



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Robotic Systems I

Lecture 8: Introduction to Optimization-based Control

Konstantinos Chatzilygeroudis - costashatz@upatras.gr

Department of Electrical and Computer Engineering
University of Patras

Template made by Panagiotis Papagiannopoulos



Manipulator Equation Recap v2

Manipulator Equation Reminder:

$$\underbrace{\boldsymbol{M}(\boldsymbol{q})}_{\text{"Mass Matrix"}} \dot{\boldsymbol{v}} + \underbrace{\boldsymbol{C}(\boldsymbol{q}, \boldsymbol{v})}_{\text{"Coriolis/Gravity Forces"}} = \boldsymbol{B}(\boldsymbol{q}) \underbrace{\boldsymbol{u}}_{\text{"Usually } \boldsymbol{\tau}} + \underbrace{\boldsymbol{F}_{\text{ext}}}_{\text{"External forces"}}$$

Velocity Kinematics:

$$\dot{\boldsymbol{q}} = \boldsymbol{G}(\boldsymbol{q})\boldsymbol{v}$$

Forward Dynamics:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{G}(\mathbf{q})\mathbf{v} \\ \mathbf{M}^{-1}(\mathbf{q})\left(\mathbf{B}(\mathbf{q})\mathbf{u} + \mathbf{F}_{\text{ext}} - \mathbf{C}(\mathbf{q}, \mathbf{v})\right) \end{bmatrix}$$

Inverse Dynamics:

$$\boldsymbol{\tau} = \mathbf{B}(\mathbf{q})^{-1}(\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{C}(\mathbf{q}, \mathbf{v}) - \mathbf{F}_{\text{ext}})$$

Manipulator Equation Recap v3

Manipulator Equation Reminder:

$$\underbrace{\mathbf{M}(\mathbf{q})}_{\text{"Mass Matrix"}} \dot{\mathbf{v}} + \underbrace{\mathbf{C}(\mathbf{q}, \mathbf{v})}_{\text{"Coriolis/Gravity Forces"}} = \underbrace{\mathbf{u}}_{\text{"Usually } \boldsymbol{\tau}\text{"}} + \underbrace{\mathbf{F}_{\text{ext}}}_{\text{"External forces"}}$$

Velocity Kinematics:

$$\dot{\mathbf{q}} = \mathbf{G}(\mathbf{q})\mathbf{v}$$

Forward Dynamics:

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{q}) \left(\mathbf{u} + \mathbf{F}_{\text{ext}} - \mathbf{C}(\mathbf{q}, \mathbf{v}) \right)$$

Inverse Dynamics:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\dot{\mathbf{v}} + \mathbf{C}(\mathbf{q}, \mathbf{v}) - \mathbf{F}_{\text{ext}}$$

Joint Control with Velocities (1)

- We assume that we have a desired trajectory $\mathbf{q}_d(t)$
- For example, $\mathbf{q}_d(t) = \alpha \sin(t)$
- The easiest way to control the joint is to give velocity commands:

$$\dot{\mathbf{q}}(t) = \dot{\mathbf{q}}_d(t) = \alpha \cos(t)$$

- **Feedforward** or **open-loop** controller
- There is not *feedback* from sensors!

Joint Control with Velocities (2)

- How can we exploit sensor readings? In other words, how can we do **feedback control**?
- **P-controller:** $\dot{\mathbf{q}}(t) = \mathbf{K}_p(\mathbf{q}_d(t) - \mathbf{q}(t))$, $\mathbf{K}_p > 0$
- $\mathbf{q}_e(t) = (\mathbf{q}_d(t) - \mathbf{q}(t))$
- **PI-controller:** $\dot{\mathbf{q}}(t) = \mathbf{K}_p \mathbf{q}_e(t) + \mathbf{K}_i \int_0^t \mathbf{q}_e(t) dt$, $\mathbf{K}_p, \mathbf{K}_i > 0$
- **PID-controller:** $\dot{\mathbf{q}}(t) = \mathbf{K}_p \mathbf{q}_e(t) + \mathbf{K}_i \int_0^t \mathbf{q}_e(t) dt + \mathbf{K}_d \dot{\mathbf{q}}_e(t)$,
 $\mathbf{K}_p, \mathbf{K}_i, \mathbf{K}_d > 0$
- If $\mathbf{q}_d(t) = \text{constant}^1$, then the PI-controller removes the steady state error.

