2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Regression-based Linear Quadratic Regulator

Hugo Carlos[1], Jean-Bernard Hayet[1], and Rafael Murrieta-Cid[1]

***Abstract*— We present the Regression-based Linear Quadratic Regulator (R-LQR), a new approach for determining locally-optimal control feedback policies for robots with non-linear dynamics and non-quadratic cost functions. Our proposal uses a free-derivative algorithm based on local quadratic regressions to obtain the robot motion policy. In addition, our methodology allows to define a notion of scale that translates into the definition of neighborhoods of valid policy and into the exploration of larger areas of the search space to find the optimal policies. The results show that our formulation allows to reach policies with lower costs than existing algorithms and to avoid problems when the behavior of the cost function makes the optimization difficult.**

I. INTRODUCTION

The problem of motion planning in mobile robotics has been tackled in numerous ways for decades now. In general, the proposed approaches rely on tools from geometry and optimal control, and for a long time, they have focused on providing solutions to essentially deterministic problems.

To handle uncertainties on controls and/or states, the stochastic optimal control framework is a tool of choice and has been tackled through a wide range of approaches in the literature for more than two decades [2]. If theoretical solutions (referred to as the value iteration algorithm) do exist, the infinite-dimensional nature of the search space makes general solutions often intractable. Exact solutions do exist only in the case of low dimensional or discrete state and control spaces, and most existing approximate methods use some form of discretization.

Based on the simplicity and neat derivation of the Linear Quadratic Regulator (LQR), a whole class of methods for finite horizon problems extend the LQR approach outside its normal boundaries (linear systems, quadratic penalizing functions, deterministic setting…) to give locally optimal solutions to stochastic optimal control problems, based on local approximations. They allow to work with a continuous (although of reduced dimension) belief space, and a continuous control space, and they have the enormous benefit to give as an output a closed-loop policy, valid in some neighborhood of the optimal open-loop solution.

This work belongs to the aforementioned category of approaches, and focuses on improving two drawbacks of existing algorithms. First, outside of the strict conditions required by the LQR, numerical approximations are necessary and, in general, the neighborhood on which the computed policy is valid does not appear explicitly. Because linearization and quadratization processes induce errors that accumulate on the horizon window, this notion of validity domain may play a critical role when executing an optimal policy. Second, most of the existing methods are by definition very local and subject to numerical instability. Our work uses local regressions on the cost-to-go functions with adaptive neighborhood sizes to overcome partially these problems.

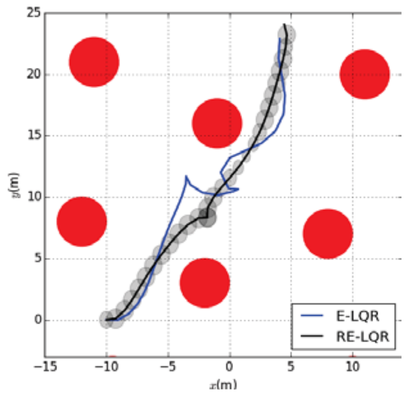


Fig. 1: Our algorithm computes locally-optimal control feedback policies for robots with non-linear dynamics and non-quadratic cost functions, in this case for a differential drive robot and obstacle repulsion potentials. The blue path is the output of [11], while our output is in black. The grey ellipses are trust regions where the computed policies are valid.

In Section II, we give a brief overview on related works. In Section III, we recall the basic elements of the extensions of the LQR to larger classes of problems, and in Section IV, we present our regression-based variant. Finally, in Section V, we present quantitative evaluations of our algorithm in different scenarios, with comparisons with existing methods.

II. RELATED WORK

The Linear Quadratic Regulator is a fundamental tool that has been studied for decades in its finite/infinite horizon versions, and for discrete-time/continuous-time problems [6]. Its solution applies for a very restricted class of systems and cost functions but serves as a basis for sub-optimal algorithms defined on broader classes of problems.

Differential Dynamic Programming (DDP) [4] maintains a representation of a single trajectory and improves it locally [13]. DDP is a second-order shooting method with quadratic convergence for any system with smooth dynamics [12]. Classic DDP requires second-order derivatives of the dynamics, and, if only the first-order terms are kept, one gets a Gauss-Newton approximation. In [13], the authors present

[1]All authors are with the Department of Computer Science, Centro de Investigación en Matemáticas, S/N, Col. Valenciana CP: 36023 Guanajuato, Gto, México {hucarlos,jbhayet,murrieta}@cimat.mx

a method which is closely related to DDP but turns out to be significantly more efficient on complex control problems.

In [7], Weiwei et al. showed that, for a *deterministic* nonlinear system, a locally-optimal solution can be derived for the closed-loop optimal control problem at a finite horizon and with a quadratic (or quadratizable) cost function of the state and the controls. The proposed solution, coined iLQR, can be seen as an extension of the Linear Quadratic Regulator (LQR). Its idea is to iteratively compute a linearization of the dynamics of the system around the previously computed open-loop controls and states and to generate the new optimal controls by backward recursion. This idea is at the core of many more works, such as [13], which describes an algorithm named iLQG that handles uncertain dynamics (Gaussian noise) and constrained controls. More recently, [11] have focused on improving a critical issue in these linearization/quadratization-based approaches, by selecting cleverly the linearization points as the local optima of the sum of cost-to-go and cost-to-come functions. It requires a double-pass (backward/forward) mechanism, but exhibits significantly better results than previous approaches.

