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) ⇌ **Newton-Euler method (balance of forces/moments)**

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/moments)

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

Derivative of a vector in a moving frame

… from velocity to acceleration

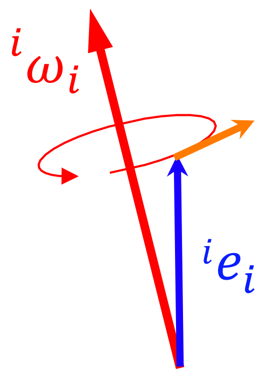
$$ {}^0v_i = {}^0R_i\, {}^iv_i \qquad\qquad {}^0\dot{R}_i = S({}^0\omega_i)\, {}^0R_i $$

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

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

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

derivative of a "unit" vector in a moving frame



$$ \frac{d\, {}^ie_i}{dt} = {}^i\omega_i \times {}^ie_i $$

Dynamics of a rigid body

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

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

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

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

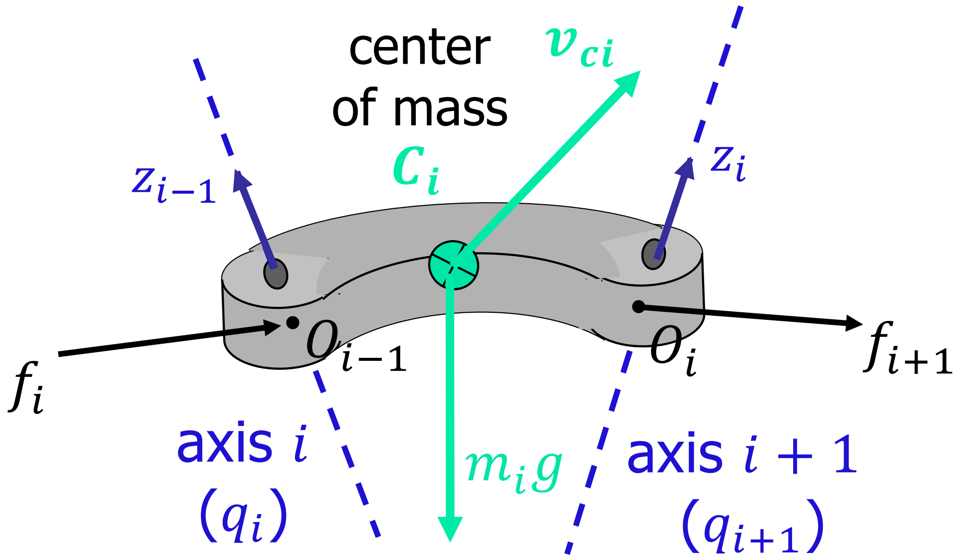
$$= I\dot{\omega} + S(\omega)R\bar{I}R^T\omega + R\bar{I}R^T S^T\cancel{(\omega)\omega} = I\dot{\omega} + \omega \times I\omega$$

- principle of action and reaction
  - forces/moments: 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 in RF$_i$ ...)

