Robotics 2

Dynamic model of robots: Newton-Euler approach

Prof. Alessandro De Luca

Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti

Sapienza Università di Roma

Approaches to dynamic modeling
(reprise)

energy-based approach (Euler-Lagrange) $\rightleftharpoons$ **Newton-Euler method (balance of forces/torques)**

energy-based approach (Euler-Lagrange)

- multi-body robot seen as a whole
- constraint (internal) reaction forces between the links are automatically eliminated: in fact, they do not perform work
- closed-form (symbolic) equations are directly obtained
- best suited for study of dynamic properties and **analysis** of control schemes

Newton-Euler method (balance of forces/torques)

- dynamic equations written separately for each link/body
- inverse dynamics in real time
  - equations are evaluated in a numeric and recursive way
  - best for **synthesis** (=implementation) of model-based control schemes
- by elimination of reaction forces and back-substitution of expressions, we still get closed-form dynamic equations (identical to those of Euler-Lagrange!)

Derivative of a vector in a moving frame

… from velocity to acceleration

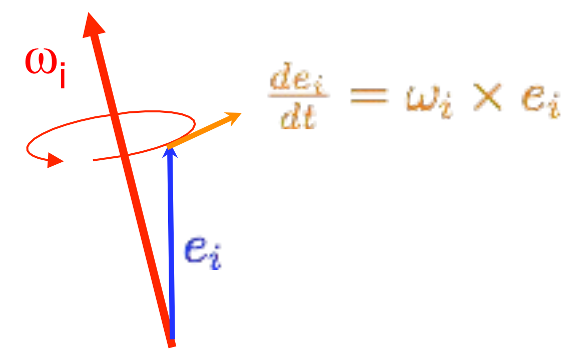
$$ {}^0v_i = {}^0R_i\, {}^iv_i \qquad\qquad {}^0\dot{R}_i = S\left({}^0\omega_i\right){}^0R_i $$

$$ {}^0\dot{v}_i = {}^0a_i = {}^0R_i\, {}^ia_i = {}^0R_i\, {}^i\dot{v}_i + {}^0\dot{R}_i\, {}^iv_i $$

$$ = {}^0R_i\, {}^i\dot{v}_i + {}^0\omega_i \times {}^0R_i\, {}^iv_i = {}^0R_i \left({}^i\dot{v}_i + {}^i\omega_i \times {}^iv_i \right) $$

$$ \boxed{{}^ia_i = {}^i\dot{v}_i + {}^i\omega_i \times {}^iv_i} $$

derivative of "unit" vector



Dynamics of a rigid body

- Newton dynamic equation
  - balance: sum of forces = variation of linear momentum

$$\sum f_i = \frac{d}{dt}\left(m v_c\right) = m \dot{v}_c$$

- Euler dynamic equation
  - balance: sum of torques = variation of angular momentum

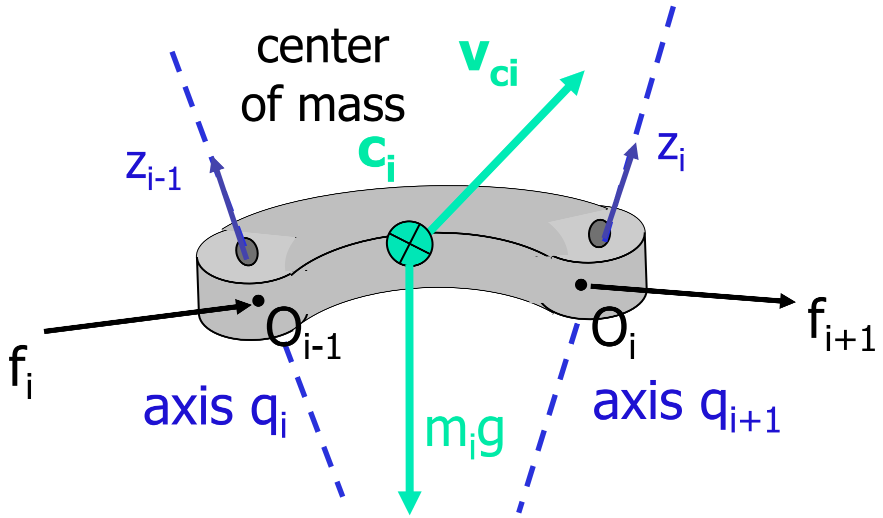
$$\sum \mu_i = \frac{d}{dt}(I\omega) = I\dot{\omega} + \frac{d}{dt}\left(R\bar{I}R^T\right)\omega = I\dot{\omega} + \left(\dot{R}\bar{I}R^T + R\bar{I}\dot{R}^T\right)\omega$$
$$= I\dot{\omega} + S(\omega)R\bar{I}R^T\omega + R\bar{I}R^T S^T(\omega)\cancel{\omega} = I\dot{\omega} + \omega \times I\omega$$

