THE UNIVERSITY of EDINBURGH
informatics

Advanced Robotics

Dynamics II

Subramanian Ramamoorthy
School of Informatics
University of Edinburgh

Outline

- We discuss the following three topics today:
  - 1D point mass
  - A ‘general’ dynamic robot (=> Dynamics II)
  - Joint space control

- For now we assume that the robot is fully actuated and that $\mathbf{v_q} = \dot{\mathbf{q}}$

  (ie velocity and configuration space have the same dimension)

- We also assume motors are equipped with accurate **position** sensors (ie we know **q** accurately)

Forward and inverse dynamics

- As for geometry and kinematics, we are interested in two formulations of the dynamics of a system

  - Forward dynamics: Given $\mathbf{q}$, $\mathbf{v}_q$ and τ, compute joint accelerations $\dot{\mathbf{v}}_{\mathbf{q}}$

    Useful for *simulation*

  - Inverse dynamics: Given $\mathbf{q}$, $\mathbf{v}_q$ and $\dot{\mathbf{v}}_{\mathbf{q}}$, compute torque commands τ

    Useful for *control*

Two approaches to dynamics

- Lagrangian dynamics
  - Intuitive variational formulation (principles you may already be familiar with)
  - Equations get messy quite quickly
- Newton-Euler dynamics
  - Not so intuitive, we'll focus mainly on the ideas (detailed derivations in all standard texts)
  - Newton-Euler is practically used in robotics because efficient *recursive* algorithms can be derived from the formulation

Lagrangian dynamics

- In classic mechanics, as you would have seen in secondary school, we have the core concept of mechanical energy

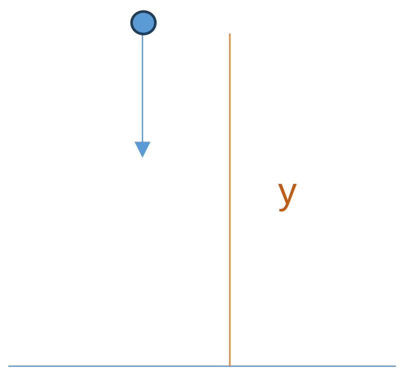
- Example: free fall of a point mass
  - Kinetic Energy T

    $$T = \frac{1}{2} m \dot{y}^2$$

  - Potential Energy U (gravitational)

    $$U = mgy$$

  - If only *conservative* forces are applied, total energy E = T + U is constant



Lagrangian dynamics –intuitions

- ❑ Let's write $$L(y,\dot{y}) = T - U = \frac{1}{2}m\dot{y}^2 - mgy$$

- ❑ We can verify that $$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) = \frac{\partial L}{\partial y}$$

$$\frac{\partial L}{\partial \dot{y}} = m\dot{y}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) = m\ddot{y}$$

$$\frac{\partial L}{\partial y} = -mg$$

- ❑ Which brings us back to the very familiar Newton's law F = ma.

- ❑ L is called the Lagrangian, and concisely captures the physics

Lagrangian mechanics

❑ In the general case, it can be proven that this equation equates to the torques

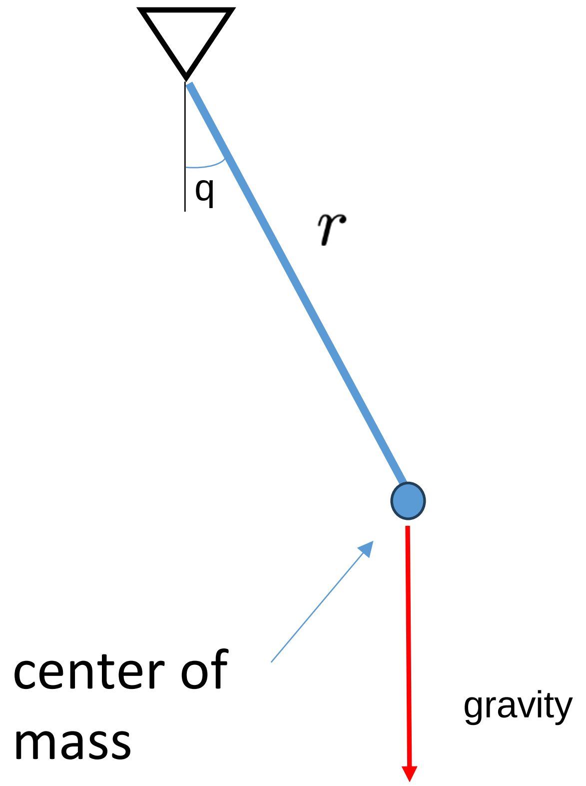
$$\frac{d}{dt}\left(\frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial \dot{y}}\right) - \frac{\partial L(\mathbf{q},\dot{\mathbf{q}})}{\partial y} = \tau$$

❑ We won't prove this (all variational calculus and mechanics books will cover this - if you were interested), just show it on another example

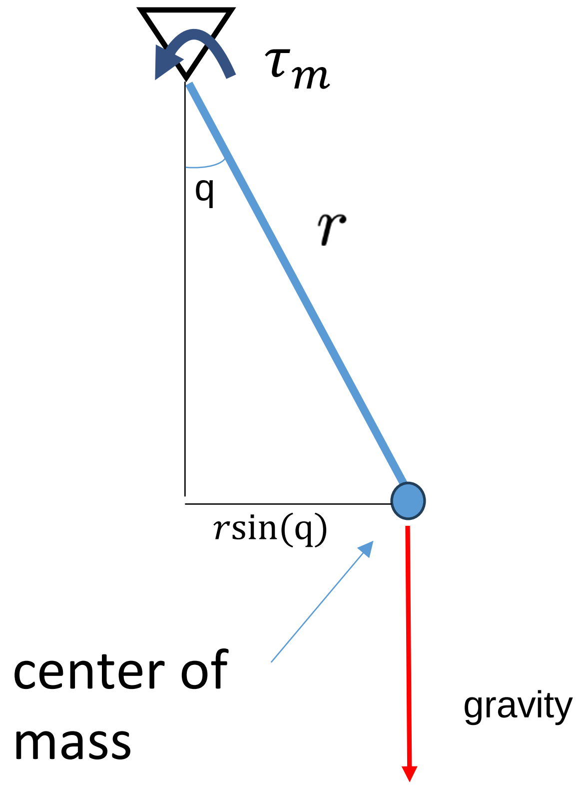
Example: Equation of motion of a 1-DOF robot arm

