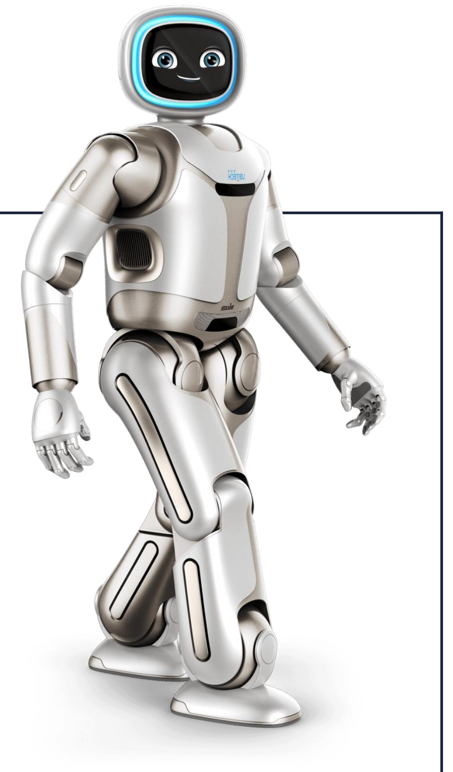
Advanced Robotics

ENGG5402 Spring 2023

Fei Chen

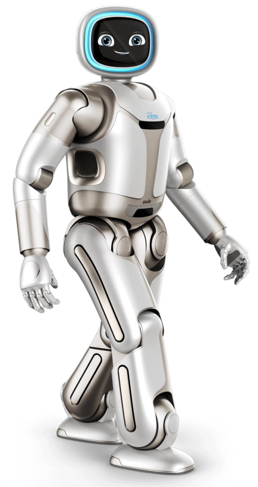


Topics:

- Dynamic model of robots: Newton-Euler approach

Readings:

- Siciliano: Sec. 7.5



Comparison of Two Methods

Approaches to dynamic modeling (reprise)

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!)

Vector in Moving Frame

Derivative of a vector in a moving frame

… from velocity to acceleration

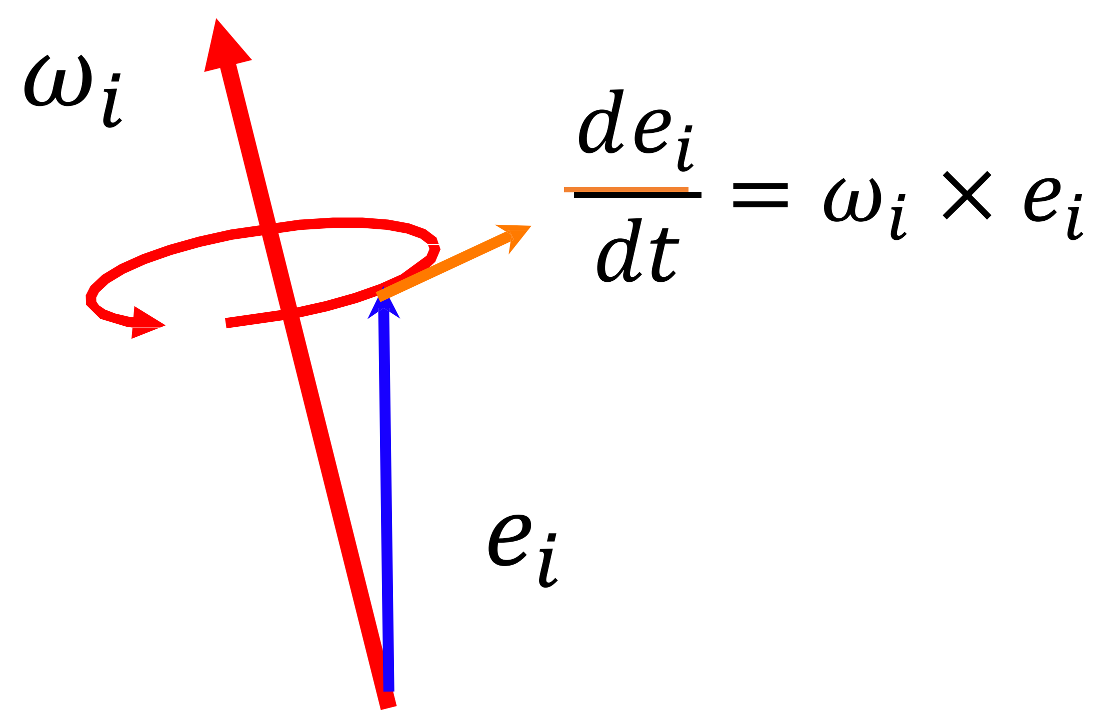
$$ {}^{0}v_i = {}^{0}R_i \, {}^{i}v_i \qquad\qquad {}^{0}\dot{R}_i = S({}^{0}\omega_i)\, {}^{0}R_i $$