- principle of action and reaction
  - forces/torques: applied by body i to body i+1

    = - applied by body i+1 to body i

Newton-Euler equations - 1

link i



FORCES

f_i force applied from link (i-1) on link i

f_{i+1} force applied from link i on link (i+1)

$m_i g$ gravity force

all vectors expressed in the same RF (better RF_i)

Newton equation

$$\boxed{f_i - f_{i+1} + m_i g = m_i a_{ci}}$$

N

absolute linear acceleration of c_i

Newton-Euler equations - 2

link i

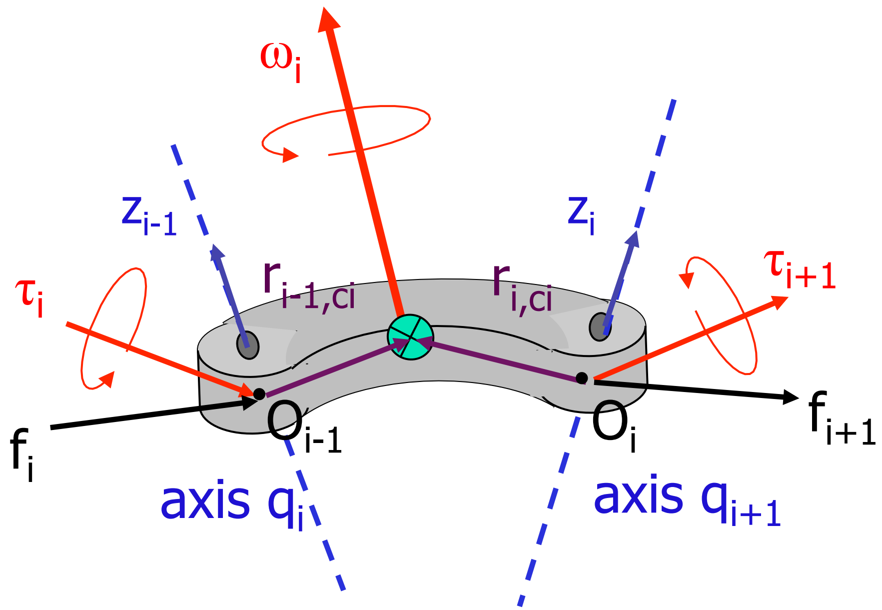
TORQUES

τ_i torque applied from link (i-1) on link i

τ_{i+1} torque applied from link i on link (i+1)

$f_i \times r_{i-1,c}$ torque due to f_i w.r.t. c_i

$- f_{i+1} \times r_{i,c}$ torque due to $- f_{i+1}$ w.r.t. c_i



all vectors expressed in the same RF (RF_i !!)

Euler equation

$$\tau_i - \tau_{i+1} + f_i \times r_{i-1,ci} - f_{i+1} \times r_{i,ci} = I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i) \quad \boxed{E}$$

Forward recursion

Computing velocities and accelerations

- "moving frames" algorithm (as for velocities in Lagrange)
- wherever there is no leading superscript, it is the same as the subscript $(\omega_i = {}^i\omega_i)$
- for simplicity, only revolute joints
  (see textbook for the more general treatment)

initializations

$$\omega_i = {}^{i-1}R_i^T \left[\omega_{i-1} + \dot{q}_i z_{i-1}\right] \longleftarrow \omega_0$$

