A practical guide to learning and control problems in robotics solved with second-order optimization

Sylvain Calinon, Idiap Research Institute

Contents

1 Introduction

This technical report presents several learning and optimal control techniques in robotics in the form of simple toy problems that can be easily coded. It comes as part of **Robotics Codes From Scratch (RCFS)**, a website containing interactive sandbox examples and exercises, together with a set of standalone source code examples gathered in a git repository, which can be accessed at:

`https://robotics-codes-from-scratch.github.io`

Each section in this report lists the corresponding source codes in Python and Matlab (ensuring full compatibility with GNU Octave), as well as in C++ and Julia for some of the principal examples, which can be accessed at:

`https://gitlab.idiap.ch/rli/robotics-codes-from-scratch`

2 Newton's method for minimization

We would like the find the value of a decision variable x that would give us a cost $c(x)$ that is a small as possible, see Figure 1. Imagine that we start from an initial guess x_1 and that can observe the behavior of this cost function within a very small region around our initial guess. Now let's assume that we can make several consecutive guesses that will each time provide us with similar local information about the behavior of the cost around the points that we guessed. From this information, what point would you select as second guess (see question marks in the figure), based on the information that you obtained from the first guess? There were two relevant information in the small portion of curve that we can observe in the figure to make a smart choice. First, the trend of the curve indicates that the cost seems to decrease if we move on the left side, and increase if we move on the right side. Namely, the slope $c'(x_1)$ of the function $c(x)$ at point x_1 is positive. Second, we can observe that the portion of the curve has some curvature that can be also be informative about the way the trend of the curve $c(x)$ will change by moving to the left or to the right. Namely, how much the slope $c'(x_1)$ will change, corresponding to an acceleration $c''(x_1)$ around our first guess x_1. This is informative to estimate how much we should move to the left of the first guess to wisely select a second guess.

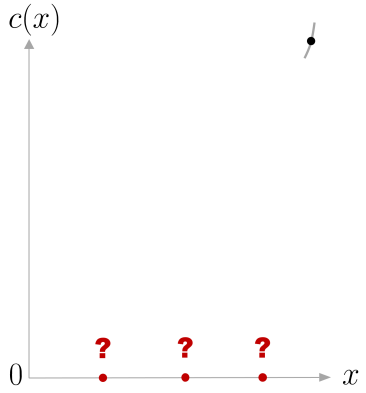


Figure 1: Problem formulation.

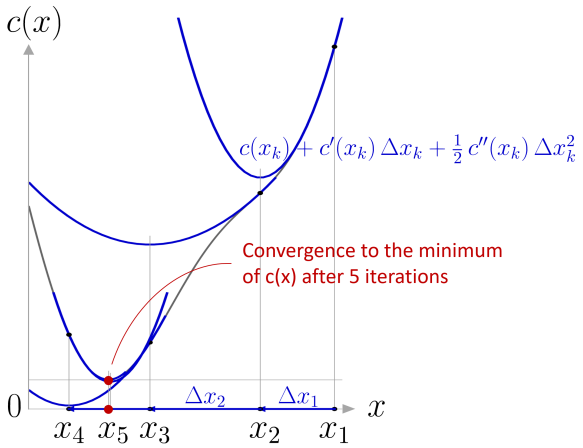


Figure 2: Newton's method for minimization, starting from an initial estimate x_1 and converging to a local minimum (red point) after 5 iterations.

Now that we have this intuition, we can move to a more formal problem formulation. Newton's method attempts to solve $\min_x c(x)$ or $\max_x c(x)$ from an initial guess x_1 by using a sequence of **second-order Taylor approximations** of $c(x)$ around the iterates, see Fig. 2.

The second-order Taylor expansion around the point x_k can be expressed as

$$c(x_k+\Delta x_k) \approx c(x_k) + c'(x_k)\,\Delta x_k + \frac{1}{2}c''(x_k)\,\Delta x_k^2, \quad (1)$$

where $c'(x_k)$ and $c''(x_k)$ are the first and second derivatives at point x_k.

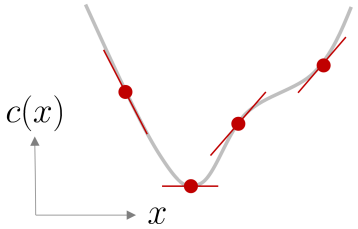


Figure 3: Finding local optima by localizing the points whose derivatives are zero (horizontal slopes).

We are interested in solving minimization problems with this approximation. If the second derivative $c''(x_k)$ is positive, the quadratic approximation is a convex function of Δx_k, and its minimum can be found by setting the derivative to zero.

To find the local optima of a function, we can localize the points whose derivatives are zero, see Figure 3 for an illustration.

Thus, by differentiating (1) w.r.t. Δx_k and equating to zero, we obtain

$$c'(x_k) + c''(x_k)\,\Delta x_k = 0,$$

meaning that the minimum is given by

$$\Delta \hat{x}_k = -\frac{c'(x_k)}{c''(x_k)}$$

which corresponds to the offset to apply to x_k to minimize the second-order polynomial approximation of the cost at this point.

It is important that $c''(x_k)$ is positive if we want to find local minima, see Figure 4 for an illustration.

By starting from an initial estimate x_1 and by recursively refining the current estimate by computing the offset that would minimize the polynomial approximation of the cost at the current estimate, we obtain at each iteration k the recursion

$$x_{k+1} = x_k - \frac{c'(x_k)}{c''(x_k)}. \quad (2)$$

The geometric interpretation of Newton's method is that at each iteration, it amounts to the fitting of a paraboloid to the surface of $c(x)$ at x_k, having the same slopes and curvature as the surface at that point, and then proceeding to the maximum or minimum of that paraboloid. Note that if $c(x)$ is a quadratic function, then the exact extremum is found in one step, which corresponds to the resolution of a least-squares problem.

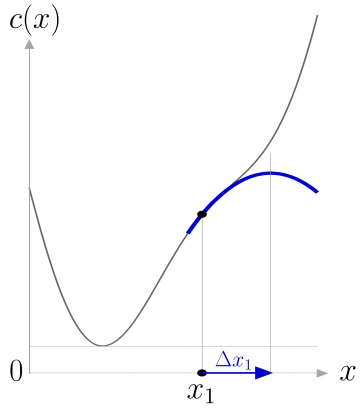


Figure 4: Newton update that would be achieved when the second derivative is negative.

Multidimensional case

For functions that depend on multiple variables stored as multidimensional vectors $\boldsymbol{x}$, the cost function $c(\boldsymbol{x})$ can similarly be approximated by a second-order Taylor expansion around the point $\boldsymbol{x}_k$ with

$$c(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k) \approx c(\boldsymbol{x}_k) + \Delta\boldsymbol{x}_k^\top \frac{\partial c}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} + \frac{1}{2}\Delta\boldsymbol{x}_k^\top \frac{\partial^2 c}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k} \Delta\boldsymbol{x}_k, \quad (3)$$

which can also be rewritten in augmented vector form as

$$c(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k) \approx c(\boldsymbol{x}_k) + \frac{1}{2}\begin{bmatrix}1\\ \Delta\boldsymbol{x}_k\end{bmatrix}^\top \begin{bmatrix}0 & \boldsymbol{g}_{\boldsymbol{x}}^\top \\ \boldsymbol{g}_{\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}}\end{bmatrix} \begin{bmatrix}1\\ \Delta\boldsymbol{x}_k\end{bmatrix},$$

with gradient vector

$$\boldsymbol{g}(\boldsymbol{x}_k) = \frac{\partial c}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k}, \quad (4)$$

and Hessian matrix

$$\boldsymbol{H}(\boldsymbol{x}_k) = \frac{\partial^2 c}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k}. \quad (5)$$

We are interested in solving minimization problems with this approximation. If the Hessian matrix $\boldsymbol{H}(\boldsymbol{x}_k)$ is positive definite, the quadratic approximation is a convex function of $\Delta\boldsymbol{x}_k$, and its minimum can be found by setting the derivatives to zero, see Figure 5.

By differentiating (3) w.r.t. $\Delta\boldsymbol{x}_k$ and equation to zero, we obtain that

$$\Delta\hat{\boldsymbol{x}}_k = -\boldsymbol{H}(\boldsymbol{x}_k)^{-1}\,\boldsymbol{g}(\boldsymbol{x}_k),$$

is the offset to apply to $\boldsymbol{x}_k$ to minimize the second-order polynomial approximation of the cost at this point.

By starting from an initial estimate $\boldsymbol{x}_1$ and by recursively refining the current estimate by computing the offset that would minimize the polynomial approximation of the cost at the current estimate, we obtain at each iteration k the recursion

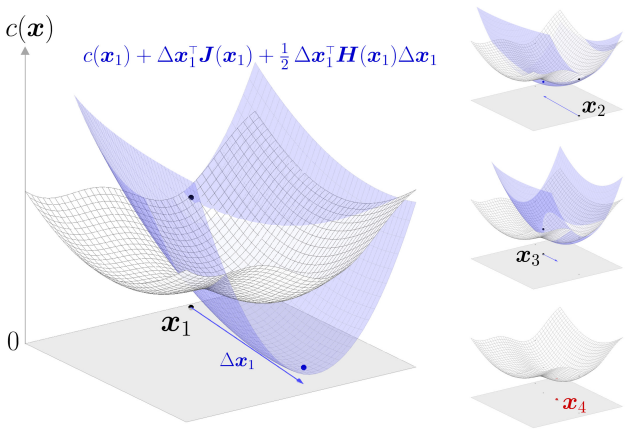


Figure 5: Newton's method for minimization with 2D decision variables.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}(\boldsymbol{x}_k)^{-1}\,\boldsymbol{g}(\boldsymbol{x}_k). \tag{6}$$

2.1 Gauss–Newton algorithm

The Gauss–Newton algorithm is a special case of Newton's method in which the cost is quadratic (sum of squared function values), with $c(\boldsymbol{x}) = \sum_{i=1}^{R} r_i^2(\boldsymbol{x}) = \boldsymbol{r}^\top\boldsymbol{r}$, where $\boldsymbol{r}$ are residual vectors. We neglect in this case the second-order derivative terms of the Hessian. The gradient and Hessian can in this case be computed with

$$\boldsymbol{g} = 2\boldsymbol{J}_{\boldsymbol{r}}^\top\boldsymbol{r}, \quad \boldsymbol{H} \approx 2\boldsymbol{J}_{\boldsymbol{r}}^\top\boldsymbol{J}_{\boldsymbol{r}},$$

where $\boldsymbol{J}_{\boldsymbol{r}} \in \mathbb{R}^{R\times D}$ is the Jacobian matrix of $\boldsymbol{r} \in \mathbb{R}^R$. This definition of the Hessian matrix makes it positive definite, which is useful to solve minimization problems as for well conditioned Jacobian matrices, we do not need to verify the positive definiteness of the Hessian matrix at each iteration.

The update rule in (6) then becomes

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \left(\boldsymbol{J}_{\boldsymbol{r}}^\top(\boldsymbol{x}_k)\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\right)^{-1}\boldsymbol{J}_{\boldsymbol{r}}^\top(\boldsymbol{x}_k)\,\boldsymbol{r}(\boldsymbol{x}_k)\\ &= \boldsymbol{x}_k - \boldsymbol{J}_{\boldsymbol{r}}^\dagger(\boldsymbol{x}_k)\,\boldsymbol{r}(\boldsymbol{x}_k),\end{aligned}$$

where $\boldsymbol{J}_{\boldsymbol{r}}^\dagger$ denotes the pseudoinverse of $\boldsymbol{J}_{\boldsymbol{r}}$.

The Gauss–Newton algorithm is the workhorse of many robotics problems, including inverse kinematics and optimal control, as we will see in the next sections.

2.2 Least squares

When the cost $c(\boldsymbol{x})$ is a quadratic function w.r.t. $\boldsymbol{x}$, the optimization problem can be solved directly, without requiring iterative steps. Indeed, for any matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$, we can see that if

$$c(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

derivating $c(\boldsymbol{x})$ w.r.t. $\boldsymbol{x}$ and equating to zero yields

$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0} \iff \boldsymbol{x} = \boldsymbol{A}^\dagger\boldsymbol{b},$$

which corresponds to the standard analytic least squares estimate. We will see later in the inverse kinematics section that for the complete solution can also include a nullspace structure.

2.3 Least squares with constraints

A constrained minimization problem of the form

$$\min_{\boldsymbol{x}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}), \quad \text{s.t.} \quad \boldsymbol{C}\boldsymbol{x} = \boldsymbol{h}, \tag{7}$$

can also be solved analytically by considering a Lagrange multiplier variable $\boldsymbol{\lambda}$ allowing us to rewrite the objective as

$$\min_{\boldsymbol{x},\boldsymbol{\lambda}} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{\lambda}^\top(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{h}).$$

Differentiating with respect to $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ and equating to zero yields

$$\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})+\boldsymbol{C}^\top\boldsymbol{\lambda}=\boldsymbol{0}, \quad \boldsymbol{C}\boldsymbol{x}-\boldsymbol{h}=\boldsymbol{0},$$

which can rewritten in matrix form as

$$\begin{bmatrix}\boldsymbol{A}^\top\boldsymbol{A} & \boldsymbol{C}^\top \\ \boldsymbol{C} & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{x} \\ \boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}\boldsymbol{A}^\top\boldsymbol{b} \\ \boldsymbol{h}\end{bmatrix}.$$

With this augmented state representation, we can then see that

$$\begin{bmatrix}\boldsymbol{x} \\ \boldsymbol{\lambda}\end{bmatrix}=\begin{bmatrix}\boldsymbol{A}^\top\boldsymbol{A} & \boldsymbol{C}^\top \\ \boldsymbol{C} & \boldsymbol{0}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{A}^\top\boldsymbol{b} \\ \boldsymbol{h}\end{bmatrix}$$

minimizes the constrained cost. The first part of this augmented state then gives us the solution of (7).

3 Forward kinematics (FK) for a planar robot manipulator

IK.*

The *forward kinematics* (FK) function of a planar robot manipulator is defined as

$$\begin{aligned}\boldsymbol{f} &= \begin{bmatrix}\boldsymbol{\ell}^\top \cos(\boldsymbol{L}\boldsymbol{x}) \\ \boldsymbol{\ell}^\top \sin(\boldsymbol{L}\boldsymbol{x}) \\ \mathbf{1}^\top \boldsymbol{x}\end{bmatrix} \\ &= \begin{bmatrix}\ell_1\cos(x_1)+\ell_2\cos(x_1+x_2)+\ell_3\cos(x_1+x_2+x_3)+\dots \\ \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2)+\ell_3\sin(x_1+x_2+x_3)+\dots \\ x_1+x_2+x_3+\dots\end{bmatrix},\end{aligned}$$

with $\boldsymbol{x}$ the state of the robot (joint angles), $\boldsymbol{f}$ the position of the robot end-effector, $\boldsymbol{\ell}$ a vector of robot links lengths, $\boldsymbol{L}$ a lower triangular matrix with unit elements, and $\mathbf{1}$ a vector of unit elements, see Fig. 6.

The position and orientation of all articulations can similarly be computed with the forward kinematics function

$$\begin{aligned}\tilde{\boldsymbol{f}} &= \left[\boldsymbol{L}\,\mathrm{diag}(\boldsymbol{\ell})\cos(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\,\mathrm{diag}(\boldsymbol{\ell})\sin(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\boldsymbol{x}\right]^\top \\ &= \begin{bmatrix}\tilde{f}_{1,1} & \tilde{f}_{1,2} & \tilde{f}_{1,3} & \dots \\ \tilde{f}_{2,1} & \tilde{f}_{2,2} & \tilde{f}_{2,3} & \dots \\ \tilde{f}_{3,1} & \tilde{f}_{3,2} & \tilde{f}_{3,3} & \dots\end{bmatrix},\end{aligned} \tag{8}$$

$$\begin{aligned}&\tilde{f}_{1,1} = \ell_1\cos(x_1), && \tilde{f}_{1,2} = \ell_1\cos(x_1)+\ell_2\cos(x_1+x_2), && \tilde{f}_{1,3} = \ell_1\cos(x_1)+\ell_2\cos(x_1+x_2)+\ell_3\cos(x_1+x_2+x_3), \\ \text{with}\quad &\tilde{f}_{2,1} = \ell_1\sin(x_1), && \tilde{f}_{2,2} = \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2), && \tilde{f}_{2,3} = \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2)+\ell_3\sin(x_1+x_2+x_3), \quad \dots \\ &\tilde{f}_{3,1} = x_1, && \tilde{f}_{3,2} = x_1+x_2, && \tilde{f}_{3,3} = x_1+x_2+x_3.\end{aligned}$$

$$\begin{bmatrix}f_1 \\ f_2 \\ f_3\end{bmatrix} = \begin{bmatrix}\ell_1\cos(x_1)+\ell_2\cos(x_1+x_2) \\ \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2) \\ x_1+x_2\end{bmatrix}$$

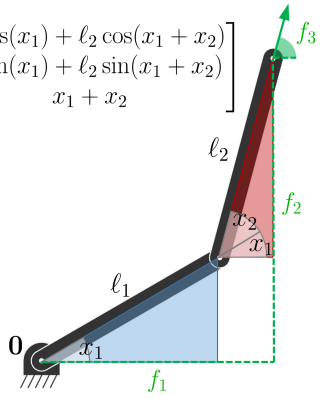


Figure 6: Forward kinematics for a planar robot with two links.

In Python, this can be coded for the end-effector position part as

```
D = 3 #State space dimension (joint angles)
x = np.ones(D) * np.pi / D #Robot pose
l = np.array([2, 2, 1]) #Links lengths
L = np.tril(np.ones([D,D])) #Transformation matrix
f = np.array([L @ np.diag(l) @ np.cos(L @ x), L @ np.diag(l) @ np.sin(L @ x)]) #Forward kinematics
```

4 Inverse kinematics (IK) for a planar robot manipulator

IK.*

By differentiating the forward kinematics function, a least norm inverse kinematics solution can be computed with

$$\dot{\boldsymbol{x}} = \boldsymbol{J}^{\dagger}(\boldsymbol{x})\ \dot{\boldsymbol{f}}, \tag{9}$$

where the Jacobian $\boldsymbol{J}(\boldsymbol{x})$ corresponding to the end-effector forward kinematics function $\boldsymbol{f}(\boldsymbol{x})$ can be computed as (with a simplification for the orientation part by ignoring the manifold aspect)

$$\begin{aligned}\boldsymbol{J} &= \begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x})^{\top}\mathrm{diag}(\boldsymbol{\ell})\boldsymbol{L} \\ \cos(\boldsymbol{L}\boldsymbol{x})^{\top}\mathrm{diag}(\boldsymbol{\ell})\boldsymbol{L} \\ \boldsymbol{1}^{\top} \end{bmatrix} \\ &= \begin{bmatrix} J_{1,1} & J_{1,2} & J_{1,3} & \dots \\ J_{2,1} & J_{2,2} & J_{2,3} & \dots \\ J_{3,1} & J_{3,2} & J_{3,3} & \dots \end{bmatrix},\end{aligned}$$

with

$$\begin{aligned} &J_{1,1} = -\ell_1\sin(x_1) - \ell_2\sin(x_1+x_2) - \ell_3\sin(x_1+x_2+x_3) - \dots, \\ &J_{2,1} = \ell_1\cos(x_1) + \ell_2\cos(x_1+x_2) + \ell_3\cos(x_1+x_2+x_3) + \dots, \\ &J_{3,1} = 1, \end{aligned} \quad \begin{aligned} &J_{1,2} = -\ell_2\sin(x_1+x_2) - \ell_3\sin(x_1+x_2+x_3) - \dots, \\ &J_{2,2} = \ell_2\cos(x_1+x_2) + \ell_3\cos(x_1+x_2+x_3) + \dots, \\ &J_{3,2} = 1, \end{aligned} \quad \begin{aligned} &J_{1,3} = -\ell_3\sin(x_1+x_2+x_3) - \dots, \\ &J_{2,3} = \ell_3\cos(x_1+x_2+x_3) + \dots, \\ &J_{3,3} = 1, \end{aligned} \quad \dots$$

In Python, this can be coded for the end-effector position part as

```
J = np.array([-np.sin(L @ x).T @ np.diag(l) @ L, np.cos(L @ x).T @ np.diag(l) @ L]) #Jacobian (for end-effector)
```

This Jacobian can be used to solve the *inverse kinematics* (IK) problem that consists of finding a robot pose to reach a target with the robot endeffector. The underlying optimization problem consists of minimizing a quadratic cost $c = \|\boldsymbol{f} - \boldsymbol{\mu}\|^2$, where $\boldsymbol{f}$ is the position of the endeffector and $\boldsymbol{\mu}$ is a target to reach with this endeffector. An optimization problem with a quadratic cost can be solved iteratively with a Gauss–Newton method, requiring to compute Jacobian pseudoinverses $\boldsymbol{J}^{\dagger}$, see Section 2.1 for details.

4.1 Numerical estimation of the Jacobian

IK_num.*

Section 4 above presented an analytical solution for the Jacobian. A numerical solution can alternatively be estimated by computing

$$J_{i,j} = \frac{\partial f_i(\boldsymbol{x})}{\partial x_j} \approx \frac{f_i(\boldsymbol{x}^{(j)}) - f_i(\boldsymbol{x})}{\Delta} \quad \forall i, \forall j,$$

with $\boldsymbol{x}^{(j)}$ a vector of the same size as $\boldsymbol{x}$ with elements

$$x_k^{(j)} = \begin{cases} x_k + \Delta, & \text{if } k = j, \\ x_k, & \text{otherwise,} \end{cases}$$

where Δ is a small value for the approximation of the derivatives.