mass of pendulum: m

moment of inertia (about pivot point): I_p



Example: Equation of motion of a 1-DOF robot arm



mass of pendulum: m

moment of inertia (about pivot point): I_p

Newton's second law :

$$\mathrm{I_p}\ddot{\mathbf{q}} = \sum_i \tau_i$$

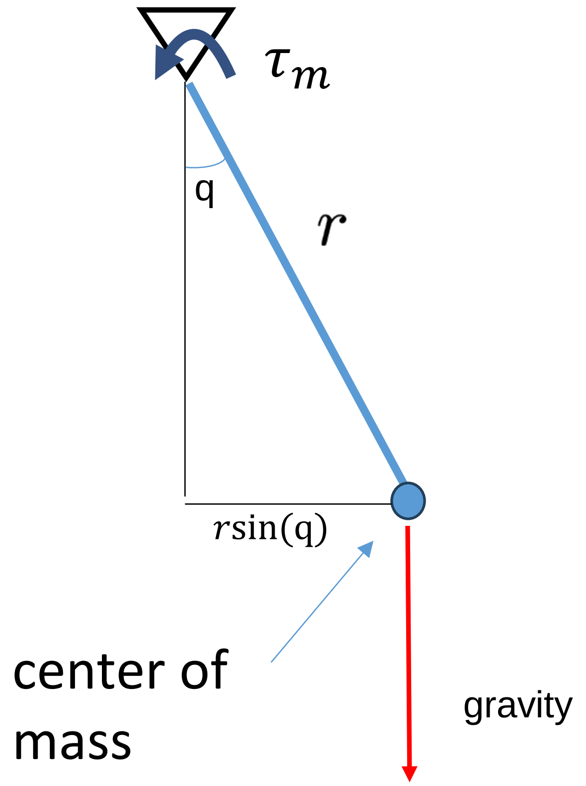
$\tau_g = -rmg\sin(\mathrm{q})$ Gravity torque

τ_m Motor torque

Example: Equation of motion of a 1-DOF robot arm

mass of pendulum: m

moment of inertia (about pivot point): I_p



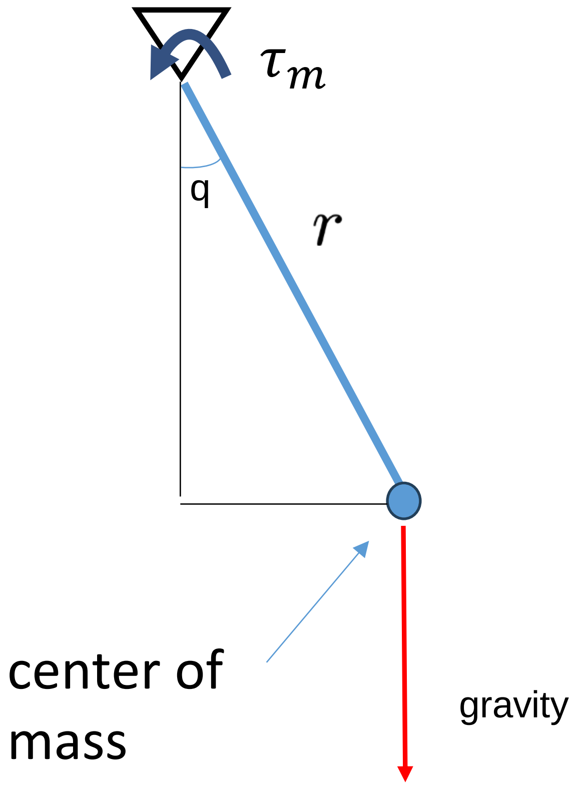
Newton's second law :

$$\mathrm{I_p}\ddot{\mathbf{q}} = \sum_i \tau_i$$

$\tau_g = -rmg\sin(\mathrm{q})$ Gravity torque

τ_m Motor torque

Lagrangian computation



$$T(q) = \frac{1}{2} I_p \dot{q}^2 \quad U(q) = mgh, \quad \text{where} \quad h = r(1 - \cos(q))$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) = I_p \ddot{q} \qquad \frac{\partial L}{\partial q} = \frac{\partial}{\partial q}(mgr(1 - cos(q)) = -mgrsin(q)$$

$$\tau_m = \frac{d}{dt}\left(\frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial \dot{q}}\right) - \frac{\partial L(\mathbf{q}, \dot{\mathbf{q}})}{\partial q}$$

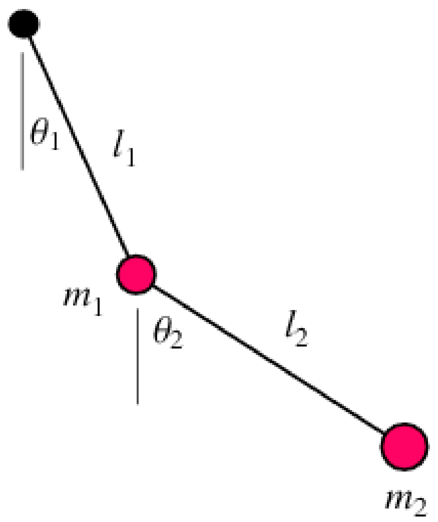
$$\tau_m = \mathrm{I_p} \ddot{\mathbf{q}} + rmg\sin(\mathrm{q})$$

Why use the Lagrangian?

- Newton's law equations consider each rigid body *individually.* The problem is that we will have to also consider constraint forces (e.g. double pendulum equations below – source: wolfram)

