until now, dedicated approaches have been proposed to compute the derivatives of various types of optimization problems (LPs, QPs, SOCPs, etc.). However, these approaches assume the problems are well-posed (e.g., satisfaction of the linearly independent constraint qualifications), limiting *de facto* their application to ill-posed problems. In this work, we focus on the differentiation of optimal control solvers widely used in robotics. We notably introduce a differentiable proximal formulation for solving equality-constrained LQR problems that is effective in solving ill-posed and rank-deficient problems accurately. Importantly, we show that this proximal formulation allows us to compute accurate gradients even in the case of ill-posed problems which do not satisfy the classical constraints qualification. Because any optimal control problem can be casted as an equality-constrained LQR problem in the vicinity of the optimal solution, ours robust LQR derivatives computation can then be exploited to obtain the derivatives of general optimal control problems. We demonstrate the effectiveness of our approach in dynamics learning and system parameters identification experiments in linear optimal control problems.**

I. Introduction

A key part of programming robot movements is often an instance of trajectory optimization, where the task to solve is encoded as a cost term and physical constraints are encoded as path constraints. Optimal control offers a sound mathematical framework to generically solve trajectory optimization problems. Over the past decades, differential dynamic programming (DDP) [1] and the iterative linear quadratic regulator (iLQR) [2] have become widespread and tractable approaches to solving real and complex robotic problems, ranging from UAVs navigation [3] to the control of legged locomotion [4]. While DDP was originally designed for unconstrained problems, variants have been proposed to account for various levels of path constraints, ranging from simple bounds on the control inputs [5] to the handling of generic equality [6], [7] and inequality constraints [8], [9], [10].

With recent progress in the development of advanced computational frameworks in machine learning (e.g., PyTorch [11], TensorFlow [12], JAX [13]), differentiable optimization appears as a generic approach to compute the derivatives of computational layers involving mathematical programming problems. It can be used, for instance, to find the optimal design parameters of a robotic mechanism (a.k.a. co-design) given a task to solve [14], or to estimate the sensitivities of an optimal solution with respect to the parameters of the problem [15]. In general, differentiable optimization relies on differentiating the optimality conditions (e.g., the Karush–Kuhn–Tucker conditions) associated to a given problem. In the field of convex optimization, recent works have introduced efficient approaches to compute derivatives of standard quadratic programming (QP) instances [16] and more general convex programming problems [17], which are very common in robotics. When deriving the KKT conditions, these works often assume the well-posedness of the KKT conditions at the optimum (e.g., linearly independent constraints qualification), which cannot be ensured for the large majority of control problems in robotics and more generally when integrating an optimization stage as a layer within a computational graph. For instance, in the case of redundant constraints (e.g., multiple contacts between a robot segment and the environment), one needs to adequately regularize the problem in order to robustly converge to an optimal solution satisfying the constraints [18]. This is even more true in machine learning applications, where mathematical programs are used as differentiable layers. Indeed, the intermediate problems that need to be solved during the learning process might likely not satisfy the required optimality qualifications as they are never enforced during training.

The closest work to ours is the work of Amos et al. in [19], where the authors introduce a generic approach to differentiating constrained LQR problems, and more generically, DDP-based approaches. In their experimental validation, the authors notably report a high failure rate of their differentiable solver when using its computed sensitivities in standard machine learning problems. In this paper, we argue that these failures are partly due to the lack of regularity when evaluating the derivatives of the considered optimal control problems. To overcome these limitations, we leverage the proximal method of multipliers [20] to robustly evaluate the derivatives of optimal control problems. Particularly, we develop a generic solution to compute these derivatives in the equality-constrained linear quadratic regulator (LQR), which is, as highlighted in [19], the core block required to compute the derivatives of nonlinear optimal control problems.

[*]Corresponding author
[1]Inria
[2]Département d'Informatique de l'École normale supérieure, (ENS-PSL, CNRS, Inria)
[3]Center for Data Science, New York University

Following the set of experimental validations introduced in [19], we notably demonstrate that our proximal approach is able to properly cope with ill-posed or difficult system identification problems, unlike existing solutions of the state of the art.

II. Proximal solving of LQR problems

Several approaches have been proposed to efficiently solve LQR problems numerically. In this work, we focus on the dynamic programming formulation of LQR which we regularize with the proximal method of multipliers [20]. Let us first briefly recall this method, and explain how we include it in our approach.

A. Background on the proximal method of multipliers

Throughout this paper, we address convex minimization problems of the form:

$$\min_x \ell(x), \quad \text{s.t. } c(x) = 0, \tag{1}$$

where $\ell : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function and c is a linear constraint in x. Typically, in the case of quadratic programming problems (QP problems), we have $\ell(x) = x^T H x + g^T x$, where H is a square symmetric and semi-definite matrix of appropriate dimensions, g is a vector and $c(x) = Ax - b$, with A a potentially rank-deficient matrix. Let us define the Lagrangian for this problem as

$$\mathcal{L}(x, \lambda) = \ell(x) + \lambda^T c(x), \tag{2}$$

where λ is a vector of multipliers. Rockafellar proves in [20] that solving Eq. (1) as a min max problem with saddle points solutions x, λ:

$$x, \lambda = \arg\min_x \max_\lambda \ \mathcal{L}(x, \lambda) \tag{3}$$

is equivalent to iteratively solving the regularized min max problem

$$\begin{aligned} x^{k+1}, \lambda^{k+1} = \arg\min_x \max_\lambda \ \mathcal{L}(x, \lambda) &+ \frac{1}{2\mu}\|x - x^k\|_2^2 \\ &- \frac{1}{2\mu}\|\lambda - \lambda^k\|_2^2, \end{aligned} \tag{4}$$

starting from initial vectors x^0, λ^0 until convergence, with μ a fixed positive scalar. In (4), the iterative $\min\max$ problem solved is the same as (3) with additional L2-norm regularization on the primal and dual variables x and λ, which vanishes over the iterations. Essentially, this proximal regularization [21] of the saddle points associated with problem (1) allows us to solve potentially simpler problems at the price of more iterations. It is central to our approach and will be notably used to solve constrained LQR problems accurately and compute their sensitivities robustly.

B. Proximal LQR solving

Problem definition. We address the constrained LQR problem of finding the optimal sequence of states and controls that minimize a quadratic cost function l_t under given linear dynamics f_t, path constraints encapsulated in c_t (e.g., initial and terminal constraints), with a control horizon of length N. The problem can be formulated as:

$$\begin{aligned} &\min_{X,U} \sum_{t=0}^{N-1} l_t(x_t, u_t) + l_N(x_N), \\ &\text{s.t} \begin{cases} x_{t+1} = f_t(x_t, u_t) \\ x_0 = x_0^* \\ c_t(x_t, u_t) = 0 \end{cases}, \end{aligned} \tag{5}$$

with $X = \begin{bmatrix} x_0^T & \dots & x_N^T \end{bmatrix}^T$ and $U = \begin{bmatrix} u_0^T & \dots & u_{N-1}^T \end{bmatrix}^T$. While many methods can be used to solve problem (5), we follow here the dynamic programming approach [1]. Starting from $t = N$, we solve recursively backward in time:

$$\begin{aligned} V_t(x_t) = \min_{u_t, x_{t+1}} \ & l_t(x_t, u_t) + V_{t+1}(x_{t+1}) \\ & \text{s.t. } x_{t+1} = f_t(x_t, u_t) \text{ and } c_t(x_t, u_t) = 0. \end{aligned} \tag{6}$$

We define the problem cost and dynamics functions and the path constraints:

$$\begin{cases} l_t(x_t, u_t) = \frac{1}{2}(x_t^T Q_t x_t + u_t^T R_t u_t) + q_{x,t}^T x_t + q_{u,t}^T u_t, \\ l_N(x_N) = \frac{1}{2}(x_N^T Q_N x_N) + q_{x,N}^T x_N, \\ f_t(x_t, u_t) = A_t x_t + B_t u_t + d_t, \\ C_t x_t + D_t u_t = e_t, \\ x_0 = x_0^*. \end{cases} \tag{7}$$

Here, Q_t is a symmetric positive semi-definite matrix, R_t is a symmetric positive-definite matrix, A_t, B_t, C_t and D_t are general matrices, and e_t and d_t are vectors, for all t. x_0^* is the initial condition.

Relaxing the dynamic constraints. Following the approach introduced in [22], we relax the dynamics constraints using multiple shooting and introducing an auxiliary variable y_t, such that problem (6) becomes:

$$\begin{aligned} V_t(x_t) = \min_{u_t, y_t} \ & l_t(x_t, u_t) + V_{t+1}(y_t) \\ & \text{s.t. } y_t = f_t(x_t, u_t) \text{ and } c_t(x_t, u_t) = 0. \end{aligned} \tag{8}$$

This inexact method aims at stabilizing the optimization procedure and does not prevent the solver from converging to an optimal solution, as demonstrated by Schmidt et al. in [23].

In the following, we use the usual shorthands V', V_x and V_{xx} to denote resepctively V_{t+1}, the first and the second derivatives of V_t. At time t, we solve problem (8) by solving the following $\min\max$ problem:

$$V_t(x_t) = \min_{y_t, u_t} \max_{\lambda_t, \nu_t} \ \mathcal{L}_t(x_t, u_t, y_t, \lambda_t, \nu_t), \tag{9}$$

where $\mathcal{L}_t$ is the Lagrangian of the problem, defined as:

$$\begin{aligned} \mathcal{L}_t(x_t, u_t, y_t, \lambda_t, \nu_t) = \ & l_t(x_t, u_t) + V'(y_t) \\ & + \lambda_t^T (y_t - f_t(x_t, u_t)) \\ & + \nu_t^T c_t(x_t, u_t). \end{aligned} \tag{10}$$

Here, y_t and u_t are the so-called primal variables, while λ_t and ν_t are the dual ones. In the LQR context, V_N is quadratic i.e., $V_N(x_N) = \frac{1}{2} x_N^T Q_f x_N + q_N^T x_N$. Thus, for each t, V_t is also quadratic. The LQR problem can then be solved using the dynamic programming principle, by solving at each time t a linear system where the KKT matrix is

$$\begin{bmatrix} R_t & 0 & B_t^T & D_t^T \\ 0 & V'_{xx} & -I & 0 \\ B_t & -I & 0 & 0 \\ D_t & 0 & 0 & 0 \end{bmatrix}. \tag{11}$$

Such a matrix can have very poor conditioning, especially in the cases where V'_{xx} and/or R_t are not strictly symmetric positive definite or where the matrix D_t is rank-deficient. As noted in the introduction, such a scenario is likely to happen in practice, especially in a learning framework where the cost matrices R_t and Q_t and the path constraints matrix D_t are learnable parameters. To cope with this, we propose to solve a proximal reformulation of this problem that we detail in the next paragraph.

