A Math Cookbook for Robot Manipulation

Sylvain Calinon, Idiap Research Institute

Contents

1 Introduction

This cookbook presents learning and optimal control recipes for robotics (essentially for robot manipulators), complemented by simple toy problems that can be easily coded. It accompanies **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://rcfs.ch`

Each section in this document 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 Quadratic costs minimization as a product of Gaussians (PoG)

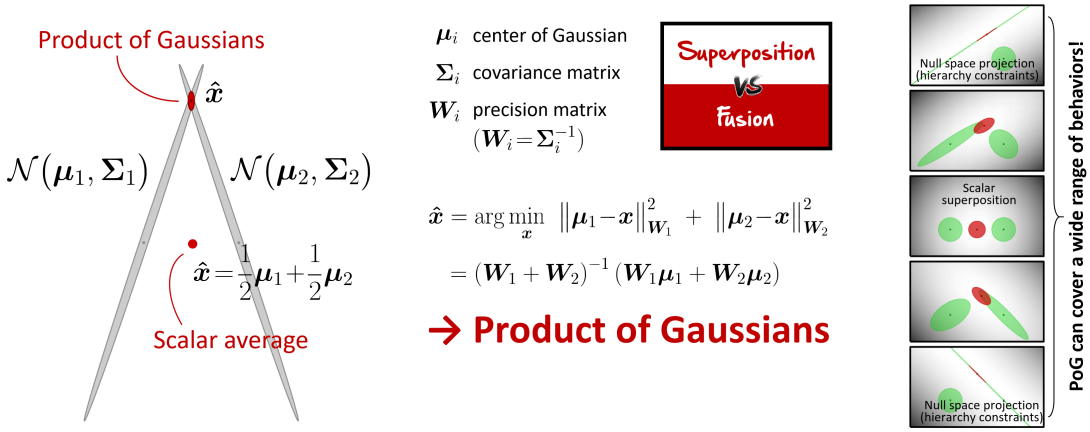


Figure 1: Quadratic costs minimization as a product of Gaussians (PoG).

The solution of a quadratic cost function can be viewed probabilistically as corresponding to a Gaussian distribution. Indeed, given a precision matrix $\boldsymbol{W}$, the quadratic cost

$$c(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{W} (\boldsymbol{x} - \boldsymbol{\mu}), \tag{1}$$
$$= \|\boldsymbol{x} - \boldsymbol{\mu}\|^2_{\boldsymbol{W}}, \tag{2}$$

has an optimal solution $\boldsymbol{x}^* = \boldsymbol{\mu}$. This solution does not contain much information about the cost function itself. Alternatively, we can view $\boldsymbol{x}$ as a random variable with a Gaussian distribution, i.e., $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma} = \boldsymbol{W}^{-1}$ are the mean vector and covariance matrix of the Gaussian, respectively. The negative log-likelihood of this Gaussian distribution is equivalent to (2) up to a constant factor. According to $p(\boldsymbol{x})$, $\boldsymbol{x}$ has the highest probability at $\bar{\boldsymbol{x}}$, and $\boldsymbol{\Sigma}$ gives the directional information on how this probability changes as we move away from $\boldsymbol{\mu}$. The point having the lowest cost in (2) is therefore associated with the point having the highest probability.

Similarly, the solution of a cost function composed of several quadratic terms

$$\hat{\boldsymbol{\mu}} = \arg\min_{\boldsymbol{x}} \sum_{k=1}^{K} (\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{W}_k (\boldsymbol{x} - \boldsymbol{\mu}_k) \tag{3}$$

can be seen as a product of Gaussians $\prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{W}_k^{-1})$, with centers $\boldsymbol{\mu}_k$ and covariance matrices $\boldsymbol{\Sigma}_k = \boldsymbol{W}_k^{-1}$. The Gaussian $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{W}}^{-1})$ resulting from this product has parameters

$$\hat{\boldsymbol{\mu}} = \left(\sum_{k=1}^{K} \boldsymbol{W}_k\right)^{-1} \left(\sum_{k=1}^{K} \boldsymbol{W}_k \boldsymbol{\mu}_k\right), \quad \hat{\boldsymbol{W}} = \sum_{k=1}^{K} \boldsymbol{W}_k.$$

$\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{W}}$ are the same as the solution of (3) and its Hessian, respectively. Viewing the quadratic cost probabilistically can capture more information about the cost function in the form of the covariance matrix $\hat{\boldsymbol{\Sigma}} = \hat{\boldsymbol{W}}^{-1}$.

There are also computation alternatives in case of Gaussians close to singularity. To be numerically more robust when computing the product of two Gaussians $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, the Gaussian $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$ resulting from the product can indeed be computed with

$$\hat{\boldsymbol{\mu}} = \boldsymbol{\Sigma}_2(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_2, \quad \hat{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\Sigma}_2,$$

by exploiting the fact that $\boldsymbol{\Sigma}_1\boldsymbol{\Sigma}_1^{-1}=\boldsymbol{I}$ and $\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_2=\boldsymbol{I}$, we can observe that

$$\begin{aligned}\boldsymbol{W}_1+\boldsymbol{W}_2 &= \boldsymbol{\Sigma}_2^{-1}+\boldsymbol{\Sigma}_1^{-1}\\ &= \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\boldsymbol{\Sigma}_1^{-1}+\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_2\boldsymbol{\Sigma}_1^{-1}\\ &= \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\Sigma}_1+\boldsymbol{\Sigma}_2)\boldsymbol{\Sigma}_1^{-1},\end{aligned}$$

so that

$$\begin{aligned}\hat{\boldsymbol{\Sigma}} &= (\boldsymbol{W}_1+\boldsymbol{W}_2)^{-1} = \boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1+\boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\Sigma}_2,\\ \hat{\boldsymbol{\mu}} &= (\boldsymbol{W}_1+\boldsymbol{W}_2)^{-1}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1+\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2) = \boldsymbol{\Sigma}_2(\boldsymbol{\Sigma}_1+\boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_1+\boldsymbol{\Sigma}_1(\boldsymbol{\Sigma}_1+\boldsymbol{\Sigma}_2)^{-1}\boldsymbol{\mu}_2.\end{aligned}$$

Figure 1 shows an illustration for 2 Gaussians in a 2-dimensional space. It also shows that when one of the Gaussians is singular, the product corresponds to a projection operation, that we for example find in nullspace projections to solve prioritized tasks in robotics.

3 Cost function minimization problems

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 2. 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.

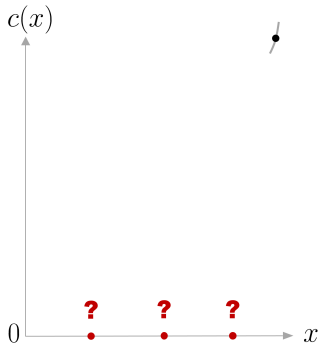


Figure 2: Problem formulation.

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 **first-order Taylor approximations** (gradient descent) or **second-order Taylor approximations** (Newton's method) of $c(x)$ around the iterates.

3.1 Gradient descent

Algorithm 1: Backtracking line search method with parameter $\alpha_{\min}$ (presented here for decision variable $\boldsymbol{x}$)

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

Figure 3 shows how consecutive **first-order Taylor approximations** of $c(x)$ around the iterates allow to solve a minimization problem.

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

$$c(x_k + \Delta x_k) \approx c(x_k) + c'(x_k)\, \Delta x_k,$$

where $c'(x_k)$ is the derivative of $c(x_k)$ at point x_k.

By starting from a point x_k at each step k, we are interested in applying a correction Δx_k that would decrease the cost $c(x_k)$. If the cost function follows a linear trend at point x_k with a slope defined by its gradient $c'(x_k)$, one direction would increase the cost while the other would decrease it. Thus, by applying a correction $\Delta x_k = -\alpha c'(x_k)$, where α is a positive scaling factor, we go down the slope estimated at x_k.

The scaling factor α can be either constant or variable. If α is too large, there is the risk that our local linear approximation is not valid anymore when we move far away from x_k. If it is too small, the iterative algorithm will require many iteration steps to converge to a local minimum of the cost function.

In practice, a simple backtracking line search procedure can be considered with Algorithm 1, by considering a small value for $\alpha_{\min}$, see Figure 4. For more elaborated methods, see Ch. 3 of [13].

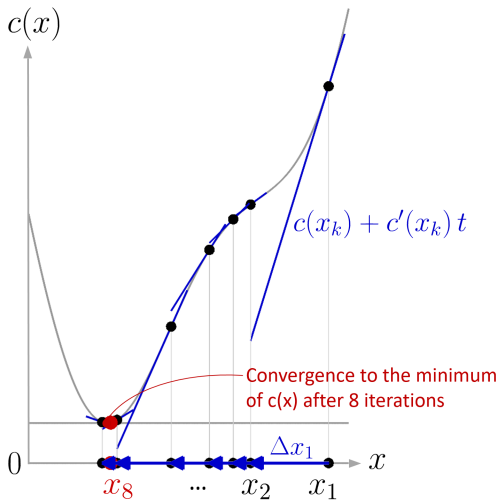


Figure 3: Gradient descent for minimization, starting from an initial estimate x_1 and converging to a local minimum (red point) after 8 iterations.

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 first-order Taylor expansion around the point $\boldsymbol{x}_k$ with

$$c(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k) \approx c(\boldsymbol{x}_k) + \boldsymbol{g}(\boldsymbol{x}_k)^\top \Delta\boldsymbol{x}_k,$$

with gradient vector

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

By starting from an initial estimate $\boldsymbol{x}_1$ and by recursively refining the current estimate by following the gradient, we obtain at each iteration k the recursion

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{g}(\boldsymbol{x}_k).$$

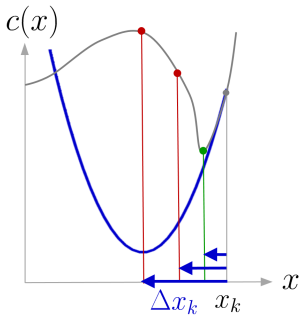


Figure 4: backtracking line search to scale the update vector Δx_k until the update decreases the cost. In this example, by starting with $\alpha = 1$ and by iteratively dividing α by two, the procedure provides a scaling factor $\alpha = 0.25$.

3.2 Newton's method

Figure 5 shows how consecutive **second-order Taylor approximations** of $c(x)$ around the iterates allow to solve a minimization problem.

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

$$c(x_k+\Delta x_k) \approx c(x_k) + c'(x_k)\,\Delta x_k + \frac{1}{2}c''(x_k)\,\Delta x_k^2, \tag{4}$$

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

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.

Indeed, to find the local optima of a function, we can localize the points whose derivatives are zero (horizontal slopes), see Figure 6 for an illustration.

By differentiating (4) w.r.t. Δx_k and equating to zero, we then 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.

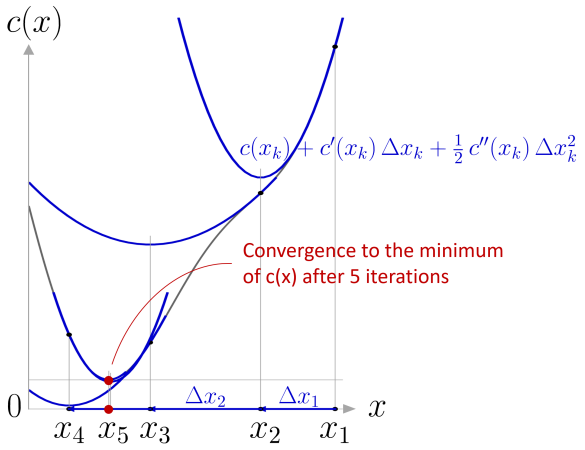


Figure 5: Newton's method for minimization, starting from an initial estimate x_1 and converging to a local minimum (red point) after 5 iterations.

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

$$x_{k+1} = x_k - \frac{c'(x_k)}{c''(x_k)}. \tag{5}$$

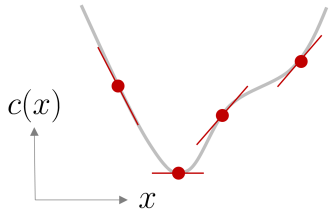


Figure 6: Finding local optima.

It is important that at each iteration, $c''(x_k)$ is positive, as we want to find local minima, see Figure 7 for an illustration. If $c''(x_k)$ is negative or too close to 0, it is in practice replaced by as small positive value in (5).

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.

Similarly as for gradient descent, in practice, a simple backtracking line search procedure can also be considered with Algorithm 1.

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

$$c(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k) \approx c(\boldsymbol{x}_k) + \Delta\boldsymbol{x}_k^\top \frac{\partial c}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k} + \frac{1}{2}\Delta\boldsymbol{x}_k^\top \frac{\partial^2 c}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k} \Delta\boldsymbol{x}_k, \quad (6)$$

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

$$\boldsymbol{g}(\boldsymbol{x}_k) = \frac{\partial c}{\partial \boldsymbol{x}}\Big|_{\boldsymbol{x}_k}, \quad (7)$$

and Hessian matrix

$$\boldsymbol{H}(\boldsymbol{x}_k) = \frac{\partial^2 c}{\partial \boldsymbol{x}^2}\Big|_{\boldsymbol{x}_k}. \quad (8)$$

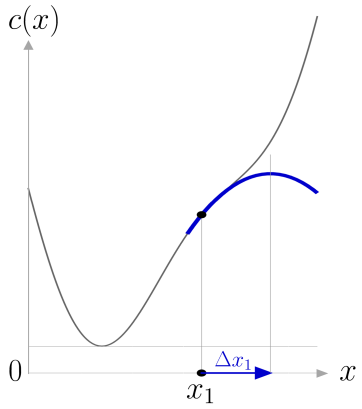


Figure 7: Newton update that would be achieved when the second derivative is negative.

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 8.

By differentiating (6) 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

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}(\boldsymbol{x}_k)^{-1}\,\boldsymbol{g}(\boldsymbol{x}_k). \quad (9)$$

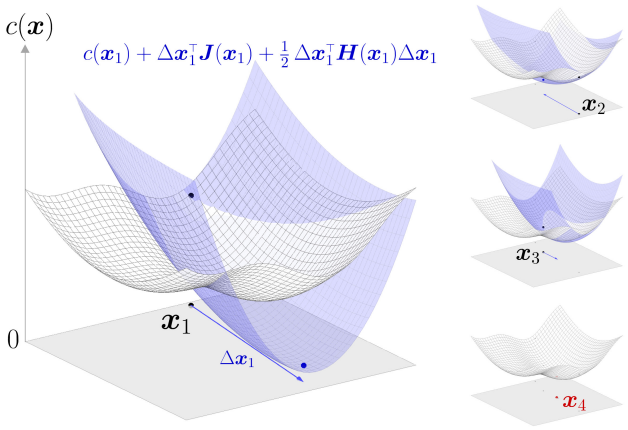


Figure 8: Newton's method for minimization with 2D decision variables.

In practice, we need to verify if the Hessian matrix $\boldsymbol{H}(\boldsymbol{x}_k)$ is positive definite at each iteration step.

Figure 9 shows the iteration steps taken by gradient descent and Newton's method to solve two minimization problems. The minimization problem in the first row is a quadratic cost function, where Newton's method converges in one iteration, while gradient descent requires multiple oscillatory steps to reach the minimum.

3.3 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 the scalar cost $c(\boldsymbol{x}) = \|\boldsymbol{f}(\boldsymbol{x})\|^2 = \boldsymbol{f}(\boldsymbol{x})^\top\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{R} f_i^2(\boldsymbol{x})$, where $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^R$ is a residual vector. By neglecting the second-order derivative terms, the gradient and Hessian can be computed with

$$\boldsymbol{g}(\boldsymbol{x}) = 2\boldsymbol{J}(\boldsymbol{x})^\top\boldsymbol{f}(\boldsymbol{x}), \quad \boldsymbol{H}(\boldsymbol{x}) \approx 2\boldsymbol{J}(\boldsymbol{x})^\top\boldsymbol{J}(\boldsymbol{x}),$$

where $\boldsymbol{J}(\boldsymbol{x}) \in \mathbb{R}^{R\times D}$ is the Jacobian matrix of $\boldsymbol{f}(\boldsymbol{x})$. 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 (9) then becomes

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \left(\boldsymbol{J}^\top(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)\right)^{-1}\boldsymbol{J}^\top(\boldsymbol{x}_k)\,\boldsymbol{f}(\boldsymbol{x}_k)\\ &= \boldsymbol{x}_k - \boldsymbol{J}^\dagger(\boldsymbol{x}_k)\,\boldsymbol{f}(\boldsymbol{x}_k),\end{aligned} \quad (10)$$

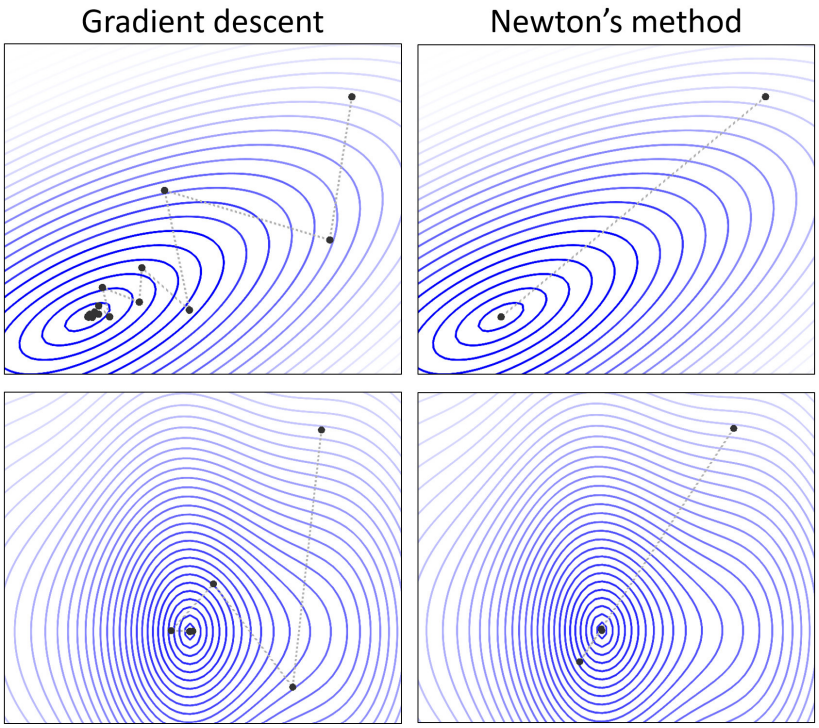


Figure 9: Convergence of gradient descent and Newton's method to solve minimization problems.

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

For comparison, the corresponding gradient descent problem would be computed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{J}^\top(\boldsymbol{x}_k)\,\boldsymbol{f}(\boldsymbol{x}_k).$$

Thus, for costs that can be expressed as $c(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{f}(\boldsymbol{x})$, the difference between first order or second order optimization is that the latter transforms the gradient of the former by the inverse of the Hessian matrix $\boldsymbol{H}(\boldsymbol{x}_k) = \boldsymbol{J}^\top(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)$. Another way to described this difference is to explain the former as an approximation of the latter with an Hessian matrix defined as identity (i.e., $\boldsymbol{H} = \boldsymbol{I}$).

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

3.4 Least squares

We show in this section the direct link between Gauss–Newton algorithm and least squares. Extensions of least square are also presented here, which can also directly be used for optimization problems.

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

$$c(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}), \tag{11}$$

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

$$\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0} \iff \boldsymbol{x} = (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{b} = \boldsymbol{A}^\dagger\boldsymbol{b}, \tag{12}$$

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.

In contrast to the Gauss–Newton algorithm presented in Section 3.3, the optimization problem in (11), described by a quadratic cost on the decision variable $\boldsymbol{x}$, admits the solution (12) that can be computed directly without relying on an iterative procedure. We can observe that if we follow the Gauss–Newton procedure, the residual vector corresponding to the cost (11) is $\boldsymbol{f} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ and its Jacobian matrix is $\boldsymbol{J} = \boldsymbol{A}$. By starting from an initial estimate $\boldsymbol{x}_0$, the Gauss–Newton update in (10) then takes the form

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \boldsymbol{A}^\dagger(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}) \\
&= \boldsymbol{x}_k - (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}) \\
&= \boldsymbol{x}_k - \cancel{(\boldsymbol{A}^\top\boldsymbol{A})^{-1}}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})}\boldsymbol{x}_k + \boldsymbol{A}^\dagger\boldsymbol{b} \\
&= \boldsymbol{A}^\dagger\boldsymbol{b},
\end{aligned}$$

which converges in a single iteration, independently of the initial guess $\boldsymbol{x}_0$. Indeed, a cost that takes a quadratic form with respect to the decision variable can be soled in batch form (least squares solution), which will be a useful property that we will exploit later in the context of linear quadratic controllers.

A **ridge regression** problem can similarly be defined as

$$\begin{aligned}\hat{\boldsymbol{x}} &= \arg\min_{\boldsymbol{x}} \ \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2 + \alpha\|\boldsymbol{x}\|^2 \\ &= (\boldsymbol{A}^\top\boldsymbol{A}+\alpha\boldsymbol{I})^{-1}\boldsymbol{A}^\top\boldsymbol{b},\end{aligned} \tag{13}$$

which is also called **penalized least squares**, **robust regression**, **damped least squares** or **Tikhonov regularization**. In (13), $\boldsymbol{I}$ denotes an identity matrix (diagonal matrix with 1 as elements in the diagonal). α is typically a small scalar value acting as a regularization term when inverting the matrix $\boldsymbol{A}^\top\boldsymbol{A}$.

The cost can also be weighted by a matrix $\boldsymbol{W}$, providing the **weighted least squares** solution

$$\begin{aligned}\hat{\boldsymbol{x}} &= \arg\min_{\boldsymbol{x}} \ \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2_{\boldsymbol{W}} \\ &= \arg\min_{\boldsymbol{x}} \ (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^\top\boldsymbol{W}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \\ &= (\boldsymbol{A}^\top\boldsymbol{W}\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{W}\boldsymbol{b}.\end{aligned} \tag{14}$$

By combining (13) and (14), a **weighted ridge regression** problem can be defined as

$$\begin{aligned}\hat{\boldsymbol{x}} &= \arg\min_{\boldsymbol{x}} \ \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2_{\boldsymbol{W}} + \alpha\|\boldsymbol{x}\|^2 \\ &= (\boldsymbol{A}^\top\boldsymbol{W}\boldsymbol{A}+\alpha\boldsymbol{I})^{-1}\boldsymbol{A}^\top\boldsymbol{W}\boldsymbol{b}.\end{aligned} \tag{15}$$

If the matrices and vectors in (15) have the structure

$$\boldsymbol{W} = \begin{bmatrix}\boldsymbol{W}_1 & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{W}_2\end{bmatrix}, \quad \boldsymbol{A} = \begin{bmatrix}\boldsymbol{A}_1\\ \boldsymbol{A}_2\end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{bmatrix},$$

we can consider an optimization problem using only the first part of the matrices, yielding the estimate

$$\begin{aligned}\hat{\boldsymbol{x}} &= \arg\min_{\boldsymbol{x}} \ \|\boldsymbol{A}_1\boldsymbol{x}-\boldsymbol{b}_1\|^2_{\boldsymbol{W}_1} + \alpha\|\boldsymbol{x}\|^2 \\ &= (\boldsymbol{A}_1^\top\boldsymbol{W}_1\boldsymbol{A}_1+\alpha\boldsymbol{I})^{-1}\boldsymbol{A}_1^\top\boldsymbol{W}_1\boldsymbol{b}_1.\end{aligned} \tag{16}$$

which is the same as the result of computing (15) with $\boldsymbol{W}_2 = \boldsymbol{0}$.

3.5 Least squares with constraints

A constrained minimization problem of the form

$$\min_{\boldsymbol{x}} \ (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})^\top(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}), \quad \text{s.t.} \quad \boldsymbol{C}\boldsymbol{x} = \boldsymbol{h}, \tag{17}$$

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 (17).

4 Forward kinematics (FK) for a planar robot manipulator

IK_manipulator.*

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

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

with $\boldsymbol{x}$ the state of the robot (joint angles), $\boldsymbol{f}^{\text{ee}}$ the position of the robot endeffector, $\boldsymbol{\ell}$ 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. 10.

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

$$\begin{aligned}\tilde{\boldsymbol{f}}^{\text{ee}} &= \Big[\boldsymbol{L}\ \text{diag}(\boldsymbol{\ell})\cos(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\ \text{diag}(\boldsymbol{\ell})\sin(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\boldsymbol{x}\Big]^\top \\ &= \begin{bmatrix}\tilde{f}^{\text{ee}}_{1,1} & \tilde{f}^{\text{ee}}_{1,2} & \tilde{f}^{\text{ee}}_{1,3} & \dots \\ \tilde{f}^{\text{ee}}_{2,1} & \tilde{f}^{\text{ee}}_{2,2} & \tilde{f}^{\text{ee}}_{2,3} & \dots \\ \tilde{f}^{\text{ee}}_{3,1} & \tilde{f}^{\text{ee}}_{3,2} & \tilde{f}^{\text{ee}}_{3,3} & \dots\end{bmatrix},\end{aligned} \tag{18}$$

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

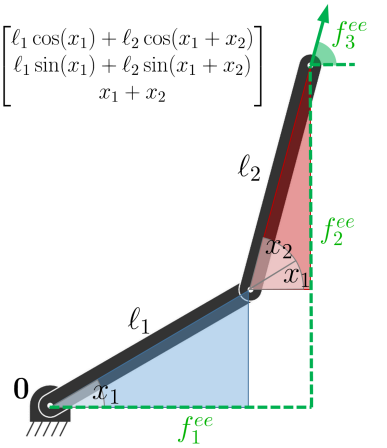


Figure 10: Forward kinematics for a planar robot with two links.