¹Or converges to a static point.

- Feedback control needs an error signal to “begin”!
- Let’s combine the open-loop and the feedback control loops:

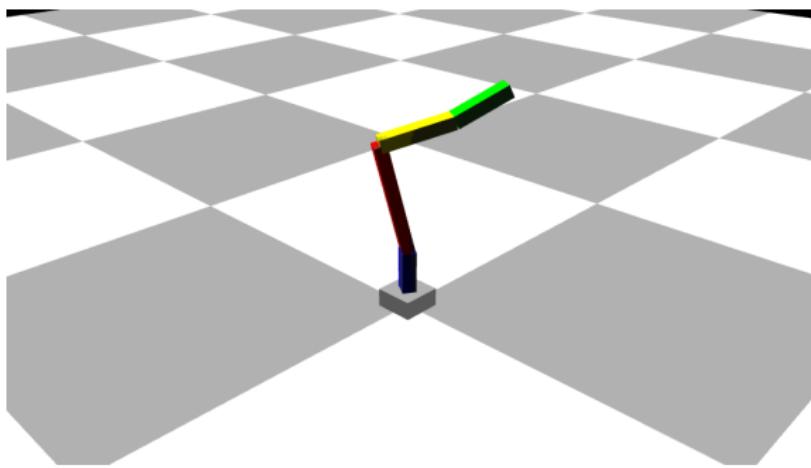
$$\dot{\mathbf{q}}(t) = \dot{\mathbf{q}}_d(t) + \boldsymbol{\mathcal{K}}_p \mathbf{q}_e(t) + \boldsymbol{\mathcal{K}}_i \int_0^t \mathbf{q}_e(t) dt + \boldsymbol{\mathcal{K}}_d \dot{\mathbf{q}}_e(t),$$
$$\boldsymbol{\mathcal{K}}_p, \boldsymbol{\mathcal{K}}_i, \boldsymbol{\mathcal{K}}_d > 0$$

- **Feedforward Plus Feedback Control**

What if we have multiple joints?

- We assume that every joint is “independent”
- Thus we have n controllers (where n is the number of joints)
- $\boldsymbol{K}_p \in \mathbb{R}^{n \times n}$
- $\boldsymbol{K}_i \in \mathbb{R}^{n \times n}$
- $\boldsymbol{K}_d \in \mathbb{R}^{n \times n}$
- $\boldsymbol{K}_p, \boldsymbol{K}_i, \boldsymbol{K}_d$ usually have values only in the diagonal
- Otherwise, there is correlation between joints

Let's control a robot!



Jacobians - Reminder (1)

Let's assume that the *end-effector* of our robot is moving with velocity¹ $\dot{\mathbf{x}}$. Let's also write the forward kinematics problem as a function of time:

$$\mathbf{x}(t) = f_{fk}(\mathbf{q}(t))$$

where f_{fk} is the function that gives us the forward kinematics, $\mathbf{x} \in \mathbb{R}^m$ the pose of the end-effector, and $\mathbf{q} \in \mathbb{R}^n$ the joint values of the robot. If we take the derivative over time:

$$\begin{aligned}\dot{\mathbf{x}} &= \frac{\partial f_{fk}(\mathbf{q})}{\partial \mathbf{q}} \frac{\partial \mathbf{q}(t)}{\partial t} \\ &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}\end{aligned}$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{m \times n}$ is the Jacobian matrix.

¹This is an abstract velocity here.

Jacobians - Reminder (2)

- Twist \mathcal{V}_w of the end-effector expressed in world frame
- We have: $\mathcal{V}_w = \mathbf{J}_w(\mathbf{q})\dot{\mathbf{q}}$, where $\mathbf{J}_w(\mathbf{q}) \in \mathbb{R}^{6 \times n}$
- We have: $\mathcal{V}_b = \mathbf{J}_b(\mathbf{q})\dot{\mathbf{q}}$, where $\mathbf{J}_b(\mathbf{q}) \in \mathbb{R}^{6 \times n}$ is the Jacobian expressed in body frame
- We also have: $\mathbf{J}_b = [Ad_{T_{bw}}]\mathbf{J}_w$ and $\mathbf{J}_w = [Ad_{T_{wb}}]\mathbf{J}_b$

From the principle of energy conservation, we can also derive an equation for the *wrenches*:

$$\begin{aligned}\boldsymbol{\tau} &= \mathbf{J}_w(\mathbf{q})^T \mathcal{F}_w \\ \boldsymbol{\tau} &= \mathbf{J}_b(\mathbf{q})^T \mathcal{F}_b\end{aligned}$$

where $\boldsymbol{\tau} \in \mathbb{R}^n$ are the joint torques/forces.

