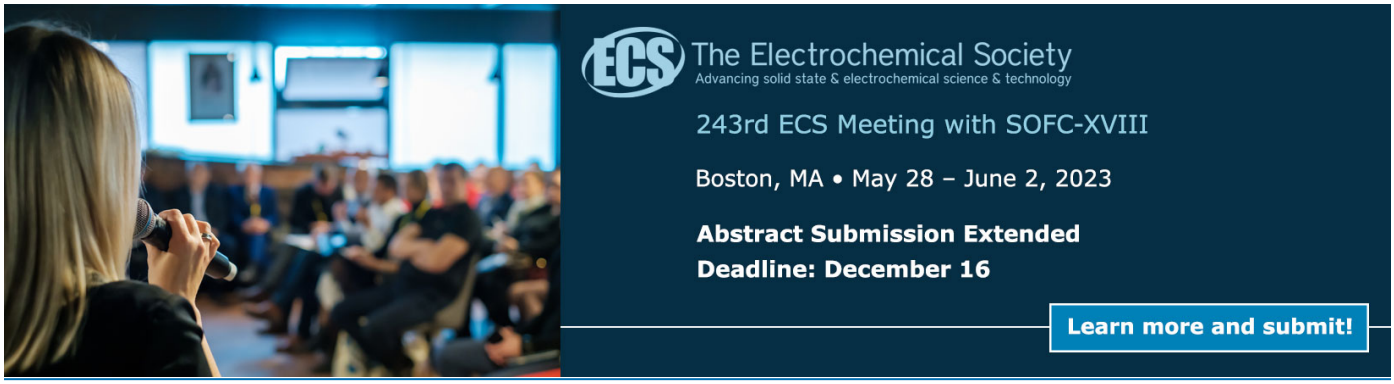
Locally optimal control under unknown dynamics with learnt cost function: application to industrial robot positioning



IOP Conf. Series: Journal of Physics: Conf. Series **783** (2017) 012036 doi:10.1088/1742-6596/783/1/012036

Locally optimal control under unknown dynamics with learnt cost function: application to industrial robot positioning

Joris Guérin, Olivier Gibaru, Stéphane Thiery and Eric Nyiri
Arts et Métiers ParisTech, 8 Boulervard Louis XIV, 59800, Lille, FRANCE
E-mail: `firstname.lastname@ensam.eu`

Abstract.
Recent methods of Reinforcement Learning have enabled to solve difficult, high dimensional, robotic tasks under unknown dynamics using iterative Linear Quadratic Gaussian control theory. These algorithms are based on building a local time-varying linear model of the dynamics from data gathered through interaction with the environment. In such tasks, the cost function is often expressed directly in terms of the state and control variables so that it can be locally quadratized to run the algorithm. If the cost is expressed in terms of other variables, a model is required to compute the cost function from the variables manipulated. We propose a method to learn the cost function directly from the data, in the same way as for the dynamics. This way, the cost function can be defined in terms of any measurable quantity and thus can be chosen more appropriately for the task to be carried out. With our method, any sensor information can be used to design the cost function. We demonstrate the efficiency of this method through simulating, with the V-REP software, the learning of a Cartesian positioning task on several industrial robots with different characteristics. The robots are controlled in joint space and no model is provided a priori. Our results are compared with another model free technique, consisting in writing the cost function as a state variable.

1. Introduction

Optimal feedback control theory provides an efficient framework for robot manipulators movement generation as it enables to compute both an optimal open-loop trajectory and a feedback controller at once. To carry out such tasks, classical optimal control uses a model of the dynamics and selects the solution that minimizes a properly designed cost function. Under linear dynamics and quadratic cost, an optimal solution can be found analytically thanks to the widely studied theory of Linear-Quadratic-Regulators (LQR) [1]. When the dynamics are more complex and non-linear, the problem becomes more difficult. However, iterative Linear-Quadratic-Regulator algorithm (iLQR) [2], [3] still enables to converge towards a locally optimal solution by iteratively fitting local linear approximations to the dynamics.

On the other hand, autonomous learning of manipulation skills have received much interest over the past decades. Thanks to different Reinforcement Learning (RL) techniques [4], various problems involving robot manipulators have been explored and successfully applied [5], [6], [7]. Among RL techniques, policy search methods seem to be the most appropriate for high dimensional robotics control problems [8]. Recently, several researchers have proposed methods

to link these two fields by using iLQR to compute control laws under unknown dynamics [9], [10]. This approach is interesting as it removes the need to model the dynamics, which can be difficult for complex systems and environments.

Besides the dynamics, an optimal control problem is also defined by its cost function. In order to find a good trajectory, we need to design a proper cost function in relation to the task to be carried out. Nevertheless, in most applications, we acknowledge that the cost function is quadratic and expressed explicitly in terms of the state and control variables. Such form of the cost function eases the implementation of the iLQR algorithm and appears to suit many applications quite well. However, in some cases, it can be useful to formulate the cost function in terms of variables that are not explicitly the state and control vectors. For instance, [11] proposes a cost function that is defined in the Cartesian space whereas the robot is controlled in the joint space. In such cases, we need to have geometric and kinematic models of the system (e.g., for robot manipulator, a direct Denavit-Hartenberg (DH) model [12]) to be able to express analytically the quadratic expansion of the cost function, which is necessary to run iLQR. In robotics, the need for a DH-model can be problematic as it requires a lot of efforts to be calibrated precisely [13], [14]. Moreover, the calibration is very specific to a certain robot [15], [16]. It is literally impossible to derive a model taking into account every dynamic phenomena. For example, if the payload on the robot is changed, the model calibration might not be valid anymore as strains might appear. When it comes to autonomous learning, it does not seem appropriate to have to redesign a model after any change in the system.