C. Proximal regularization

We find a solution to problem (9) by iteratively solving for all t the equivalent regularized problem

$$\begin{aligned} \mathcal{L}_{\rho\mu}(x_t, u_t, y_t, \lambda_t) = {} & l_t(x_t, u_t) + V_{t+1}(y_t) \\ & + \lambda_t^T(y_t - f(x_t, u_t)) + \nu_t c_t(x_t, u_t) \\ & + \frac{\rho}{2}\|u_t - u_t^-\|_2^2 + \frac{\rho}{2}\|y_t - y_t^-\|_2^2 \\ & - \frac{\mu}{2}\|\lambda_t - \lambda_t^-\|_2^2 - \frac{\mu}{2}\|\nu_t - \nu_t^-\|_2^2, \end{aligned} \tag{12}$$

starting from initial guesses u_t^0, y_t^0, λ_t^0 and ν_t^0 as shown in [20] and detailed in Sec. II-A. The superscript "$^-$" is used for variables from the previous proximal iteration. $\mathcal{L}_{\rho\mu}$ is a regularized version of $\mathcal{L}_t$ with additional L2 penalties on primal (u, y) and dual (λ, ν) variables, and ρ and μ are positive regularization coefficients.
The new KKT conditions at time t at the optimum are given by:

$$\begin{bmatrix} R_t + \rho I & 0 & B_t^T & D_t^T \\ 0 & V'_{xx} + \rho I & -I & 0 \\ B_t & -I & -\mu I & 0 \\ D_t & 0 & 0 & -\mu I \end{bmatrix} \begin{bmatrix} du_t \\ dy_t \\ d\lambda_t \\ d\nu_t \end{bmatrix} = - \begin{bmatrix} R_t u_t^- + B_t^T \lambda_t^- + D_t^T \nu_t^- + q_{u,t} \\ V'_{xx} y_t^- + V'_x - \lambda_t^- \\ A_t x_t + B_t u_t^- + d_t - y_t^- \\ C_t x_t + D_t u_t^- - e_t \end{bmatrix}, \tag{13}$$

where $dv = v - v^-$, for v in $\{u_t, y_t, \lambda_t, \nu_t\}$. When solving Eq. (13) backward in time, x_t is unknown, but $du_t, dy_t, d\lambda_t, d\nu_t$ can be expressed as affine functions of x_t:

$$\begin{cases} du_t & = \Gamma_t x_t + \gamma_t, \\ dy_t & = M_t x_t + m_t, \\ d\lambda_t & = \Xi_t x_t + \xi_t, \\ d\nu_t & = \Omega_t x_t + \omega_t, \end{cases} \tag{14}$$

and the coefficients Γ_t, γ_t, M_t, m_t, Ξ_t and ξ_t can be obtained by solving the linear system:

$$\mathrm{K} \begin{bmatrix} \Gamma_t & \gamma_t \\ M_t & m_t \\ \Xi_t & \xi_t \\ \Omega_t & \omega_t \end{bmatrix} = - \begin{bmatrix} 0 & R_t u_t^- + B_t^T \lambda_t^- + D_t^T \nu_t^- + q_{u,t} \\ 0 & V'_{xx} y_t^- + V'_x - \lambda_t^- \\ A_t & B u_t^- + d_t - y_t^- \\ -C_t & D_t u_t^- - e_t \end{bmatrix}. \tag{15}$$

In the backward pass of the algorithm, we compute the coefficients from Eq. (15) and update the value function first and second derivatives V_{xx} and V_x for each t. Then, we perform forward passes in time to update the variables $u_t, y_t, \lambda_t, \nu_t$, with

$$\begin{cases} u_t^+ = u_t^- + du_t, \\ y_t^+ = y_t^- + dy_t, \\ \lambda_t^+ = \lambda_t^- + d\lambda_t, \\ \nu_t^+ = \nu_t^- + d\nu_t. \end{cases} \tag{16}$$

We thus solve the LQR problem by iteratively repeating this backward / forward procedure until convergence. Our stopping criterion is the infinite norm of the gradient of $\mathcal{L}_{\rho,\mu}$ (i.e., the infinite norm of the right hand side of Eq. (13) over all times t):

$$\left\| \begin{bmatrix} \nabla_{y,u}\mathcal{L}(y, u) \\ \nabla_{\lambda,\nu}\mathcal{L}(\lambda, \nu) \end{bmatrix} \right\|_\infty \leq \epsilon, \tag{17}$$

where ϵ is a fixed tolerance parameter.