Inverse Kinematics

- This is the “opposite” problem of forward kinematics
- We can find closed-form solutions for many systems
- We can develop iterative algorithms based on the Jacobians
- We can view the problem as an optimization problem
 - We can still use the Jacobians
 - Or we can go black-box!
 - We can take advantage of numerical optimization methods

Inverse Kinematics via Optimization (1)

- We have the end-effector pose, $\mathbf{x} \in \mathbb{R}^m$, which is given by the forward kinematics $\mathbf{x} = f(\mathbf{q})$, where $\mathbf{q} \in \mathbb{R}^n$ are the joint positions (n degrees of freedom)
- Let's say that we want to go to \mathbf{x}_d
- We have the following error $e(\mathbf{q}) = \mathbf{x}_d - f(\mathbf{q})$
- So in fact we want to find \mathbf{q}_d such that $e(\mathbf{q}_d) = \mathbf{x}_d - f(\mathbf{q}_d) = \mathbf{0}$
- We have (Taylor Expansion around \mathbf{q}_0):

$$\mathbf{x}_d - f(\mathbf{q}_d) = \mathbf{0}$$

$$\mathbf{x}_d = f(\mathbf{q}_d) = f(\mathbf{q}_0) + \frac{\partial f}{\partial \mathbf{q}} \Big|_{\mathbf{q}=\mathbf{q}_0} (\mathbf{q}_d - \mathbf{q}_0) + \dots$$

Inverse Kinematics via Optimization (2)

What does $\frac{\partial f}{\partial q}$ remind us of?

Inverse Kinematics via Optimization (2)

What does $\frac{\partial f}{\partial \mathbf{q}}$ remind us of?

$$\begin{aligned}\mathbf{x}_d &= f(\mathbf{q}_d) = f(\mathbf{q}_0) + \left. \frac{\partial f}{\partial \mathbf{q}} \right|_{\mathbf{q}=\mathbf{q}_0} (\mathbf{q}_d - \mathbf{q}_0) + \dots \\ &= f(\mathbf{q}_0) + \mathbf{J}(\mathbf{q}_0) \Delta \mathbf{q} + \dots \\ \mathbf{J}(\mathbf{q}_0) \Delta \mathbf{q} &= \mathbf{x}_d - f(\mathbf{q}_0)\end{aligned}$$

If we assume that \mathbf{J} is invertible:

$$\Delta \mathbf{q} = \mathbf{J}^{-1}(\mathbf{q}_0)(\mathbf{x}_d - f(\mathbf{q}_0))$$

This is Newton's Algorithm for root finding!:

- 1 $\mathbf{q}_k = \mathbf{q}_0, k = 0$
- 2 $\Delta \mathbf{q} = \mathbf{J}^{-1}(\mathbf{q}_k)(\mathbf{x}_d - f(\mathbf{q}_k))$
- 3 $\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta \mathbf{q}$
- 4 If $\mathbf{x}_d - f(\mathbf{q}_{k+1}) \approx 0$, we stop, otherwise $k = k + 1$ and we go back to step 2

What if J is NOT invertible?

What if J is NOT invertible?

No problem! We can use the Moore–Penrose pseudoinverse:

$$\begin{aligned} J^\dagger &= J^T(JJ^T)^{-1}, \text{ if } n > m, (J^\dagger J = I) \\ J^\dagger &= (J^T J)^{-1} J^T, \text{ if } n < m, (JJ^\dagger = I) \end{aligned}$$

And thus:

$$\Delta q = J^\dagger(q_0)(x_d - f(q_0))$$

Inverse Kinematics via Optimization (4)

But can we take differences for full transformation matrices?!