III. The LQR and its extensions

For a discrete-time dynamical system with state $\mathbf{x}_t \in \mathbb{R}^n$ and control $\mathbf{u}_t \in \mathbb{R}^m$, at time t, we suppose that the motion model is additive-Gaussian, given by

$$\mathbf{x}_t \quad = \quad g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mu_t, \tag{1}$$

where $\mu_t \sim \mathcal{N}(0, M_t)$. We chose a simple noise model for the clarity of the explanations; as we will see later on, it can be made much more general. Let $c_t(\mathbf{x}, \mathbf{u})$ be a cost function depending on the state and control applied at t. We aim at minimizing the expected accumulated cost along an $N-$step trajectory, starting from a state $\mathbf{x}_0$, and with the expectation taken over the realizations of the motion noise

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_N} E_{\mu_1,\dots,\mu_N}[\sum_{\tau=1}^{N} c_{\tau-1}(\mathbf{x}_{\tau-1}, \mathbf{u}_\tau) + c(\mathbf{x}_N)],$$

subject to Eq. 1. Let $s_t^*(\mathbf{x}_t)$ be the cost-to-go function, that gives the optimal cost for partial trajectories starting from t, then the dynamic programming principle leads to

$$s_t^*(\mathbf{x}_t) = \min_{\mathbf{u}_{t+1}} c_t(\mathbf{x}_t, \mathbf{u}_{t+1}) + E_{\mu_{t+1}}[s_{t+1}^*(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})],$$

It is solved easily when s_t^* is quadratic for all t, which occurs when all c_t are quadratic in state and control, g is linear, and μ is additive-Gaussian. In that case, the cost-to-go function is quadratic and the recursion gives a closed form feedback policy $\mathbf{u}_{t+1}^*(\mathbf{x}_t)$ [13]. The second term in this sum, $s_{t+1}^*(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})$, takes the following form

$$\begin{aligned} &s_{t+1}^*(g(\mathbf{x}_t, \mathbf{u}) + \mu_{t+1}) = \\ &(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})^T S_{t+1}(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1}) + \\ &(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})^T \mathbf{s}_{t+1} + s_{t+1}. \end{aligned} \tag{2}$$

where $S_{t+1}, \mathbf{s}_{t+1}, s_{t+1}$ are the quadratic coefficients of $s_{t+1}^*(\mathbf{x}_{t+1})$. When the conditions enounced above (quadratic cost functions and linear motion model) are not satisfied, one way to solve the problem is to quadratize the optimal cost functions. In methods such as iLQG [13], the quadratization is done by linearization of g, which translates Eq. 2 into

$$\begin{aligned} &s_{t+1}^*(g(\mathbf{x}_t, \mathbf{u}) + \mu_{t+1}) \approx \\ &(\tilde{g}_{\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}}(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})^T S_{t+1}(\tilde{g}_{\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}}(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1}) + \\ &(\tilde{g}_{\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}}(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})^T \mathbf{s}_{t+1} + s_{t+1}. \end{aligned}$$

where $\tilde{g}_{\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}}(\mathbf{x}_t, \mathbf{u}_{t+1})$ is the linearized version of g around the linearization point $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$. One of the major difficulties is to correctly choose these linearization points.

For instance, the Stochastic Extended LQR (SE-LQR) formulation [11] alternates forward and backward passes that refine the linearization points, alternating the optimization of an approximate cost-to-come function and an approximate cost-to-go function. The backward passes assume that the cost-to-come parameters $\bar{S}_t, \bar{\mathbf{s}}_t, s_t$ are available and they use an estimate of the optimal inverse control policies from the previous pass, altogether with an initial quadratization point $\hat{\mathbf{x}}_N$ at N. One first quadratizes the final cost at $\hat{\mathbf{x}}_N$ and starts the recursions on the coefficients $S_t, \mathbf{s}_t, s_t$ of the quadratized cost-to-go, in function of their counterparts at step $t+1$, with

$$S_t = D_t - C_t^T E_t^{-1} C_t \qquad \mathbf{s}_t = \mathbf{d}_t - C_t^T E_t^{-1} \mathbf{e}_t$$

where the involved matrices and vectors depend on [11]:

- the linearizations of the motion model g at the linearization/quadratization point $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t$, $A_t = \frac{\partial g}{\partial \mathbf{x}_t}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, $B_t = \frac{\partial g}{\partial \mathbf{u}_{t+1}}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, $\mathbf{c}_t = \hat{\mathbf{x}}_{t+1} - A_t \hat{\mathbf{x}}_t - B_t \hat{\mathbf{u}}_{t+1}$,
- the Hessian matrices at $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}$, $Q_t = \frac{\partial^2 c_t}{\partial \mathbf{x}_t^2}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, $R_t = \frac{\partial^2 c_t}{\partial \mathbf{u}_{t+1}^2}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, $P_t = \frac{\partial^2 c_t}{\partial \mathbf{x}_t \partial \mathbf{u}_{t+1}}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, and gradients $\mathbf{q}_t, \mathbf{r}_t$,
- the noise covariance M_t (with this model, only in s_t).