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

In Python, this can be coded for the endeffector 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
```

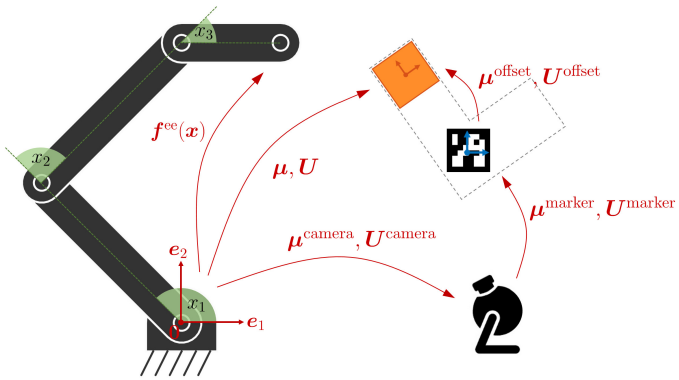


Figure 11: Typical transformations involved in a manipulation task involving a robot, a vision system, a visual marker on the object, and a desired grasping location on the object.

5 Inverse kinematics (IK) for a planar robot manipulator

`IK_manipulator.*`

We define a manipulation task involving a set of transformations as in Fig. 11. By relying on these transformation operators, we will describe all variables in the robot frame of reference (defined by $\mathbf{0}$, $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in the figure).

We first define $\boldsymbol{f}(\boldsymbol{x})$ as the residual vector between a target $\boldsymbol{\mu}$ and the endeffector position $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x})$ computed by the forward kinematics function, namely

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{\text{ee}}(\boldsymbol{x}) - \boldsymbol{\mu}.$$

The *inverse kinematics* (IK) problem consists of finding a robot pose to reach a target with the robot endeffector. The underlying optimization problem consists of minimizing a cost $c(\boldsymbol{x}) = \|\boldsymbol{f}(\boldsymbol{x})\|^2 = \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{f}(\boldsymbol{x})$, which can be solved iteratively with a Gauss–Newton method.

As seen in Section 3.3, by differentiating $c(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ and equating to 0, we get an update rule as in (9), namely

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \big(\boldsymbol{J}^\top(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)\big)^{-1} \boldsymbol{J}^\top(\boldsymbol{x}_k)\, \boldsymbol{f}(\boldsymbol{x}_k) \\ &= \boldsymbol{x}_k - \boldsymbol{J}^\dagger(\boldsymbol{x}_k)\, \big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_k) - \boldsymbol{\mu}\big), \end{aligned} \tag{19}$$

where $\boldsymbol{J} \in \mathbb{R}^{R \times D}$ is the Jacobian matrix of $\boldsymbol{f} \in \mathbb{R}^R$, and $\boldsymbol{J}^\dagger$ denotes the pseudoinverse of $\boldsymbol{J}$.

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

The approach can also be extended to target objects/landmarks with positions $\boldsymbol{\mu}$ and rotation matrices $\boldsymbol{U}$, as depicted in Fig. 11. We can then define an error between the robot endeffector and an object/landmark expressed in the object/landmark coordinate system as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}) &= \boldsymbol{U}^\top \big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}) - \boldsymbol{\mu}\big), \\ \boldsymbol{J}(\boldsymbol{x}) &= \boldsymbol{U}^\top \boldsymbol{J}^{\text{ee}}(\boldsymbol{x}). \end{aligned} \tag{20}$$

The inverse kinematics update in (19) can also be used to compute controllers.

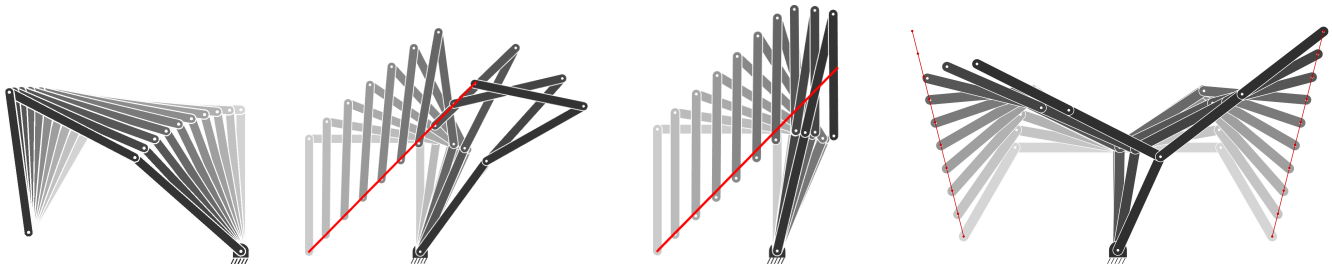
For a manipulator controlled by joint angle velocity commands $\boldsymbol{u} = \dot{\boldsymbol{x}} \approx \frac{\boldsymbol{\Delta x}}{\Delta t}$, we can for example define a controller

$$\boldsymbol{u} = \boldsymbol{J}^\dagger(\boldsymbol{x})\, (\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}) - \boldsymbol{\mu}) \frac{\alpha}{\Delta t}, \tag{21}$$

where α can typically be set as a small constant value, by taking into account the desired reactivity and the velocity capability of the robot.

For the planar robot manipulator example shown in Fig. 10, the Jacobian $\boldsymbol{J}(\boldsymbol{x})$ of the endeffector forward kinematics function $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x})$ can be computed as

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



(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 12: Examples of nullspace controllers. The red paths represent the trajectories of moving objects that need to be tracked by the robot endeffectors.

with

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

In Python, this can be coded for the endeffector 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 endeffector)
```

5.1 Numerical estimation of the Jacobian

IK_num.*

Section 5 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.

5.2 Inverse kinematics (IK) with task prioritization

IK_nullspace.*

In (19), the pseudoinverse provides a single least norm solution. This result can be generalized to obtain all solutions of the linear system with

$$\dot{\boldsymbol{x}} = \boldsymbol{J}^{\dagger}(\boldsymbol{x})\ \dot{\boldsymbol{f}} + \boldsymbol{N}(\boldsymbol{x})\ \boldsymbol{g}(\boldsymbol{x}), \tag{22}$$

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

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

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 12 presents examples of nullspace controllers. Figure 12a shows a robot controller keeping its endeffector still as primary objective, while trying to move the first joint as secondary objective, which is achieved with the controller

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

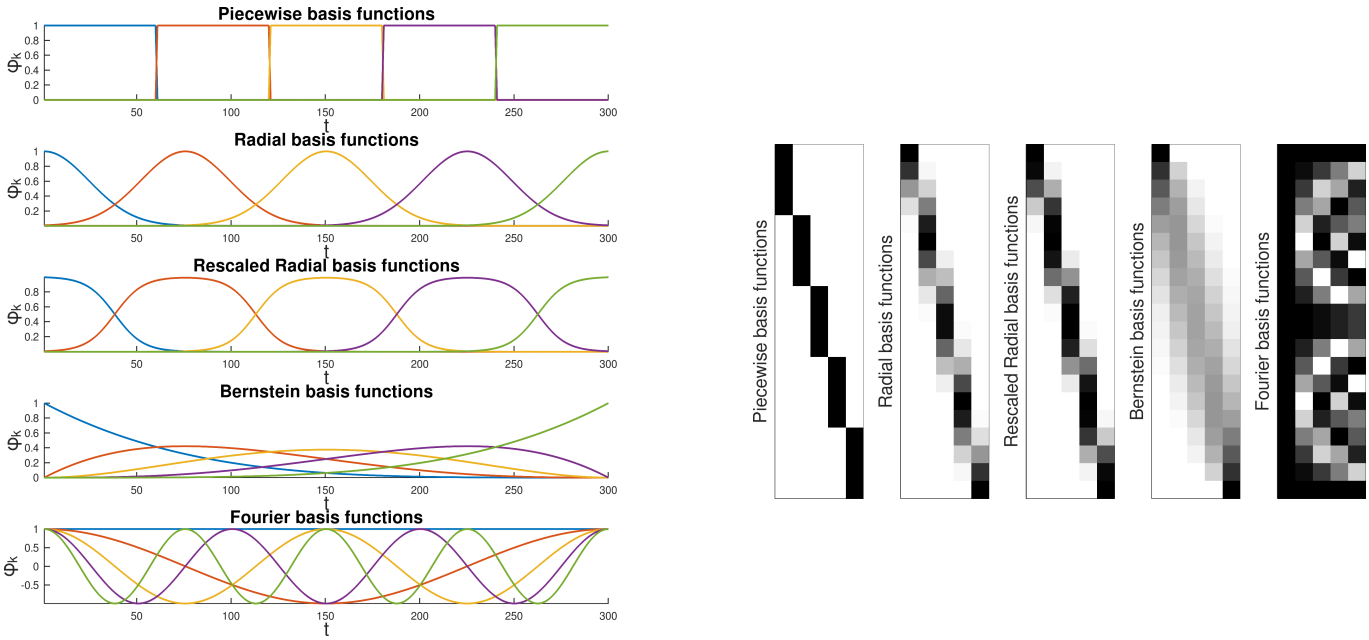


Figure 13: Examples of basis functions. *Left:* Representation in timeline form for $K = 5$ and $T = 300$. *Right:* Representation in matrix form for $K = 5$ and $T = 20$, with a grayscale colormap where white pixels are 0 and black pixels are 1.

6 Encoding with basis functions

MP.*

Basis functions can be used to encode signals in a compact manner through a weighted superposition of basis functions, acting as a dictionary of simpler signals that are superposed 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. 13. This principle, coined in the context of motor control [12], remains valid for a wide range of continuous time signals (for both analysis and synthesis).

The simpler form of *movement primitives* consists of encoding a movement as a weighted superposition of simpler movements. The compression aims at working in a subspace of reduced dimensionality, while denoising the signal and capturing the essential aspects of a movement.

6.1 Univariate trajectories

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

$$\boldsymbol{x}^{\text{1D}} = \sum_{k=1}^{K} \boldsymbol{\phi}_k \, w_k^{\text{1D}} = \boldsymbol{\phi}\, \boldsymbol{w}^{\text{1D}}, \tag{23}$$

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

$$\boldsymbol{\phi} = \boldsymbol{I}_K \otimes \mathbf{1}_{\frac{T}{K}}, \tag{24}$$

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. 13.

Radial basis functions (RBFs)

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

$$\boldsymbol{\phi} = \exp(-\lambda\, \boldsymbol{E} \odot \boldsymbol{E}), \quad \text{with} \quad \boldsymbol{E} = \boldsymbol{t}\, \mathbf{1}_K^\top - \mathbf{1}_T\, \boldsymbol{\mu}^{s\top}, \tag{25}$$

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. 13.

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

$$\boldsymbol{\phi}_k = \frac{(K-1)!}{(k-1)!(K-k)!}\, (\mathbf{1}_T - \boldsymbol{t})^{K-k} \;\odot\; \boldsymbol{t}^{k-1}, \tag{26}$$

$\forall k \in \{1, \dots, K\}$, 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. 13.

Fourier basis functions

Fourier basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \exp(\boldsymbol{t}\, \tilde{\boldsymbol{k}}^\top\, 2\pi i), \tag{27}$$

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

$$\boldsymbol{\phi} = \cos(\boldsymbol{t}\, \boldsymbol{k}^\top\, 2\pi i), \tag{28}$$

with $\boldsymbol{k} = [0, 1, \dots, K-2, K-1]^\top$, see Fig. 13.

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

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

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 (28).

6.2 Multidimensional outputs

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

$$\boldsymbol{x} = \sum_{k=1}^{K} \boldsymbol{\Psi}_k\, w_k = \boldsymbol{\Psi}\, \boldsymbol{w}, \quad \text{with} \quad \boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{I} = \begin{bmatrix} \boldsymbol{I}\phi_{1,1} & \boldsymbol{I}\phi_{2,1} & \cdots & \boldsymbol{I}\phi_{K,1} \\ \boldsymbol{I}\phi_{1,2} & \boldsymbol{I}\phi_{2,2} & \cdots & \boldsymbol{I}\phi_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{I}\phi_{1,T} & \boldsymbol{I}\phi_{2,T} & \cdots & \boldsymbol{I}\phi_{K,T} \end{bmatrix}, \tag{29}$$

where $\otimes$ the Kronecker product operator and $\boldsymbol{I}$ is an identity matrix of size $D \times D$.

In the above, $\boldsymbol{I}$ can alternatively be replaced by a rectangular matrix $\boldsymbol{S}$ acting as a coordination matrix.

6.3 Multidimensional inputs

`spline2d.*`

Basis functions can also be used to encode signals generated by multivariate inputs t_i. For example, a Bézier surface uses two input variables t_1 and t_2 to cover a spatial range and generates an output variable describing the height of the surface within this rectangular region. This surface (represented as a colormap in Fig. 14-*center*) can be constructed from a 1D input signal by leveraging the Kronecker product operation, namely

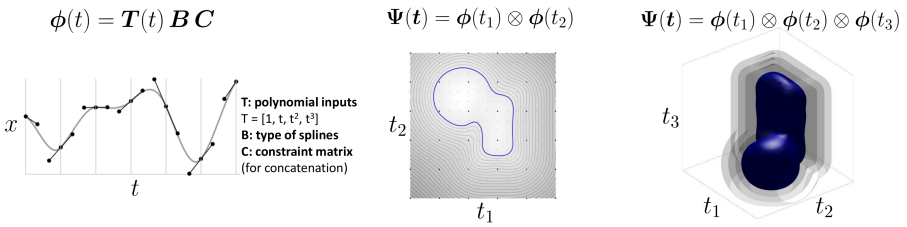


Figure 14: Concatenated Bernstein basis functions used with inputs of various dimensions to encode signed distance functions (SDFs). *Left:* Concatenation of 6 cubic Bézier curves with a 1D input (time variable t). *Center:* Concatenation of 5×5 cubic Bézier curves with a 2D input. *Right:* Concatenation of 5×5×5 cubic Bézier curves with a 3D input. Each cubic Bézier curve corresponds to the superposition of 4 Bernstein basis functions, with superposition weights acting as control points (displayed as black points in the 1D example), with constraints on the control points ensuring continuity.

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{\phi} \tag{30}$$

in matrix form, or

$$\boldsymbol{\Psi}(\boldsymbol{t}) = \boldsymbol{\phi}(t_1) \otimes \boldsymbol{\phi}(t_2) \tag{31}$$

in analytic form.

When using splines of the form $\boldsymbol{\phi}(t) = \boldsymbol{T}(t)\boldsymbol{B}\boldsymbol{C}$, (31) can equivalently be computed as

$$\boldsymbol{\Psi}(\boldsymbol{t}) = \big(\boldsymbol{T}(t_1) \otimes \boldsymbol{T}(t_2)\big)\,\big(\boldsymbol{B}\boldsymbol{C} \otimes \boldsymbol{B}\boldsymbol{C}\big), \tag{32}$$

which means that $\big(\boldsymbol{B}\boldsymbol{C} \otimes \boldsymbol{B}\boldsymbol{C}\big)$ can be precomputed.

As for the unidimensional version, we still maintain a linear reconstruction $\boldsymbol{x} = \boldsymbol{\Psi}\,\boldsymbol{w}$, 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 dimensions d^{in} and d^{out} can be considered, with

$$\boldsymbol{\Psi} = \underbrace{\boldsymbol{\phi} \otimes \boldsymbol{\phi} \otimes \cdots \otimes \boldsymbol{\phi}}_{d^{\text{in}}} \otimes \boldsymbol{I}_{d^{\text{out}}}. \tag{33}$$

For example, a vector field in 3D can be encoded with basis functions

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{\phi} \otimes \boldsymbol{\phi} \otimes \boldsymbol{I}_3, \tag{34}$$

where $\boldsymbol{I}_3$ is a 3×3 identity matrix to encode the 3 elements of the vector.

Figure 14 shows examples with various input dimensions to encode a signed distance field (SDF). In Fig. 14-*center*, several equidistant contours are displayed as closed paths, with the one corresponding to the object contour (distance zero) represented in blue. In Fig. 14-*right*, several isosurfaces are displayed as 3D shapes, with the one corresponding to the object surface (distance zero) represented in blue. A marching algorithm has been used here for visualization purpose to compute closed contours (in 2D) and isosurfaces (in 3D).

The analytic expression provided by the proposed encoding can be used to express the derivatives as analytic expressions, which is useful for control and planning problem, such as moving closer or away from objects, or orienting the robot gripper to be locally aligned with the surface of an object.

6.4 Derivatives

By using basis functions as analytic expressions, the derivatives are easy to compute. For example, for 2D inputs as in (31), we have

$$\frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_1} = \frac{\partial \boldsymbol{\phi}(t_1)}{\partial t_1} \otimes \boldsymbol{\phi}(t_2), \qquad \frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_2} = \boldsymbol{\phi}(t_1) \otimes \frac{\partial \boldsymbol{\phi}(t_2)}{\partial t_2}, \tag{35}$$

providing the derivatives of $\boldsymbol{\Psi}(\boldsymbol{t})$ with respect to $\boldsymbol{t}$ expressed as

$$\boldsymbol{\nabla}\boldsymbol{\Psi}(\boldsymbol{t}) = \frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_1} \otimes \frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_2}, \tag{36}$$

which can be used to compute the 2D gradient of the SDF at location $\boldsymbol{t}$ with

$$\boldsymbol{\nabla} x = \boldsymbol{\nabla}\boldsymbol{\Psi}(\boldsymbol{t})\,\boldsymbol{w}. \tag{37}$$

When using splines of the form $\boldsymbol{\phi}(t) = \boldsymbol{T}(t)\boldsymbol{B}\boldsymbol{C}$, the derivatives in (35) are simply computed as $\frac{\partial \boldsymbol{\phi}(t)}{\partial t} = \frac{\partial \boldsymbol{T}(t)}{\partial t}\boldsymbol{B}\boldsymbol{C}$. For a cubic splines, it corresponds to $\boldsymbol{T}(t) = [1, t, t^2, t^3]$ and $\frac{\partial \boldsymbol{T}(t)}{\partial t} = [0, 1, 2t, 3t^2]$.

Example: finding the closest point on a contour

By modeling a signed distance function as $x = f(\boldsymbol{t})$, the derivatives can for example be used to find a point on the contour (with distance zero). Such problem can be solved with Gauss–Newton optimization with the cost $c(\boldsymbol{t}) = \frac{1}{2} f(\boldsymbol{t})^2$, by starting from an initial guess $\boldsymbol{t}_0$. In the above example, the Jacobian is then given by

$$\boldsymbol{J}(\boldsymbol{t}) = \begin{bmatrix} \frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_1} \boldsymbol{w} \\ \frac{\partial \boldsymbol{\Psi}(\boldsymbol{t})}{\partial t_2} \boldsymbol{w} \\ \vdots \end{bmatrix}, \tag{38}$$

and the update can be computed recursively with

$$\boldsymbol{t}_{k+1} \leftarrow \boldsymbol{t}_k - \alpha \boldsymbol{J}(\boldsymbol{t}_k)^\dagger f(\boldsymbol{t}_k), \tag{39}$$

as in Equation (10), where $\boldsymbol{J}(\boldsymbol{t}_k)^\top f(\boldsymbol{t}_k)$ is a gradient and $\boldsymbol{J}(\boldsymbol{t}_k)^\top \boldsymbol{J}(\boldsymbol{t}_k)$ a Hessian matrix, and α is a line search parameter.

Example: finding two contour points with closest distance

For two shapes encoded as SDFs $f_1(\boldsymbol{t}_1)$ and $f_2(\boldsymbol{t}_2)$ using basis functions, finding the shortest segment between the two shapes boils down to the Gauss–Newton optimization of a pair of points $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are expressed in their respective shape coordinate frame and are organized as $\boldsymbol{t} = \begin{bmatrix} \boldsymbol{t}_1 \\ \boldsymbol{t}_2 \end{bmatrix}$, with the cost

$$c(\boldsymbol{t}) = \frac{1}{2} f_1(\boldsymbol{t}_1)^2 + \frac{1}{2} f_2(\boldsymbol{t}_2)^2 + \frac{1}{2} \left\| \boldsymbol{A}\, \boldsymbol{t}_2 + \boldsymbol{b} - \boldsymbol{t}_1 \right\|^2, \tag{40}$$

which is composed of quadratic residual terms, where $\boldsymbol{A}$ and $\boldsymbol{b}$ are the rotation matrix and translation vector offset between the two objects, respectively. The Jacobian of the residual terms in the above cost is given by

$$\boldsymbol{J}(\boldsymbol{t}) = \begin{bmatrix} \boldsymbol{J}_1(\boldsymbol{t}_1) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{J}_2(\boldsymbol{t}_2) \\ -\boldsymbol{I} & \boldsymbol{A} \end{bmatrix}, \tag{41}$$

where $\boldsymbol{J}_1(\boldsymbol{t}_1)$ and $\boldsymbol{J}_2(\boldsymbol{t}_2)$ are the derivatives of the two SDFs as given by (38).

Similarly to (39), the Gauss–Newton update step is then given by

$$\boldsymbol{t}_{k+1} \leftarrow \boldsymbol{t}_k - \alpha \boldsymbol{J}(\boldsymbol{t}_k)^\dagger \begin{bmatrix} f_1(\boldsymbol{t}_{1,k}) \\ f_2(\boldsymbol{t}_{2,k}) \\ \boldsymbol{A}\, \boldsymbol{t}_{2,k} + \boldsymbol{b} - \boldsymbol{t}_{1,k} \end{bmatrix}. \tag{42}$$

A prioritized optimization scheme can also be used to have the two points on the shape boundaries as primary objective and to move these points closer as secondary objective, with a corresponding Gauss–Newton update step given by

$$\boldsymbol{t}_{k+1} \leftarrow \boldsymbol{t}_k - \alpha_1 \boldsymbol{J}_{12}^\dagger \begin{bmatrix} f_1(\boldsymbol{t}_{1,k}) \\ f_2(\boldsymbol{t}_{2,k}) \end{bmatrix} + \alpha_2 \boldsymbol{N}_{12}\, \boldsymbol{J}_3^\dagger\, (\boldsymbol{A}\, \boldsymbol{t}_{2,k} + \boldsymbol{b} - \boldsymbol{t}_{1,k}), \tag{43}$$

$$\text{where} \quad \boldsymbol{J}_{12} = \begin{bmatrix} \boldsymbol{J}_1(\boldsymbol{t}_1) & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{J}_2(\boldsymbol{t}_2) \end{bmatrix}, \quad \boldsymbol{N}_{12} = \boldsymbol{I} - \boldsymbol{J}_{12}^\dagger \boldsymbol{J}_{12}, \quad \boldsymbol{J}_3 = \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{A} \end{bmatrix}, \tag{44}$$

with α_1 and α_2 are two line search parameters.

6.5 Concatenated basis functions

`spline1d.*` `spline2d.*`

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 constrain the last point of a curve and the first point of the next curve to maintain the continuity of the curve. We also typically constrain the control points before and after this joint to be symmetric, effectively imposing smoothness.

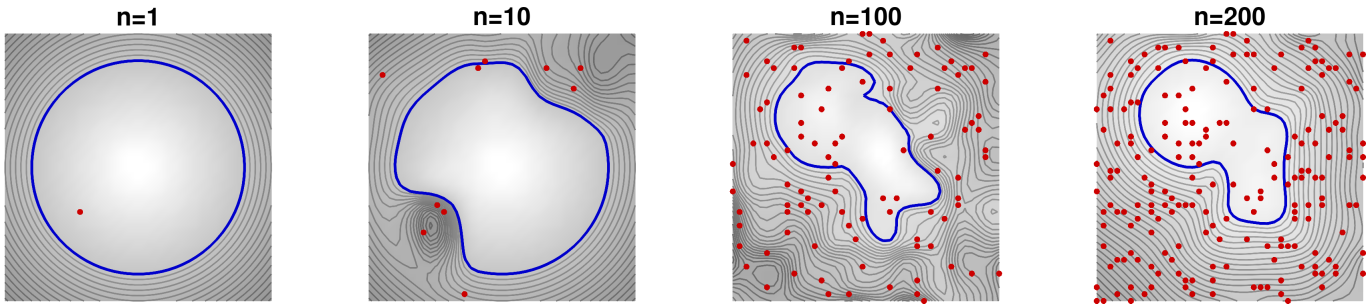


Figure 15: Iterative estimation of a 2D SDF, where the weights are initialized to form a circular object. Points are then sampled one-by-one (in red) to probe the value of the function as the sampled location. Each point used to refine the estimate and is then discarded.

In practice, this can be achieved efficiently by simply replacing $\boldsymbol{\phi}$ with $\boldsymbol{\phi C}$ in the above equations, where $\boldsymbol{C}$ 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 $\boldsymbol{C}$ as

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots \\ 0 & 1 & \cdots & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ 0 & 0 & \cdots & 1 & 0 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & 0 & 1 & \cdots \\ 0 & 0 & \cdots & -1 & 2 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{45}$$

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 cubic Bézier curves, each composed of 4 Bernstein basis functions, we can see locally that this operator yields a constraint of the form

$$\begin{bmatrix} w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}, \tag{46}$$

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, see Fig. 14-*left*.

6.6 Batch computation of basis functions coefficients

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

$$\hat{\boldsymbol{w}} = \boldsymbol{\Psi}^{\dagger}\boldsymbol{x} = (\boldsymbol{\Psi}^{\top}\boldsymbol{\Psi})^{-1}\boldsymbol{\Psi}^{\top}\boldsymbol{x}, \tag{47}$$

or as the regularized version (ridge regression)

$$\hat{\boldsymbol{w}} = (\boldsymbol{\Psi}^{\top}\boldsymbol{\Psi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Psi}^{\top}\boldsymbol{x}. \tag{48}$$

For example, if we want to fit a reference path, which can also be sparse or composed of a set of viapoints, while minimizing velocities (or similarly, any other derivatives, such as computing a minimum jerk trajectories), we can solve

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{\Psi w} - \boldsymbol{x}\|^2 + \frac{\lambda}{2}\|\boldsymbol{\nabla\Psi w}\|^2 \tag{49}$$

$$= (\boldsymbol{\Psi}^{\top}\boldsymbol{\Psi} + \lambda\boldsymbol{\nabla\Psi}^{\top}\boldsymbol{\nabla\Psi})^{-1}\boldsymbol{\Psi}^{\top}\boldsymbol{x}. \tag{50}$$

Algorithm 2: Recursive computation of weights

$\tilde{\boldsymbol{B}} = \frac{1}{\lambda}\boldsymbol{I}$ // Initialize $\tilde{B}$, corresponding to $(\Psi^\top\Psi)^{-1}$
$\boldsymbol{w} = \boldsymbol{w}_0$ // Initialize superposition weights (e.g., with prior shape w_0)
for $n \leftarrow 1$ *to* N **do**
  $\boldsymbol{\Psi}_n = \boldsymbol{\Psi}(\boldsymbol{t}_n)$ // Evaluate basis functions at location t_n, where the corresponding datapoint is x_n
  $\boldsymbol{K} = \tilde{\boldsymbol{B}}\boldsymbol{\Psi}_n^\top \left(\boldsymbol{I} + \boldsymbol{\Psi}_n\tilde{\boldsymbol{B}}\boldsymbol{\Psi}_n^\top\right)^{-1}$ // Compute Kalman gain K
  $\tilde{\boldsymbol{B}} \leftarrow \tilde{\boldsymbol{B}} - \boldsymbol{K}\boldsymbol{\Psi}_n\tilde{\boldsymbol{B}}$ // Update $\tilde{B}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{K}\Big(\boldsymbol{x}_n - \boldsymbol{\Psi}_n\boldsymbol{w}\Big)$ // Update superposition weights w
end

6.7 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 initial estimate of $\boldsymbol{w}$ that is iteratively refined with the arrival of new datapoints, see Fig. 15.

