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

Sylvain Calinon, Idiap Research Institute

1 Introduction

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

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

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

2 Movement primitives

. MP.*

The term *movement primitives* refers 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. 1. This principle, coined in the context of motor control [?], remains valid for a wide range of continuous time signals (for both analysis and synthesis).

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

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

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

$$\boldsymbol{x} = \sum_{k=1}^{K} \boldsymbol{\Psi}_k \, w_k = \boldsymbol{\Psi} \, \boldsymbol{w}, \tag{2}$$

$$\text{with } \boldsymbol{\Psi} = \boldsymbol{\phi} \otimes \boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}\phi_{1,1} & \boldsymbol{C}\phi_{2,1} & \cdots & \boldsymbol{C}\phi_{K,1} \\ \boldsymbol{C}\phi_{1,2} & \boldsymbol{C}\phi_{2,2} & \cdots & \boldsymbol{C}\phi_{K,2} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{C}\phi_{1,T} & \boldsymbol{C}\phi_{2,T} & \cdots & \boldsymbol{C}\phi_{K,T} \end{bmatrix},$$

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

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

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

Piecewise constant basis functions

Piecewise constant basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \boldsymbol{I}_K \otimes \boldsymbol{1}_{\frac{T}{K}}, \tag{3}$$

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

Radial basis functions (RBFs)

Gaussian radial basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \exp(-\lambda \, \boldsymbol{E} \odot \boldsymbol{E}), \quad \text{with} \quad \boldsymbol{E} = \boldsymbol{t} \, \boldsymbol{1}_K^\top - \boldsymbol{1}_T \, \boldsymbol{\mu}^{\text{s}\top}, \tag{4}$$

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, $\boldsymbol{\mu}^{\text{s}} \in \mathbb{R}^K$ 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. 1.

Bernstein basis functions

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

$$\boldsymbol{\phi}_k = \frac{(K-1)!}{(k-1)!(K-k)!} \, (\boldsymbol{1}_T - \boldsymbol{t})^{K-k} \odot \boldsymbol{t}^{k-1}, \tag{5}$$

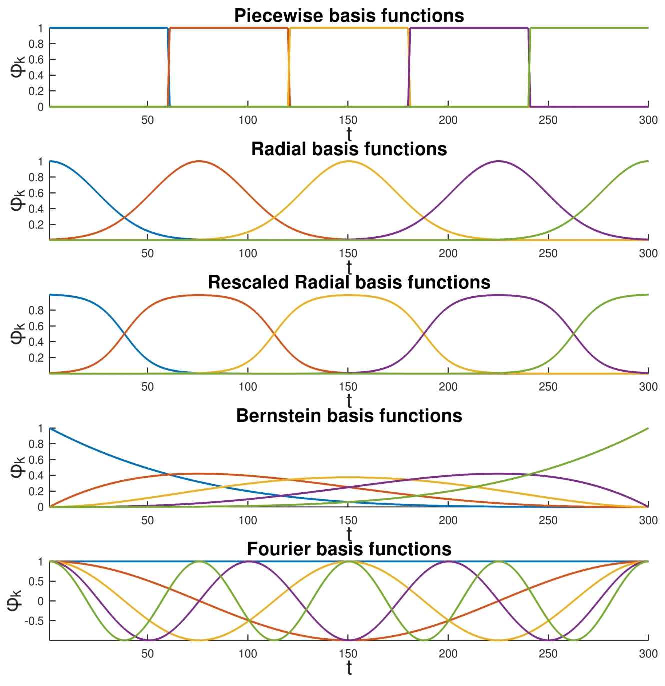
$\forall k \in \{1, \ldots, K\}$, where $\boldsymbol{t} \in \mathbb{R}^T$ is a vector with entries linearly spaced between 0 to 1, and $(\cdot)^d$ is applied elementwise, see Fig. 1.

Fourier basis functions

Fourier basis functions can be computed in matrix form as

$$\boldsymbol{\phi} = \exp(\boldsymbol{t} \, \tilde{\boldsymbol{k}}^\top 2\pi i), \tag{6}$$

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, $\tilde{\boldsymbol{k}} = [-K+1, -K+2, \ldots, K-2, K-1]^\top$, and i is the imaginary unit ($i^2 = -1$).



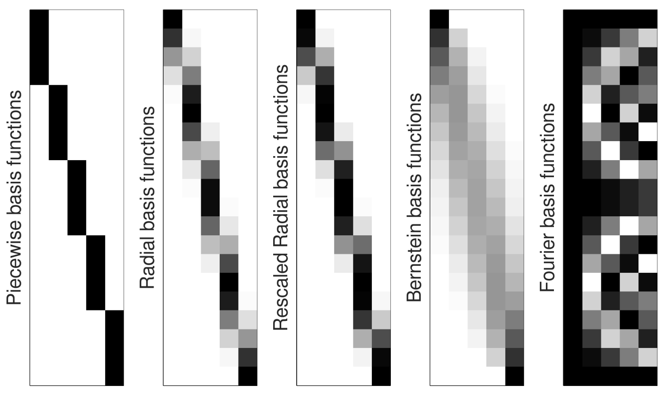
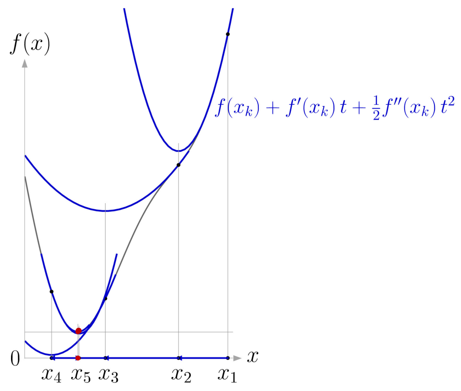


Figure 1: Examples of basis functions. *top:* Representation in timeline form for $K = 5$ and $T = 300$. *Bottom:* Representation in matrix form for $K = 5$ and $T = 20$, with a grayscale colormap where white pixels are 0 and black pixels are 1.



$f(x)$
$f(x_k) + f'(x_k)\,t + \frac{1}{2}f''(x_k)\,t^2$
0
x
x_4 x_5 x_3 x_2 x_1

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

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

$$\boldsymbol{\phi} = \cos(\boldsymbol{t}\,\boldsymbol{k}^\top 2\pi i), \tag{7}$$

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

Indeed, we first note that $\exp(ai)$ is composed of a real part and an imaginary part with $\exp(ai) = \cos(a) + i\sin(a)$. We can see that for a given time step t, a real state x_t can be constructed with the Fourier series

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

where we used the properties $\cos(0) = 1$ and $\cos(-a) = \cos(a)$ of the cosine function. Since we do not need direct correspondences between the Fourier transform and discrete cosine transform as in the above, we can omit the scaling factors for w_k and directly write the decomposition as in (7).

3 Newton's method for minimization

Newton's method attempts to solve $\min_x f(x)$ or $\max_x f(x)$ from an initial guess x_1 by using a sequence of second-order Taylor approximations of f around the iterates, see Fig. 2. The second-order Taylor expansion of f around x_k is

$$f(x_k + t) \approx f(x_k) + f'(x_k)\,t + \frac{1}{2}f''(x_k)\,t^2. \tag{8}$$

The next iterate $x_{k+1} = x_k + t$ is defined so as to minimize this quadratic approximation in t. If the second derivative is positive, the quadratic approximation is a convex function of t, and its minimum can be found by setting the derivative to zero. Since

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(f(x_k) + f'(x_k)\,t + \frac{1}{2}f''(x_k)\,t^2\right) = f'(x_k) + f''(x_k)\,t, \tag{9}$$

the minimum is achieved for

$$t = -\frac{f'(x_k)}{f''(x_k)}. \tag{10}$$

Newton's method thus performs the iteration

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{11}$$

The geometric interpretation of Newton's method is that at each iteration, it amounts to the fitting of a paraboloid to the surface of $f(x)$ 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 f happens to be a quadratic function, then the exact extremum is found in one step.

The multidimensional case similarly provides

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}(\boldsymbol{x}_k)^{-1}\,\boldsymbol{g}(\boldsymbol{x}_k), \tag{12}$$

with $\boldsymbol{g}$ and $\boldsymbol{H}$ the gradient and Hessian matrix of f (vector and square matrix, respectively).