Linearization/quadratization ensures that the resulting approximating function is convex if S_{t+1} is positive. However, it also leads to modeling errors. For instance, to get a faithful second-order approximation of $E_{\mu_{t+1}}[s_{t+1}^*(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})]$, second-order terms on g should be included. Also, these modeling errors are accumulated along the recursion steps, leading to poor approximations close to $\mathbf{x}_0$, when N is large. Finally, when estimating the derivatives with finite differences, the linearization process may induce numerical errors that also accumulate along the recursion steps.

IV. Regression-based LQR

The Iterative LQR algorithm and its variants [13], [11] approximate the objective function as a quadratic determined by the first and second derivatives of the cost-to-go function, with the limitations mentioned above. In this paper, rather than using derivatives, we interpolate the function values as a quadratic function, at a specified scale. These strategies are known as "free-derivative" algorithms and have been used to optimize complex functions with good results [10].

A. Approximating the cost-to-go function by regression

Let us define

$$s_t(\mathbf{x}_t, \mathbf{u}_{t+1}) = c_t(\mathbf{x}_t, \mathbf{u}_{t+1}) + E_{\mu_{t+1}}[s^*_{t+1}(g(\mathbf{x}_t, \mathbf{u}_{t+1}) + \mu_{t+1})].$$

Instead of derivating c_t and g at the linearization/quadratization point $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, consider R representative points $\xi_t^{(i)} = (\mathbf{x}_t^{(i)}, \mathbf{u}_{t+1}^{(i)})$ in $\mathbb{R}^{m+n}$, in the vicinity of $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$. For each $\xi_t^{(i)}$, we compute the value of s_t as

$$\phi_t^{(i)} = c_t(\mathbf{x}_t^{(i)}, \mathbf{u}_{t+1}^{(i)}) + \sum_{j=1}^{J} \gamma_j s^*_{t+1}(g(\mathbf{x}_t^{(i)}, \mathbf{u}_{t+1}^{(i)}) + \mu_{t+1}^{(j)}),$$

where $\mu_{t+1}^{(j)}$ are realizations of the noise chosen as sigma-points [5] and γ_j are their corresponding coefficients.

If g is known and an approximation of s^*_{t+1} is given, then $\phi_t^{(i)}$ can be evaluated. From the set of $(\xi_t^{(i)}, \phi_t^{(i)})$, one can solve a regression problem that adjusts a quadratic form $\tilde{s}_t(\mathbf{x}_t, \mathbf{u}_{t+1})$ to $s_t(\mathbf{x}_t, \mathbf{u}_{t+1})$ at these points,

$$\begin{array}{rcl} \tilde{s}_t(\mathbf{x}_t, \mathbf{u}_{t+1}) & = & \begin{pmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_{t+1} - \hat{\mathbf{u}}_{t+1} \end{pmatrix}^T M_t \begin{pmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_{t+1} - \hat{\mathbf{u}}_{t+1} \end{pmatrix} \\ & + & \mathbf{m}_t^T \begin{pmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_{t+1} - \hat{\mathbf{u}}_{t+1} \end{pmatrix} + \phi_t^{(i)}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}). \end{array}$$

Let us define $\Delta\xi_t^{(i)} \triangleq (\mathbf{x}_t^{(i)} - \hat{\mathbf{x}}_t, \mathbf{u}_{t+1}^{(i)} - \hat{\mathbf{u}}_{t+1})^T$. We solve

$$\min_{M_t, \mathbf{m}_t} \sum_{i=1}^{R} \| \left(\Delta\xi_t^{(i)}\right)^T M_t \Delta\xi_t^{(i)} + \mathbf{m}_t^T \Delta\xi_t^{(i)} + \phi_t^{(i)}(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) - \phi_t^{(i)}(\mathbf{x}_t, \mathbf{u}_t) \|^2.$$

By stacking all the entries of $M, \mathbf{m}$ in a vector v, we obtain a least squares linear regression problem

$$\min_{\mathbf{v}} \sum_{i=1}^{R} \|((l^{(i)})^T \mathbf{v} - \phi_t^{(i)}\|^2, \tag{3}$$

where each vector $l^{(i)}$ holds the terms corresponding to the sample i. As M is symmetric, $\mathbf{v}$ has $\frac{1}{2}(m+n)(m+n+1)$ entries. Then the optimal solution is given by

$$\mathbf{v}^* = (\sum_{i=1}^{R} l^{(i)} l^{(i)T})^{-1} (\sum_{i=1}^{R} \phi_t^{(i)} l^{(i)}). \tag{4}$$

Then we deduce M_t, $\mathbf{m}_t$ and write them as

$$M_t \triangleq \begin{bmatrix} C_t & E_t^T \\ E_t & D_t \end{bmatrix} \text{ and } \mathbf{m}_t \triangleq \begin{bmatrix} \mathbf{c}_t \\ \mathbf{d}_t \end{bmatrix}. \tag{5}$$