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

$$(\boldsymbol{B}+\boldsymbol{U}\boldsymbol{V})^{-1} = \boldsymbol{B}^{-1} - \overbrace{\boldsymbol{B}^{-1}\boldsymbol{U}\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{V}\boldsymbol{B}^{-1}}^{\boldsymbol{E}} \tag{51}$$

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 (48) when new datapoints become available. Indeed, if $\boldsymbol{\Psi}_{\text{new}} = \big[\boldsymbol{\Psi}, \boldsymbol{V}\big]$ and $\boldsymbol{x}_{\text{new}} = \begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{v}\end{bmatrix}$, we can see that

$$\begin{aligned}\boldsymbol{B}_{\text{new}} &= \boldsymbol{\Psi}_{\text{new}}^\top\boldsymbol{\Psi}_{\text{new}} &(52)\\ &= \boldsymbol{\Psi}^\top\boldsymbol{\Psi} + \boldsymbol{V}^\top\boldsymbol{V} &(53)\\ &= \boldsymbol{B} + \boldsymbol{V}^\top\boldsymbol{V}, &(54)\end{aligned}$$

whose inverse can be computed using (51), yielding

$$\boldsymbol{B}_{\text{new}}^{-1} = \boldsymbol{B}^{-1} - \underbrace{\boldsymbol{B}^{-1}\boldsymbol{V}^\top\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^\top\right)^{-1}}_{\boldsymbol{K}}\boldsymbol{V}\boldsymbol{B}^{-1}. \tag{55}$$

This is exploited to estimate the update as

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \boldsymbol{K}\Big(\boldsymbol{v} - \boldsymbol{V}\boldsymbol{w}\Big), \tag{56}$$

with Kalman gain

$$\boldsymbol{K} = \boldsymbol{B}^{-1}\boldsymbol{V}^\top\left(\boldsymbol{I}+\boldsymbol{V}\boldsymbol{B}^{-1}\boldsymbol{V}^\top\right)^{-1}. \tag{57}$$

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 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 (48) computed in batch form. Algorithm 2 summarizes the computation steps.

6.8 Computation of basis functions coefficients using eikonal cost

`spline2d_eikonal.*`

We can also estimate an SDF by privileging a unit norm on the derivatives. From a dataset of M distances x_m observed at locations $\boldsymbol{t}_m$, the basis functions coefficients $\boldsymbol{w}$ of an SDF can be computed by evaluating

$$\min_{\boldsymbol{w}} \sum_{m=1}^{M} \big\|\boldsymbol{\Psi}(\boldsymbol{t}_m)\,\boldsymbol{w} \;-\; x_m\big\|^2 + \lambda\sum_{n=1}^{N}\big\|\boldsymbol{\nabla}x_n^\top\,\boldsymbol{\nabla}x_n \;-\; 1\big\|^2, \tag{58}$$

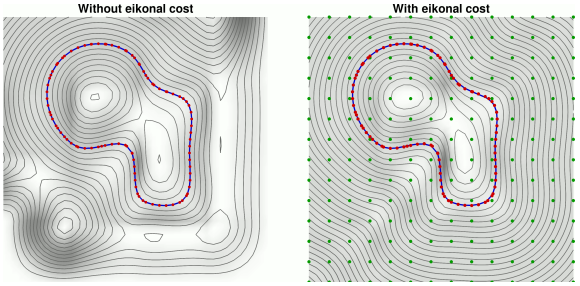


Figure 16: Estimation of an SDF from only contour points (in red), without (*left*) and with (*right*) an eikonal cost to privilege a

where λ is a weighting factor balancing the distance matching objective and the eikonal objective (regularization). The gradient of x_n with respect to $\boldsymbol{t}$ is expressed as $\boldsymbol{\nabla} x_n = \boldsymbol{\nabla}\boldsymbol{\Psi}(\boldsymbol{t}_n)\,\boldsymbol{w}$, computed using (36) for a set of N locations $\boldsymbol{t}_n$, that can typically be defined to cover homogeneously the area/volume encoded in the SDF.

The first component of the objective in (58) is to find a SDF that can fit the data. If only this first part would be required, the minimization boils down to the least squares solution of Section 6.6. The second term adds the objective of keeping the norm of the derivatives close to one, which is related to the eikonal equation $\|\boldsymbol{\nabla} x\| = 1$, see [4] for the use of the eikonal equation in the context of SDF modeling. Intuitively, as the basis functions encodes a SDF, it means that we expect that if we move away from a shape by a small distance Δd in the direction given by the gradient, the expected distance at this new point should also increase by Δd, which corresponds to a slope of 1 (namely, a unit norm for the gradient), as expressed in the second part of the objective function.

By following the notation in Section 3.3, the above objective can be written as a sum of squared functions that can be solved by Gauss–Newton optimization, with residuals and Jacobians given by

$$\begin{aligned} f_{1,m} &= \boldsymbol{\Psi}(\boldsymbol{t}_m)\,\boldsymbol{w} - x_m, & \boldsymbol{J}_{1,m} &= \boldsymbol{\Psi}(\boldsymbol{t}_m), & \forall m \in \{1,\dots,M\},\\ f_{2,n} &= \boldsymbol{\nabla} x_n^\top\,\boldsymbol{\nabla} x_n - 1, & \boldsymbol{J}_{2,n} &= 2\,\boldsymbol{\nabla} x_n{}^\top\,\boldsymbol{\nabla}\boldsymbol{\Psi}(\boldsymbol{t}_n), & \forall n \in \{1,\dots,N\}, \end{aligned}$$

which provide at each iteration step k the Gauss–Newton update rule

$$\boldsymbol{w}_{k+1} \;\leftarrow\; \boldsymbol{w}_k - \alpha\left(\sum_{m=1}^{M} \boldsymbol{J}_{1,m}^\top \boldsymbol{J}_{1,m} + \lambda \sum_{n=1}^{N} \boldsymbol{J}_{2,n}^\top \boldsymbol{J}_{2,n}\right)^{-1} \left(\sum_{m=1}^{M} \boldsymbol{J}_{1,m}^\top f_{1,m} + \lambda \sum_{n=1}^{N} \boldsymbol{J}_{2,n}^\top f_{2,n}\right),$$

as detailed in Section 3.3, where the learning rate α can be determined by line search.

Note that the above computation can be rewritten with concatenated vectors and matrices for more efficient computation exploiting linear algebra. By concatenating vertically the Jacobian matrices and residuals, we have the equivalent Gauss–Newton update rule

$$\boldsymbol{w}_{k+1} \;\leftarrow\; \boldsymbol{w}_k - \alpha\left(\boldsymbol{J}_1^\top \boldsymbol{J}_1 + \lambda \boldsymbol{J}_2^\top \boldsymbol{J}_2\right)^{-1} \left(\boldsymbol{J}_1^\top \boldsymbol{f}_1 + \lambda \boldsymbol{J}_2^\top \boldsymbol{f}_2\right).$$

Figure 16 presents an example in 2D.

7 Linear quadratic tracking (LQT)

`LQT.*` `LQT_nullspace.*`

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

$$\begin{aligned} c &= (\boldsymbol{\mu}_T - \boldsymbol{x}_T)^\top \boldsymbol{Q}_T (\boldsymbol{\mu}_T - \boldsymbol{x}_T) + \sum_{t=1}^{T-1} \Big((\boldsymbol{\mu}_t - \boldsymbol{x}_t)^\top \boldsymbol{Q}_t (\boldsymbol{\mu}_t - \boldsymbol{x}_t) \;+\; \boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t \Big) \\ &= (\boldsymbol{\mu} - \boldsymbol{x})^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x}) \;+\; \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \end{aligned} \tag{59}$$

with $\boldsymbol{x} = \begin{bmatrix}\boldsymbol{x}_1^\top, \boldsymbol{x}_2^\top, \dots, \boldsymbol{x}_T^\top\end{bmatrix}^\top$ the evolution of the state variables, $\boldsymbol{u} = \begin{bmatrix}\boldsymbol{u}_1^\top, \boldsymbol{u}_2^\top, \dots, \boldsymbol{u}_{T-1}^\top\end{bmatrix}^\top$ the evolution of the control commands, and $\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_1^\top, \boldsymbol{\mu}_2^\top, \dots, \boldsymbol{\mu}_T^\top\end{bmatrix}^\top$ 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

$$\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t, \tag{60}$$

yielding at trajectory level

$$\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}, \tag{61}$$

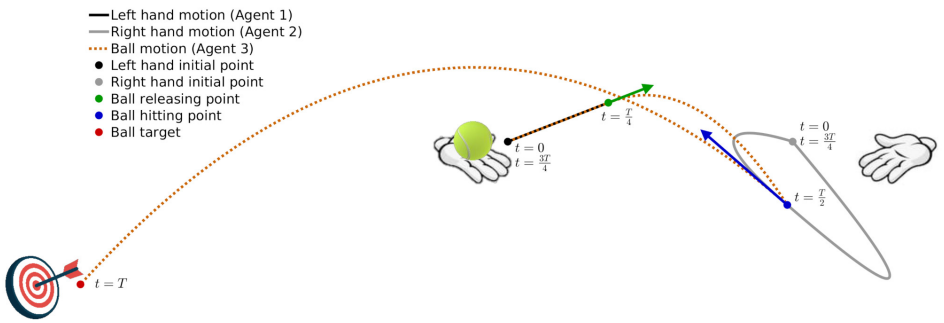


Figure 17: Tennis serve problem solved by linear quadratic tracking (LQT).

see Appendix A for details.

With open loop control commands organized as a vector $\boldsymbol{u}$, the solution of (59) subject to $\boldsymbol{x} = \boldsymbol{S_x}\boldsymbol{x}_1 + \boldsymbol{S_u}\boldsymbol{u}$ is analytic, given by

$$\hat{\boldsymbol{u}} = \left(\boldsymbol{S_u}^\top \boldsymbol{Q}\boldsymbol{S_u} + \boldsymbol{R}\right)^{-1} \boldsymbol{S_u}^\top \boldsymbol{Q}\left(\boldsymbol{\mu} - \boldsymbol{S_x}\boldsymbol{x}_1\right). \tag{62}$$

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)

$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{u}} = \left(\boldsymbol{S_u}^\top \boldsymbol{Q}\boldsymbol{S_u} + \boldsymbol{R}\right)^{-1}, \tag{63}$$

which provides a probabilistic interpretation of LQT, see Section 2.

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. 17. The problem is formulated as in (59), namely

$$\min_{\boldsymbol{u}} \left(\boldsymbol{\mu} - \boldsymbol{x}\right)^\top \boldsymbol{Q}\left(\boldsymbol{\mu} - \boldsymbol{x}\right) \; + \; \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u}, \quad \text{s.t.} \quad \boldsymbol{x} = \boldsymbol{S_x}\boldsymbol{x}_1 + \boldsymbol{S_u}\boldsymbol{u}, \tag{64}$$

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.

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 (64) can be solved by least squares, providing an analytical solution given by (62), see also Fig. 17 for the visualization of the result for a given target and for given initial poses of the hands.

7.1 LQT with initial state optimization

LQT can easily be extended to determine the optimal initial state $\boldsymbol{x}_1$ together with the optimal control commands $\boldsymbol{u}$, by defining $\tilde{\boldsymbol{u}} = [\boldsymbol{x}_1^\top, \boldsymbol{u}^\top]^\top$ as a vector concatenating $\boldsymbol{x}_1$ and $\boldsymbol{u}$ which is used as the variable to optimize. The linear system evolution $\boldsymbol{x} = \boldsymbol{S_x}\boldsymbol{x}_1 + \boldsymbol{S_u}\boldsymbol{u}$, as defined in (61), then takes the form $\boldsymbol{x} = \boldsymbol{S}\tilde{\boldsymbol{u}}$ with $\boldsymbol{S} = [\boldsymbol{S_x}, \boldsymbol{S_u}]$. Similarly, the optimization problem becomes

$$\begin{aligned} \hat{\tilde{\boldsymbol{u}}} = \begin{bmatrix} \hat{\boldsymbol{x}}_1 \\ \hat{\boldsymbol{u}} \end{bmatrix} &= \arg\min_{\tilde{\boldsymbol{u}}} \left(\boldsymbol{\mu} - \boldsymbol{S}\tilde{\boldsymbol{u}}\right)^\top \boldsymbol{Q}\left(\boldsymbol{\mu} - \boldsymbol{S}\tilde{\boldsymbol{u}}\right) + \tilde{\boldsymbol{u}}^\top \tilde{\boldsymbol{R}}\tilde{\boldsymbol{u}} \\ &= \left(\boldsymbol{S}^\top \boldsymbol{Q}\boldsymbol{S} + \tilde{\boldsymbol{R}}\right)^{-1} \boldsymbol{S}^\top \boldsymbol{Q}\boldsymbol{\mu}, \end{aligned} \tag{65}$$

by using $\tilde{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R} \end{bmatrix}$ to remove any constraint on the initial state. Thus, the above optimization problem estimates both the initial state and the sequence of control commands.

7.2 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 18 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 (59) with

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

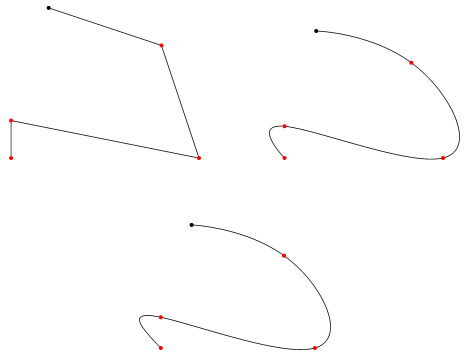


Figure 18: Examples of linear quadratic tracking (LQT) applied to the task of reaching a set of viapoints (red dots) by starting from an initial position (black dot). *Top-left:* With velocity commands, with a system described as a single integrator and a standard control cost. *Top-right:* With velocity 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.

7.3 LQT with control primitives

LQT_CP.*

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

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

with a solution given by

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

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 (63), 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 $\hat{\boldsymbol{x}} = \boldsymbol{S}_{\boldsymbol{x}}\boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}}\hat{\boldsymbol{u}}$. 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) [15] 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

$$\boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{C}, \tag{66}$$

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.

7.4 LQR with a recursive formulation

LQT_recursive.*

Algorithm 3: 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 (75) and (73)
end
// Forward pass
for $t \leftarrow 1$ *to* $T-1$ **do**
  Compute $\boldsymbol{K}_t$ and $\hat{\boldsymbol{u}}_t$ with (74)
  Compute $\boldsymbol{x}_{t+1}$ with (60)
end

The LQR problem is formulated as the minimization of the cost

$$c(\boldsymbol{x}_1, \boldsymbol{u}) = c_T(\boldsymbol{x}_T) + \sum_{t=1}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \ = \boldsymbol{x}_T^\top \boldsymbol{Q}_T \boldsymbol{x}_T + \sum_{t=1}^{T-1} \Big(\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t\Big), \quad \text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{u}_t, \tag{67}$$

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

$$v_t(\boldsymbol{x}_t, \boldsymbol{u}_{t:T-1}) = c_T(\boldsymbol{x}_T) + \sum_{s=t}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{68}$$

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

$$\hat{v}_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_{t:T-1}} v_t(\boldsymbol{x}_t, \boldsymbol{u}_{t:T-1}). \tag{69}$$

We will use the dynamic programming principle to reduce the minimization in (69) 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 (68) in (69), we observe that

$$\begin{aligned}
\hat{v}_t(\boldsymbol{x}_t) &= \min_{\boldsymbol{u}_{t:T-1}} \Big(c_T(\boldsymbol{x}_T) + \sum_{s=t}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big) \\
&= \min_{\boldsymbol{u}_t} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \min_{\boldsymbol{u}_{t+1:T-1}} \Big(c_T(\boldsymbol{x}_T) + \sum_{s=t+1}^{T-1} c_s(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big) \\
&= \min_{\boldsymbol{u}_t} \underbrace{c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \hat{v}_{t+1}(\boldsymbol{x}_{t+1})}_{q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)},
\end{aligned} \tag{70}$$

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

$$\hat{v}_T = \boldsymbol{x}_T^\top \boldsymbol{V}_T \boldsymbol{x}_T, \tag{71}$$

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 (70), the dynamic programming recursion takes the form

$$\hat{v}_t = \min_{\boldsymbol{u}_t} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R}_t \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \boldsymbol{x}_{t+1}^\top \boldsymbol{V}_{t+1} \boldsymbol{x}_{t+1}. \tag{72}$$

By substituting $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t$ into (72), $\hat{v}_t$ can be rewritten as

$$\hat{v}_t = \min_{\boldsymbol{u}_t} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{xx},t} & \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{Q}_{\boldsymbol{ux},t} & \boldsymbol{Q}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}, \quad \text{where} \quad \begin{cases} \boldsymbol{Q}_{\boldsymbol{xx},t} = \boldsymbol{A}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{A}_t + \boldsymbol{Q}_t, \\ \boldsymbol{Q}_{\boldsymbol{uu},t} = \boldsymbol{B}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{B}_t + \boldsymbol{R}_t, \\ \boldsymbol{Q}_{\boldsymbol{ux},t} = \boldsymbol{B}_t^\top \boldsymbol{V}_{t+1} \boldsymbol{A}_t. \end{cases} \tag{73}$$

An optimal control command $\hat{\boldsymbol{u}}_t$ can be computed by differentiating (73) with respect to $\boldsymbol{u}_t$ and equating to zero, providing a feedback law

$$\hat{\boldsymbol{u}}_t = -\boldsymbol{K}_t \, \boldsymbol{x}_t, \quad \text{with} \quad \boldsymbol{K}_t = \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \boldsymbol{Q}_{\boldsymbol{ux},t}. \tag{74}$$

By introducing (74) back into (73), the resulting value function $\hat{v}_t$ has the quadratic form

$$\hat{v}_t = \boldsymbol{x}_t^\top \boldsymbol{V}_t \boldsymbol{x}_t, \quad \text{with} \quad \boldsymbol{V}_t = \boldsymbol{Q}_{\boldsymbol{xx},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \boldsymbol{Q}_{\boldsymbol{ux},t}. \tag{75}$$

We observe that (75) has the same quadratic form as (71), so that (70) 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 3 for the summary of the overall procedure.

7.5 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

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{t+1} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_{t+1}} = \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{\tilde{\boldsymbol{A}}} \underbrace{\begin{bmatrix} \boldsymbol{x}_t \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_t} + \underbrace{\begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix}}_{\tilde{\boldsymbol{B}}} \boldsymbol{u}_t, \tag{76}$$

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

$$\begin{aligned} c &= \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big)^\top \boldsymbol{Q}_T \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big) + \sum_{t=1}^{T-1} \Big(\big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big)^\top \boldsymbol{Q}_t \big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big) \ + \ \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t \Big) \\ &= \tilde{\boldsymbol{x}}_T^\top \tilde{\boldsymbol{Q}}_T \tilde{\boldsymbol{x}}_T \ + \ \sum_{t=1}^{T-1} \Big(\tilde{\boldsymbol{x}}_t^\top \tilde{\boldsymbol{Q}}_t \tilde{\boldsymbol{x}}_t \ + \ \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t \Big), \end{aligned} \tag{77}$$

where the augmented form in (77) has the same form as the standard LQR cost in (67), 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

$$\hat{\boldsymbol{u}}_t = -\tilde{\boldsymbol{K}}_t \, \tilde{\boldsymbol{x}}_t = \boldsymbol{K}_t \, (\boldsymbol{\mu}_t - \boldsymbol{x}_t) + \boldsymbol{u}_t^{\text{ff}}, \tag{78}$$

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 $\boldsymbol{\mu}_t$.

7.6 Least squares formulation of recursive LQR

`LQT_recursive_LS.*`

We have seen in Section 7.4 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

$$\hat{\boldsymbol{u}} = -\big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1, \tag{79}$$

corresponding to open loop control commands.

We can see in (79) that the open loop control commands $\hat{\boldsymbol{u}}$ (in a vector form) is linear with respect to $\boldsymbol{x}_1$. Thus, 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

$$\boldsymbol{F} = \big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{x}}. \tag{80}$$

By decomposing $\boldsymbol{F}$ as block matrices $\boldsymbol{F}_t$ with $t \in \{1, \dots, T-1\}$, $\boldsymbol{F}$ can 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

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} (\boldsymbol{A}_{t-1} - \boldsymbol{B}_{t-1} \boldsymbol{K}_{t-1})^{-1}, \quad \boldsymbol{K}_t = \boldsymbol{F}_t \, \boldsymbol{P}_t, \tag{81}$$

which can then be used in a feedback controller as in (74).

It is straightforward to extend this least squares formulation of recursive LQR to linear quadratic tracking and use (78) 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 7.5).

This least squares formulation of LQT (LQT-LS) yields the same controller as with the standard recursive computation presented in Section 7.5. However, the linear form in (80) 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 7.7. 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 19 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.

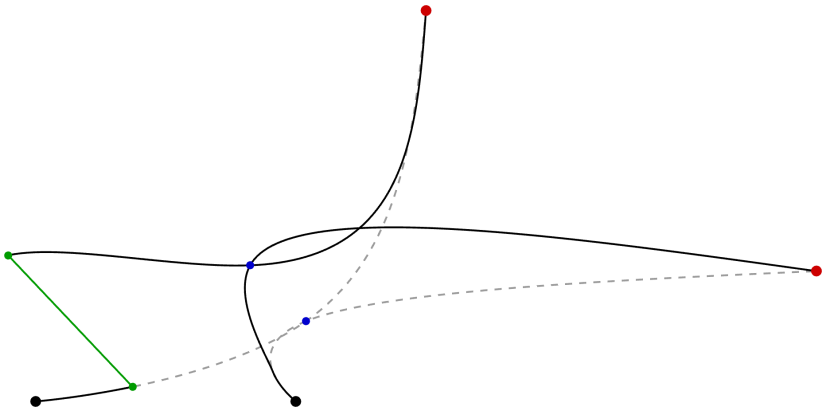


Figure 19: Least squares formulation of recursive LQR to control multiple agents (as point mass systems), where the task of each agent is to reach a desired target at the end of the motion, and to meet the other agent in the middle of the motion (e.g., for a handover task, see main text for the alternative 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.

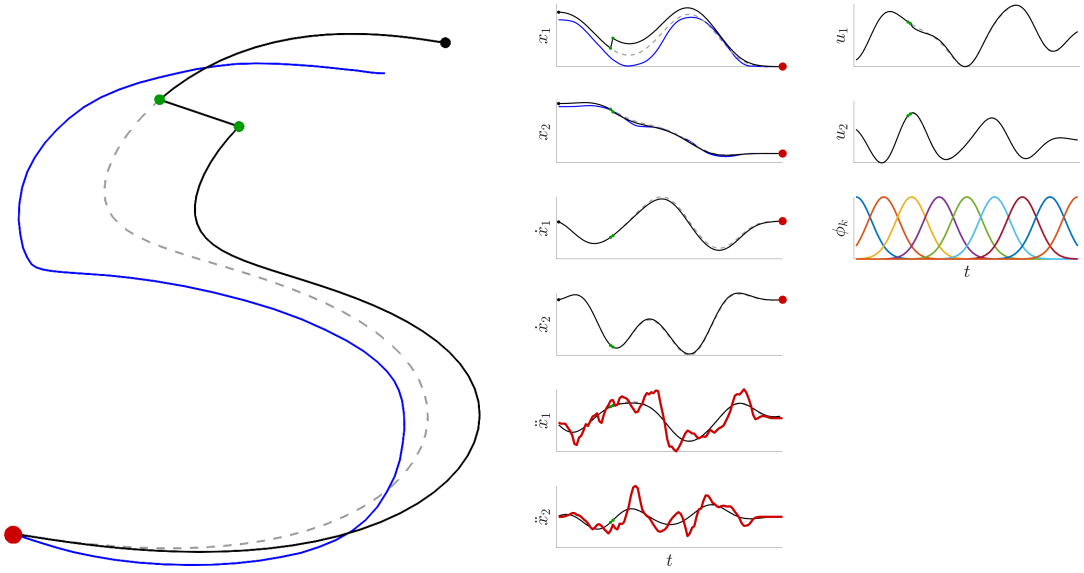


Figure 20: Linear quadratic tracking (LQT) with control primitives applied to a trajectory tracking task, with a formulation similar to dynamical movement primitives (DMP). *Left:* The observed "S" shape (in blue) is reproduced by starting from a different initial position (black point), with a perturbation 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 $\boldsymbol{\phi}_k$.

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

`LQT_CP_DMP.*`

The dynamical movement primitives (DMP) [5] 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 (62), or as the closed-loop controller (78).

In the latter case, the matrix $\boldsymbol{F}$ in (80) is estimated by using control primitives and an augmented state space formulation, namely

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

which is used to compute feedback gains $\tilde{\boldsymbol{K}}_t$ on the augmented state with (81).

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 20 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.

8 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) [8] 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 [11, 17].

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

$$\begin{aligned} & \boldsymbol{x}_{t+1} \approx \boldsymbol{d}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}_t}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{u}_t}(\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t) \\ \iff & \Delta \boldsymbol{x}_{t+1} \approx \boldsymbol{A}_t \Delta \boldsymbol{x}_t + \boldsymbol{B}_t \Delta \boldsymbol{u}_t, \end{aligned} \tag{82}$$

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

$$\begin{aligned} & c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \Delta \boldsymbol{x}_t^\top \frac{\partial c}{\partial \boldsymbol{x}_t} + \Delta \boldsymbol{u}_t^\top \frac{\partial c}{\partial \boldsymbol{u}_t} + \frac{1}{2} \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t^2} \Delta \boldsymbol{x}_t + \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t \boldsymbol{u}_t} \Delta \boldsymbol{u}_t + \frac{1}{2} \Delta \boldsymbol{u}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2} \Delta \boldsymbol{u}_t, \\ \iff & c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{1}{2} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x}_t \\ \Delta \boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x},t}^\top & \boldsymbol{g}_{\boldsymbol{u},t}^\top \\ \boldsymbol{g}_{\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t}^\top \\ \boldsymbol{g}_{\boldsymbol{u},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x}_t \\ \Delta \boldsymbol{u}_t \end{bmatrix}, \end{aligned} \tag{83}$$

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

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

8.1 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 3.3.