- Lagrangian formulations nicely *incorporates* all of these considerations

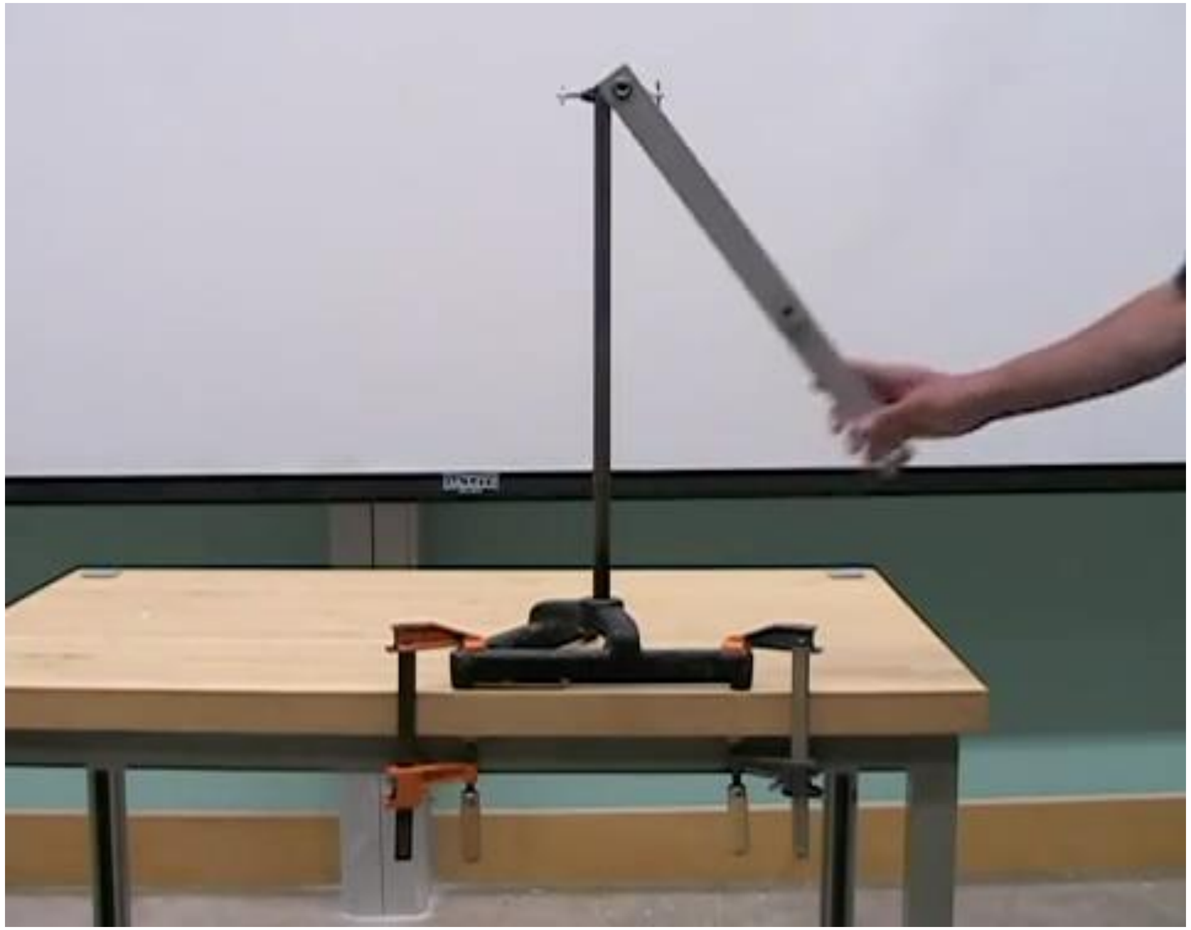
- Things get rapidly complicated when we write out complete equations of motion. The differential equations for the double pendulum system are respectively:



$$(m_1 + m_2) L_1^2 \ddot{\theta}_1 + m_2 L_1 L_2 \cos(\theta_1 - \theta_2) \ddot{\theta}_2 + m_2 L_1 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2^2 + g(m_1 + m_2) L_1 \sin(\theta_1) = 0$$

$$m_2 L_1 L_2 \cos(\theta_1 - \theta_2) \ddot{\theta}_1 + m_2 L_2^2 \ddot{\theta}_2 + -m_2 L_1 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_1^2 + g m_2 L_2 \sin(\theta_2) = 0$$

Indeed the behaviour is … chaotic



Canonical form for articulated rigid bodies

- We can regroup terms as follows

$$\underbrace{\mathbf{M}}_{\text{Inertia Matrix}}(\mathbf{q})\,\ddot{\mathbf{q}} + \underbrace{\mathbf{B}}_{\text{Coriolis Matrix}}(\mathbf{q})\,[\dot{\mathbf{q}}\dot{\mathbf{q}}] + \underbrace{\mathbf{C}}_{\text{Centrifugal Matrix}}(\mathbf{q})\,[\dot{\mathbf{q}}^2] + \underbrace{\mathbf{G}}_{\text{Gravity Vector}}(\mathbf{q}) = \underbrace{\tau}_{\text{External Forces}}$$

$$[\dot{\mathbf{q}}\dot{\mathbf{q}}] = \begin{bmatrix} \dot{q}_1\dot{q}_2 & \dot{q}_1\dot{q}_3 & \dots & \dot{q}_{n-1}\dot{q}_n \end{bmatrix}^T$$

$$[\dot{\mathbf{q}}^2] = \begin{bmatrix} \dot{q}_1^2 & \dot{q}_2^2 & \dots & \dot{q}_n^2 \end{bmatrix}^T$$

M,B,C,G are only configuration dependent

What are Coriolis forces?

Other representations in the literature

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau$$

C is a vector with Coriolis plus centrifugal terms

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \tau$$

C is a matrix with Coriolis plus centrifugal terms

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \tau$$

More about the inertia matrix M(q):

- M defines the Kinetic energy of our robot:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$$

- Because $T \geq 0, \forall(\mathbf{q}, \dot{\mathbf{q}})$, **M** is ?

More about the inertia matrix M(q):

- M defines the Kinetic energy of our robot:

$$T(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$$

- Because $T \geq 0, \forall(\mathbf{q}, \dot{\mathbf{q}})$, **M** is positive-definite

- As such it is invertible:

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau \quad \Longrightarrow \quad \ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\left(-\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) + \tau\right)$$