We can re-write $\tilde{s}_t(\mathbf{x}_t, \mathbf{u}_{t+1})$ as a quadratic in $\Delta\mathbf{u}_{t+1}$

$$\begin{array}{rcl} \tilde{s}_t(\mathbf{x}_t, \mathbf{u}_{t+1}) & = & \frac{1}{2}(\Delta\mathbf{u}_{t+1})^T D_t \Delta\mathbf{u}_{t+1} + \mathbf{d}_t^T \Delta\mathbf{u}_{t+1} \\ & + & (\Delta\mathbf{x}_t)^T E_t \Delta\mathbf{u}_{t+1} + \frac{1}{2}\Delta\mathbf{x}_t^T C_t \Delta\mathbf{x}_t \\ & + & \mathbf{c}_x^T \Delta\mathbf{x}_t + \phi_t^{(i)}(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_{t+1}), \end{array}$$

and optimize it to get the optimal feedback policy: $\Delta\mathbf{u}^*_{t+1} = -D_t^{-1}(E_t \Delta\mathbf{x}_t + \mathbf{d}_t)$. It leads to the expected affine form

$$\mathbf{u}^*_{t+1} = -D_t^{-1} E_t^T \mathbf{x}_t - \left(D_t^{-1}\mathbf{d}_t - D_t^{-1} E_t^T \bar{\mathbf{x}}_t - \bar{\mathbf{u}}_{t+1}\right), \tag{6}$$

and the optimal cost-to-go function at t, $s^*_t(\mathbf{x}_t) = \tilde{s}_t(\mathbf{x}_t, \mathbf{u}^*_{t+1})$, which is quadratic in $\mathbf{x}_t$ once the substitution of $\mathbf{u}^*_{t+1}$ is done. We call its parameters $S_t, \mathbf{s}_t, s_t$,

$$\begin{array}{rcl} S_t & = & C_t - E_t^T D_t^{-1} E_t, \\ \mathbf{s}_t & = & \mathbf{c}_t - E_t^T D_t^{-1} \mathbf{d}_t. \end{array}$$

Note that this strategy is, in essence, similar to [11]. However, the regression on $s_t(\mathbf{x}_t, \mathbf{u}_{t+1})$ implicitly includes the missing second order terms mentioned above and obtains a better approximation than the linearization/quadratization scheme. Also, it manages a scale factor (neighbourhood size) which can be adapted in the overall optimization strategy, as a trust region-type method, as described in the next section.

B. Trust region-based regression

Our trust region-based algorithm builds a model of the cost-to-go function around the current state $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$. To approximate the cost-to-go function as well as possible, in a region as large as possible [9], we consider the relative regression error e_t of Eq. 3, and a threshold ϵ that controls the regression quality. To estimate $\tilde{s}_t$, we use samples $\xi_t^{(i)}$ in an adaptive neighborhood around $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, so that the regression on that neighborhood satisfies the criterion error. The neighborhood size is specified by an $(n+m) \times 1$ vector $\mathbf{r}_t$ defining radii (neighborhood sizes) along each direction of the state/control vector. All the samples used to estimate $\tilde{s}_t$ are contained in the region defined by $\mathbf{r}_t$. Using this idea, Algorithm 1 adapts $\mathbf{r}_t$ and builds samples $\{\xi_t^{(i)}\}$ around $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, until the regression satisfies $e_t < \epsilon$.

Algorithm 1 Trust region-based regression

```
Require: Cost function $s_t(\xi)$.
Require: Reference state and control $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1}$
Require: Max. relative regression error $\epsilon$, scale reduction $\lambda$
Require: Initial neighborhood radii $\mathbf{r}_0$

 1: function REGRESSION($s_t$,$\hat{\mathbf{x}}_t$,$\hat{\mathbf{u}}_{t+1}$,$\epsilon$, $\mathbf{r}_0$,$\lambda$)
 2:     $\mathbf{r}_t \leftarrow \mathbf{r}_0$
 3:     repeat
 4:         Sample $\{\xi_t^{(i)}\}$ according to $\mathbf{r}_t$ (see Section IV-C)
 5:         Estimate $M$ and $\mathbf{m}$ from Eq. 3
 6:         Calculate the error $e_t = \sqrt{\frac{\sum_{i=1}^n \left(\tilde{s}_t(\xi_t^i) - s_t(\xi_t^i)\right)^2}{\sum_{i=1}^n s_t(\xi_t^i)^2}}$.
 7:         $\mathbf{r}_t \leftarrow \lambda \mathbf{r}_t$
 8:     until $e_t < \epsilon$
 9:     return $M$ and $\mathbf{m}$
10: end function
```

The Algorithm 1 requires, in addition to the error threshold ϵ, two parameters: $\mathbf{r}_0$ and λ. The vector $\mathbf{r}_0$ gives the initial scale. The scalar λ is a *scale reduction factor* that reduces the neighborhood size while s_t can not be approximated adequately. In the following, we will show how it is possible to choose λ and $\mathbf{r}_0$ from the characteristics of the problem.