4.2 Inverse kinematics (IK) with task prioritization

IK_nullspace.*

In (9), the pseudoinverse provides a single least norm solution. This result can be generalized to obtain all solutions of the linear system with

$$\dot{\boldsymbol{x}} = \boldsymbol{J}^{\dagger}(\boldsymbol{x})\ \dot{\boldsymbol{f}} + \boldsymbol{N}(\boldsymbol{x})\ \boldsymbol{g}(\boldsymbol{x}), \tag{10}$$

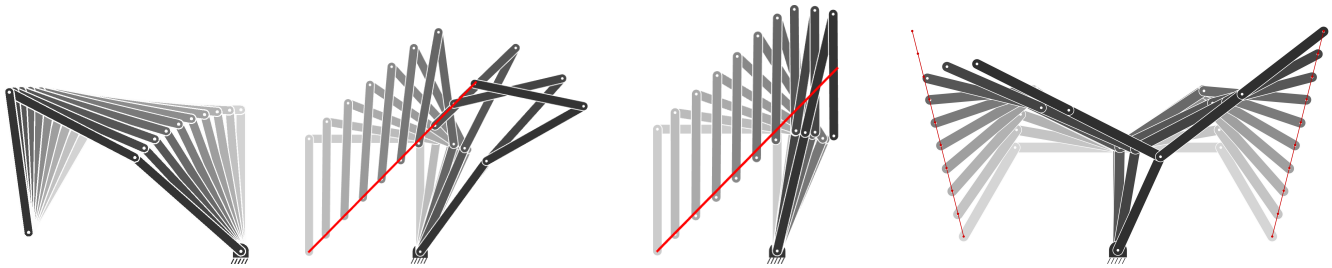
where $\boldsymbol{g}(\boldsymbol{x})$ is any desired gradient function and $\boldsymbol{N}(\boldsymbol{x})$ the **nullspace projection operator**

$$\boldsymbol{N}(\boldsymbol{x}) = \boldsymbol{I} - \boldsymbol{J}(\boldsymbol{x})^{\dagger}\boldsymbol{J}(\boldsymbol{x}).$$

In the above, the solution is unique if and only if $\boldsymbol{J}(\boldsymbol{x})$ has full column rank, in which case $\boldsymbol{N}(\boldsymbol{x})$ is a zero matrix.

An alternative way of computing the nullspace projection matrix, numerically more robust, is to exploit the singular value decomposition

$$\boldsymbol{J}^{\dagger} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top},$$



(a) Moving the first joint as secondary task

(b) Orientation tracking a as secondary task

(c) Position tracking as secondary task

(d) Tracking of a moving object with the right hand as secondary task

Figure 7: Examples of nullspace controllers. The red paths represent the trajectories of moving objects that need to be tracked by the robot end-effectors.

to compute

$$\boldsymbol{N} = \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top,$$

where $\tilde{\boldsymbol{U}}$ is a matrix formed by the columns of $\boldsymbol{U}$ that span for the corresponding zero rows in $\boldsymbol{\Sigma}$.

Figure 7 presents examples of nullspace controllers. Figure 7a shows a robot controller keeping its end-effector still as primary objective, while trying to move the first joint as secondary objective, which is achieved with the controller

$$\dot{\boldsymbol{x}} = \boldsymbol{J}^\dagger(\boldsymbol{x}) \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \boldsymbol{N}(\boldsymbol{x}) \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

5 Encoding with basis functions

MP.*

Basis functions can be used to encode signals in a compact manner. It can be used to provide prior information about the encoded signals through a weighted superposition of basis functions acting as a dictionary of simpler signals that are combined to form more complex signals.

Basis functions can for example be used to encode trajectories, whose input is a 1D time variable and whose output can be multidimensional. For basis functions $\boldsymbol{\phi}(t)$ described by a time or phase variable t, the corresponding continuous signal $\boldsymbol{x}(t)$ is encoded with $\boldsymbol{x}(t) = \boldsymbol{\phi}(t)\,\boldsymbol{w}$, with t a continuous time variable and $\boldsymbol{w}$ a vector containing the superposition weights. Basis functions can also be employed in a discretized form by providing a specific list of time steps, which is the form we will employ next.

The term *movement primitive* is often used in robotics to refer to the use of basis functions to encode trajectories. It corresponds to an organization of continuous motion signals in the form of a superposition in parallel and in series of simpler signals, which can be viewed as "building blocks" to create more complex movements, see Fig. 8. This principle, coined in the context of motor control [9], remains valid for a wide range of continuous time signals (for both analysis and synthesis).

5.1 Univariate trajectories

A univariate trajectory $\boldsymbol{x}^{1\text{D}} \in \mathbb{R}^T$ of T datapoints can be represented as a weighted sum of K basis functions with

$$\boldsymbol{x}^{1\text{D}} = \sum_{k=1}^{K} \boldsymbol{\phi}_k \, w_k^{1\text{D}} = \boldsymbol{\phi}\, \boldsymbol{w}^{1\text{D}}, \tag{11}$$

where $\boldsymbol{\phi}$ can be any set of basis functions, including some common forms that are presented below (see also [2] for more details).

Piecewise constant basis functions

Piecewise constant basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \boldsymbol{I}_K \otimes \mathbf{1}_{\frac{T}{K}}, \tag{12}$$

where $\boldsymbol{I}_K$ is an identity matrix of size K and $\mathbf{1}_{\frac{T}{K}}$ is a vector of length $\frac{T}{K}$ compose of unit elements, see Fig. 8.

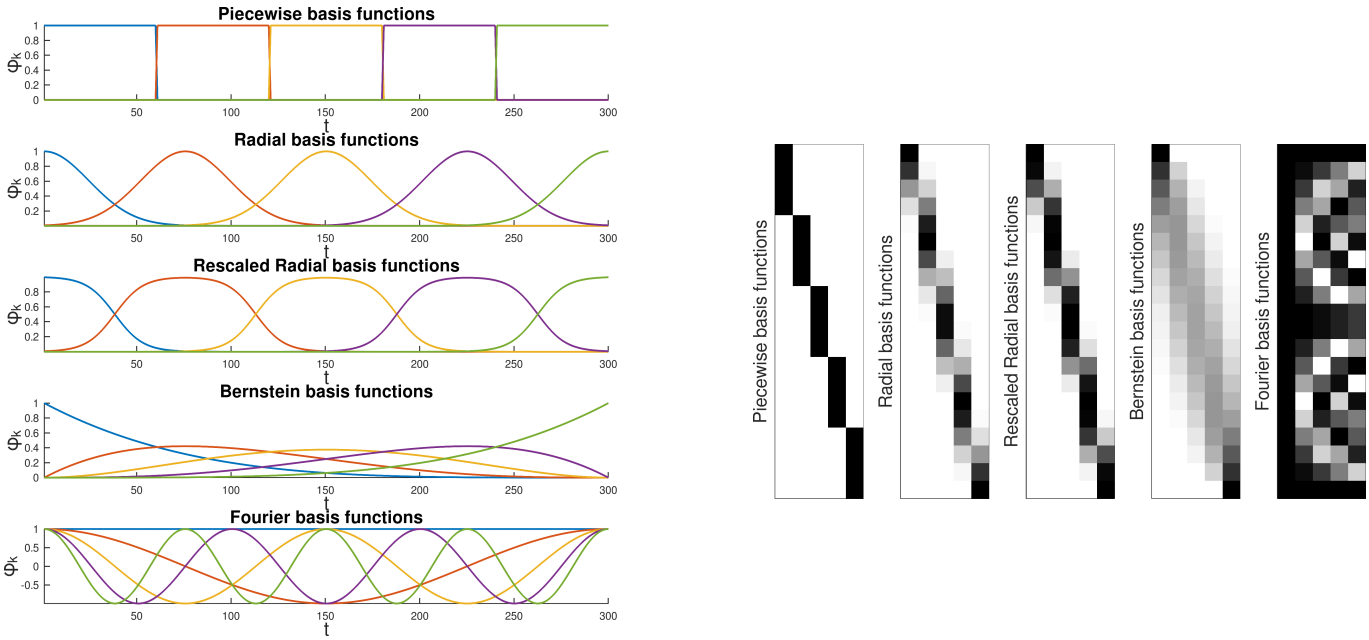


Figure 8: Examples of basis functions. *Left:* Representation in timeline form for $K = 5$ and $T = 300$. *Right:* Representation in matrix form for $K = 5$ and $T = 20$, with a grayscale colormap where white pixels are 0 and black pixels are 1.

Radial basis functions (RBFs)

Gaussian radial basis functions (RBFs) can be computed in matrix form as

$$\boldsymbol{\phi} = \exp(-\lambda\, \boldsymbol{E} \odot \boldsymbol{E}), \quad \text{with} \quad \boldsymbol{E} = \boldsymbol{t}\, \mathbf{1}_K^\top - \mathbf{1}_T\, \boldsymbol{\mu}^{s\top}, \tag{13}$$

where λ is bandwidth parameter, $\odot$ is the elementwise (Hadamard) product operator, $\boldsymbol{t} \in \mathbb{R}^T$ is a vector with entries linearly spaced between 0 to 1, $\boldsymbol{\mu}^s \in \mathbb{R}^K$ is a vector containing the RBF centers linearly spaced on the $[0, 1]$ range, and the $\exp(\cdot)$ function is applied to each element of the matrix, see Fig. 8.

RBFs can also be employed in a rescaled form by replacing each row $\boldsymbol{\phi}_k$ with $\frac{\boldsymbol{\phi}_k}{\sum_{i=1}^K \boldsymbol{\phi}_i}$.

Bernstein basis functions

Bernstein basis functions (used for Bézier curves) can be computed as

$$\boldsymbol{\phi}_k = \frac{(K-1)!}{(k-1)!(K-k)!}\, (\mathbf{1}_T - \boldsymbol{t})^{K-k} \;\odot\; \boldsymbol{t}^{k-1}, \tag{14}$$

$\forall k \in \{1, \dots, K\}$, where $\boldsymbol{t} \in \mathbb{R}^T$ is a vector with entries linearly spaced between 0 to 1, and $(\cdot)^d$ is applied elementwise, see Fig. 8.

Fourier basis functions

Fourier basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \exp(\boldsymbol{t}\, \tilde{\boldsymbol{k}}^\top\, 2\pi i), \tag{15}$$

where the $\exp(\cdot)$ function is applied to each element of the matrix, $\boldsymbol{t} \in \mathbb{R}^T$ is a vector with entries linearly spaced between 0 to 1, $\tilde{\boldsymbol{k}} = [-K+1, -K+2, \dots, K-2, K-1]^\top$, and i is the imaginary unit ($i^2 = -1$).

If $\boldsymbol{x}$ is a real and even signal, the above formulation can be simplified to

$$\boldsymbol{\phi} = \cos(\boldsymbol{t}\, \boldsymbol{k}^\top\, 2\pi i), \tag{16}$$

with $\boldsymbol{k} = [0, 1, \dots, K-2, K-1]^\top$, see Fig. 8.

Indeed, we first note that $\exp(ai)$ is composed of a real part and an imaginary part with $\exp(ai) = \cos(a) + i \sin(a)$.

We can see that for a given time step t, a real state x_t can be constructed with the Fourier series

$$\begin{aligned} x_t &= \sum_{k=-K+1}^{K-1} w_k \exp(t\, k\, 2\pi i) \\ &= \sum_{k=-K+1}^{K-1} w_k \cos(t\, k\, 2\pi) \\ &= w_0 + \sum_{k=1}^{K-1} 2w_k \cos(t\, k\, 2\pi), \end{aligned}$$

where we used the properties $\cos(0) = 1$ and $\cos(-a) = \cos(a)$ of the cosine function. Since we do not need direct correspondences between the Fourier transform and discrete cosine transform as in the above, we can omit the scaling factors for w_k and directly write the decomposition as in (16).

5.2 Multivariate trajectories

A multivariate trajectory $\boldsymbol{x} \in \mathbb{R}^{DT}$ of T datapoints of dimension D can similarly be computed as

$$\boldsymbol{x} = \sum_{k=1}^{K} \boldsymbol{\Psi}_k\, w_k = \boldsymbol{\Psi}\, \boldsymbol{w}, \quad \text{with} \quad \boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}\phi_{1,1} & \boldsymbol{C}\phi_{2,1} & \cdots & \boldsymbol{C}\phi_{K,1} \\ \boldsymbol{C}\phi_{1,2} & \boldsymbol{C}\phi_{2,2} & \cdots & \boldsymbol{C}\phi_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C}\phi_{1,T} & \boldsymbol{C}\phi_{2,T} & \cdots & \boldsymbol{C}\phi_{K,T} \end{bmatrix}, \tag{17}$$

where $\otimes$ the Kronecker product operator and $\boldsymbol{C}$ is a coordination matrix that can for example be set to identity.

Such encoding aims at encoding the movement as a weighted superposition of simpler movements, whose compression aims at working in a subspace of reduced dimensionality, while denoising the signal and capturing the essential aspects of a movement. This can be illustrated as basic building blocks that can be differently assembled to form more elaborated movements, often referred to as *movement primitives* (MP).

5.3 Multidimensional inputs

Basis functions can also be used to encode signals characterized by multivariate inputs. For example, Bézier surfaces use two input variables to cover a spatial range and an output variable describing the height for each point in this rectangular region. In this example, the surface can be constructed from 1D input signal by leveraging the Kronecker product operation, namely

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{\phi}. \tag{18}$$

We still have $\boldsymbol{x} = \boldsymbol{\Psi}\, \boldsymbol{w}$ as in the unidimensional version, where $\boldsymbol{x}$ and $\boldsymbol{w}$ are vectorized versions of surface heights and superposition weights, which can be reorganized as 2D arrays if required.

Successive Kronecker products can be used so that any number of input and output variables can be considered. For example, a vector field in 3D can be encoded with basis functions

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{\phi} \otimes \boldsymbol{\phi} \otimes \boldsymbol{I}, \tag{19}$$

where $\boldsymbol{I}$ is a 3×3 identity matrix to encode the 3 elements of the vector.

5.4 Concatenated basis functions

When encoding entire signals, some dictionaries such as Bernstein basis functions require to set polynomials of high order to encode long or complex signals. Instead of considering a global encoding, it can be useful to split the problem as a set of local fitting problems which can consider low order polynomials. A typical example is the encoding of complex curves as a concatenation of simple Bézier curves. When concatenating curves, constraints on the superposition weights are typically considered. These weights can be represented as control points in the case of Bézier curves and splines. We typically constraint the last point of a curve and the first point of the next curve to maintain the continuity of the curve. We also typically constraint the control points before and after this joint to by symmetric, effectively imposing smoothness.

In practice, this can be achieved efficiently by simply replacing $\boldsymbol{\phi}$ with $\boldsymbol{\phi S}$ in the above equations, where $\boldsymbol{S}$ is a tall rectangular matrix. This further reduces the number of superposition weights required in the encoding, as the constraints also reduce the number of free variables.

For example, for the concatenation of Bézier curves, we can define $\boldsymbol{S}$ as

$$\boldsymbol{S} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots \\ 0 & 1 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ 0 & 0 & \cdots & 1 & 0 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & -1 & 2 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{20}$$

where the pattern $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ -1 & 2 \end{bmatrix}$ is repeated for each junction of two consecutive Bézier curves. For two concatenated Bézier curves, each composed of 4 Bernstein basis functions, we can see locally that this operator yields a constraint of the form

$$\begin{bmatrix} w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}, \tag{21}$$

which ensures that $w_4 = w_5$ and $w_6 = -w_3 + 2w_5$. These constraints guarantee that the last control point and the first control point of the next segment are the same, and that the control point before and after are symmetric with respect to this junction point.

5.5 Batch computation of basis functions coefficients

Based on observed data $\boldsymbol{x}$, the superposition weights $\boldsymbol{w}$ can be estimated as a simple least squares estimate

$$\boldsymbol{w} = \boldsymbol{\Psi}^{\dagger}\boldsymbol{x}, \tag{22}$$

or as the regularized version (ridge regression)

$$\boldsymbol{w} = (\boldsymbol{\Psi}^{\top}\boldsymbol{\Psi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Psi}^{\top}\boldsymbol{x}. \tag{23}$$

5.6 Recursive computation of basis functions coefficients

The same result can be obtained by recursive computation, by providing the datapoints one-by-one or by groups of points. The algorithm starts from an iniital estimate of $\boldsymbol{w}$ that is iteratively refined with the arrival of new datapoints.

This recursive least squares algorithm exploits the **Sherman-Morrison-Woodbury formulas** that relate the inverse of a matrix after a small-rank perturbation to the inverse of the original matrix, namely

$$(\boldsymbol{B} + \boldsymbol{U}\boldsymbol{V})^{-1} = \boldsymbol{B}^{-1} - \overbrace{\boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{I} + \boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}}^{\boldsymbol{E}} \tag{24}$$

with $\boldsymbol{U} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{m \times n}$. When $m \ll n$, the correction term $\boldsymbol{E}$ can be computed more efficiently than inverting $\boldsymbol{B} + \boldsymbol{U}\boldsymbol{V}$.

By defining $\boldsymbol{B} = \boldsymbol{\Psi}^{\top}\boldsymbol{\Psi}$, the above relation can be exploited to update the least squares solution (23) when new datapoints become available. Indeed, if $\boldsymbol{\Psi}_{\text{new}} = \begin{bmatrix} \boldsymbol{\Psi}, \boldsymbol{V} \end{bmatrix}$ and $\boldsymbol{x}_{\text{new}} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{bmatrix}$, we can see that

$$\begin{aligned} \boldsymbol{B}_{\text{new}} &= \boldsymbol{\Psi}_{\text{new}}^{\top}\boldsymbol{\Psi}_{\text{new}} && (25) \\ &= \boldsymbol{\Psi}^{\top}\boldsymbol{\Psi} + \boldsymbol{V}^{\top}\boldsymbol{V} && (26) \\ &= \boldsymbol{B} + \boldsymbol{V}^{\top}\boldsymbol{V}, && (27) \end{aligned}$$

whose inverse can be computed using (24), yielding

$$\boldsymbol{B}_{\text{new}}^{-1} = \boldsymbol{B}^{-1} - \underbrace{\boldsymbol{B}^{-1}\boldsymbol{V}^{\top}\left(\boldsymbol{I} + \boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^{\top}\right)^{-1}}_{\boldsymbol{K}}\boldsymbol{V}\boldsymbol{B}^{-1}. \tag{28}$$

This is exploited to estimate the update as

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \boldsymbol{K}\Big(\boldsymbol{v} - \boldsymbol{V}\boldsymbol{w}\Big), \tag{29}$$

with Kalman gain

$$\boldsymbol{K} = \boldsymbol{B}^{-1}\boldsymbol{V}^\top \left(\boldsymbol{I} + \boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^\top\right)^{-1}. \tag{30}$$

The above equations can be used recursively, with $\boldsymbol{B}_{\text{new}}^{-1}$ and $\boldsymbol{w}_{\text{new}}$ the updates that will become $\boldsymbol{B}^{-1}$ and $\boldsymbol{w}$ for the next iteration. Note that the above iterative computation only uses $\boldsymbol{B}^{-1}$, which is the variable stored in the memory. Namely, we never use $\boldsymbol{B}$, only the inverse. For recursive ridge regression, the algorithm starts with $\boldsymbol{B}^{-1} = \frac{1}{\lambda}\boldsymbol{I}$. After all datapoints are used, the estimate of $\boldsymbol{w}$ is exactly the same as the ridge regression result (23) computed in batch form.

6 Linear quadratic tracking (LQT)

LQT.*

Linear quadratic tracking (LQT) is a simple form of optimal control that trades off tracking and control costs expressed as quadratic terms over a time horizon, with the evolution of the state described in a linear form. The LQT problem is formulated as the minimization of the cost

$$\begin{aligned} c &= (\boldsymbol{\mu}_T - \boldsymbol{x}_T)^\top \boldsymbol{Q}_T (\boldsymbol{\mu}_T - \boldsymbol{x}_T) + \sum_{t=1}^{T-1} \Big((\boldsymbol{\mu}_t - \boldsymbol{x}_t)^\top \boldsymbol{Q}_t (\boldsymbol{\mu}_t - \boldsymbol{x}_t) \;+\; \boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t \Big) \\ &= (\boldsymbol{\mu} - \boldsymbol{x})^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x}) \;+\; \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \end{aligned} \tag{31}$$

with $\boldsymbol{x} = \left[\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \dots, \boldsymbol{x}_T^\top\right]^\top$ the evolution of the state variables, $\boldsymbol{u} = \left[\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \dots, \boldsymbol{u}_{T-1}^\top\right]^\top$ the evolution of the control commands, and $\boldsymbol{\mu} = \left[\boldsymbol{\mu}_1^\top, \boldsymbol{\mu}_2^\top, \dots, \boldsymbol{\mu}_T^\top\right]^\top$ the evolution of the tracking targets. $\boldsymbol{Q} = \text{blockdiag}(\boldsymbol{Q}_1, \boldsymbol{Q}_2, \dots, \boldsymbol{Q}_T)$ represents the evolution of the precision matrices $\boldsymbol{Q}_t$, and $\boldsymbol{R} = \text{blockdiag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \dots, \boldsymbol{R}_{T-1})$ represents the evolution of the control weight matrices $\boldsymbol{R}_t$.