AR

$$\begin{aligned}\dot{\omega}_i &= {}^{i-1}R_i^T \left[\dot{\omega}_{i-1} + \ddot{q}_i z_{i-1} - \dot{q}_i z_{i-1} \times (\omega_{i-1} + \dot{q}_i z_{i-1})\right] \\ &= {}^{i-1}R_i^T \left[\dot{\omega}_{i-1} + \ddot{q}_i z_{i-1} + \dot{q}_i \omega_{i-1} \times z_{i-1}\right] \longleftarrow \dot{\omega}_0\end{aligned}$$

$$a_i = {}^{i-1}R_i^T a_{i-1} + \dot{\omega}_i \times {}^i r_{i-1,i} + \omega_i \times (\omega_i \times {}^i r_{i-1,i}) \longleftarrow a_0 - {}^0 g$$

$$a_{ci} = a_i + \dot{\omega}_i \times r_{i,ci} + \omega_i \times (\omega_i \times r_{i,ci})$$

the gravity force term can be skipped in Newton equation, <u>if added here</u>

Backward recursion

Computing forces and torques

from N_i → to N_{i-1}

<u>eliminated</u>, if inserted in forward recursion (i=0)

<u>initializations</u>

F/TR

$$f_i = f_{i+1} + m_i(a_{ci} - \cancel{{}^i g}) \qquad \longleftarrow \quad f_{N+1} \quad \tau_{N+1}$$

$$\tau_i = \tau_{i+1} - f_i \times (r_{i-1,i} + r_{i,ci}) + f_{i+1} \times r_{i_c i} + I_i \dot{\omega}_i + \omega_i \times (I_i \omega_i)$$

from E_i → to E_{i-1}

at each step of this recursion, we have two vector equations (N_i + E_i) at the joint providing f_i and τ_i : these contain ALSO the reaction forces/torques at the joint axis ⇒ they should be next "projected" along/around this axis

FP

$$u_i = \begin{cases} f_i^T \, {}^i z_{i-1} + \eta_i \dot{q}_i & \text{for prismatic joint} \\ \tau_i^T \, {}^i z_{i-1} + \eta_i \dot{q}_i & \text{for revolute joint} \end{cases}$$

⇒ N scalar equations at the end

generalized forces (in rhs of Euler-Lagrange eqs)

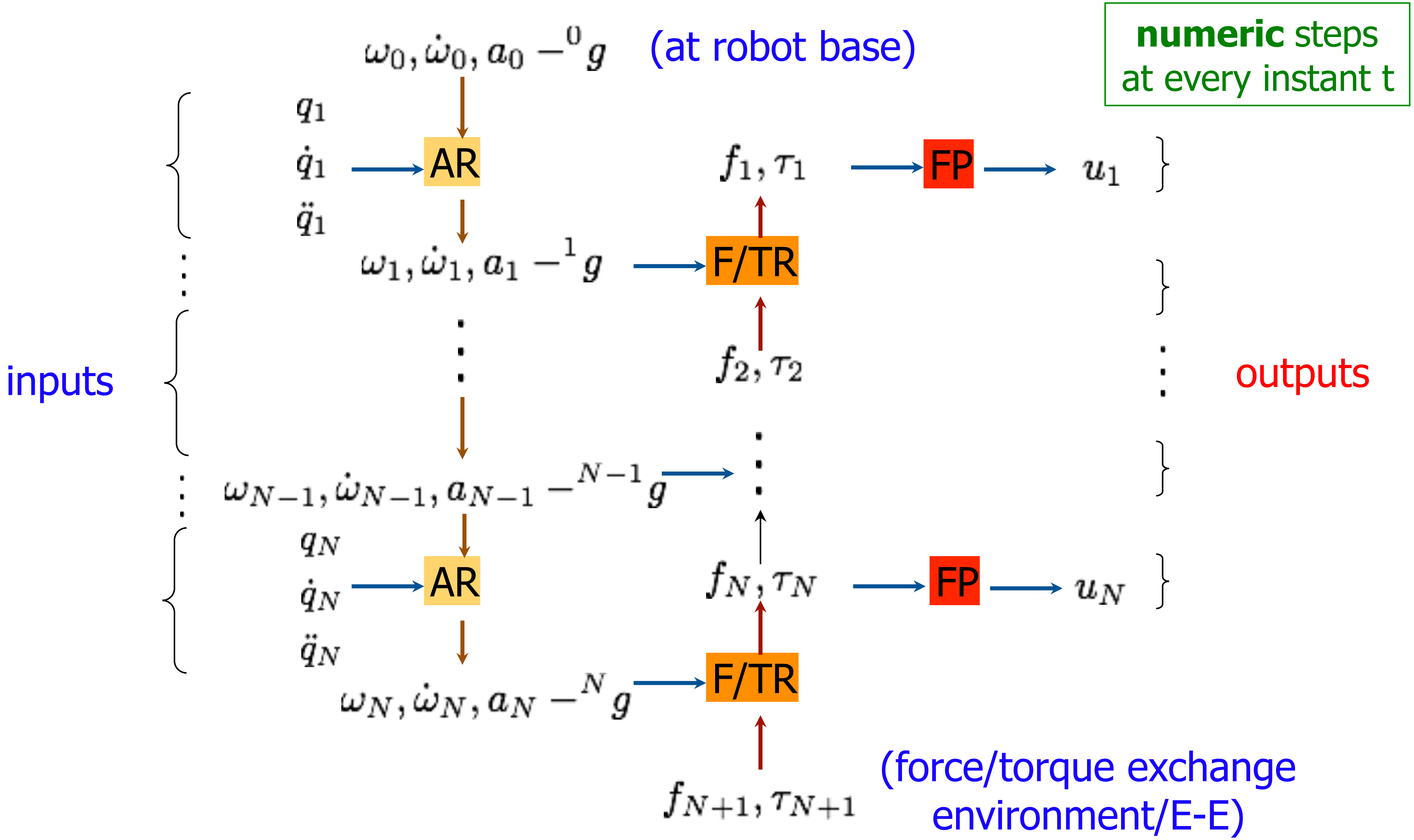
add here dissipative terms (here viscous friction only)