3.1 Gauss-Newton algorithm

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

$$\boldsymbol{g} = 2\boldsymbol{J}_{\boldsymbol{r}}^\top\boldsymbol{r}, \quad \boldsymbol{H} \approx 2\boldsymbol{J}_{\boldsymbol{r}}^\top\boldsymbol{J}_{\boldsymbol{r}}, \tag{13}$$

where $\boldsymbol{J}_{\boldsymbol{r}} \in \mathbb{R}^{R\times D}$ is the Jacobian matrix of $\boldsymbol{r} \in \mathbb{R}^R$. The update rule then becomes

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \big(\boldsymbol{J}_{\boldsymbol{r}}^\top(\boldsymbol{x}_k)\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{x}_k)\big)^{-1}\,\boldsymbol{J}_{\boldsymbol{r}}^\top(\boldsymbol{x}_k)\,\boldsymbol{r}(\boldsymbol{x}_k) \quad (14)\\ &= \boldsymbol{x}_k - \boldsymbol{J}_{\boldsymbol{r}}^\dagger(\boldsymbol{x}_k)\,\boldsymbol{r}(\boldsymbol{x}_k). \quad (15)\end{aligned}$$

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

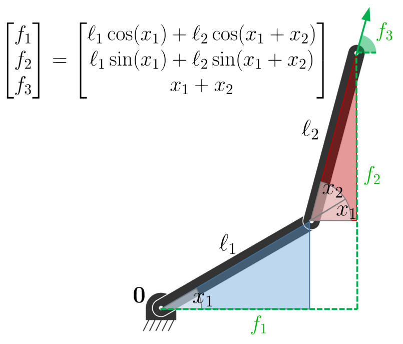
4 Forward kinematics (FK) for a planar robot manipulator

IK.*

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

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

with $\boldsymbol{x}$ the state of the robot (joint angles), $\boldsymbol{f}$ the position of the robot end-effector, $\boldsymbol{\ell}$ a vector of robot links lengths, $\boldsymbol{L}$ a lower triangular matrix with unit elements, and $\boldsymbol{1}$ a vector of unit elements, see Fig. 3.



$\begin{bmatrix}f_1\\f_2\\f_3\end{bmatrix} = \begin{bmatrix}\ell_1\cos(x_1)+\ell_2\cos(x_1+x_2)\\ \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2)\\ x_1+x_2\end{bmatrix}$
f_3
ℓ_2
f_2
x_2
x_1
ℓ_1
$\boldsymbol{0}$
x_1
f_1

Figure 3: Forward kinematics for a planar robot with two links.

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

$$\begin{aligned}\tilde{\boldsymbol{f}} &= \Big[\boldsymbol{L}\,\mathrm{diag}(\boldsymbol{\ell})\cos(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\,\mathrm{diag}(\boldsymbol{\ell})\sin(\boldsymbol{L}\boldsymbol{x}),\quad \boldsymbol{L}\boldsymbol{x}\Big]^\top\\ &= \begin{bmatrix}\tilde{f}_{1,1} & \tilde{f}_{1,2} & \tilde{f}_{1,3} & \dots\\ \tilde{f}_{2,1} & \tilde{f}_{2,2} & \tilde{f}_{2,3} & \dots\\ \tilde{f}_{3,1} & \tilde{f}_{3,2} & \tilde{f}_{3,3} & \dots\end{bmatrix},\end{aligned} \tag{16}$$

with

$$\begin{aligned}\tilde{f}_{1,1} &= \ell_1\cos(x_1),\\ \tilde{f}_{2,1} &= \ell_1\sin(x_1),\\ \tilde{f}_{3,1} &= x_1,\\[1ex] \tilde{f}_{1,2} &= \ell_1\cos(x_1)+\ell_2\cos(x_1+x_2),\\ \tilde{f}_{2,2} &= \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2),\\ \tilde{f}_{3,2} &= x_1+x_2,\\[1ex] \tilde{f}_{1,3} &= \ell_1\cos(x_1)+\ell_2\cos(x_1+x_2)+\ell_3\cos(x_1+x_2+x_3),\\ \tilde{f}_{2,3} &= \ell_1\sin(x_1)+\ell_2\sin(x_1+x_2)+\ell_3\sin(x_1+x_2+x_3),\\ \tilde{f}_{3,3} &= x_1+x_2+x_3,\\ &\vdots\end{aligned}$$

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

```python
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
```

5 Inverse kinematics (IK) for a planar robot manipulator

IK.*

The Jacobian corresponding to the end-effector forward kinematics function can be computed as (with a simplification for the orientation part by ignoring the manifold aspect)

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

with

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

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

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

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

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.

6 Linear quadratic tracking (LQT)

LQT.*

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

$$\begin{aligned} c &= \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big)^\top \boldsymbol{Q}_T \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big) \\ &\quad + \sum_{t=1}^{T-1} \Big(\big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big)^\top \boldsymbol{Q}_t \big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big) \; + \; \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t \Big) \\ &= \underbrace{(\boldsymbol{\mu} - \boldsymbol{x})^\top \boldsymbol{Q} (\boldsymbol{\mu} - \boldsymbol{x})}_{\text{tracking cost}} \; + \; \underbrace{\boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}}_{\text{control cost}}, \end{aligned} \tag{17}$$

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

The evolution of the system is linear, described by $\boldsymbol{x}_{t+1} = \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t$, yielding $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}$ at trajectory level, see Appendix A for details.

With open loop control commands organized as a vector $\boldsymbol{u}$, the solution of (17) subject to $\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}$ is analytic, given by

$$\hat{\boldsymbol{u}} = \big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \big(\boldsymbol{\mu} - \boldsymbol{S}_{\boldsymbol{x}} \boldsymbol{x}_1\big). \tag{18}$$

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)

$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{u}} = \big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1}, \tag{19}$$

which provides a probabilistic interpretation of LQT.

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

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

Example: Bimanual tennis serve

LQT_tennisServe.*

LQT can be used to solve a ballistic task mimicking a bimanual tennis serve problem. In this problem, a ball is thrown by one hand and then hit by the other, with the

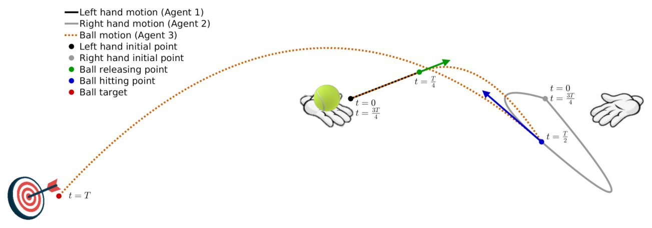


Figure 4: Tennis serve problem solved by linear quadratic tracking (LQT).

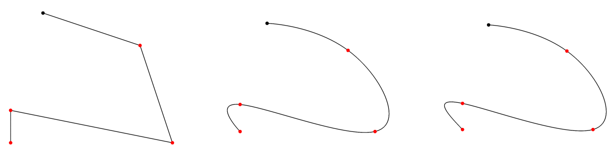


Figure 5: Examples of linear quadratic tracking (LQT) applied to the task of reaching a set of viapoints (red dots) by starting from an initial position (black dot). *Left:* With velocity commands, with a system described as a single integrator and a standard control cost. *Center:* With velocity commands, with a system described as a single integrator and a control cost ensuring smoothness. *Right:* With acceleration commands, with a system described as a double integrator and a standard control cost.

goal of bringing the ball to a desired target, see Fig. 4. The problem is formulated as in (17), namely

$$\min_{\boldsymbol{u}} \, (\boldsymbol{\mu}-\boldsymbol{x})^{\top} \boldsymbol{Q} (\boldsymbol{\mu}-\boldsymbol{x}) \;+\; \boldsymbol{u}^{\top}\boldsymbol{R}\boldsymbol{u}, \text{ s.t. } \boldsymbol{x}=\boldsymbol{S_x}\boldsymbol{x}_1+\boldsymbol{S_u}\boldsymbol{u}, \tag{20}$$

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 B for details. As shown above, the constrained objective in (20) can be solved by least squares, providing an analytical solution given by (18), see also Fig. 4 for the visualization of the result for a given target and for given initial poses of the hands.

6.1 LQT with smoothness cost

LQT.*

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

Figure 5 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

调整了控制 cost

matrix $\boldsymbol{R}$ of the control cost to ensure smoothness. This can be done by replacing the standard diagonal $\boldsymbol{R}$ matrix in (17) with

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

6.2 LQT with control primitives

LQT_CP.*

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

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

with a solution given by

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

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 (19), the residuals of the least squares solution provides a probabilistic approach to LQT with control primitives. Note also that the trajectory in control space can be converted to a trajectory in state space with $\hat{\boldsymbol{x}}=\boldsymbol{S_x}\boldsymbol{x}_1+\boldsymbol{S_u}\hat{\boldsymbol{u}}$. Thus, since all operations are linear, a covariance matrix on $\boldsymbol{w}$ can be converted to a covariance on $\boldsymbol{u}$ and on $\boldsymbol{x}$. A similar linear transformation is used in the context of probabilistic movement primitives (ProMP) [9] 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

$$\boldsymbol{\Psi}=\boldsymbol{\phi}\otimes\boldsymbol{C}, \tag{21}$$

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 $d\in\{1,\ldots,D\}$.