The evolution of the system is linear, described by

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t, \tag{32}$$

yielding at trajectory level

$$\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}, \tag{33}$$

see Appendix A for details.

With open loop control commands organized as a vector $\boldsymbol{u}$, the solution of (31) subject to $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}$ is analytic, given by

$$\hat{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1). \tag{34}$$

The residuals of this least squares solution provides information about the uncertainty of this estimate, in the form of a full covariance matrix (at control trajectory level)

$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1}, \tag{35}$$

which provides a probabilistic interpretation of LQT.

The batch computation approach facilitates the creation of bridges between learning and control. For example, in learning from demonstration, the observed (co)variations in a task can be formulated as an LQT objective function, which then provides a trajectory distribution in control space that can be converted to a trajectory distribution in state space. All the operations are analytic and only exploit basic linear algebra.

The approach can also be extended to model predictive control (MPC), iterative LQR (iLQR) and differential dynamic programming (DDP), whose solution needs this time to be interpreted locally at each iteration step of the algorithm, as we will see later in the technical report.

Example: Bimanual tennis serve

LQT_tennisServe.*

LQT can be used to solve a ballistic task mimicking a bimanual tennis serve problem. In this problem, a ball is thrown by one hand and then hit by the other, with the goal of bringing the ball to a desired target, see Fig. 9. The problem is formulated as in (31), namely

$$\min_{\boldsymbol{u}} (\boldsymbol{\mu} - \boldsymbol{x})^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x}) \;+\; \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \text{ s.t. } \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}, \tag{36}$$

where $\boldsymbol{x}$ represents the state trajectory of the 3 agents (left hand, right hand and ball), where only the hands can be controlled by a series of acceleration commands $\boldsymbol{u}$ that can be estimated by LQT.

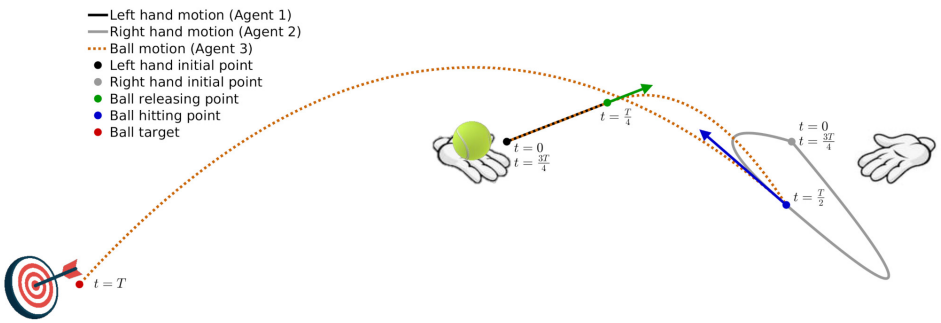


Figure 9: Tennis serve problem solved by linear quadratic tracking (LQT).

In the above problem, $\boldsymbol{Q}$ is a precision matrix and $\boldsymbol{\mu}$ is a reference vector describing at specific time steps the targets that the three agents must reach. The linear systems are described according to the different phases of the task, see Appendix C for details. As shown above, the constrained objective in (36) can be solved by least squares, providing an analytical solution given by (34), see also Fig. 9 for the visualization of the result for a given target and for given initial poses of the hands.

6.1 LQT with smoothness cost

`LQT.*`

Linear quadratic tracking (LQT) can be used with dynamical systems described as simple integrators. With single integrators, the states correspond to positions, with velocity control commands. With double integrators, the states correspond to positions and velocities, with acceleration control commands. With triple integrators, the states correspond to positions, velocities and accelerations, with jerk control commands, etc.

Figure 10 shows an example of LQT for a task that consist of passing through a set of viapoints. The left and right graphs show the result for single and double integrators, respectively. The graph in the center also considers a single integrator, by redefining the weight matrix $\boldsymbol{R}$ of the control cost to ensure smoothness. This can be done by replacing the standard diagonal $\boldsymbol{R}$ matrix in (31) with

$$\boldsymbol{R} = \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix} \otimes \boldsymbol{I}_D.$$

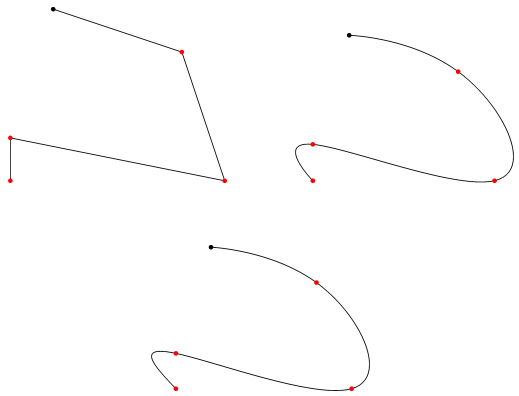


Figure 10: Examples of linear quadratic tracking (LQT) applied to the task of reaching a set of viapoints (red dots) by starting from an initial position (black dot). *Top-left:* With velocity commands, with a system described as a single integrator and a standard control cost. *Top-right:* With velocity commands, with a system described as a single integrator and a control cost ensuring smoothness. *Bottom:* With acceleration commands, with a system described as a double integrator and a standard control cost.

6.2 LQT with control primitives

LQT_CP.*

If we assume that the control commands profile $\boldsymbol{u}$ is composed of *control primitives* (CP) with $\boldsymbol{u} = \boldsymbol{\Psi w}$, the objective becomes

$$\min_{\boldsymbol{w}} (\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{Q}(\boldsymbol{x}-\boldsymbol{\mu}) \; + \; \boldsymbol{w}^\top \boldsymbol{\Psi}^\top \boldsymbol{R} \boldsymbol{\Psi} \boldsymbol{w}, \quad \text{s.t.} \quad \boldsymbol{x} = \boldsymbol{S_x x}_1 + \boldsymbol{S_u \Psi w},$$

with a solution given by

$$\hat{\boldsymbol{w}} = \big(\boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S_u \Psi} + \boldsymbol{\Psi}^\top \boldsymbol{R \Psi}\big)^{-1} \boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q}\big(\boldsymbol{\mu} - \boldsymbol{S_x x}_1\big),$$

which is used to compute the trajectory $\hat{\boldsymbol{u}} = \boldsymbol{\Psi}\hat{\boldsymbol{w}}$ in control space, corresponding to the list of control commands organized in vector form. Similarly to (35), the residuals of the least squares solution provides a probabilistic approach to LQT with control primitives. Note also that the trajectory in control space can be converted to a trajectory in state space with $\hat{\boldsymbol{x}} = \boldsymbol{S_x x}_1 + \boldsymbol{S_u}\hat{\boldsymbol{u}}$. Thus, since all operations are linear, a covariance matrix on $\boldsymbol{w}$ can be converted to a covariance on $\boldsymbol{u}$ and on $\boldsymbol{x}$. A similar linear transformation is used in the context of probabilistic movement primitives (ProMP) [12] to map a covariance matrix on $\boldsymbol{w}$ to a covariance on $\boldsymbol{x}$. LQT with control primitives provides a more general approach by considering both control and state spaces. It also has the advantage of reframing the method to the more general context of optimal control, thus providing various extension opportunities.

Several forms of basis functions can be considered, including stepwise control commands, radial basis functions (RBFs), Bernstein polynomials, Fourier series, or learned basis functions, which are organized as a dictionary matrix $\boldsymbol{\Psi}$. Dictionaries for multivariate control commands can be generated with a Kronecker product operator $\otimes$ as

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{C}, \tag{37}$$

where the matrix $\boldsymbol{\phi}$ is a horizontal concatenation of univariate basis functions, and $\boldsymbol{C}$ is a coordination matrix, which can be set to an identity matrix $\boldsymbol{I}_D$ for the generic case of control variables with independent basis functions for each dimension $d \in \{1, \dots, D\}$.

Note also that in some situations, $\boldsymbol{R}$ can be set to zero because the regularization role of $\boldsymbol{R}$ in the original problem formulation can sometimes be redundant with the use of sparse basis functions.

6.3 LQR with a recursive formulation

LQT_recursive.*

Algorithm 1: Recursive formulation of LQR

Define quadratic cost function with $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$
Define dynamics with $\boldsymbol{A}_t$, $\boldsymbol{B}_t$, and initial state $\boldsymbol{x}_1$
// Backward pass
Set $\boldsymbol{V}_T = \boldsymbol{Q}_T$
for $t \leftarrow T-1$ *to* 1 **do**
  Compute $\boldsymbol{V}_t$ with (46) and (44)
end
// Forward pass
for $t \leftarrow 1$ *to* $T-1$ **do**
  Compute $\boldsymbol{K}_t$ and $\hat{\boldsymbol{u}}_t$ with (45)
  Compute $\boldsymbol{x}_{t+1}$ with (32)
end

The LQR problem is formulated as the minimization of the cost

$$c(\boldsymbol{x}_1, \boldsymbol{u}) = c_T(\boldsymbol{x}_T) + \sum_{t=1}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \; = \boldsymbol{x}_T^\top \boldsymbol{Q}_T \boldsymbol{x}_T + \sum_{t=1}^{T-1} \Big(\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t\Big), \quad \text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t, \tag{38}$$

where c without index refers to the total cost (cumulative cost). We also define the partial cumulative cost

$$v_t(\boldsymbol{x}_t, \boldsymbol{u}_{t:T-1}) = c_T(\boldsymbol{x}_T) + \sum_{s=t}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{39}$$

which depends on the states and control commands, except for v_T that only depends on the state.

We define the *value function* as the value taken by v_t when applying optimal control commands, namely

$$\hat{v}_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_{t:T-1}} v_t(\boldsymbol{x}_t, \boldsymbol{u}_{t:T-1}). \tag{40}$$

We will use the dynamic programming principle to reduce the minimization in (40) over the entire sequence of control commands $\boldsymbol{u}_{t:T-1}$ to a sequence of minimization problems over control commands at a single time step, by proceeding backwards in time.

By inserting (39) in (40), we observe that

$$\begin{aligned}
\hat{v}_t(\boldsymbol{x}_t) &= \min_{\boldsymbol{u}_{t:T-1}} \Big(c_T(\boldsymbol{x}_T) + \sum_{s=t}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big) \\
&= \min_{\boldsymbol{u}_t} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \min_{\boldsymbol{u}_{t+1:T-1}} \Big(c_T(\boldsymbol{x}_T) + \sum_{s=t+1}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big) \\
&= \min_{\boldsymbol{u}_t} \underbrace{c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \hat{v}_{t+1}(\boldsymbol{x}_{t+1})}_{q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)},
\end{aligned} \tag{41}$$

where q_t is called the *q-function*.

By starting from the last time step T, and by relabeling $\boldsymbol{Q}_T$ as $\boldsymbol{V}_T$, we can see that $\hat{v}_T(\boldsymbol{x}_T) = c_T(\boldsymbol{x}_T)$ is independent of the control commands, taking the quadratic error form

$$\hat{v}_T = \boldsymbol{x}_T^\top \boldsymbol{V}_T \boldsymbol{x}_T, \tag{42}$$

which only involves the final state $\boldsymbol{x}_T$. We will show that $\hat{v}_{T-1}$ has the same quadratic form as $\hat{v}_T$, enabling the backward recursive computation of $\hat{v}_t$ from $t = T-1$ to $t = 1$.

With (41), the dynamic programming recursion takes the form

$$\hat{v}_t = \min_{\boldsymbol{u}_t} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R}_t \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \boldsymbol{x}_{t+1}^\top \boldsymbol{V}_{t+1} \boldsymbol{x}_{t+1}. \tag{43}$$

By substituting $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t$ into (43), $\hat{v}_t$ can be rewritten as

$$\hat{v}_t = \min_{\boldsymbol{u}_t} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{xx},t} & \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{Q}_{\boldsymbol{ux},t} & \boldsymbol{Q}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}, \quad \text{where} \quad \begin{cases} \boldsymbol{Q}_{\boldsymbol{xx},t} = \boldsymbol{A}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{A}_t + \boldsymbol{Q}_t, \\ \boldsymbol{Q}_{\boldsymbol{uu},t} = \boldsymbol{B}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{B}_t + \boldsymbol{R}_t, \\ \boldsymbol{Q}_{\boldsymbol{ux},t} = \boldsymbol{B}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{A}_t. \end{cases} \tag{44}$$

An optimal control command $\hat{\boldsymbol{u}}_t$ can be computed by differentiating (44) with respect to $\boldsymbol{u}_t$ and equating to zero, providing a feedback law

$$\hat{\boldsymbol{u}}_t = -\boldsymbol{K}_t \, \boldsymbol{x}_t, \quad \text{with} \quad \boldsymbol{K}_t = \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \boldsymbol{Q}_{\boldsymbol{ux},t}. \tag{45}$$

By introducing (45) back into (44), the resulting value function $\hat{v}_t$ has the quadratic form

$$\hat{v}_t = \boldsymbol{x}_t^\top \boldsymbol{V}_t \boldsymbol{x}_t, \quad \text{with} \quad \boldsymbol{V}_t = \boldsymbol{Q}_{\boldsymbol{xx},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \boldsymbol{Q}_{\boldsymbol{ux},t}. \tag{46}$$

We observe that (46) has the same quadratic form as (42), so that (41) can be solved recursively. We thus obtain a backward recursion procedure in which $\boldsymbol{V}_t$ is evaluated recursively from $t = T-1$ to $t = 1$, starting from $\boldsymbol{V}_T = \boldsymbol{Q}_T$, which corresponds to a Riccati equation.

After all feedback gain matrices $\boldsymbol{K}_t$ have been computed by backward recursion, a forward recursion can be used to compute the evolution of the state, starting from $\boldsymbol{x}_1$, by using the linear system $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t$ and the control policy $\boldsymbol{u}_t = -\boldsymbol{K}_t \boldsymbol{x}_t$, see Algorithm 1 for the summary of the overall procedure.

6.4 LQT with a recursive formulation and an augmented state space

`LQT_recursive.*`

In order to extend the above development to linear quadratic tracking (LQT), the problem of tracking a reference signal $\{\boldsymbol{\mu}_t\}_{t=1}^T$ can be recast as a regulation problem by considering a dynamical system with an augmented state

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{t+1} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_{t+1}} = \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{\tilde{\boldsymbol{A}}} \underbrace{\begin{bmatrix} \boldsymbol{x}_t \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_t} + \underbrace{\begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix}}_{\tilde{\boldsymbol{B}}} \boldsymbol{u}_t, \tag{47}$$

and an augmented tracking weight

$$\tilde{\boldsymbol{Q}}_t = \begin{bmatrix} \boldsymbol{Q}_t^{-1} + \boldsymbol{\mu}_t \boldsymbol{\mu}_t^\top & \boldsymbol{\mu}_t \\ \boldsymbol{\mu}_t^\top & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{\mu}_t^\top & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_t & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{\mu}_t \\ \boldsymbol{0} & 1 \end{bmatrix},$$

which is used to define the cost

$$\begin{aligned} c &= (\boldsymbol{\mu}_T - \boldsymbol{x}_T)^\top \boldsymbol{Q}_T (\boldsymbol{\mu}_T - \boldsymbol{x}_T) + \sum_{t=1}^{T-1} \Big((\boldsymbol{\mu}_t - \boldsymbol{x}_t)^\top \boldsymbol{Q}_t (\boldsymbol{\mu}_t - \boldsymbol{x}_t) \;+\; \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t \Big) \\ &= \tilde{\boldsymbol{x}}_T^\top \tilde{\boldsymbol{Q}}_T \tilde{\boldsymbol{x}}_T \;+\; \sum_{t=1}^{T-1} \Big(\tilde{\boldsymbol{x}}_t^\top \tilde{\boldsymbol{Q}}_t \tilde{\boldsymbol{x}}_t \;+\; \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t \Big), \end{aligned} \tag{48}$$

where the augmented form in (48) has the same form as the standard LQR cost in (38), allowing the tracking problem to be solved in the same way by using this augmented state representation. Additional verification details can be found in Appendix D.

For a tracking problem, we can see that the resulting optimal control policy takes the form

$$\hat{\boldsymbol{u}}_t = -\tilde{\boldsymbol{K}}_t \, \tilde{\boldsymbol{x}}_t = \boldsymbol{K}_t \, (\boldsymbol{\mu}_t - \boldsymbol{x}_t) + \boldsymbol{u}_t^{\text{ff}}, \tag{49}$$

characterized by a feedback gain matrix $\boldsymbol{K}_t$ extracted from $\tilde{\boldsymbol{K}}_t = \big[\boldsymbol{K}_t, \boldsymbol{k}_t\big]$, and a feedforward term $\boldsymbol{u}_t^{\text{ff}} = -\boldsymbol{k}_t - \boldsymbol{K}_t \boldsymbol{\mu}_t$ depending on $\boldsymbol{\mu}_t$.

6.5 Least squares formulation of recursive LQR

`LQT_recursive_LS.*`

We have seen in Section 6.3 that a standard LQR problem is formulated as

$$\min_{\boldsymbol{u}} \; \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \quad \text{s.t.} \quad \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u},$$

whose solution is

$$\hat{\boldsymbol{u}} = -\big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1,$$

corresponding to open loop control commands.

By introducing a matrix $\boldsymbol{F}$ to describe $\boldsymbol{u} = -\boldsymbol{F}\boldsymbol{x}_1$, we can alternatively define the optimization problem as

$$\min_{\boldsymbol{F}} \; \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} + (-\boldsymbol{F}\boldsymbol{x}_1)^\top \boldsymbol{R} (-\boldsymbol{F}\boldsymbol{x}_1), \quad \text{s.t.} \quad \boldsymbol{x} = (\boldsymbol{S}_{\boldsymbol{x}} - \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{F}) \boldsymbol{x}_1,$$

whose least squares solution is

$$\boldsymbol{F} = \big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{x}}. \tag{50}$$

We decompose $\boldsymbol{F}$ as block matrices $\boldsymbol{F}_t$ with $t \in \{1, \dots, T-1\}$. $\boldsymbol{F}$ can then be used to iteratively reconstruct regulation gains $\boldsymbol{K}_t$, by starting from $\boldsymbol{K}_1 = \boldsymbol{F}_1$, $\boldsymbol{P}_1 = \boldsymbol{I}$, and by computing recursively

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} (\boldsymbol{A}_{t-1} - \boldsymbol{B}_{t-1} \boldsymbol{K}_{t-1})^{-1}, \quad \boldsymbol{K}_t = \boldsymbol{F}_t \, \boldsymbol{P}_t, \tag{51}$$

which can then be used in a feedback controller as in (45).

It is straightforward to extend this least squares formulation of recursive LQR to linear quadratic tracking and use (49) as feedback controller on an augmented state space, since the recursive LQT problem can be transformed to a recursive LQR problem with an augmented state space representation (see Section 6.4).

This least squares formulation of LQT (LQT-LS) yields the same controller as with the standard recursive computation presented in Section 6.4. However, the linear form in (50) used by LQT-LS has several advantages. First, it allows the use of full precision matrices $\boldsymbol{Q}$, which will be demonstrated in the example below. It also allows to extend LQT-LS to the use of control primitives, which will be further discussed in Section 6.6. Moreover, it provides a nullspace structure that can be exploited in recursive LQR/LQT problems.

Example with the control of multiple agents

`LQT_recursive_LS_multiAgents.*`

Figure 11 presents an example with the control of multiple agents, with a precision matrix involving nonzero offdiagonal elements. The corresponding example code presents two options: it either requests the two agents to meet in the middle of the motion (e.g., for a handover task), or it requests the two agents to find a location to reach at different time steps (e.g., to drop and pick-up an object), involving nonzero offdiagonal elements at different time steps.

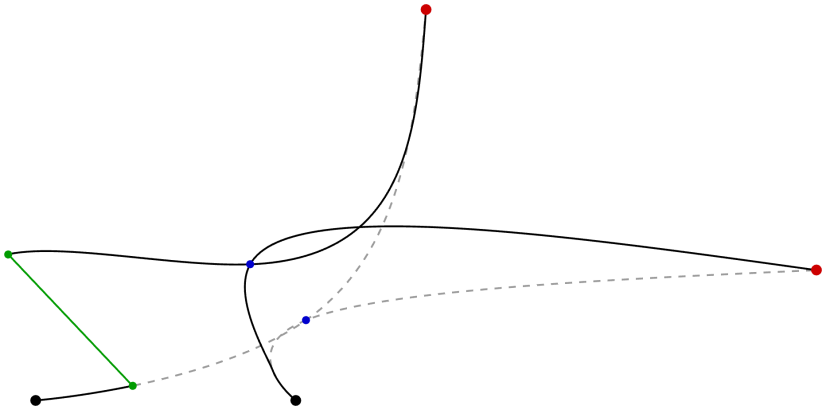


Figure 11: Least squares formulation of recursive LQR to control multiple agents (as point mass systems), where the task of each agent is to reach a desired target at the end of the motion, and to meet the other agent in the middle of the motion (e.g., for a handover task, see main text for the alternative option of nonzero offdiagonal elements at different time steps). We then test the adaptation capability of the agents by simulating a perturbation at 1/4 of the movement. The original and adapted movements are depicted in dashed and solid lines, respectively. The initial positions are represented with black points, the targets are in red, the optimized meeting points are in blue, and the perturbation is in green.