In this paper, we propose a method to avoid such problem by learning the quadratic approximation of the cost function in the same way as for the dynamics. Thanks to this method, the cost function can be defined from any measurable quantity, which allows to have a more appropriate cost in some cases. Moreover, the cost function can be defined in a more intuitive way and robot programming can become more accessible to non-specialists. The method is validated through simulation using the V-REP software [17]. Robots with very different specifications (number of joints, length of links, ...) are trained to learn a simple Cartesian positioning task using only position sensors information and no DH-model of any robot. The cost function is the distance measured between the end effector and the target Cartesian position while the robot is controlled in joint space. We also compare our results with another method consisting in including the Cartesian distance in the state vector, thus making the cost a function of the dynamics.

The paper is organized as follows. The derivation of the iLQR algorithm is presented in Section 2 and a new first order method to update the controller is proposed. Section 3 introduces a quadratic regression method for the cost function estimation. Simulation validation of the method is presented in Section 4 and Section 6 proposes a discussion on the results and future work.

2. TRAJECTORY OPTIMIZATION USING ILQR

iLQR is a trajectory optimization method to compute iteratively a locally optimal linear-Gaussian controller. In this section, the algorithm is derived (following [3]) in order to ease the understanding of the main contribution in Section 3. In this paper, no noise is included in the equations, yet it does not require much changes to take noise into account.

2.1. Definitions and overview

To understand the concept of iLQR, a few terms need to be introduced.

A trajectory of length T is defined by the set

$$\{x_0, u_0, l_0, x_1, u_1, l_1, ..., u_{T-1}, l_{T-1}, x_T\},$$

where x_t and u_t represent respectively state and control vectors, and l_t represents the value of the cost function. For example, for serial robot control, x can be joint positions and u target angular positions for next step. Such a state vector is easy to obtain with industrial robots and thus it makes sense to consider full-state feedback in the remaining of this section.

The cost and dynamics functions which map actual state and action to cost and new state are represented by

$$l_t = L_t(x_t, u_t), \tag{1}$$

$$x_{t+1} = F_t(x_t, u_t). \tag{2}$$

In the general framework of iLQR, L_t and F_t are complex non-linear functions. In Section 3, it is shown that they can even be unknown functions which values are only measured.

The deterministic controller to be learned is designated by Π. This controller is in the special form of a time-varying linear controller

$$u_t = \Pi(x_t) = K_t x_t + k_t. \tag{3}$$

We underline that the controller Π depends on time, however, the t index is dropped for better clarity.

The basic principle of iLQR is the following: from a nominal trajectory, denoted by

$$\{\bar{x}_0, \bar{u}_0, \bar{l}_0, \ldots, \bar{x}_T\},$$

a new controller, better than the previous one, is derived. From this new controller, a new nominal trajectory is computed. This process is repeated until a satisfactory controller is obtained. In most application, this iterative process turns out to converge rather rapidly towards a locally optimal trajectory.

2.2. Cost-to-go computation

From a nominal trajectory, the goal is to find a new, better controller. To do so, the first step is to compute local approximations of both (1) and (2) around the nominal trajectory. The dynamics are approximated linearly and the cost function quadratically as follows:

$$F_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) = \bar{x}_{t+1} + F_{x_t}\delta x_t + F_{u_t}\delta u_t, \tag{4}$$

$$\begin{aligned} L_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) &= \bar{l}_t + L_{x_t}\delta x_t + \frac{1}{2}\delta x_t^T L_{x,x_t}\delta x_t \\ &+ L_{u_t}\delta u_t + \frac{1}{2}\delta u_t^T L_{u,u_t}\delta u_t + \delta x_t^T L_{x,u_t}\delta u_t, \end{aligned} \tag{5}$$

where δx_t and δu_t represent variations from the nominal trajectory. Capital letters indexed by one (respectively two) letters represent sub-vectors (respectively sub-matrices) of the appropriate Jacobian (respectively Hessian) matrices.

These two local approximations are the key to all the derivation that follows. One way to compute these approximations is explored in Section 3.

For the sake of clarity, we also define an intermediate controller that is created during each improvement pass:

$$\delta u_t = \pi(\delta x_t) = P_t\delta x_t + p_t. \tag{6}$$

This controller enables to compute improvements on the time-varying linear controller Π. The P_t term improves the controller response to state variations and the p_t term is a shift in the nominal trajectory.