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.

what's this used for

6.3 LQR with a recursive formulation

LQT_recursive.*

The LQR problem is formulated as the minimization of the cost

$$c = \boldsymbol{x}_T^\top \boldsymbol{Q}_T \boldsymbol{x}_T + \sum_{t=1}^{T-1} \Big(\boldsymbol{x}_t^\top \boldsymbol{Q}_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top \boldsymbol{R}_t \, \boldsymbol{u}_t \Big). \tag{22}$$

By relabeling the final precision matrix $\boldsymbol{Q}_T$ as $\boldsymbol{P}_T$, the one-step cost-to-go c_{T-1} to move from $T-1$ to T corresponds to

$$c_{T-1} = \boldsymbol{x}_T^\top \boldsymbol{P}_T \boldsymbol{x}_T + \boldsymbol{x}_{T-1}^\top \boldsymbol{Q}_{T-1} \boldsymbol{x}_{T-1} + \boldsymbol{u}_{T-1}^\top \boldsymbol{R}_{T-1} \, \boldsymbol{u}_{T-1}. \tag{23}$$

By substituting $\boldsymbol{x}_T = \boldsymbol{A}\boldsymbol{x}_{T-1} + \boldsymbol{B}\boldsymbol{u}_{T-1}$ into (24), the cost-to-go can be rewritten as

$$c_{T-1} = \boldsymbol{x}_{T-1}^\top (\boldsymbol{A}^\top \boldsymbol{P}_T \boldsymbol{A} + \boldsymbol{Q}_{T-1}) \boldsymbol{x}_{T-1} + \\ 2\boldsymbol{x}_{T-1}^\top (\boldsymbol{A}^\top \boldsymbol{P}_T \boldsymbol{B}) \boldsymbol{u}_{T-1} + \boldsymbol{u}_{T-1}^\top (\boldsymbol{B}^\top \boldsymbol{P}_T \boldsymbol{B} + \boldsymbol{R}_{T-1}) \, \boldsymbol{u}_{T-1}. \tag{24}$$

An optimal control command $\hat{\boldsymbol{u}}_{T-1}$ can be computed by differentiating (24) with respect to $\boldsymbol{u}_{T-1}$ and equating to zero, providing a feedback law

$$\hat{\boldsymbol{u}}_{T-1} = -\boldsymbol{K}_{T-1} \, \boldsymbol{x}_{T-1}, \tag{25}$$
$$\text{with} \quad \boldsymbol{K}_{T-1} = \big(\boldsymbol{B}^\top \boldsymbol{P}_T \boldsymbol{B} + \boldsymbol{R}_{T-1}\big)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_T \boldsymbol{A}.$$

By introducing (25) back into (24), ~~we can see~~ that the resulting cost-to-go (also called value function) has the quadratic form

$$c_{T-1} = \boldsymbol{x}_{T-1}^\top \boldsymbol{P}_{T-1} \boldsymbol{x}_{T-1}, \tag{26}$$
$$\text{with} \quad \boldsymbol{P}_{T-1} = \boldsymbol{A}^\top \boldsymbol{P}_T \boldsymbol{A} \; + \; \boldsymbol{Q}_{T-1} \; - \\ \boldsymbol{A}^\top \boldsymbol{P}_T \boldsymbol{B} \big(\boldsymbol{B}^\top \boldsymbol{P}_T \boldsymbol{B} + \boldsymbol{R}_{T-1}\big)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_T \boldsymbol{A}.$$

We then consider the two-step problem to move from $T-2$ to T, with an associated cost c_{T-2} that can be decomposed as that of moving from $T-2$ to $T-1$, and from $T-1$ to T, which we computed in (26). The cost can then be described as

$$c_{T-2} = \boldsymbol{x}_{T-1}^\top \boldsymbol{P}_{T-1} \boldsymbol{x}_{T-1} + \\ \boldsymbol{x}_{T-2}^\top \boldsymbol{Q}_{T-2} \boldsymbol{x}_{T-2} + \boldsymbol{u}_{T-2}^\top \boldsymbol{R}_{T-2} \, \boldsymbol{u}_{T-2},$$

which has the same form as (23), so that we can also apply the expression found in (26), namely

$$c_{T-2} = \boldsymbol{x}_{T-2}^\top \boldsymbol{P}_{T-2} \boldsymbol{x}_{T-2}.$$

By applying the same reasoning to all time steps, we obtain a backward recursion procedure yielding a control policy in the form of a feedback law

$$\hat{\boldsymbol{u}}_{t-1} = -\boldsymbol{K}_{t-1} \, \boldsymbol{x}_{t-1}, \tag{27}$$

with feedback gain matrices

$$\boldsymbol{K}_{t-1} = \big(\boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{B} + \boldsymbol{R}_{t-1}\big)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{A}, \tag{28}$$

where $\boldsymbol{P}_t$ is evaluated recursively as

$$\boldsymbol{P}_{t-1} = \boldsymbol{A}^\top \boldsymbol{P}_t \boldsymbol{A} \; + \; \boldsymbol{Q}_{t-1} \; - \\ \boldsymbol{A}^\top \boldsymbol{P}_t \boldsymbol{B} \big(\boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{B} + \boldsymbol{R}_{t-1}\big)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{A}, \tag{29}$$

starting from $\boldsymbol{P}_T = \boldsymbol{Q}_T$. Equation (29) corresponds to a Riccati equation, and the overall procedure is an example of dynamic programming (DP).

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 $\boldsymbol{x}_1$, and using the linear system $\boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t$ with control policy $\boldsymbol{u}_t = -\boldsymbol{K}_t \boldsymbol{x}_t$.

6.4 LQT with a recursive formulation and an augmented state space

LQT_recursive.*

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

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{t+1} \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_{t+1}} = \underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}}_{\tilde{\boldsymbol{A}}} \underbrace{\begin{bmatrix} \boldsymbol{x}_t \\ 1 \end{bmatrix}}_{\tilde{\boldsymbol{x}}_t} + \underbrace{\begin{bmatrix} \boldsymbol{B} \\ 0 \end{bmatrix}}_{\tilde{\boldsymbol{B}}} \boldsymbol{u}_t, \tag{30}$$

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

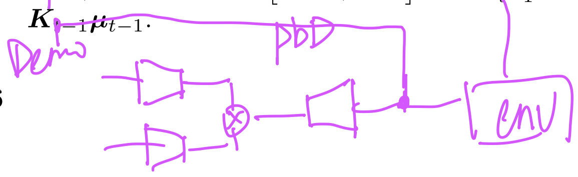
$$c = \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big)^\top \boldsymbol{Q}_T \big(\boldsymbol{\mu}_T - \boldsymbol{x}_T\big) \\ + \sum_{t=1}^{T-1} \Big(\big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big)^\top \boldsymbol{Q}_t \big(\boldsymbol{\mu}_t - \boldsymbol{x}_t\big) \; + \; \boldsymbol{u}_t^\top \boldsymbol{R}_t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

where the augmented form in (31) has the same form as the standard LQR cost in (22), 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 C.

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

$$\hat{\boldsymbol{u}}_{t-1} = -\tilde{\boldsymbol{K}}_{t-1} \, \tilde{\boldsymbol{x}}_{t-1} \\ = \boldsymbol{K}_{t-1} \big(\boldsymbol{\mu}_{t-1} - \boldsymbol{x}_{t-1}\big) + \boldsymbol{u}_{t-1}^{\text{ff}}, \tag{32}$$

characterized by a feedback gain and a feedforward term, with $\tilde{\boldsymbol{K}}_{t-1} = \big[\boldsymbol{K}_{t-1}, \boldsymbol{k}_{t-1}\big]$ and $\boldsymbol{u}_{t-1}^{\text{ff}} = -\boldsymbol{k}_{t-1} - \boldsymbol{K}_{t-1}\boldsymbol{\mu}_{t-1}$.



6.5 Least squares formulation of recursive LQR

LQT_recursive_LS.*

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

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

whose solution is

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

corresponding to open loop control commands.