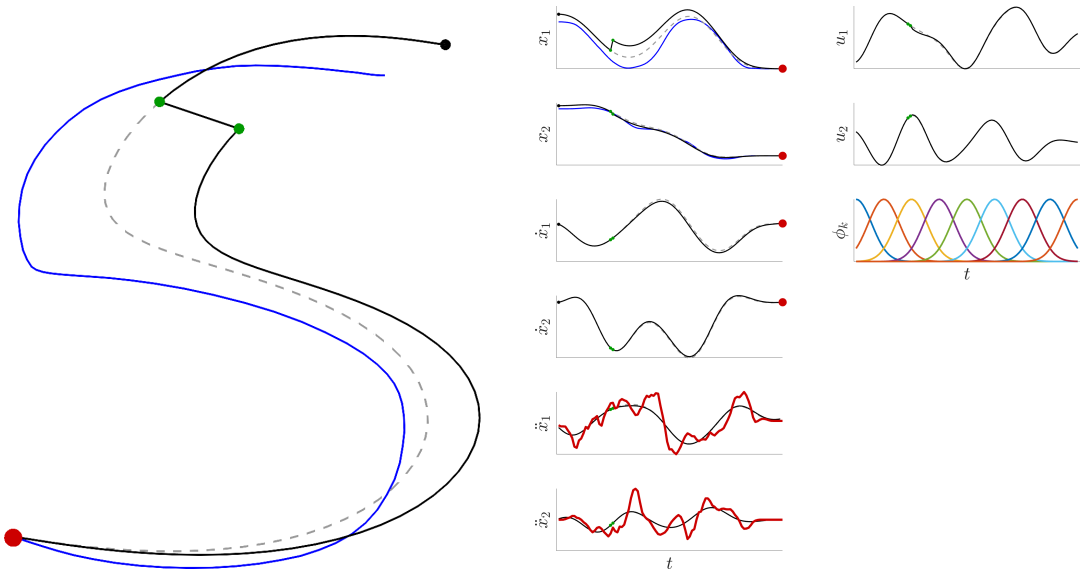


Figure 12: Linear quadratic tracking (LQT) with control primitives applied to a trajectory tracking task, with a formulation similar to dynamical movement primitives (DMP). *Left:* The observed "S" shape (in blue) is reproduced by starting from a different initial position (black point), with a perturbation simulated at 1/4 of the movement (green points) to show the capability of the approach to recover from perturbations. The trajectory in dashed line shows the result without perturbation and trajectory in solid line shows the result with perturbation. *Right:* The corresponding LQT problem formulation consists of requesting an end-point to be reached (red points) and an acceleration profile to be tracked (red lines), where the control commands $\boldsymbol{u}$ are represented as a superposition of radial basis functions $\boldsymbol{\phi}_k$.

6.6 Dynamical movement primitives (DMP) reformulated as LQT with control primitives

`LQT_CP_DMP.*`

The dynamical movement primitives (DMP) [4] approach proposes to reproduce an observed movement by crafting a controller composed of two parts: a closed-loop spring-damper system reaching the final point of the observed movement, and an open-loop system reproducing the acceleration profile of the observed movement. These two controllers are weighted so that the spring-damper system part is progressively increased until it becomes the only active controller. In DMP, the acceleration profile (also called forcing terms) is encoded with radial basis functions [1], and the spring-damper system parameters are defined heuristically, usually as a critically damped system.

Linear quadratic tracking (LQT) with control primitives can be used in a similar fashion as in DMP, by requesting a target to be reached at the end of the movement and by requesting the observed acceleration profile to be tracked, while encoding the control commands as radial basis functions. The controller can be estimated either as the open-loop control commands (34), or as the closed-loop controller (49).

In the latter case, the matrix $\boldsymbol{F}$ in (50) is estimated by using control primitives and an augmented state space formulation, namely

$$\hat{\boldsymbol{W}} = \left(\boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \tilde{\boldsymbol{Q}} \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{\Psi} + \boldsymbol{\Psi}^\top \boldsymbol{R} \boldsymbol{\Psi}\right)^{-1} \boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \tilde{\boldsymbol{Q}} \boldsymbol{S}_{\boldsymbol{x}}, \quad \boldsymbol{F} = \boldsymbol{\Psi} \hat{\boldsymbol{W}},$$

which is used to compute feedback gains $\tilde{\boldsymbol{K}}_t$ on the augmented state with (51).

The resulting controller $\hat{\boldsymbol{u}}_t = -\tilde{\boldsymbol{K}}_t \tilde{\boldsymbol{x}}_t$ tracks the acceleration profile while smoothly reaching the desired goal at the end of the movement, with a smooth transition between the two. The main difference with DMP is that the smooth transition between the two behaviors is directly optimized by the system, and the parameters of the feedback controller are automatically optimized (in DMP, stiffness and damping ratio).

Figure 12 presents an example of reproducing an "S" trajectory with simulation perturbations.

In addition, the LQT formulation allows the resulting controller to be formalized in the form of a cost function, which allows the approach to be combined more fluently with other optimal control strategies. Notably, the approach can be extended to multiple viapoints without any modification. It allows multiple demonstrations of a movement to be used to estimate a feedback controller that will exploit the (co)variations in the demonstrations to provide a minimal intervention control strategy that will selectively reject perturbations based on the impact they can have on the task to achieve. This is effectively attained by automatically regulating the gains in accordance to the variations in the demonstrations, with low gains in parts of the movement allowing variations, and higher gains for parts of the movement that are invariant in the demonstrations. It is also important to highlight that the solution of the LQT problem formulated as in the above is analytical, corresponding to a simple least squares problem.

Moreover, the above problem formulation can be extended to iterative LQR, providing an opportunity to consider obstacle avoidance and constraints within the DMP formulation, as well as to describe costs in task space with a DMP acting in joint angle space.

7 iLQR optimization

Optimal control problems are defined by a cost function $\sum_{t=1}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ to minimize and a dynamical system $\boldsymbol{x}_{t+1} = \boldsymbol{d}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ describing the evolution of a state $\boldsymbol{x}_t$ driven by control commands $\boldsymbol{u}_t$ during a time window of length T.

Iterative LQR (iLQR) [6] solves such constrained nonlinear models by carrying out Taylor expansions on the cost and on the dynamical system so that a solution can be found iteratively by solving a LQR problem at each iteration, similarly to differential dynamic programming approaches [8, 14].

iLQR employs a first order Taylor expansion of the dynamical system $\boldsymbol{x}_{t+1} = \boldsymbol{d}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ around the point $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, namely

$$\begin{aligned} & \boldsymbol{x}_{t+1} \approx \boldsymbol{d}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}_t}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{u}_t}(\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t) \\ \iff & \Delta \boldsymbol{x}_{t+1} \approx \boldsymbol{A}_t \Delta \boldsymbol{x}_t + \boldsymbol{B}_t \Delta \boldsymbol{u}_t, \end{aligned} \tag{52}$$

with residual vectors

$$\Delta \boldsymbol{x}_t = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t, \quad \Delta \boldsymbol{u}_t = \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t,$$

and Jacobian matrices

$$\boldsymbol{A}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \quad \boldsymbol{B}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{u}_t}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}.$$

The cost function $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ for time step t can similarly be approximated by a second order Taylor expansion around the point $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, namely

$$\begin{aligned} & c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \Delta \boldsymbol{x}_t^\top \frac{\partial c}{\partial \boldsymbol{x}_t} + \Delta \boldsymbol{u}_t^\top \frac{\partial c}{\partial \boldsymbol{u}_t} + \frac{1}{2} \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t^2} \Delta \boldsymbol{x}_t + \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t \boldsymbol{u}_t} \Delta \boldsymbol{u}_t + \frac{1}{2} \Delta \boldsymbol{u}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2} \Delta \boldsymbol{u}_t, \\ \iff & c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{1}{2} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x}_t \\ \Delta \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x},t}^\top & \boldsymbol{g}_{\boldsymbol{u},t}^\top \\ \boldsymbol{g}_{\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t}^\top \\ \boldsymbol{g}_{\boldsymbol{u},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x}_t \\ \Delta \boldsymbol{u}_t \end{bmatrix}, \end{aligned} \tag{53}$$

with gradients

$$\boldsymbol{g}_{\boldsymbol{x},t} = \frac{\partial c}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \quad \boldsymbol{g}_{\boldsymbol{u},t} = \frac{\partial c}{\partial \boldsymbol{u}_t}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t},$$

and Hessian matrices

$$\boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x},t} = \frac{\partial^2 c}{\partial \boldsymbol{x}_t^2}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \quad \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u},t} = \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}, \quad \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t} = \frac{\partial^2 c}{\partial \boldsymbol{u}_t \partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t}.$$

7.1 Batch formulation of iLQR

`iLQR_manipulator.*`

A solution in batch form can be computed by minimizing over $\boldsymbol{u} = \left[\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \ldots, \boldsymbol{u}_{T-1}^\top\right]^\top$, yielding a series of open loop control commands $\boldsymbol{u}_t$, corresponding to a Gauss–Newton iteration scheme, see Section 2.1.

At a trajectory level, we denote $\boldsymbol{x} = \left[\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \ldots, \boldsymbol{x}_T^\top\right]^\top$ the evolution of the state and $\boldsymbol{u} = \left[\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \ldots, \boldsymbol{u}_{T-1}^\top\right]^\top$ the evolution of the control commands. The evolution of the state in (52) becomes $\Delta \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u}$, see Appendix A for details.[1]

The minimization problem can then be rewritten in batch form as

$$\min_{\Delta \boldsymbol{u}} \Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}), \quad \text{s.t.} \quad \Delta \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u}, \quad \text{where} \quad \Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}) = \frac{1}{2} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x} \\ \Delta \boldsymbol{u} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x}}^\top & \boldsymbol{g}_{\boldsymbol{u}}^\top \\ \boldsymbol{g}_{\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \\ \boldsymbol{g}_{\boldsymbol{u}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x} \\ \Delta \boldsymbol{u} \end{bmatrix}. \tag{54}$$

By inserting the constraint into the cost, we obtain the optimization problem

$$\min_{\Delta \boldsymbol{u}} \quad \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} + \Delta \boldsymbol{u}^\top \boldsymbol{g}_{\boldsymbol{u}} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u} + \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \Delta \boldsymbol{u} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \Delta \boldsymbol{u}, \tag{55}$$

which can be solved analytically by differentiating with respect to $\Delta \boldsymbol{u}$ and equating to zero, namely,

$$\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{u}} + \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u} + 2 \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \Delta \boldsymbol{u} + \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \Delta \boldsymbol{u} = 0, \tag{56}$$

providing the least squares solution

$$\Delta \hat{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} + 2 \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top + \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}}\right)^{-1} \left(-\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} - \boldsymbol{g}_{\boldsymbol{u}}\right), \tag{57}$$

which can be used to update the control commands estimate at each iteration step of the iLQR algorithm.

The complete iLQR procedure is described in Algorithm 3 (including backtracking line search). Figure 13 also presents an illustrative summary of the iLQR optimization procedure in batch form.

[1] Note that $\boldsymbol{S}_{\boldsymbol{x}} \Delta \boldsymbol{x}_1 = \boldsymbol{0}$ because $\Delta \boldsymbol{x}_1 = \boldsymbol{0}$ (as we want our motion to start from $\boldsymbol{x}_1$).

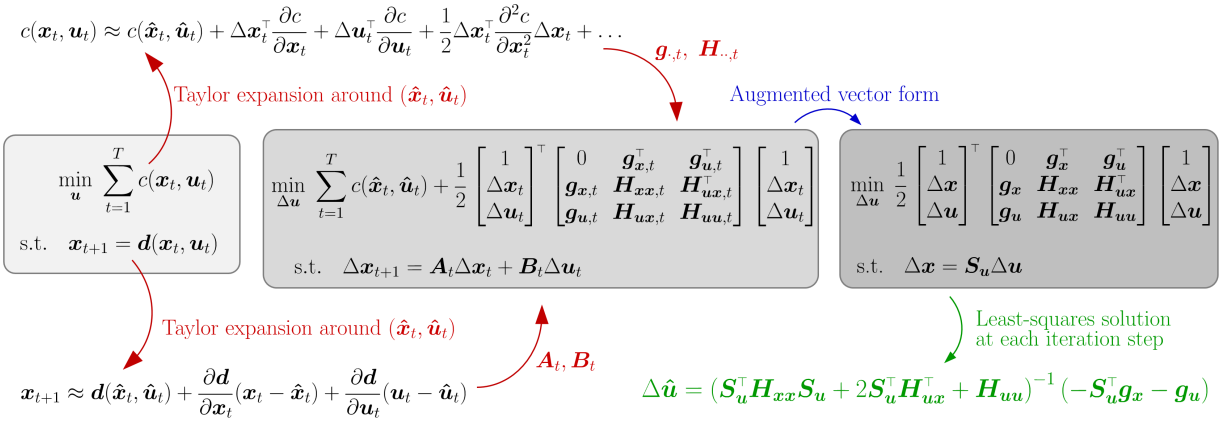


Figure 13: Summary of the iLQR optimization procedure in batch form.

7.2 Recursive formulation of iLQR

`iLQR_manipulator_recursive.*`

A solution can alternatively be computed in a recursive form to provide a controller with feedback gains. Section 6.3 presented the dynamic programming principle in the context of linear quadratic regulation problems, which allowed us to reduce the minimization over an entire sequence of control commands to a sequence of minimization problems over control commands at a single time step, by proceeding backwards in time. In this section, the approach is extended to iLQR.

Similarly to (41), we have here

$$\hat{v}_t(\Delta\boldsymbol{x}_t) = \min_{\Delta\boldsymbol{u}_t} \underbrace{c_t(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t) + \hat{v}_{t+1}(\Delta\boldsymbol{x}_{t+1})}_{q_t(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t)}, \tag{58}$$

Similarly to LQR, by starting from the last time step T, $\hat{v}_T(\Delta\boldsymbol{x}_T)$ is independent of the control commands. By relabeling $\boldsymbol{g}_{\boldsymbol{x},T}$ and $\boldsymbol{H}_{\boldsymbol{xx},T}$ as $\boldsymbol{v}_{\boldsymbol{x},T}$ and $\boldsymbol{V}_{\boldsymbol{xx},T}$, we can show that $\hat{v}_{T-1}$ has the same quadratic form as $\hat{v}_T$, enabling the backward recursive computation of $\hat{v}_t$ from $t = T-1$ to $t = 1$.

This dynamic programming recursion takes the form

$$\hat{v}_{t+1} = \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{v}_{\boldsymbol{x},t+1}^\top \\ \boldsymbol{v}_{\boldsymbol{x},t+1} & \boldsymbol{V}_{\boldsymbol{xx},t+1} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}, \tag{59}$$

and we then have with (58) that

$$\hat{v}_t = \min_{\Delta\boldsymbol{u}_t} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x},t}^\top & \boldsymbol{g}_{\boldsymbol{u},t}^\top \\ \boldsymbol{g}_{\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{xx},t} & \boldsymbol{H}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{g}_{\boldsymbol{u},t} & \boldsymbol{H}_{\boldsymbol{ux},t} & \boldsymbol{H}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix} + \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{v}_{\boldsymbol{x},t+1}^\top \\ \boldsymbol{v}_{\boldsymbol{x},t+1} & \boldsymbol{V}_{\boldsymbol{xx},t+1} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}. \tag{60}$$

By substituting $\Delta\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\Delta\boldsymbol{x}_t + \boldsymbol{B}_t\Delta\boldsymbol{u}_t$ into (60), $\hat{v}_t$ can be rewritten as

$$\hat{v}_t = \min_{\Delta\boldsymbol{u}_t} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{q}_{\boldsymbol{x},t}^\top & \boldsymbol{q}_{\boldsymbol{u},t}^\top \\ \boldsymbol{q}_{\boldsymbol{x},t} & \boldsymbol{Q}_{\boldsymbol{xx},t} & \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{q}_{\boldsymbol{u},t} & \boldsymbol{Q}_{\boldsymbol{ux},t} & \boldsymbol{Q}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}, \quad \text{where} \quad \begin{cases} \boldsymbol{q}_{\boldsymbol{x},t} = \boldsymbol{g}_{\boldsymbol{x},t} + \boldsymbol{A}_t^\top \boldsymbol{v}_{\boldsymbol{x},t+1}, \\ \boldsymbol{q}_{\boldsymbol{u},t} = \boldsymbol{g}_{\boldsymbol{u},t} + \boldsymbol{B}_t^\top \boldsymbol{v}_{\boldsymbol{x},t+1}, \\ \boldsymbol{Q}_{\boldsymbol{xx},t} \approx \boldsymbol{H}_{\boldsymbol{xx},t} + \boldsymbol{A}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{A}_t, \\ \boldsymbol{Q}_{\boldsymbol{uu},t} \approx \boldsymbol{H}_{\boldsymbol{uu},t} + \boldsymbol{B}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{B}_t, \\ \boldsymbol{Q}_{\boldsymbol{ux},t} \approx \boldsymbol{H}_{\boldsymbol{ux},t} + \boldsymbol{B}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{A}_t, \end{cases} \tag{61}$$

with gradients

$$\boldsymbol{g}_{\boldsymbol{x},t} = \frac{\partial c}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{g}_{\boldsymbol{u},t} = \frac{\partial c}{\partial \boldsymbol{u}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{v}_{\boldsymbol{x},t} = \frac{\partial \hat{v}}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{q}_{\boldsymbol{x},t} = \frac{\partial q}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{q}_{\boldsymbol{u},t} = \frac{\partial q}{\partial \boldsymbol{u}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t},$$

Jacobian matrices

$$\boldsymbol{A}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{B}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{u}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t},$$

and Hessian matrices

$$\boldsymbol{H}_{\boldsymbol{xx},t} = \frac{\partial^2 c}{\partial \boldsymbol{x}_t^2}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{H}_{\boldsymbol{uu},t} = \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{H}_{\boldsymbol{ux},t} = \frac{\partial^2 c}{\partial \boldsymbol{u}_t \partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{V}_{\boldsymbol{xx},t} = \frac{\partial^2 \hat{v}}{\partial \boldsymbol{x}_t^2}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t},$$

$$\boldsymbol{Q}_{\boldsymbol{xx},t} = \frac{\partial^2 q}{\partial \boldsymbol{x}_t^2}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{Q}_{\boldsymbol{uu},t} = \frac{\partial^2 q}{\partial \boldsymbol{u}_t^2}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}, \quad \boldsymbol{Q}_{\boldsymbol{ux},t} = \frac{\partial^2 q}{\partial \boldsymbol{u}_t \partial \boldsymbol{x}_t}\Big|_{\hat{\boldsymbol{x}}_t,\hat{\boldsymbol{u}}_t}.$$

Minimizing (61) w.r.t. $\Delta\boldsymbol{u}_t$ can be achieved by differentiating the equation and equating to zero, yielding the controller

$$\Delta\hat{\boldsymbol{u}}_t = \boldsymbol{k}_t + \boldsymbol{K}_t\,\Delta\boldsymbol{x}_t, \text{ with } \begin{cases} \boldsymbol{k}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1}\,\boldsymbol{q}_{\boldsymbol{u},t}, \\ \boldsymbol{K}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1}\,\boldsymbol{Q}_{\boldsymbol{ux},t}, \end{cases} \tag{62}$$

where $\boldsymbol{k}_t$ is a feedforward command and $\boldsymbol{K}_t$ is a feedback gain matrix.

By inserting (62) into (61), we get the recursive updates

$$\begin{aligned} \boldsymbol{v}_{\boldsymbol{x},t} &= \boldsymbol{q}_{\boldsymbol{x},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^{\top}\,\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1}\,\boldsymbol{q}_{\boldsymbol{u},t}, \\ \boldsymbol{V}_{\boldsymbol{xx},t} &= \boldsymbol{Q}_{\boldsymbol{xx},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^{\top}\,\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1}\,\boldsymbol{Q}_{\boldsymbol{ux},t}. \end{aligned} \tag{63}$$

In this recursive iLQR formulation, at each iteration step of the optimization algorithm, the nominal trajectories $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$ are refined, together with feedback matrices $\boldsymbol{K}_t$. Thus, at each iteration step of the optimization algorithm, a backward and forward recursion is performed to evaluate these vectors and matrices. There is thus two types of iterations: one for the optimization algorithm, and one for the dynamic programming recursion performed at each given iteration.

After convergence, by using (62) and the nominal trajectories in the state and control spaces $(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$, the resulting controller at each time step t is

$$\boldsymbol{u}_t = \hat{\boldsymbol{u}}_t + \boldsymbol{K}_t\,(\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t), \tag{64}$$

where the evolution of the state is described by $\boldsymbol{x}_{t+1} = \boldsymbol{d}(\boldsymbol{x}_t, \boldsymbol{u}_t)$.

The complete iLQR procedure is described in Algorithm 4 (including backtracking line search).