Finally, The two following value functions are defined:

$$Q_t^{\Pi}(x_t, u_t) = \sum_{j=t}^{T}(l_t), \tag{7}$$

$$V_t^{\Pi}(x_t) = \sum_{j=t}^{T}(l_t). \tag{8}$$

They represent the cost-to-go until the end of the trajectory if following Π. In the case of stochastic controllers, we would consider the expected values of the sums. These functions depend on Π, Q is defined with respect to x and u whereas V depends only on x. For clarity purposes, we drop the Π superscripts of Q and V in the remainder of this paper.

In the context of iLQR, quadratic Taylor expansions of both value functions are required to compute an improved controller:

$$\begin{aligned} Q_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) &= Q_{0_t} + Q_{x_t}\delta x_t + \frac{1}{2}\delta x_t^T Q_{x,x_t}\delta x_t \\ &+ Q_{u_t}\delta u_t + \frac{1}{2}\delta u_t^T Q_{u,u_t}\delta u_t + \delta x_t^T Q_{x,u_t}\delta u_t, \end{aligned} \tag{9}$$

$$V_t(\bar{x}_t + \delta x_t) = V_{0_t} + V_{x_t}\delta x_t + \frac{1}{2}\delta x_t^T V_{x,x_t}\delta x_t. \tag{10}$$

These quadratic approximations must be computed at each time step. This is done using different formulations for Q and V, widely used in RL:

$$Q_t(x_t, u_t) = L_t(x_t, u_t) + V_{t+1}(x_{t+1}), \tag{11}$$

$$V_t(x_t) = Q_t(x_t, \Pi(x_t)). \tag{12}$$

This point being reached, Q_t can be expressed from V_{t+1}. By plugging (4), (5) and (10) into (11), the coefficients of (9) are easily obtained:

$$\begin{aligned} Q_{0_t} &= \bar{l}_t + V_{0_{t+1}} + V_{x_{t+1}}\bar{x}_{t+1} + \frac{1}{2}\bar{x}_{t+1}^T V_{x,x_{t+1}}\bar{x}_{t+1}, \\ Q_{x_t} &= L_{x_t} + V_{x_{t+1}}F_{x_t} + \bar{x}_{t+1}^T V_{x,x_{t+1}}F_{x_t}, \\ Q_{x,x_t} &= L_{x,x_t} + F_{x_t}^T V_{x,x_{t+1}}F_{x_t}, \\ Q_{u_t} &= L_{u_t} + V_{x_{t+1}}F_{u_t} + \bar{x}_{t+1}^T V_{x,x_{t+1}}F_{u_t}, \\ Q_{u,u_t} &= L_{u,u_t} + F_{u_t}^T V_{x,x_{t+1}}F_{u_t}, \\ Q_{x,u_t} &= L_{x,u_t} + F_{x_t}^T V_{x,x_{t+1}}F_{u_t}. \end{aligned} \tag{13}$$

Then, we get (10) using (12) and (6) as follows,

$$\begin{aligned} V_{0_t} &= Q_{0_t} + Q_{u_t}p_t + \frac{1}{2}p_t^T Q_{u,u_t}p_t, \\ V_{x_t} &= Q_{x_t} + Q_{u_t}P_t + p_t^T Q_{u,u_t}P_t + p_t^T Q_{x,u_t}^T, \\ V_{x,x_t} &= Q_{x,x_t} + P_t^T Q_{u,u_t}P_t + Q_{x,u_t}P_t. \end{aligned} \tag{14}$$

Once all the value functions have been computed backward in time (starting from $V_T = 0$), the idea is to find the controller with the lowest cost-to-go at each time step.

2.3. Update the controller

In order to improve the controller, δu_t that minimizes Q_t needs to be computed at each time step t. Hence, we start by isolating the terms of Q_t depending on δu_t, which gives the following function to minimize,

$$f(\delta x_t, \delta u_t) = (Q_{u_t} + \delta x_t^T Q_{x,u_t})\delta u_t + \frac{1}{2}\delta u_t^T Q_{u,u_t}\delta u_t. \tag{15}$$

Then, we derive this function with respect to δu_t:

$$\frac{\partial f}{\partial \delta u_t}(.) = Q_{u_t} + \delta x_t^T Q_{x,u_t} + \delta u_t^T Q_{u,u_t}. \tag{16}$$

Note that this partial derivative is in the Jacobian form (row wise).

To optimize the controller, several methods (first and second order) have been applied [3]. In this paper, it has been chosen to implement gradient descent, which is more intuitive, in order to ease the characterization of the method (see Section 3). Thus, steps towards better controllers are made in the opposite direction of the gradient.

$$\nabla_{\delta u_t} f(\delta x_t, 0) = Q_{u_t}^T + Q_{x,u_t}^T \delta x_t. \tag{17}$$

That is to say that given the same input δx_t, the new chosen δu_t should be

$$\delta u_t = -\epsilon(Q_{u_t}^T + Q_{x,u_t}^T \delta x_t). \tag{18}$$

Obviously, the model being only valid locally, the step ϵ should be chosen small enough not to violate a "proximity to nominal" constraint.

In practice, we have implemented a new method, similar to super-SAB [18]. The variance for exploration (see Section 3) is modified according to the evolution of the partial derivatives signs, and ϵ is tuned to stay inside the variance boundaries. In this way, as we get closer to the optimal solution, the exploration gets tighter and the improvements are more accurate. This controller improvement method appears to converge within a reasonable number of steps for positioning task (Section 4).