Newton equation

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

N

linear acceleration of C_i

Newton-Euler equations - 2

link i

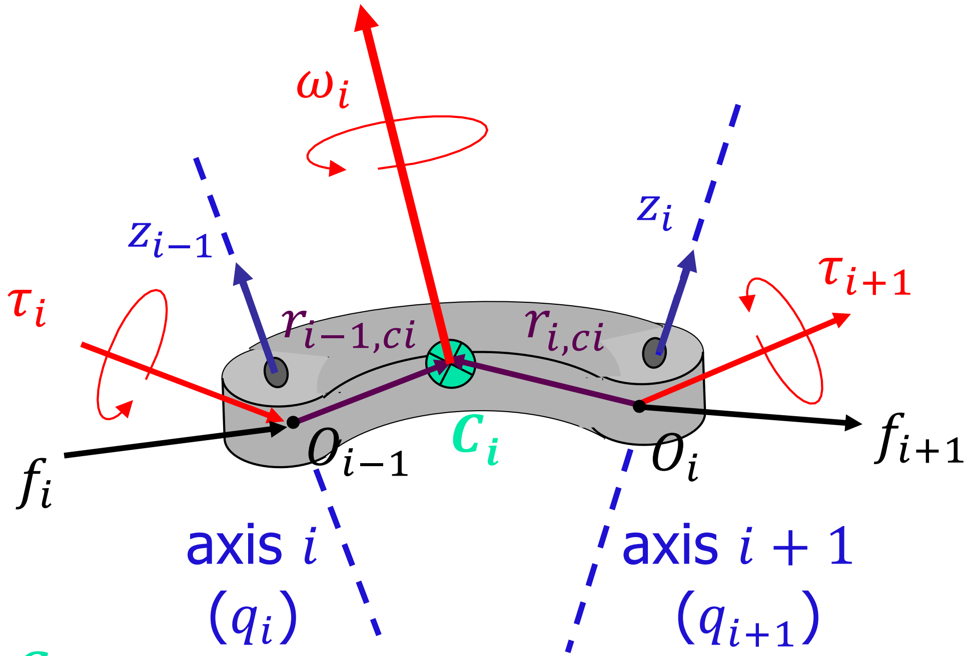
MOMENTS

τ_i moment applied from link $(i-1)$ on link i

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

$f_i \times r_{i-1,ci}$ moment due to f_i w.r.t. C_i

$-f_{i+1} \times r_{i,ci}$ moment due to $-f_{i+1}$ w.r.t. C_i



gravity force gives no moment at C_i

all vectors expressed in the same RF (... RF$_i$!!)

Euler equation

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

angular acceleration of body i

Forward recursion

Computing velocities and accelerations

- "moving frames" algorithm (as for velocities in Lagrange)
- for simplicity, only revolute joints here
  (see textbook for the more general treatment)

initializations

$$ {}^{i}\omega_i = {}^{i-1}R_i^T \left[{}^{i-1}\omega_{i-1} + \dot{q}_i \, {}^{i-1}z_{i-1} \right] \longleftarrow {}^{0}\omega_0 $$

$$ {}^{i}\dot{\omega}_i = {}^{i-1}R_i^T \left[{}^{i-1}\dot{\omega}_{i-1} + \ddot{q}_i \, {}^{i-1}z_{i-1} \right] + {}^{i-1}\dot{R}_i^T \left[{}^{i-1}\omega_{i-1} + \dot{q}_i \, {}^{i-1}z_{i-1} \right] $$

AR

$$ = {}^{i-1}R_i^T \left[{}^{i-1}\dot{\omega}_{i-1} + \ddot{q}_i \, {}^{i-1}z_{i-1} + \dot{q}_i \, {}^{i-1}\omega_{i-1} \times {}^{i-1}z_{i-1} \right] \longleftarrow {}^{0}\dot{\omega}_0 $$

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

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

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

Backward recursion

Computing forces and moments

from N_i → to N_{i-1}

eliminated, if inserted in forward recursion (i=0)

initializations

$$ {}^i f_i = {}^i R_{i+1}\, {}^{i+1} f_{i+1} + m_i \left({}^i a_{ci} - \cancel{{}^i g} \right) \longleftarrow f_{N+1} $$

F/MR

τ_{N+1}

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

from E_i → to E_{i-1}

at each recursion step, the two vector equations ($N_i + E_i$) at joint i provide a wrench $(f_i, \ \tau_i) \in \mathbb{R}^6$): this contains ALSO reaction forces/moments at the joint axis ⇒ to be "projected" along/around this axis to produce work

FP

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

⇒ N scalar equations at the end

generalized forces (in rhs of Euler-Lagrange eqs)

add any dissipative term (here, viscous friction only)