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 (82) 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

$$\min_{\Delta \boldsymbol{u}} \Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}), \quad \text{s.t.} \quad \Delta \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u}, \quad \text{where} \quad \Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}) = \frac{1}{2} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x} \\ \Delta \boldsymbol{u} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x}}^\top & \boldsymbol{g}_{\boldsymbol{u}}^\top \\ \boldsymbol{g}_{\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \\ \boldsymbol{g}_{\boldsymbol{u}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}} & \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta \boldsymbol{x} \\ \Delta \boldsymbol{u} \end{bmatrix}. \tag{84}$$

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

$$\min_{\Delta \boldsymbol{u}} \quad \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} + \Delta \boldsymbol{u}^\top \boldsymbol{g}_{\boldsymbol{u}} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u} + \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \Delta \boldsymbol{u} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \Delta \boldsymbol{u}, \tag{85}$$

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

$$\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{u}} + \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u} + 2 \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top \Delta \boldsymbol{u} + \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}} \Delta \boldsymbol{u} = 0, \tag{86}$$

providing the least squares solution

$$\Delta \hat{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} + 2 \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x}}^\top + \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u}}\right)^{-1} \left(-\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} - \boldsymbol{g}_{\boldsymbol{u}}\right), \tag{87}$$

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 4 (including backtracking line search). Figure 21 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 $\boldsymbol{x}_1$).

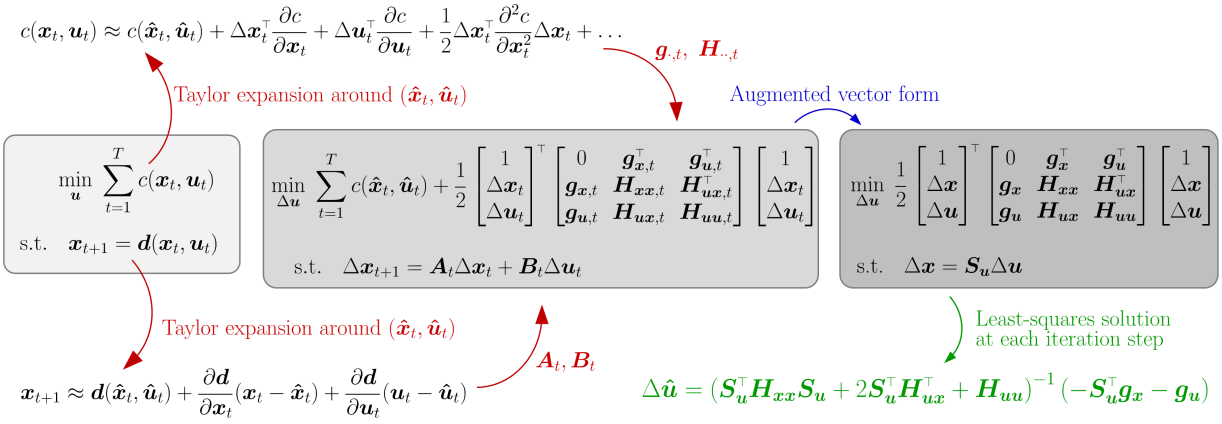


Figure 21: Summary of the iLQR optimization procedure in batch form.

8.2 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 7.4 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 (70), we have here

$$\hat{v}_t(\Delta\boldsymbol{x}_t) = \min_{\Delta\boldsymbol{u}_t} \underbrace{c_t(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t) + \hat{v}_{t+1}(\Delta\boldsymbol{x}_{t+1})}_{q_t(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t)}, \tag{88}$$

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

$$\hat{v}_{t+1} = \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{v}_{\boldsymbol{x},t+1}^\top \\ \boldsymbol{v}_{\boldsymbol{x},t+1} & \boldsymbol{V}_{\boldsymbol{xx},t+1} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}, \tag{89}$$

and we then have with (88) that

$$\hat{v}_t = \min_{\Delta\boldsymbol{u}_t} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{g}_{\boldsymbol{x},t}^\top & \boldsymbol{g}_{\boldsymbol{u},t}^\top \\ \boldsymbol{g}_{\boldsymbol{x},t} & \boldsymbol{H}_{\boldsymbol{xx},t} & \boldsymbol{H}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{g}_{\boldsymbol{u},t} & \boldsymbol{H}_{\boldsymbol{ux},t} & \boldsymbol{H}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix} + \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{v}_{\boldsymbol{x},t+1}^\top \\ \boldsymbol{v}_{\boldsymbol{x},t+1} & \boldsymbol{V}_{\boldsymbol{xx},t+1} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_{t+1} \end{bmatrix}. \tag{90}$$

By substituting $\Delta\boldsymbol{x}_{t+1} = \boldsymbol{A}_t\Delta\boldsymbol{x}_t + \boldsymbol{B}_t\Delta\boldsymbol{u}_t$ into (90), $\hat{v}_t$ can be rewritten as

$$\hat{v}_t = \min_{\Delta\boldsymbol{u}_t} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & \boldsymbol{q}_{\boldsymbol{x},t}^\top & \boldsymbol{q}_{\boldsymbol{u},t}^\top \\ \boldsymbol{q}_{\boldsymbol{x},t} & \boldsymbol{Q}_{\boldsymbol{xx},t} & \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \\ \boldsymbol{q}_{\boldsymbol{u},t} & \boldsymbol{Q}_{\boldsymbol{ux},t} & \boldsymbol{Q}_{\boldsymbol{uu},t} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\boldsymbol{x}_t \\ \Delta\boldsymbol{u}_t \end{bmatrix}, \quad \text{where} \quad \begin{cases} \boldsymbol{q}_{\boldsymbol{x},t} = \boldsymbol{g}_{\boldsymbol{x},t} + \boldsymbol{A}_t^\top \boldsymbol{v}_{\boldsymbol{x},t+1}, \\ \boldsymbol{q}_{\boldsymbol{u},t} = \boldsymbol{g}_{\boldsymbol{u},t} + \boldsymbol{B}_t^\top \boldsymbol{v}_{\boldsymbol{x},t+1}, \\ \boldsymbol{Q}_{\boldsymbol{xx},t} \approx \boldsymbol{H}_{\boldsymbol{xx},t} + \boldsymbol{A}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{A}_t, \\ \boldsymbol{Q}_{\boldsymbol{uu},t} \approx \boldsymbol{H}_{\boldsymbol{uu},t} + \boldsymbol{B}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{B}_t, \\ \boldsymbol{Q}_{\boldsymbol{ux},t} \approx \boldsymbol{H}_{\boldsymbol{ux},t} + \boldsymbol{B}_t^\top \boldsymbol{V}_{\boldsymbol{xx},t+1} \boldsymbol{A}_t, \end{cases} \tag{91}$$

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

$$\Delta \hat{\boldsymbol{u}}_t = \boldsymbol{k}_t + \boldsymbol{K}_t \, \Delta \boldsymbol{x}_t, \text{ with } \begin{cases} \boldsymbol{k}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \, \boldsymbol{q}_{\boldsymbol{u},t}, \\ \boldsymbol{K}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \, \boldsymbol{Q}_{\boldsymbol{ux},t}, \end{cases} \tag{92}$$

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

By inserting (92) into (91), we get the recursive updates

$$\begin{aligned} \boldsymbol{v}_{\boldsymbol{x},t} &= \boldsymbol{q}_{\boldsymbol{x},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \, \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \, \boldsymbol{q}_{\boldsymbol{u},t}, \\ \boldsymbol{V}_{\boldsymbol{xx},t} &= \boldsymbol{Q}_{\boldsymbol{xx},t} - \boldsymbol{Q}_{\boldsymbol{ux},t}^\top \, \boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \, \boldsymbol{Q}_{\boldsymbol{ux},t}. \end{aligned} \tag{93}$$

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 (92) 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

$$\boldsymbol{u}_t = \hat{\boldsymbol{u}}_t + \boldsymbol{K}_t \, (\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t), \tag{94}$$

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 5 (including backtracking line search).

8.3 Least squares formulation of recursive iLQR

First, note that (92) can be rewritten as

$$\Delta \hat{\boldsymbol{u}}_t = \tilde{\boldsymbol{K}}_t \begin{bmatrix} \Delta \boldsymbol{x}_t \\ 1 \end{bmatrix}, \text{ with } \quad \tilde{\boldsymbol{K}}_t = -\boldsymbol{Q}_{\boldsymbol{uu},t}^{-1} \Big[\boldsymbol{Q}_{\boldsymbol{ux},t}, \; \boldsymbol{q}_{\boldsymbol{u},t} \Big]. \tag{95}$$

Also, (85) 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 7.6, we set

$$\Delta \boldsymbol{u} = -\boldsymbol{F} \begin{bmatrix} \Delta \boldsymbol{x}_1 \\ 1 \end{bmatrix} = -\boldsymbol{F} \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix} = -\boldsymbol{F} \Delta \tilde{\boldsymbol{x}}_1, \tag{96}$$

and redefine the optimization problem as

$$\min_{\boldsymbol{F}} \quad \frac{1}{2} \Delta \tilde{\boldsymbol{x}}_1^\top \boldsymbol{F}^\top \boldsymbol{C} \boldsymbol{F} \Delta \tilde{\boldsymbol{x}}_1 - \Delta \tilde{\boldsymbol{x}}_1^\top \boldsymbol{F}^\top \boldsymbol{c}. \tag{97}$$

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

$$\boldsymbol{F} \Delta \tilde{\boldsymbol{x}}_1 = \boldsymbol{C}^{-1} \boldsymbol{c}. \tag{98}$$

Similarly to Section 7.6, 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

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} (\boldsymbol{A}_{t-1} - \boldsymbol{B}_{t-1} \boldsymbol{K}_{t-1})^{-1}, \quad \boldsymbol{K}_t = \boldsymbol{F}_t \, \boldsymbol{P}_t, \tag{99}$$

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

Algorithm 4: 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 (82)
  Compute $\boldsymbol{S}_{\boldsymbol{u}}$ with (188)
  Compute $\boldsymbol{g}_{\cdot}$ and $\boldsymbol{H}_{\cdot}$ in (84), using (83)
  Compute $\Delta\hat{\boldsymbol{u}}$ with (87)
  Compute α with Algorithm 1
  Update $\hat{\boldsymbol{u}}$ with (100)
until $\|\Delta\hat{\boldsymbol{u}}\| < \Delta_{\min}$;

Algorithm 5: 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 (82)
  Compute all $\boldsymbol{g}_{\cdot,t}$ and $\boldsymbol{H}_{\cdot,t}$ in (83)
  `// 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 (91)
    Compute $\boldsymbol{k}_t$ and $\boldsymbol{K}_t$ with (92)
    Compute $\boldsymbol{v}_{\boldsymbol{x},t}$ and $\boldsymbol{V}_{\boldsymbol{xx},t}$ with (93)
  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 (101)
      Compute $\hat{\boldsymbol{x}}_{t+1}$ with $\boldsymbol{d}(\cdot)$
    end
  end
until $\|\Delta\hat{\boldsymbol{u}}\| < \Delta_{\min}$;
Use (94) and $\boldsymbol{d}(\cdot)$ for reproduction

8.4 Updates by considering step sizes

iLQR_manipulator.*

iLQR typically requires to estimate a step size α at each iteration to scale the control command updates. For the batch formulation in Section 8.1, this can be achieved by setting the update (87) as

$$\hat{\boldsymbol{u}} \leftarrow \hat{\boldsymbol{u}} + \alpha\, \Delta\hat{\boldsymbol{u}}, \tag{100}$$

where the resulting procedure consists of estimating a descent direction with (87) along which the objective function will be reduced, and then estimating with (100) a step size that determines how far one can move along this direction.

For the recursive formulation in Section 8.2, this can be achieved by setting the update (92) as

$$\hat{\boldsymbol{u}}_t \leftarrow \hat{\boldsymbol{u}}_t + \alpha\, \boldsymbol{k}_t + \boldsymbol{K}_t\, \Delta\boldsymbol{x}_t. \tag{101}$$

In practice, a simple backtracking line search procedure can be considered, as shown in Algorithm 1, by considering a small value for $\alpha_{\min}$.

The complete iLQR procedures are described in Algorithms 4 and 5 for the batch and recursive formulations, respectively.

9 iLQR with quadratic cost on f(x)

iLQR_manipulator.*

We consider a cost defined by

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{f}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t) + \boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t, \tag{102}$$

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 (83) then becomes

$$\Delta c(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t) \approx 2\Delta\boldsymbol{x}_t^\top \boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t) + 2\Delta\boldsymbol{u}_t^\top \boldsymbol{R}_t\, \boldsymbol{u}_t + \Delta\boldsymbol{x}_t^\top \boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{J}(\boldsymbol{x}_t)\Delta\boldsymbol{x}_t + \Delta\boldsymbol{u}_t^\top \boldsymbol{R}_t \Delta\boldsymbol{u}_t, \tag{103}$$

which used gradients

$$\boldsymbol{g}_{\boldsymbol{x},t} = 2\boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{f}(\boldsymbol{x}_t), \quad \boldsymbol{g}_{\boldsymbol{u},t} = 2\boldsymbol{R}_t \boldsymbol{u}_t, \tag{104}$$

and Hessian matrices

$$\boldsymbol{H}_{\boldsymbol{x}\boldsymbol{x},t} \approx 2\boldsymbol{J}(\boldsymbol{x}_t)^\top \boldsymbol{Q}_t \boldsymbol{J}(\boldsymbol{x}_t), \quad \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{x},t} = \boldsymbol{0}, \quad \boldsymbol{H}_{\boldsymbol{u}\boldsymbol{u},t} = 2\boldsymbol{R}_t. \tag{105}$$

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 (91).

At a trajectory level, the cost can be written as

$$c(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{u}^\top \boldsymbol{R}\, \boldsymbol{u}, \tag{106}$$

where the tracking and control weights are represented by the diagonally concatenated matrices $\boldsymbol{Q} = \text{blockdiag}(\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_T)$ and $\boldsymbol{R} = \text{blockdiag}(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{T-1})$, respectively. In the above, with a slight abuse of notation, we defined $\boldsymbol{f}(\boldsymbol{x})$ as a vector concatenating the vectors $\boldsymbol{f}(\boldsymbol{x}_t)$. Similarly, $\boldsymbol{J}(\boldsymbol{x})$ will represent a block-diagonal concatenation of the Jacobian matrices $\boldsymbol{J}(\boldsymbol{x}_t)$.

With this notation, the minimization problem (85) then becomes

$$\min_{\Delta\boldsymbol{u}} \quad 2\Delta\boldsymbol{x}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) + 2\Delta\boldsymbol{u}^\top \boldsymbol{R}\, \boldsymbol{u} + \Delta\boldsymbol{x}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \Delta\boldsymbol{u}^\top \boldsymbol{R}\Delta\boldsymbol{u}, \quad \text{s.t.} \quad \Delta\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}}\Delta\boldsymbol{u}, \tag{107}$$

whose least squares solution is given by

$$\Delta\hat{\boldsymbol{u}} = \Big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x}) \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\Big)^{-1} \Big(- \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{R}\, \boldsymbol{u}\Big), \tag{108}$$

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.

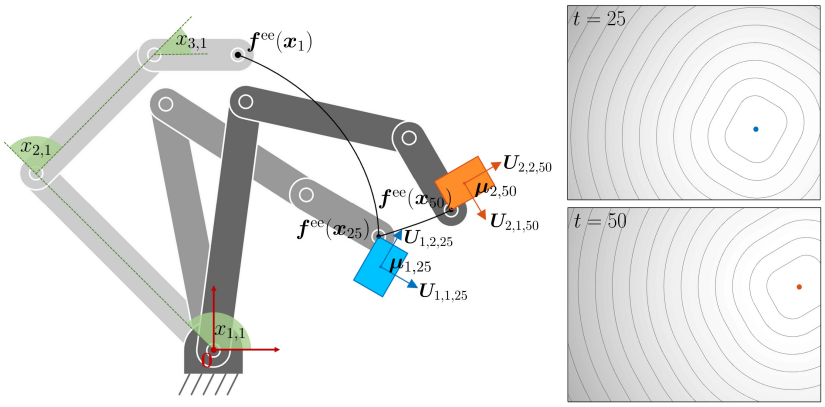


Figure 23: Example of a viapoints task in which a planar robot with 3 joints needs to sequentially reach 2 objects, with object boundaries defining the allowed reaching points on the objects surfaces. *Left:* Reaching task with two viapoints at $t = 25$ and $t = 50$. *Right:* Corresponding values of the cost function for the endeffector space at $t = 25$ and $t = 50$.

9.1 Robot manipulator

`iLQR_manipulator.*`

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 (82) 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.

For a robot manipulator, $\boldsymbol{f}(\boldsymbol{x}_t)$ in (108) typically represents the error between a reference $\boldsymbol{\mu}_t$ and the endeffector position computed by the forward kinematics function $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t)$. We then have

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

9.2 Bounded joint space

The iLQR solution in (108) 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

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

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

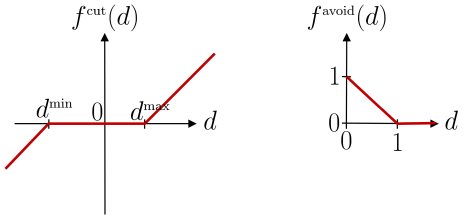


Figure 22: ReLU-like functions used in optimization costs. The derivatives of these functions are simple, providing Jacobians whose entries are either 0 or ± 1.

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. 22-*left*.

We can see with (104) 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.

9.3 Bounded task space

The iLQR solution in (108) can be used to keep the endeffector within a boundary (e.g., endeffector position limits). Based on the above definitions, $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{J}(\boldsymbol{x})$ are in this case defined as

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

9.3.1 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 (20) 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. 23.

9.4 Initial state optimization

iLQR_manipulator_initStateOptim.*

Similarly as in Section 7.1, iLQR can easily be extended to determine the optimal initial state $\boldsymbol{x}_1$ together with the optimal control commands $\boldsymbol{u}$, by defining $\tilde{\boldsymbol{u}} = [\boldsymbol{x}_1^\top, \boldsymbol{u}^\top]^\top$ as a vector concatenating $\boldsymbol{x}_1$ and $\boldsymbol{u}$.

If the system evolution is linear, $\boldsymbol{x} = \boldsymbol{S}\tilde{\boldsymbol{u}}$ with $\boldsymbol{S} = [\boldsymbol{S}_{\boldsymbol{x}}, \boldsymbol{S}_{\boldsymbol{u}}]$ can additionally be used, as in Section 7.1.

By defining $\tilde{\boldsymbol{R}} = \begin{bmatrix}\boldsymbol{0} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{R}\end{bmatrix}$ to remove any constraint on the initial state, the optimization problem to estimate both the initial state $\boldsymbol{x}_1$ and the sequence of control commands $\boldsymbol{u}$ then becomes

$$\min_{\tilde{\boldsymbol{u}}} \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) + \tilde{\boldsymbol{u}}^\top\tilde{\boldsymbol{R}}\tilde{\boldsymbol{u}}, \qquad (109)$$

where $\boldsymbol{f}(\boldsymbol{x})$ denotes the concatenation of $\boldsymbol{f}(\boldsymbol{x}_t)$ for all time steps.

If only some parts of $\boldsymbol{x}_1$ need to be estimated, the corresponding diagonal entries in $\tilde{\boldsymbol{R}}$ can be set with arbitrary high values, by reformulating the minimization problem as

$$\min_{\tilde{\boldsymbol{u}}} \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) + (\tilde{\boldsymbol{u}} - \boldsymbol{\mu})^\top\tilde{\boldsymbol{R}}(\tilde{\boldsymbol{u}} - \boldsymbol{\mu}), \qquad (110)$$

where $\boldsymbol{\mu}$ is a vector containing the initial elements in $\boldsymbol{x}_1$ that do not need to be estimated, where the other elements are zero.

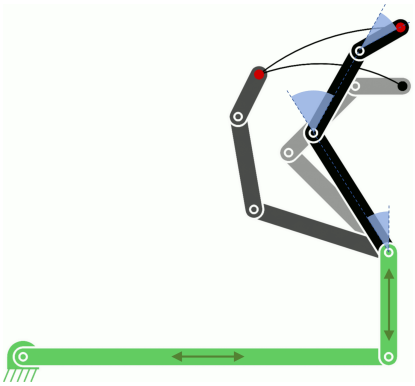


Figure 24: Optimal robot base location for a viapoint task, by defining a kinematic chain with both revolute and prismatic joints, which is employed to encode the location of the robot base in the plane as two prismatic joint variables. In this example, the prismatic joints in green represent the initial states that we estimate, and the angular joints in blue represent the state variables that can be moved during the task.

The above approach can for example be used to estimate the optimal placement of a robot manipulator. For a planar robot, this can for example be implemented by defining the kinematic chain starting with two prismatic joints along the two axes directions, providing a way to represent the location of a free floating platform as two additional prismatic joints in the kinematic chain. If we would like the base of the robot to remain static, we can request that these first two prismatic joints will stay still during the motion. By using the above formulation, this can be done by setting arbitrary large control weights for these corresponding joints, see Figure 24 and the corresponding source code example.

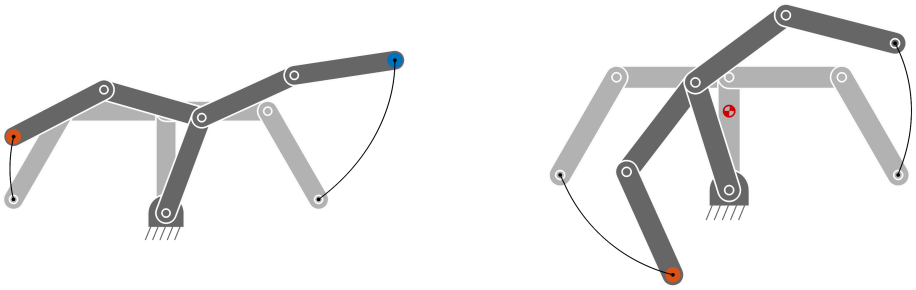


Figure 26: Reaching tasks with a bimanual robot (frontal view). *Left:* with a target for each hand. *Right:* with a target for the hand on the left, while keeping the center of mass at the same location during the whole movement.

9.5 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 (18) for the first two columns (position). The corresponding Jacobian matrix is

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

The forward kinematics function $\boldsymbol{f}^{\text{CoM}}$ can be used in tracking tasks similar to the ones considering the endeffector, as in Section 9.3.1. Fig. 25 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 endeffector at the end of the movement, while always keeping the center of mass within the desired bounding box during the movement.

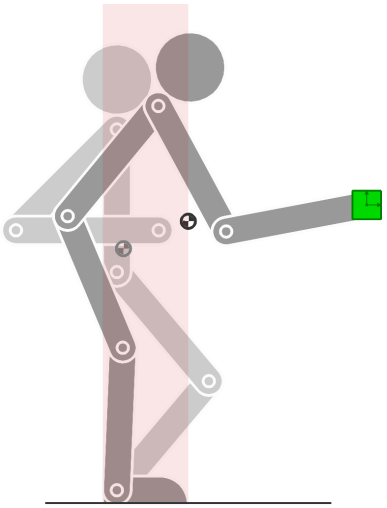


Figure 25: Reaching task with a humanoid (side view) by keeping the center of mass in an area defined by the feet location.

9.6 Bimanual robot

`iLQR_bimanual.*`

We consider a 5-link planar bimanual robot with a torso link shared by the two arms, see Fig. 26. 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 endeffectors, respectively. The corresponding Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{4\times 5}$ has entries

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

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. 26 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

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

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

9.7 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. 27 for an example). We first define a ReLU-like function and its gradient as (see also Fig. 22-*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 (108) 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 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. 27.

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

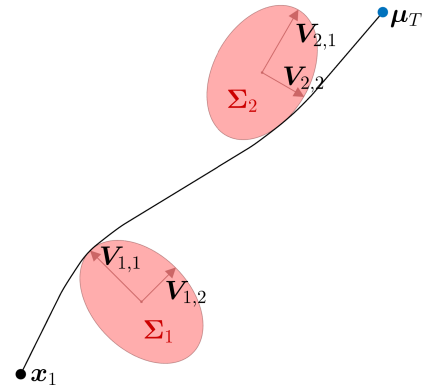


Figure 27: Reaching task with obstacle avoidance, by starting from $\boldsymbol{x}_1$ and reaching $\boldsymbol{\mu}_T$ while avoiding the two obstacles in red.

9.8 Distance to a target

`iLQR_distMaintenance.*`

The obstacle example above can easily be extended to the problem of reaching/maintaining a desired distance to a target, 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 (108) 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 target (defined by a center $\boldsymbol{\mu}$), and increasing/decreasing when moving away from this surface in one direction or the other.

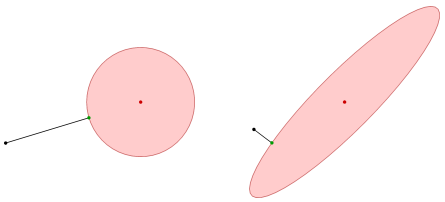


Figure 28: *Left:* Optimal control problem to generate a motion by considering a point mass starting from an initial point (in black), with a cost asking to reach or maintain a desired distance to a target point (in red). *Right:* Similar problem for a contour to reach/maintain defined as a covariance matrix.

Similarly, the error can be weighted by a covariance matrix $\boldsymbol{\Sigma}$, by using $e(\boldsymbol{x}_t) = (\boldsymbol{x}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_t - \boldsymbol{\mu})$ and $\boldsymbol{J}(\boldsymbol{x}_t) = -2(\boldsymbol{x}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}$, which corresponds to the problem of reaching/maintaining the contour of an ellipse.

Figure 28 presents examples with a point mass.

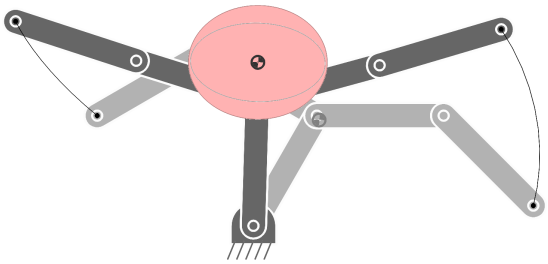


Figure 29: Example of iLQR optimization with a cost on manipulability, where the goal is to reach, at the end of the motion, a desired manipulability defined at the center of mass (CoM) of a bimanual planar robot. The initial pose of the bimanual robot is displayed in light gray and the final pose of 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.