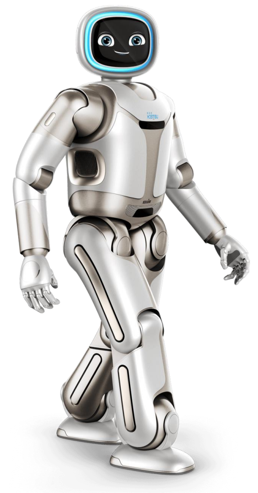
$$ {}^{0}\dot{v}_i = {}^{0}a_i = {}^{0}R_i \, {}^{i}a_i = {}^{0}R_i \, {}^{i}\dot{v}_i + {}^{0}\dot{R}_i \, {}^{i}v_i $$

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

$$ {}^{i}a_i = {}^{i}\dot{v}_i + {}^{i}\omega_i \times {}^{i}v_i $$

derivative of "unit" vector





Dynamics of a Rigid Body

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

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

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

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

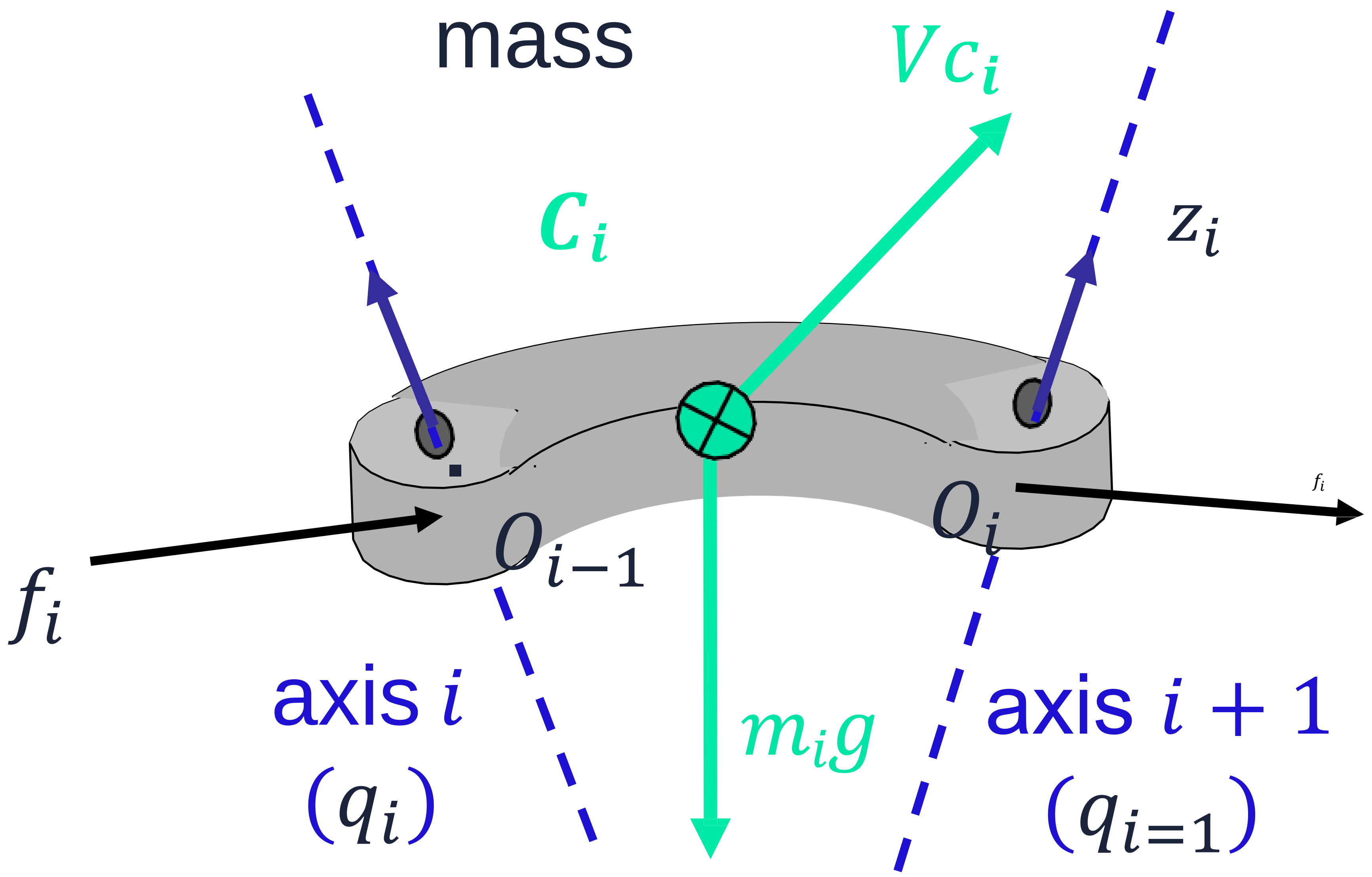
$$= I\dot{\omega} + S(\omega)R\bar{I}R^T\omega + R\bar{I}R^T S^T(\omega)\omega = I\dot{\omega} + \omega \times I\omega$$

- 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

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)