This gives us the equations for forward and inverse dynamics

$$\boxed{\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau}$$

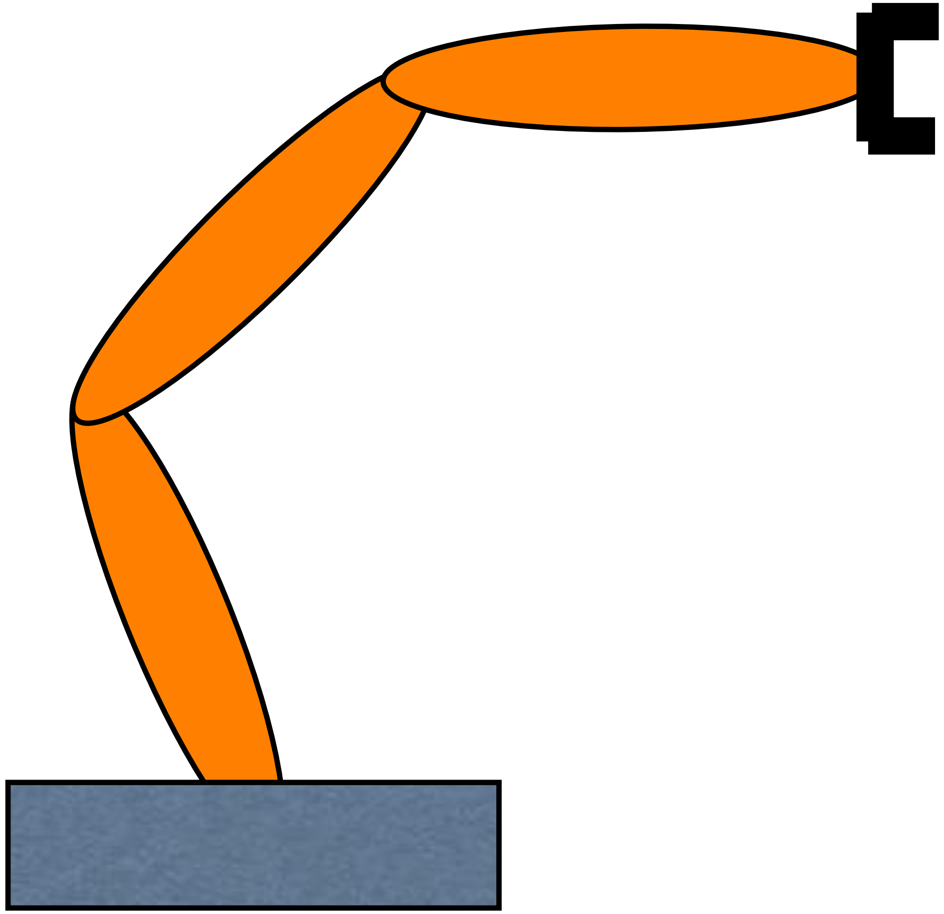
Inverse Dynamics equation = control

$$\boxed{\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\left(-\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) + \tau\right)}$$