9.9 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 [16]. 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 endeffector(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 [19], with the drawback of ignoring the specific shape of this ellipsoid.

In [7], we showed that a geometric cost on manipulability can alternatively be defined with the geodesic distance

$$\begin{aligned} & c = \|\boldsymbol{A}\|_{\mathrm{F}}^2, \quad \text{with} \quad \boldsymbol{A} = \log\big(\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{M}(\boldsymbol{x})\boldsymbol{S}^{-\frac{1}{2}}\big), \\ \Longleftrightarrow \quad & c = \mathrm{trace}(\boldsymbol{A}\boldsymbol{A}^\top) = \sum_i \mathrm{trace}(\boldsymbol{A}_i\boldsymbol{A}_i^\top) = \sum_i \boldsymbol{A}_i^\top\boldsymbol{A}_i = \mathrm{vec}(\boldsymbol{A})^\top\mathrm{vec}(\boldsymbol{A}), \end{aligned} \tag{111}$$

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 (111) 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 (111) 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 [7]), or by automatic differentiation. The quadratic form of (111) can be exploited to solve the problem with Gauss–Newton optimization, by using Jacobians (see description in Section 5.1 within the context of inverse kinematics). Marić *et al.* demonstrated the advantages of a geometric cost as in (111) for planning problems, by comparing it to alternative widely used metrics such as the manipulability index and the dexterity index [9].

Note here that manipulability can be defined at several points of interest, including endeffectors and centers of mass. Figure 29 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 [14], 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).

9.10 Decoupling of control commands

In some situations, we would like to control a robot by avoiding that all degrees of freedom are controlled at the same time. For example, we might in some cases prefer that a mobile manipulator stops moving the arm while the platform is moving and vice versa.

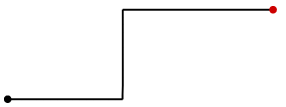


Figure 30: 2D reaching task problem (initial point in black and target point in red), with an additional cost to favor the decoupling of the control commands.

For a 2 DOFs robot controlled at each time step t with two control command $u_{1,t}$ and $u_{2,t}$, the corresponding cost to find the full sequence of control commands for a duration T (gathered in a concatenated vector $\boldsymbol{u}$) is

$$\min_{\boldsymbol{u}} \sum_{t=1}^{T} (u_{1,t}\, u_{2,t})^2,$$

with corresponding residuals and Jacobian given by

$$f_t = u_{1,t}\, u_{2,t}, \quad \boldsymbol{J}_t = \big[u_{2,t},\, u_{1,t}\big], \quad \forall t \in \{1, \dots, T\}.$$

The Gauss–Newton update rule in concatenated vector form is given by

$$\boldsymbol{u}_{k+1} \;\leftarrow\; \boldsymbol{u}_k - \alpha \left(\boldsymbol{J}^\top \boldsymbol{J}\right)^{-1} \boldsymbol{J}^\top \boldsymbol{f},$$

where α is a line search parameter, $\boldsymbol{f}$ is a vector concatenating vertically all residuals f_t, and $\boldsymbol{J}$ is a Jacobian matrix with $\boldsymbol{J}_t$ as block diagonal elements. With some linear algebra machinery, this can also be computed in batch form using $\boldsymbol{f} = \boldsymbol{u}_1 \odot \boldsymbol{u}_2$, and $\boldsymbol{J} = (\boldsymbol{I}_{T-1} \otimes \boldsymbol{1}_{1\times 2})\, \mathrm{diag}\Big(\big(\boldsymbol{I}_{T-1} \otimes (\boldsymbol{1}_{2\times 2} - \boldsymbol{I}_2)\big)\boldsymbol{u}\Big)$, with $\odot$ and $\otimes$ the elementwise product and Kronecker product operators, respectively.

Figure 30 presents a simple example within a 2D reaching problem with a point mass agent and velocity commands.

9.11 Curvature

The curvature of a 2-dimensional curve is defined as

$$\kappa = \frac{\dot{\mathrm{x}}_1 \ddot{\mathrm{x}}_2 - \dot{\mathrm{x}}_2 \ddot{\mathrm{x}}_1}{(\dot{\mathrm{x}}_1^2 + \dot{\mathrm{x}}_2^2)^{\frac{3}{2}}}. \tag{112}$$

We describe the system state in vector form as

$$\boldsymbol{x} = \begin{bmatrix} \mathrm{x} \\ \dot{\mathrm{x}} \\ \ddot{\mathrm{x}} \end{bmatrix}. \tag{113}$$

By defining a set of selection vectors $\boldsymbol{s}_{i,j}$ so that $\dot{\mathrm{x}}_1 = \boldsymbol{s}_{1,1}^\top \boldsymbol{x}$, $\dot{\mathrm{x}}_2 = \boldsymbol{s}_{1,2}^\top \boldsymbol{x}$, $\ddot{\mathrm{x}}_1 = \boldsymbol{s}_{2,1}^\top \boldsymbol{x}$, $\ddot{\mathrm{x}}_2 = \boldsymbol{s}_{2,2}^\top \boldsymbol{x}$, we can observe that

$$\dot{\mathrm{x}}_1 \ddot{\mathrm{x}}_2 = (\boldsymbol{s}_{1,1}^\top \boldsymbol{x})(\boldsymbol{s}_{2,2}^\top \boldsymbol{x}) = \boldsymbol{x}^\top (\boldsymbol{s}_{1,1} \boldsymbol{s}_{2,2}^\top) \boldsymbol{x}, \tag{114}$$

where $(\boldsymbol{s}_{1,1} \boldsymbol{s}_{2,2}^\top)$ is a selection matrix.

By leveraging this matrix formulation, the curvature in (112) can be expressed as

$$\kappa(\boldsymbol{x}) = (\boldsymbol{x}^\top \boldsymbol{S}_B\, \boldsymbol{x})^{-\frac{3}{2}}\; \boldsymbol{x}^\top \boldsymbol{S}_A\, \boldsymbol{x}, \tag{115}$$

with selection matrices $\boldsymbol{S}_A = \boldsymbol{s}_{1,1} \boldsymbol{s}_{2,2}^\top - \boldsymbol{s}_{1,2} \boldsymbol{s}_{2,1}^\top$ and $\boldsymbol{S}_B = \boldsymbol{s}_{1,1} \boldsymbol{s}_{1,1}^\top + \boldsymbol{s}_{1,2} \boldsymbol{s}_{1,2}^\top$.

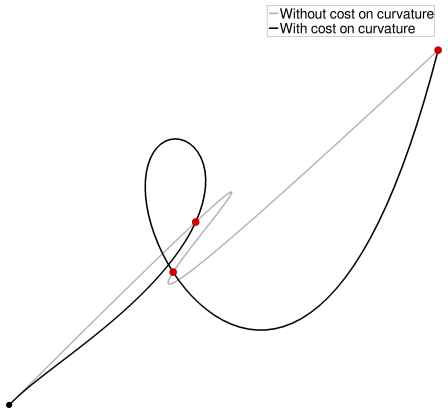


Figure 31: Optimal control problem to generate a motion by considering a point mass starting from an initial point (in black), with a cost asking to pass through a set of viapoints (in red), together with a cost on curvature (black path).

With this formulation, by using the derivative property $(fg)' = f'g + fg'$, and by observing that $\boldsymbol{S}_B$ is a symmetric matrix and that $\boldsymbol{S}_A$ is an asymmetric matrix, the derivatives of (112) w.r.t $\boldsymbol{x}$ form the Jacobian

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \kappa(\boldsymbol{x})}{\partial \boldsymbol{x}} = (\boldsymbol{x}^\top \boldsymbol{S}_B \boldsymbol{x})^{-\frac{3}{2}}\; (\boldsymbol{S}_A + \boldsymbol{S}_A^\top)\boldsymbol{x} - 3\, (\boldsymbol{x}^\top \boldsymbol{S}_A\, \boldsymbol{x})\, (\boldsymbol{x}^\top \boldsymbol{S}_B\, \boldsymbol{x})^{-\frac{5}{2}}\; \boldsymbol{S}_B \boldsymbol{x}. \tag{116}$$

Figure 31 presents an example with a point mass.

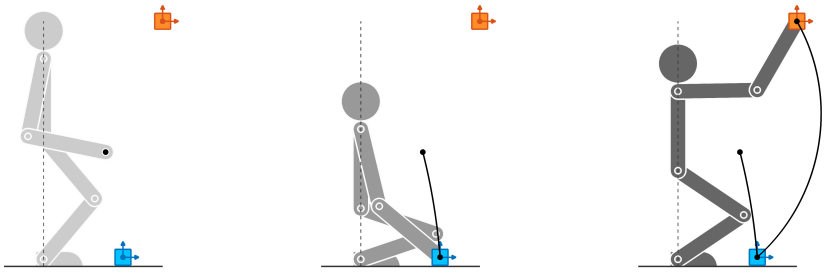


Figure 32: Reaching task with control primitives.

9.12 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

$$\Delta \hat{\boldsymbol{w}} = \Big(\boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{J}(\boldsymbol{x}) \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{\Psi} + \boldsymbol{\Psi}^\top \boldsymbol{R}\, \boldsymbol{\Psi} \Big)^{-1} \Big(- \boldsymbol{\Psi}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q} \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{\Psi}^\top \boldsymbol{R}\, \boldsymbol{u} \Big). \tag{117}$$

For a 5-link kinematic chain, by setting the coordination matrix in (66) 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 32 shows an example for a reaching task, with the 5-link kinematic chain depicting a humanoid robot that needs to keep its torso upright.

Note also that the iLQR updates in (117) can usually be computed faster than in the original formulation, since a matrix of much smaller dimension needs to be inverted.

9.13 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

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

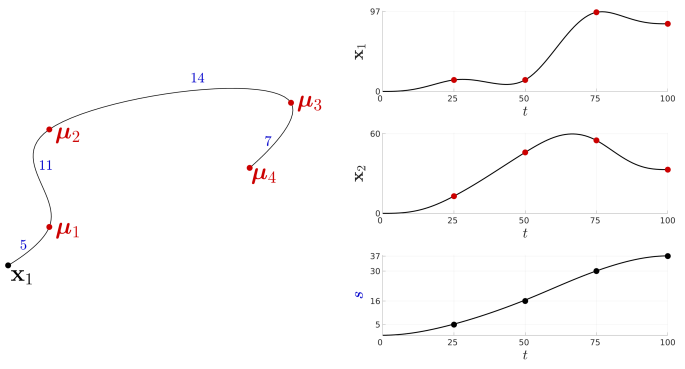


Figure 33: Spacetime optimization with iLQR for a viapoints task, where the estimated durations between consecutive viapoints are shown in blue.

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 33 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.

9.14 iLQR with offdiagonal elements in the precision matrix

`iLQR_manipulator_object_affordance.*`

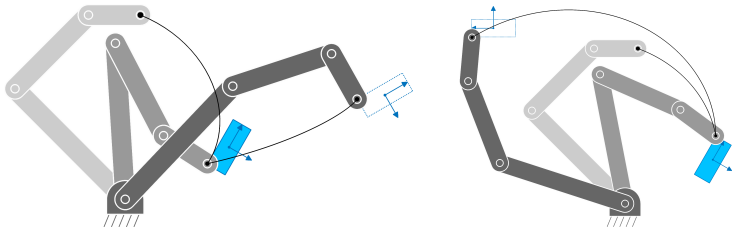


Figure 34: Manipulation with object affordances cost. The depicted *pick-and-place* task requires the offset at the picking location to be the same as the offset at the placing location, which can be achieved by setting a precision matrix with nonzero offdiagonal entries, see main text for details. In 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. 23. 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.

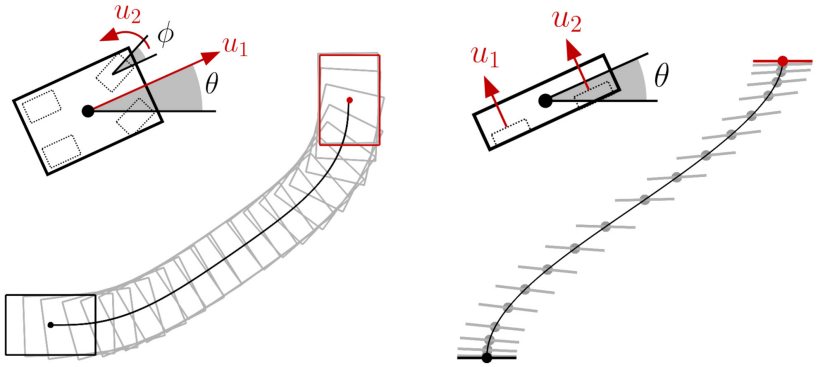


Figure 35: iLQR for car and bicopter control/planning problems.

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 constrain 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. 34 for an example.

9.15 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. 35-*left*) is characterized by equations

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

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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{u}_1 \sin(\hat{\theta}) & 0 \\ 0 & 0 & \hat{u}_1 \cos(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{\hat{u}_1}{\ell}(\tan(\hat{\phi})^2+1) \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} \cos(\hat{\theta}) & 0 \\ \sin(\hat{\theta}) & 0 \\ \frac{1}{\ell}\tan(\hat{\phi}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{118}$$

which can then be converted to a discrete form with (192).

Acceleration Control Input

A car motion with position (x_1, x_2), car orientation θ, and front wheels orientation ϕ (see Fig. 35-*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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{v} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{v} \sin(\hat{\theta}) & \cos(\hat{\theta}) & 0 \\ 0 & 0 & \hat{v} \cos(\hat{\theta}) & \sin(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{1}{\ell}\tan(\hat{\phi}) & \frac{\hat{v}}{\ell}(\tan(\hat{\phi})^2+1) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta v \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{119}$$

which can then be converted to a discrete form with (192).

9.16 Bicopter

`iLQR_bicopter.*`

A planar bicopter of mass m and inertia I actuated by two propellers at distance ℓ (see Fig. 35-*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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\ddot{x}_1 \\ \Delta\ddot{x}_2 \\ \Delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1+\hat{u}_2)\cos(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1+\hat{u}_2)\sin(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{1}{m}\sin(\hat{\theta}) & -\frac{1}{m}\sin(\hat{\theta}) \\ \frac{1}{m}\cos(\hat{\theta}) & \frac{1}{m}\cos(\hat{\theta}) \\ \frac{\ell}{I} & -\frac{\ell}{I} \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \tag{120}$$

which can then be converted to a discrete form with (192).

10 Ergodic control

Ergodic theory studies the connection between the time-averaged and space-averaged behaviors of a dynamical system. By using it in a search and exploration context, it enables optimal exploration of an information distribution. Exploration problems can be formulated in various ways, see Figures 36, 37 and 39.

Research in ergodic control addresses fundamental challenges linking machine learning, optimal control, signal processing and information theory. A conventional tracking problem in robotics is characterized by a target to reach, requiring a controller to be computed to reach this target. In *ergodic control*, instead of providing a single target point, a probability distribution is given to the robot, which must cover the distribution in an efficient way. Ergodic control thus consists of moving within a spatial distribution by spending time in each part of the distribution in proportion to its density. The term ergodicity corresponds here to the difference between the time-averaged spatial statistics of the agent's trajectory and the target distribution to search in. By minimizing this difference, the resulting controller generates natural exploration behaviors.

In robotics, ergodic control can be exploited in a wide range of problems requiring the automatic exploration of regions of interest, which can be used in a wide range of tasks, including active sensing, localization, surveillance, or insertion (see Figure 38 for an example). This is particularly helpful when the available sensing information or robot actuators are not accurate enough to fulfill the task with a standard controller (limited vision, soft robotic manipulator, etc.), but where this information can still guide the robot towards promising areas. In a collaborative task, it can also be used when the operator's input is not accurate enough to fully reproduce the task, which then requires the robot to explore around the requested input (e.g., a point of interest selected by the operator). For picking and insertion problems, ergodic control can be applied to move around the picking/insertion point, thereby facilitating the prehension/insertion. It can also be employed for active sensing and localization (either detected autonomously, or with help by the operator). Here, the robot can plan movements based on the current information density, and can recompute the commands when new measurements are available (i.e., updating the spatial distribution used as target).

Ergodic control is originally formulated as a *spectral multiscale coverage* (SMC) objective [10], which we will see next. Other ergodic control techniques have later been proposed, including the *heat equation driven area coverage* (HEDAC) [6], which we will also be introduced next.

10.1 Spectral-based ergodic control (SMC)

The underlying objective of *spectral multiscale coverage* (SMC) takes a simple form, corresponding to a tracking problem in the spectral domain (matching of frequency components) [10]. The advantage of such a simple control formulation is that it can be easily combined with other control objectives and constraints.

It requires the spatial distribution to be decomposed as Fourier series, with a cost function comparing the spectral decomposition of the robot path with the spectral decomposition of the distribution, in the form of a weighted distance function with a decreasing weight from low frequency to high frequency components. The resulting controller allows the robot to explore the given spatial distribution in a natural manner, **by starting from a crude exploration and by refining the search progressively** (i.e., matching the Fourier coefficients with an increasing importance from low to high frequency components).

10.1.1 Unidimensional SMC

We will adopt a notation to make links with the superposition of basis functions seen in Section 6. By starting with the unidimensional case (system moving along a line segment), we will consider a signal $g(x)$ varying along a variable x, where x will be used as a generic variable that can for example be a time variable or the coordinates of a pixel in an image. The signal $g(x)$ can be approximated as a weighted superposition of basis functions with

$$g(x) = \sum_{k=-K+1}^{K-1} w_k \, \phi_k(x). \tag{121}$$

The use of Fourier basis functions provides useful connections between the spatial domain and the spectral domain, where w_k and $\phi_k(x)$ denote the coefficients and basis functions of the Fourier series.

For a spatial signal x on the interval $[-\frac{L}{2}, \frac{L}{2}]$ of period L, the basis functions of the Fourier series with complex exponential functions will be defined as

$$\begin{aligned} \phi_k(x) &= \frac{1}{2L} \exp\left(-i \frac{2\pi k x}{L}\right) \\ &= \frac{1}{2L} \left(\cos\left(\frac{2\pi k x}{L}\right) - i \, \sin\left(\frac{2\pi k x}{L}\right)\right), \; \forall k \in [-K+1, \ldots, K-1], \end{aligned} \tag{122}$$

with i the imaginary unit of a complex number ($i^2 = -1$).

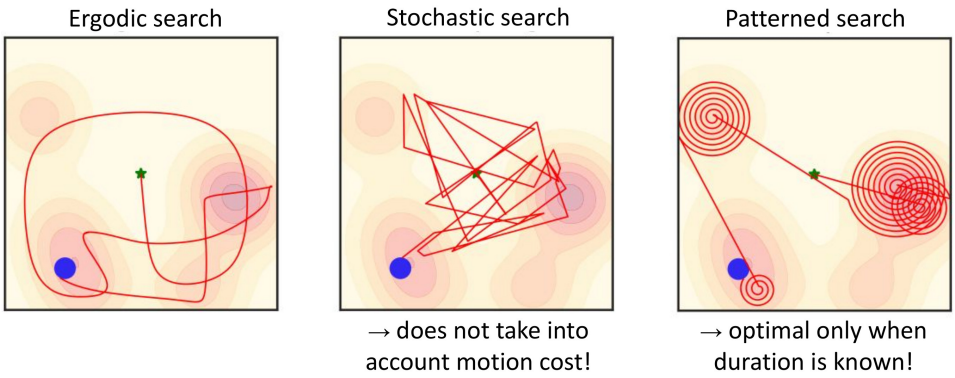


Figure 36: Search and exploration problems can be formulated in various ways. Ergodic control can be used in a search problem in which we do not know the duration of the exploration (e.g., we would like to find a hidden target as soon as possible). In contrast to stochastic sampling of a distribution, ergodic control takes into account the cost to move the agent from one point to another. If the search optimization problem is formulated as a coverage problem with fixed duration, the optimal path typically reveals some regular coverage patterns (here, illustrated as spirals, but also called lawnmower strategy). Ergodic control instead generates a more natural search behavior to localize the object to search by starting with a crude coverage of the workspace and by then generating progressively more detailed paths to cover the distribution. This for example resembles the way we would search for keys in a house by starting with a distribution about the locations in the rooms in which we believe the keys could be, and by covering these distribution coarsely as a first pass, followed by progressive refinement about the search with more detailed coverage until the keys are found. Both stochastic samples or patterned search would be very inefficient here!

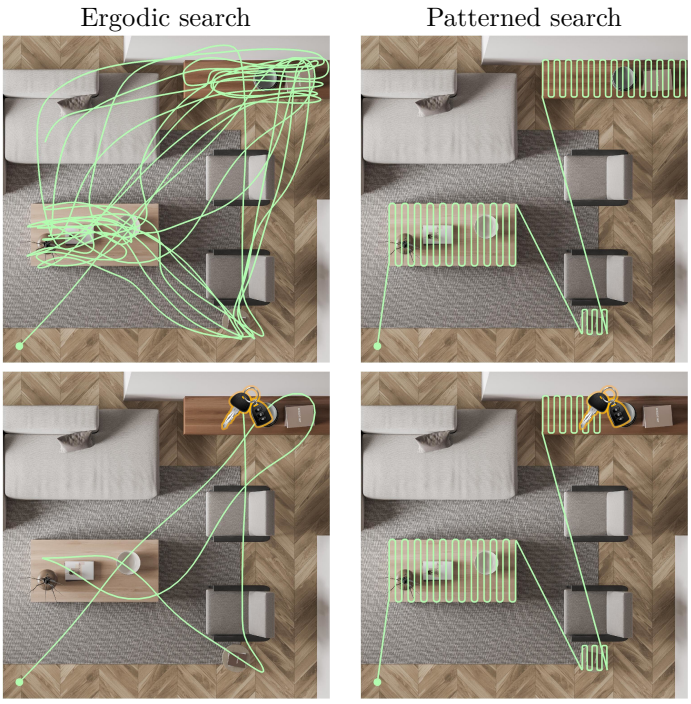


Figure 37: Example illustrating the search of a key in a living room, where regions of interest are specified in the form of a distribution over the spatial domain. These regions of interest correspond here to the different tables in the room where it is more likely to find the key. The left and right columns respectively depict an ergodic search and patterned search processes. For a coverage problem in which the duration is given in advance, a patterned search can typically be better suited than an ergodic control search, because the agent will optimize the path to reduce the number of transitions from one area to the other. In contrast, an ergodic control search will typically result in multiple transitions between the different regions of interest. When searching for a key, since we are interested in finding the key as early as possible, we have a problem in which the total search duration cannot be specified in advance. In this problem setting, an ergodic search will be better suited than a patterned search, as it generates a natural way of exploring the environment by first proceeding with a crude search in the whole space and by then progressively fine-tuning the search with additional details until the key is finally found. The bottom row shows an example in which the key location is unknown, with the search process stopping when the key is found. Here, a patterned search would provide an inefficient and unnatural way of scanning the whole environment regularly by looking at all details regions by regions, instead of treating the problem at different scales (i.e., at different spatial frequencies), which is what ergodic exploration provides.

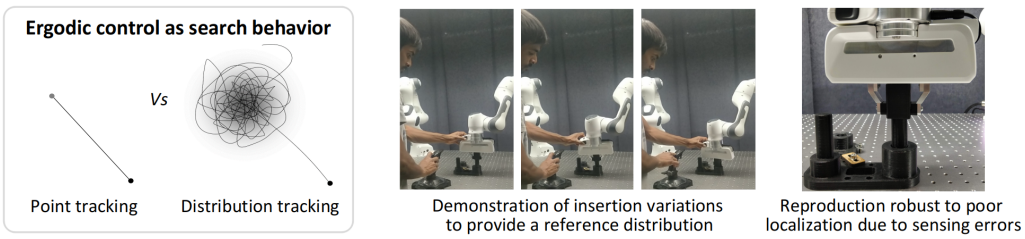


Figure 38: Insertion tasks can be achieved robustly with ergodic control, by not only relying on accurate sensors, but instead using a control strategy to cope with limited or inaccurate sensing information, see [18] for details.

Table 1: Fourier series properties (1D case).

Symmetry property:
If $g(x)$ is real and even, $\phi_k(x)$ in (122) is also real and even, simplifying to $\phi_k(x) = \frac{1}{L}\cos\big(\frac{2\pi kx}{L}\big)$, which then, in practice, only needs an evaluation on the range $k \in [0, \dots, K-1]$, as the basis functions are even. We then have $g(x) = w_0 + \sum_{k=1}^{K-1} w_k \, 2\cos\big(\frac{2\pi kx}{L}\big)$, by exploiting $\cos(0)=1$.

Shift property:
If w_k are the Fourier series coefficients of a function $g(x)$, $\exp(-i\frac{2\pi k\mu}{L})w_k$ are the Fourier coefficients of $g(x-\mu)$.

Combination property:
If $w_{k,1}$ (resp. $w_{k,2}$) are the Fourier series coefficients of a function $g_1(x)$ (resp. $g_2(x)$), then $\alpha_1 w_{k,1} + \alpha_2 w_{k,2}$ are the Fourier coefficients of $\alpha_1 g_1(x) + \alpha_2 g_2(x)$.

Gaussian property:
If $g_0(x) = \mathcal{N}(x \,|\, 0, \sigma^2) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp(-\frac{x^2}{2\sigma^2})$ is mirrored to create a real and even periodic function $g(x)$ of period $L \gg \sigma$ (implementation details will follow), the corresponding Fourier series coefficients are of the form $w_k = \exp(-\frac{2\pi^2 k^2 \sigma^2}{L^2})$.