N Newton equation

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

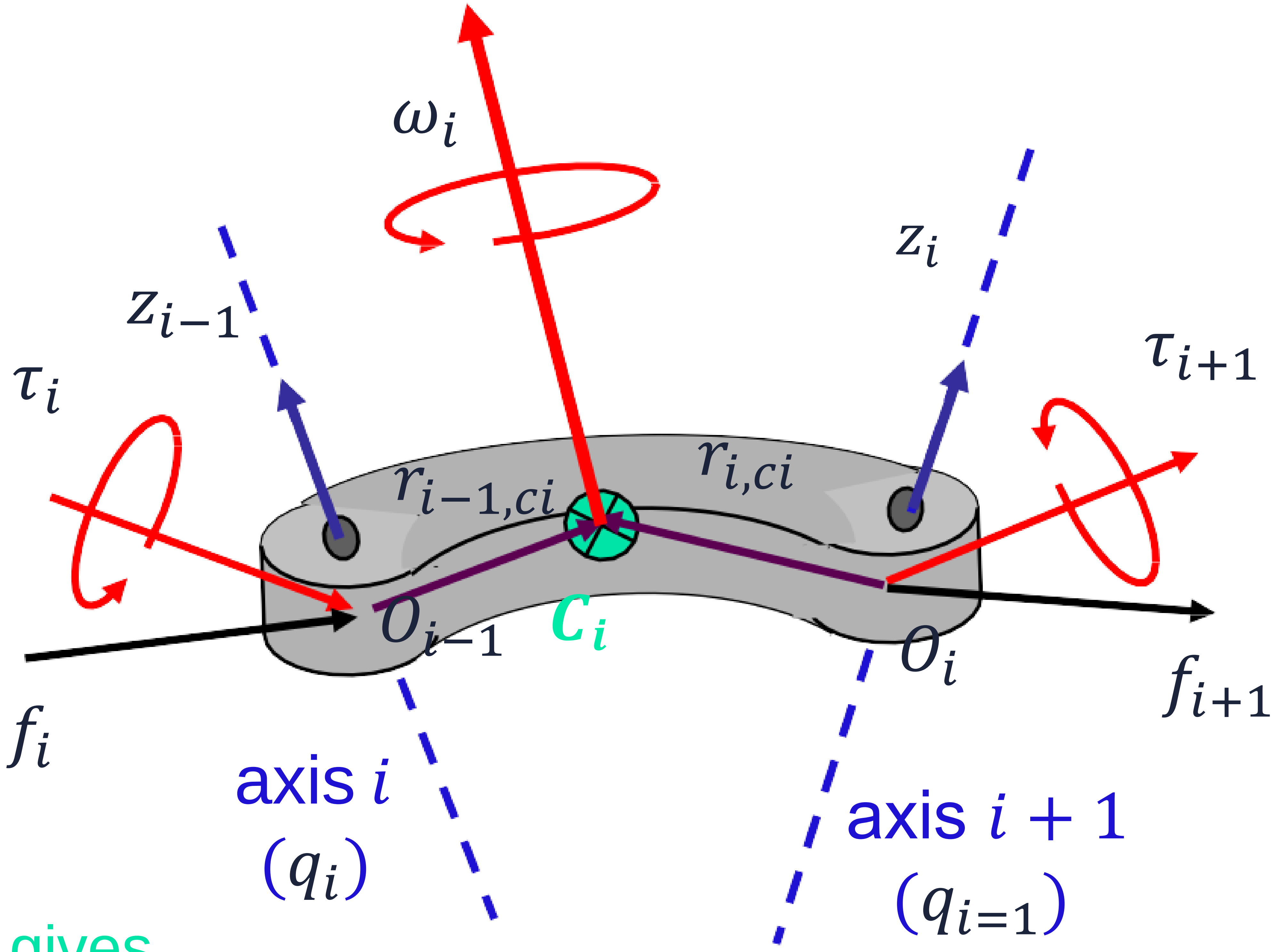
linear acceleration of C_i

Newton-Euler Equations

link i

TORQUES

- τ_i torque applied from link $(i-1)$ on link i
- τ_{i+1} torque applied from link i on link $(i+1)$
- $f_i \times r_{i-1,ci}$ torque due to f_i w.r.t. C_i
- $-f_{i+1} \times r_{i,ci}$ torque due to $-f_{i+1}$ w.r.t. C_i



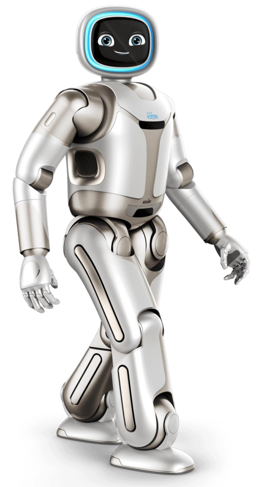
gravity force gives no torque at C_i

E Euler equation

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

angular acceleration of body i

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



Forward Recursion

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

AR

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

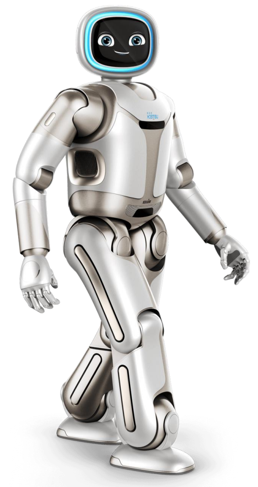
$$\dot{\omega}_i = {}^{i-1}R_i^T[\dot{\omega}_{i-1} + \ddot{q}_i z_{i-1} - \dot{q}_i z_{i-1} \times (\omega_{i-1} + \dot{q}_i z_{i-1})]$$