7.3 Least squares formulation of recursive iLQR

First, note that (62) can be rewritten as

$$\Delta\hat{\boldsymbol{u}}_t = \tilde{\boldsymbol{K}}_t \begin{bmatrix} \Delta\boldsymbol{x}_t \\ 1 \end{bmatrix}, \text{ with } \quad \tilde{\boldsymbol{K}}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1}\Big[\boldsymbol{Q}_{\boldsymbol{ux},t},\ \boldsymbol{q}_{\boldsymbol{u},t}\Big]. \tag{65}$$

Also, (55) can be rewritten as

$$\min_{\Delta\boldsymbol{u}} \underbrace{\frac{1}{2}\begin{bmatrix} \Delta\boldsymbol{u} \\ 1 \end{bmatrix}^{\top}\begin{bmatrix} \boldsymbol{C} & \boldsymbol{c} \\ \boldsymbol{c}^{\top} & 0 \end{bmatrix}\begin{bmatrix} \Delta\boldsymbol{u} \\ 1 \end{bmatrix}}_{\frac{1}{2}\Delta\boldsymbol{u}^{\top}\boldsymbol{C}\Delta\boldsymbol{u} + \Delta\boldsymbol{u}^{\top}\boldsymbol{c}}, \quad \text{where} \quad \begin{cases} \boldsymbol{c} = \boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{u}}, \\ \boldsymbol{C} = \boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{xx}}\boldsymbol{S}_{\boldsymbol{u}} + 2\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{ux}}^{\top} + \boldsymbol{H}_{\boldsymbol{uu}}. \end{cases}$$

Similarly to Section 6.5, we set

$$\Delta\boldsymbol{u} = -\boldsymbol{F}\begin{bmatrix} \Delta\boldsymbol{x}_1 \\ 1 \end{bmatrix} = -\boldsymbol{F}\begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} = -\boldsymbol{F}\Delta\tilde{\boldsymbol{x}}_1, \tag{66}$$

and redefine the optimization problem as

$$\min_{\boldsymbol{F}} \quad \frac{1}{2}\Delta\tilde{\boldsymbol{x}}_1^{\top}\boldsymbol{F}^{\top}\boldsymbol{C}\boldsymbol{F}\Delta\tilde{\boldsymbol{x}}_1 - \Delta\tilde{\boldsymbol{x}}_1^{\top}\boldsymbol{F}^{\top}\boldsymbol{c}. \tag{67}$$

By differentiating w.r.t. $\boldsymbol{F}$ and equating to zero, we get

$$\boldsymbol{F}\Delta\tilde{\boldsymbol{x}}_1 = \boldsymbol{C}^{-1}\boldsymbol{c}. \tag{68}$$

Similarly to Section 6.5, we decompose $\boldsymbol{F}$ as block matrices $\boldsymbol{F}_t$ with $t \in \{1, \dots, T-1\}$. $\boldsymbol{F}$ can then be used to iteratively reconstruct regulation gains $\boldsymbol{K}_t$, by starting from $\boldsymbol{K}_1 = \boldsymbol{F}_1$, $\boldsymbol{P}_1 = \boldsymbol{I}$, and by computing with forward recursion

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1}(\boldsymbol{A}_{t-1} - \boldsymbol{B}_{t-1}\boldsymbol{K}_{t-1})^{-1}, \quad \boldsymbol{K}_t = \boldsymbol{F}_t\,\boldsymbol{P}_t, \tag{69}$$

from $t = 2$ to $t = T - 1$.

Algorithm 2: Backtracking line search method with parameter $\alpha_{\min}$

$\alpha \leftarrow 1$
while $c(\hat{\boldsymbol{u}} + \alpha\, \Delta\hat{\boldsymbol{u}}) > c(\hat{\boldsymbol{u}})$ ***and*** $\alpha > \alpha_{\min}$ **do**
  $\alpha \leftarrow \frac{\alpha}{2}$
end

Algorithm 3: Batch formulation of iLQR

Define cost $c(\cdot)$, dynamics $\boldsymbol{d}(\cdot)$, $\boldsymbol{x}_1$, $\alpha_{\min}$, $\Delta_{\min}$
Initialize $\hat{\boldsymbol{u}}$
repeat
  Compute $\hat{\boldsymbol{x}}$ using $\hat{\boldsymbol{u}}$, $\boldsymbol{d}(\cdot)$, and $\boldsymbol{x}_1$
  Compute $\boldsymbol{A}_t, \boldsymbol{B}_t$ in (52)
  Compute $\boldsymbol{S}_{\boldsymbol{u}}$ with (92)
  Compute $\boldsymbol{g}_{\cdot}$ and $\boldsymbol{H}_{\cdot}$ in (54), using (53)
  Compute $\Delta\hat{\boldsymbol{u}}$ with (57)
  Compute α with Algorithm 2
  Update $\hat{\boldsymbol{u}}$ with (70)
until $\|\Delta\hat{\boldsymbol{u}}\| < \Delta_{\min}$;

Algorithm 4: Recursive formulation of iLQR

Define cost $c(\cdot)$, dynamics $\boldsymbol{d}(\cdot)$, $\boldsymbol{x}_1$, $\alpha_{\min}$, $\Delta_{\min}$
Initialize all $\hat{\boldsymbol{u}}_t$
repeat
  Compute $\hat{\boldsymbol{x}}$ using $\hat{\boldsymbol{u}}$, $\boldsymbol{d}(\cdot)$, and $\boldsymbol{x}_1$
  `// Evaluating derivatives`
  Compute all $\boldsymbol{A}_t, \boldsymbol{B}_t$ in (52)
  Compute all $\boldsymbol{g}_{\cdot,t}$ and $\boldsymbol{H}_{\cdot,t}$ in (53)
  `// Backward pass`
  Set $\boldsymbol{v}_{\boldsymbol{x},T} = \boldsymbol{g}_{\boldsymbol{x},T}$ and $\boldsymbol{V}_{\boldsymbol{xx},T} = \boldsymbol{H}_{\boldsymbol{xx},T}$
  for $t \leftarrow T-1$ *to* 1 **do**
    Compute $\boldsymbol{q}_{\cdot,t}$ and $\boldsymbol{Q}_{\cdot,t}$ with (61)
    Compute $\boldsymbol{k}_t$ and $\boldsymbol{K}_t$ with (62)
    Compute $\boldsymbol{v}_{\boldsymbol{x},t}$ and $\boldsymbol{V}_{\boldsymbol{xx},t}$ with (63)
  end
  `// Forward pass`
  $\alpha \leftarrow 2$
  while $c(\hat{\boldsymbol{u}} + \alpha\, \Delta\hat{\boldsymbol{u}}) > c(\hat{\boldsymbol{u}})$ ***and*** $\alpha > \alpha_{\min}$ **do**
    $\alpha \leftarrow \frac{\alpha}{2}$
    for $t \leftarrow 1$ *to* $T-1$ **do**
      Update $\hat{\boldsymbol{u}}_t$ with (71)
      Compute $\hat{\boldsymbol{x}}_{t+1}$ with $\boldsymbol{d}(\cdot)$
    end
  end
until $\|\Delta\hat{\boldsymbol{u}}\| < \Delta_{\min}$;
Use (64) and $\boldsymbol{d}(\cdot)$ for reproduction

7.4 Updates by considering step sizes

iLQR_manipulator.*

To be more efficient, iLQR most often requires at each iteration to estimate a step size α to scale the control command updates. For the batch formulation in Section 7.1, this can be achieved by setting the update (57) as

$$\hat{\boldsymbol{u}} \leftarrow \hat{\boldsymbol{u}} + \alpha\, \Delta\hat{\boldsymbol{u}}, \tag{70}$$

where the resulting procedure consists of estimating a descent direction with (57) along which the objective function will be reduced, and then estimating with (70) a step size that determines how far one can move along this direction.

For the recursive formulation in Section 7.2, this can be achieved by setting the update (62) as

$$\hat{\boldsymbol{u}}_t \leftarrow \hat{\boldsymbol{u}}_t + \alpha\, \boldsymbol{k}_t + \boldsymbol{K}_t\, \Delta\boldsymbol{x}_t. \tag{71}$$

In practice, a simple backtracking line search procedure can be considered with Algorithm 2, by considering a small value for $\alpha_{\min}$. For more elaborated methods, see Ch. 3 of [10].

The complete iLQR procedures are described in Algorithms 3 and 4 for the batch and recursive formulations, respectively.

7.5 iLQR with quadratic cost on $f(x_t)$

iLQR_manipulator.*

We consider a cost defined by

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{f}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t) + \boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t, \tag{72}$$

where $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ are weight matrices trading off task and control costs. Such cost is quadratic on $\boldsymbol{f}(\boldsymbol{x}_t)$ but non-quadratic on $\boldsymbol{x}_t$.

For the batch formulation of iLQR, the cost in (53) then becomes

$$\Delta c(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t) \approx 2\Delta\boldsymbol{x}_t^\top \boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t) + 2\Delta\boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t + \Delta\boldsymbol{x}_t^\top \boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{J}(\boldsymbol{x}_t)\Delta\boldsymbol{x}_t + \Delta\boldsymbol{u}_t^\top \boldsymbol{R}_t \Delta\boldsymbol{u}_t, \tag{73}$$

which used gradients

$$\boldsymbol{g}_{\boldsymbol{x},t} = 2\boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t), \quad \boldsymbol{g}_{\boldsymbol{u},t} = 2\boldsymbol{R}_t \boldsymbol{u}_t, \tag{74}$$

and Hessian matrices

$$\boldsymbol{H}_{\boldsymbol{xx},t} \approx 2\boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{J}(\boldsymbol{x}_t), \quad \boldsymbol{H}_{\boldsymbol{ux},t} = \boldsymbol{0}, \quad \boldsymbol{H}_{\boldsymbol{uu},t} = 2\boldsymbol{R}_t. \tag{75}$$

with $\boldsymbol{J}(\boldsymbol{x}_t) = \frac{\partial \boldsymbol{f}(\boldsymbol{x}_t)}{\partial \boldsymbol{x}_t}$ a Jacobian matrix. The same results can be used in the recursive formulation in (61).

At a trajectory level, the evolution of the tracking and control weights is represented by $\boldsymbol{Q} = \text{blockdiag}(\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R} = \text{blockdiag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{T-1})$, respectively.

With a slight abuse of notation, we define $\boldsymbol{f}(\boldsymbol{x})$ as a vector concatenating the vectors $\boldsymbol{f}(\boldsymbol{x}_t)$, and $\boldsymbol{J}(\boldsymbol{x})$ as a block-diagonal concatenation of the Jacobian matrices $\boldsymbol{J}(\boldsymbol{x}_t)$. The minimization problem (55) then becomes

$$\min_{\Delta\boldsymbol{u}} \quad 2\Delta\boldsymbol{x}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) + 2\Delta\boldsymbol{u}^\top \boldsymbol{R}\, \boldsymbol{u} \; + \Delta\boldsymbol{x}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \Delta\boldsymbol{u}^\top \boldsymbol{R}\Delta\boldsymbol{u}, \quad \text{s.t.} \quad \Delta\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}}\Delta\boldsymbol{u}, \tag{76}$$

whose least squares solution is given by

$$\Delta\hat{\boldsymbol{u}} = \Big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x}) \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\Big)^{-1} \Big(- \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{R}\, \boldsymbol{u}\Big), \tag{77}$$

which can be used to update the control commands estimates at each iteration step of the iLQR algorithm.

In the next sections, we show examples of functions $\boldsymbol{f}(\boldsymbol{x})$ that can rely on this formulation.

7.5.1 Robot manipulator

iLQR_manipulator.*

We define a manipulation task involving a set of transformations as in Fig. 14. By relying on these transformation operators, we will next describe all variables in the robot frame of reference (defined by $\boldsymbol{0}$, $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in the figure).

For a manipulator controlled by joint angle velocity commands $\boldsymbol{u} = \dot{\boldsymbol{x}}$, the evolution of the system is described by $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{u}_t \Delta t$, with the Taylor expansion (52) simplifying to $\boldsymbol{A}_t = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_t} = \boldsymbol{I}$ and $\boldsymbol{B}_t = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}_t} = \boldsymbol{I}\Delta t$. Similarly, a double integrator can alternatively be considered, with acceleration commands $\boldsymbol{u} = \ddot{\boldsymbol{x}}$ and states composed of both positions and velocities.

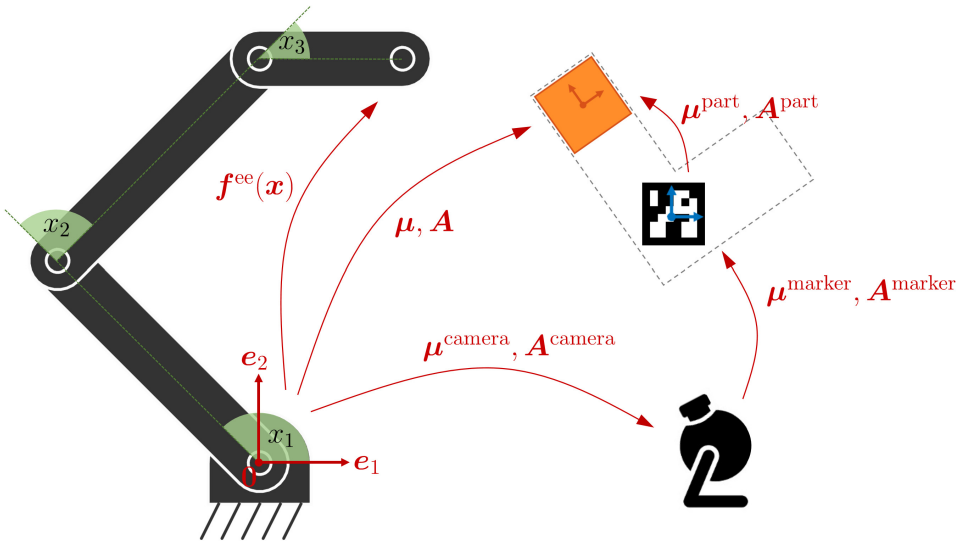


Figure 14: Typical transformations involved in a manipulation task involving a robot, a vision system, a visual marker on the object, and a desired grasping location on the object.

For a robot manipulator, $\boldsymbol{f}(\boldsymbol{x}_t)$ in (77) typically represents the error between a reference $\boldsymbol{\mu}_t$ and the end-effector position computed by the forward kinematics function $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t)$. We then have

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}_t) &= \boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t) - \boldsymbol{\mu}_t,\\ \boldsymbol{J}(\boldsymbol{x}_t) &= \boldsymbol{J}^{\text{ee}}(\boldsymbol{x}_t).\end{aligned}$$

For the orientation part of the data (if considered), the Euclidean distance vector $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t) - \boldsymbol{\mu}_t$ is replaced by a geodesic distance measure computed with the logarithmic map $\log_{\boldsymbol{\mu}_t}\big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t)\big)$, see [3] for details.

The approach can similarly be extended to target objects/landmarks with positions $\boldsymbol{\mu}_t$ and orientation matrices $\boldsymbol{U}_t$, whose columns are basis vectors forming a coordinate system, see Fig. 16. We can then define an error between the robot end-effector and an object/landmark expressed in the object/landmark coordinate system as

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}_t) &= \boldsymbol{U}_t^{\top}\big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t) - \boldsymbol{\mu}_t\big),\\ \boldsymbol{J}(\boldsymbol{x}_t) &= \boldsymbol{U}_t^{\top}\boldsymbol{J}^{\text{ee}}(\boldsymbol{x}_t).\end{aligned} \tag{78}$$

7.5.2 Bounded joint space

The iLQR solution in (77) can be used to keep the state within a boundary (e.g., joint angle limits). We denote $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{\text{cut}}(\boldsymbol{x})$ as the vertical concatenation of $\boldsymbol{f}^{\text{cut}}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{J}^{\text{cut}}(\boldsymbol{x})$ as a diagonal concatenation of diagonal Jacobian matrices $\boldsymbol{J}^{\text{cut}}(\boldsymbol{x}_t)$. Each element i of $\boldsymbol{f}^{\text{cut}}(\boldsymbol{x}_t)$ and $\boldsymbol{J}^{\text{cut}}(\boldsymbol{x}_t)$ is defined as

$$f_i^{\text{cut}}(x_{t,i}) = \begin{cases} x_{t,i} - x_i^{\min}, & \text{if } x_{t,i} < x_i^{\min}\\ x_{t,i} - x_i^{\max}, & \text{if } x_{t,i} > x_i^{\max}\\ 0, & \text{otherwise}\end{cases},$$

$$J_{i,i}^{\text{cut}}(x_{t,i}) = \begin{cases} 1, & \text{if } x_{t,i} < x_i^{\min}\\ 1, & \text{if } x_{t,i} > x_i^{\max}\\ 0, & \text{otherwise}\end{cases},$$

where $f_i^{\text{cut}}(x_{t,i})$ describes continuous ReLU-like functions for each dimension. $f_i^{\text{cut}}(x_{t,i})$ is 0 inside the bounded domain and takes the signed distance value outside the boundary, see Fig. 15-*left*.

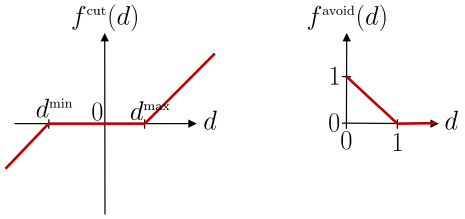


Figure 15: ReLU-like functions used in optimization costs. The derivatives of these functions are simple, providing Jacobians whose entries are either 0 or ±1.

We can see with (74) that for $\boldsymbol{Q} = \frac{1}{2}\boldsymbol{I}$, if $\boldsymbol{x}$ is outside the domain during some time steps t, $\boldsymbol{g}_{\boldsymbol{x}} = \frac{\partial c}{\partial \boldsymbol{x}} = 2\boldsymbol{J}^{\top}\boldsymbol{Q}\boldsymbol{f}$ generates a vector bringing it back to the boundary of the domain.

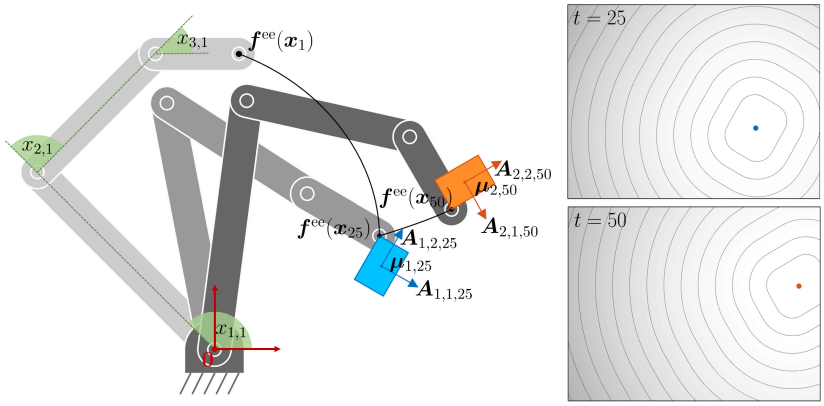


Figure 16: Example of a viapoints task in which a planar robot with 3 joints needs to sequentially reach 2 objects, with object boundaries defining the allowed reaching points on the objects surfaces. *Left:* Reaching task with two viapoints at $t = 25$ and $t = 50$. *Right:* Corresponding values of the cost function for the end-effector space at $t = 25$ and $t = 50$.

7.5.3 Bounded task space

The iLQR solution in (77) can be used to keep the end-effector within a boundary (e.g., end-effector position limits). Based on the above definitions, $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{J}(\boldsymbol{x})$ are in this case defined as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}_t) &= \boldsymbol{f}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big), \\ \boldsymbol{J}(\boldsymbol{x}_t) &= \boldsymbol{J}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big)\, \boldsymbol{J}^{\text{ee}}(\boldsymbol{x}_t), \\ \text{with} \quad \boldsymbol{e}(\boldsymbol{x}_t) &= \boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t). \end{aligned}$$

7.5.4 Reaching task with robot manipulator and prismatic object boundaries

`iLQR_manipulator.*`

The definition of $\boldsymbol{f}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}_t)$ in (78) can also be extended to objects/landmarks with boundaries by defining

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}_t) &= \boldsymbol{f}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big), \\ \boldsymbol{J}(\boldsymbol{x}_t) &= \boldsymbol{J}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big)\, \boldsymbol{U}_t^{\top}\, \boldsymbol{J}^{\text{ee}}(\boldsymbol{x}_t), \\ \text{with} \quad \boldsymbol{e}(\boldsymbol{x}_t) &= \boldsymbol{U}_t^{\top}\big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t) - \boldsymbol{\mu}_t\big), \end{aligned}$$

see also Fig. 16.

7.5.5 Center of mass

`iLQR_manipulator_CoM.*`

If we assume an equal mass for each link concentrated at the joint (i.e., assuming that the motors and gripper are heavier than the link structures), the forward kinematics function to determine the center of a mass of an articulated chain can simply be computed with

$$\boldsymbol{f}^{\text{CoM}} = \frac{1}{D}\begin{bmatrix} \boldsymbol{\ell}^{\top}\boldsymbol{L}\cos(\boldsymbol{L}\boldsymbol{x}) \\ \boldsymbol{\ell}^{\top}\boldsymbol{L}\sin(\boldsymbol{L}\boldsymbol{x}) \end{bmatrix},$$

which corresponds to the row average of $\tilde{f}$ in (8) for the first two columns (position). The corresponding Jacobian matrix is

$$\boldsymbol{J}^{\text{CoM}} = \frac{1}{D}\begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x})^{\top}\boldsymbol{L}\,\operatorname{diag}(\boldsymbol{\ell}^{\top}\boldsymbol{L}) \\ \cos(\boldsymbol{L}\boldsymbol{x})^{\top}\boldsymbol{L}\,\operatorname{diag}(\boldsymbol{\ell}^{\top}\boldsymbol{L}) \end{bmatrix}.$$

The forward kinematics function $\boldsymbol{f}^{\text{CoM}}$ can be used in tracking tasks similar to the ones considering the end-effector, as in Section 7.5.4. Fig. 17 shows an example in which the starting and ending poses are depicted in different grayscale levels. The corresponding center of mass is depicted by a darker semi-filled disc. The area allowed for the center of mass is depicted as a transparent red rectangle. The task consists of reaching the green object with the end-effector at the end of the movement, while always keeping the center of mass within the desired bounding box during the movement.