From (18) and (6), we identify the temporary controller parameters

$$P_t = -\epsilon Q_{x,u_t}^T, \tag{19}$$

$$p_t = -\epsilon Q_{u_t}^T. \tag{20}$$

From this temporary controller, the global controller can be updated as follows

$$\left.\begin{array}{l} u_t = \hat{u}_t + \delta u_t \\ x_t = \bar{x}_t + \delta x_t \\ \hat{u}_t = \hat{K}_t x_t + \hat{k}_t \end{array}\right\} u_t = \hat{K}_t x_t + \hat{k}_t + P_t(x_t - \bar{x}_t) + p_t$$

where $\hat{u}_t$ represents the action chosen under the current controller, represented by $\hat{K}_t$ and $\hat{k}_t$.

Finally, the global controller is modified

$$K_t = \hat{K}_t + P_t, \tag{21}$$

$$k_t = \hat{k}_t + p_t - P_t \bar{x}_t. \tag{22}$$

To obtain the new average trajectory, this new controller just need to be applied to the initial state.

3. COMPUTE COST AND DYNAMICS FROM REAL-WORLD INTERACTION

Several authors have already proposed regression methods to compute the dynamics [9], [10]. The idea is to replace the current controller by a stochastic Gaussian controller of same mean and a with certain covariance matrix Σ_t:

$$P(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, \Sigma_t),$$

where $\mathcal{N}$ represents a normal distribution. Then, we generate several trajectories on the system using this controller and record all the x_t's and u_t's. In this way, the environment is explored around the nominal trajectory and (4) can be computed in this area, using linear regression.

In this section, the focus is on a method to compute the cost function, expressed in the output space, from experience. To our knowledge, such approach of combining cost function regression and iLQR has never been proposed yet.

3.1. Usefulness of the method

As shown in Section 2, once the dynamics have been linearized and the cost quadratized, improving the controller is just a few calculations away. In the original paper for iLQR [3], equations (1) and (2) are considered to be known and their partial derivatives computable. Hence, both Taylor expansions are computed analytically. Later on, it was proposed to compute the linear dynamics through regression on observed values [10], [9]. Thus, the need to have a representation of the environment, which can be tricky, is avoided.

However, the cost function still needs to be expressed directly in terms of the state and action variables. If we want to design a cost function from sensor measurements, it might not be straight forward to obtain the partial derivatives with respect to the state and control vectors. A good example is robot positioning task, where the cost function is a Cartesian distance and the variables controlled are joint positions (Section 4). In such case, a geometric model of the robot [12] is required to express the cost with respect to the state variables. Such model can be hard to obtain with enough precision (e.g. humanoid robots with a lot of degrees of freedom). Indeed, building a precise model of a robot manipulator requires a lot of work [13]. Moreover, the need for a model of the robot reduces the generality of the algorithm and its precision is dependant on the model, which varies with the robot used, its loading, ...

One possibility to avoid modelling the robot is to increase the state vector with a measurement directly proportional to the cost function. For example, the Cartesian distance for the positioning task in Section 4, can be considered to be part of the state. Doing so, the cost function can be expressed directly from the state and the dynamics, thus, the derivation can be done. This trick enables to define the cost function from any measurable quantity.

The method detailed in the following section provides a learning technique to compute the cost function. This method uses data to directly compute the quadratic approximation from regression.

Figure 1, illustrates all three methods in the form of block diagrams. This helps to visualize that the first method (Figure 1.a.) relies on a geometric model of the robot, which we do not want for the reasons explained above. In the second method (Figure 1.b.), the cost computation is dependent on a first order regression whereas in the proposed method (Figure 1.c.) the cost function is directly approximated quadratically. These two methods efficiencies are compared in Section 4.

3.2. Technical explanations

The technique described here uses the same idea as it was done for the dynamics. However, the model is no more linear but quadratic. For this reason, we need to use polynomial regression