Comments on Newton-Euler method

- the previous forward/backward recursive formulas can be evaluated in symbolic or numeric form
  - symbolic
    - substituting expressions in a recursive way
    - at the end, a closed-form dynamic model is obtained, which is identical to the one obtained using Euler-Lagrange (or any other) method
    - this has no special convenience in using N-E in this way
  - numeric
    - substituting numeric values (numbers!) at each step
    - computational complexity of each step remains constant ⇒ grows in a linear fashion with the number N of joints (O(N))
    - strongly recommended for real-time use, especially when the number N of joints is large

Newton-Euler algorithm

efficient computational scheme for inverse dynamics



Matlab (or C) script

general routine $NE_{\alpha}(arg_1,arg_2,arg_3)$

- data file (of a specific robot)
  - number N and types $\sigma = \{0,1\}^N$ of joints (revolute/prismatic)
  - table of DH kinematic parameters
  - list of dynamic parameters of the links (and of the motors)
- input
  - vector parameter $\alpha = \{{}^0g, 0\}$ (presence or absence of gravity)
  - three ordered vector arguments
    - typically, samples of joint position, velocity, acceleration taken from a desired trajectory
- output
  - generalized force u for the complete inverse dynamics
  - … or single terms of the dynamic model

Examples of output

- complete inverse dynamics

$$u = NE_{0_g}(q_d,\dot{q}_d,\ddot{q}_d) = B(q_d)\ddot{q}_d + c(q_d,\dot{q}_d) + g(q_d) = u_d$$