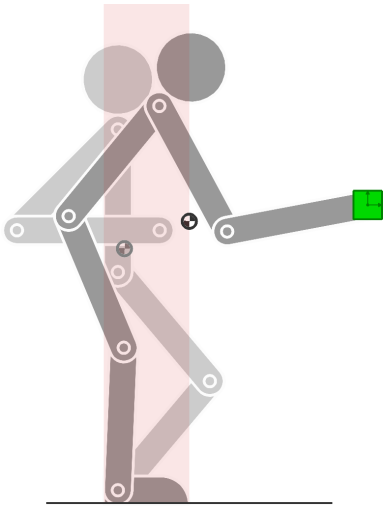


Figure 17: Reaching task with a humanoid (side view) by keeping the center of mass in an area defined by the feet location.

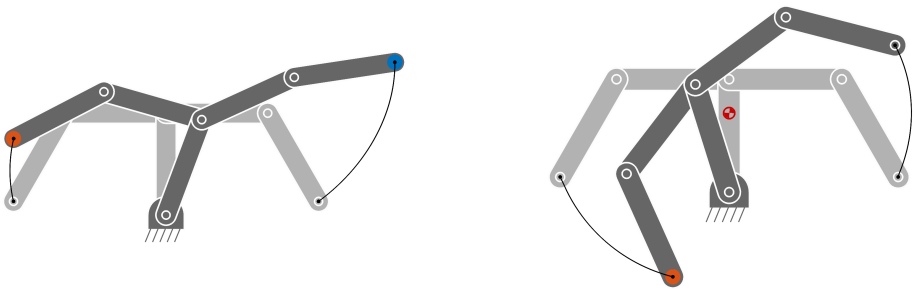


Figure 18: Reaching tasks with a bimanual robot (frontal view). *Left:* with a target for each hand. *Right:* with a target for the hand on the left, while keeping the center of mass at the same location during the whole movement.

7.5.6 Bimanual robot

`iLQR_bimanual.*`

We consider a 5-link planar bimanual robot with a torso link shared by the two arms, see Fig. 18. We define the forward kinematics function as

$$\boldsymbol{f} = \begin{bmatrix} \boldsymbol{\ell}_{[1,2,3]}^\top \cos(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]}) \\ \boldsymbol{\ell}_{[1,2,3]}^\top \sin(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]}) \\ \boldsymbol{\ell}_{[1,4,5]}^\top \cos(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]}) \\ \boldsymbol{\ell}_{[1,4,5]}^\top \sin(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]}) \end{bmatrix},$$

where the first two and last two dimensions of $\boldsymbol{f}$ correspond to the position of the left and right end-effectors, respectively. The corresponding Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{4\times 5}$ has entries

$$\boldsymbol{J}_{[1,2],[1,2,3]} = \begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]})^\top \mathrm{diag}(\boldsymbol{\ell}_{[1,2,3]})\boldsymbol{L} \\ \cos(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]})^\top \mathrm{diag}(\boldsymbol{\ell}_{[1,2,3]})\boldsymbol{L} \end{bmatrix}, \quad \boldsymbol{J}_{[3,4],[1,4,5]} = \begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]})^\top \mathrm{diag}(\boldsymbol{\ell}_{[1,4,5]})\boldsymbol{L} \\ \cos(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]})^\top \mathrm{diag}(\boldsymbol{\ell}_{[1,4,5]})\boldsymbol{L} \end{bmatrix},$$

where the remaining entries are zeros.

If we assume a unit mass for each arm link concentrated at the joint and a mass of two units at the tip of the first link (i.e., assuming that the motors and gripper are heavier than the link structures, and that two motors are located at the tip of the first link to control the left and right arms), the forward kinematics function to determine the center of a mass of the bimanual articulated chain in Fig. 18 can be computed with

$$\boldsymbol{f}^{\text{CoM}} = \frac{1}{6} \begin{bmatrix} \boldsymbol{\ell}_{[1,2,3]}^\top \boldsymbol{L} \cos(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]}) + \boldsymbol{\ell}_{[1,4,5]}^\top \boldsymbol{L} \cos(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]}) \\ \boldsymbol{\ell}_{[1,2,3]}^\top \boldsymbol{L} \sin(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]}) + \boldsymbol{\ell}_{[1,4,5]}^\top \boldsymbol{L} \sin(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]}) \end{bmatrix},$$

with the corresponding Jacobian matrix $\boldsymbol{J}^{\text{CoM}} \in \mathbb{R}^{2\times 5}$ computed as

$$\boldsymbol{J}^{\text{CoM}} = \begin{bmatrix} \boldsymbol{J}^{\text{CoM-L}}_{[1,2],1} + \boldsymbol{J}^{\text{CoM-R}}_{[1,2],1}, & \boldsymbol{J}^{\text{CoM-L}}_{[1,2],[2,3]}, & \boldsymbol{J}^{\text{CoM-R}}_{[1,2],[2,3]} \end{bmatrix},$$

$$\text{with} \quad \boldsymbol{J}^{\text{CoM-L}} = \frac{1}{6} \begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]})^\top \boldsymbol{L}\, \mathrm{diag}(\boldsymbol{\ell}_{[1,2,3]}^\top \boldsymbol{L}) \\ \cos(\boldsymbol{L}\boldsymbol{x}_{[1,2,3]})^\top \boldsymbol{L}\, \mathrm{diag}(\boldsymbol{\ell}_{[1,2,3]}^\top \boldsymbol{L}) \end{bmatrix}, \quad \boldsymbol{J}^{\text{CoM-R}} = \frac{1}{6} \begin{bmatrix} -\sin(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]})^\top \boldsymbol{L}\, \mathrm{diag}(\boldsymbol{\ell}_{[1,4,5]}^\top \boldsymbol{L}) \\ \cos(\boldsymbol{L}\boldsymbol{x}_{[1,4,5]})^\top \boldsymbol{L}\, \mathrm{diag}(\boldsymbol{\ell}_{[1,4,5]}^\top \boldsymbol{L}) \end{bmatrix}.$$

7.5.7 Obstacle avoidance with ellipsoid shapes

`iLQR_obstacle.*`

By taking as example a point-mass system, iLQR can be used to solve an ellipsoidal obstacle avoidance problem with the cost (typically used with other costs, see Fig. 19 for an example). We first define a ReLU-like function and its gradient as (see also Fig. 15-*right*)

$$f^{\text{avoid}}(d) = \begin{cases} 0, & \text{if } d > 1 \\ 1-d, & \text{otherwise} \end{cases}, \qquad g^{\text{avoid}}(d) = \begin{cases} 0, & \text{if } d > 1 \\ -1, & \text{otherwise} \end{cases},$$

that we exploit to define $\boldsymbol{f}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}_t)$ in (77) as

$$\boldsymbol{f}(\boldsymbol{x}_t) = f^{\text{avoid}}\Big(e(\boldsymbol{x}_t)\Big), \qquad \boldsymbol{J}(\boldsymbol{x}_t) = 2\, g^{\text{avoid}}\Big(e(\boldsymbol{x}_t)\Big)\, (\boldsymbol{x}_t - \boldsymbol{\mu})^\top\, \boldsymbol{\Sigma}^{-1},$$

$$\text{with} \quad e(\boldsymbol{x}_t) = (\boldsymbol{x}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_t - \boldsymbol{\mu}).$$

In the above, $\boldsymbol{f}(\boldsymbol{x})$ defines a continuous function that is 0 outside the obstacle boundaries and 1 at the center of the obstacle. The ellipsoid is

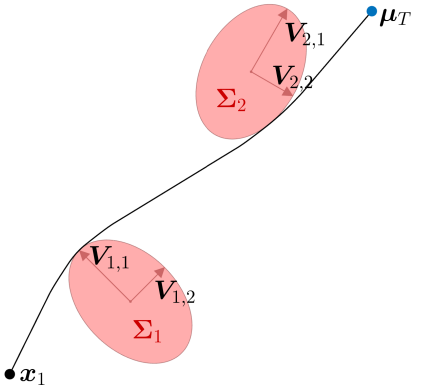


Figure 19: Reaching task with obstacle avoidance, by starting from $\boldsymbol{x}_1$ and reaching $\boldsymbol{\mu}_T$ while avoiding the two obstacles in red.

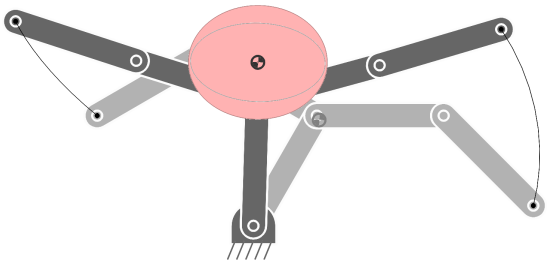


Figure 20: Example of iLQR optimization with a cost on manipulability, where the goal is to reach, at the end of the motion, a desired manipulability defined at the center of mass (CoM) of a bimanual planar robot. The initial pose of the bimanual robot is displayed in light gray and the final pose of the robot is displayed in dark gray. The desired manipulability is represented by the red ellipse. The achieved manipulability at the end of the motion is represented by the gray ellipse. Both manipulability ellipses are displayed at the CoM reached at the end of the motion, displayed as a semi-filled dark gray disc (the CoM at the beginning of the motion is displayed in lighter gray color). We can see that the posture adopted by the robot to approach the desired manipulability is to extend both arms, which is the pose allowing the robot to swiftly move its center of mass in case of unexpected perturbations.

defined by a center $\boldsymbol{\mu}$ and a shape $\boldsymbol{\Sigma} = \boldsymbol{V}\boldsymbol{V}^\top$, where $\boldsymbol{V}$ is a horizontal concatenation of vectors $\boldsymbol{V}_i$ describing the principal axes of the ellipsoid, see Fig. 19.

A similar principle can be applied to robot manipulators by composing forward kinematics and obstacle avoidance functions.

7.5.8 Maintaining a desired distance to an object

`iLQR_distMaintenance.*`

The obstacle example above can easily be extended to the problem of maintaining a desired distance to an object, which can also typically used with other objectives. We first define a function and a gradient

$$f^{\text{dist}}(d) = 1 - d, \qquad g^{\text{dist}}(d) = -1,$$

that we exploit to define $\boldsymbol{f}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}_t)$ in (77) as

$$\boldsymbol{f}(\boldsymbol{x}_t) = f^{\text{dist}}\Big(e(\boldsymbol{x}_t)\Big), \qquad \boldsymbol{J}(\boldsymbol{x}_t) = -\frac{2}{r^2}(\boldsymbol{x}_t - \boldsymbol{\mu})^\top,$$
$$\text{with} \quad e(\boldsymbol{x}_t) = \frac{1}{r^2}(\boldsymbol{x}_t - \boldsymbol{\mu})^\top(\boldsymbol{x}_t - \boldsymbol{\mu}).$$

In the above, $\boldsymbol{f}(\boldsymbol{x})$ defines a continuous function that is 0 on a sphere of radius r centered on the object (defined by a center $\boldsymbol{\mu}$), and increasing/decreasing when moving away from this surface in one direction or the other.

7.5.9 Manipulability tracking

`iLQR_bimanual_manipulability.*`

Skills transfer can exploit manipulability ellipsoids in the form of geometric descriptors representing the skills to be transferred to the robot. As these ellipsoids lie on symmetric positive definite (SPD) manifolds, Riemannian geometry can be used to learn and reproduce these descriptors in a probabilistic manner [3].

Manipulability ellipsoids come in different flavors. They can be defined for either positions or forces (including the orientation/rotational part). Manipulability can be described at either kinematic or dynamic levels, for either open or closed linkages (e.g., for bimanual manipulation or in-hand manipulation), and for either actuated or non-actuated joints [13]. This large set of manipulability ellipsoids provide rich descriptors to characterize robot skills for robots with legs and arms.

Manipulability ellipsoids are helpful descriptors to handle skill transfer problems involving dissimilar kinematic chains, such as transferring manipulation skills between humans and robots, or between two robots with different kinematic chains or capabilities. In such transfer problems, imitation at joint angles level is not possible due to the different structures, and imitation at end-effector(s) level is limited, because it does not encapsulate postural information, which is often an essential aspect of the skill that we would like to transfer. Manipulability ellipsoids provide intermediate descriptors that allows postural information to be transferred indirectly, with the advantage that it allows different embodiments and capabilities to be considered in the skill transfer process.

The manipulability ellipsoid $\boldsymbol{M}(\boldsymbol{x}) = \boldsymbol{J}(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})^\top$ is a symmetric positive definite matrix representing the manipulability at a given point on the kinematic chain defined by a forward kinematics function $\boldsymbol{f}(\boldsymbol{x})$, given the joint angle posture $\boldsymbol{x}$, where $\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian of $\boldsymbol{f}(\boldsymbol{x})$. The determinant of $\boldsymbol{M}(\boldsymbol{x})$ is often used as a scalar manipulability index indicating the volume of the ellipsoid [15], with the drawback of ignoring the specific shape of this ellipsoid.

In [5], we showed that a geometric cost on manipulability can alternatively be defined with the geodesic distance

$$\begin{aligned} & c = \|\boldsymbol{A}\|_{\text{F}}^2, \quad \text{with} \quad \boldsymbol{A} = \log\big(\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{M}(\boldsymbol{x})\boldsymbol{S}^{\frac{1}{2}}\big), \\ \iff \quad & c = \text{trace}(\boldsymbol{A}\boldsymbol{A}^\top) = \sum_i \text{trace}(\boldsymbol{A}_i\boldsymbol{A}_i^\top) = \sum_i \boldsymbol{A}_i^\top\boldsymbol{A}_i = \text{vec}(\boldsymbol{A})^\top\text{vec}(\boldsymbol{A}), \end{aligned} \tag{79}$$

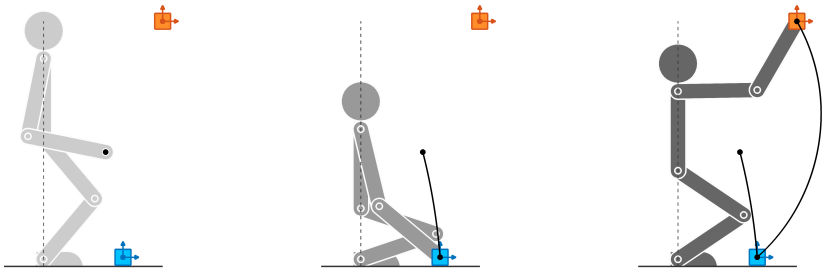


Figure 21: Reaching task with control primitives.

where $\boldsymbol{S}$ is the desired manipulability matrix to reach, $\log(\cdot)$ is the logarithm matrix function, and $\|\cdot\|_{\mathrm{F}}$ is a Frobenius norm. By exploiting the trace properties, we can see that (79) can be expressed in a quadratic form, where the vectorization operation $\mathrm{vec}(\boldsymbol{A})$ can be computed efficiently by exploiting the symmetry of the matrix $\boldsymbol{A}$. This is implemented by keeping only the lower half of the matrix, with the elements below the diagonal multiplied by a factor $\sqrt{2}$.

The derivatives of (79) with respect to the state $\boldsymbol{x}$ and control commands $\boldsymbol{u}$ can be computed numerically (as in the provided example), analytically (by following an approach similar to the one presented in [5]), or by automatic differentiation. The quadratic form of (79) can be exploited to solve the iLQR problem with Gauss–Newton optimization, by using Jacobians (see description in Section 4.1 within the context of inverse kinematics). Marić *et al.* demonstrated the advantages of a geometric cost as in (79) for planning problems, by comparing it to alternative widely used metrics such as the manipulability index and the dexterity index [7].

Note here that manipulability can be defined at several points of interest, including endeffectors and centers of mass. Figure 20 presents a simple example with a bimanual robot. Manipulability ellipsoids can also be exploited as descriptors for object affordances, by defining manipulability at the level of specific points on objects or tools held by the robot. For example, we can consider a manipulability ellipsoid at the level of the head of a hammer. When the robot grasps the hammer, its endeffector is extended so that the head of the hammer becomes the extremity of the kinematic chain. In this situation, the different options that the robot has to grasp the hammer will have an impact on the resulting manipulability at the head of the hammer. Thus, grabbing the hammer at the extremity of the handle will improve the resulting manipulability. Such descriptors offer promises for learning and optimization in manufacturing environments, in order to let robots automatically determine the correct ways to employ tools, including grasping points and body postures.

The above iLQR approach, with a geodesic cost on SPD manifolds, has been presented for manipulability ellipsoids, but it can be extended to other descriptors represented as ellipsoids. In particular, stiffness, feedback gains, inertia, and centroidal momentum have similar structures. For the latter, the centroidal momentum ellipsoid quantifies the momentum generation ability of the robot [11], which is another useful descriptor for robotics skills. Similarly to manipulability ellipsoids, it is constructed from Jacobians, which are in this case the centroidal momentum matrices mapping the joint angle velocities to the centroidal momentum, which sums over the individual link momenta after projecting each to the robot's center of mass.

The approach can also be extended to other forms of symmetric positive definite matrices, such as kernel matrices used in machine learning algorithms to compute similarities, or graph graph Laplacians (a matrix representation of a graph that can for example be used to construct a low dimensional embedding of a graph).

7.6 iLQR with control primitives

`iLQR_CP.*`

The use of control primitives can be extended to iLQR, by considering $\Delta\boldsymbol{u} = \boldsymbol{\Psi}\Delta\boldsymbol{w}$. For problems with quadratic cost on $\boldsymbol{f}(\boldsymbol{x}_t)$ (see previous Section), the update of the weights vector is then given by

$$\Delta\hat{\boldsymbol{w}} = \Big(\boldsymbol{\Psi}^\top \boldsymbol{S}_u^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x}) \boldsymbol{S}_u \boldsymbol{\Psi} + \boldsymbol{\Psi}^\top \boldsymbol{R}\, \boldsymbol{\Psi}\Big)^{-1} \Big(- \boldsymbol{\Psi}^\top \boldsymbol{S}_u^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{\Psi}^\top \boldsymbol{R}\, \boldsymbol{u}\Big). \tag{80}$$

For a 5-link kinematic chain, by setting the coordination matrix in (37) as

$$\boldsymbol{C} = \begin{bmatrix} -1 & 0 & 0 \\ 2 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

the first three joints are coordinated such that the third link is always oriented in the same direction, while the last two joints can move independently. Figure 21 shows an example for a reaching task, with the 5-link kinematic chain depicting a humanoid robot that needs to keep its torso upright.

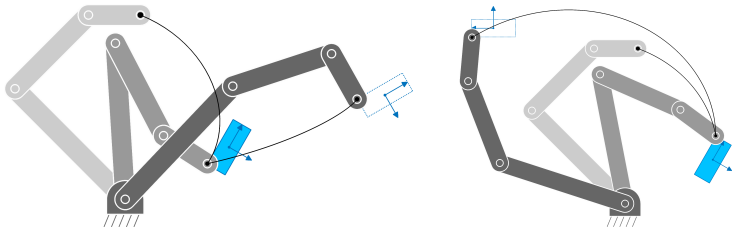


Figure 23: Manipulation with object affordances cost. The depicted *pick-and-place* task requires the offset at the picking location to be the same as the offset at the placing location, which can be achieved by setting a precision matrix with non-zero offdiagonal entries, see main text for details. In this example, an additional cost is set for choosing the picking and placing points within the object boundaries, which was also used for the task depicted in Fig. 16. We can see with the resulting motions that when picking the object, the robot anticipates what will be done later with this object, which can be viewed as a basic form of object affordance. In the two situations depicted in the figure, the robot efficiently chooses a grasping point close to one of the corners of the object, different in the two situations, so that the robot can bring the object at the desired location with less effort and without reaching joint angle limits.

Note also that the iLQR updates in (80) can usually be computed faster than in the original formulation, since a matrix of much smaller dimension needs to be inverted.

7.7 iLQR for spacetime optimization

We define a phase variable s_t starting from $s_1 = 0$ at the beginning of the movement. With an augmented state $\boldsymbol{x}_t = [\mathbf{x}_t^\top, s_t]^\top$ and control command $\boldsymbol{u}_t = [\boldsymbol{u}_t^{\mathbf{x}\top}, u_t^s]^\top$, we define the evolution of the system $\boldsymbol{x}_{t+1} = \boldsymbol{d}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ as

$$\begin{aligned}\mathbf{x}_{t+1} &= \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\boldsymbol{u}_t^{\mathbf{x}}\, u_t^s,\\ s_{t+1} &= s_t + u_t^s.\end{aligned}$$

Its first order Taylor expansion around $\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}$ provides the linear system

$$\underbrace{\begin{bmatrix}\Delta\mathbf{x}_{t+1}\\ \Delta s_{t+1}\end{bmatrix}}_{\Delta\boldsymbol{x}_{t+1}} = \underbrace{\begin{bmatrix}\mathbf{A}_t & \mathbf{0}\\ \mathbf{0} & 1\end{bmatrix}}_{\boldsymbol{A}_t}\underbrace{\begin{bmatrix}\Delta\mathbf{x}_t\\ \Delta s_t\end{bmatrix}}_{\Delta\boldsymbol{x}_t} + \underbrace{\begin{bmatrix}\mathbf{B}_t u_t^s & \mathbf{B}_t\boldsymbol{u}_t^{\mathbf{x}}\\ \mathbf{0} & 1\end{bmatrix}}_{\boldsymbol{B}_t}\underbrace{\begin{bmatrix}\Delta\boldsymbol{u}_t^{\mathbf{x}}\\ \Delta u_t^s\end{bmatrix}}_{\Delta\boldsymbol{u}_t},$$

with Jacobian matrices $\boldsymbol{A}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}_t}$, $\boldsymbol{B}_t = \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{u}_t}$. Note that in the above equations, we used here the notation $\mathbf{x}_t$ and $\boldsymbol{x}_t$ to describe the standard state and augmented state, respectively. We similarly used $\{\mathbf{A}_t, \mathbf{B}_t\}$ and $\{\boldsymbol{A}_t, \boldsymbol{B}_t\}$ (with slanted font) for standard linear system and augmented linear system.