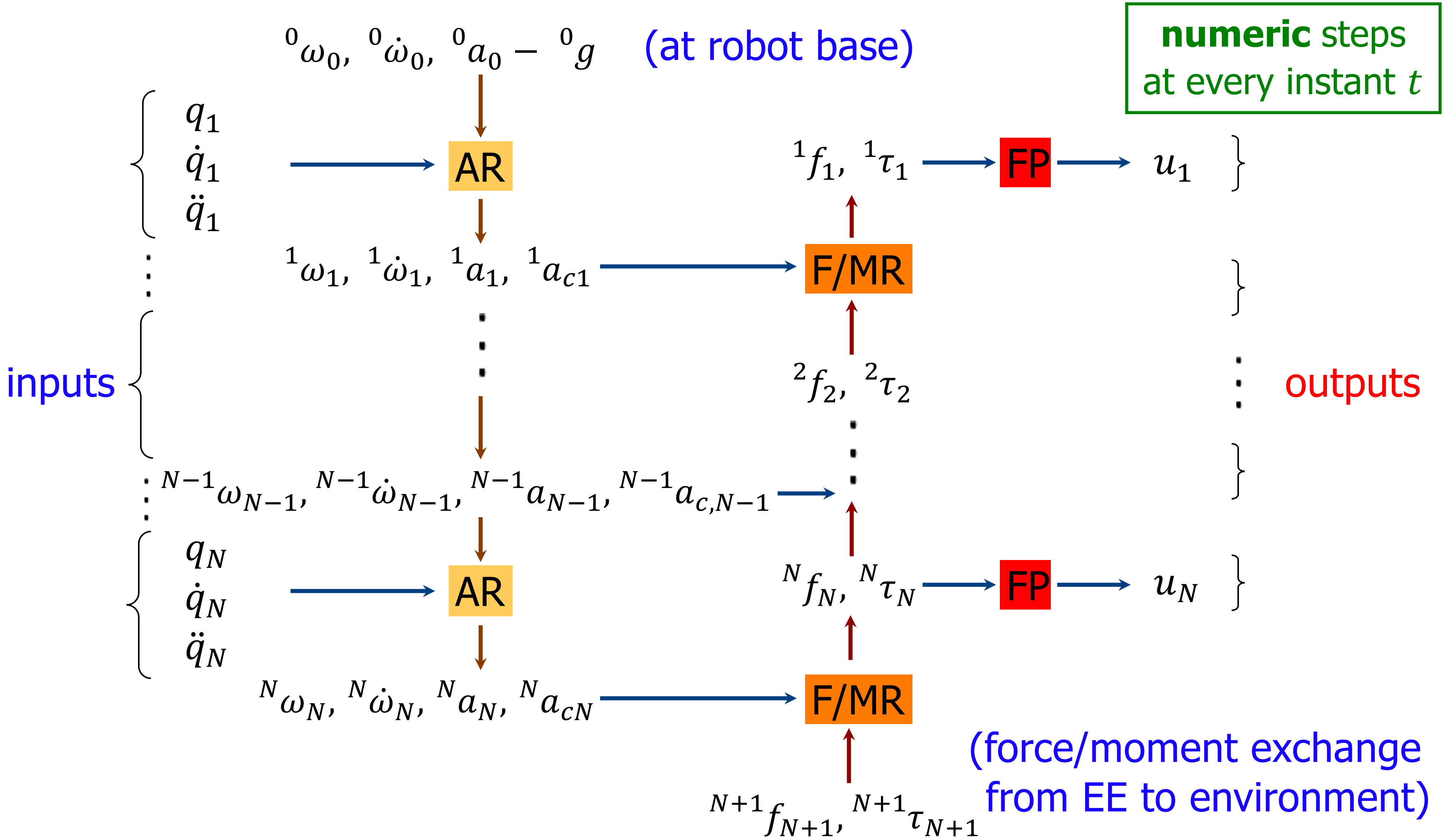
Comments on Newton-Euler method

- 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
    - there is 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

In the lagrange method you have to derive the model for the specific robot, here you just use the numerical tool that is general for each robot

Newton-Euler algorithm

efficient computational scheme for inverse dynamics



Matlab (or C) script

assuming no interaction with the environment ($f_{N+1} = \tau_{N+1} = 0$)

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 ALL 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

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

- gravity term

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

- centrifugal and Coriolis term

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

- i-th column of the inertia matrix

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

e_i = i-th column of identity matrix

- generalized momentum

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

A further example of output

- factorization of centrifugal and Coriolis term

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

- for later use, what about a "mixed" velocity term?

$$S(q, \dot{q})\dot{q}_r \quad \not\Leftrightarrow \quad \begin{cases} u = NE_0(q, \dot{q}_r, 0) = S(q, \dot{q}_r)\dot{q}_r \\ u = NE_0(q, e_i\dot{q}_{ri}, 0) = S_i(q, e_i\dot{q}_{ri})\dot{q}_{ri} \end{cases}$$

no good!

a) $S(q, \dot{q})\dot{q}_r = S(q, \dot{q}_r)\dot{q}$, when using Christoffel symbols

b) $S(q, \dot{q} + \dot{q}_r)(\dot{q} + \dot{q}_r) = S(q, \dot{q})\dot{q} + S(q, \dot{q}_r)\dot{q}_r + 2S(q, \dot{q})\dot{q}_r$

$$\Rightarrow \quad u = \frac{1}{2}\left(NE_0(q, \dot{q} + \dot{q}_r, 0) - NE_0(q, \dot{q}, 0) - NE_0(q, \dot{q}_r, 0)\right)$$

$= S(q, \dot{q})\dot{q}_r$ (i.e., with 3 calls of standard NE algorithm)