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

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

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

the gravity force term can be skipped in Newton equation, if added here



Backward Recursion

Backward recursion (Computing forces and torques)

from N_i → to N_{i-1}

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

initializations

F/TR

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

τ_{N+1} ↓

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

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 "projected" next along/around this axis

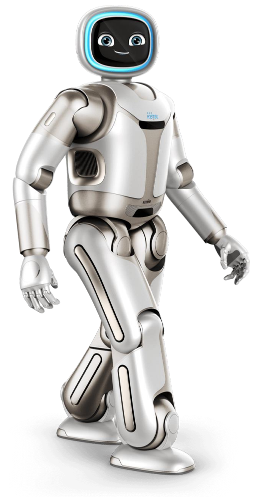
FP

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

generalized forces (in rhs of Euler-Lagrange eqs)

add here dissipative terms (here viscous friction only)

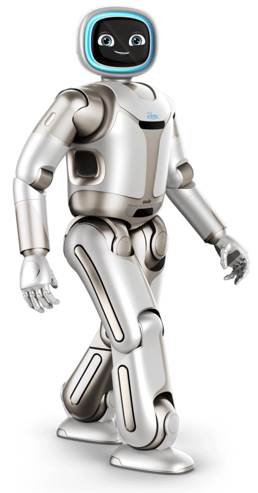
⇒ N scalar equations at the end



Comments

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



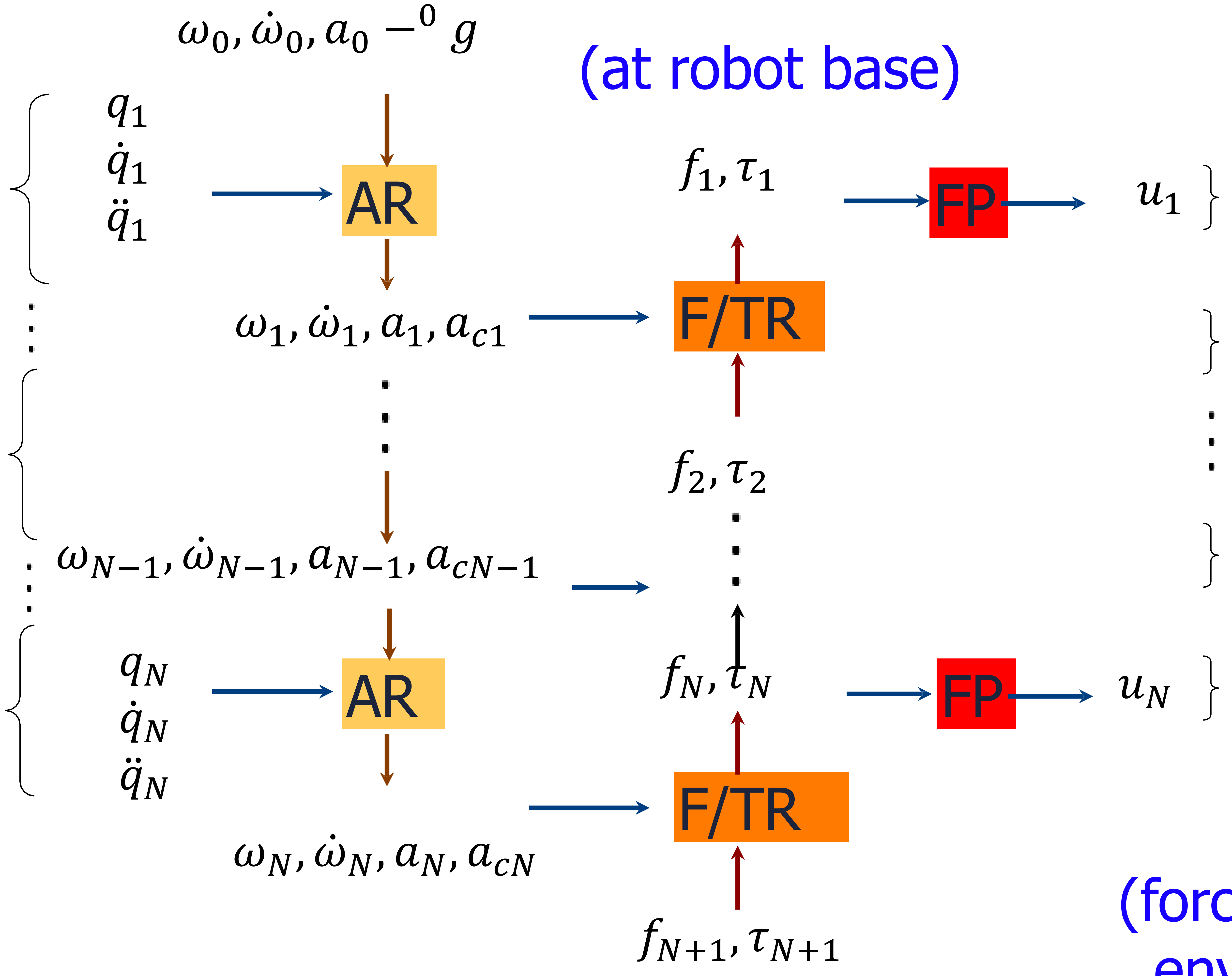
Newton-Euler Algorithm

Newton-Euler algorithm

(efficient computational scheme for inverse dynamics)