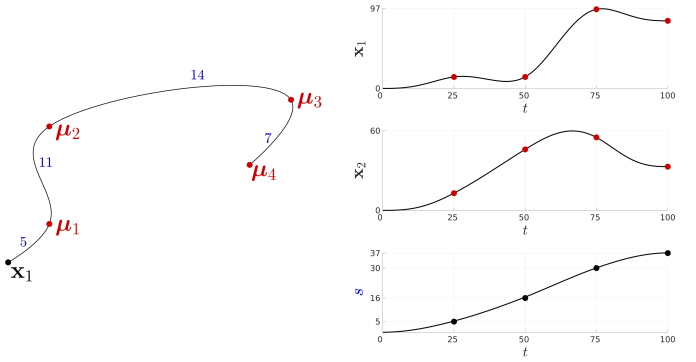


Figure 22: Spacetime optimization with iLQR for a viapoints task, where the estimated durations between consecutive viapoints are shown in blue.

Figure 22 shows a viapoints task example in which both path and duration are optimized (convergence after 10 iterations when starting from zero commands). The example uses a double integrator as linear system defined by constant $\mathbf{A}_t$ and $\mathbf{B}_t$ matrices.

7.8 iLQR with offdiagonal elements in the precision matrix

`iLQR_manipulator_object_affordance.*`

Spatial and/or temporal constraints can be considered by setting $\left[\begin{smallmatrix} \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I}\end{smallmatrix}\right]$ in the corresponding entries of the precision matrix $\boldsymbol{Q}$. With this formulation, we can constraint two positions to be the same without having to predetermine the position at which the two should meet. Indeed, we can see that a cost $c = \left[\begin{smallmatrix} x_i \\ x_j\end{smallmatrix}\right]^\top \left[\begin{smallmatrix} 1 & -1 \\ -1 & 1\end{smallmatrix}\right] \left[\begin{smallmatrix} x_i \\ x_j\end{smallmatrix}\right] = (x_i - x_j)^2$ is minimized when $x_i = x_j$.

Such cost can for example be used to define costs on object affordances, see Fig. 23 for an example.

7.9 Car steering

`iLQR_car.*`

Velocity Control Input

A car motion with position (x_1, x_2), car orientation θ, and front wheels orientation ϕ (see Fig. 24-*left*) is characterized by equations

$$\dot{x}_1 = u_1\cos(\theta), \quad \dot{x}_2 = u_1\sin(\theta), \quad \dot{\theta} = \frac{u_1}{\ell}\tan(\phi), \quad \dot{\phi} = u_2,$$

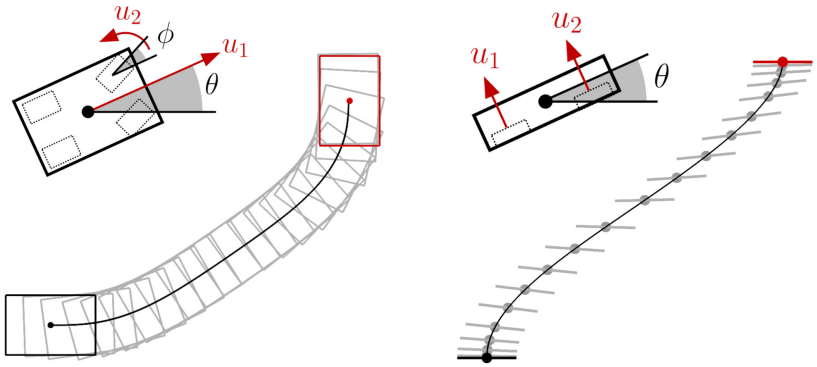


Figure 24: iLQR for car and bicopter control/planning problems.

where ℓ is the distance between the front and back axles, u_1 is the back wheels velocity command and u_2 is the front wheels steering velocity command. Its first order Taylor expansion around $(\hat{\theta}, \hat{\phi}, \hat{u}_1)$ is

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{u}_1 \sin(\hat{\theta}) & 0 \\ 0 & 0 & \hat{u}_1 \cos(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{\hat{u}_1}{\ell}(\tan(\hat{\phi})^2+1) \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} \cos(\hat{\theta}) & 0 \\ \sin(\hat{\theta}) & 0 \\ \frac{1}{\ell}\tan(\hat{\phi}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{81}$$

which can then be converted to a discrete form with (96).

Acceleration Control Input

A car motion with position (x_1, x_2), car orientation θ, and front wheels orientation ϕ (see Fig. 24-*left*) is characterized by equations

$$\dot{x}_1 = v\cos(\theta), \quad \dot{x}_2 = v\sin(\theta), \quad \dot{\theta} = \frac{v}{\ell}\tan(\phi), \quad \dot{v} = u_1, \quad \dot{\phi} = u_2,$$

where ℓ is the distance between the front and back axles, u_1 is the back wheels acceleration command and u_2 is the front wheels steering velocity command. Its first order Taylor expansion around $(\hat{\theta}, \hat{\phi}, \hat{v})$ is

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{v} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{v}\sin(\hat{\theta}) & \cos(\hat{\theta}) & 0 \\ 0 & 0 & \hat{v}\cos(\hat{\theta}) & \sin(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{1}{\ell}\tan(\hat{\phi}) & \frac{\hat{v}}{\ell}(\tan(\hat{\phi})^2+1) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta v \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{82}$$

which can then be converted to a discrete form with (96).

7.10 Bicopter

`iLQR_bicopter.*`

A planar bicopter of mass m and inertia I actuated by two propellers at distance ℓ (see Fig. 24-*right*) is characterized by equations

$$\ddot{x}_1 = -\frac{1}{m}(u_1+u_2)\sin(\theta), \quad \ddot{x}_2 = \frac{1}{m}(u_1+u_2)\cos(\theta) - g, \quad \ddot{\theta} = \frac{\ell}{I}(u_1-u_2),$$

with acceleration $g = 9.81$ due to gravity. Its first order Taylor expansion around $(\hat{\theta}, \hat{u}_2, \hat{u}_2)$ is

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\ddot{x}_1 \\ \Delta\ddot{x}_2 \\ \Delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1+\hat{u}_2)\cos(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1+\hat{u}_2)\sin(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{1}{m}\sin(\hat{\theta}) & -\frac{1}{m}\sin(\hat{\theta}) \\ \frac{1}{m}\cos(\hat{\theta}) & \frac{1}{m}\cos(\hat{\theta}) \\ \frac{\ell}{I} & -\frac{\ell}{I} \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{83}$$

which can then be converted to a discrete form with (96).

8 Forward dynamics (FD) for a planar robot manipulator

`FD.*`

The dynamic equation of a planar robot with an arbitrary number of links can be derived using the Lagrangian formulation. The first step to use this method is to represent the kinetic and potential energies of the robot as functions of generalized coordinates, which are joint angles in this case. We assume that all masses are located at the end of each

link, and we neglect the terms that contain rotational energies (i.e, zero moments of inertia). To do this, we exploit the formulation derived in Section 3 without the orientation part, so the position of the j-*th* mass can be written as

$$\boldsymbol{f}_j = \begin{bmatrix} f_{j,1} \\ f_{j,2} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{j} l_k c_k \\ \sum_{k=1}^{j} l_k s_k \end{bmatrix}, \quad \text{where} \quad c_k = \cos(\sum_{i=1}^{k} x_i), \quad s_k = \sin(\sum_{i=1}^{k} x_i), \tag{84}$$

whose corresponding velocity can be calculated as

$$\dot{\boldsymbol{f}}_j = \begin{bmatrix} -\sum_{k=1}^{j} l_k (\sum_{i=1}^{k} \dot{x}_i) s_k \\ \sum_{k=1}^{j} l_k (\sum_{i=1}^{k} \dot{x}_i) c_k \end{bmatrix}.$$

By knowing the velocity, the j-*th* point kinetic energy can be expressed as

$$T_j = \frac{1}{2} m_j (\dot{f}_{j,1}^2 + \dot{f}_{j,2}^2) = \frac{1}{2} m_j \left(\Big(-\sum_{k=1}^{j} l_k \Big(\sum_{i=1}^{k} \dot{x}_i \Big) s_k \Big)^2 + \Big(\sum_{k=1}^{j} l_k \Big(\sum_{i=1}^{k} \dot{x}_i \Big) c_k \Big)^2 \right),$$

and its potential energy as

$$U_j = m_j \, g \, f_{j,2} = m_j g \Big(\sum_{k=1}^{j} l_k s_k \Big).$$

The kinetic and potential energies are more dependent on absolute angles $\sum_{i=1}^{k} x_i$ rather than relative ones x_i. This is because the energies of the system are coordinate invariant and are dependent on absolute values. For simplicity, we define the absolute angles as

$$q_k = \sum_{i=1}^{k} x_i,$$

so the kinetic energy can be redefined as

$$T_j = \frac{1}{2} m_j \Big(\Big(-\sum_{k=1}^{j} l_k \dot{q}_k S_k \Big)^2 + \Big(\sum_{k=1}^{j} l_k \dot{q}_k C_k \Big)^2 \Big).$$

The robot's total kinetic and potential energies are the sum of their corresponding values in each point mass. So, the Lagrangian of the whole system can be written as

$$L = \sum_{j=1}^{N} T_j - U_j,$$

where N is the number of links. Using the Euler-Lagrange equation, we can write

$$u_z = \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_z} - \frac{\partial L}{\partial q_z} \quad = \frac{d}{dt} \frac{\partial (\sum_{j=z}^{N} T_j)}{\partial \dot{q}_z} - \frac{\partial \sum_{j=z}^{N} (T_j - U_j)}{\partial q_z} \quad = \sum_{j=z}^{N} \Big(\frac{d}{dt} \frac{\partial T_j}{\partial \dot{q}_z} - \frac{\partial T_j}{\partial q_z} + \frac{\partial U_j}{\partial q_z} \Big), \tag{85}$$

where u_z is the generalized force related to q_z. This force can be found from the corresponding generalized work w_z, which can be described by

$$w_z = \begin{cases} (\tau_z - \tau_{z+1}) \dot{q}_z & z < N \\ \tau_N \dot{q}_N & z = N \end{cases}, \tag{86}$$

where τ_z is the torque applied at z-*th* joint. The fact that we need subtraction in the first line of (86) is that when τ_{z+1} is applied on link l_{z+1}, its reaction is applied on link l_z, except for the last joint where there is no reaction from the proceeding links. Please note that if we had used relative angles in our formulation, we did not need to consider this reaction as it will be cancelled by itself at link $z+1$ (as $\dot{q}_{z+1} = \dot{q}_z + \dot{x}_{z+1}$). That said, generalized forces can be defined as

$$u_z = \frac{\partial w_z}{\partial \dot{q}_z} = \begin{cases} (\tau_z - \tau_{z+1}) & z < N \\ \tau_N & z = N \end{cases}.$$

By substituting the derived equations into (85), the dynamic equation of a general planar robot with an arbitrary number of links can be expressed as (for more details, please refer to Appendix B)

$$\sum_{j=z}^{N} m_j \Big(\sum_{k=1}^{j} l_z l_k c_{z-k} \ddot{q}_k \Big) = u_z - \sum_{j=z}^{N} m_j \Big(\sum_{k=1}^{j} l_z l_k s_{z-k} \dot{q}_k^2 \Big) - \sum_{j=z}^{N} m_j g l_z c_z, \quad \text{where} \quad \begin{cases} c_{h-k} = c_h c_k - s_h s_k, \\ s_{h-k} = s_h c_k - c_h s_k. \end{cases} \tag{87}$$

In (87), the coefficient of $\ddot{q}_k$ can be written as

$$l_z l_k c_{z-k} \sum_{j=z}^{N} m_j \mathbb{1}(j-k),$$

where $\mathbb{1}(\cdot)$ is a function that returns 0 if it receives a negative input, and 1 otherwise. The coefficient of $\dot{q}_k^2$ can be found in a similar way as

$$-l_z l_k s_{z-k} \sum_{j=z}^{N} m_j \mathbb{1}(j-k).$$

By concatenation of the results for all z's, the dynamic equation of the whole system can be expressed as

$$\boldsymbol{M}\ddot{\boldsymbol{q}} = \boldsymbol{u} + \boldsymbol{g} + \boldsymbol{C}\text{diag}(\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}, \tag{88}$$

where

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_{N-1} \\ q_N \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix}, \quad \boldsymbol{M}(z,k) = l_z l_k C_{z-k} \sum_{j=z}^{N} m_j \mathbb{1}(j-k), \quad \boldsymbol{C}(z,k) = -l_z l_k S_{z-k} \sum_{j=z}^{N} m_j \mathbb{1}(j-k),$$

$$\boldsymbol{g} = \begin{bmatrix} -\sum_{j=1}^{N} m_j l_1 g \cos(q_1) \\ -\sum_{j=2}^{N} m_j l_2 g \cos(q_2) \\ \vdots \\ -\sum_{j=N-1}^{N} m_j l_{N-1} g \cos(q_{N-1}) \\ -m_N l_N g \cos(q_N) \end{bmatrix}.$$

To use this equation with relative angles and torque commands, one can replace the absolute angles and general forces by

$$\boldsymbol{q} = \boldsymbol{L}\boldsymbol{x}, \quad \boldsymbol{u} = \boldsymbol{L}^{-\top}\boldsymbol{\tau}, \quad \text{where} \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}, \quad \boldsymbol{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_{N-1} \\ \tau_N \end{bmatrix}.$$

By substituting these variables into (88), we would have

$$\boldsymbol{L}^\top \boldsymbol{M}\boldsymbol{L}\ddot{\boldsymbol{x}} = \boldsymbol{\tau} + \boldsymbol{L}^\top \boldsymbol{g} + \boldsymbol{L}^\top \boldsymbol{C}\text{diag}(\boldsymbol{L}\dot{\boldsymbol{x}})\boldsymbol{L}\dot{\boldsymbol{x}}. \tag{89}$$

Please note that elements in $\boldsymbol{M}, \boldsymbol{C}$ and $\boldsymbol{g}$ are defined with absolute angles, which can be replaced with the relative angles by

$$q_i = \boldsymbol{L}_i \boldsymbol{x},$$

where $\boldsymbol{L}_i$ is the *i-th* row of $\boldsymbol{L}$.

8.1 Robot manipulator with dynamics equation

`iLQR_manipulator_dynamics.*`

The nonlinear dynamic equation of a planar robot presented in Section 8 can be expressed as

$$\begin{bmatrix} \boldsymbol{q}_{t+1} \\ \dot{\boldsymbol{q}}_{t+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t dt \\ \dot{\boldsymbol{q}}_t + \boldsymbol{M}^{-1}\Big(\boldsymbol{u} + \boldsymbol{g} + \boldsymbol{C}\text{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big)dt \end{bmatrix}. \tag{90}$$

For simplicity we start with (88), and at the end, we will describe how the result can be converted to the other choice of variables. The iLQR method requires (90) to be linearized around the current states and inputs.The corresponding $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ can be found according to (52) as

$$\boldsymbol{A}_t = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I}dt \\ \boldsymbol{A}_{21}dt & \boldsymbol{I} + 2\boldsymbol{M}^{-1}\boldsymbol{C}\text{diag}(\dot{\boldsymbol{q}}_t)dt \end{bmatrix}, \quad \boldsymbol{B}_t = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{M}^{-1}dt \end{bmatrix}.$$

$\boldsymbol{A}_{21} = \frac{\partial \ddot{\boldsymbol{q}}_t}{\partial \boldsymbol{q}_t}$ can be found by taking the derivation of (88) w.r.t. $\boldsymbol{q}_t$. The *p-th* column of $\boldsymbol{A}_{21}$ is the partial derivation of $\ddot{\boldsymbol{q}}_t$ w.r.t. p-emphth joint angle at time t q_p^t which can be formulated as

$$\frac{\partial \ddot{\boldsymbol{q}}_t}{\partial q_p^t} = \frac{\partial \boldsymbol{M}^{-1}}{\partial q_p^t}\Big(\boldsymbol{u} + \boldsymbol{g} + \boldsymbol{C}\mathrm{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big) + \boldsymbol{M}^{-1}\Big(\frac{\partial \boldsymbol{g}}{\partial q_p^t} + \frac{\partial \boldsymbol{C}}{\partial q_p^t}\mathrm{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big).$$

Having this done for all p's at time t, we can write

$$\frac{\partial \ddot{\boldsymbol{q}}_t}{\partial \boldsymbol{q}_t} = \Big(\boldsymbol{M}^{-1}\Big)'\Big(\boldsymbol{u} + \boldsymbol{g} + \boldsymbol{C}\mathrm{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big) + \boldsymbol{M}^{-1}\Big(\boldsymbol{g}' + \boldsymbol{C}'\mathrm{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big),$$

where

$$\boldsymbol{g}(i,j) = \begin{cases} \sum_{j=i}^{N} m_j l_i g s_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases},$$

$$\boldsymbol{C}' = \begin{bmatrix} \frac{\partial \boldsymbol{C}}{\partial q_1^t} & \frac{\partial \boldsymbol{C}}{\partial q_2^t} & \cdots & \frac{\partial \boldsymbol{C}}{\partial q_N^t} \end{bmatrix}, \quad \boldsymbol{C}'(h,k,p) = \begin{cases} -l_h l_k c_{h-k} \sum_{j=h}^{N} m_j \mathbb{1}(j-k) & \text{if } p = h \neq k \\ l_h l_k c_{h-k} \sum_{j=h}^{N} m_j \mathbb{1}(j-k) & \text{if } p = k \neq h \\ 0 & \text{otherwise} \end{cases},$$

$$(\boldsymbol{M}^{-1})' = \begin{bmatrix} \frac{\partial \boldsymbol{M}^{-1}}{\partial q_1^t} & \frac{\partial \boldsymbol{M}^{-1}}{\partial q_2^t} & \cdots & \frac{\partial \boldsymbol{M}^{-1}}{\partial q_N^t} \end{bmatrix}, \quad \Big(\boldsymbol{M}^{-1}\Big)' = -\boldsymbol{M}^{-1}\boldsymbol{M}'\boldsymbol{M}^{-1},$$

$$\boldsymbol{M}'(h,k,p) = \begin{cases} -l_h l_k s_{h-k} \sum_{j=k}^{N} m_j \mathbb{1}(j-k) & \text{if } p = h \neq k \\ l_h l_k s_{h-k} \sum_{j=h}^{N} m_j \mathbb{1}(j-k) & \text{if } p = k \neq h \\ 0 & \text{otherwise} \end{cases}.$$

According to Section 7.1, we can concatenate the results for all time steps as

$$\begin{bmatrix} \Delta \boldsymbol{q}_1 \\ \Delta \dot{\boldsymbol{q}}_1 \\ \vdots \\ \Delta \boldsymbol{q}_T \\ \Delta \dot{\boldsymbol{q}}_T \end{bmatrix} = \boldsymbol{S}_u^q \begin{bmatrix} \Delta \boldsymbol{u}_1 \\ \Delta \boldsymbol{u}_2 \\ \vdots \\ \Delta \boldsymbol{u}_{T-1} \end{bmatrix}, \tag{91}$$

where T is the total time horizon. These variables can be converted to relative angles and joint torque commands by

$$\begin{bmatrix} \Delta \boldsymbol{q}_1 \\ \Delta \dot{\boldsymbol{q}}_1 \\ \vdots \\ \Delta \boldsymbol{q}_T \\ \Delta \dot{\boldsymbol{q}}_T \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{T} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{T} \end{bmatrix}}_{\boldsymbol{T}_x} \begin{bmatrix} \Delta \boldsymbol{x}_1 \\ \Delta \dot{\boldsymbol{x}}_1 \\ \vdots \\ \Delta \boldsymbol{x}_T \\ \Delta \dot{\boldsymbol{x}}_T \end{bmatrix}, \quad \begin{bmatrix} \Delta \boldsymbol{u}_1 \\ \Delta \boldsymbol{u}_2 \\ \vdots \\ \Delta \boldsymbol{u}_{T-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{T}^{-\top} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{T}^{-\top} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{T}^{-\top} \end{bmatrix}}_{\boldsymbol{T}_u} \begin{bmatrix} \Delta \boldsymbol{\tau}_1 \\ \Delta \boldsymbol{\tau}_2 \\ \vdots \\ \Delta \boldsymbol{\tau}_{T-1} \end{bmatrix},$$

so (91) can be rewritten as

$$\begin{bmatrix} \Delta \boldsymbol{x}_1 \\ \Delta \dot{\boldsymbol{x}}_1 \\ \vdots \\ \Delta \boldsymbol{x}_T \\ \Delta \dot{\boldsymbol{x}}_T \end{bmatrix} = \boldsymbol{S}_\tau^x \begin{bmatrix} \Delta \boldsymbol{\tau}_1 \\ \Delta \boldsymbol{\tau}_2 \\ \vdots \\ \Delta \boldsymbol{\tau}_{T-1} \end{bmatrix} = \boldsymbol{T}_x^{-1} \boldsymbol{S}_u^q \boldsymbol{T}_u \begin{bmatrix} \Delta \boldsymbol{\tau}_1 \\ \Delta \boldsymbol{\tau}_2 \\ \vdots \\ \Delta \boldsymbol{\tau}_{T-1} \end{bmatrix}.$$

References

[1] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2(3):321–355, 1988.