D. Derivatives

We follow the approach of [19] to compute derivatives of LQR outputs (i.e., the trajectories $\mathbf{x}$ and $\mathbf{u}$) with respect to the problem parameters (i.e., the dynamics and cost matrices A_t, B_t, Q_t, R_t, the path constraints C_t, D_t, d_t, e_t, and the initial condition x_0^*) to be able to plug the LQR solver as a differentiable layer in a model. Because our formulation of the LQR problem in Eq. (7) is generic and accounts for additional path constraints, our differentiation formulation can be used to differentiate through more general optimal control problems with potentially nonlinear equality and/or inequality constraints, by differentiating through the solutions at the optimum.
We first reformulate the resolution of the general LQR problem defined in Eq. (7) as the minimization of a large QP problem

$$\min_X \frac{1}{2} X^T H X + q^T X, \quad \text{s.t } \hat{A} X = b, \tag{18}$$

with the following augmented matrices:

$$H = diag(Q_t, ..., R_t, ...),$$

$$q = \begin{bmatrix} q_{x,0}^T & ... & q_{x,N}^T & q_{u,0}^T & ... & q_{u,N-1}^T \end{bmatrix}^T,$$

$$\hat{A} = \begin{bmatrix} -I & 0 & 0 & ... & 0 & 0 & 0 & ... & 0 \\ A_0 & -I & 0 & ... & 0 & B_0 & 0 & ... & 0 \\ 0 & A_1 & -I & ... & 0 & 0 & B_1 & ... & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ ... & ... & ... & A_{N-1} & -I & 0 & ... & 0 & B_{N-1} \\ C_0 & ... & ... & 0 & 0 & D_0 & ... & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & ... & ... & C_{N-1} & 0 & D_{N-1} & ... & 0 & 0 \\ 0 & ... & ... & 0 & C_N & 0 & ... & 0 & 0 \end{bmatrix},$$

$$X = \begin{bmatrix} x_0^T & ... & x_N^T & u_0^T & ... & u_{N-1}^T \end{bmatrix}^T,$$

and

$$b = -\begin{bmatrix} x_0^{*,T} & d_1^T & ... & d_N^T & e_0^T & ... & e_N^T \end{bmatrix}.$$

The KKT system associated to this problem at the optimum is:

$$\begin{bmatrix} H & \hat{A}^T \\ \hat{A} & 0 \end{bmatrix} \begin{bmatrix} X^* \\ \Lambda^* \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix}, \tag{19}$$

where Λ^* is a vector of multipliers associated to the constraints $\hat{A}X = b$ and containing the stacked multipliers λ_t, ν_t from Eq. (10). for all t. We denote as $\hat{K}$ the KKT matrix from Eq. (19) associated with the large QP problem. In the following, we drop the "$*$" superscript for clarity. In practice, the derivative of interest in learning frameworks is $\partial r/\partial p$, where p is in $\mathcal{P}_t = \{A_t, B_t, Q_t, q_{x,t}, q_{u,t}, C_t, D_t, x_0^*, d_t, e_t\}$, and r is a scalar function of the parameters $\mathcal{P}_t$. To obtain $\partial r/\partial p$ directly, we use the same trick as in [19] and [24], and obtain expressions for the partial derivatives

$$\begin{aligned}
\partial r/\partial A_t &= \lambda_{t+1} G_{x,t}^T + x_t G_{\lambda,t+1}^T \\
\partial r/\partial B_t &= \lambda_{t+1} G_{u,t}^T + G_{\lambda,t+1} u_t^T \\
\partial r/\partial Q_t &= x_t G_{x,t}^T \qquad \partial r/\partial R_t = G_{u,t} u_t^T \\
\partial r/\partial C_t &= -x_t G_{\nu,t}^T \qquad \partial r/\partial e_t = G_{\nu,t} \\
\partial r/\partial D_t &= -u_t G_{\nu,t}^T \qquad \partial r/\partial d_t = G_{\lambda,t} \\
\partial r/\partial q_{x,t} &= G_{x,t} \qquad \partial r/\partial q_{u,t} = G_{u,t}
\end{aligned}, \tag{20}$$

as functions of a vector G that verifies

$$\hat{K} G = Z, \tag{21}$$

where Z is the vector resulting from stacking all the vectors $\partial r/\partial v_t$ with v_t in $\{x_t, u_t, \lambda_t, \nu_t\}$, for all t, which can be obtained with backpropagation. We can also write G as the stacking of vectors $G_{v,t}$ for all t, with v_t in $\{x_t, u_t, \lambda_t, \nu_t\}$, where each vector $G_{v,t}$ is the same size as v_t. In [19], Amos et al. notice that G verifies the same equation the vector $[X^T \quad \Lambda^T]^T$ verifies in Eq. (19). Thus, G is solution to a new LQR problem, and can be obtained efficiently without explicitly inverting the large KKT matrix $\hat{K}$ in Eq. (21). Instead, we solve a similar LQR problem to the one $\mathbf{x}$ and $\mathbf{u}$ are solutions to. The KKT matrices of this new problem, which we refer to as the LQR derivatives problem, are the same as the original one, and only the values of the right hand side term in the large QP formulation are modified, which correspond to the parameters $q_{x,t}$, $q_{u,t}$, d_t and e_t in the LQR formulation. Their values for the LQR derivatives problem are:

$$\begin{cases} q_{x,t} = \partial r/\partial x_t & q_{u,t} = \partial r/\partial u_t \\ d_t = \partial r/\partial \lambda_t & e_t = \partial r/\partial \nu_t \end{cases}. \tag{22}$$