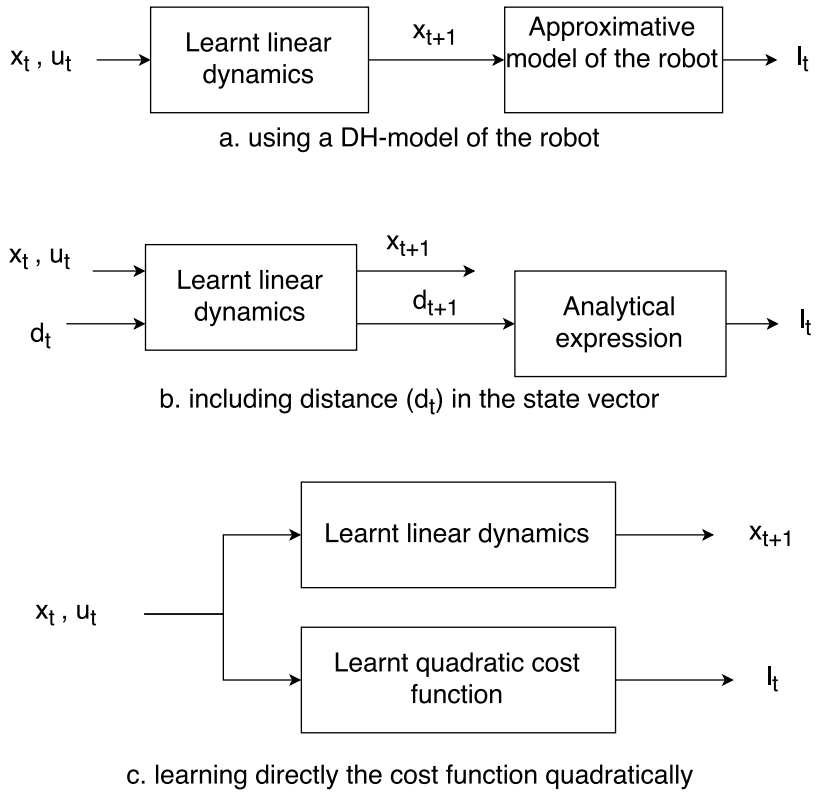


Figure 1. Different ways to compute a Cartesian distance cost function (l_t) from angular state (x_t) and control (u_t) vectors.

[19]. The trick for quadratic regression consists in noticing that a second order polynomial is a linear combination of the squared coefficients. For example, in the two dimensional case, a quadratic regression of

$$f : \mathbb{R}^2 \to \mathbb{R},$$
$$z = f(x, y),$$

is in fact a simple multivariate linear regression with input data of the form

$$v = [1, x, x^2, y, y^2, xy]^T,$$

which provides the coefficients of

$$z = a_0 + a_1 x + a_2 x^2 + a_3 y + a_4 y^2 + a_5 xy.$$

In the general case, a quadratic expansion of the cost with respect to δx_t and δu_t(respectively n and m entries) is obtained with a training set in the form

$$v = [1, \underbrace{\delta x_1, \delta x_2, ..., \delta x_n}_{n}, \underbrace{\delta x_1^2, \delta x_1 \delta x_2, ..., \delta x_n^2}_{\frac{n(n+1)}{2}}, \underbrace{\delta u_1, ..., \delta u_n}_{m}, \underbrace{\delta u_1^2, ..., \delta u_n^2}_{\frac{m(m+1)}{2}}, \underbrace{\delta x_1 \delta u_1, ..., \delta x_n \delta u_n}_{n*m}]^T.$$

The only drawback is that the new regression is of higher dimension than the one for dynamics, which involves a higher number of samples to be an accurate approximation. However, all the samples found during the dynamics exploration phase work just as well and do not have to

be re-explored. All that is required is to record the cost at the same time than the new state observed. Moreover, the controller is learned off-line and even if the learning time is longer, the on-line speed of the controller is not impacted.

Once this regression is done, an easy reshaping of the linear coefficients found provides the matrix coefficients of

$$\begin{aligned} L_t(\bar{x}_t + \delta x_t, \bar{u}_t + \delta u_t) &= \bar{l}_t + L_{x_t}\delta x_t + \frac{1}{2}\delta x_t^T L_{x,x_t}\delta x_t \\ &+ L_{u_t}\delta u_t + \frac{1}{2}\delta u_t^T L_{u,u_t}\delta u_t + \delta x_t^T L_{x,u_t}\delta u_t \end{aligned}$$

which can be used directly for an iLQR update.

4. SIMULATION RESULTS

4.1. Problem definition

In order to show the effectiveness of the method, a robot manipulator positioning task has been implemented using the V-REP simulation software. The robot is controlled in angular position, the state vector is the vector of joint coordinates and the control vector is the desired angular positions for next step. The goal is to reach a target Cartesian point with the end-effector of the robot. The cost function is expressed in the most natural way possible: it is the Cartesian distance between the connector and the target. On Figures 2 and 3, the end-effector is represented by the black connectors at the extremity of the robots. The objective is displayed with the red floating sphere (the target is actually the center of the sphere). The distance (cost function) is extracted directly from the software. In real physical applications, such a cost can be provided by any distance measurement sensor (e.g. laser tracker).

We have chosen this demonstration problem because results can be interpreted intuitively. Distances can be compared easily and are directly related to precision. Such an application, even if not really challenging in robotics, is a good start to evaluate the efficiency of the proposed cost regression method before implementing it for more complex robotics tasks.