Small values of ϵ lead to small radii, since the regression gets closer to a second-order Taylor series approximation. On the other hand, large values of ϵ may cause the approximation of s_t to be very poor, resulting in erroneous motion policies and slow pace of convergence of the planning algorithm.

Finally, note that, by using adaptive radii for the regression neighborhoods, we handle explicitly a local scale that is useful in two ways. First, it allows to search for solutions on larger ranges, and select better local optima. Second, it defines a notion of trust region, that translates into validity of computed policies and could be used in a planning scheme to decide when to do re-planning (topic not covered here).

C. Sampling $\xi_t^{(i)}$ according to $\mathbf{r}_t$

Given a trust region $\mathbf{r}_t$ we sample ξ_t^i around $(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_{t+1})$, within a neighborhood of size $\mathbf{r}_t$. The minimum number of samples to solve Eq. 3 is $R_{min} = \frac{(m+n)(m+n+1)}{2}$, but to improve the quality of the regression, we need a larger number of samples. We consider the following ellipsoid:

$$\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{u}\end{bmatrix} - \begin{bmatrix}\hat{\mathbf{x}}_t\\ \hat{\mathbf{u}}_t\end{bmatrix}\right)^T \Sigma \left(\begin{bmatrix}\mathbf{x}\\ \mathbf{u}\end{bmatrix} - \begin{bmatrix}\hat{\mathbf{x}}_t\\ \hat{\mathbf{u}}_t\end{bmatrix}\right) = 1,$$

where $\Sigma = \text{diag}(\frac{1}{(\mathbf{r}_t^0)^2} \cdots \frac{1}{(\mathbf{r}_t^n)^2})$. Given Σ, we obtain the random samples $\xi_t^{(i)}$ using the method presented in [3].

D. Handling non-convex quadratic regression outputs

The second-order information of s_t is contained in M. Because the optimization is done on the controls $\mathbf{u}_{t+1}$, D_t in Eq. 5 should be positive semi-definite (PSD) for using Eq. 6. If not, then M_t is not positive definite either. In that case, we modify M_t into a PSD matrix M_t' as in [13]:

$$M_t' = (M_t + (\delta - \lambda_{min}(M_t))I),$$

where $\lambda_{min}(M_t)$ is the minimum eigenvalue of M_t and $\delta > 0$. The matrix M_t' is used to obtain the optimal control $\mathbf{u}_{t+1}^*$ which converges to a point satisfying second-order necessary conditions for optimality [9]. Another advantage is that it is possible to plug it into different optimization strategies, for example by using the negative curvature information contained in M_t [8], or by selecting δ optimally [10], etc.

E. Regression-Based LQR algorithms

Given the basic blocks described above, we incorporate our regression-based approach for the approximation $\tilde{s}_t(\mathbf{x}_t, \mathbf{u}_{t+1})$ into different extensions of the LQR algorithm for non-linear models, non quadratic costs, where we replace the linearization/quadratization stages by the quadratic regression. For example, the Iterative LQR (I-LQR) algorithm presented in [13], is rewritten into Algorithm 2 (RI-LQR).

Similarly, for the Extended LQR (E-LQR) algorithm, in the backward (resp. forward) pass, we use the regression to compute an approximation of the cost-to-go (resp. cost-to-come) function. We have labeled this version of the algorithm as Regression-Based Extended LQR (RE-LQR).

V. Experimental results

To evaluate our algorithms and compare them to existing approaches, we have used two systems: a differential drive robot and a quadrotor. For presenting fair comparisons, the cost functions used here are the ones presented in [11]:

Algorithm 2 Regression-Based Iterative LQR (RI-LQR)

Require: Max. relative regression error ϵ, scale reduction λ
Require: Initial search radii $\mathbf{r}_0$
Require: Target configuration $\mathbf{x}_0$

```
1: function ComputePolicy
2:     while convergence not reached do
3:         $S_N = Q_N$, $\mathbf{s}_N = \mathbf{q}_N$, and $s_N = q_N$
4:         for $t = N-1$; $t \geq 0$; $t = t-1$ do
5:             $M_t, \mathbf{m}_t \leftarrow$ Regression($s_t$,$\hat{\mathbf{x}}_t$,$\hat{\mathbf{u}}_{t+1}$,$\epsilon$, $\mathbf{r}_0$,$\lambda$)
6:             if $M_t$ is not PSD then
7:                 $M_t \leftarrow (M_t + (\delta - \lambda_{min}(M_t))I)$
8:             end if
9:             From $M_t, \mathbf{m}_t$ compute $S_t, \mathbf{s}_t, s_t, \mathbf{u}_t^*$ (Section IV-A)
10:        end for
11:    end while
12: end function
```

$$\begin{aligned} c_0(\mathbf{x},\mathbf{u}) &= \frac{1}{2}\left(\mathbf{x}-\mathbf{x}_0^*\right)^T Q_0 \left(\mathbf{x}-\mathbf{x}_0^*\right) + \frac{1}{2}(\mathbf{u}-\mathbf{u}^*)^T R_u \left(\mathbf{u}-\mathbf{u}^*\right) \\ c_t(\mathbf{x},\mathbf{u}) &= \frac{1}{2}(\mathbf{u}-\mathbf{u}^*)^T R_u \left(\mathbf{u}-\mathbf{u}^*\right) + f(\mathbf{x}) \\ c_N(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{x}_N^*)^T Q_N (\mathbf{x}-\mathbf{x}_N^*)^T \end{aligned} \tag{7}$$