Unlike [19], we solve the new LQR derivatives problem using the regularized solver formulation we introduced in sections II-B and II-C. We solve the LQR problem the vectors $G_{x,t}$, $G_{u,t}$ are solutions to, and the multipliers $G_{\lambda,t}$ and $G_{\nu,t}$ associated to the constraints. Starting from initial vectors $G_{u,t}^0, G_{y,t}^0, G_{\lambda,t}^0, G_{\nu,t}^0$, we solve iteratively and until convergence the following linear system for all t, backward in time starting from $t = N$:

$$\mathrm{K}_t \begin{bmatrix} dG_{u,t} \\ dG_{y,t} \\ dG_{\lambda,t} \\ dG_{\nu,t} \end{bmatrix} = - \begin{bmatrix} R_t G_{u,t}^- + B_t^T G_{\lambda,t}^- + D_t^T G_{\nu,t}^- + \partial r/\partial u_t \\ V_{xx}' G_{y,t}^- + V_x' - G_{\lambda,t}^- \\ A_t G_{x,t} + B_t G_{u,t}^- + \partial r/\partial \lambda_t - G_{y,t}^- \\ C_t G_{x,t} + D_t G_{u,t}^- - \partial r/\partial \nu_t \end{bmatrix}, \tag{23}$$

where

$$K_t = \begin{bmatrix} R_t + \rho I & 0 & B_t^T & D_t^T \\ 0 & V_{xx}' + \rho I & -I & 0 \\ B_t & -I & -\mu I & 0 \\ D_t & 0 & 0 & -\mu I \end{bmatrix}. \tag{24}$$

This allows us to find the optimal variable G accurately, even in ill-posed problems (e.g., rank-deficient matrices). This is extremely important in practice since the gradients of interest, $\partial r/\partial p$ with p in $\mathcal{P}$, are functions of the values of G (equation (20)). Thus, any inaccurate solution G would lead to back-propagating wrong gradients, resulting in unstable training procedures.

E. Leveraging LQR derivatives for nonlinear optimal control problems

The iLQR algorithm introduced in [2] solves problem (5) with general nonlinear dynamics and cost functions f and l. It linearizes the dynamics around nominal state and control trajectories, makes quadratic approximations of the cost around these trajectories, then iteratively solves the LQR problem obtained with this approximation. Derivatives of iLQR solutions with respect to the problem parameters can be obtained from the LQR approximation of the iLQR problem at optimality.

Let us first introduce the notations for the iLQR problem. We use the usual Q_t notation for the Lagrangian of problem (5) at time t and introduce

$$\begin{aligned}
Q_{\rho\mu}(x_t, u_t, y_t, \lambda_t) = {} & l_t(x_t, u_t) + V_{t+1}(y_t) + \lambda_t^T (f(x_t, u_t) - y_t) \\
& + \nu_t^T c(x_t, u_t) + \tfrac{\rho}{2}\|u_t - u_t^-\|_2^2 + \tfrac{\rho}{2}\|y_t - y_t^-\|_2^2 \\
& - \tfrac{\mu}{2}\|\lambda_t - \lambda_t^-\|_2^2 - \tfrac{\mu}{2}\|\nu_t - \nu_t^-\|_2^2,
\end{aligned} \tag{25}$$

with ρ and μ positive regularization scalars. The goal is to find the optimal solutions $y_t, u_t, \lambda_t, \nu_t$ to problem

$$\arg\min_{y_t, u_t} \max_{\lambda_t, \nu_t} Q_{\rho\mu}(x_t, u_t, y_t, \lambda_t, \nu_t), \tag{26}$$

using the dynamic programming recursion

$$V_t(x_t) = \min_{u_t, y_t} \max_{\lambda_t, \nu_t} Q_{\rho\mu}(x_t, u_t, y_t, \lambda_t, \nu_t). \tag{27}$$

In our formalism, the approximation of the function Q_t is:

$$\begin{aligned}
\delta Q_t = {} & \frac{1}{2} \begin{bmatrix} \delta x_t \\ \delta y_t \\ \delta u_t \\ \delta \lambda_t \\ \delta \nu_t \end{bmatrix}^T \begin{bmatrix} Q_{xx} & Q_{xy} & Q_{xu} & Q_{x\lambda} & Q_{x\nu} \\ Q_{yx} & Q_{yy} & Q_{yu} & Q_{y\lambda} & Q_{y\nu} \\ Q_{ux} & Q_{uy} & Q_{uu} & Q_{u\lambda} & Q_{u\nu} \\ Q_{\lambda x} & Q_{\lambda y} & Q_{\lambda u} & Q_{\lambda\lambda} & Q_{\lambda_\nu} \\ Q_{\nu_x} & Q_{\nu y} & Q_{\nu u} & Q_{\nu\lambda} & Q_{\nu\nu} \end{bmatrix} \begin{bmatrix} \delta x_t \\ \delta y_t \\ \delta u_t \\ \delta \lambda_t \\ \delta \nu_t \end{bmatrix} \\
& + Q_x^T \delta x_t + Q_y^T \delta y_t + Q_u^T \delta u_t + Q_\lambda^T \delta \lambda_t + Q_\nu \delta \nu_t.
\end{aligned} \tag{28}$$