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

$$\min_{\boldsymbol{F}} \; \boldsymbol{x}^\top \boldsymbol{Q} \boldsymbol{x} + (-\boldsymbol{F}\boldsymbol{x}_1)^\top \boldsymbol{R} (-\boldsymbol{F}\boldsymbol{x}_1), \qquad \text{s.t.} \quad \boldsymbol{x} = (\boldsymbol{S}_{\boldsymbol{x}} - \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{F}) \boldsymbol{x}_1, \quad (33)$$

whose least squares solution is

$$\boldsymbol{F} = \big(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{x}}. \qquad (34)$$

$\boldsymbol{F}$ can then be used to iteratively reconstruct regulation gains $\boldsymbol{K}_t$, by starting from $\boldsymbol{K}_1 = \boldsymbol{F}_1$, $\boldsymbol{P}_1 = \boldsymbol{I}$, and by computing recursively

$$\begin{aligned} \boldsymbol{P}_t &= \boldsymbol{P}_{t-1} (\boldsymbol{A}_{t-1} - \boldsymbol{B}_{t-1} \boldsymbol{K}_{t-1})^{-1}, \\ \boldsymbol{K}_t &= \boldsymbol{F}_t \, \boldsymbol{P}_t, \end{aligned} \qquad (35)$$

which can then be used in a feedback controller as in (27).

It is straightforward to extend this least squares formulation of recursive LQR to linear quadratic tracking and use (32) as feedback controller on an augmented state space, since the recursive LQT problem can be transformed to a recursive LQR problem with an augmented state space representation (see Section 6.4).

This least squares formulation of LQT (LQT-LS) yields the same controller as with the standard recursive computation presented in Section 6.4. However, the linear form in (34) used by LQT-LS has several advantages. First, it allows the use of full precision matrices $\boldsymbol{Q}$, which will be demonstrated in the example below. It also allows to extend LQT-LS to the use of control primitives, which will be further discussed in Section 6.6. Moreover, it provides a nullspace structure that can be exploited in recursive LQR/LQT problems.

Example with the control of multiple agents

LQT_recursive_LS_multiAgents.*

Figure 6 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.

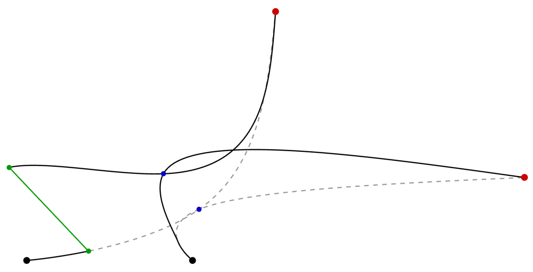


Figure 6: Least squares formulation of recursive LQR to control multiple agents (as point mass systems), where the task of each agent is to reach a desired target at the end of the motion, and to meet the other agent in the middle of the motion (e.g., for a handover task, see main text for the alternative 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.

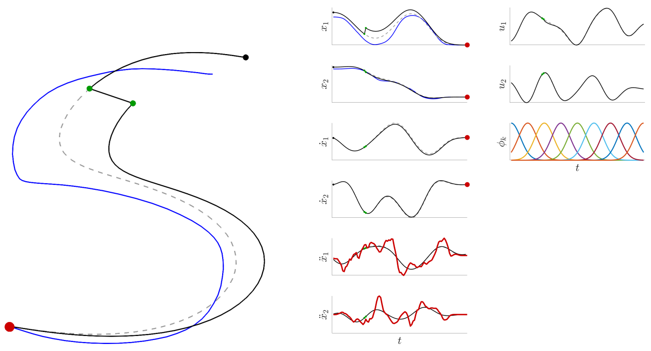


Figure 7: Linear quadratic tracking (LQT) with control primitives applied to a trajectory tracking task, with a formulation similar to dynamical movement primitives (DMP). *Left:* The observed "S" shape (in blue) is reproduced by starting from a different initial position (black point), with a perturbation simulated at 1/4 of the movement (green points) to show the capability of the approach to recover from perturbations. The trajectory in dashed line shows the result without perturbation and trajectory in solid line shows the result with perturbation. *Right:* The corresponding LQT problem formulation consists of requesting an end-point to be reached (red points) and an acceleration profile to be tracked (red lines), where the control commands $\boldsymbol{u}$ are represented as a superposition of radial basis functions $\boldsymbol{\phi}_k$.

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

LQT_CP_DMP.*

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

Linear quadratic tracking (LQT) with control primitives can be used in a similar fashion as in DMP, by requesting a target to be reached at the end of the movement and by requesting the observed acceleration profile to be tracked, while encoding the control commands as radial basis functions. The controller can be estimated either as the open-loop control commands (18), or as the closed-loop controller (32).

In the latter case, the matrix $\boldsymbol{F}$ in (34) is estimated by using control primitives and an augmented state space formulation, namely

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

which is used to compute feedback gains $\tilde{\boldsymbol{K}}_t$ on the augmented state with (35).

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

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

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

7 iLQR optimization

`iLQR_manipulator.*`

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 $\boldsymbol{x}_{t+1} = \boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ describing the evolution of a state $\boldsymbol{x}_t$ driven by control commands $\boldsymbol{u}_t$ during a time window of length T.

Iterative LQR (iLQR) [6] solves such constrained nonlinear models by carrying out Taylor expansions on the cost and on the dynamical system so that a solution can be found iteratively by solving a LQR problem at each iteration, similarly to differential dynamic programming approaches [8, 10].

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 3.1. A solution can alternatively be computed in a recursive form to provide a controller with feedback gains. We consider first the batch iLQR form, as we primarily focus on the planning capability of the approach.

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

$$\boldsymbol{x}_{t+1} \approx \boldsymbol{g}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_t}(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t) + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}_t}(\boldsymbol{u}_t - \hat{\boldsymbol{u}}_t)$$
$$\iff \Delta \boldsymbol{x}_{t+1} \approx \boldsymbol{A}_t \Delta \boldsymbol{x}_t + \boldsymbol{B}_t \Delta \boldsymbol{u}_t, \qquad (36)$$

with error terms $\{\Delta \boldsymbol{x}_t = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t, \Delta \boldsymbol{u}_t = \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t\}$, and Jacobian matrices $\{\boldsymbol{A}_t = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_t}, \boldsymbol{B}_t = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}_t}\}$.

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

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \Delta \boldsymbol{x}_t^\top \frac{\partial c}{\partial \boldsymbol{x}_t} + \Delta \boldsymbol{u}_t^\top \frac{\partial c}{\partial \boldsymbol{u}_t} +$$
$$\frac{1}{2} \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t^2} \Delta \boldsymbol{x}_t + \Delta \boldsymbol{x}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{x}_t \boldsymbol{u}_t} \Delta \boldsymbol{u}_t + \frac{1}{2} \Delta \boldsymbol{u}_t^\top \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2} \Delta \boldsymbol{u}_t, \qquad (37)$$

with gradients $\{\frac{\partial c}{\partial \boldsymbol{x}_t}, \frac{\partial c}{\partial \boldsymbol{u}_t}\}$, and Hessian matrices $\{\frac{\partial^2 c}{\partial \boldsymbol{x}_t^2}, \frac{\partial^2 c}{\partial \boldsymbol{x}_t \boldsymbol{u}_t}, \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2}\}$.

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 (36) 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

$$\min_{\Delta \boldsymbol{u}} \Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}), \quad \text{s.t.} \quad \Delta \boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u}, \quad \text{where}$$
$$\Delta c(\Delta \boldsymbol{x}, \Delta \boldsymbol{u}) = \Delta \boldsymbol{x}^\top \boldsymbol{g}_{\boldsymbol{x}} + \Delta \boldsymbol{u}^\top \boldsymbol{g}_{\boldsymbol{u}} +$$
$$\frac{1}{2} \Delta \boldsymbol{x}^\top \boldsymbol{H}_{\boldsymbol{x}} \Delta \boldsymbol{x} + \Delta \boldsymbol{x}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{u}} \Delta \boldsymbol{u} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{H}_{\boldsymbol{u}} \Delta \boldsymbol{u}, \qquad (38)$$

with gradients $\boldsymbol{g}_{\boldsymbol{x}} = \frac{\partial c}{\partial \boldsymbol{x}}$, $\boldsymbol{g}_{\boldsymbol{u}} = \frac{\partial c}{\partial \boldsymbol{u}}$, and Hessian matrices $\boldsymbol{H}_{\boldsymbol{x}} = \frac{\partial^2 c}{\partial \boldsymbol{x}^2}$, $\boldsymbol{H}_{\boldsymbol{x}\boldsymbol{u}} = \frac{\partial^2 c}{\partial \boldsymbol{x}\boldsymbol{u}}$ and $\boldsymbol{H}_{\boldsymbol{u}} = \frac{\partial^2 c}{\partial \boldsymbol{u}^2}$.