As detailed in Table 1, when $g(x)$ is real and even (our case of interest), $\phi_k(x)$ in (122) is also real and even, simplifying to $\phi_k(x) = \frac{1}{L}\cos\big(\frac{2\pi kx}{L}\big)$, which then, in practice, only needs an evaluation on the range $k \in [0, \dots, K-1]$, using

$$\phi_k(x) = \frac{1}{L}\cos\left(\frac{2\pi kx}{L}\right), \quad \forall k \in [0, \dots, K-1]. \tag{123}$$

The derivatives of (123) with respect to x are easy to compute, namely

$$\nabla_x \phi_k(x) = -\frac{2\pi k}{L^2}\sin\left(\frac{2\pi kx}{L}\right). \tag{124}$$

Ergodic control aims to build a controller so that the Fourier series coefficients w_k along a trajectory $x(t)$ of duration t, defined as

$$w_k = \frac{1}{t}\int_{\tau=0}^{t} \phi_k\big(x(\tau)\big)\, d\tau, \tag{125}$$

will match the Fourier series coefficients $\hat{w}_k$ on the spatial domain $\mathcal{X}$, defined as

$$\hat{w}_k = \int_{x\in\mathcal{X}} \hat{g}(x)\, \phi_k(x)\, \mathrm{d}x. \tag{126}$$

The discretized version of (125) for a discrete number of time steps T is

$$w_k = \frac{1}{T}\sum_{s=1}^{T} \phi_k(x_s), \tag{127}$$

which can be computed recursively by updating w_k at each iteration step.

Similarly, the discretized version of (126) for a given spatial resolution can for example be computed on a domain $\mathcal{X}$ discretized in X bins as

$$\hat{w}_k = \sum_{s=1}^{X} \hat{g}(x_s)\, \phi_k(x_s), \tag{128}$$

where x_s splits the domain $\mathcal{X}$ into X regular intervals.

To compute these Fourier series coefficients efficiently, several Fourier series properties can be exploited, including general property such as symmetry, shift, and linear combination, as well as more specific property in the special case when the spatial distribution takes the form of a mixture of Gaussians. These properties are reported in Table 1 for the 1D case.

Well-known applications of Fourier basis functions in the context of time series include speech processing and the analysis of periodic motions such as gaits. In ergodic control, the aim is instead to find a series of control commands $u(t)$ so that the retrieved trajectory $x(t)$ covers a bounded space $\mathcal{X}$ in proportion of a desired spatial distribution $\hat{g}(x)$. This can be achieved by defining a metric in the spectral domain, by decomposing in Fourier series coefficients both the desired spatial distribution $\hat{g}(x)$ and the (partially) retrieved trajectory $x(t)$, which can exploit the Fourier series properties presented in Table 1, which can also be extended to more dimensions.

The goal of ergodic control is to minimize

$$\epsilon = \frac{1}{2}\sum_{k=0}^{K-1} \Lambda_k \Big(w_k - \hat{w}_k\Big)^2 \tag{129}$$

where Λ_k are weights, $\hat{w}_k$ are the Fourier series coefficients of $\hat{g}(x)$, and w_k are the Fourier series coefficients along the trajectory $x(t)$. $k \in [0, 1, \dots, K-1]$ is a set of K index vectors. In (129), the weights

$$\Lambda_k = \big(1 + k^2\big)^{-1} \tag{130}$$

assign more importance on matching low frequency components (related to a metric for Sobolev spaces of negative order).

Practically, the controller will mainly focus on the lowest frequency components at the beginning as these components have a large effect on the cost due to the large weights. When the errors become zero for these components, the controller can then try to progressively reduce higher frequency components.

Ergodic control is then set as the constrained problem of computing a control command $\hat{u}(t)$ at each time step t with

$$\hat{u}(t) = \arg\min_{u(t)} \epsilon\big(x(t{+}\Delta t)\big), \quad \text{s.t.} \quad \dot{x}(t) = f\big(x(t), u(t)\big), \quad \|u(t)\| \leqslant u^{\max}, \tag{131}$$

where the simple system $u(t) = \dot{x}(t)$ is considered (control with velocity commands), and where the error term is approximated with the Taylor series

$$\epsilon\big(x(t{+}\Delta t)\big) \;\approx\; \epsilon\big(x(t)\big) \;+\; \dot{\epsilon}\big(x(t)\big)\Delta t \;+\; \frac{1}{2}\ddot{\epsilon}\big(x(t)\big)\Delta t^2. \tag{132}$$

By using (129), (125), (123) and the chain rule $\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t}$, the Taylor series is composed of the control term $u(t)$ and $\nabla_x \phi_k\big(x(t)\big) \in \mathbb{R}$, the gradient of $\phi_k\big(x(t)\big)$ with respect to $x(t)$. Solving the constrained objective in (157) then results in the analytical solution (see [10] for the complete derivation)

$$u(t) = \tilde{u}(t)\frac{u^{\max}}{\|\tilde{u}(t)\|} = \operatorname{sign}\big(u(t)\big)u^{\max}, \quad \text{with} \quad \tilde{u}(t) = -\sum_{k=0}^{K-1} \Lambda_k \big(w_k - \hat{w}_k\big)\nabla_x \phi_k\big(x(t)\big)$$

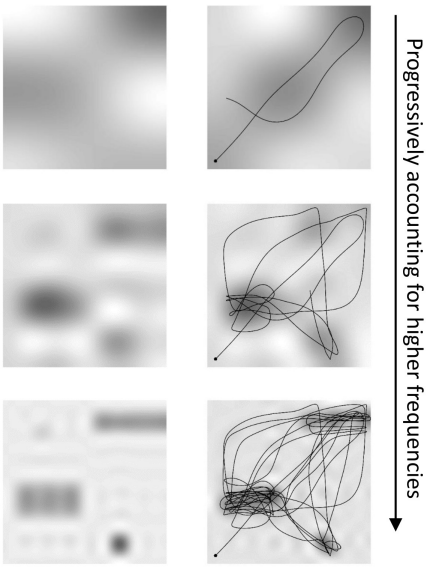


Figure 39: Although ergodic control with SMC is specified in the spectral domain, it can also be interpreted in the spatial domain as a coverage problem in which both the path of the agent and the targeted distributions are blurred with filters of varying strength. With very strong blurring, the agent can match the desired distribution very quickly (first row). The weaker the filter, the more time the agent will need to cover the path to match the details of the distribution (second and third row). As an optimization problem, the weights Λ_k are set to promote a good match of the distributions that are strongly blurred (i.e., low frequency components). When this part of the cost goes to zero, the agent can then progressively refine the path so that less blurred distributions can also be matched (i.e., higher frequency components).

It is important to emphasize that the implementation of this controller does not rely on any random number generation: this search behavior is instead deterministic.

Computation in vector form

`ergodic_control_SMC_1D.*`

The superposition of basis functions in (121) can be rewritten as

$$g(x) = \boldsymbol{w}^\top \boldsymbol{\phi}(x), \tag{133}$$

where $\boldsymbol{w}$ and $\boldsymbol{\phi}(x)$ are vectors formed with the K elements of w_k and $\phi_k(x)$, respectively.

The goal of ergodic control in (129) is to minimize

$$\epsilon = \frac{1}{2}\Big(\boldsymbol{w} - \hat{\boldsymbol{w}}\Big)^\top \boldsymbol{\Lambda} \Big(\boldsymbol{w} - \hat{\boldsymbol{w}}\Big), \tag{134}$$

where $\hat{\boldsymbol{w}}$ is a vector formed with the K elements $\hat{w}_k$, and $\boldsymbol{\Lambda}$ is a weighting matrix formed with the K diagonal elements Λ_k.

By defining $\boldsymbol{\phi}(x_s)$ as the vector concatenating the K elements $\phi_k(x_s)$, (127) can be rewritten in a vector form as

$$\boldsymbol{w} = \frac{1}{T}\sum_{s=1}^{T} \boldsymbol{\phi}(x_s). \tag{135}$$

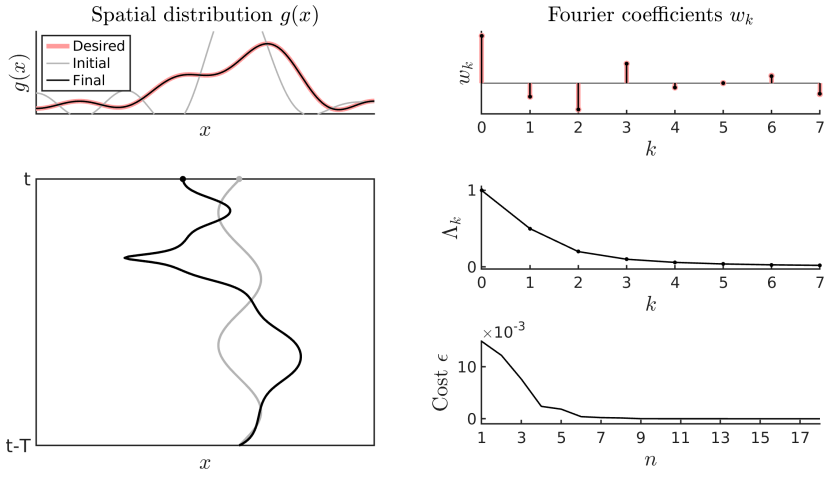


Figure 40: Unidimensional SMC problem computed with iLQR. A series of control commands is used as initial guess (see timeline plot in gray on the bottom-left), which is then iteratively refined by iLQR (see timeline plot in black on the bottom-left). The top plots shows the reference distribution (in red) and the reconstructed distributions (in gray for initial guess and black after iLQR), in both spatial domain (top-left) and spectral domain (top-right).

With some abuse of notation, by defining a vector $\tilde{\boldsymbol{v}} = [0, \dots, K-1]^\top$, we can conveniently compute

$$\boldsymbol{\phi}(x) = \frac{1}{L}\cos(\boldsymbol{v}x), \tag{136}$$

$$\nabla_x \boldsymbol{\phi}(x) = -\frac{\boldsymbol{v}}{L}\sin(\boldsymbol{v}x), \quad \text{with} \quad \boldsymbol{v} = \frac{2\pi\tilde{\boldsymbol{v}}}{L}, \tag{137}$$

where $\boldsymbol{\nabla}_x\boldsymbol{\phi}(x)$ is a vector formed by the K elements $\nabla_x\phi_k(x)$, yielding the controller

$$u(t) = -\boldsymbol{\nabla}_x\boldsymbol{\phi}(x)^\top \, \boldsymbol{\Lambda}\left(\boldsymbol{w} - \hat{\boldsymbol{w}}\right). \tag{138}$$

Figure 39 shows how this controller can be interpreted in the spatial domain.

Planning formulation of unidimensional SMC with iLQR

`ergodic_control_SMC_DDP_1D.*`

By exploiting the results of Section 9, the batch formulation of iLQR consists of minimizing the cost

$$c(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{f}(\boldsymbol{x})^\top \boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{u}^\top \boldsymbol{R}\,\boldsymbol{u}, \tag{139}$$

with $\boldsymbol{Q} = \boldsymbol{\Lambda}$ and $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{w}(\boldsymbol{x}) - \hat{\boldsymbol{w}}$, with

$$\boldsymbol{w}(\boldsymbol{x}) = \frac{1}{T}\sum_{s=1}^{T}\boldsymbol{\phi}(\boldsymbol{x}_s), \tag{140}$$

By starting from an initial guess, the iterative updates of (139) are given by

$$\Delta\hat{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q}\boldsymbol{J}(\boldsymbol{x})\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1}\left(-\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{R}\,\boldsymbol{u}\right), \tag{141}$$

with $\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}$ a Jacobian matrix given by

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{1}{T}\left[\nabla_x\boldsymbol{\phi}(x_1), \nabla_x\boldsymbol{\phi}(x_2), \dots \nabla_x\boldsymbol{\phi}(x_T)\right]. \tag{142}$$

Figure 40 presents the output of the accompanying source codes.

10.1.2 Multidimensional SMC

`ergodic_control_SMC_2D.*`

`ergodic_control_SMC_DDP_2D.*`

For the multidimensional case, we will define $\mathcal{K}$ as a set of index vectors covering a D-dimensional array $\boldsymbol{k} = \boldsymbol{r} \times \boldsymbol{r} \times \cdots \times \boldsymbol{r}$, with $\boldsymbol{r} = [0, 1, \dots, K-1]$ and K the resolution of the array. For example, with $D = 2$ and $K = 2$, we will have $\mathcal{K} = \left\{ \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right] \right\}$.

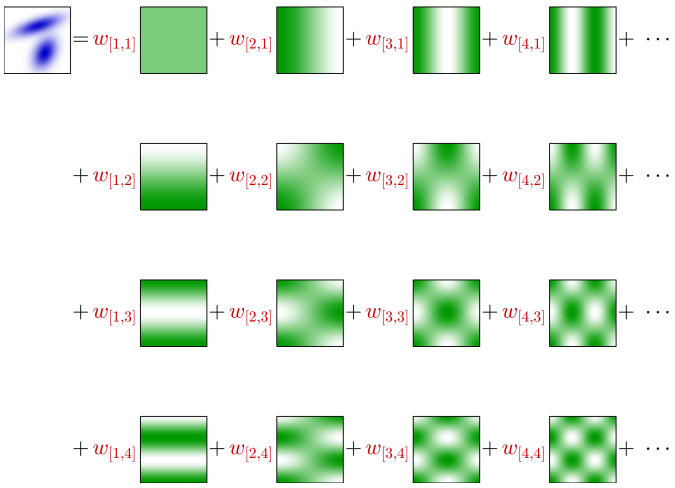


Figure 41: Decomposition of the spatial distribution using Fourier basis functions.

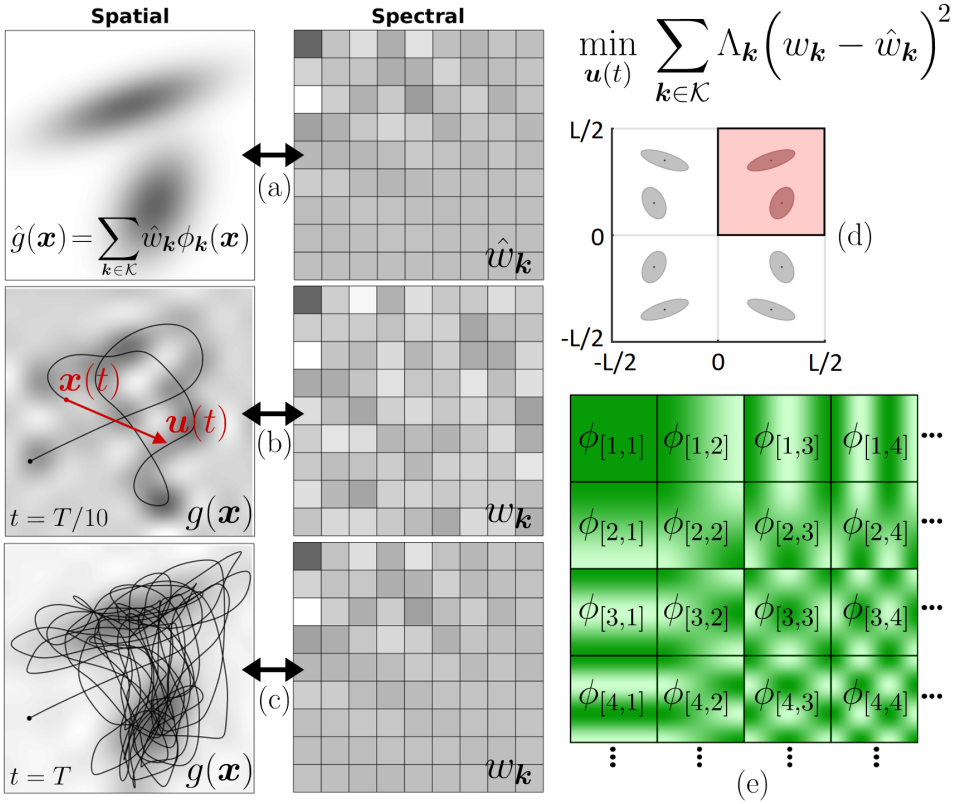


Figure 42: 2D ergodic control based on SMC used to generate search behaviors. *(a)* shows the spatial distribution $\hat{g}(\boldsymbol{x})$ that the agent has to explore, encoded here as a mixture of two Gaussians (gray colormap in left graph). The right graphs show the corresponding Fourier series coefficients $\hat{w}_{\boldsymbol{k}}$ in the frequency domain ($K = 9$ coefficients per dimension). *(b)* shows the evolution of the reconstructed spatial distribution $g(\boldsymbol{x})$ (left graph) and the computation of the next control command $\boldsymbol{u}$ (red arrow) after $T/10$ iterations. The corresponding Fourier series coefficients $w_{\boldsymbol{k}}$ are shown in the right graph. *(c)* shows that after T iterations, the agent covers the space in proportion to the desired spatial distribution, with a good match of coefficients in the frequency domain (we can see that $\hat{w}_{\boldsymbol{k}}$ and $w_{\boldsymbol{k}}$ are nearly the same). *(d)* shows how a periodic signal $\hat{g}(\boldsymbol{x})$ (with range $[-L/2, L/2]$ for each dimension) can be constructed from an initial mixture of two Gaussians $\hat{g}_0(\boldsymbol{x})$ (red area). The constructed signal $\hat{g}(\boldsymbol{x})$ is composed of eight Gaussians in this 2D example, by mirroring the Gaussians along horizontal and vertical axes to construct an even signal of period L. *(e)* depicts the first few basis functions of the Fourier series for the first four coefficients in each dimension, represented as a 2D colormap corresponding to periodic signals of different frequencies along two axes.

Figure 43: Bidimensional SMC problem computed with iLQR.

Similarly as the unidimensional case, we will build a controller so that the Fourier series coefficients $w_{\boldsymbol{k}}$ along a trajectory $\boldsymbol{x}(t)$ of duration t, defined as

$$w_{\boldsymbol{k}} = \frac{1}{t}\int_{\tau=0}^{t} \phi_{\boldsymbol{k}}\big(\boldsymbol{x}(\tau)\big)\, d\tau, \tag{143}$$

will match the Fourier series coefficients $\hat{w}_{\boldsymbol{k}}$ on the spatial domain $\mathcal{X}$, defined as

$$\hat{w}_{\boldsymbol{k}} = \int_{\boldsymbol{x}\in\mathcal{X}} \hat{g}(\boldsymbol{x})\, \phi_{\boldsymbol{k}}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}. \tag{144}$$

The discretized version of (143) for a discrete number of time steps T is

$$w_{\boldsymbol{k}} = \frac{1}{T}\sum_{s=1}^{T} \phi_{\boldsymbol{k}}(\boldsymbol{x}_s), \tag{145}$$

or equivalently in vector form $\boldsymbol{w} = \frac{1}{T}\sum_{s=1}^{T} \boldsymbol{\phi}(\boldsymbol{x}_s)$, which can be computed recursively by updating $\boldsymbol{w}$ at each iteration step.

Similarly, the discretized version of (144) for a given spatial resolution can for example be computed on a domain $[0, 1]$ discretized in X bins for each dimension as

$$\hat{w}_{\boldsymbol{k}} = \sum_{s_1=1}^{X}\sum_{s_2=1}^{X}\cdots\sum_{s_d=0}^{X} \hat{g}(\boldsymbol{x_s})\, \phi_{\boldsymbol{k}}(\boldsymbol{x_s}), \tag{146}$$

where each dimension i of $\boldsymbol{x}$ is given by index s_i splitting each domain dimension into X equal bins.

The goal of ergodic control is to minimize

$$\epsilon = \frac{1}{2}\sum_{\boldsymbol{k}\in\mathcal{K}} \Lambda_{\boldsymbol{k}}\Big(w_{\boldsymbol{k}} - \hat{w}_{\boldsymbol{k}}\Big)^2 \tag{147}$$

$$= \frac{1}{2}\Big(\boldsymbol{w} - \hat{\boldsymbol{w}}\Big)^{\top} \boldsymbol{\Lambda}\, \Big(\boldsymbol{w} - \hat{\boldsymbol{w}}\Big), \tag{148}$$

where $\Lambda_{\boldsymbol{k}}$ are weights, $\hat{w}_{\boldsymbol{k}}$ are the Fourier series coefficients of $\hat{g}(\boldsymbol{x})$, and $w_{\boldsymbol{k}}$ are the Fourier series coefficients along the trajectory $\boldsymbol{x}(t)$. $\boldsymbol{w} \in \mathbb{R}^{K^D}$ and $\hat{\boldsymbol{w}} \in \mathbb{R}^{K^D}$ are vectors composed of elements $w_{\boldsymbol{k}}$ and $\hat{w}_{\boldsymbol{k}}$, respectively. $\boldsymbol{\Lambda} \in \mathbb{R}^{K^D \times K^D}$ is a diagonal weighting matrix with elements $\Lambda_{\boldsymbol{k}}$. In (147), the weights

$$\Lambda_{\boldsymbol{k}} = \big(1 + \|\boldsymbol{k}\|^2\big)^{-\frac{D+1}{2}} \tag{149}$$

assign more importance on matching low frequency components (related to a metric for Sobolev spaces of negative order).

Practically, the controller will mainly focus on the lowest frequency components at the beginning as these components have a large effect on the cost due to the large weights. When the errors become zero for these components, the controller can then try to progressively reduce higher frequency components.

For a spatial signal $\boldsymbol{x} \in \mathbb{R}^D$, where x_d is on the interval $[-\frac{L}{2}, \frac{L}{2}]$ of period L, $\forall d \in \{1, \dots, D\}$, the basis functions of the Fourier series with complex exponential functions are defined as (see Fig. 42-*(e)*)

$$\phi_{\boldsymbol{k}}(\boldsymbol{x}) = \frac{1}{L^D}\prod_{d=1}^{D} \cos\left(\frac{2\pi k_d x_d}{L}\right), \quad \forall \boldsymbol{k} \in \mathcal{K}. \tag{150}$$

By concatenating the K^D index vectors $\boldsymbol{k} \in \mathcal{K}$ into a matrix $\tilde{\boldsymbol{V}} \in \mathbb{R}^{D \times K^D}$, we can conveniently compute $\boldsymbol{\phi}(\boldsymbol{x})$ as the vector concatenating all K^D elements $\phi_{\boldsymbol{k}}(\boldsymbol{x})$, $\forall \boldsymbol{k} \in \mathcal{K}$, namely

$$\boldsymbol{\phi}(\boldsymbol{x}) = \frac{1}{L^D} \bigodot_{d=1}^{D} \cos(\boldsymbol{V}_d x_d) \tag{151}$$

$$= \frac{1}{L^D} \cos(\boldsymbol{V}_1 x_1) \odot \cos(\boldsymbol{V}_2 x_2) \odot \cdots \odot \cos(\boldsymbol{V}_D x_D), \quad \text{with} \quad \boldsymbol{V} = \frac{2\pi \tilde{\boldsymbol{V}}}{L}, \tag{152}$$

where $\odot$ the elementwise/Hadamard product operator.

The derivatives of the vector $\boldsymbol{\phi}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ are given by the matrix $\nabla_{\boldsymbol{x}} \boldsymbol{\phi}(\boldsymbol{x})$ of size $K^D \times D$ by concatenating horizontally the D vectors

$$\nabla_{\boldsymbol{x}} \boldsymbol{\phi}_d(\boldsymbol{x}) = -\frac{1}{L^D} \boldsymbol{V}_d \odot \sin(\boldsymbol{V}_d x_d) \bigodot_{i \neq d} \cos(\boldsymbol{V}_i x_i) \tag{153}$$

$$= -\frac{1}{L^D} \boldsymbol{V}_d \odot \cos(\boldsymbol{V}_1 x_1) \odot \cdots \odot \cos(\boldsymbol{V}_{d-1} x_{d-1}) \odot \sin(\boldsymbol{V}_d x_d) \odot \cos(\boldsymbol{V}_{d+1} x_{d+1}) \odot \cdots \odot \cos(\boldsymbol{V}_D x_D). \tag{154}$$

For the specific example of a 2D space, (152) and (154) yield

$$\boldsymbol{\phi}(\boldsymbol{x}) = \frac{1}{L^2} \cos(\boldsymbol{V}_1 x_1) \odot \cos(\boldsymbol{V}_2 x_2), \tag{155}$$

$$\nabla_{\boldsymbol{x}} \boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} -\frac{1}{L} \boldsymbol{V}_1 \odot \sin(\boldsymbol{V}_1 x_1) \odot \cos(\boldsymbol{V}_2 x_2) & -\frac{1}{L} \boldsymbol{V}_2 \odot \cos(\boldsymbol{V}_1 x_1) \odot \sin(\boldsymbol{V}_2 x_2) \end{bmatrix}. \tag{156}$$

Ergodic control is set as the constrained problem of computing a control command $\hat{\boldsymbol{u}}(t)$ at each time step t with

$$\hat{\boldsymbol{u}}(t) = \arg\min_{\boldsymbol{u}(t)} \epsilon\big(\boldsymbol{x}(t+\Delta t)\big), \quad \text{s.t.} \quad \dot{\boldsymbol{x}}(t) = f\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), \quad \|\boldsymbol{u}(t)\| \leqslant u^{\max}, \tag{157}$$

where the simple system $\boldsymbol{u}(t) = \dot{\boldsymbol{x}}(t)$ is considered (control with velocity commands), and where the error term is approximated with the Taylor series

$$\epsilon\big(\boldsymbol{x}(t+\Delta t)\big) \;\approx\; \epsilon\big(\boldsymbol{x}(t)\big) \;+\; \dot{\epsilon}\big(\boldsymbol{x}(t)\big)\Delta t \;+\; \frac{1}{2}\ddot{\epsilon}\big(\boldsymbol{x}(t)\big)\Delta t^2. \tag{158}$$

By using (147), (143), (150) and the chain rule $\frac{\partial f}{\partial t} = \frac{\partial f}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial t}$, the Taylor series is composed of the control term $\boldsymbol{u}(t)$ and $\nabla_{\boldsymbol{x}} \phi_{\boldsymbol{k}}\big(\boldsymbol{x}(t)\big) \in \mathbb{R}^{1 \times D}$, the gradient of $\phi_{\boldsymbol{k}}\big(\boldsymbol{x}(t)\big)$ with respect to $\boldsymbol{x}(t)$. Solving the constrained objective in (157) then results in the analytical solution (see [10] for the complete derivation)

$$\boldsymbol{u}(t) = \tilde{\boldsymbol{u}}(t) \frac{u^{\max}}{\|\tilde{\boldsymbol{u}}(t)\|}, \quad \text{with} \quad \tilde{\boldsymbol{u}}(t) = -\nabla_{\boldsymbol{x}} \boldsymbol{\phi}(\boldsymbol{x})^{\top} \boldsymbol{\Lambda} \big(\boldsymbol{w} - \hat{\boldsymbol{w}}\big). \tag{159}$$

Figures 41 and 42 show a 2D example of ergodic control to create a motion approximating the distribution given by a mixture of two Gaussians. A remarkable characteristic of such approach is that the controller produces natural exploration behaviors without relying on stochastic noise in the formulation, see Fig. 42-*(c)*. In the limit case, if the distribution $g(\boldsymbol{x})$ is a single Gaussian with a very small isotropic covariance, the controller results in a conventional target reaching behavior.