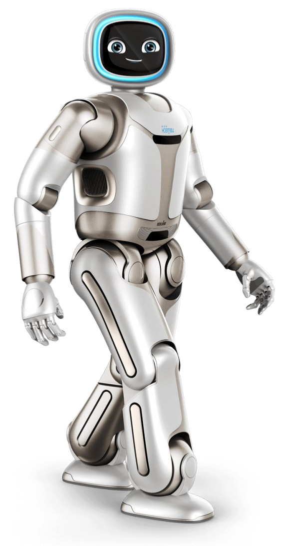
numeric steps at every instant t

inputs



outputs

(force/torque exchange environment/E-E)

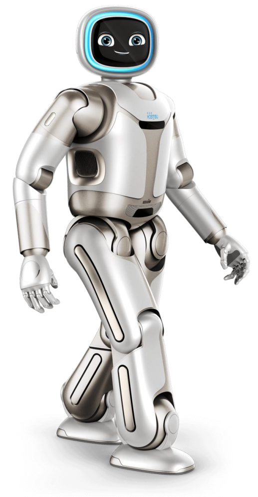


Coding

Matlab (or C, Python) script

general routine $NE_{\alpha}(\mathrm{arg}_1, \mathrm{arg}_2, \mathrm{arg}_3)$

- 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



Output

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 terms

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