[Kawasaki et al., IEEE T-RA 1996]

Modified NE algorithm

modified routine $\widehat{NE}_{\alpha}(\mathrm{arg}_1, \mathrm{arg}_2, \mathrm{arg}_3, \mathrm{arg}_4)$ with 4 arguments

[De Luca, Ferrajoli, ICRA 2009]

$$\widehat{NE}_{\alpha}(x, y, y, z) = NE_{\alpha}(x, y, z)$$ consistency property

e.g., $u = \widehat{NE}_{{}^0 g}(q, 0, 0, 0) = NE_{{}^0 g}(q, 0, 0) = g(q)$

$u = \widehat{NE}_0(q, \dot{q}, \dot{q}, 0) = NE_0(q, \dot{q}, 0) = c(q, \dot{q}) = S(q, \dot{q})\dot{q}$

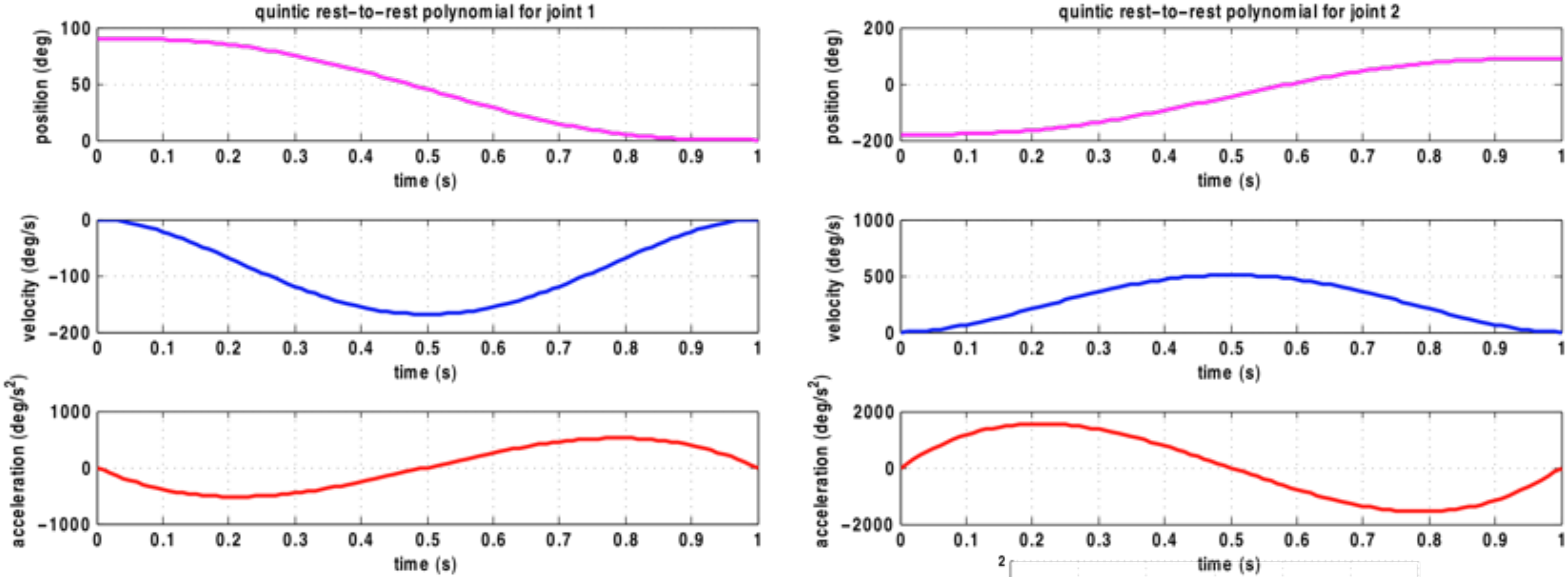
$\Rightarrow$ $u = \widehat{NE}_0(q, \dot{q}, \dot{q}_r, 0) = S(q, \dot{q})\dot{q}_r$ with $\dot{M} - 2S$ skew-symmetric

(i.e., with 1 call of modified NE algorithm)