Figure 43 presents the output of the accompanying source codes, where the initial guess is computed by standard SMC (myopic), whose resulting control command trajectory is then refined with iLQR (SMC as a planning problem).

10.2 Diffusion-based ergodic control (HEDAC)

`ergodic_control_HEDAC_1D.*`
`ergodic_control_HEDAC_2D.*`

Heat equation driven area coverage (HEDAC) is another area coverage method that relies on diffusion processes instead of spectral analysis [6].

The coverage density currently covered by the agent(s) is defined as the convolution of an instantaneous action and the trajectory, which results in a field occupying the space around the path of the agent. A radial basis function can for example be used as the instantaneous action.

The algorithm designs a potential field based on the heat equation by using a source term constructed as the difference between the given goal density and the current coverage density. The movements of the agent(s) are directed by the gradient of this potential field, leading them to the area of interest and producing paths to achieve the desired coverage density. The diffusion of the heat allows the agent(s) to get access to this gradient, bringing them to the regions of the target coverage density that have not been explored yet.

HEDAC considers a target distribution $p(\boldsymbol{x})$ to be covered, and the time-dependent coverage $c(\boldsymbol{x}, t)$ that one or multiple moving agents have already covered. The coverage residual $e(\boldsymbol{x}, t)$ is defined as the scalar field

$$e(\boldsymbol{x}, t) = p(\boldsymbol{x}) - c(\boldsymbol{x}, t), \tag{160}$$

which is used to build a virtual heat source with nonnegative values

$$\tilde{s}(\boldsymbol{x}, t) = \max\left(e(\boldsymbol{x}, t), 0\right)^2. \tag{161}$$

For proper scaling, the heat source is further normalized over the domain using

$$s(\boldsymbol{x}, t) = \frac{\tilde{s}(\boldsymbol{x}, t)}{\int_\Omega \tilde{s}(\boldsymbol{x}, t) dx}. \tag{162}$$

The heat source is then diffused over the domain to propagate the information about the unexplored regions to the agent(s). For that purpose, we use the heat equation: a second-order partial differential equation (PDE) that models heat conduction by relating spatial and time derivatives of a scalar field with

$$\frac{\partial u(\boldsymbol{x}, t)}{\partial t} = \alpha \Delta u(\boldsymbol{x}, t) + s(\boldsymbol{x}, t), \tag{163}$$

where $u(\boldsymbol{x}, t)$ corresponds to the temperature field, α is a thermal diffusivity parameter, and $\Delta u = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + \ldots$ is the Laplacian of the function u.

In order to control the agents, the potential field exerts a fictitious force on each of the i-th agent, based on the gradient of the temperature field

$$\boldsymbol{f}_i = \boldsymbol{\nabla} u(\boldsymbol{x}_i, t). \tag{164}$$

Neumann boundary conditions can be imposed (corresponding to thermal insulation), which means that information propagation stays within the domain boundary with

$$\frac{\partial u}{\partial \mathbf{n}} = 0, \quad \text{on} \quad \partial\Omega, \tag{165}$$

where Ω is a n-dimensional domain with Lipschitz continuous boundary.

At each iteration step of the algorithm, we compute the coverage of the agent(s), which cool the temperature field and mark the regions they cover by updating the coverage distribution $c_i(\boldsymbol{x}, t)$. For the i-th agent, the coverage is computed by the relation

$$c_i(\boldsymbol{x}, t) = \int_0^t \phi(\boldsymbol{x} - \boldsymbol{x}_i(\tau)) d\tau, \tag{166}$$

where $\phi(\boldsymbol{x}_i)$ is the footprint or shape of the virtual agent (arbitrary shape). For example, a Gaussian radial basis functions (RBF) with adjustable shape parameter ϵ can be used, given by

$$\phi(\boldsymbol{x}) = e^{-(\epsilon \boldsymbol{x})^2}. \tag{167}$$

The total coverage containing the exploration effort of all the agents is then computed with

$$\tilde{c}(\boldsymbol{x}, t) = \frac{1}{Nt} \sum_{i=1}^{N} c_i(\boldsymbol{x}, t), \tag{168}$$

that is further normalized over the domain using (162) to provide $c(\boldsymbol{x}, t)$.

11 Torque-controlled robots

Robot manipulators can sometimes be controlled by using torque commands, allowing us to consider varying stiffness and compliance behaviors when the robot is in contact with objects, users or its environment. To design efficient controllers that can make use of this capability, we will typically need to take into account the various kinds of forces that will act on the robot, which will involve various notions of physics and mechanics. We will start this section with the description of a computed torque control approach that we can use either in joint space or in task space, which will be described generically as impedance controllers in the next two sections. In a nutshell, impedance control is an approach to dynamic control relating force and position. It is often used for applications in which the robot manipulator physically interacts with its environment.

11.1 Impedance control in joint space

`impedance_control.*`

A torque-controlled robot can be expressed by the dynamical model

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{x})\ddot{\boldsymbol{x}} + \boldsymbol{c}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \boldsymbol{\tau}_{\text{ext}}, \tag{169}$$

where $\boldsymbol{\tau}$ is the actuation torque (the input variable to the robot), $\boldsymbol{M}(\boldsymbol{x})$ is the inertia matrix (symmetric positive-definite matrix), $\boldsymbol{c}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \boldsymbol{C}(\boldsymbol{x}, \dot{\boldsymbol{x}})\,\dot{\boldsymbol{x}}$ is the Coriolis and centrifugal torque, $\boldsymbol{g}(\boldsymbol{x})$ is the gravitational torque, $\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}})$ is the friction torque, $\boldsymbol{\tau}_{\text{ext}}$ is the torque resulting from external forces from the environment.

In order to best exploit the compliant control capability of such robots, the joint torque commands $\boldsymbol{\tau}$ that we provide most often need to compensate for the various physical effects that are applied to the robot. Removing these effects mean that we try to estimate what these effects produce at the level of the torques by simulating what effects will be produce in the physical world, which include (but is not limited to) gravity, inertial forces and friction. In many applications in robot manipulation, the gravity will have the strongest effect that we will try to compensate.

The estimation of these effects involve physical parameters such as the center of mass of each robot link, the mass of each robot link, the distribution of mass as an inertia matrix, etc. Robot manufacturers sometimes provide these values (which can for example be estimated from CAD models of the robot). There are otherwise approaches to estimate these parameters during a calibration phase. We will assume that we have access to these values. The estimated model parameters/functions of our robot will be denoted as $\hat{\boldsymbol{M}}(\boldsymbol{x})$, $\hat{\boldsymbol{c}}(\boldsymbol{x}, \dot{\boldsymbol{x}})$, $\hat{\boldsymbol{g}}(\boldsymbol{x})$ and $\hat{\boldsymbol{h}}(\boldsymbol{x}, \dot{\boldsymbol{x}})$, which might be close to the real physical values $\boldsymbol{M}(\boldsymbol{x})\ddot{\boldsymbol{x}}$, $\boldsymbol{c}(\boldsymbol{x}, \dot{\boldsymbol{x}})$, $\boldsymbol{g}(\boldsymbol{x})$ and $\boldsymbol{h}(\boldsymbol{x}, \dot{\boldsymbol{x}})$, but not necessarily the same.

We will consider a desired reference $\{\boldsymbol{x}^d, \dot{\boldsymbol{x}}^d, \ddot{\boldsymbol{x}}^d\}$ that we generically described as position, velocity and acceleration components.

By using the estimated model parameters of our robot, we can design a control law to reach the desired reference as

$$\boldsymbol{\tau} = \boldsymbol{K}^{JP}(\boldsymbol{x}^d - \boldsymbol{x}) + \boldsymbol{K}^{JV}(\dot{\boldsymbol{x}}^d - \dot{\boldsymbol{x}}) + \hat{\boldsymbol{M}}(\boldsymbol{x})\,\ddot{\boldsymbol{x}}^d + \hat{\boldsymbol{c}}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \hat{\boldsymbol{g}}(\boldsymbol{x}) + \hat{\boldsymbol{h}}(\boldsymbol{x}, \dot{\boldsymbol{x}}), \tag{170}$$

where $\boldsymbol{K}^{JP}$ and $\boldsymbol{K}^{JV}$ are stiffness and damping matrices in joint space.

If we apply this control law to our robot, which will affected by the different physical effects of the physical world as described in (169), we will obtain a closed-loop system (controlled robot) of the form

$$\boldsymbol{K}^{JP}\boldsymbol{e} + \boldsymbol{K}^{JV}\dot{\boldsymbol{e}} + \boldsymbol{M}\ddot{\boldsymbol{e}} = \boldsymbol{\tau}_{\text{ext}}, \tag{171}$$

with error term $\boldsymbol{e} = \boldsymbol{x}^d - \boldsymbol{x}$. This closed-loop system is simply obtained by inserting our designed control commands (170) into (169). Thus, with the proposed control law, we can see with (171) that the controlled robot acts as a mechanical impedance to the environment, corresponding to a mass-spring-damper system in joint space.

11.2 Impedance control in task space

The same principle can also be applied to task space by expressing all parts composing the dynamical model of (169) in the endeffector coordinate system.

Since we used the term *joint torques* in the previous section to refer to force commands at the joint angle level, we will use the term *wrench* to refer to the forces at the level of the endeffector in task space. In the most generic case, the wrench $\boldsymbol{w}$ will be a 6D force vector by considering both translation and rotational parts (3D for each). The wrench $\boldsymbol{w}$ applied to the endeffector will then produce reaction torques at joint level, with $\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{w}$, corresponding to the principle of virtual work.

By using the relations $\dot{\boldsymbol{f}} = \boldsymbol{J}(\boldsymbol{x})\dot{\boldsymbol{x}}$ and $\ddot{\boldsymbol{f}} = \dot{\boldsymbol{J}}(\boldsymbol{x})\dot{\boldsymbol{x}} + \boldsymbol{J}(\boldsymbol{x})\ddot{\boldsymbol{x}} \approx \boldsymbol{J}(\boldsymbol{x})\ddot{\boldsymbol{x}}$, we can see that $\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{x})\,\ddot{\boldsymbol{x}}$ in the joint coordinate system becomes $\boldsymbol{w} = \boldsymbol{\Lambda}(\boldsymbol{x})\,\ddot{\boldsymbol{f}}$, with $\boldsymbol{\Lambda}(\boldsymbol{x}) = \left(\boldsymbol{J}(\boldsymbol{x})\,\boldsymbol{M}(\boldsymbol{x})^{-1}\boldsymbol{J}(\boldsymbol{x})^\top\right)^{-1}$ in the endeffector coordinate system.[2]

Similarly to (170), we can then define a control law as

$$\boldsymbol{\tau} = \boldsymbol{J}(\boldsymbol{x})^\top \left(\boldsymbol{K}^P(\boldsymbol{f}^d - \boldsymbol{f}) + \boldsymbol{K}^V(\dot{\boldsymbol{f}}^d - \dot{\boldsymbol{f}}) + \hat{\boldsymbol{\Lambda}}(\boldsymbol{x})\,\ddot{\boldsymbol{f}}^d\right) + \hat{\boldsymbol{c}}(\boldsymbol{x}, \dot{\boldsymbol{x}}) + \hat{\boldsymbol{g}}(\boldsymbol{x}) + \hat{\boldsymbol{h}}(\boldsymbol{x}, \dot{\boldsymbol{x}}), \tag{172}$$

with

$$\hat{\boldsymbol{\Lambda}}(\boldsymbol{x}) = \left(\boldsymbol{J}(\boldsymbol{x})\,\hat{\boldsymbol{M}}(\boldsymbol{x})^{-1}\boldsymbol{J}(\boldsymbol{x})^\top\right)^{-1}. \tag{173}$$

The controlled robot then acts as a mechanical impedance corresponding to a mass-spring-damper system in task space.

[2] We can see this with $\boldsymbol{J}^\top\boldsymbol{w} = \boldsymbol{M}\ddot{\boldsymbol{x}} \iff \boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{w} = \ddot{\boldsymbol{x}} \iff \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top\boldsymbol{w} = \boldsymbol{J}\ddot{\boldsymbol{x}} \iff \boldsymbol{w} = (\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top)^{-1}\ddot{\boldsymbol{f}}$.

11.3 Forward dynamics for a planar robot manipulator and associated control strategy

`FD.*` `iLQR_manipulator_dynamics.*`

The dynamic equation of a planar robot with an arbitrary number of links can be derived using the Lagrangian formulation, by representing the kinetic and potential energies of the robot as functions of the joint angles. The dynamic equation can be used in the context of iLQR to control a robot with torque commands, see the provided examples and Appendix E for computation details.

12 Orientation representations and Riemannian manifolds

Representing orientation is crucial in robotics. For example, controlling the 3D position of a gripper with respect to the 3D position of an object is not enough for a successful grasp with a robot manipulator. Instead, we need to consider the full 6D poses of the object and the gripper, which require to consider both position and orientation to describe the location in space of the robot's end-effector.

There are many ways to represent orientations, including rotation matrices, Euler angles, axis-angle, unit quaternions. Each representation has pros and cons that often depend on what we want to do with these data. It sometimes requires us to consider multiple representations in the same application, by converting one representation into another.

Rotation matrices

Rotations in the three-dimensional space can be represented by 3 rotation matrices, i.e., by means of 9 parameters. These parameters, however, are not independent, but constrained by orthonormal conditions (its columns are orthogonal unit vectors). The main advantage of rotation matrices is that geometric operation can be directly computed with standard linear algebra, by multiplying matrices with vectors. The main disadvantage is that it involves the use of 9 parameters that are not 9 independent parameters but that are instead constrained to 3 independent parameters.

For human analysis, the columns of a rotation matrix correspond to a set of 3 unit vectors forming the base of a coordinate system, where each vector is orthogonal to the other. By plotting these 3 vectors, we can visualize the corresponding orientation that the rotation matrix encodes.

Euler angles

A minimal representation of rotations in space requires three independent parameters, which is fulfilled by Euler angles. With Euler angles, a rotation in space is viewed as a sequence of three elementary rotations. The sequence ZYX of Euler angles correspond for example to yaw-pitch-roll angles. There are 16 different sequences, some being more used than others. When using Euler angles, in addition to the three values, it is thus important to specify which sequence has been used, because the use of these different conventions can easily introduce mistakes.

The main advantage of Euler angles is that they are easy to interpret for human analysis in a compact form, where the rotation can be mentally interpreted by applying the three orientations in sequence. This might be an advantage over the rotation matrix, that often requires the corresponding coordinate system to be visualized on a screen to be interpreted.

One disadvantage of Euler angles is that they are not easy to compose, but there is another more important disadvantage that drastically limits the use of Euler angles for computation in learning and optimization problems. With Euler angles, for a given orientation, the 3 angles are uniquely determined except for singular cases when some of the considered axes have the same or opposite directions. These ambiguities are known as gimbal locks. A gimbal is a ring that is suspended so it can rotate about an axis. Gimbals are nested one within another to accommodate rotation about multiple axes. Gimbal lock refers to the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of 2 of the 3 gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space. For example, in a pitch/yaw/roll rotational system, if the pitch is rotated 90 degrees up or down, the yaw and roll will correspond to the same motion and a degree of freedom is lost.

Axis-angle

Axis-angle is a representation of orientation that defines a unit axis $\boldsymbol{u}$ around which a rotation is made with angle θ. By rescaling the unit axis by the rotation angle, we can obtain a representation with 3 parameters. The rotation vector is not well defined for $\theta = 0$.

Unit quaternions

The unit quaternion is a nonminimal representation of rotations which shares some similarities with the axis-angle representation but that overcomes its limitations. A unit quaternion can be stored as a vector of 4 elements, but it can also be interpreted as a single number, in the same way as a complex number can be stored as a vector of two

elements with the real and imaginary part. Indeed, the quaternion is an extension of complex numbers (with $i^2 = -1$) to additional abstract symbols i, j, k satisfying the rules

$$i^2 = j^2 = k^2 = ijk = -1. \tag{174}$$

Unit quaternions can be used to represent orientations. Similarly to the rotation matrix, this representation avoids the gimbal lock issue, but with a more compact representation (4 instead of 9 parameters).

While the quaternion describes a number, it is convenient in practice to store the parameters of the quaternion number $q_1 + q_2 i + q_3 j + q_4 k$ as a vector

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}. \tag{175}$$

The conversion from an axis-angle representation is simple and shows the connections between the two representations. For a unit axis $\boldsymbol{u}$ and a rotation angle θ, we have

$$\boldsymbol{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \boldsymbol{u}\sin\frac{\theta}{2} \end{bmatrix}. \tag{176}$$

The main motivation of working with quaternions (which can be viewed as an advantage or as a disadvantage) is that they come with a specific algebra that allows to fully leverage the quaternion parameterization. For example, in the quaternion algebra, a quaternion rotation can be defined as $\boldsymbol{p}' = \boldsymbol{q}\boldsymbol{p}\boldsymbol{q}^{-1}$, which can be computed efficiently. This has led unit quaternions to be the most widely used representation in computer graphics, for example.

Conversions between quaternion algebra and linear algebra are possible. For example, the above quaternion rotation corresponds to the matrix rotation $\boldsymbol{p}' = \boldsymbol{R}\boldsymbol{p}$ in standard linear algebra, where $\boldsymbol{R}$ is a rotation matrix given by

$$\boldsymbol{R} = \begin{bmatrix} 1-2q_3^2-2q_4^2 & 2q_2q_3-2q_4q_1 & 2q_2q_4+2q_3q_1 \\ 2q_2q_3+2q_4q_1 & 1-2q_2^2-2q_4^2 & 2q_3q_4-2q_2q_1 \\ 2q_2q_4-2q_3q_1 & 2q_3q_4+2q_2q_1 & 1-2q_2^2-2q_3^2 \end{bmatrix}. \tag{177}$$

A rotation matrix $\boldsymbol{R}$ can reversely be converted to a quaternion $\boldsymbol{q}$ by constructing a symmetric 4×4 matrix

$$K = \frac{1}{3}\begin{bmatrix} R_{11}-R_{22}-R_{33} & R_{21}+R_{12} & R_{31}+R_{13} & R_{23}-R_{32} \\ R_{21}+R_{12} & R_{22}-R_{11}-R_{33} & R_{32}+R_{23} & R_{31}-R_{13} \\ R_{31}+R_{13} & R_{32}+R_{23} & R_{33}-R_{11}-R_{22} & R_{12}-R_{21} \\ R_{23}-R_{32} & R_{31}-R_{13} & R_{12}-R_{21} & R_{11}+R_{22}+R_{33} \end{bmatrix}, \tag{178}$$

whose largest eigenvalue corresponds to $\boldsymbol{q}$.

By storing the four elements of a unit quaternion as a vector $\boldsymbol{q}$, we can interpret $\boldsymbol{q}$ as a point on a (hyper)sphere of 4 dimensions. Indeed, the norm of 1 induces this geometry. One way to interpret and visualize this easily is to consider the corresponding problem of a 3D vector constrained to have a unit norm, which means that the points lie on a 3D sphere of radius 1. Such unit 3D vectors for example correspond to the representation of direction vectors in robotics (e.g., to represent surface normals for environment modeling). These representations form a manifold that can leverage the exploitation of Riemannian geometry, which is discussed next.

12.1 Riemannian manifolds

Data in robotics are essentially geometric. Riemannian manifolds can facilitate the processing of these geometric data, which can be applied to a wide range of learning and optimization problems in robotics. In particular, it provides a principled and simple way to extend algorithms initially developed for Euclidean data to other manifolds, by efficiently taking into account prior geometric knowledge about these manifolds.

A smooth d-dimensional manifold $\mathcal{M}$ is a topological space that locally behaves like the Euclidean space $\mathbb{R}^d$. A *Riemannian manifold* is a smooth and differentiable manifold equipped with a positive definite metric tensor. For each point $\boldsymbol{p} \in \mathcal{M}$, there exists a tangent space $\mathcal{T}_{\boldsymbol{p}}\mathcal{M}$ that locally linearizes the manifold. On a Riemannian manifold, the metric tensor induces a positive definite inner product on each tangent space $\mathcal{T}_{\boldsymbol{p}}\mathcal{M}$, which allows vector lengths and angles between vectors to be measured. The affine connection, computed from the metric, is a differential operator that provides, among other functionalities, a way to compute geodesics and to transport vectors on tangent spaces along any smooth curves on the manifold. It also fully characterizes the intrinsic curvature and torsion of the manifold. The Cartesian product of two Riemannian manifolds is also a Riemannian manifold (often called manifold bundles or manifold composites), which allows joint distributions to be constructed on any combination of Riemannian manifolds.

Riemannian geometry is a rich field of research. For applications in robotics, two main notions of Riemannian geometry are crucial:

Geodesics: The minimum length curves between two points on a Riemannian manifold are called geodesics. Similarly

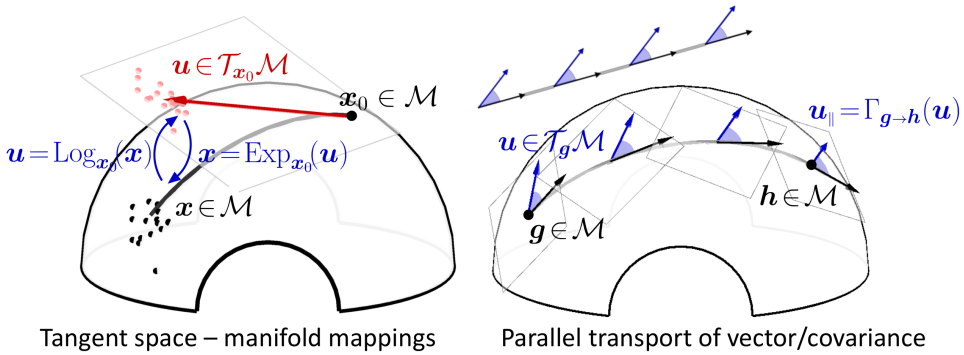


Figure 44: Applications in robotics using Riemannian manifolds rely on two well-known principles of Riemannian geometry: exponential/logarithmic mapping (*left*) and parallel transport (*right*), which are depicted here on a $\mathcal{S}^2$ manifold embedded in $\mathbb{R}^3$. *Left:* Bidirectional mappings between tangent space and manifold. *Right:* Parallel transport of a vector along a geodesic (see main text for details).

to straight lines in Euclidean space, the second derivative is zero everywhere along a geodesic. The exponential map $\mathrm{Exp}_{\boldsymbol{x}_0} : \mathcal{T}_{\boldsymbol{x}_0}\mathcal{M} \to \mathcal{M}$ maps a point $\boldsymbol{u}$ in the tangent space of $\boldsymbol{x}_0$ to a point $\boldsymbol{x}$ on the manifold, so that $\boldsymbol{x}$ lies on the geodesic starting at $\boldsymbol{x}_0$ in the direction $\boldsymbol{u}$. The norm of $\boldsymbol{u}$ is equal to the geodesic distance between $\boldsymbol{x}_0$ and $\boldsymbol{x}$. The inverse map is called the logarithmic map $\mathrm{Log}_{\boldsymbol{x}_0} : \mathcal{M} \to \mathcal{T}_{\boldsymbol{x}_0}\mathcal{M}$. Figure 44-*left* depicts these mapping functions.

Parallel transport: Parallel transport $\Gamma_{\boldsymbol{g}\to\boldsymbol{h}} : \mathcal{T}_{\boldsymbol{g}}\mathcal{M} \to \mathcal{T}_{\boldsymbol{h}}\mathcal{M}$ moves vectors between tangent spaces such that the inner product between two vectors in a tangent space is conserved. It employs the notion of connection, defining how to associate vectors between infinitesimally close tangent spaces. This connection allows the smooth transport of a vector from one tangent space to another by sliding it (with infinitesimal moves) along a curve. Figure 44-*right* depicts this operation. The flat surfaces show the coordinate systems of several tangent spaces along the geodesic. The black vectors represent the directions of the geodesic in the tangent spaces. The blue vectors are transported from $\boldsymbol{g}$ to $\boldsymbol{h}$. Parallel transport allows a vector $\boldsymbol{u}$ in the tangent space of $\boldsymbol{g}$ to be transported to the tangent space of $\boldsymbol{h}$, by ensuring that the angle (i.e., inner product) between $\boldsymbol{u}$ and the direction of the geodesic (represented as black vectors) are conserved. At point $\boldsymbol{g}$, this direction is expressed as $\mathrm{Log}_{\boldsymbol{g}}(\boldsymbol{h})$. This operation is crucial to combine information available at $\boldsymbol{g}$ with information available at $\boldsymbol{h}$, by taking into account the rotation of the coordinate systems along the geodesic. In Euclidean space (top-left inset), the parallel transport is simply the identity operator (a vector operation can be applied to any point without additional transformation). By extension, a covariance matrix $\boldsymbol{\Sigma}$ can be transported with $\boldsymbol{\Sigma}_\parallel = \sum_{i=1}^{d} \Gamma_{\boldsymbol{g}\to\boldsymbol{h}}(\boldsymbol{v}_i)\, \Gamma_{\boldsymbol{g}\to\boldsymbol{h}}^\top(\boldsymbol{v}_i)$, using the eigendecomposition $\boldsymbol{\Sigma} = \sum_{i=1}^{d} \boldsymbol{v}_i\, \boldsymbol{v}_i^\top$. For many manifolds in robotics, this transport operation can equivalently be expressed as a linear mapping $\boldsymbol{\Sigma}_\parallel = \boldsymbol{A}_{\boldsymbol{g}\to\boldsymbol{h}}\, \boldsymbol{\Sigma}\, \boldsymbol{A}_{\boldsymbol{g}\to\boldsymbol{h}}^\top$

The most common manifolds in robotics are homogeneous, with the same curvature at any point on the manifold, which provides simple analytic expressions for exponential/logarithmic mapping and parallel transport.

12.2 Sphere manifolds $\mathcal{S}^d$ in robotics

The sphere manifold $\mathcal{S}^d$ is the most widely used non-Euclidean manifold in robotics, as it can encode direction and orientation information. Unit quaternions $\mathcal{S}^3$ can be used to represent endeffector (tool tip) orientations. $\mathcal{S}^2$ can be used to represent unit directional vector perpendicular to surfaces (e.g., for contact planning). Revolute joints can be represented on the torus $\mathcal{S}^1 \times \mathcal{S}^1 \times \ldots \times \mathcal{S}^1$.