Forward Dynamics equation = simulation

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

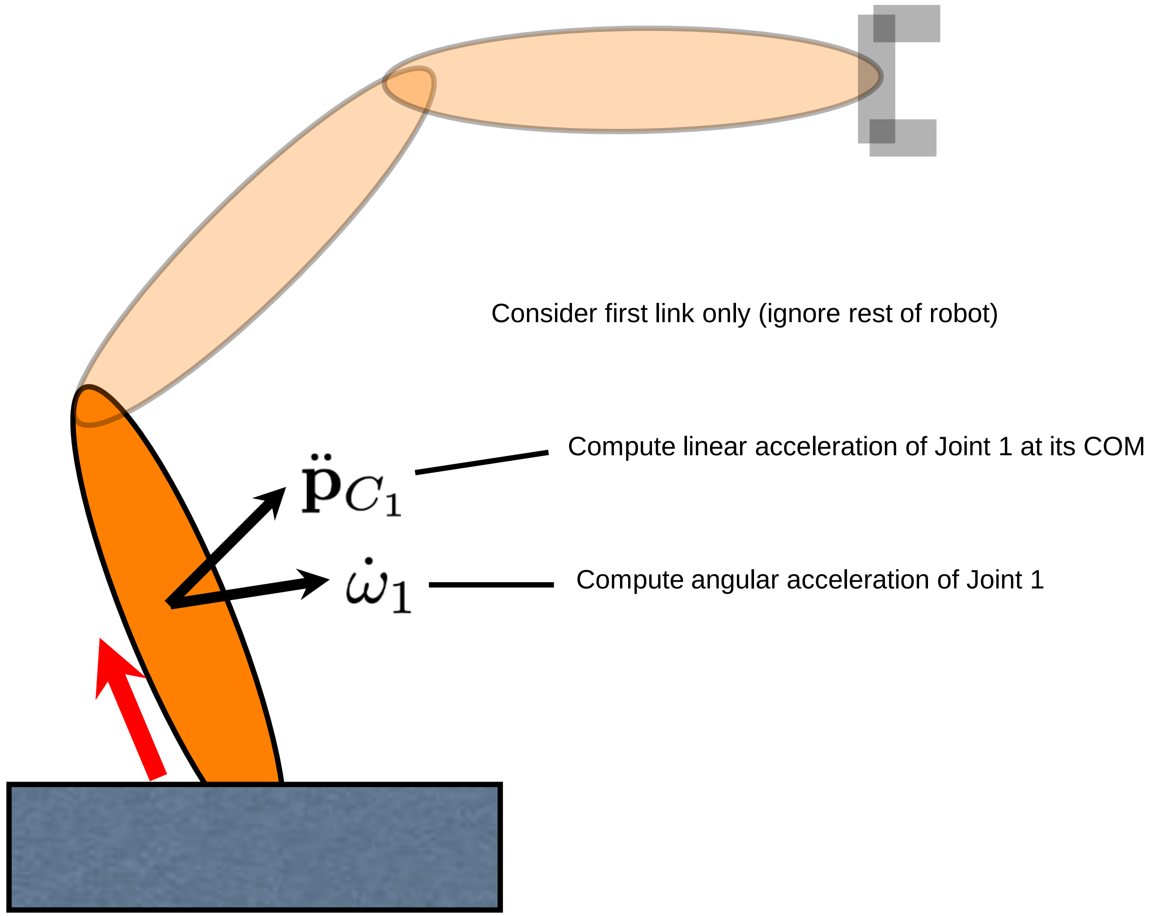


First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

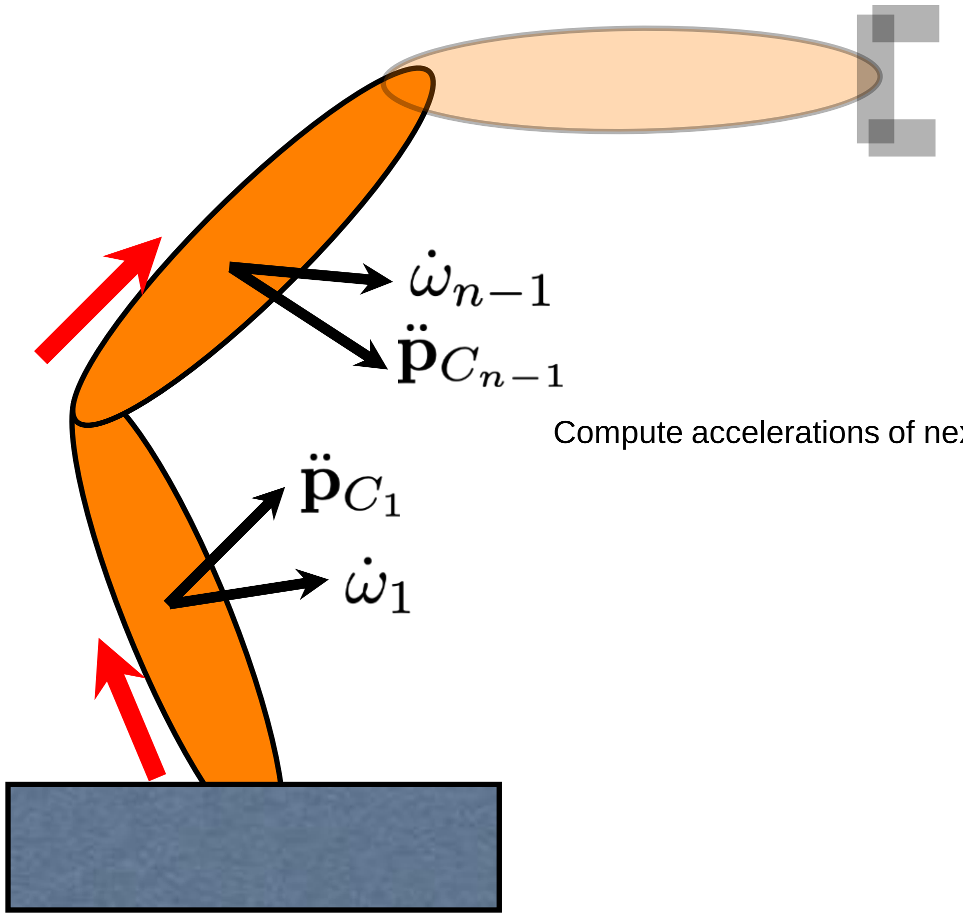
$\mathbf{p}_{C1}$ denotes the coordinates of the first link C_1, *more on this next week*