Let $\mathbf{T}_{wd} \in SE(3)$ be the target transformation matrix, then we get the following algorithm:

- 1 $\mathbf{q}_k = \mathbf{q}_0, k = 0$
- 2 $[\mathcal{V}_b] = \log(\mathbf{T}_{wb}^{-1}(\theta_i) \mathbf{T}_{wd}), \quad \mathbf{T}_{wb}(\theta)$ gives the forward kinematics
- 3 $\Delta \mathbf{q} = \mathbf{J}_b^\dagger(\mathbf{q}_k) \mathcal{V}_b, \quad \mathbf{J}_b$ is the body Jacobian
- 4 $\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta \mathbf{q}$
- 5 If $\|\mathcal{V}_b\| \approx 0$, we stop, otherwise $k = k + 1$ and we go back to step 2

We can do the same computations using the world space Jacobians (\mathbf{J}_w) and errors ($[\mathcal{V}_w] = [Ad_{\mathbf{T}_{wb}}] \mathcal{V}_b = \log(\mathbf{T}_{wd} \mathbf{T}_{wb}^{-1}(\theta_i)))$.

IK via Optimization - Code Example

```
IT += 1
print("Found solution in %d iterations with error: %s" % (it, error.T))

Found solution in 6 iterations with error: [-6.97036514e-08  1.79748729e-16  1.01023193e-07 -3.88578059e-16
 1.84283303e-08  7.63278329e-17]

[65]: # Validation
fk_all(model, data, q)
for frame, oMf in zip(model.frames, data.oMf):
    if "link" not in frame.name:
        continue
    print("%(name)s : %(x).2f %(y).2f %(z).2f"
          .format(frame.name, *oMf.translation.T.flat)))
print("====")

# target configuration
fk_all(model, data, qd)
for frame, oMf in zip(model.frames, data.oMf):
    if "link" not in frame.name:
        continue
    print("%(name)s : %(x).2f %(y).2f %(z).2f"
          .format(frame.name, *oMf.translation.T.flat)))
print("====")

base_link :  0.00  0.00  0.00
arm_link_0 :  0.00  0.00  0.00
arm_link_1 :  0.00  0.00  0.05
arm_link_2 :  0.00  0.00  0.18
arm_link_3 : -0.03  0.09  0.48
arm_link_4 : -0.11  0.28  0.46
arm_link_5 : -0.16  0.43  0.49
=====
base_link :  0.00  0.00  0.00
arm_link_0 :  0.00  0.00  0.00
arm_link_1 :  0.00  0.00  0.05
arm_link_2 :  0.00  0.00  0.18
arm_link_3 : -0.11  0.29  0.26
arm_link_4 : -0.11  0.28  0.46
arm_link_5 : -0.16  0.43  0.49
=====
```

Newton's method for IK has several problems:

- It is local, i.e. \mathbf{q}_0 should close to the solution
- It can “break” when we are close to singularities:
 - the Jacobian matrix loses one rank!
 - the end-effector cannot move in some direction(s)
- The final \mathbf{q}^* values can violate the robot joint limits!!

Alternative method:

- Minimize the error
- $[\mathcal{V}_b] = \log(\mathbf{T}_{wb}^{-1}(\mathbf{q}_k) \mathbf{T}_{wd})$ gives us the error
- We want this to be as small as possible; this is an optimization problem!
- We know how to add constraints, no?

Quadratic Programming (QP)

$$\begin{aligned}\min_{\mathbf{x}} f(\mathbf{x}) &= \frac{1}{2} \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{q}^T \mathbf{x} \\ \text{s.t. } \mathbf{A} \mathbf{x} - \mathbf{b} &= \mathbf{0} \\ \mathbf{C} \mathbf{x} - \mathbf{d} &\leq \mathbf{0}\end{aligned}$$

where $\mathbf{x}, \mathbf{q} \in \mathbb{R}^N$, $\mathbf{Q} > 0 \in \mathbb{R}^{N \times N}$. Let's define the Langragian and KKT conditions.

Langragian:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T (\mathbf{A} \mathbf{x} - \mathbf{b}) + \boldsymbol{\mu}^T (\mathbf{C} \mathbf{x} - \mathbf{d})$$

KKT Conditions:

- 1) $\mathbf{Q} \mathbf{x} + \mathbf{q} + \mathbf{A}^T \boldsymbol{\lambda} + \mathbf{C}^T \boldsymbol{\mu} = \mathbf{0}$
- 2) $\mathbf{A} \mathbf{x} - \mathbf{b} = \mathbf{0}$ and $\mathbf{C} \mathbf{x} - \mathbf{d} \leq \mathbf{0}$
- 3) $\boldsymbol{\mu} \geq \mathbf{0}$
- 4) $\boldsymbol{\mu}^T (\mathbf{C} \mathbf{x} - \mathbf{d}) = \mathbf{0}$

Inverse Kinematics as a QP

- How can we frame IK as a QP?