- gravity terms

$$u = NE_{0_g}(q,0,0) = g(q)$$

- i-th column of the inertia matrix

$$u = NE_0(q,0,e_i) = b_i(q)$$

e_i = i-th column of identity matrix

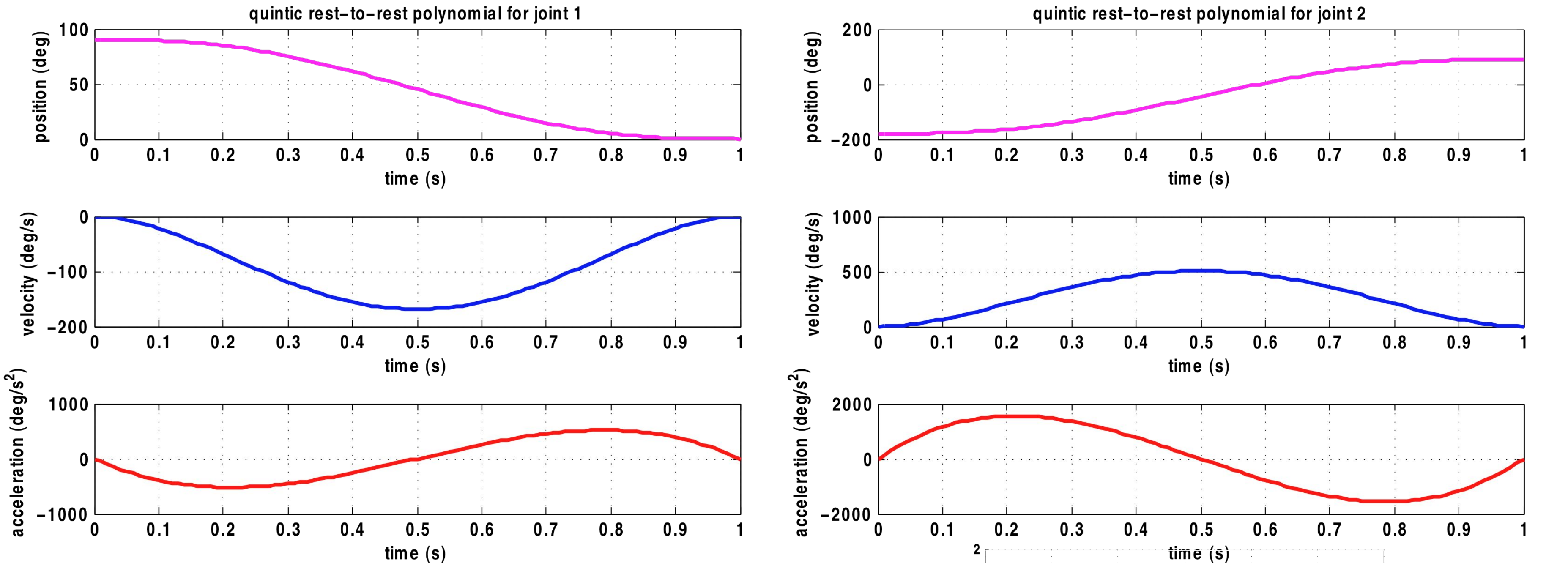
- centrifugal and Coriolis terms

$$u = NE_0(q,\dot{q},0) = c(q,\dot{q})$$

- generalized momentum

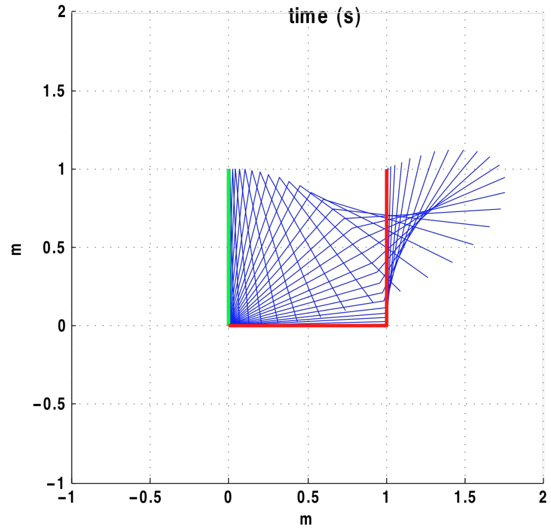
$$u = NE_0(q,0,\dot{q}) = B(q)\dot{q}$$