$\Rightarrow$ $u = \widehat{NE}_0(q, \dot{q}, e_i, 0) = S_i(q, \dot{q})$

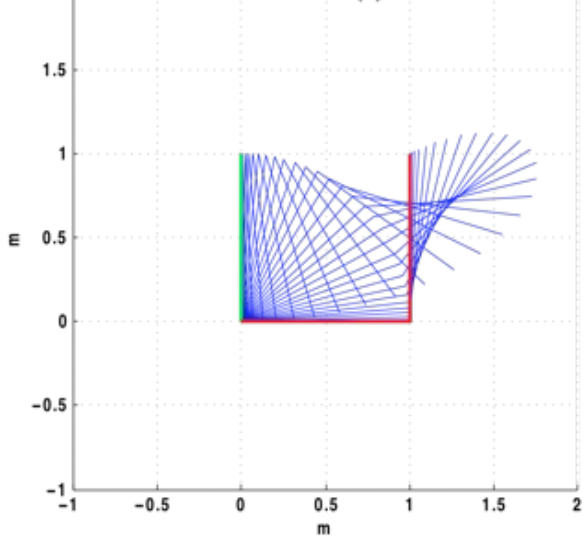
(i.e., the full matrix S satisfying the skew-symmetry of $\dot{M} - 2S$ with N calls of the modified NE algorithm)

Inverse dynamics of a 2R planar robot



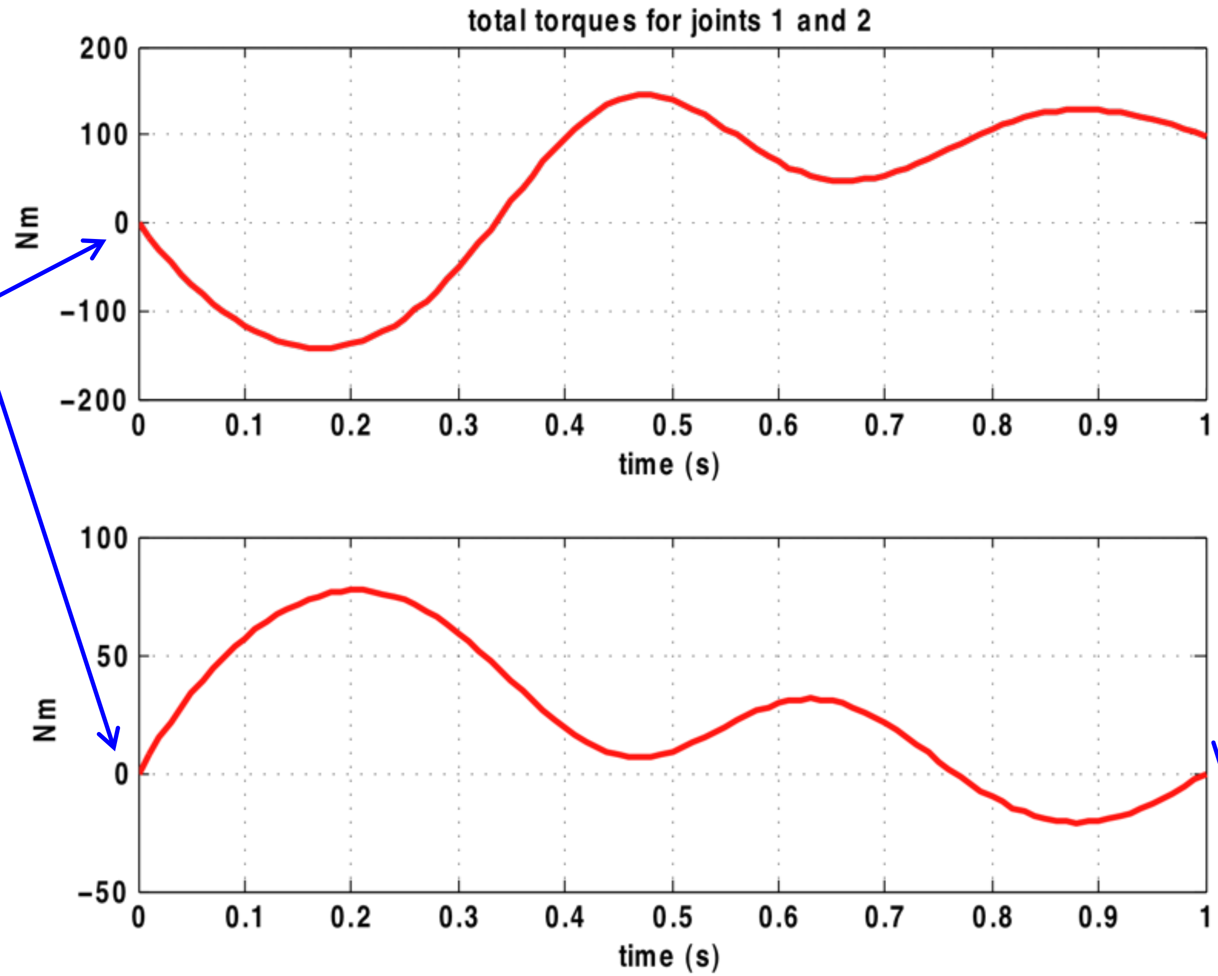
desired (smooth) joint motion:
quintic polynomials for q_1, q_2 with
zero vel/acc boundary conditions
from (90°, -180°) to (0°, 90°) in $T = 1$ s