Let us assume the dynamics and cost function f and l are parameterized by some parameter θ. Q and its set of first and second order derivatives are evidently also parameterized by θ, namely the ones at optimality. In a learning framework, the derivative of interest, i.e., $\partial r/\partial\theta$, with r some scalar loss function over the parameters can be obtained through the chain rule

$$\frac{\partial r}{\partial \theta} = \sum_{P \in \mathcal{D}_Q^*} \frac{\partial r}{\partial P} \frac{\partial P}{\partial \theta}, \tag{29}$$

where $\mathcal{D}_Q^* = \{Q_x^*, Q_{xx}^*, \dots, Q_{\lambda\lambda}^*\}$. The derivatives $\{\partial r/\partial P\}$ can be obtained with the method described in paragraph II-D, and the derivatives $\partial P/\partial\theta$ are obtained with backpropagation.

III. Experiments

A. LQR problems with terminal constraints

We solve an LQR problem with a terminal constraint ((5) and (7)) following the approach described in section II-C. We compare the solutions of both our solver and CVXPY [25], [26], which is based on OSQP [27]. We run both solvers on three sets of parameter sizes $\{n, d, N\}$, with n the system dimension, d the control dimension and N the time horizon. For each set of parameter sizes, we run 100 experiments with randomly generated time-invariant LQR problems. The dynamics matrix A is forced to have all singular values lower than 1, and B is a matrix with random coefficients sampled uniformly in $[0, 1]$. The cost matrices Q and R are set to respectively $10^{-2}I_n$ and $10^{-1}I_d$. All experiments are run on a single CPU. We report the average results in Table I. A solver is considered successful when both primal and dual constraints are satisfied. Feasibility denotes the infinite norm on primal constraints (dynamics constraints $\|x^*_{t+1} - Ax^*_t - Bu^*_t\|_\infty$). Distance to goal is the average infinite norm $\|x_N - x^*_N\|_\infty$, with x^*_N the target terminal constraint. In the rank-deficient cases (first and second rows), our solver converges to a solution for almost every experiment, while CVXPY only converges to a solution for up to 32% of them on average. On examples where both solvers converge, ours converges to a more accurate solution. We also report the average results on well-posed cases (third row). On such cases, both solvers converge to equally good solutions, but CVXPY is much faster, since our implementation is in an interepreted language (Python) that could easily be moved to a compiled one (e.g., C++) for much better efficiency.

TABLE I: **Solvers performance.** Comparison on LQR problems with terminal constraints.

Parameters	solver	success	feasibility	distance to goal
$N = 20$, $n = 10$, $d = 2$	cvxpy	32	2.10^{-8}	2.10^{-8}
	ours	**98**	$\mathbf{4.10^{-9}}$	$\mathbf{4.10^{-9}}$
$N = 20$, $n = 15$, $d = 3$	cvxpy	2	6.10^{-6}	6.10^{-6}
	ours	**93**	$\mathbf{9.10^{-9}}$	$\mathbf{9.10^{-9}}$
$N = 20$, $n = 15$, $d = 5$	cvxpy	**100**	9.10^{-12}	9.10^{-12}
	ours	**100**	$\mathbf{2.10^{-14}}$	$\mathbf{3.10^{-15}}$

B. System identification

Identifying dynamics and cost matrices (A, B, Q, Q_f, R). We reproduce the experimental setting of [19]. Given optimal trajectories in states and controls of systems with linear dynamics and quadratic cost, the goal is to identify these dynamics and cost parameters. Formally, we solve

$$\begin{aligned}\min_{X,U}\ & \frac{1}{2}\sum_{t=0}^{N-1}(x_t^TQx_t + u_t^TRu_t) + \frac{1}{2}x_N^TQ_fx_N,\\ \text{s.t } & x_{t+1} = Ax_t + Bu_t \text{ for } t \text{ in } 0,\dots,N-1,\\ & x_0 = x_0^*.\end{aligned}$$

We observe M trajectories $[x_0^{*,i}, \dots x_N^{*,i}]$ and $[u_0^{*,i}, \dots, u_{N-1}^{*,i}]$ (i in $1, \dots, M$) that are optimal solutions to problem (30), with different random initial conditions x_0^*. The states x_t are vectors of size n (between 2 and 10), the controls u_t are vectors of size d (between 3 and 10), and the control horizon is N (between 5 and 20). Our goal is to identify the dynamics matrices (A, B) and the cost matrices (Q, Q_f, R). In other words, we want to estimate θ, where θ can be any of A, B, Q, Q_f, R, or a combination of two or more of these matrices. We solve:

$$\min_\theta \sum_i\sum_t \|x_t^{*,i} - x_t^i(\theta)\|_2^2 + \|u_t^{*,i} - u_t^i(\theta)\|_2^2 \qquad (30)$$

where x_t^i and u_t^i are solutions to the LQR problem parameterized by θ with initial condition $x_0^{*,i}$. Using the same notations as in [19], we define the optimal trajectory vector i, $\tau^{*,i}$ as

$$\tau^{*,i} = \begin{bmatrix} x_0^{*,iT} & \dots & x_T^{*,iT} & u_0^{*,iT} & \dots & u_{T-1}^{*,i}{}^T\end{bmatrix}^T. \qquad (31)$$