- centrifugal and Coriolis terms

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

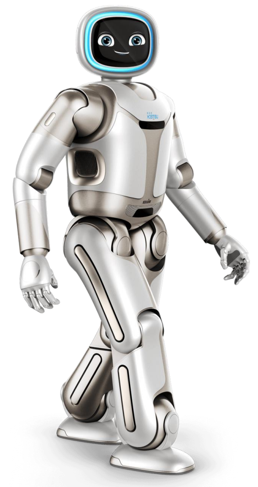
- i-th column of the inertia matrix

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

$e_i = i - th$ column of identity matrix

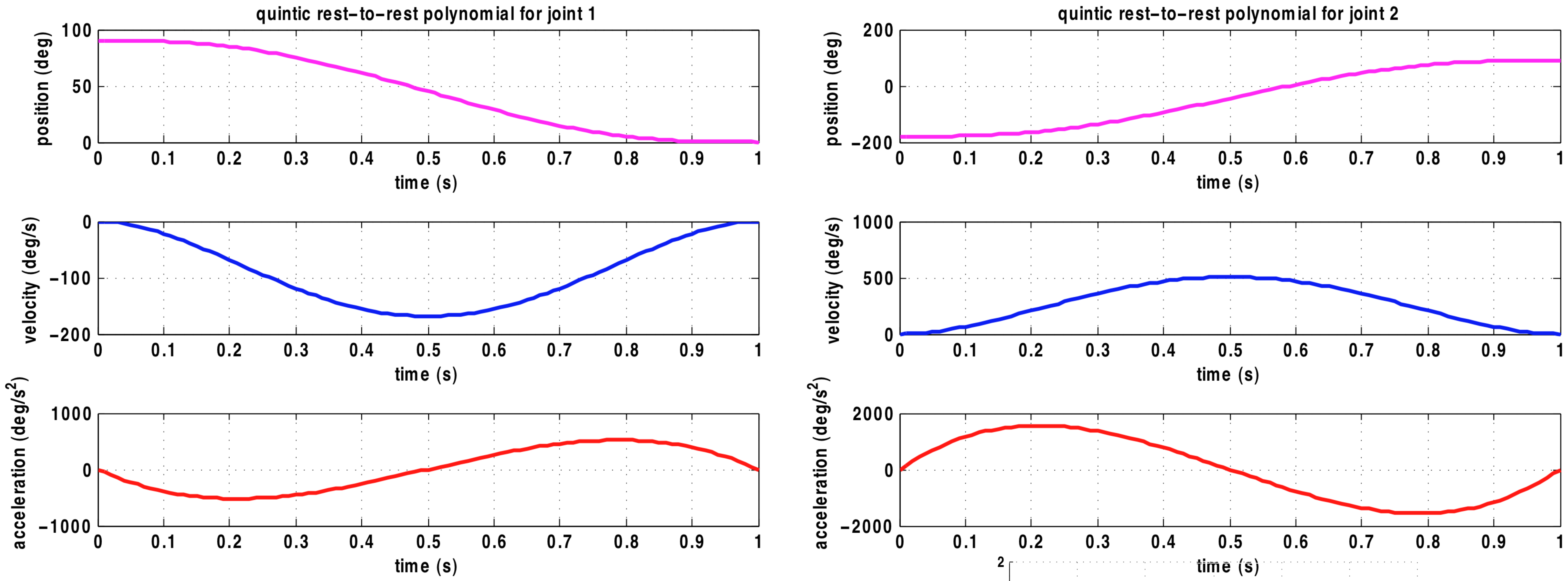
- generalized momentum

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



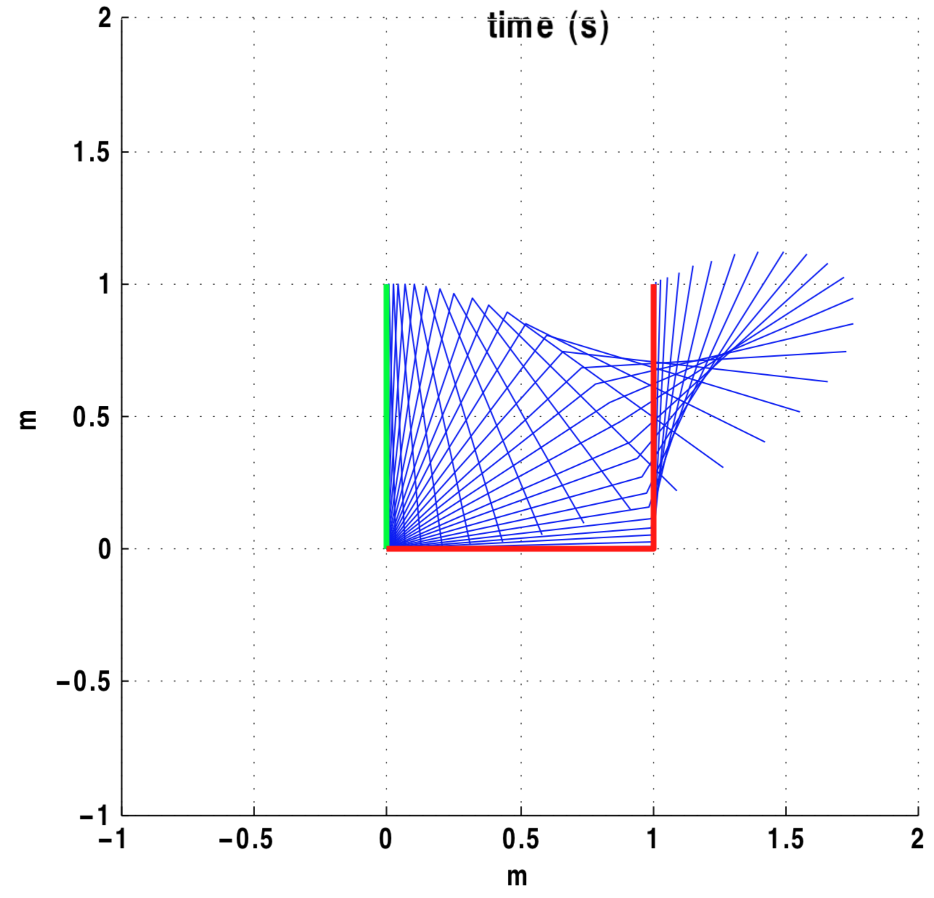
Example

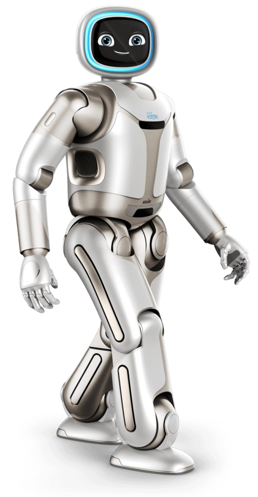