where Q_0, Q_N, and R_u are positive definite, $\mathbf{x}_0^*$ is the initial state, $\mathbf{x}_N^*$ the goal state, and $\mathbf{u}^*$ the reference control. As in [11], Q_0 and Q_N have large values to respect the initial and goal states. For obstacle avoidance, we introduce

$$f(\mathbf{x}) = q \sum_{i \in \mathcal{O}} \exp(-d_i(\mathbf{x})), \tag{8}$$

where i indexes the obstacles set $\mathcal{O}$, $q \in \mathbb{R}^+$, and $d_i(\mathbf{x})$ is the signed distance between the robot $\mathbf{x}$ and the obstacle i.

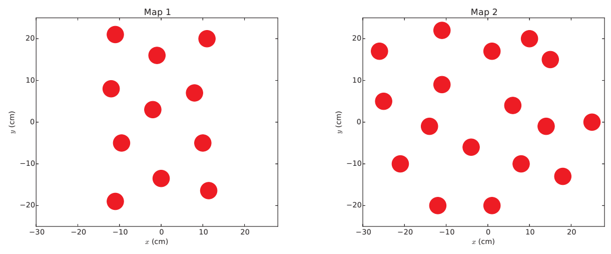
For high-dimensional systems (Case 2, hereafter), the regressions cost can become prohibitive, so we use numerical algorithms that solve the regression problem efficiently [1]. Moreover, each input of the matrices and vectors of Eq. 4, can be obtained independently so we have paralelized them.

A. Case 1: Differential drive robot

We first use a differential drive robot (DDR). Its state $\mathbf{x}_t = [x_t, y_t, \theta_t]^T$ includes its position (x_t, y_t) (m) and orientation θ (rad). Its control input $\mathbf{u}_t = [v_t, \omega_t]^T$ consists of the linear (m/s) and angular (rad/s) speeds. Its dynamics are non-linear:

$$\dot{\mathbf{x}}_t = \begin{bmatrix}\cos\theta_t & 0\\ \sin\theta_t & 0\\ 0 & 1\end{bmatrix} \begin{bmatrix} v_t\\ \omega_t\end{bmatrix}. \tag{9}$$

We have used the two environments depicted in Fig. 2. The obstacles have a radius of 0.2m, while the DDR has a radius of 0.17m. We compare the execution time, number of iterations needed to converge, and final accumulated costs of different algorithms as an average over 100 independent runs. In each run, the initial state $\mathbf{x}_0$ is sampled randomly on the boundaries of the environment, and we set $\mathbf{x}_N = -\mathbf{x}_0$. We run our approach for various fixed time-steps Δ (rows of Table I), with a fixed number of steps $N = 150$. The algorithms are not seeded with an initial trajectory, and run



(a) Map1 (b) Map 2

Fig. 2: Maps used for the experiments in Case 1 (DDR).

until the relative improvement drop below 10^{-4} (convergence criterion in Algorithm 2). The cost function parameters are $Q_0 = Q_N = 50I$, $R_u = I$, $q = 1.0$ and $\mathbf{u}^* = 0$. Table I gives quantitative results. Our methodology (RI-LQR and RE-LQR) reduces significantly the final cost and the number of iterations needed to get a solution. The computational time is higher, because of the quadratic regression parameter estimation, but in quite moderate proportions, as the total number of iterations is lower than E-LQR.

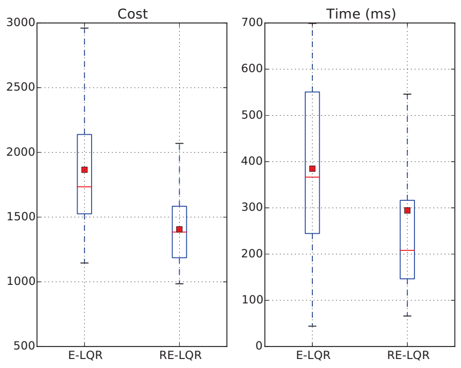
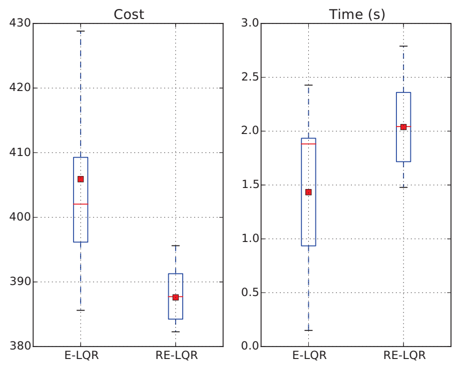


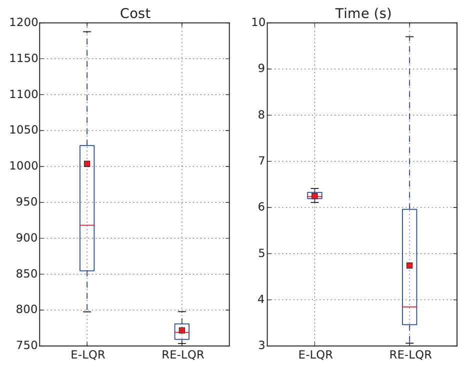
Fig. 3: Cost and time distributions for E-LQR and RE-LQR on Map 2. $R_u = 1.0I$, $Q = 50I$, $\epsilon = 0.01$, $N = 150$.