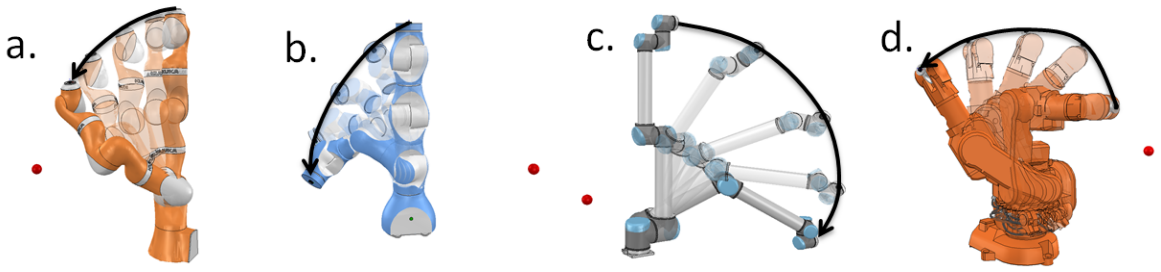


Figure 2. Initial, randomly chosen, nominal trajectory for different industrial robots. a. KUKA LBR iiwa / b. F&P P-ARM / c. UR 10 / d. ABB IRB 140

To solve such a control problem with classical methods, a geometric model of the robot (e.g. DH-model), giving the end-effector position with respect to joint coordinates needs to be built. Indeed, to be able to get the quadratic approximation of the cost, we need to map analytically the joint space to the Cartesian space in order to differentiate it. Obviously, no such model has been derived for the simulations and, using V-REP, we implemented the algorithm on four industrial robots with different specifications (number of links, lengths of links, ...) to illustrate that the method is model-free (Figure 2 and 3). Since the method does not need to explicit a

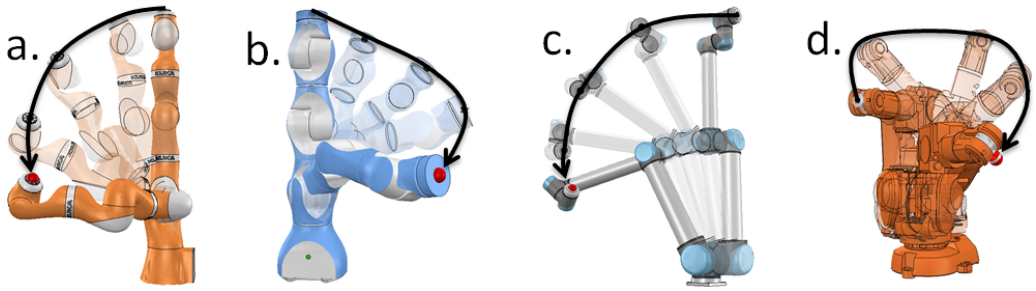


Figure 3. Trajectory learned for positioning task for different industrial robots. a. KUKA LBR iiwa / b. F&P P-ARM / c. UR 10 / d. ABB IRB 140

model, it can be applied directly to any robot. Thanks to this method, we get rid of modelling imprecision and only sensor precision is involved in the learning process.

It is also worth underlining that the task would not have been different if adding orientation constraint to the end-effector. All that is needed is to take distances between three points instead of only one.

4.2. Trajectory found on different robots

The nominal trajectory used to initialize the iLQR algorithm was chosen randomly. We chose to set all desired joint positions to 60° and explore around this initial command. Figure 2 represents the initial nominal trajectory, the distance between the end-effector and the target is 490 mm. Note that the angular point-to-point (PTP) movement between two joints configurations is considered as a black box, part of the V-REP environment. In this way, the dynamics can really be considered unknown.

Figure 3 represents a set of final trajectories found for the positioning task. The algorithm stopped running when a precision of 0.1 mm was reached. As it can be seen, iLQR with no model of the robot and Cartesian cost function succeeded to reach its objective point with various robots. Indeed, they have different lengths, different numbers of links, but the algorithm can still converge towards a solution without being restricted on the cost function definition.

4.3. Number of samples needed

Figure 4 shows how the number of samples influences the rate of convergence on one specific robot (KUKA LBR iiwa). The y-axis scale is too small to see the distances for the first iterations, but this choice is necessary to be able to differentiate curves in the final stages.

The first thing we notice is that for a sample count above 40, the solution does not evolve anymore, which means that 40 samples is enough to characterize with high precision the local cost function. In simulation, we found that for the positioning task, it is possible to find a trajectory with a precision greater than 0.1 mm if we do the off-line learning with at least 40 samples during 20 iterations.

On the other hand, if the number of samples gathered is too small (e.g. 5), the cost and dynamics will not be properly approximated and the algorithm might not converge towards the desired solution, as shown in Figure 4.

4.4. Comparison with state modification method

As mentioned in Section 3, another method to avoid robot modelling consist in modifying the state vector and include the Cartesian distance in it. This technique corresponds to diagram