Inverse dynamics of a 2R planar robot



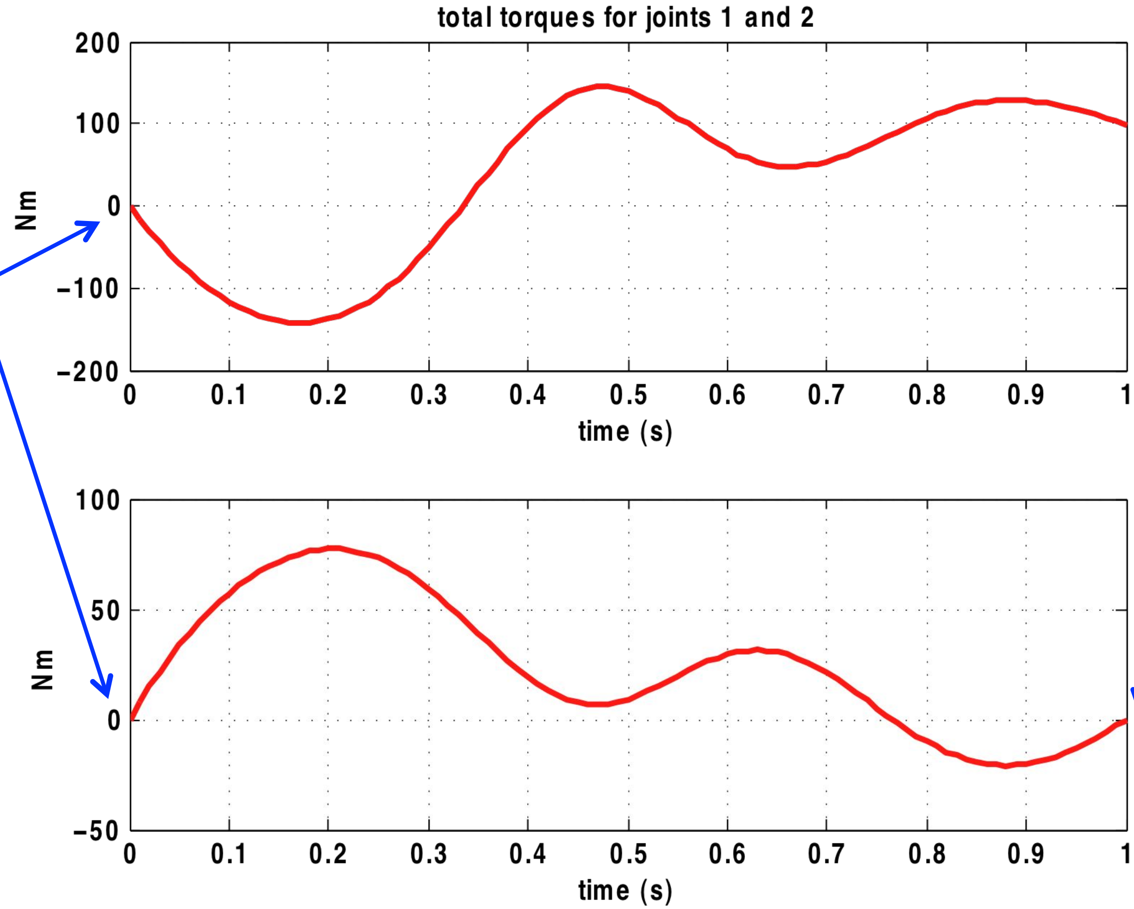
desired (smooth) joint motion:
quintic polynomials for q_1, q_2 with
zero vel/acc boundary conditions
from (90°,-180°) to (0,90°) in T=1 s

⇔



Inverse dynamics of a 2R planar robot

zero
initial torques =
free equilibrium
configuration
+
zero initial
accelerations