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

$$\min_{\Delta \boldsymbol{u}} \quad \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{g}_{\boldsymbol{x}} + \Delta \boldsymbol{u}^\top \boldsymbol{g}_{\boldsymbol{u}} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}} \boldsymbol{S}_{\boldsymbol{u}} \Delta \boldsymbol{u} +$$
$$\Delta \boldsymbol{u}^\top \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{H}_{\boldsymbol{x}\boldsymbol{u}} \Delta \boldsymbol{u} + \frac{1}{2} \Delta \boldsymbol{u}^\top \boldsymbol{H}_{\boldsymbol{u}} \Delta \boldsymbol{u}, \qquad (39)$$

[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 $\boldsymbol{x}_1$).

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

$$\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{g}_{\boldsymbol{x}} + \boldsymbol{g}_{\boldsymbol{u}} + \boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{x}}\boldsymbol{S}_{\boldsymbol{u}}\Delta\boldsymbol{u} + 2\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{x}\boldsymbol{u}}\Delta\boldsymbol{u} + \boldsymbol{H}_{\boldsymbol{u}}\Delta\boldsymbol{u} = 0, \tag{40}$$

providing the least squares solution

$$\Delta\hat{\boldsymbol{u}} = \left(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{x}}\boldsymbol{S}_{\boldsymbol{u}} + 2\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{H}_{\boldsymbol{x}\boldsymbol{u}} + \boldsymbol{H}_{\boldsymbol{u}}\right)^{-1}\left(-\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{g}_{\boldsymbol{x}} - \boldsymbol{g}_{\boldsymbol{u}}\right), \tag{41}$$

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

7.1 Recursive formulation of iLQR

iLQR_manipulator_recursive.*

Section 6.3 presented the dynamic programming principle in the context of linear quadratic regulation problems, which allowed us to reduce the minimization over an entire sequence of control commands to a sequence of minimization problems over control commands at a single time step, by proceeding backwards in time. In this section, the approach is extended to iLQR. We start from the minimization of a cost-to-go function as in (23), namely

$$c_t = \min_{\boldsymbol{u}_t} \; \ell_t + c_{t+1}. \tag{42}$$

If we define the argument minimizing (42) as a function of perturbations around $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$, we obtain

$$q_t = \ell_t(\boldsymbol{x}_t + \boldsymbol{\Delta x}_t, \boldsymbol{u}_t + \boldsymbol{\Delta u}_t) - \ell_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \\ c_{t+1}(\boldsymbol{x}_t + \boldsymbol{\Delta x}_t, \boldsymbol{u}_t + \boldsymbol{\Delta u}_t) - c_{t+1}(\boldsymbol{x}_t, \boldsymbol{u}_t),$$

whose second order expansion is characterized by

$$q_t \approx \begin{bmatrix} 1 \\ \boldsymbol{\Delta x}_t \\ \boldsymbol{\Delta u}_t \end{bmatrix}^{\top} \begin{bmatrix} 0 & \boldsymbol{q}_t^{\boldsymbol{x}\top} & \boldsymbol{q}_t^{\boldsymbol{u}\top} \\ \boldsymbol{q}_t^{\boldsymbol{x}} & \boldsymbol{Q}_t^{\boldsymbol{x}\boldsymbol{x}} & \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}\top} \\ \boldsymbol{q}_t^{\boldsymbol{u}} & \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}} & \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}} \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{\Delta x}_t \\ \boldsymbol{\Delta u}_t \end{bmatrix}, \tag{43}$$

with

$$\begin{aligned} \boldsymbol{q}_t^{\boldsymbol{x}} &= \boldsymbol{\ell}_t^{\boldsymbol{x}} + \boldsymbol{F}_t^{\boldsymbol{x}\top}\boldsymbol{c}_{t+1}^{\boldsymbol{x}}, \\ \boldsymbol{q}_t^{\boldsymbol{u}} &= \boldsymbol{\ell}_t^{\boldsymbol{u}} + \boldsymbol{F}_t^{\boldsymbol{u}\top}\boldsymbol{c}_{t+1}^{\boldsymbol{x}}, \\ \boldsymbol{Q}_t^{\boldsymbol{x}\boldsymbol{x}} &\approx \boldsymbol{L}_t^{\boldsymbol{x}\boldsymbol{x}} + \boldsymbol{F}_t^{\boldsymbol{x}\top}\boldsymbol{C}_{t+1}^{\boldsymbol{x}\boldsymbol{x}}\boldsymbol{F}_t^{\boldsymbol{x}}, \\ \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}} &\approx \boldsymbol{L}_t^{\boldsymbol{u}\boldsymbol{u}} + \boldsymbol{F}_t^{\boldsymbol{u}\top}\boldsymbol{C}_{t+1}^{\boldsymbol{x}\boldsymbol{x}}\boldsymbol{F}_t^{\boldsymbol{u}}, \\ \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}} &\approx \boldsymbol{L}_t^{\boldsymbol{u}\boldsymbol{x}} + \boldsymbol{F}_t^{\boldsymbol{u}\top}\boldsymbol{C}_{t+1}^{\boldsymbol{x}\boldsymbol{x}}\boldsymbol{F}_t^{\boldsymbol{x}}. \end{aligned}$$

Minimizing (43) w.r.t. $\boldsymbol{\Delta u}_t$ yields the controller

$$\hat{\boldsymbol{\Delta u}}_t = -\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}^{-1}}(\boldsymbol{q}_t^{\boldsymbol{u}} + \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}}\boldsymbol{\Delta x}_t), \tag{44}$$

with feedback gain matrix $\boldsymbol{K}_t = -\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}^{-1}}\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}}$ and feedforward term $\boldsymbol{k}_t = -\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}^{-1}}\boldsymbol{q}_t^{\boldsymbol{u}}$.

By inserting (44) into (43), we get the recursive updates

$$\begin{aligned} \boldsymbol{c}_t^{\boldsymbol{x}} &= \boldsymbol{q}_t^{\boldsymbol{x}} - \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}\top}\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}^{-1}}\boldsymbol{q}_t^{\boldsymbol{u}}, \\ \boldsymbol{C}_t^{\boldsymbol{x}\boldsymbol{x}} &= \boldsymbol{Q}_t^{\boldsymbol{x}\boldsymbol{x}} - \boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}\top}\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{u}^{-1}}\boldsymbol{Q}_t^{\boldsymbol{u}\boldsymbol{x}}. \end{aligned}$$

Algorithm 1: Backtracking line search method with parameter $\alpha_{\min}$

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

7.2 Updates by considering step sizes

iLQR_manipulator.*

To be more efficient, iLQR most often requires at each iteration to estimate a step size α to scale the update computed in (41) with

$$\hat{\boldsymbol{u}} \leftarrow \hat{\boldsymbol{u}} + \alpha\,\Delta\hat{\boldsymbol{u}}. \tag{45}$$

In practice, a simple backtracking line search procedure can be considered with Algorithm 1, by taking a small value for ϵ. The resulting procedure first estimates with (41) a descent direction along which the objective function will be reduced and then estimates with (45) a step size that determines how far we should move along this direction.

7.3 iLQR with quadratic cost on $\boldsymbol{f}(\boldsymbol{x}_t)$

iLQR_manipulator.*

We consider a cost defined by

$$c(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{f}(\boldsymbol{x}_t)^{\top}\boldsymbol{Q}_t\boldsymbol{f}(\boldsymbol{x}_t) + \boldsymbol{u}_t^{\top}\boldsymbol{R}_t\,\boldsymbol{u}_t, \tag{46}$$

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 $\boldsymbol{x}_t$.