First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

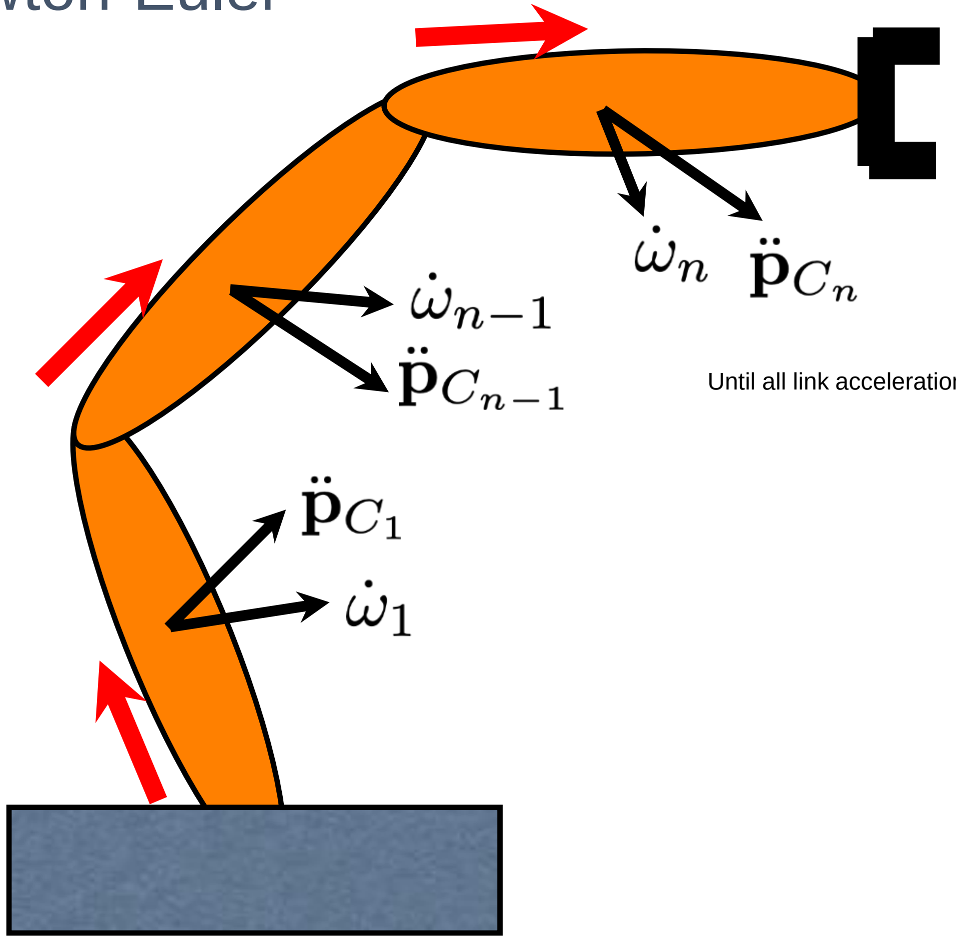


Compute accelerations of next link in chain

First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

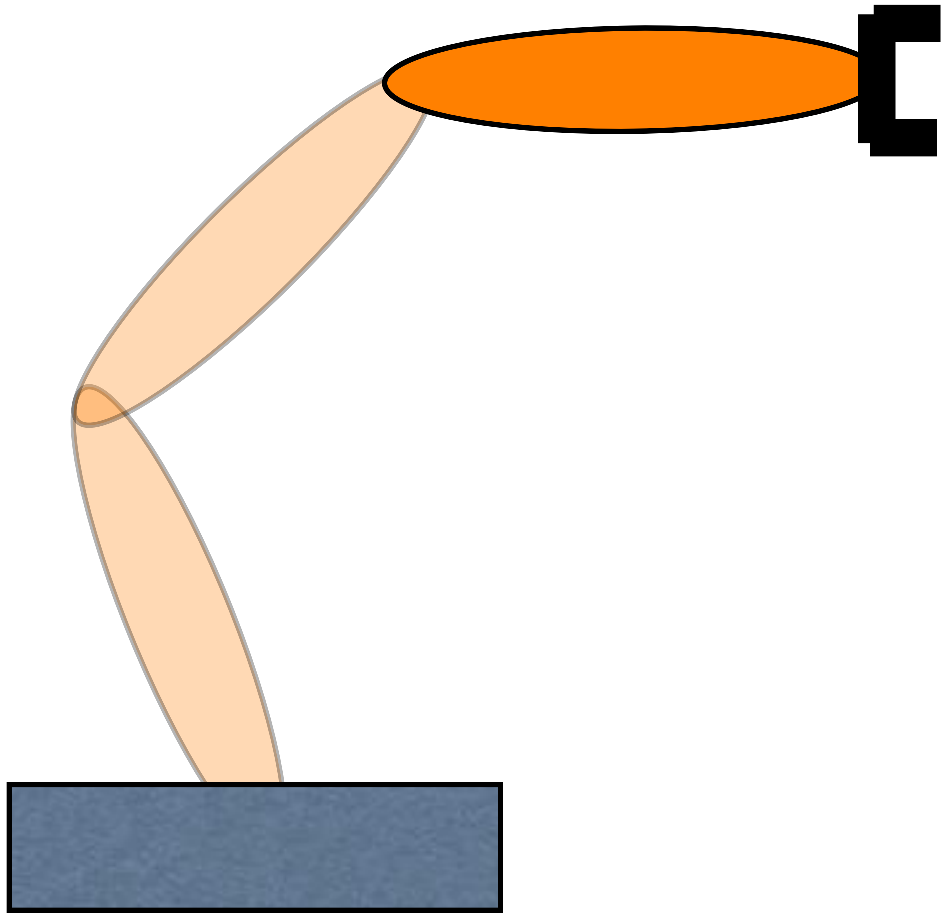


Until all link accelerations are known

First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

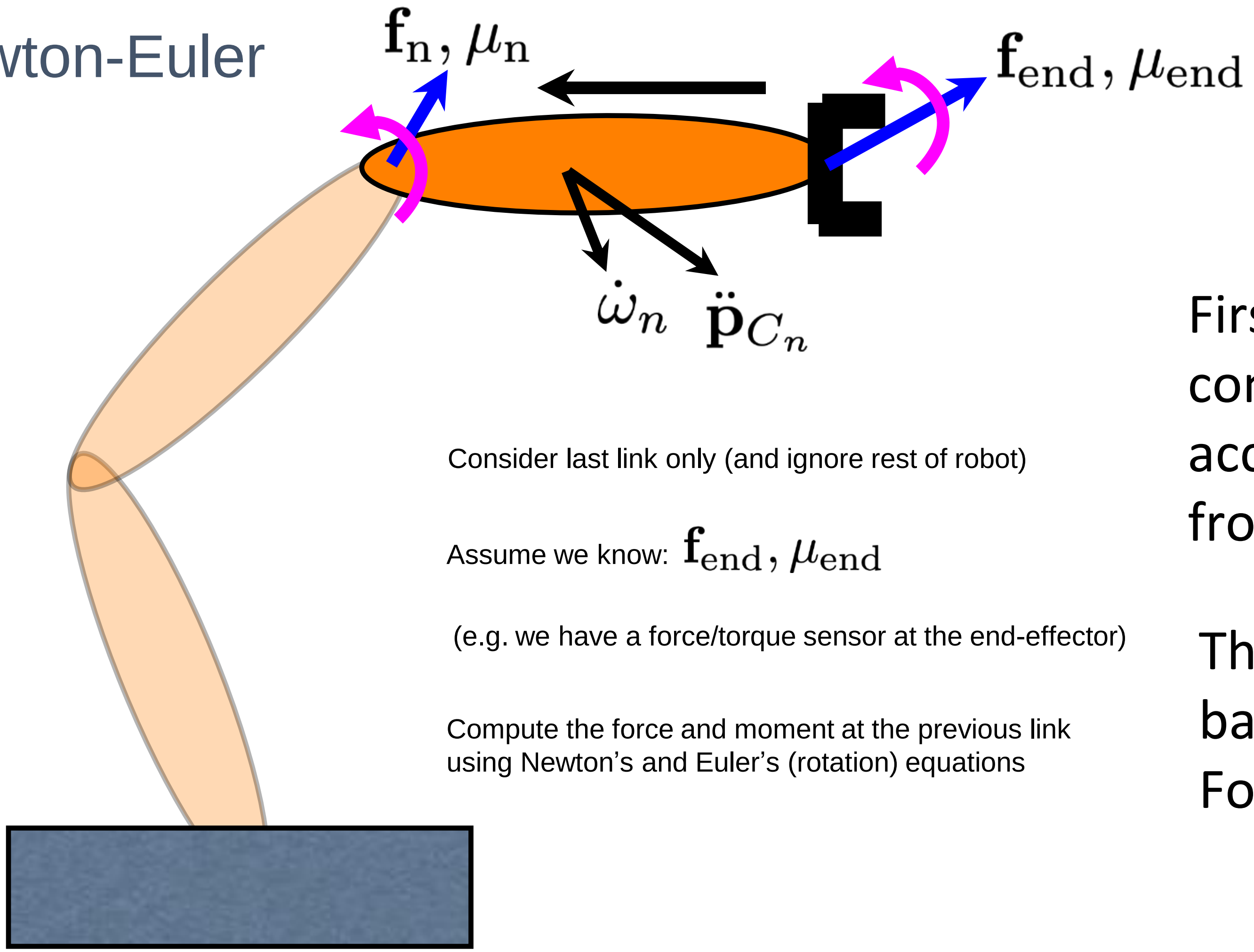


First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

Then, proceed backwards to compute Forces and moments

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler



Consider last link only (and ignore rest of robot)

Assume we know: $\mathbf{f}_{\mathrm{end}}, \mu_{\mathrm{end}}$

(e.g. we have a force/torque sensor at the end-effector)