Here, $\tau^{*,i}$ is a vector of size $p = (N + 1)n + Nd$. The problem is now reduced to solving

$$\min_\theta \sum_{i=1}^M\sum_{k=0}^p \|\tau_k^{*,i} - \tau_k^i(\theta)\|_2^2 = \min_\theta \sum_{k=0}^{Np}\|r_k(\theta)\|_2^2, \qquad (32)$$

where $r_k(\theta) = \Gamma_k^* - \Gamma_k$, $\tau_k^{*,i}$ (respectively $\tau_k^i(\theta)$) is the k-th component of vector $\tau^{*,i}$(respectively τ^i), and Γ_k^*(respectively Γ_k) contains stacked vectors $\tau^{*,i}$ (respectively τ^i). Problem (32) is a non-linear least-squares problem that can be solved using methods such as Gauss-Newton [28] or Levenberg-Marquardt [29]. In fact, when a step in a solution of an optimal control problem boils down to least squares, stochastic gradient descent should not be used for this step (as expected from the optimization literature), which partly explains what happens in [19], where the identification experiments fail in half the trials when using gradient descent to optimize Eq. (30). We reproduced their experiments and show results averaged on 100 experiments in Table II. An experiment is considered successful if the identification error $\|\theta - \theta^*\|_\infty$ reaches the threshold 5.10^{-6} in less than 50 iterations of the Levenberg-Marquardt algorithm or 2000 epochs of RMSProp. Table II shows that the identification succeeds 85% of the time when using a least-squares method, while it only succeeds 20% of the time when using RMSProp, which demonstrate the ineffectiveness of stochastic gradient methods in leasts-squares problems. It should be noted however that the implementation of Levenberg-Marquardt-like methods require computing the full Jacobian matrix of the residual function r (instead of just Jacobian-vector products as in SGD methods), which is a function that scales linearly with N, n and d.

TABLE II: **Success rate of identification methods.** We compare Levenberg-Marquardt and RMSProp optimization success rates (in %).

	N=5, n=3, d=3	N=20, n=3, d=3	N=10, n=5, d=2
RMSProp	21	20	21
LM	**91**	**85**	**90**

In the following experiments, the identification problem was solved using the Levenberg-Marquardt method for both our solver and the solver from [19].

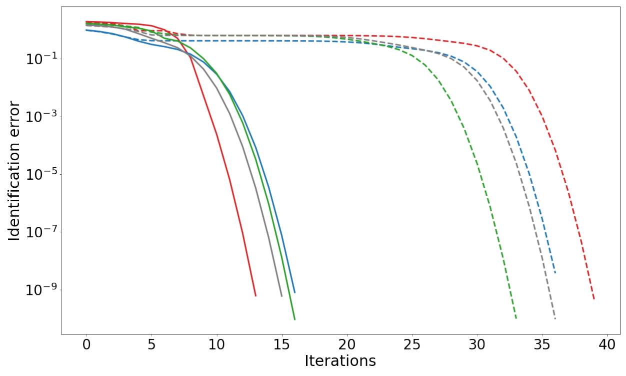

Fig. 1: **Identification error** on identifying the matrices A and B with $Q = 10^{-4}I$. Pairs of same color curves are identification experiments on the same problem parameters solved using different solvers: diff-mpc in dashed-lines, and ours in solid lines.

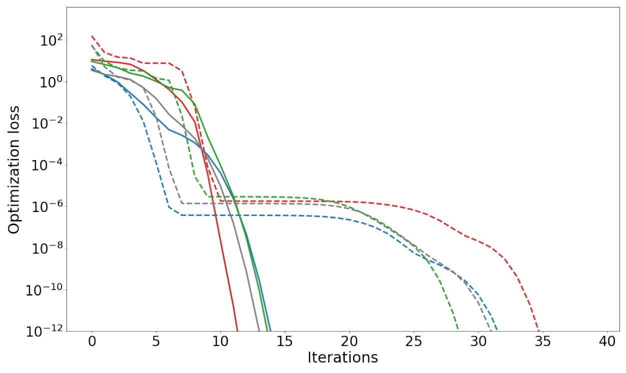

Fig. 2: **Optimization loss** on identifying the matrices A and B with $Q = 10^{-4}I$. Pairs of same color curves are identification experiments on the same problem parameters solved using different solvers: diff-mpc in dashed-lines, and ours in solid lines.

Figures 1 and 2 show the system identification and optimization errors as functions of the number of iterations using both the solver from [19], which we refer to as diff-mpc, and our solver on four trials (100 experiments were run, but we only show four randomly selected ones here). In this experiment, the parameters to identify were the dynamics and control matrices A and B, the control cost matrix R was set to the identity matrix, and the states cost matrix Q was set to 10^{-4}. In this experiment, both solvers converge, but our regularized solver converges in twice less iterations on all the trials.
Figures 3 and 4 show the system identification and optimization errors as functions of the number of iterations using both solvers again, on experiments where the parameter to identify was the matrix Q. The approach of [19] fails on all trials, demonstrating the importance of regularizing the LQR solver