$\Leftrightarrow$



Inverse dynamics of a 2R planar robot

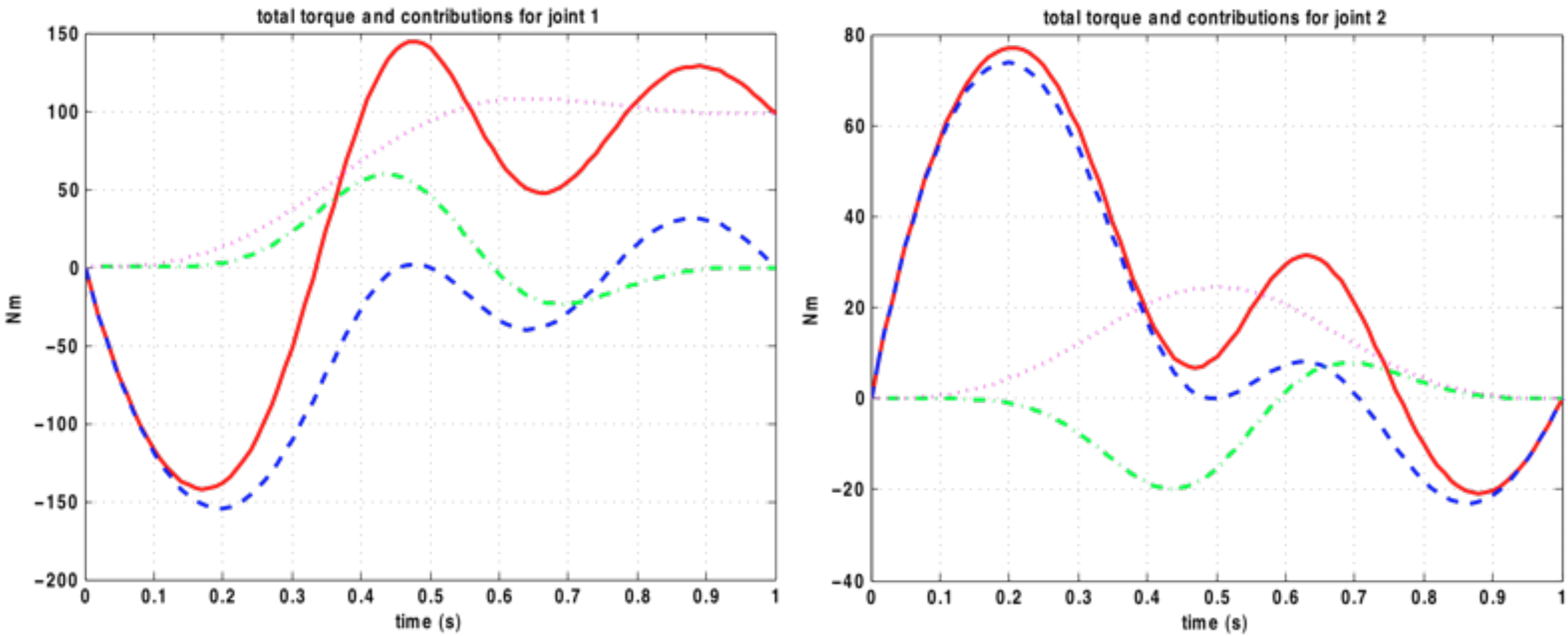
zero
initial torques =
free equilibrium
configuration
+
zero initial
accelerations



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
direct dynamics

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

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

- Coriolis, centrifugal, and gravity terms

$$n = NE_{^0g}(q,\dot{q},0)$$ complexity $O(N)$

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

$$M_i = NE_0(q,0,e_i)$$ $O(N^2)$

- numerical inversion of inertia matrix

$$InvM = \mathrm{inv}(M)$$ $O(N^3)$ but with small coefficient

- given u, integrate acceleration computed as

$$\ddot{q} = InvM * [u - n]$$ ➡ new state $(q,\dot{q})$ and repeat over time ...