Inverse dynamics of a 2R planar robot



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

⇔

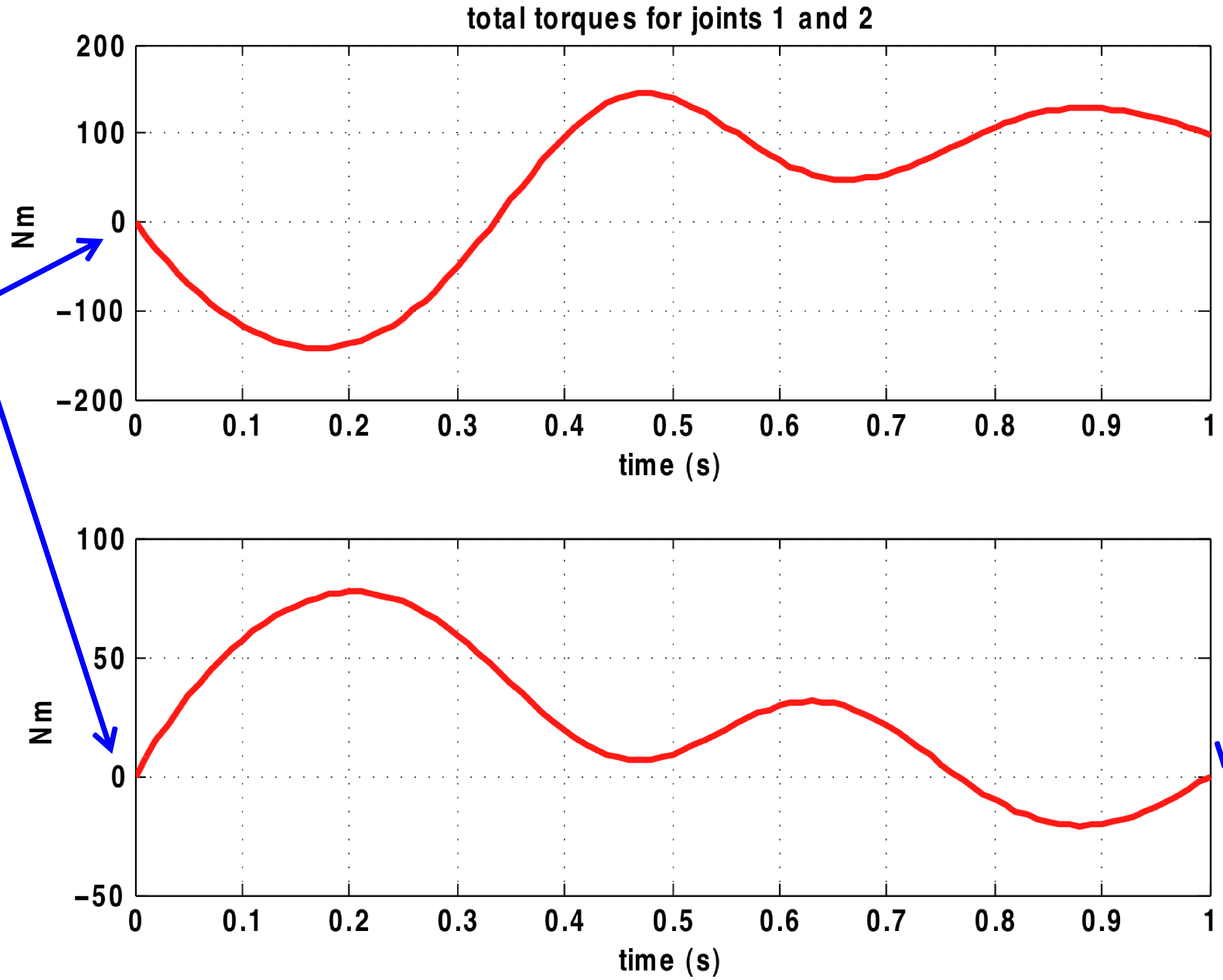




Example

Inverse dynamics of a 2R planar robot

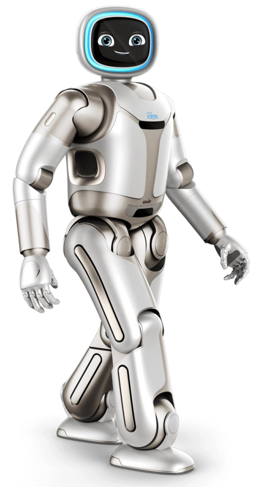
zero initial torques = free equilibrium configuration
+
zero initial accelerations



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

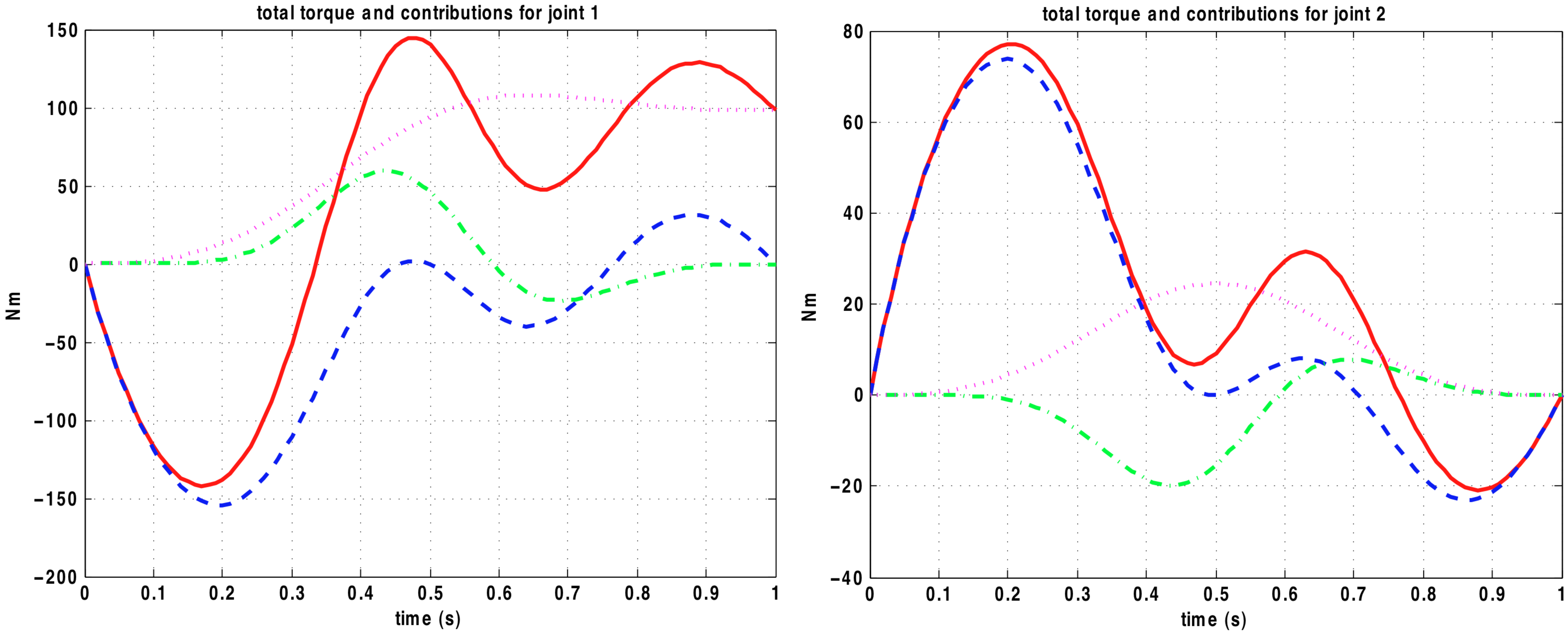
motion in vertical plane (under gravity)