For the second set of experiments, we use Map 2, with a higher density of obstacles. The evaluation of the cost function has a higher computational cost and the search for trajectories free of collisions is harder. The cost-to-go functions are less smooth than for Map 1. E-LQR and RE-LQR are compared in Fig. 3, with the same parameters as in the previous experiments and $\Delta = 0.1$. The computational time of RE-LQR is, in average, even lower than E-LQR because of the lower number of iterations. Moreover, the average cost and variability are substantially reduced.

The Fig. 4 shows the algorithm behavior for varying ϵ. Note that different ϵ may lead to different policies. For higher values of ϵ, the approximation of the cost-to-go or cost-to-come functions can be made on a larger spatial extent, and the algorithm explores more possible solutions. However, too high values of ϵ lead to poor regressions and inconsistent policies. Moreover, regardless of the ϵ values, the trajectories obtained by RE-LQR have smaller rotations and smoother



(a) $R_u = 20.0I$, $Q = 500I$, $\epsilon = 0.01$, $\Delta = 0.1$, $N = 150$.



(b) $R_u = 10.0I$, $Q = 500I$, $\epsilon = 0.01$, $\Delta = 0.05$, $N = 200$.

Fig. 5: Comparison E-LQR/ER-LQR (quadrotor example).

paths. We explain it by the good approximation to s_t that improves the quality of the obtained optimal controls.

B. Case 2: Quadrotor in 3-D Environment

The second experiment considers a simulated quadrotor helicopter [11]. Its state $\mathbf{x}_t = [\mathbf{p}^T, \mathbf{v}^T, \mathbf{r}^T, \mathbf{w}^T]^T$ is 12-dimensional, and includes its position $\mathbf{p}$ (m), velocity $\mathbf{v}$ (m/s), orientation $\mathbf{r}$ (rotation about axis $\mathbf{r}$ by angle $||\mathbf{r}||$ (rad)), and angular velocity $\mathbf{w}$ (rad/s). Its control input $\mathbf{u}_t = [u_1, u_2, u_3, u_4]^T$ (N) consists of the forces exerted by each of the four rotors. The dynamics are highly non-linear,

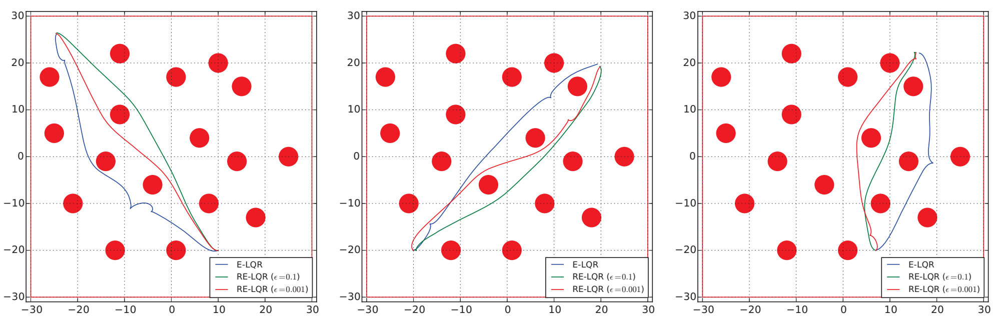
$$\begin{aligned}
\dot{\mathbf{p}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= -g\mathbf{e}_3 + ((u_1+u_2+u_3+u_4)\exp([\mathbf{r}])\mathbf{e}_3 - k_v\mathbf{v})/m \\
\dot{\mathbf{r}} &= \mathbf{w} + \frac{1}{2}[\mathbf{r}]\mathbf{w} + (1 - \frac{1}{2}||\mathbf{r}||/\tan(\frac{1}{2}||\mathbf{r}||))[\mathbf{r}]^2\mathbf{w}/||\mathbf{r}||^2 \\
\dot{\mathbf{w}} &= J^{-1}(\rho(u_2-u_4)\mathbf{e}_1 + \rho(u_3-u_1)\mathbf{e}_2 \\
&+ k_m(u_1-u_2+u_3-u_4)\mathbf{e}_3 - [\mathbf{w}]J\mathbf{w}),
\end{aligned}$$

where $\mathbf{e}_i$ are the standard basis vectors, $g = 9.8m/s^2$ is the gravity, $k_v = 0.15$ is a constant relating the velocity to an opposite force, $m = 0.5$ kg is the mass, $J = 0.05I$ (kg m^2) is the moment of inertia matrix, $\rho = 0.17$ m is the distance between the center of mass and the center of the rotors, and $k_m = 0.025$ relates the force of a rotor to its torque.