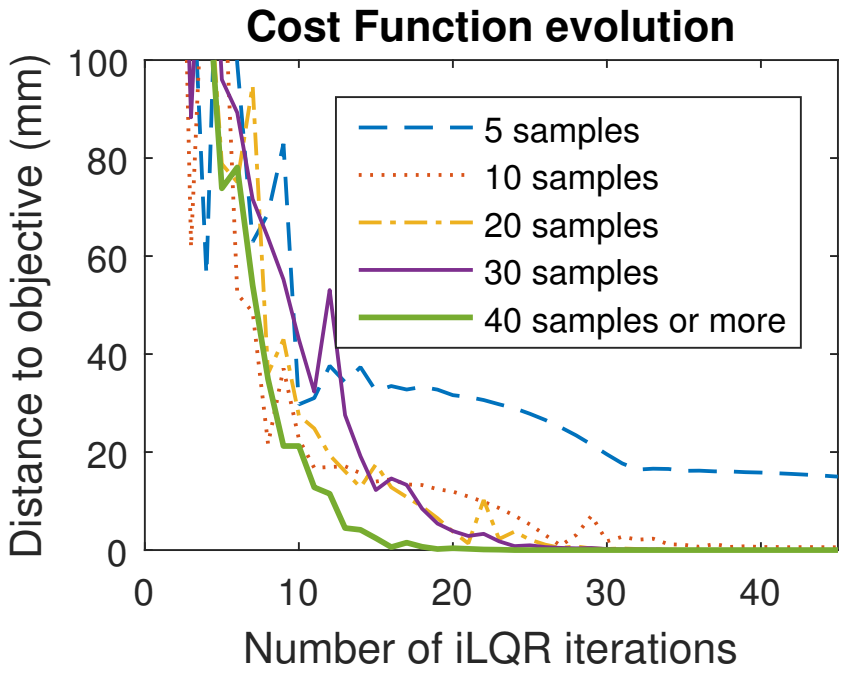


Figure 4. Distance to target point with respect to the number of iLQR passes. Different curves represent different number of samples generated for learning the cost and dynamics. Initial distance was around 650 mm

b. of Figure 1. In this section, our new quadratic regression method is compared with this relatively simple technique. Figure 5 illustrates the learning curves for both methods under different learning sample sizes.

We acknowledge that state modification is less stable and robust. Indeed, in this simulation, the results using 30 samples were better than the ones with 40, which did not happen using quadratic approximation. The state modification method is more sensitive to the samples generated whereas quadratic regression is behaving more regularly for a given number of samples. The state modification does not exhibit the nice property of being independent of the number of samples when this number becomes high (> 40).

On the other hand, with very small number of samples (e.g. 5), it is better to stick to first order approximation. Indeed, as shown on the top left plot of Figure 5, state modification works better for small number of samples because quadratic regression is not accurate enough.

Finally, the quadratic regression appears to be more efficient in the final stages of the algorithm, which might come from the fact that a better model of the cost is required to reach high precision in the positioning.

5. Conclusion

In order to democratize robotics in small companies, there is a need for self-programming robots. An unqualified worker should be able to program a robot only by specifying the task to accomplish. iLQR under unknown dynamics seems a good approach for learning local policies in a short amount of time and has already been used to solve robotics problem by several authors [9],[10]. However, such a learning algorithm should be independent of the robot model, tool and calibration so that it can be applied by any worker. This limitation eliminates approaches based on the geometric model of the robot [11]. The only real model-free method was proposed in [10], and was compared to our polynomial regression learning of the cost function approach.

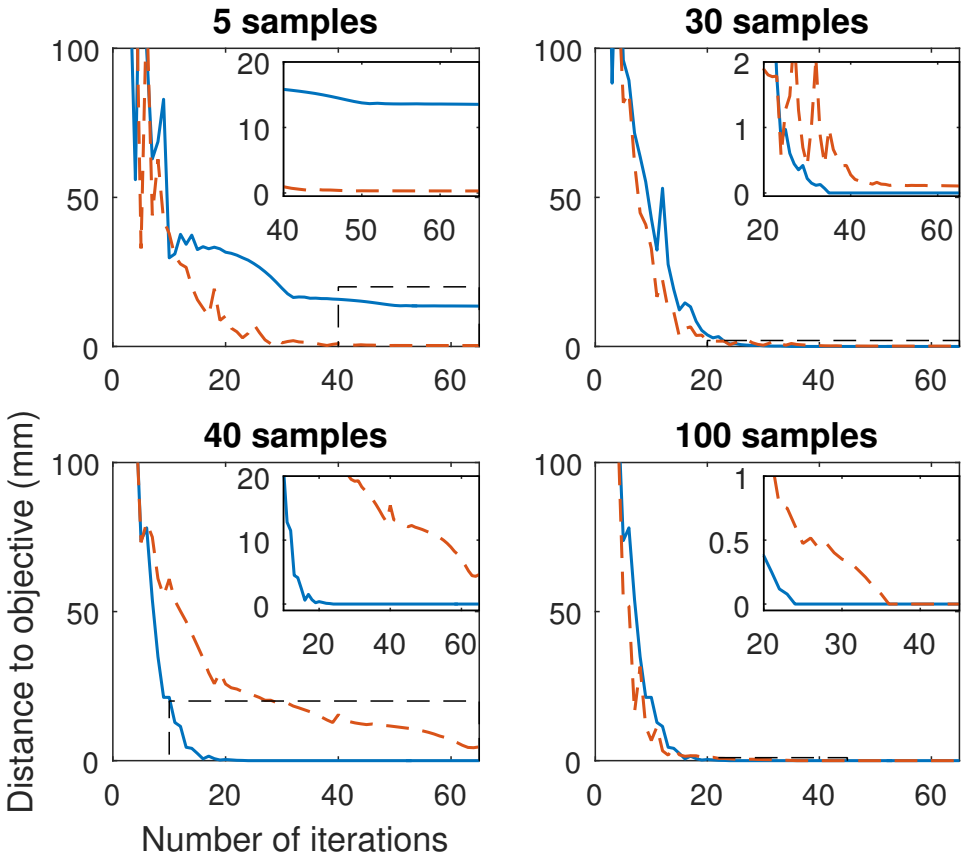


Figure 5. Comparisons of two methods: Quadratic regression (plain blue lines) and modified state (dotted orange lines). Right-upper boxes on each figures are zoom on the final stages of iLQR. Initial distance was around 650 mm

Results obtained on simulation show that our approach converges faster towards high precision behaviors and is more stable and robust if the exploration phase gathers enough samples. Hence, for industrial application, we recommend to learn the quadratic cost function directly from data.