The exponential and logarithmic maps corresponding to the distance

$$d(\boldsymbol{x}, \boldsymbol{y}) = \arccos(\boldsymbol{x}^\top \boldsymbol{y}), \tag{179}$$

with $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}^d$ computed as

$$\boldsymbol{y} = \mathrm{Exp}_{\boldsymbol{x}}(\boldsymbol{u}) = \boldsymbol{x}\cos(\|\boldsymbol{u}\|) + \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}\sin(\|\boldsymbol{u}\|), \tag{180}$$

$$\boldsymbol{u} = \mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}) = d(\boldsymbol{x}, \boldsymbol{y})\, \frac{\boldsymbol{y} - \boldsymbol{x}^\top \boldsymbol{y}\, \boldsymbol{x}}{\|\boldsymbol{y} - \boldsymbol{x}^\top \boldsymbol{y}\, \boldsymbol{x}\|}. \tag{181}$$

The parallel transport of $\boldsymbol{v} \in \mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$ to $\mathcal{T}_{\boldsymbol{y}}\mathcal{S}^d$ is given by

$$\Gamma_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{v}) = \boldsymbol{v} - \frac{\mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y})^\top \boldsymbol{v}}{d(\boldsymbol{x}, \boldsymbol{y})^2}\Big(\mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}) + \mathrm{Log}_{\boldsymbol{y}}(\boldsymbol{x})\Big). \tag{182}$$

In some applications, it can be convenient to define the parallel transport with the alternative equivalent form

$$\begin{aligned}&\Gamma_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{v}) = \boldsymbol{A}_{\boldsymbol{x}\to\boldsymbol{y}}\,\boldsymbol{v}, \quad \text{with} \\ &\boldsymbol{A}_{\boldsymbol{x}\to\boldsymbol{y}} = -\boldsymbol{x}\sin(\|\boldsymbol{u}\|)\overline{\boldsymbol{u}}^\top + \overline{\boldsymbol{u}}\cos(\|\boldsymbol{u}\|)\overline{\boldsymbol{u}}^\top + (\boldsymbol{I} - \overline{\boldsymbol{u}}\,\overline{\boldsymbol{u}}^\top), \\ &\boldsymbol{u} = \mathrm{Log}_{\boldsymbol{x}}(\boldsymbol{y}), \quad \text{and} \quad \overline{\boldsymbol{u}} = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|},\end{aligned} \tag{183}$$

highlighting the linear structure of the operation.

Note that in the above representation, $\boldsymbol{u}$ and $\boldsymbol{v}$ are described as vectors with $d+1$ elements contained in $\mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$. An alternative representation consists of expressing $\boldsymbol{u}$ and $\boldsymbol{v}$ as vectors of d elements in the coordinate system attached to $\mathcal{T}_{\boldsymbol{x}}\mathcal{S}^d$.

12.3 Gaussian distributions on Riemannian manifolds

Several approaches can be used to extend Gaussian distributions to Riemannian manifolds. The most simple approach consists of estimating the mean of the Gaussian as a centroid on the manifold (also called Karcher/Fréchet mean), and representing the dispersion of the data as a covariance expressed in the tangent space of the mean [3]. This is an approximation, because distortions arise when points are too far apart from the mean. However, this distortion is negligible in most robotics applications. In particular, this effect is strongly attenuated when the Gaussian covers a small part of the manifold. This Gaussian representation on a manifold $\mathcal{M}$ is defined as

$$\mathcal{N}_{\mathcal{M}}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \left((2\pi)^d|\boldsymbol{\Sigma}|\right)^{-\frac{1}{2}} e^{-\frac{1}{2}\mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x})^\top\,\boldsymbol{\Sigma}^{-1}\,\mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x})},$$

where $\boldsymbol{x}\in\mathcal{M}$ is a point of the manifold, $\boldsymbol{\mu}\in\mathcal{M}$ is the mean of the distribution (origin of the tangent space), and $\boldsymbol{\Sigma}\in\mathcal{T}_{\boldsymbol{\mu}}\mathcal{M}$ is the covariance defined in this tangent space.

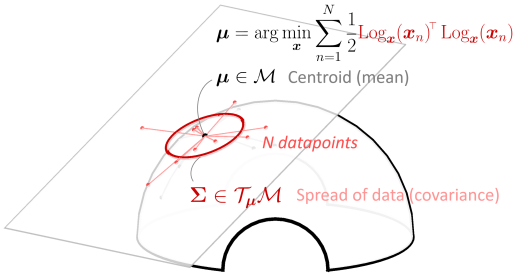


Figure 45: Gaussian distribution, where the center of the data is described by a point on the manifold and the spread of the data is described by a covariance matrix in the tangent space at this point.

For a set of N datapoints, this geometric mean corresponds to the minimization

$$\min_{\boldsymbol{\mu}} \sum_{n=1}^{N} \mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n)^\top\,\mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n),$$

which can be solved by a simple and fast Gauss–Newton iterative algorithm. The algorithm starts from an initial estimate on the manifold and an associated tangent space. The datapoints $\{\boldsymbol{x}_n\}_{n=1}^N$ are projected in this tangent space to compute a direction vector, which provides an updated estimate of the mean. This process is repeated by iterating

$$\boldsymbol{u} = \frac{1}{N}\sum_{n=1}^{N} \mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n), \qquad \boldsymbol{\mu} \leftarrow \mathrm{Exp}_{\boldsymbol{\mu}}(\boldsymbol{u}),$$

until convergence. In practice, for homogeneous manifolds with constant curvatures, this iterative algorithm will converge very fast, with only a couple of iterations.

After convergence, a covariance is computed in the tangent space as $\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N} \mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n)\,\mathrm{Log}_{\boldsymbol{\mu}}(\boldsymbol{x}_n)^\top$, see Fig. 45.

This distribution can for example be used in a control problem to represent a reference to track with an associated required precision (e.g., learned from a set of demonstrations). Importantly, this geometric mean can be directly extended to weighted distances, which will be exploited in the next sections for mixture modeling, fusion (product of Gaussians), regression (Gaussian conditioning) and planning problems.

12.4 Other homogeneous manifolds in robotics

Figure 46 shows four examples of homogeneous Riemannian manifolds that can be employed in robotics. For these four manifolds, the bottom graphs depict $\mathcal{S}^2$, $\mathcal{S}^2_{++}$, $\mathcal{H}^2$ and $\mathcal{G}^{3,2}$, with a clustering problem in which the datapoints (black dots/planes) are segmented in two classes, each represented by a center (red and blue dots/planes).

The geodesics depicted in Fig. 46 show the specificities of each manifold, see [3] for details. The sphere manifold $\mathcal{S}^d$ is characterized by constant positive curvature. The elements of $\mathcal{S}^d_{++}$ can be represented as the interior of a convex cone embedded in its tangent space Sym^d. Here, the three axes correspond to A_{11}, A_{12} and A_{22} in the SPD matrix $\left(\begin{smallmatrix} A_{11} & A_{12} \\ A_{12} & A_{22}\end{smallmatrix}\right)$. The hyperbolic manifold $\mathcal{H}^d$ is characterized by constant negative curvature. Several representations exist: $\mathcal{H}^2$ can for example be represented as the interior of a unit disk in Euclidean space, with the boundary of the disk representing infinitely remote point (Poincaré disk model, as depicted here). In this model, geodesic paths are arcs of circles intersecting the boundary perpendicularly. $\mathcal{G}^{d,p}$ is the Grassmann manifold of all p-dimensional subspaces of $\mathbb{R}^d$.

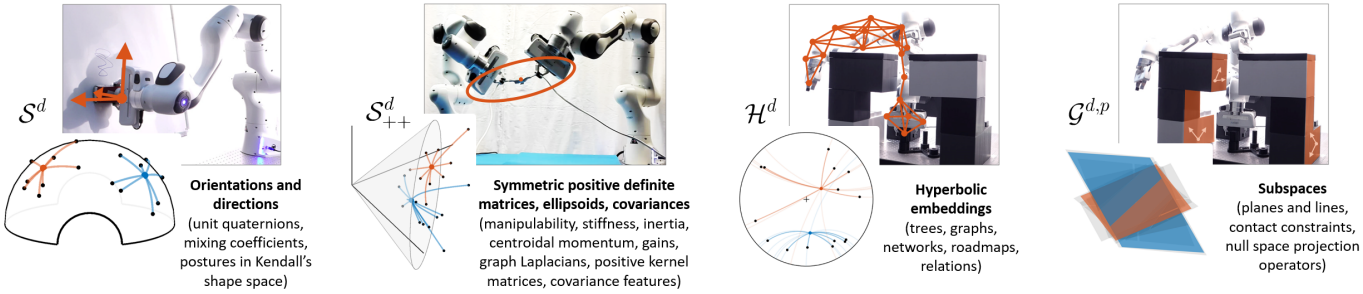


Figure 46: Examples of manifolds relevant for robotics.

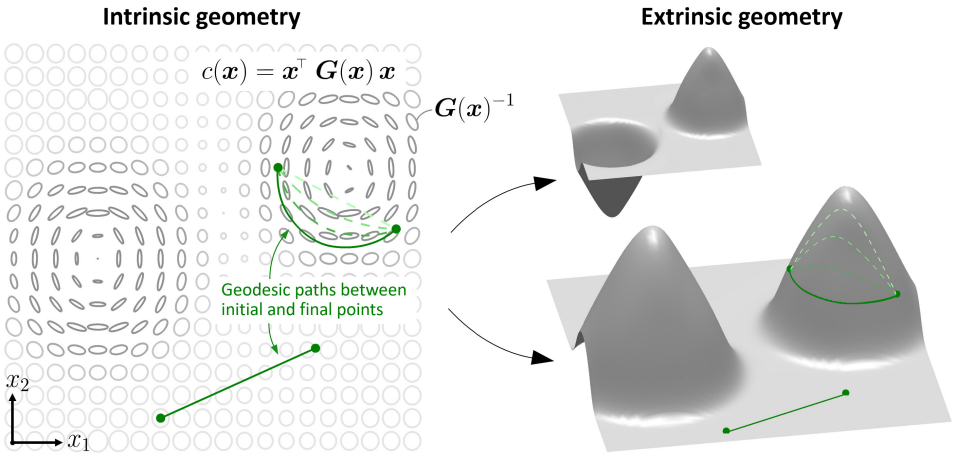


Figure 47: Intrinsic geometry defined by a Riemannian metric, with two examples of corresponding extrinsic geometries (here, embedded in a 3D space).

12.5 Ellipsoids and SPD matrices as $\mathcal{S}^d_{++}$ manifolds

Symmetric positive definite (SPD) matrices are widely used in robotics. They can represent stiffness, manipulability and inertia ellipsoids. They can be used as weight matrices to compute distances. They can also be used to process sensory data in the form of spatial covariances. For example, to represent the robot's manipulability capability in translation and rotation can be encoded as a manipulability ellipsoid $\mathcal{S}^6_{++}$.

For an affine-invariant distance between $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{S}^d_{++}$

$$d(\boldsymbol{X}, \boldsymbol{Y}) = \left\| \log(\boldsymbol{X}^{-\frac{1}{2}} \boldsymbol{Y} \boldsymbol{X}^{-\frac{1}{2}}) \right\|_{\mathrm{F}}, \tag{184}$$

the exponential and logarithmic maps on the SPD manifold can be computed as

$$\boldsymbol{Y} = \mathrm{Exp}_{\boldsymbol{X}}(\boldsymbol{U}) = \boldsymbol{X}^{\frac{1}{2}} \exp\left(\boldsymbol{X}^{-\frac{1}{2}} \boldsymbol{U} \boldsymbol{X}^{-\frac{1}{2}}\right) \boldsymbol{X}^{\frac{1}{2}}, \tag{185}$$

$$\boldsymbol{U} = \mathrm{Log}_{\boldsymbol{X}}(\boldsymbol{Y}) = \boldsymbol{X}^{\frac{1}{2}} \log\left(\boldsymbol{X}^{-\frac{1}{2}} \boldsymbol{Y} \boldsymbol{X}^{-\frac{1}{2}}\right) \boldsymbol{X}^{\frac{1}{2}}. \tag{186}$$

The parallel transport of $\boldsymbol{V} \in \mathcal{T}_{\boldsymbol{X}} \mathcal{S}^d_{++}$ to $\mathcal{T}_{\boldsymbol{Y}} \mathcal{S}^d_{++}$ is given by

$$\Gamma_{\boldsymbol{X} \to \boldsymbol{Y}}(\boldsymbol{V}) = \boldsymbol{A}_{\boldsymbol{X} \to \boldsymbol{Y}}\, \boldsymbol{V}\, \boldsymbol{A}^\top_{\boldsymbol{X} \to \boldsymbol{Y}}, \text{ with } \boldsymbol{A}_{\boldsymbol{X} \to \boldsymbol{Y}} = \boldsymbol{Y}^{\frac{1}{2}} \boldsymbol{X}^{-\frac{1}{2}}. \tag{187}$$

12.6 Non-homogeneous manifolds in robotics

Manifolds with nonconstant curvature can also be employed, such as spaces endowed with a metric, characterized by a weighting matrix used to compute distances. Many problems in robotics can be formulated with such a smoothly varying matrix $\boldsymbol{G}(\boldsymbol{x})$ that can for example be used to evaluate displacements $\Delta\boldsymbol{x}$ as a quadratic error term $c(\Delta\boldsymbol{x}) = \Delta\boldsymbol{x}^\top \boldsymbol{G}(\boldsymbol{x}) \Delta\boldsymbol{x}$, forming a Riemannian metric that describes the underlying manifold (with non-homogeneous curvature). This weighting matrix can for example represent levels of kinetic energy or stiffness gains to model varying impedance behaviors. Computation is often more costly for these manifolds with nonconstant curvature, because it typically requires iterative algorithms instead of the direct analytic expressions typically provided by homogeneous manifolds.

Figures 47 and 48 presents examples exploiting non-homogeneous Riemannian manifolds.

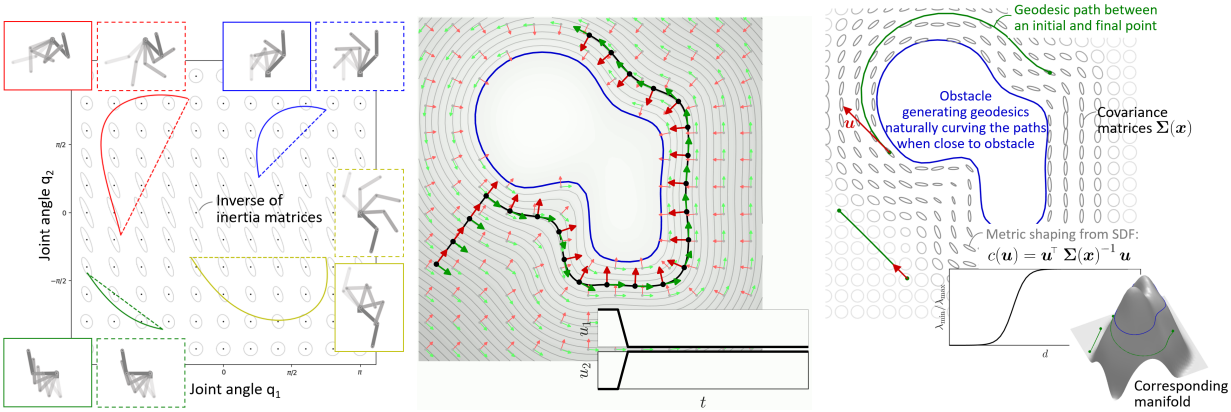


Figure 48: *Left:* A typical example of smoothly varying metric in robotics is when the weight matrix is a mass inertia matrix so that inertia is taken into account to generate a movement from a starting joint configuration to a target joint configuration while minimizing kinetic energy (see geodesics in solid lines). In the figure, inverse weight matrices are depicted to visualize locally the equidistant points (w.r.t the metric) that the system can reach as ellipsoids. The movement in dashed lines show the baseline movements that would be produced by ignoring inertia (corresponding to linear interpolation between the two poses). The metric can similarly be weighted by other precision matrices, including stiffness and manipulability ellipsoids. *Center:* Transformation of a scalar signed distance field (SDF), as in Figure 14, and its associated derivatives to a smoothly varying coordinate system to facilitate teleoperation. This coordinate system is shape-centric (both user-centric and object-centric), in the sense that it is oriented toward the surface normal while ensuring smoothness (by constructing the SDF as piecewise polynomials). In this example, the commands to approach the surface and move along the surface at a desired distance then corresponds to piecewise constant commands (see control command profiles in the graph inset). *Right:* This coordinate system can be used to define a cost function with a quadratic error defined by a covariance matrix. This matrix can be shaped by the distance, to provide a Riemannian metric that smoothly distorts distances when being close to objects, users and obstacles (represented as covariance ellipses in the figure). This can be used in control and planning to define geodesic paths based on this metric. This will generate paths that naturally curve around obstacles when the obstacles are close (see the two point-to-point trajectories in green, either far from the obstacle or close to the obstacle). The figure inset shows that the metric can also be interpreted as a surface lying in a higher dimensional space (here, in 3D), where the geodesics then correspond to the shortest Euclidean path on this surface. It is relevant to note here that we don't need to know about this corresponding embedding, which is for some problems hard to construct), and that we instead only need the metric to compute geodesics on the Riemannian manifold.

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. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit geometric regularization for learning shapes. In *Proc. Intl Conf. on Machine Learning (ICML)*, pages 3789–3799, 2020.

[5] 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.

[6] S. Ivić, B. Crnković, and I. Mezić. Ergodicity-based cooperative multiagent area coverage via a potential field. *IEEE Trans. on Cybernetics*, 47(8):1983–1993, 2017.

[7] 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.

[8] 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.

[9] 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.

[10] G. Mathew and I. Mezic. Spectral multiscale coverage: A uniform coverage algorithm for mobile sensor networks. In *Proc. IEEE Conf. on Decision and Control*, pages 7872–7877, Dec 2009.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] S. Shetty, J. Silvério, and S. Calinon. Ergodic exploration using tensor train: Applications in insertion tasks. *IEEE Trans. on Robotics*, 38(2):906–921, 2022.

[19] 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

$$\underbrace{\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \vdots\\ \boldsymbol{x}_T\end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix}\boldsymbol{I}\\ \boldsymbol{A}_1\\ \boldsymbol{A}_2\boldsymbol{A}_1\\ \vdots\\ \prod_{t=1}^{T-1}\boldsymbol{A}_{T-t}\end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{x}}}\boldsymbol{x}_1 + \underbrace{\begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{B}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{A}_2\boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots\\ \left(\prod_{t=1}^{T-2}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_1 & \left(\prod_{t=1}^{T-3}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_2 & \cdots & \boldsymbol{B}_{T-1}\end{bmatrix}}_{\boldsymbol{S}_{\boldsymbol{u}}}\underbrace{\begin{bmatrix}\boldsymbol{u}_1\\ \boldsymbol{u}_2\\ \vdots\\ \boldsymbol{u}_{T-1}\end{bmatrix}}_{\boldsymbol{u}}, \tag{188}$$

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 (194) 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 (194), 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 (194), 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 (194) 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.[3]

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{189}
$$

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{190}
$$

[3] 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{\begin{bmatrix}\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{bmatrix}}_{\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{\begin{bmatrix}\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{bmatrix}}_{\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{191}$$

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

$$\boldsymbol{A}_t = \boldsymbol{I} + \boldsymbol{A}_t^c\Delta t, \quad \boldsymbol{B}_t = \boldsymbol{B}_t^c\Delta t, \quad \forall t \in \{1,\dots,T\}, \tag{192}$$

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

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

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

E Forward dynamics (FD) for a planar robot manipulator

The dynamic equation of a planar robot with an arbitrary number of links can be derived using the Lagrangian formulation, by representing the kinetic and potential energies of the robot as functions of generalized coordinates, which are joint angles in this case. For simplicity, we will assume that all masses are located at the end of each link, and we will neglect the terms that contain rotational energies (i.e, zero moments of inertia). To do this, we exploit the formulation derived in Section 4 without the orientation part, so the position of the j-*th* mass can be written as

$$\boldsymbol{f}_j = \begin{bmatrix}f_{j,1}\\ f_{j,2}\end{bmatrix} = \begin{bmatrix}\sum_{k=1}^{j} l_k c_k\\ \sum_{k=1}^{j} l_k s_k\end{bmatrix}, \quad \text{where} \quad c_k = \cos(\sum_{i=1}^{k} x_i), \quad s_k = \sin(\sum_{i=1}^{k} x_i), \tag{193}$$

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 easier to express with absolute angles $\sum_{i=1}^{k} x_i$ instead of relative angles 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

$$E_{\text{lag}} = \sum_{j=1}^{N} T_j - U_j,$$

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

$$u_z = \frac{d}{dt}\frac{\partial E_{\text{lag}}}{\partial \dot{q}_z} - \frac{\partial E_{\text{lag}}}{\partial q_z} \quad = \frac{d}{dt}\frac{\partial (\sum_{j=z}^{N} T_j)}{\partial \dot{q}_z} - \frac{\partial \sum_{j=z}^{N} (T_j - U_j)}{\partial q_z} \quad = \sum_{j=z}^{N} \Big(\frac{d}{dt}\frac{\partial T_j}{\partial \dot{q}_z} - \frac{\partial T_j}{\partial q_z} + \frac{\partial U_j}{\partial q_z} \Big), \tag{194}$$

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

$$w_z = \begin{cases} (\tau_z - \tau_{z+1})\dot{q}_z & z < N \\ \tau_N \dot{q}_N & z = N \end{cases}, \tag{195}$$

where τ_z is the torque applied at z-*th* joint. The fact that we need subtraction in the first line of (195) 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 (194), 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)

$$\sum_{j=z}^{N} m_j \Big(\sum_{k=1}^{j} l_z l_k c_{z-k} \ddot{q}_k \Big) = u_z - \sum_{j=z}^{N} m_j \Big(\sum_{k=1}^{j} l_z l_k s_{z-k} \dot{q}_k^2 \Big) - \sum_{j=z}^{N} m_j g l_z c_z, \quad \text{where} \quad \begin{cases} c_{h-k} = c_h c_k - s_h s_k, \\ s_{h-k} = s_h c_k - c_h s_k. \end{cases} \tag{196}$$

In (196), 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

$$\tilde{\boldsymbol{M}} \ddot{\boldsymbol{q}} + \tilde{\boldsymbol{g}} + \tilde{\boldsymbol{C}} \text{diag}(\dot{\boldsymbol{q}}) \dot{\boldsymbol{q}} = \boldsymbol{u}, \tag{197}$$

where

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

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

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 (197), we would have

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{g} + \boldsymbol{C}\text{diag}(\boldsymbol{L}\dot{\boldsymbol{x}})\boldsymbol{L}\dot{\boldsymbol{x}} = \boldsymbol{\tau}, \tag{198}$$

where

$$\boldsymbol{M} = \boldsymbol{L}^{\top}\tilde{\boldsymbol{M}}\boldsymbol{L}, \quad \boldsymbol{C} = \boldsymbol{L}^{\top}\tilde{\boldsymbol{C}}, \quad \boldsymbol{g} = \boldsymbol{L}^{\top}\tilde{\boldsymbol{g}}$$

Please note that elements in $\tilde{\boldsymbol{M}}, \tilde{\boldsymbol{C}}$ and $\tilde{\boldsymbol{g}}$ 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 $\boldsymbol{L}$.

iLQR for a robot manipulator with dynamics equation

The nonlinear dynamic equation of a planar robot presented in the above can be expressed as

$$\begin{bmatrix} \boldsymbol{q}_{t+1} \\ \dot{\boldsymbol{q}}_{t+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t dt \\ \dot{\boldsymbol{q}}_t + \tilde{\boldsymbol{M}}^{-1}\Big(\boldsymbol{u} - \tilde{\boldsymbol{g}} - \tilde{\boldsymbol{C}}\text{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big)dt \end{bmatrix}. \tag{199}$$

For simplicity we start with (197), and at the end, we will describe how the result can be converted to the other choice of variables. The iLQR method requires (199) to be linearized around the current states and inputs.The corresponding $\boldsymbol{A}_t$ and $\boldsymbol{B}_t$ can be found according to (82) as

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

$\boldsymbol{A}_{21} = \frac{\partial \ddot{\boldsymbol{q}}_t}{\partial \boldsymbol{q}_t}$ can be found by taking the derivation of (197) 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-th joint angle at time t q_t^p which can be formulated as

$$\frac{\partial \ddot{\boldsymbol{q}}_t}{\partial q_t^p} = \frac{\partial \tilde{\boldsymbol{M}}^{-1}}{\partial q_t^p}\Big(\boldsymbol{u} - \tilde{\boldsymbol{g}} - \tilde{\boldsymbol{C}}\text{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big) - \tilde{\boldsymbol{M}}^{-1}\Big(\frac{\partial \tilde{\boldsymbol{g}}}{\partial q_t^p} + \frac{\partial \tilde{\boldsymbol{C}}}{\partial q_t^p}\text{diag}(\dot{\boldsymbol{q}}_t)\dot{\boldsymbol{q}}_t\Big).$$

Having this done for all p at time t, we can write

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

where

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

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

$$(\tilde{\boldsymbol{M}}^{-1})' = \begin{bmatrix} \frac{\partial \tilde{\boldsymbol{M}}^{-1}}{\partial q_t^1} & \frac{\partial \tilde{\boldsymbol{M}}^{-1}}{\partial q_t^2} & \cdots & \frac{\partial \tilde{\boldsymbol{M}}^{-1}}{\partial q_t^N} \end{bmatrix}, \quad \Big(\tilde{\boldsymbol{M}}^{-1}\Big)' = -\tilde{\boldsymbol{M}}^{-1}\tilde{\boldsymbol{M}}'\tilde{\boldsymbol{M}}^{-1},$$

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

According to Section 8.1, we can concatenate the results for all time steps as

$$\begin{bmatrix} \Delta \boldsymbol{q}_1 \\ \Delta \dot{\boldsymbol{q}}_1 \\ \vdots \\ \Delta \boldsymbol{q}_T \\ \Delta \dot{\boldsymbol{q}}_T \end{bmatrix} = \boldsymbol{S}_u^q \begin{bmatrix} \Delta \boldsymbol{u}_1 \\ \Delta \boldsymbol{u}_2 \\ \vdots \\ \Delta \boldsymbol{u}_{T-1} \end{bmatrix}, \tag{200}$$

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

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

so (200) can be rewritten as

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