For these experiments, we use a configuration similar to [14], with a 6m by 6m by 6m 3-D environment. The geometry of the quadrotor is approximated by a sphere with

TABLE I: Quantitative comparison of LQR algorithms with DRR for Map 1.

Δ	Cost			Iterations			Time (ms)		
	E-LQR	RI-LQR	RE-LQR	E-LQR	RI-LQR	RE-LQR	E-LQR	RI-LQR	RE-LQR
0.2	555.18192	502.7634	424.59696	44.56	49.68	33.66	202.4	388.54	512.02
0.1	1967.9286	1848.9188	1425.566	73.18	35.98	33.82	325.14	308.18	646.1
0.05	9770.28	6760.448	5537.87	79.58	28.62	51.7	351.98	221.24	742.18



(a) $\mathbf{x}_0^* = (10, -20, 0.52)$, $\mathbf{x}_N^* = (-25, 27, 0.26)$ (b) $\mathbf{x}_0^* = (-20, -20, 0.26)$, $\mathbf{x}_N^* = (20, 20, 0.34)$ (c) $\mathbf{x}_0^* = (7, -20, 0.26)$, $\mathbf{x}_N^* = (16, 23, 0.34)$

Fig. 4: Trajectories obtained by E-LQR and RE-LQR for different ϵ values.

a radius of 0.3m. The cost parameters are $Q_0 = Q_l = 500I$, and $R_u = 20I$. The reference control input u^* is $\frac{1}{4}$ mgN for each rotor, which is the minimum to let the quadrotor hover. As before, the algorithms were not initialized with a given trajectory. The initial state $\mathbf{x}_0$ is $(\mathbf{p}, 0, 0, 0)$, where the initial position $\mathbf{p}$ is randomly sampled from the edges of the environment, and the target state $\mathbf{x}_N = -\mathbf{x}_0$.

Figure 5a compares E-LQR and RE-LQR, with $\Delta = 0.1$ and $N = 150$. With RE-LQR, the average cost is significantly lower than with E-LQR. As expected, the average processing time is higher but the overcost is not excessive, and is compensated by a lower total number of iterations.

When the behavior of the cost function causes D_t to be non-positive or close to it, E-LQR has a very slow convergence rate and may oscillate or diverge. In addition, without initial policy, the first policies are often very poor. One solution is to use R_u as a regularization term, so that D_t tends to be PSD. Our formulation faces the same problem, but in general, it deals better with it, even in high dimensions. This can be seen in Figure 5b, the average cost and time of ER-LQR are significantly smaller than those of E-LQR.

Overall, our proposal gives a significant reduction in the final costs. It may require higher processing times in higher dimensions or longer planning horizons, even if we observed the same order of magnitude in the systems presented above.

VI. Conclusion and Future Work

In this work, we have presented an LQR extension that solves planning problems with non-linear dynamics/non-quadratic costs and obtains locally optimal policies using free-derivative optimization. It includes a notion of neighborhood in which the policy is valid, which allows to explore potential solutions at larger scales, and to evaluate potential re-planning decisions at execution. In spite of higher computational times for individual iterations, our approach converges quicker to locally optimal solutions because of the lower number of iterations required for convergence.

As a future work, we will design a proper strategy to adapt $\mathbf{r}_t$ and use them during execution. We also want to explore new parallelization strategies for high dimensional systems.

References

[1] *Intel Math Kernel Library. Reference Manual*. Intel Corporation, 2009. Santa Clara, USA. ISBN 630813-054US.

[2] D. P. Bertsekas. *Dynamic programming and optimal control. Volume I.* Athena Scientific optimization and computation series. Belmont, Mass. Athena Scientific, 2005.

[3] J. Dezert and C. Musso. An efficient method for generating points uniformly distributed in hyperellipsoids. In *Proc. of the Workshop on Estimation, Tracking and Fusion: A Tribute to Y. Bar-Shalom*, 2001.

[4] D. Jacobson and D. Mayne. *Differential Dynamic Programming*. Elsevier, New York, NY, 1970.

[5] S. J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. In *Proc. of the IEEE*, pages 401–422, 2004.

[6] D. E. Kirk. *Optimal control theory : an introduction*. Dover Publications, April 2004.

[7] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proc. of ICINCO (1)*, pages 222–229, 2004.

[8] J. J Moré and D. C Sorensen. On the use of directions of negative curvature in a modified newton method. *Mathematical Programming*, 16(1):1–20, 1979.

[9] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[10] L. M. Rios and N. V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, Jul 2013.

[11] W. Sun, J. van den Berg, and R. Alterovitz. *Stochastic Extended LQR: Optimization-Based Motion Planning Under Uncertainty*, pages 609–626. Springer International Publishing, 2015.

[12] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1168–1175, May 2014.

[13] E. Todorov and W. Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proc. of the Am. Control Conf.*, pages 300–306 vol. 1, June 2005.

[14] J. van den Berg. *Extended LQR: Locally-Optimal Feedback Control for Systems with Non-Linear Dynamics and Non-Quadratic Cost*, pages 39–56. Springer International Publishing, 2016.