The cost in (37) then becomes

$$\Delta c(\Delta\boldsymbol{x}_t, \Delta\boldsymbol{u}_t) \approx 2\Delta\boldsymbol{x}_t^{\top}\boldsymbol{J}(\boldsymbol{x}_t)^{\top}\boldsymbol{Q}_t\boldsymbol{f}(\boldsymbol{x}_t) + 2\Delta\boldsymbol{u}_t^{\top}\boldsymbol{R}_t\,\boldsymbol{u} + \\ \Delta\boldsymbol{x}_t^{\top}\boldsymbol{J}(\boldsymbol{x}_t)^{\top}\boldsymbol{Q}_t\boldsymbol{J}(\boldsymbol{x}_t)\Delta\boldsymbol{x}_t + \Delta\boldsymbol{u}_t^{\top}\boldsymbol{R}_t\Delta\boldsymbol{u}_t, \tag{47}$$

which used

$$\frac{\partial c}{\partial \boldsymbol{x}_t} = 2\boldsymbol{J}(\boldsymbol{x}_t)^{\top}\boldsymbol{Q}_t\boldsymbol{f}(\boldsymbol{x}_t), \quad \frac{\partial c}{\partial \boldsymbol{u}_t} = 2\boldsymbol{R}_t\boldsymbol{u}, \tag{48}$$

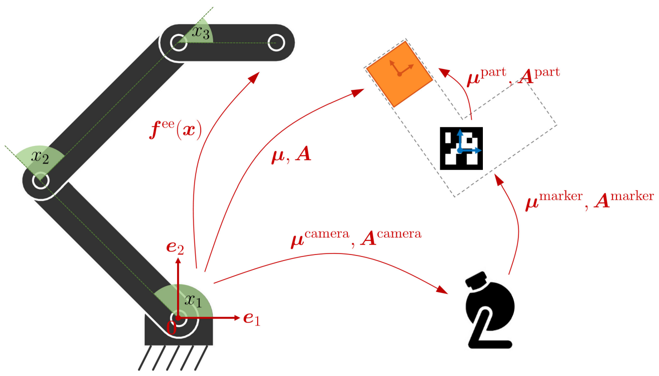
$$\frac{\partial^2 c}{\partial \boldsymbol{x}_t^2} \approx 2\boldsymbol{J}(\boldsymbol{x}_t)^{\top}\boldsymbol{Q}_t\boldsymbol{J}(\boldsymbol{x}_t), \quad \frac{\partial^2 c}{\partial \boldsymbol{x}_t\boldsymbol{u}_t} = \boldsymbol{0}, \quad \frac{\partial^2 c}{\partial \boldsymbol{u}_t^2} = 2\boldsymbol{R}_t. \tag{49}$$

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

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

With a slight abuse of notation, we define $\boldsymbol{f}(\boldsymbol{x})$ as a vector concatenating the vectors $\boldsymbol{f}(\boldsymbol{x}_t)$, and $\boldsymbol{J}(\boldsymbol{x})$ as a block-diagonal concatenation of the Jacobian matrices $\boldsymbol{J}(\boldsymbol{x}_t)$. The minimization problem (39) then becomes

$$\min_{\Delta\boldsymbol{u}} \quad 2\Delta\boldsymbol{x}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) + 2\Delta\boldsymbol{u}^{\top}\boldsymbol{R}\,\boldsymbol{u} + \\ \Delta\boldsymbol{x}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} + \Delta\boldsymbol{u}^{\top}\boldsymbol{R}\Delta\boldsymbol{u}, \quad \text{s.t.} \quad \Delta\boldsymbol{x} = \boldsymbol{S}_{\boldsymbol{u}}\Delta\boldsymbol{u}, \tag{50}$$



x_3
x_2
x_1
$\boldsymbol{f}^{\text{ee}}(\boldsymbol{x})$
$\boldsymbol{\mu}, \boldsymbol{A}$
$\boldsymbol{\mu}^{\text{part}}, \boldsymbol{A}^{\text{part}}$
$\boldsymbol{\mu}^{\text{marker}}, \boldsymbol{A}^{\text{marker}}$
$\boldsymbol{\mu}^{\text{camera}}, \boldsymbol{A}^{\text{camera}}$
$\boldsymbol{e}_2$
$\boldsymbol{e}_1$

Figure 8: 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.

whose least squares solution is given by

$$\Delta\hat{\boldsymbol{u}} = \big(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{J}(\boldsymbol{x})\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\big)^{-1} \big(-\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{R}\boldsymbol{u}\big), \quad (51)$$

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

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

7.3.1 Robot manipulator

iLQR_manipulator.*

We define a manipulation task involving a set of transformations as in Fig. 8. By relying on these transformation operators, we will next describe all variables in the robot frame of reference (defined by $\mathbf{0}$, $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ in the figure).

For a manipulator controlled by joint angle velocity commands $\boldsymbol{u} = \dot{\boldsymbol{x}}$, the evolution of the system is described by $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{u}_t\Delta t$, with the Taylor expansion (36) 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 (51) typically represents the error between a reference $\boldsymbol{\mu}_t$ and the end-effector position computed by the forward kinematics function $\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t)$. We then have

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

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

The approach can similarly be extended to target objects/landmarks with positions $\boldsymbol{\mu}_t$ and orientation matrices $\boldsymbol{U}_t$, whose columns are basis vectors forming a coordinate system, see Fig. 10. We can then define an error between the robot end-effector and an object/landmark expressed in the object/landmark coordinate system as

$$\boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{U}_t^{\top}\big(\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_t) - \boldsymbol{\mu}_t\big), \quad (52)$$
$$\boldsymbol{J}(\boldsymbol{x}_t) = \boldsymbol{U}_t^{\top}\boldsymbol{J}^{\text{ee}}(\boldsymbol{x}_t). \quad (53)$$

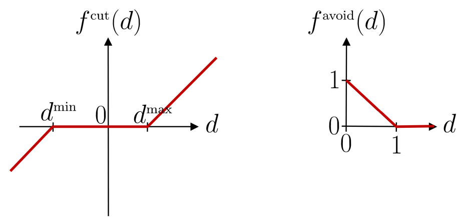


Figure 9: ReLU-like functions used in optimization costs. The derivatives of these functions are simple, providing Jacobians whose entries are either 0 or ±1.

7.3.2 Bounded joint space

The iLQR solution in (51) 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

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

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

where $f_i^{\text{cut}}(x_{t,i})$ describes continuous ReLU-like functions for each dimension. $f_i^{\text{cut}}(x_{t,i})$ is 0 inside the bounded domain and takes the signed distance value outside the boundary, see Fig. 9-*left*.

We can see with (48) 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.

7.3.3 Bounded task space

The iLQR solution in (51) can be used to keep the end-effector within a boundary (e.g., end-effector position limits). Based on the above definitions, $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{J}(\boldsymbol{x})$ are in this case defined as

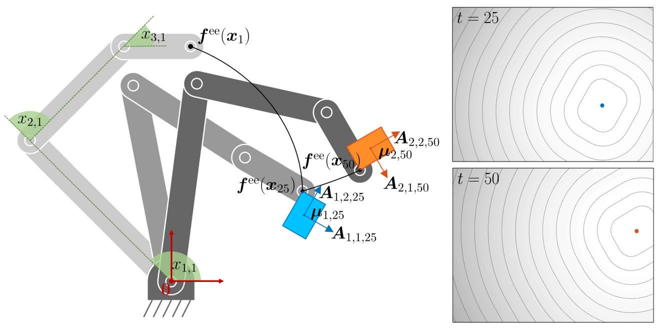
$$\boldsymbol{f}(\boldsymbol{x}_t) = \boldsymbol{f}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big),$$
$$\boldsymbol{J}(\boldsymbol{x}_t) = \boldsymbol{J}^{\text{cut}}\Big(\boldsymbol{e}(\boldsymbol{x}_t)\Big)\,\boldsymbol{J}^{\text{ee}}(\boldsymbol{x}),$$
$$\text{with}\quad \boldsymbol{e}(\boldsymbol{x}_t)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

7.3.4 Reaching task with robot manipulator and prismatic object boundaries

iLQR_manipulator.*

The definition of $\boldsymbol{f}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}_t)$ in (52) and (53) can also be extended to objects/landmarks with boundaries by defining

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



$x_{3,1}$
$\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_1)$
$x_{2,1}$
$\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_{50})$
$\boldsymbol{A}_{2,2,50}$
$\boldsymbol{\mu}_{2,50}$
$\boldsymbol{A}_{2,1,50}$
$\boldsymbol{f}^{\text{ee}}(\boldsymbol{x}_{25})$
$\boldsymbol{A}_{1,2,25}$
$\boldsymbol{\mu}_{1,25}$
$\boldsymbol{A}_{1,1,25}$
$x_{1,1}$
0
$t=25$
$t=50$

Figure 10: Example of a viapoints task in which a planar robot with 3 joints needs to sequentially reach 2 objects, with object boundaries defining the allowed reaching points on the objects surfaces. *Left:* Reaching task with two viapoints at $t = 25$ and $t = 50$. *Right:* Corresponding values of the cost function for the end-effector space at $t = 25$ and $t = 50$.

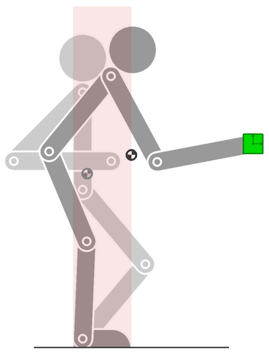


Figure 11: Reaching task with a humanoid (side view) by keeping the center of mass in an area defined by the feet location. The starting and ending poses are depicted in different grayscale levels. The corresponding center of mass is depicted by a darker semi-filled disc. The area allowed for the center of mass is depicted as a transparent red rectangle. The task consists of reaching the green object with the end-effector at the end of the movement, while always keeping the center of mass within the desired bounding box during the movement.

see also Fig. 10.