Inverse Kinematics as a QP

- How can we frame IK as a QP?
- We can't solve the whole problem with QP!

- How can we frame IK as a QP?
 - We can't solve the whole problem with QP!
 - But we can iteratively! At each iteration k , we solve a QP instead of the pseudoinverse
 - We can think of it as a *Sequential Quadratic Programming* procedure!
 - We can add any constraint that we want!
- 1 $\mathbf{q}_k = \mathbf{q}_0, k = 0$
 - 2 $\Delta\mathbf{q}$ = solution of a QP that minimizes $||\mathcal{V}_b||$
 - 3 $\mathbf{q}_{k+1} = \mathbf{q}_k + \Delta\mathbf{q}$
 - 4 If $||\mathcal{V}_b|| \approx 0$, we stop, otherwise $k = k + 1$ and we go back to step 2

Inverse Kinematics as a QP (2)

- How can we define the objectives, constraints?
- We first need to find the optimization variables!
- We then need to define Q, q, A, b, C, d !
- Let's build this:

Inverse Kinematics as a QP (2)

- How can we define the objectives, constraints?
- We first need to find the optimization variables!
- We then need to define Q, q, A, b, C, d !
- Let's build this:
 - We have the desired end-effector "velocity" (error): \mathcal{V}_b
 - We can compute the current velocity as: $\mathcal{V} = J_b(q)v$
 - We need to find v_* such that $\mathcal{V} - \mathcal{V}_b = 0$!

Inverse Kinematics as a QP (2)

- How can we define the objectives, constraints?
- We first need to find the optimization variables!
- We then need to define Q, q, A, b, C, d !
- Let's build this:
 - We have the desired end-effector "velocity" (error): \mathcal{V}_b
 - We can compute the current velocity as: $\mathcal{V} = J_b(q)v$
 - We need to find v_* such that $\mathcal{V} - \mathcal{V}_b = 0$!
 - We use for variables $x = v \in \mathbb{R}^n$!
 - $Q = J_b^T J_b \in \mathbb{R}^{n \times n}$
 - $q = -J_b^T \mathcal{V}_b \in \mathbb{R}^n$
 - We do not have any equality contraints



$$C = \begin{bmatrix} dt & \dots & \dots & \dots \\ \dots & dt & \dots & \dots \\ & & \ddots & \\ \dots & \dots & -dt & \dots \\ \dots & \dots & \dots & -dt \end{bmatrix} \in \mathbb{R}^{2n \times n}, d = \begin{bmatrix} q_{\max} - q_k \\ q_{\max} - q_k \\ \vdots \\ q_k - q_{\min} \\ q_k - q_{\min} \end{bmatrix} \in \mathbb{R}^{2n}$$

IK via QP - Code Example

```
# Let's compute the QP matrices
Q = J.T @ J
q = -J.T @ error
C = np.eye(model.nv) * step
d_min = model.lowerPositionLimit - q_k
d_max = model.upperPositionLimit - q_k
if it == 0: # in first iteration we initialize the model
    qp.init(Q, q, None, None, C, d_min, d_max)
else: # otherwise, we update the model
    qp.update(Q, q, None, None, C, d_min, d_max)
# Let's solve the QP
qp.solve()
# We get back the results
v = np.copy(qp.results.x)
# Compute next q_k given the velocity
q_k = pin.integrate(model, q_k, v * step)
it += 1
if success:
    print("Found solution in %d iterations with error: %s" % (it, error.T))
else:
    print("Could not find solution in %d iterations! Error: %s" % (it, error.T))
print(q_k.T)
```

Found solution in 4 iterations with error: [1.03588707e-09 -1.77385800e-17 2.91235072e-09 -1.38777878e-17 -2.99925196e-18 -5.55111512e-17]
[0.76933016 0.66219638 -1.41067107 0.31229063]

Velocity Control in Task-Space

If we have a velocity profile $\mathcal{V}_d(t)$ for the end-effector?