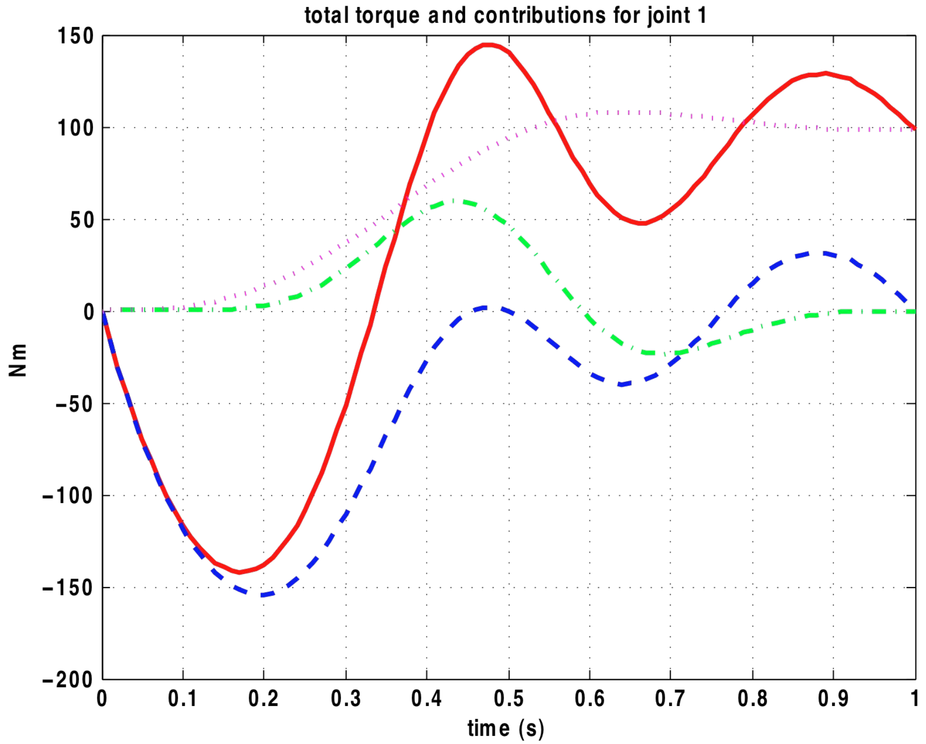


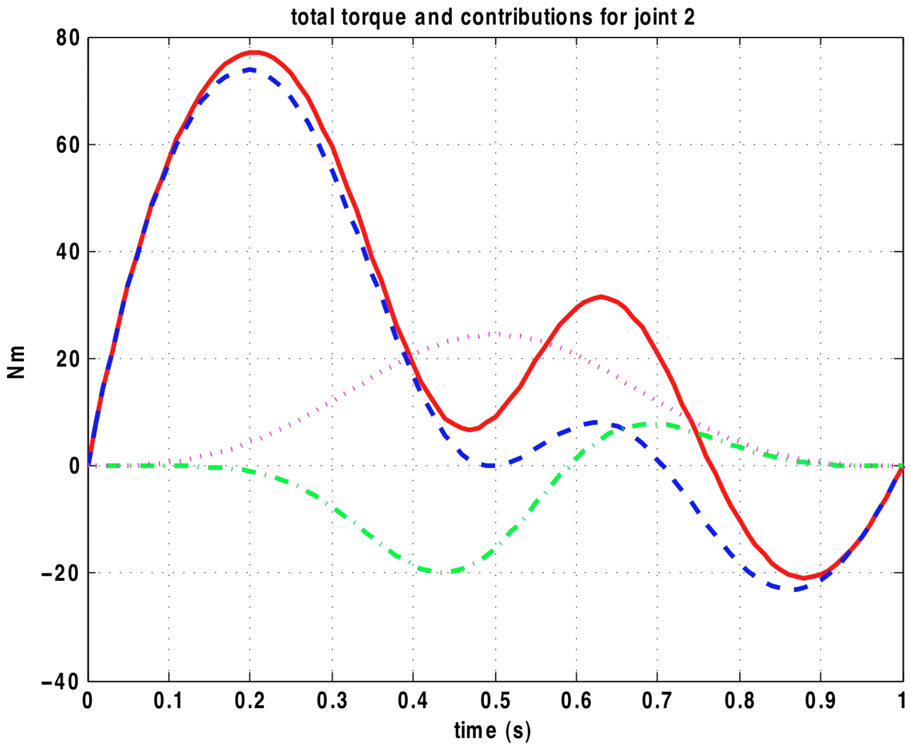
final torques
$u_1 \neq 0$, $u_2 = 0$
balance
link weights
in final (0,90°)
configuration

motion in vertical plane (under gravity)

both links are thin rods of uniform mass $m_1=10$ kg, $m_2=5$ kg

Inverse dynamics of a 2R planar robot





torque contributions at the two joints for the desired motion

—— = total, - - - - = inertial

-·-·- = Coriolis/centrifugal, ········ = gravitational

Use of NE routine for simulation

inverse dynamics

- numerical integration, at current state $(q,\dot{q})$, of

  $$\ddot{q} = B^{-1}(q)\,[u - (c(q,\dot{q}) + g(q))] = B^{-1}(q)\,[u - n(q,\dot{q})]$$

- Coriolis, centrifugal, and gravity terms

  $$n = NE_{0_g}(q,\dot{q},0)$$

  complexity
  O(N)

- i-th column of the inertia matrix, for i =1,..,N

  $$b_i = NE_0(q,0,e_i)$$

  $O(N^2)$

- numerical inversion of inertia matrix

  $$InvB = inv(B)$$

  $O(N^3)$
  but with small coefficient

- given u, integrate acceleration computed as

  $$\ddot{q} = InvB * [u - n] \Longrightarrow$$

  new state $(q,\dot{q})$
  and repeat over time