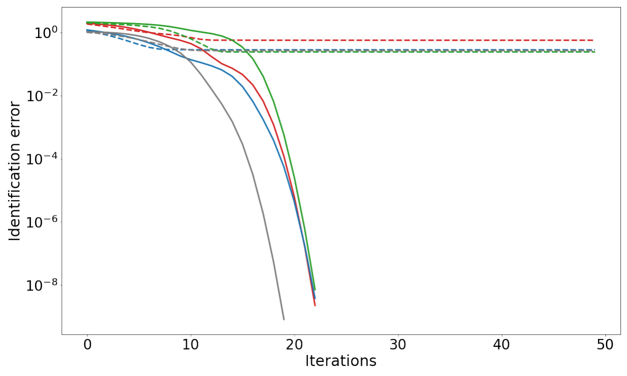

Fig. 3: **Identification error.** Identification of Q.Pairs of same color curves are identification experiments on the same problem parameters solved using different solvers: diff-mpc in dashed-lines, and ours in solid lines.

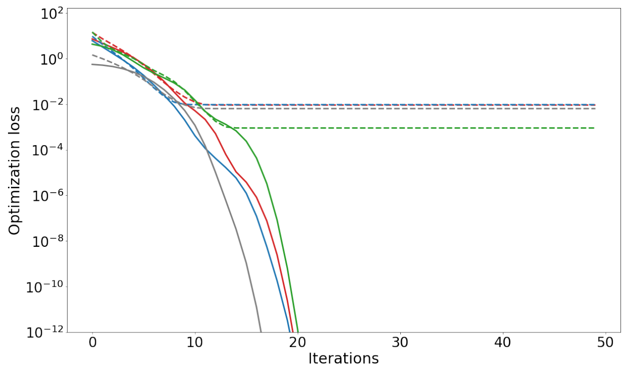

Fig. 4: **Optimization loss.** Identification of Q. Pairs of same color curves are identification experiments on the same problem parameters solved using different solvers: diff-mpc in dashed-lines, and ours in solid lines.

to avoid backpropagation of wrong gradients.

IV. Conclusion

We have introduced a regularized differentiable equality-constrained LQR solver with a generic formulation that handles path constraints. When used in learning frameworks, our solver is robust to ill-posed problems and achieves better performance on system identification experiments than the authors in [19]. Since our formulation of the LQR problem is generic, it can be applied to differentiate accurately through more general optimal control problems. Future work should include experiments on such problems, and ultimately on real robotic systems to demonstrate the effectiveness and robustness of this approach on real-world robotic tasks.

ACKNOWLEDGMENT

This work was supported in part by the Inria/NYU collaboration, the Louis Vuitton/ENS chair on artificial intelligence, the HPC resources from GENCI-IDRIS (Grand 2020-AD011011263R1) and the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA0001 (PRAIRIE 3IA Institute).

References

[1] D. Q. Mayne, "Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems," vol. 10 of *Control and Dynamic Systems*, pp. 179–254, Academic Press, 1973.

[2] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, pp. 222–229, Citeseer, 2004.

[3] J. Marti-Saumell, J. Solà, C. Mastalli, and A. Santamaria-Navarro, "Squash-box feasibility driven differential dynamic programming," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7637–7644, IEEE.

[4] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, "Differential dynamic programming for multi-phase rigid contact dynamics," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 1–9, IEEE, 2018.

[5] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1168–1175, IEEE, 2014.

[6] M. Giftthaler and J. Buchli, "A projection approach to equality constrained iterative linear quadratic optimal control," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 61–66, IEEE, 2017.

[7] S. El Kazdadi, J. Carpentier, and J. Ponce, "Equality constrained differential dynamic programming," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8053–8059, IEEE, 2021.

[8] G. Lantoine and R. P. Russell, "A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 1: Theory," *Journal of Optimization Theory and Applications*, vol. 154, no. 2, pp. 382–417, 2012.

[9] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7674–7679, IEEE, 2019.

[10] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Constrained Differential Dynamic Programming: A primal-dual augmented Lagrangian approach," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Kyoto, Japan), Oct. 2022.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[13] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018.

[14] G. Fadini, T. Flayols, A. Del Prete, N. Mansard, and P. Souères, "Computational design of energy-efficient legged robots: Optimizing for size and actuators," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9898–9904, IEEE, 2021.

[15] A. Oshin, M. Houghton, M. Acheson, I. Gregory, and E. Theodorou, "Parameterized Differential Dynamic Programming," in *Proceedings of Robotics: Science and Systems*, (New York City, NY, USA), June 2022.

[16] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, pp. 136–145, PMLR, 2017.

[17] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[18] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[19] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[20] R. T. Rockafellar, "Augmented Lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.

[21] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, p. 127–239, Jan. 2014.

[22] W. Jallet, N. Mansard, and J. Carpentier, "Implicit Differential Dynamic Programming," in *International Conference on Robotics and Automation (ICRA 2022)*, (Philadelphia, United States), IEEE Robotics and Automation Society, May 2022.

[23] M. Schmidt, N. Roux, and F. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[24] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3413–3420, 2021.

[25] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[26] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[27] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[28] C. A. Floudas and P. M. Pardalos, *Encyclopedia of Optimization*. Springer Science & Business Media, 2008.

[29] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *Siam Journal on Scientific and Statistical Computing*, vol. 4, pp. 553–572, 1983.