6. Discussion

6.1. Sample count reduction

Despite the number of samples needed can be considered high in the simulation validation, we would like to emphasis that there exists tools, that have already been used in other work [11], [10], to reduce sample count and optimize regressions. In this paper, the idea was to illustrate that it is possible to draw an optimal control problem with a cost function being any measurable quantity, without any geometric model of the system. We did not implement any algorithm optimization, which will be done in future work. Moreover, the learning time does not impact the speed of the final controller as the learning process is realized off-line.

6.2. Link with Policy Search

The process of learning Linear Gaussian Controllers (LGC) is a necessary step for Guided Policy Search (GPS) algorithm [10], [11]. GPS is a very promising method that has proven its ability to succeed for complex RL tasks, particularly in robotics. Indeed, it scales gracefully to high dimensional state spaces and can learn highly general policies for complex tasks thanks to guiding

trajectories in the form of LGC. Thus, any improvement on the trajectory optimization methods to learn such controllers benefits directly to the global method. Implementing our method in the framework of GPS is also an axis of future research for our team.

6.3. Handling contact

As explained earlier, we have chosen to control the robot in its angular space in order to ease the results interpretation and the comparison with other methods. Such high level controller has the disadvantage of being less flexible then lower level ones. For example, pure angular position control is not able to deal with contact because there are missing information for feedback. Our plan in the close future being to deal with both grasping and insertion tasks, we need to start including force and torque information in both the control and state vectors. Such future needs explain the use of the KUKA LBR iiwa robot, which is equipped with high accuracy torque sensors in each joint.

References

[1] Lewis F L, Vrabie D and Syrmos V L 2012 *Optimal Control* (John Wiley & Sons)
[2] Li W and Todorov E 2004 Iterative linear quadratic regulator design for nonlinear biological movement systems *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics* pp 222–229
[3] Todorov E and Li W 2005 A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems *American Control Conference, 2005. Proceedings of the 2005* pp 300–306 vol. 1
[4] Kober J and Peters J 2012 Reinforcement learning in robotics: A survey *Reinforcement Learning* (Springer) pp 579–610
[5] Lin C K 2009 H reinforcement learning control of robot manipulators using fuzzy wavelet networks *Fuzzy Sets and Systems* **160** 1765–1786
[6] Deisenroth M P, Rasmussen C E and Fox D 2011 Learning to control a low-cost manipulator using data-efficient reinforcement learning *MIT Press*
[7] Park J J, Kim J H and Song J B 2007 Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning *International Journal of Control Automation and Systems* **5** 674
[8] Deisenroth M P, Neumann G, Peters J *et al.* 2013 A survey on policy search for robotics. *Foundations and Trends in Robotics* **2** 1–142
[9] Mitrovic D, Klanke S and Vijayakumar S 2010 Adaptive optimal feedback control with learned internal dynamics models *From Motor Learning to Interaction Learning in Robots* (Springer) pp 65–84
[10] Levine S and Abbeel P 2014 Learning neural network policies with guided policy search under unknown dynamics *Neural Information Processing Systems (NIPS)*
[11] Levine S, Wagener N and Abbeel P 2015 Learning contact-rich manipulation skills with guided policy search *CoRR* **abs/1501.05611**
[12] Dombre E and Khalil W 2013 *Robot manipulators: modeling, performance analysis and control* (John Wiley & Sons)
[13] Majarena A C, Santolaria J, Samper D and Aguilar J J 2010 An overview of kinematic and calibration models using internal/external sensors or constraints to improve the behavior of spatial parallel mechanisms *Sensors* **10** 10256–10297
[14] Elatta A, Gen L P, Zhi F L, Daoyuan Y and Fei L 2004 An overview of robot calibration *Information Technology Journal* **3** 74–78
[15] Nubiola A and Bonev I A 2013 Absolute calibration of an abb irb 1600 robot using a laser tracker *Robotics and Computer-Integrated Manufacturing* **29** 236–245
[16] Jubien A, Gautier M and Janot A 2014 Dynamic identification of the kuka lwr robot using motor torques and joint torque sensors data *World Congress of the International Federation of Automatic Control (IFAC)* pp 1–6
[17] E Rohmer S P N Singh M F 2013 V-rep: a versatile and scalable robot simulation framework *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*
[18] Allard R and Faubert J 2004 Neural networks: different problems require different learning rate adaptive methods *Electronic Imaging 2004* vol 5298 pp 516–527
[19] De Brabanter K, De Brabanter J, De Moor B and Gijbels I 2013 Derivative estimation with local polynomial fitting *The Journal of Machine Learning Research* **14** 281–301