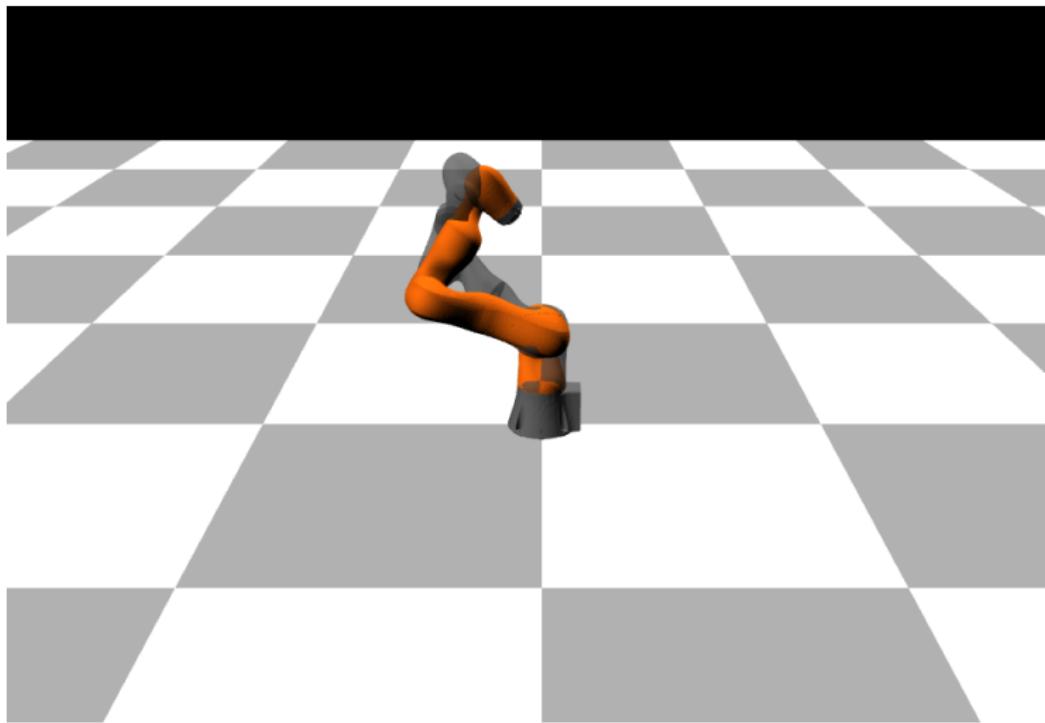
If we have a velocity profile $\mathcal{V}_d(t)$ for the end-effector?

- $\mathcal{V}_b(t) = [Ad_{\mathbf{T}_{wb}}]\mathcal{V}_d(t) + K_p \mathcal{X}_e(t) + K_i \int_0^t \mathcal{X}_e(t) dt, K_p, K_i > 0$
- $[\mathcal{X}_e] = \log(\mathbf{T}_{wb}^{-1} \mathbf{T}_{wd})$
- $\mathcal{V}_b = \mathbf{J}_b \mathbf{v} \implies \mathbf{v}(t) = \mathbf{J}_b^\dagger(\mathbf{q}) \mathcal{V}_b(t)$
- $\mathbf{v} = \mathbf{J}_b^\dagger \mathcal{V}_b$

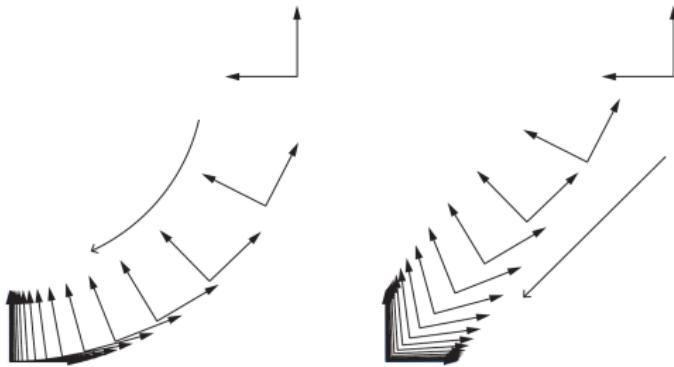
Pseudoinverse can be unstable around singularities, in practice we use:

- $\boldsymbol{v} = \alpha \boldsymbol{J}^T \mathcal{V}, \alpha \in \mathbb{R}^+$
- Damped Pseudoinverse:
 - $\boldsymbol{J}^\dagger = \boldsymbol{J}^T (\boldsymbol{J} \boldsymbol{J}^T + \lambda^2 \boldsymbol{I})^{-1}$, if $n > m$, $(\boldsymbol{J}^\dagger \boldsymbol{J} = \boldsymbol{I})$
 - $\boldsymbol{J}^\dagger = (\boldsymbol{J}^T \boldsymbol{J} + \lambda^2 \boldsymbol{I})^{-1} \boldsymbol{J}^T$, if $n < m$, $(\boldsymbol{J} \boldsymbol{J}^\dagger = \boldsymbol{I})$
 - $\lambda \in \mathbb{R}^+$
- We can use our QP!
 - We solve one QP per timestep with desired velocity $\mathcal{V}(t)$
 - Very effective!
 - We can easily add more cost functions and constraints!

Task-Space Control - Code Example



Separating Position and Orientation



Source: Modern Robotics: Mechanics, Planning, and Control, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press.

What about torque/force control?

How can we control with torque actuators?

What about torque/force control?

How can we control with torque actuators?

- Feedforward/Open-loop control:

$$\tau(t) = M(\mathbf{q}_d(t))\dot{\mathbf{v}}_d(t) + \mathbf{C}_g(\mathbf{q}_d(t), \mathbf{v}_d(t))$$

- Feedback control (PID):

$$\tau(t) = K_p \mathbf{q}_e(t) + K_i \int_0^t \mathbf{q}_e(t) dt + K_d \dot{\mathbf{q}}_e(t), \quad K_p, K_i, K_d > 0$$

- Feedforward plus feedback linearizing controller (Inverse Dynamics Controller):

$$\begin{aligned} \tau(t) = & M(\mathbf{q}(t)) \left(\mathbf{q}_d(t) + K_p \mathbf{q}_e(t) + K_i \int_0^t \mathbf{q}_e(t) dt + K_d \dot{\mathbf{q}}_e(t) \right) \\ & + \mathbf{C}_g(\mathbf{q}_d(t), \mathbf{v}_d(t)), \quad K_p, K_i, K_d > 0 \end{aligned}$$

- Gravity Compensation controller:

$$\tau(t) = K_p \mathbf{q}_e(t) + K_i \int_0^t \mathbf{q}_e(t) dt + K_d \dot{\mathbf{q}}_e(t) + \mathbf{g}(\mathbf{q}(t)), \quad K_p, K_i, K_d > 0$$

Manipulator Equation in Task-Space:

$$\underbrace{\Lambda(\mathbf{q})}_{\text{"Task-Space Inertia Matrix"} \atop \text{}} \dot{\underbrace{\mathcal{V}}_{\text{"End-effector acceleration"}}} + \underbrace{\eta(\mathbf{q}, \mathcal{V})}_{\text{"Coriolis/Gravity Forces"} \atop \text{}} = \underbrace{\mathbf{F}}_{\text{"Wrench at end-effector"}}$$

where:

$$\begin{aligned}\Lambda(\mathbf{q}) &= \mathbf{J}(\mathbf{q})^{-T} \mathbf{M}(\mathbf{q}) \mathbf{J}(\mathbf{q})^{-1} \\ \eta(\mathbf{q}, \mathcal{V}) &= \mathbf{J}(\mathbf{q})^{-T} \mathbf{C}(\mathbf{q}, \mathbf{J}(\mathbf{q})^{-1} \mathcal{V}) - \Lambda(\mathbf{q}) \dot{\mathbf{J}}(\mathbf{q}) \mathbf{J}(\mathbf{q})^{-1} \mathcal{V}\end{aligned}$$

Two Options:

- Any feedback controller in task-space quantities (aka, $SE(3)$), and use the task-space dynamics analogously to the joint space dynamics. For example:

$$\tau(t) = \mathbf{J}_b(\mathbf{q})^T \left(\boldsymbol{\Lambda}(\mathbf{q}) (K_p \mathcal{X}_e(t) + K_i \int_0^t \mathcal{X}_e(t) dt) + \boldsymbol{\eta}(\mathbf{q}, \mathcal{V}) \right),$$

$$K_p, K_i > 0$$

- Any feedback controller in task-space quantities (aka, $SE(3)$), and use no dynamics or joint-space dynamics. For example:

$$\tau = \mathbf{J}_w^T \mathcal{F}_d + \mathbf{M}(\mathbf{q}_d(t)) \dot{\mathbf{v}}_d(t) + \mathbf{C}_g(\mathbf{q}_d(t), \mathbf{v}_d(t))$$

where \mathcal{F}_d is the desired end-effector wrench in world frame.