7.3.5 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 (16) 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 end-effector, as in Section 7.3.4, see Fig. 11 for an example.

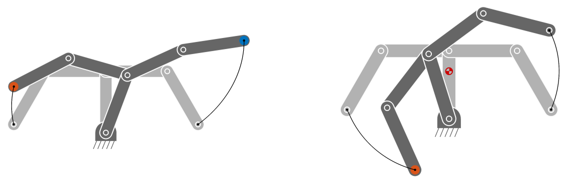


Figure 12: Reaching tasks with a bimanual robot (frontal view). *Left:* with a target for each hand. *Right:* with a target for the hand on the left, while keeping the center of mass at the same location during the whole movement.

7.3.6 Bimanual robot

iLQR_bimanual.*

We consider a 5-link planar bimanual robot with a torso link shared by the two arms, see Fig. 12. We define the forward kinematics function as

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

where the first two and last two dimensions of $\boldsymbol{f}$ correspond to the position of the left and right end-effectors, respectively. The corresponding Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{4\times5}$ has entries

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

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. 12 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 $\boldsymbol{J}^{\text{CoM}} \in \mathbb{R}^{2\times5}$ computed as

$$\boldsymbol{J}^{\text{CoM}} = \Big[\boldsymbol{J}^{\text{CoM-L}}_{[1,2],1} + \boldsymbol{J}^{\text{CoM-R}}_{[1,2],1},\quad \boldsymbol{J}^{\text{CoM-L}}_{[1,2],[2,3]},\quad \boldsymbol{J}^{\text{CoM-R}}_{[1,2],[2,3]}\Big],$$

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

7.3.7 Ellipsoidal obstacle avoidance

iLQR_obstacle.*

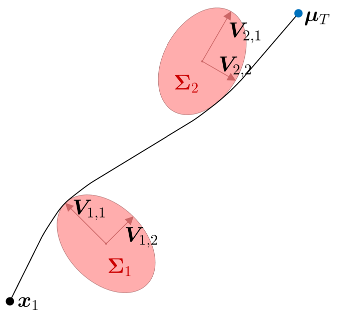


Figure 13: Reaching task with obstacle avoidance.

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. 13 for an example). We first define a ReLU-like function and its gradient as (see also Fig. 9-*right*)

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

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

that we exploit to define $\boldsymbol{f}(\boldsymbol{x}_t)$ and $\boldsymbol{J}(\boldsymbol{x}_t)$ in (51) as

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x}_t) &= f^{\text{avoid}}\Big(e(\boldsymbol{x}_t)\Big), \\ \boldsymbol{J}(\boldsymbol{x}_t) &= 2\, \boldsymbol{g}^{\text{avoid}}\Big(e(\boldsymbol{x}_t)\Big)\ (\boldsymbol{x}_t - \boldsymbol{\mu})^\top\ \boldsymbol{\Sigma}^{-1}, \\ \text{with} \quad e(\boldsymbol{x}_t) &= (\boldsymbol{x}_t - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_t - \boldsymbol{\mu}). \end{aligned}$$

In the above, $\boldsymbol{f}(\boldsymbol{x})$ defines a continuous and smooth 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. 13.

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

7.3.8 Manipulability tracking

`iLQR_bimanual_manipulability.*`

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 [11], with the drawback of ignoring the specific shape of this ellipsoid.

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

$$\begin{aligned} & c = \|\boldsymbol{A}\|^2_{\text{F}}, \quad \text{with} \quad \boldsymbol{A} = \log\!\big(\boldsymbol{S}^{\frac{1}{2}} \boldsymbol{M}(\boldsymbol{x}) \boldsymbol{S}^{\frac{1}{2}}\big), \\ \iff \ & c = \text{trace}(\boldsymbol{A}\boldsymbol{A}^\top) = \sum_i \text{trace}(\boldsymbol{A}_i\boldsymbol{A}_i^\top) = \sum_i \boldsymbol{A}_i^\top \boldsymbol{A}_i \\ & \quad = \text{vec}(\boldsymbol{A})^\top \text{vec}(\boldsymbol{A}), \end{aligned} \tag{54}$$

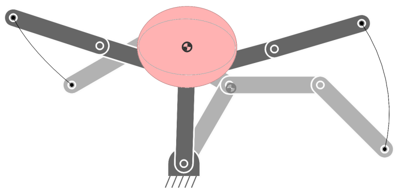


Figure 14: Example of iLQR optimization with a cost on manipulability, where the goal is to reach, at the end of the motion, a desired manipulability defined at the center of mass (CoM) of a bimanual planar robot. The initial pose of the bimanual robot is displayed in light gray and the final pose of 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.

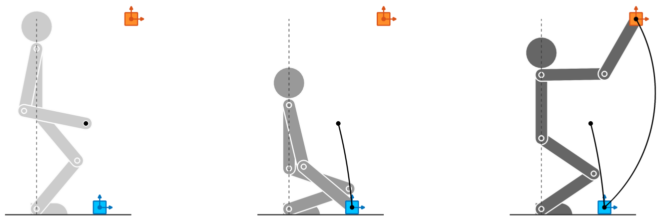


Figure 15: Reaching task with control primitives.

where $\boldsymbol{S}$ is the desired manipulability matrix to reach, $\log(\cdot)$ is the logarithm matrix function, and $\|\cdot\|_{\text{F}}$ is a Frobenius norm. By exploiting the trace properties, we can see that (54) can be expressed in a quadratic form, where the vectorization operation $\text{vec}(\boldsymbol{A})$ 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 (54) with respect to the state $\boldsymbol{x}$ and control commands $\boldsymbol{u}$ can be computed analytically by following the approach presented in [5], or numerically by exploiting the quadratic form of (54) to solve the iLQR problem with Gauss-Newton optimization, by using Jacobians estimated numerically, see description in Section 5.1 within the context of inverse kinematics.

Marić *et al.* demonstrated the advantages of a geometric cost as in (54) for planning problems, by comparing it to alternative widely used metrics such as the manipulability index and the dexterity index [7].

Note here that manipulability can be defined at several points of interest, including endeffectors and centers of mass. Moreover, it can be defined for either position or force (including the rotational part), which provide rich descriptors to characterize robot motor skills. Figure 14 presents a simple example with a bimanual robot.

7.4 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

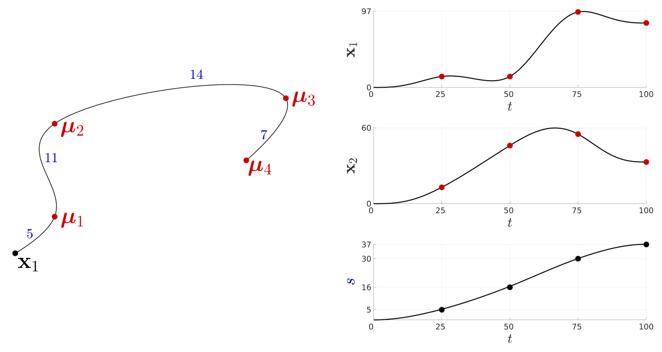


Figure 16: Spacetime optimization with iLQR for a viapoints task, where the estimated durations between consecutive viapoints are shown in blue.

quadratic cost on $\boldsymbol{f}(\boldsymbol{x}_t)$ (see previous Section), the update of the weights vector is then given by

$$\Delta\hat{\boldsymbol{w}}=\Big(\boldsymbol{\Psi}^{\top}\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{J}(\boldsymbol{x})\boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{\Psi}+\boldsymbol{R}\Big)^{-1}\Big(-\boldsymbol{\Psi}^{\top}\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{J}(\boldsymbol{x})^{\top}\boldsymbol{Q}\boldsymbol{f}(\boldsymbol{x})-\boldsymbol{R}\boldsymbol{u}\Big). \quad (55)$$

For a 5-link kinematic chain, by setting the coordination matrix in (21) 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 15 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 (55) can usually be computed faster than in the original formulation, since a matrix of much smaller dimension needs to be inverted.

7.5 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 $\boldsymbol{x}_{t+1}=\boldsymbol{g}(\boldsymbol{x}_t,\boldsymbol{u}_t)$ as

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

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

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

with Jacobian matrices $\boldsymbol{A}_t=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{x}_t}$, $\boldsymbol{B}_t=\frac{\partial\boldsymbol{g}}{\partial\boldsymbol{u}_t}$.