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



Example

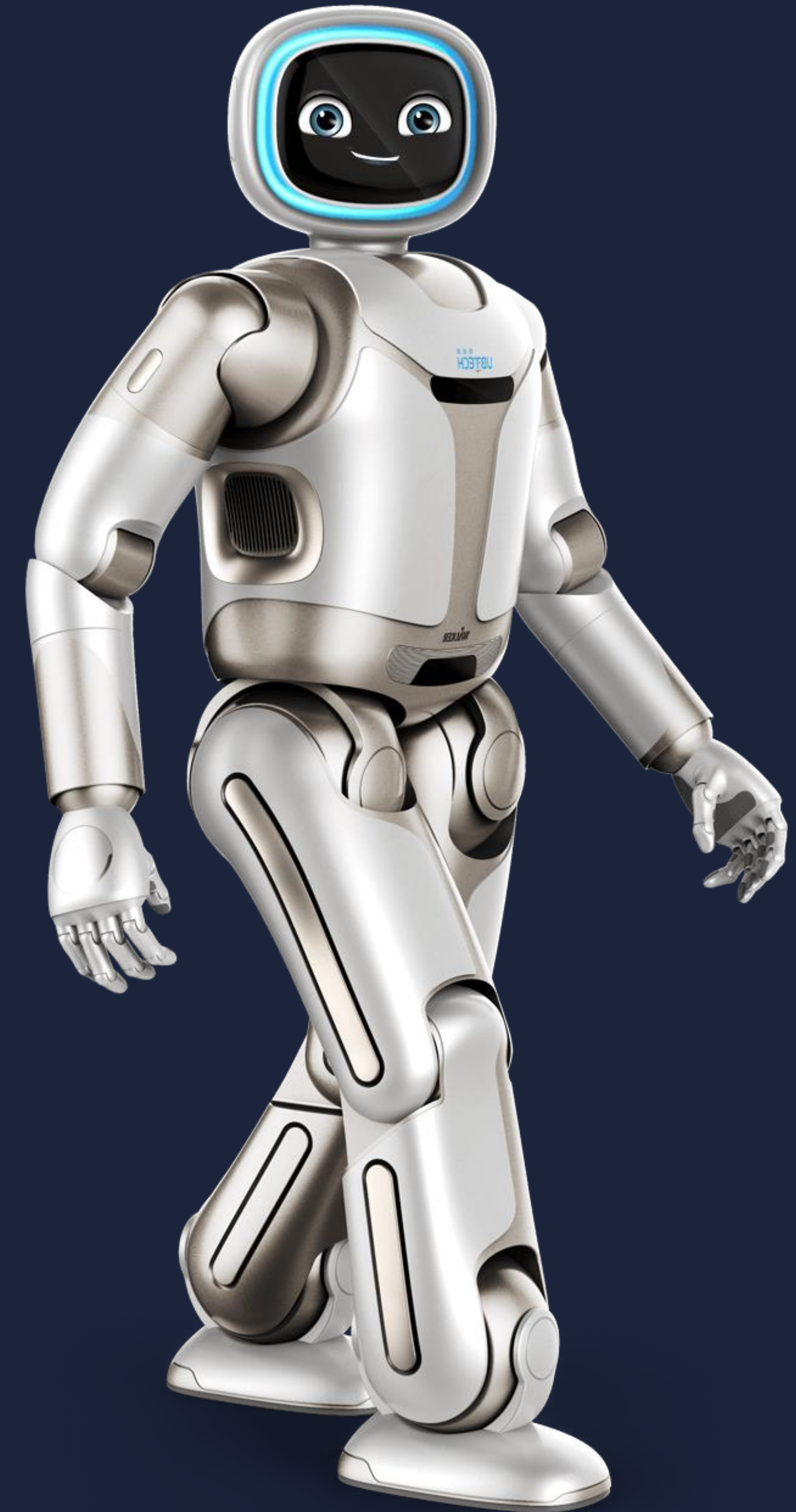
Inverse dynamics of a 2R planar robot



torque contributions at the two joints for the desired motion

= total, = inertial

= Coriolis/centrifugal, = gravitational



Q&A