Null-Space Controllers

When controlling redundant robots in task-space:

- there are multiple solutions at each time-step
- all controllers “optimize” for 1-step in the future
- we might end-up in bad situations!

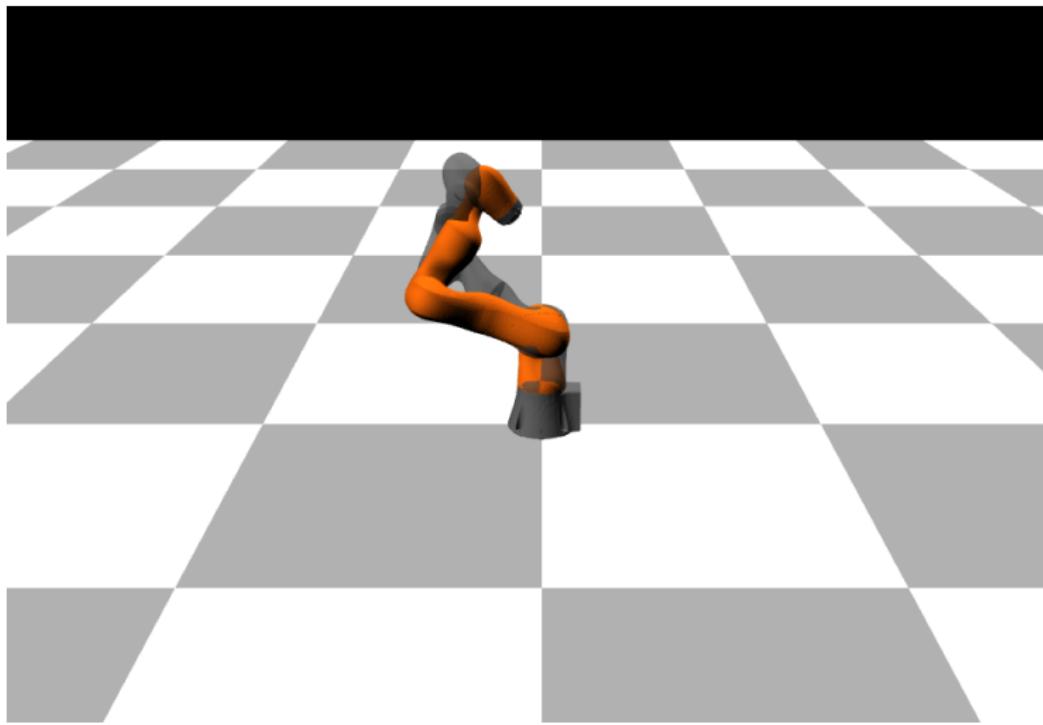
Null-space controllers to the rescue:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{control}} + \underbrace{(\mathbf{I} - \mathbf{J}^T(\mathbf{q})\mathbf{J}^{T\dagger}(\mathbf{q}))\boldsymbol{\tau}_{\text{reg}}}_{\boldsymbol{\tau}_{\text{null}}}$$

where:

- $\mathbf{J}^{T\dagger}$ is the pseudoinverse of \mathbf{J}^T
- $\boldsymbol{\tau}_{\text{reg}}$ is a regularizing control input
- $\boldsymbol{\tau}_{\text{null}}$ does not affect completion of $\boldsymbol{\tau}_{\text{control}}$
- works for velocity control as well!

Null-Space Control - Code Example



Bibliography

Chapters 6, 8 and 11 from **Modern Robotics: Mechanics, Planning, and Control**, *Kevin M. Lynch and Frank C. Park*, 2017, Cambridge University Press. [ebook](#)

Thank you

- Any Questions?
- Office Hours:
 - Tue-Wed (10:00-12:00)
 - 24/7 by email (costashatz@upatras.gr, subject: *ECE_RSI_AM*)
- Material and Announcements



Laboratory of Automation & Robotics