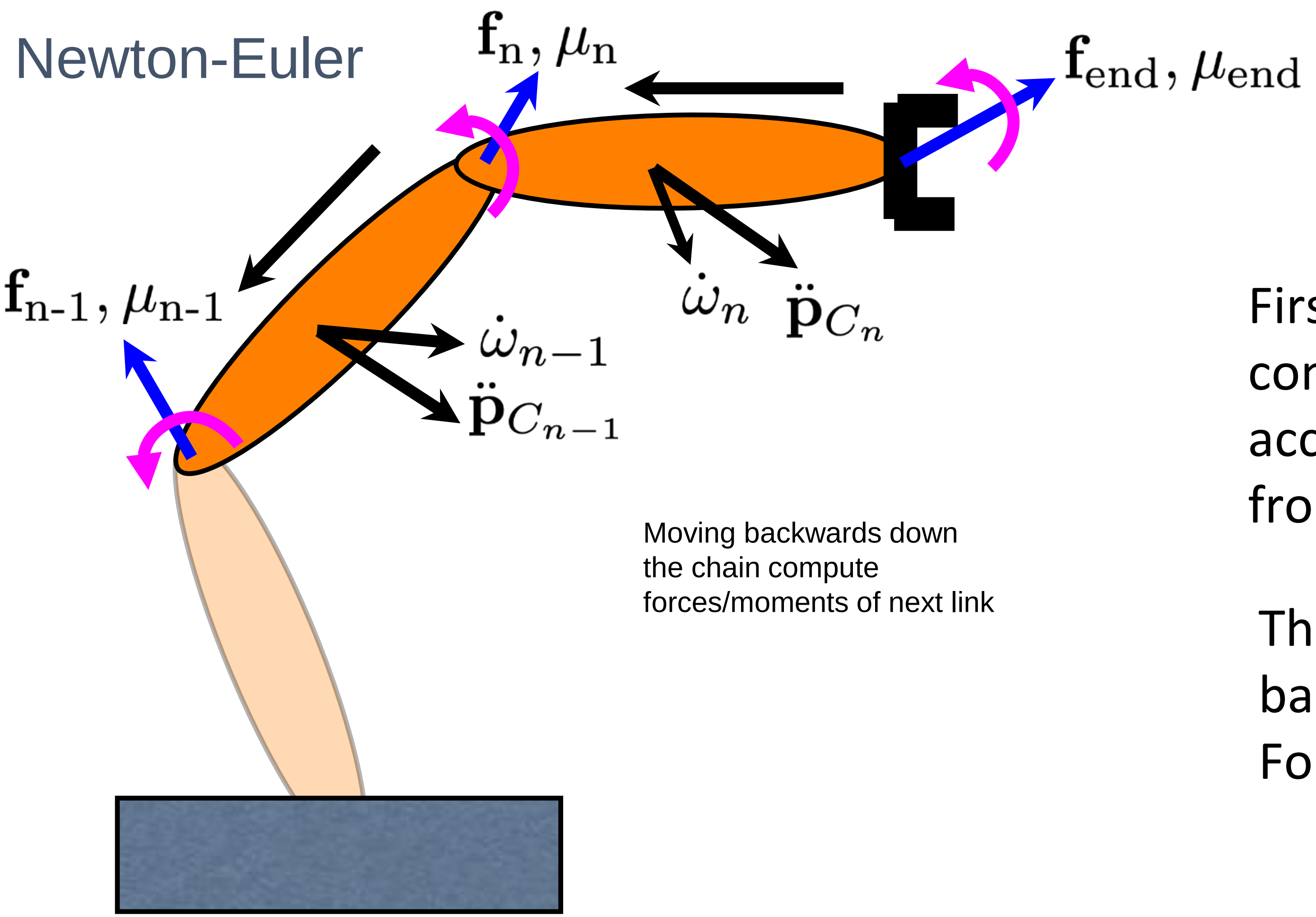
Compute the force and moment at the previous link using Newton's and Euler's (rotation) equations

First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

Then, proceed backwards to compute Forces and moments

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler

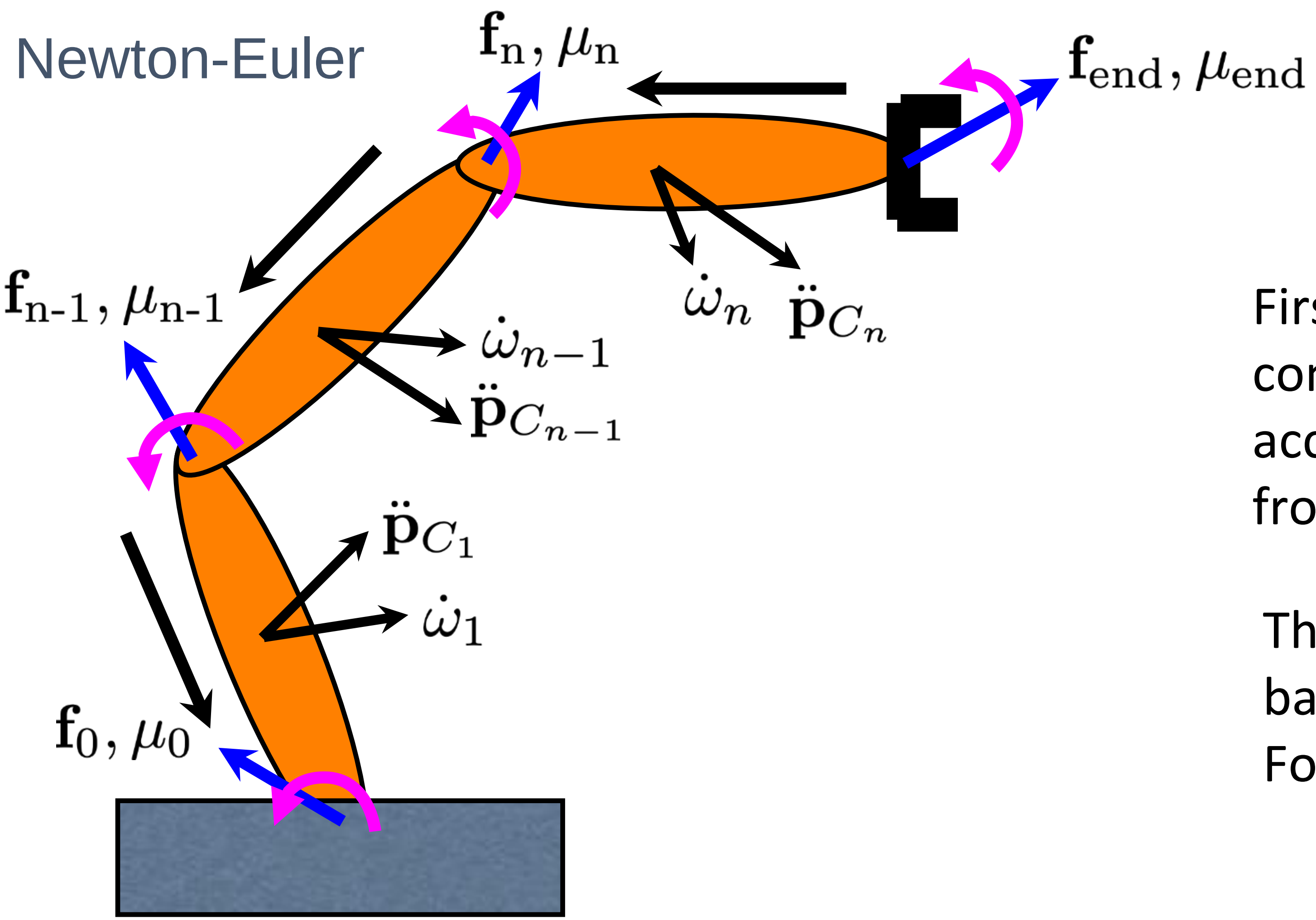


First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

Then, proceed backwards to compute Forces and moments

A recursive algorithm for articulated robots: intuitions

- Based on Newton-Euler



First given $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ compute all link accelerations, starting from the base

Then, proceed backwards to compute Forces and moments

A recursive algorithm for articulated robots

- The Recursive Newton-Euler Algorithm (RNEA) gives us an iterative way to compute Inverse Dynamics:

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \tau$$

$$RNEA(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \tau$$

- Without requiring to work out the mass matrix. However we still need M for forward dynamics

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1}\left(-\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{G}(\mathbf{q}) + \tau\right)$$

Can RNEA be used to compute the mass matrix?