[2] S. Calinon. Mixture models for the analysis, edition, and synthesis of continuous time series. In N. Bouguila and W. Fan, editors, *Mixture Models and Applications*, pages 39–57. Springer, 2019.

[3] S. Calinon. Gaussians on Riemannian manifolds: Applications for robot learning and adaptive control. *IEEE Robotics and Automation Magazine (RAM)*, 27(2):33–45, June 2020.

[4] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[5] N. Jaquier, L. Rozo, D. G. Caldwell, and S. Calinon. Geometry-aware manipulability learning, tracking and transfer. *International Journal of Robotics Research (IJRR)*, 40(2–3):624–650, 2021.

[6] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proc. Intl Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, pages 222–229, 2004.

[7] F. Marić, L. Petrović, M. Guberina, J. Kelly, and I. Petrović. A riemannian metric for geometry-aware singularity avoidance by articulated robots. *Robotics and Autonomous Systems*, 145:103865, 2021.

[8] D. Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[9] F. A. Mussa-Ivaldi, S. F. Giszter, and E. Bizzi. Linear combinations of primitives in vertebrate motor control. *Proc. National Academy of Sciences*, 91:7534–7538, 1994.

[10] J. Nocedal and S. Wright. *Numerical optimization*. Springer, New York, NY, 2006.

[11] D. E. Orin and A. Goswami. Centroidal momentum matrix of a humanoid robot: Structure and properties. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 653–659, 2008.

[12] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic movement primitives. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2616–2624. Curran Associates, Inc., USA, 2013.

[13] F. C. Park and J. W. Kim. Manipulability of closed kinematic chains. *Journal of Mechanical Design*, 120(4):542–548, 1998.

[14] H. H. Rosenbrock. Differential dynamic programming. *The Mathematical Gazette*, 56(395):78–78, 1972.

[15] T. Yoshikawa. Dynamic manipulability of robot manipulators. *Robotic Systems*, 2:113–124, 1985.

Appendices

A System dynamics at trajectory level

A linear system is described by

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\, \boldsymbol{x}_t + \boldsymbol{B}_t\, \boldsymbol{u}_t, \quad \forall t \in \{1, \dots, T\}$$

with states $\boldsymbol{x}_t \in \mathbb{R}^D$ and control commands $\boldsymbol{u}_t \in \mathbb{R}^d$.

With the above linearization, we can express all states $\boldsymbol{x}_t$ as an explicit function of the initial state $\boldsymbol{x}_1$. By writing

$$\begin{aligned}
\boldsymbol{x}_2 &= \boldsymbol{A}_1\boldsymbol{x}_1 + \boldsymbol{B}_1\boldsymbol{u}_1,\\
\boldsymbol{x}_3 &= \boldsymbol{A}_2\boldsymbol{x}_2 + \boldsymbol{B}_2\boldsymbol{u}_2 = \boldsymbol{A}_2(\boldsymbol{A}_1\boldsymbol{x}_1 + \boldsymbol{B}_1\boldsymbol{u}_1) + \boldsymbol{B}_2\boldsymbol{u}_2,\\
&\vdots\\
\boldsymbol{x}_T &= \left(\prod_{t=1}^{T-1} \boldsymbol{A}_{T-t}\right)\boldsymbol{x}_1 \;+\; \left(\prod_{t=1}^{T-2} \boldsymbol{A}_{T-t}\right)\boldsymbol{B}_1\boldsymbol{u}_1 + \left(\prod_{t=1}^{T-3} \boldsymbol{A}_{T-t}\right)\boldsymbol{B}_2\boldsymbol{u}_2 \;+\; \cdots \;+\; \boldsymbol{B}_{T-1}\boldsymbol{u}_{T-1},
\end{aligned}$$

in a matrix form, we get an expression of the form $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}$, with

$$\underbrace{\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \vdots\\ \boldsymbol{x}_T\end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{A}_1\\ \boldsymbol{A}_2\boldsymbol{A}_1\\ \vdots\\ \prod_{t=1}^{T-1}\boldsymbol{A}_{T-t}\end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{x}}}\boldsymbol{x}_1 + \underbrace{\begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{B}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{A}_2\boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots\\ \left(\prod_{t=1}^{T-2}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_1 & \left(\prod_{t=1}^{T-3}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_2 & \cdots & \boldsymbol{B}_{T-1}\end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{u}}}\underbrace{\begin{bmatrix}\boldsymbol{u}_1\\ \boldsymbol{u}_2\\ \vdots\\ \boldsymbol{u}_{T-1}\end{bmatrix}}_{\boldsymbol{u}}, \tag{92}$$

where $\boldsymbol{S}_{\boldsymbol{x}} \in \mathbb{R}^{dT\times d}$, $\boldsymbol{x}_1 \in \mathbb{R}^d$, $\boldsymbol{S}_{\boldsymbol{u}} \in \mathbb{R}^{dT\times d(T-1)}$ and $\boldsymbol{u} \in \mathbb{R}^{d(T-1)}$.

Transfer matrices for single integrator

A single integrator is simply defined as $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{u}_t\Delta t$, corresponding to $\boldsymbol{A}_t = \boldsymbol{I}$ and $\boldsymbol{B}_t = \boldsymbol{I}\Delta t$, $\forall t \in \{1, \dots, T\}$, and transfer matrices $\boldsymbol{S}_{\boldsymbol{x}} = \boldsymbol{1}_T \otimes \boldsymbol{I}_D$, and $\boldsymbol{S}_{\boldsymbol{u}} = \begin{bmatrix}\boldsymbol{0}_{D,D(T-1)}\\ \boldsymbol{L}_{T-1,T-1}\otimes \boldsymbol{I}_D\Delta t\end{bmatrix}$, where $\boldsymbol{L}$ is a lower triangular matrix and $\otimes$ is the Kronecker product operator.

B Derivation of motion equation for a planar robot

We derive each element in (85) individually for $j \geq z$, then combine them altogether to derive the dynamic equation of the system. In this regard, for the first element in (85), we can write

$$\frac{\partial T_j}{\partial \dot{q}_z} = m_j\Big(\frac{\partial \dot{f}_{j,1}}{\partial \dot{q}_z}\dot{f}_{j,1} + \frac{\partial \dot{f}_{j,2}}{\partial \dot{q}_z}\dot{f}_{j,2}\Big) = m_j\Big(\Big(-l_z s_z\Big)\dot{f}_{j,1} + \Big(l_z c_z\Big)\dot{f}_{j,2}\Big),$$

whose time derivative can be expressed as

$$\frac{d}{dt}\frac{\partial T_j}{\partial \dot{q}_z} = m_j\Big(\Big(-l_z\dot{q}_z c_z\Big)\dot{f}_{j,1} - \Big(l_z s_z\Big)\ddot{f}_{j,1} - \Big(l_z\dot{q}_z s_z\Big)\dot{f}_{j,2} + \Big(l_z c_z\Big)\ddot{f}_{j,2}\Big),$$

where

$$\ddot{f}_{j,1} = -\sum_{k=1}^{j} l_k\ddot{q}_k s_k - \sum_{k=1}^{j} l_k\dot{q}_k^2 c_k, \quad \ddot{f}_{j,2} = \sum_{k=1}^{j} l_k\ddot{q}_k c_k - \sum_{k=1}^{j} l_k\dot{q}_k^2 s_k.$$

For the second term in (85), we can write

$$\frac{\partial T_j}{\partial q_z} = m_j\Big(\frac{\partial \dot{f}_{j,1}}{\partial q_z}\dot{f}_{j,1} + \frac{\partial \dot{f}_{j,2}}{\partial q_z}\dot{f}_{j,2}\Big) = m_j\Big(\Big(-l_z\dot{q}_z c_z\Big)\dot{f}_{j,1} - \Big(l_z\dot{q}_z s_z\Big)\dot{f}_{j,2}\Big),$$

and finally, the potential energy term can be calculated as

$$\frac{\partial U_j}{\partial q_z} = m_j g l_z c_z.$$

The z-*th* generalized force can be found by substituting the derived terms to (85) as

$$
\begin{aligned}
u_z &= \sum_{j=z}^{N}\Big(m_j\Big(\Big(-l_z\dot{q}_z c_z\Big)\dot{r}_{j,1} - \Big(l_z s_k\Big)\ddot{r}_{j,1} - \Big(l_z\dot{q}_z s_z\Big)\dot{r}_{j,2} + \Big(l_z c_z\Big)\ddot{r}_{j,2}\Big) - m_j\Big(\Big(-l_z\dot{q}_z c_z\Big)\dot{r}_{j,1} - \Big(l_z\dot{q}_z s_z\Big)\dot{r}_{j,2}\Big) + m_j g l_z c_z\Big)\\
&= \sum_{j=z}^{N}\Big(m_j\Big(\Big(-l_z s_z\Big)\ddot{r}_{j,1} + \Big(l_z c_z\Big)\ddot{r}_{j,2}\Big) + m_j g l_z c_z\Big)\\
&= \sum_{j=z}^{N}\Big(m_j\Big(\Big(-l_z s_z\Big)\Big(-\sum_{k=1}^{j} l_k\ddot{q}_k s_k - \sum_{k=1}^{j} l_k\dot{q}_k^2 c_k\Big) + \Big(l_z c_z\Big)\Big(\sum_{k=1}^{j} l_k\ddot{q}_k c_k - \sum_{k=1}^{j} l_k\dot{q}_k^2 s_k\Big)\Big) + m_j g l_z c_z\Big)\\
&= \sum_{j=z}^{N} m_j\Big(\sum_{k=1}^{j} l_z l_k c_{z-k}\ddot{q}_k + \sum_{k=1}^{j} l_z l_k s_{z-k}\dot{q}_k^2\Big) + \sum_{j=z}^{N} m_j g l_z c_z,
\end{aligned}
$$

where

$$c_{h-k} = c_h c_k - s_h s_k, \quad s_{h-k} = s_h c_k - c_h s_k.$$

By rearranging the order of elements, we can write

$$\sum_{j=z}^{N} m_j\Big(\sum_{k=1}^{j} l_z l_k c_{z-k}\ddot{q}_k\Big) = u_z - \sum_{j=z}^{N} m_j\Big(\sum_{k=1}^{j} l_z l_k s_{z-k}\dot{q}_k^2\Big) - \sum_{j=z}^{N} m_j g l_z c_z.$$

C Linear systems used in the bimanual tennis serve example

$\boldsymbol{x}$ represents the state trajectory of 3 agents: the left hand, the right hand and the ball. The nonzero elements correspond to the targets that the three agents must reach. For Agent 3 (the ball), $\boldsymbol{\mu}$ corresponds to a 2D position target at time T (ball target), with a corresponding 2D precision matrix (identity matrix) in $\boldsymbol{Q}$. For Agent 1 and 2 (the hands), $\boldsymbol{\mu}$ corresponds to 2D position targets active from time $\frac{3T}{4}$ to T (coming back to the initial pose and maintaining this pose), with corresponding 2D precision matrices (identity matrices) in $\boldsymbol{Q}$. The constraint that Agents 2 and 3 collide at time $\frac{T}{2}$ is ensured by setting $\left[\begin{smallmatrix} \boldsymbol{I} & -\boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{I} \end{smallmatrix}\right]$ in the entries of the precision matrix $\boldsymbol{Q}$ corresponding to the 2D positions of Agents 2 and 3 at $\frac{T}{2}$. With this formulation, the two positions are constrained to be the same, without having to predetermine the position at which the two agents should meet.[2]

The evolution of the system is described by the linear relation $\dot{\boldsymbol{x}}_t = \boldsymbol{A}_t^c\boldsymbol{x}_t + \boldsymbol{B}_t^c\boldsymbol{u}_t$, gathering the behavior of the 3 agents (left hand, right hand and ball) as double integrators with motions affected by gravity. For $t \leq \frac{T}{4}$ (left hand holding the ball), we have

$$
\underbrace{\left[\begin{array}{c}
\dot{\mathbf{x}}_{1,t}\\ \ddot{\mathbf{x}}_{1,t}\\ \dot{\boldsymbol{f}}_{1,t}\\ \hline
\dot{\mathbf{x}}_{2,t}\\ \ddot{\mathbf{x}}_{2,t}\\ \dot{\boldsymbol{f}}_{2,t}\\ \hline
\dot{\mathbf{x}}_{3,t}\\ \ddot{\mathbf{x}}_{3,t}\\ \dot{\boldsymbol{f}}_{3,t}
\end{array}\right]}_{\dot{\boldsymbol{x}}_t}
=
\underbrace{\left[\begin{array}{ccc|ccc|ccc}
\mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{array}\right]}_{\boldsymbol{A}_t^c}
\underbrace{\left[\begin{array}{c}
\mathbf{x}_{1,t}\\ \dot{\mathbf{x}}_{1,t}\\ \boldsymbol{f}_{1,t}\\ \hline
\mathbf{x}_{2,t}\\ \dot{\mathbf{x}}_{2,t}\\ \boldsymbol{f}_{2,t}\\ \hline
\mathbf{x}_{3,t}\\ \dot{\mathbf{x}}_{3,t}\\ \boldsymbol{f}_{3,t}
\end{array}\right]}_{\boldsymbol{x}_t}
+
\underbrace{\left[\begin{array}{c|c}
\mathbf{0} & \mathbf{0}\\
\boldsymbol{I} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \boldsymbol{I}\\
\mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}
\end{array}\right]}_{\boldsymbol{B}_t^c}
\underbrace{\left[\begin{array}{c}
\boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t}
\end{array}\right]}_{\boldsymbol{u}_t}. \tag{93}
$$

At $t = \frac{T}{2}$ (right hand hitting the ball), we have

$$
\underbrace{\left[\begin{array}{c}
\dot{\mathbf{x}}_{1,t}\\ \ddot{\mathbf{x}}_{1,t}\\ \dot{\boldsymbol{f}}_{1,t}\\ \hline
\dot{\mathbf{x}}_{2,t}\\ \ddot{\mathbf{x}}_{2,t}\\ \dot{\boldsymbol{f}}_{2,t}\\ \hline
\dot{\mathbf{x}}_{3,t}\\ \ddot{\mathbf{x}}_{3,t}\\ \dot{\boldsymbol{f}}_{3,t}
\end{array}\right]}_{\dot{\boldsymbol{x}}_t}
=
\underbrace{\left[\begin{array}{ccc|ccc|ccc}
\mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}\\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}
\end{array}\right]}_{\boldsymbol{A}_t^c}
\underbrace{\left[\begin{array}{c}
\mathbf{x}_{1,t}\\ \dot{\mathbf{x}}_{1,t}\\ \boldsymbol{f}_{1,t}\\ \hline
\mathbf{x}_{2,t}\\ \dot{\mathbf{x}}_{2,t}\\ \boldsymbol{f}_{2,t}\\ \hline
\mathbf{x}_{3,t}\\ \dot{\mathbf{x}}_{3,t}\\ \boldsymbol{f}_{3,t}
\end{array}\right]}_{\boldsymbol{x}_t}
+
\underbrace{\left[\begin{array}{c|c}
\mathbf{0} & \mathbf{0}\\
\boldsymbol{I} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \boldsymbol{I}\\
\mathbf{0} & \mathbf{0}\\ \hline
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}\\
\mathbf{0} & \mathbf{0}
\end{array}\right]}_{\boldsymbol{B}_t^c}
\underbrace{\left[\begin{array}{c}
\boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t}
\end{array}\right]}_{\boldsymbol{u}_t}. \tag{94}
$$

[2] Indeed, we can see that a cost $c = \left[\begin{smallmatrix} x_i \\ x_j \end{smallmatrix}\right]^\top \left[\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right] \left[\begin{smallmatrix} x_i \\ x_j \end{smallmatrix}\right] = (x_i - x_j)^2$ is minimized when $x_i = x_j$.

For $\frac{T}{4}<t<\frac{T}{2}$ and $t>\frac{T}{2}$ (free motion of the ball), we have

$$\underbrace{\left[\begin{array}{c}\dot{\mathbf{x}}_{1,t}\\ \ddot{\mathbf{x}}_{1,t}\\ \dot{\boldsymbol{f}}_{1,t}\\ \hline \dot{\mathbf{x}}_{2,t}\\ \ddot{\mathbf{x}}_{2,t}\\ \dot{\boldsymbol{f}}_{2,t}\\ \hline \dot{\mathbf{x}}_{3,t}\\ \ddot{\mathbf{x}}_{3,t}\\ \dot{\boldsymbol{f}}_{3,t}\end{array}\right]}_{\dot{\boldsymbol{x}}_t} = \underbrace{\left[\begin{array}{ccc|ccc|ccc}\mathbf{0}&\boldsymbol{I}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}&\boldsymbol{I}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \hline \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\boldsymbol{I}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\boldsymbol{I}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\\ \hline \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\boldsymbol{I}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\boldsymbol{I}\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{0}\end{array}\right]}_{\boldsymbol{A}_t^c} \underbrace{\left[\begin{array}{c}\mathbf{x}_{1,t}\\ \dot{\mathbf{x}}_{1,t}\\ \boldsymbol{f}_{1,t}\\ \hline \mathbf{x}_{2,t}\\ \dot{\mathbf{x}}_{2,t}\\ \boldsymbol{f}_{2,t}\\ \hline \mathbf{x}_{3,t}\\ \dot{\mathbf{x}}_{3,t}\\ \boldsymbol{f}_{3,t}\end{array}\right]}_{\boldsymbol{x}_t} + \underbrace{\left[\begin{array}{c|c}\mathbf{0}&\mathbf{0}\\ \boldsymbol{I}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}\\ \hline \mathbf{0}&\mathbf{0}\\ \mathbf{0}&\boldsymbol{I}\\ \mathbf{0}&\mathbf{0}\\ \hline \mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}\\ \mathbf{0}&\mathbf{0}\end{array}\right]}_{\boldsymbol{B}_t^c} \underbrace{\left[\begin{array}{c}\boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t}\end{array}\right]}_{\boldsymbol{u}_t}. \tag{95}$$

In the above, $\boldsymbol{f}_i = m_i\boldsymbol{g}$ represent the effect of gravity on the three agents, with mass m_i and acceleration vector $\boldsymbol{g} = \left[\begin{smallmatrix}0\\ -9.81\end{smallmatrix}\right]$ due to gravity. The parameters $\{\boldsymbol{A}_t^c, \boldsymbol{B}_t^c\}_{t=1}^T$, describing a continuous system, are first converted to their discrete forms with

$$\boldsymbol{A}_t = \boldsymbol{I} + \boldsymbol{A}_t^c\Delta t, \quad \boldsymbol{B}_t = \boldsymbol{B}_t^c\Delta t, \quad \forall t \in \{1,\dots,T\}, \tag{96}$$

which can then be used to define sparse transfer matrices $\boldsymbol{S}_{\boldsymbol{x}}$ and $\boldsymbol{S}_{\boldsymbol{u}}$ describing the evolution of the system in a vector form, starting from an initial state $\boldsymbol{x}_1$, namely $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u}$.

D Equivalence between LQT and LQR with augmented state space

The equivalence between the original LQT problem and the corresponding LQR problem with an augmented state space can be shown by using the block matrices composition rule

$$\boldsymbol{M} = \begin{bmatrix}\boldsymbol{M}_{11} & \boldsymbol{M}_{12}\\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22}\end{bmatrix} \iff \boldsymbol{M}^{-1} = \begin{bmatrix}\boldsymbol{I} & \mathbf{0}\\ -\boldsymbol{M}_{22}^{-1}\boldsymbol{M}_{21} & \boldsymbol{I}\end{bmatrix}\begin{bmatrix}\boldsymbol{S}^{-1} & \mathbf{0}\\ \mathbf{0} & \boldsymbol{M}_{22}^{-1}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & -\boldsymbol{M}_{12}\boldsymbol{M}_{22}^{-1}\\ \mathbf{0} & \boldsymbol{I}\end{bmatrix},$$

where $\boldsymbol{M}_{11}\in\mathbb{R}^{d\times d}$, $\boldsymbol{M}_{12}\in\mathbb{R}^{d\times D}$, $\boldsymbol{M}_{21}\in\mathbb{R}^{D\times d}$, $\boldsymbol{M}_{22}\in\mathbb{R}^{D\times D}$, and $\boldsymbol{S} = \boldsymbol{M}_{11} - \boldsymbol{M}_{12}\boldsymbol{M}_{22}^{-1}\boldsymbol{M}_{21}$ the Schur complement of the matrix $\boldsymbol{M}$.

In our case, we have

$$\begin{bmatrix}\boldsymbol{Q}^{-1}+\boldsymbol{\mu}\boldsymbol{\mu}^\top & \boldsymbol{\mu}\\ \boldsymbol{\mu}^\top & 1\end{bmatrix}^{-1} = \begin{bmatrix}\boldsymbol{I} & \mathbf{0}\\ -\boldsymbol{\mu}^\top & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{Q} & \mathbf{0}\\ \mathbf{0} & 1\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & -\boldsymbol{\mu}\\ \mathbf{0} & 1\end{bmatrix},$$

and it is easy to see that

$$(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{Q}(\boldsymbol{x}-\boldsymbol{\mu}) = \begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}^\top\begin{bmatrix}\boldsymbol{Q}^{-1}+\boldsymbol{\mu}\boldsymbol{\mu}^\top & \boldsymbol{\mu}\\ \boldsymbol{\mu}^\top & 1\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix} - 1.$$