Figure 16 shows a viapoints task example in which both path and duration are optimized (convergence after 10 iterations when starting from zero commands).

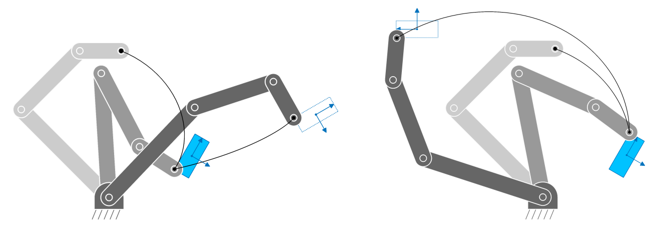


Figure 17: Manipulation with object affordances cost. The depicted *pick-and-place* task requires the offset at the picking location to be the same as the offset at the placing location, which can be achieved by setting a precision matrix with non-zero offdiagonal entries, see main text for details. In this example, an additional cost is set for choosing the picking and placing points within the object boundaries, which was also used for the task depicted in Fig. 10. 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.

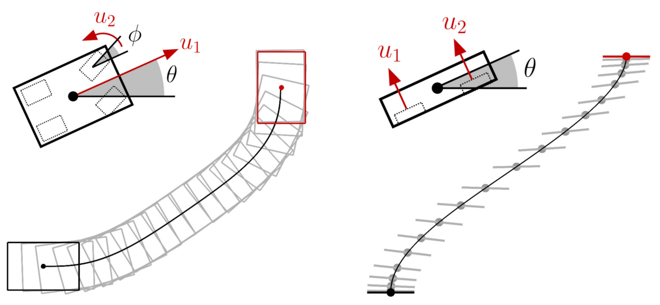


Figure 18: iLQR for car and bicopter control/planning problems.

7.6 iLQR with offdiagonal elements in the precision matrix

`iLQR_manipulator_object_affordance.*`

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

Such cost can for example be used to define costs on object affordances, see Fig. 17 for an example.

7.7 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. 18-*left*) is characterized by equations

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

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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{u}_1 \sin(\hat{\theta}) & 0 \\ 0 & 0 & \hat{u}_1 \cos(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{\hat{u}_1}{\ell}(\tan(\hat{\phi})^2\!+\!1) \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} \cos(\hat{\theta}) & 0 \\ \sin(\hat{\theta}) & 0 \\ \frac{1}{\ell}\tan(\hat{\phi}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \quad (56)$$

which can then be converted to a discrete form with (62).

Acceleration Control Input
A car motion with position (x_1, x_2), car orientation θ, and front wheels orientation ϕ (see Fig. 18-*left*) is characterized by equations

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

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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\dot{v} \\ \Delta\dot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\hat{v}\sin(\hat{\theta}) & \cos(\hat{\theta}) & 0 \\ 0 & 0 & \hat{v}\cos(\hat{\theta}) & \sin(\hat{\theta}) & 0 \\ 0 & 0 & 0 & \frac{1}{\ell}\tan(\hat{\phi}) & \frac{\hat{v}}{\ell}(\tan(\hat{\phi})^2\!+\!1) \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta v \\ \Delta\phi \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \quad (57)$$

which can then be converted to a discrete form with (62).

7.8 Bicopter

iLQR_bicopter.*

A planar bicopter of mass m and inertia I actuated by two propellers at distance ℓ (see Fig. 18-*right*) is characterized by equations

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

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

$$\begin{bmatrix} \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \\ \Delta\ddot{x}_1 \\ \Delta\ddot{x}_2 \\ \Delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1 + \hat{u}_2)\cos(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{m}(\hat{u}_1 + \hat{u}_2)\sin(\hat{\theta}) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta\theta \\ \Delta\dot{x}_1 \\ \Delta\dot{x}_2 \\ \Delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\frac{1}{m}\sin(\hat{\theta}) & -\frac{1}{m}\sin(\hat{\theta}) \\ \frac{1}{m}\cos(\hat{\theta}) & \frac{1}{m}\cos(\hat{\theta}) \\ \frac{\ell}{I} & -\frac{\ell}{I} \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \end{bmatrix}, \quad (58)$$

which can then be converted to a discrete form with (62).

References

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

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

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

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

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

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

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

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

[9] 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.

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

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

Appendices

A System dynamics at trajectory level

The evolution of a system $\boldsymbol{x}_{t+1} = g(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (expressed in discrete form), can be approximated by the linear system

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

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

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

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

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{x}_3\\ \vdots\\ \boldsymbol{x}_T \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} \boldsymbol{I}\\ \boldsymbol{A}_1\\ \boldsymbol{A}_2\boldsymbol{A}_1\\ \vdots\\ \prod_{t=1}^{T-1}\boldsymbol{A}_{T-t} \end{bmatrix}}_{\boldsymbol{S_x}} \boldsymbol{x}_1 \;+$$

$$\underbrace{\begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{B}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \boldsymbol{A}_2\boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{0}\\ \vdots & \vdots & \ddots & \vdots\\ \left(\prod_{t=1}^{T-2}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_1 & \left(\prod_{t=1}^{T-3}\boldsymbol{A}_{T-t}\right)\boldsymbol{B}_2 & \cdots & \boldsymbol{B}_{T-1} \end{bmatrix}}_{\boldsymbol{S_u}} \underbrace{\begin{bmatrix} \boldsymbol{u}_1\\ \boldsymbol{u}_2\\ \vdots\\ \boldsymbol{u}_{T-1} \end{bmatrix}}_{\boldsymbol{u}},$$

where $\boldsymbol{S_x} \in \mathbb{R}^{dT\times d}$, $\boldsymbol{x}_1 \in \mathbb{R}^d$, $\boldsymbol{S_u} \in \mathbb{R}^{dT\times d(T-1)}$ 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 $\boldsymbol{B}_t = \boldsymbol{I}\Delta t$, $\forall t \in \{1, \ldots, T\}$, and transfer matrices $\boldsymbol{S_x} = \boldsymbol{1}_T \otimes \boldsymbol{I}_D$, and $\boldsymbol{S_u} = \begin{bmatrix} \boldsymbol{0}_{D,D(T-1)}\\ \boldsymbol{L}_{T-1,T-1} \otimes \boldsymbol{I}_D\Delta t \end{bmatrix}$, where $\boldsymbol{L}$ is a lower triangular matrix and $\otimes$ is the Kronecker product operator.

B Linear systems used in the bimanual tennis serve example

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

The evolution of the system is described by the linear relation $\dot{\boldsymbol{x}}_t = \boldsymbol{A}^c_t\boldsymbol{x}_t + \boldsymbol{B}^c_t\boldsymbol{u}_t$, 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} & \color{red}{\boldsymbol{I}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \color{red}{\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}^c_t} \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}^c_t} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}. \tag{59}$$

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} & \color{red}{\boldsymbol{I}} & \mathbf{0} & \mathbf{0} & \color{red}{\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}^c_t} \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}^c_t} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}. \tag{60}$$

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

$$\underbrace{\left[\begin{array}{c} \dot{\mathbf{x}}_{1,t}\\ \ddot{\mathbf{x}}_{1,t}\\ \dot{\boldsymbol{f}}_{1,t}\\ \hline \dot{\mathbf{x}}_{2,t}\\ \ddot{\mathbf{x}}_{2,t}\\ \dot{\boldsymbol{f}}_{2,t}\\ \hline \dot{\mathbf{x}}_{3,t}\\ \ddot{\mathbf{x}}_{3,t}\\ \dot{\boldsymbol{f}}_{3,t} \end{array}\right]}_{\dot{\boldsymbol{x}}_t} = \underbrace{\left[\begin{array}{ccc|ccc|ccc} \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0}\\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{I}\\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right]}_{\boldsymbol{A}^c_t} \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}^c_t} \underbrace{\left[\begin{array}{c} \boldsymbol{u}_{1,t}\\ \hline \boldsymbol{u}_{2,t} \end{array}\right]}_{\boldsymbol{u}_t}. \tag{61}$$

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 $\{\boldsymbol{A}^c_t, \boldsymbol{B}^c_t\}_{t=1}^T$, describing a continuous system, are first converted to their discrete forms with

$$\boldsymbol{A}_t = \boldsymbol{I} + \boldsymbol{A}^c_t\Delta t, \quad \boldsymbol{B}_t = \boldsymbol{B}^c_t\Delta t, \quad \forall t \in \{1, \ldots, T\}, \tag{62}$$

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 $\boldsymbol{x} = \boldsymbol{S_x}\boldsymbol{x}_1 + \boldsymbol{S_u}\boldsymbol{u}$.

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

C 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

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

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

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

and it is easy to see that

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