- A trick is to compute M column by column by setting:

$$\mathbf{g} = 0 \quad \dot{\mathbf{q}} = 0 \quad \ddot{q}_i = 1 \quad \ddot{q}_j = 0, \forall j \neq i$$

- In such a case RNEA return the ith column of M

- As a result, computing inverse dynamics is faster than forward dynamics:

  - $O(n^2)$ for computing *direct dynamics*,
  - $O(n)$ for computing *inverse dynamics*.

How do we control this ? (transition slide)

General Robot System Dynamics

- **State** $x = (q, \dot{q}) \in \mathbb{R}^{2n}$
  – joint positions $q \in \mathbb{R}^n$
  – joint velocities $\dot{q} \in \mathbb{R}^n$
- **Controls** $u \in \mathbb{R}^n$ are the *torques* generated in each motor.
- The **system dynamics** are:

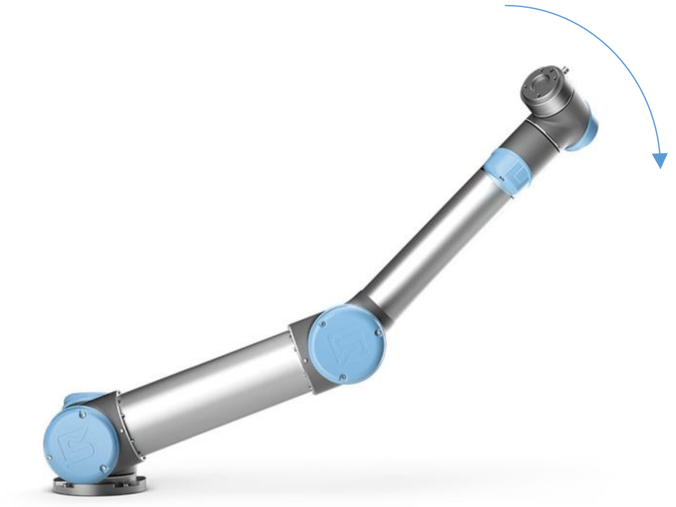
$$M(q)\,\ddot{q} + C(q,\dot{q})\,\dot{q} + G(q) = u$$

$M(q) \in \mathbb{R}^{n \times n}$	is positive definite intertia matrix (can be inverted $\rightarrow$ forward simulation of dynamics)
$C(q,\dot{q}) \in \mathbb{R}^n$	are the centripetal and coriolis forces
$G(q) \in \mathbb{R}^n$	are the gravitational forces
u	are the joint torques

Computing M(q) and F(q, dq)

- More compact form as:

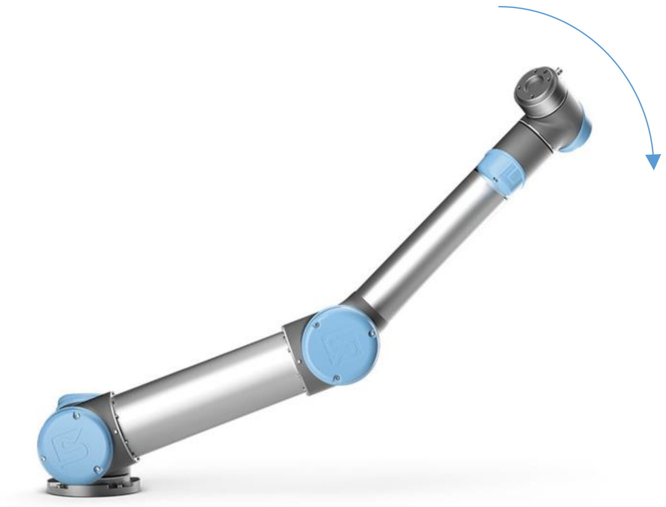
$$M(q)\,\ddot{q} + F(q,\dot{q}) = u$$



There exist efficient algorithms to compute M and F.
This is implemented in pinocchio

Implications for (multi-body) dynamics

- If we know the desired $\ddot{q}^*$ for each joint, the eqn. $M(q)\,\ddot{q}^* + F(q,\dot{q}) = u^*$ gives the desired torques.



Joint Space control

- Where could we get the desired $\ddot{q}^*$ from?
  Assume we have a nice smooth **reference trajectory** $q^{\text{ref}}_{0:T}$ (generated with some motion profile or alike), we can at each t read off the desired acceleration as

Open loop

$$\ddot{q}^{\text{ref}}_t := \frac{1}{\tau}[(q_{t+1} - q_t)/\tau - (q_t - q_{t\text{-}1})/\tau] = (q_{t\text{-}1} + q_{t+1} - 2q_t)/\tau^2$$

What if we directly use desired reference acceleration?

Tiny errors in acceleration will **accumulate** greatly over time and this makes this an **unstable** approach!

Joint Space control

Choose a desired acceleration $\ddot{q}_t^*$ that implies a *PD-like behavior around the reference trajectory*!

$$\ddot{q}_t^* = \ddot{q}_t^{\text{ref}} + K_p(q_t^{\text{ref}} - q_t) + K_d(\dot{q}_t^{\text{ref}} - \dot{q}_t)$$

$$M(q)\,\ddot{q}^* + F(q,\dot{q}) = u^*$$

This is a standard and convenient way of tracking a reference trajectory when the **robot dynamics are known**: *all the joints will behave exactly like a 1D point mass around the reference trajectory!*

Summary

- We discuss the following three topics today:
  - 1D point mass
  - A ‘general’ dynamic robot (→ Dynamics II)
  - Joint space control method $\ddot{q}_t^* = \ddot{q}_t^{\text{ref}} + K_p(q_t^{\text{ref}} - q_t) + K_d(\dot{q}_t^{\text{ref}} - \dot{q}_t)$

$$\Downarrow$$

$$M(q)\,\ddot{q}^* + F(q, \dot{q}) = u^*$$