Data-Driven Computer Animation
Lecture 4.

Forward/Inverse Dynamics

Overview

- Motivation
- Background / Notation
- Articulate Dynamics Algorithms
  - Newton-Euler Algorithm
  - Recursive Newton-Euler
  - Articulated-Body Algorithm (Featherstone)

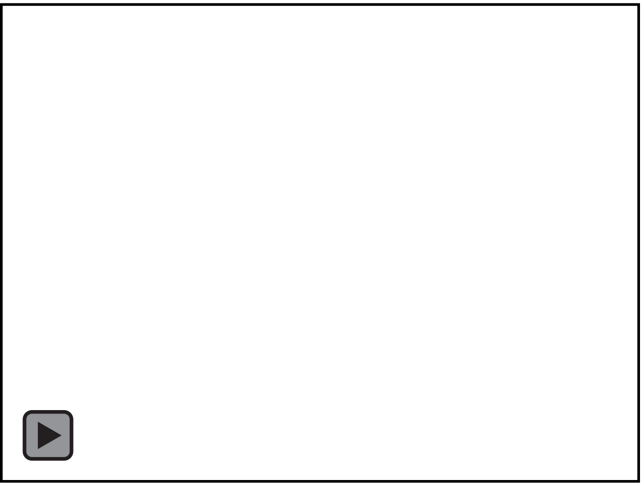
Character Animation

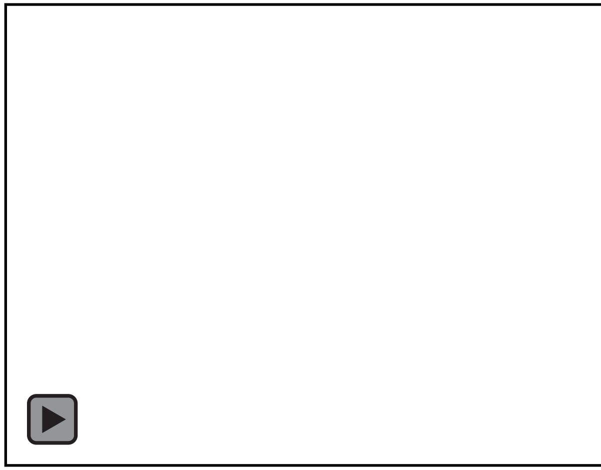
- There are three methods
  - Create them manually
  - Use real human / animal motions
  - Use physically based simulation



Physically-based Animation

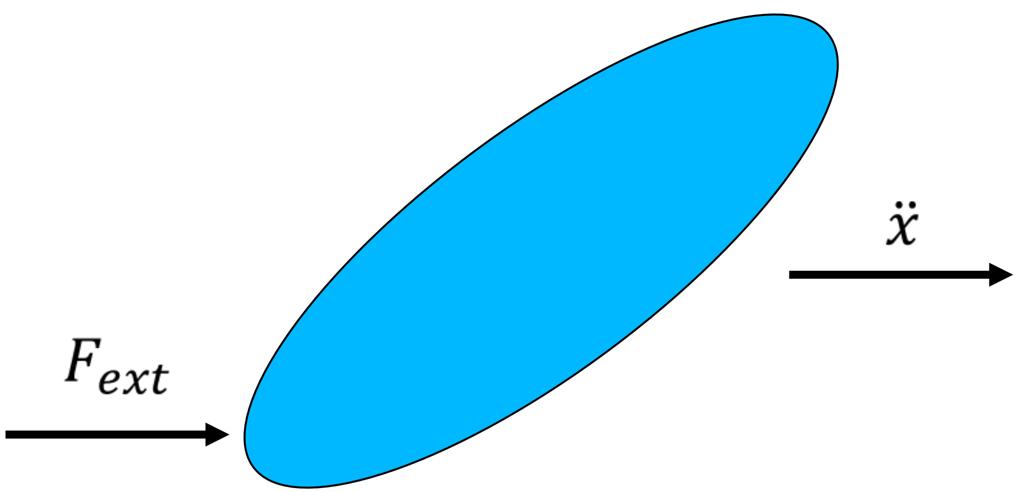
- Forward/inverse dynamics
- Voluntary motion, passive motion
- PD control for voluntary motion, response motion





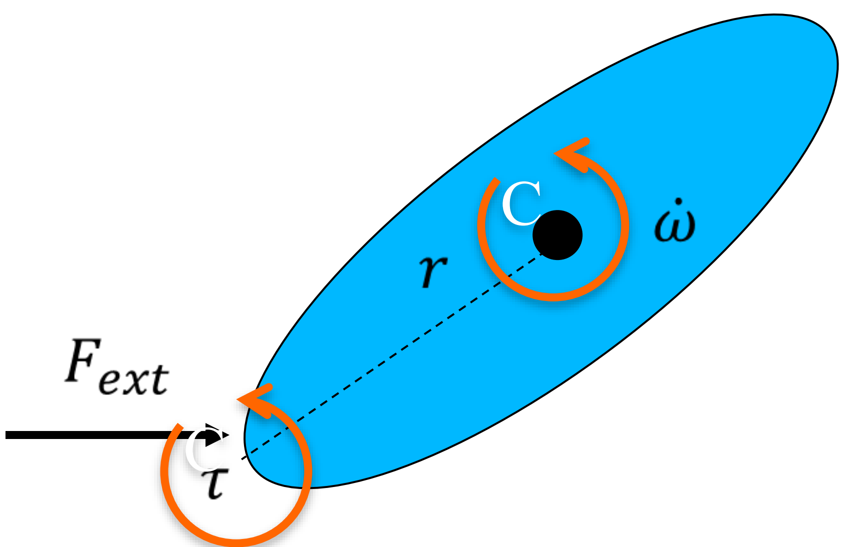
How to do the physics computation?

- For every rigid body, its motion follows Newton's law
- $F_{ext} = M\ddot{x}$ (linear motion)
- $r \times F_{ext} + \tau = I\dot{\omega} + \omega \times I\omega$(angular motion)



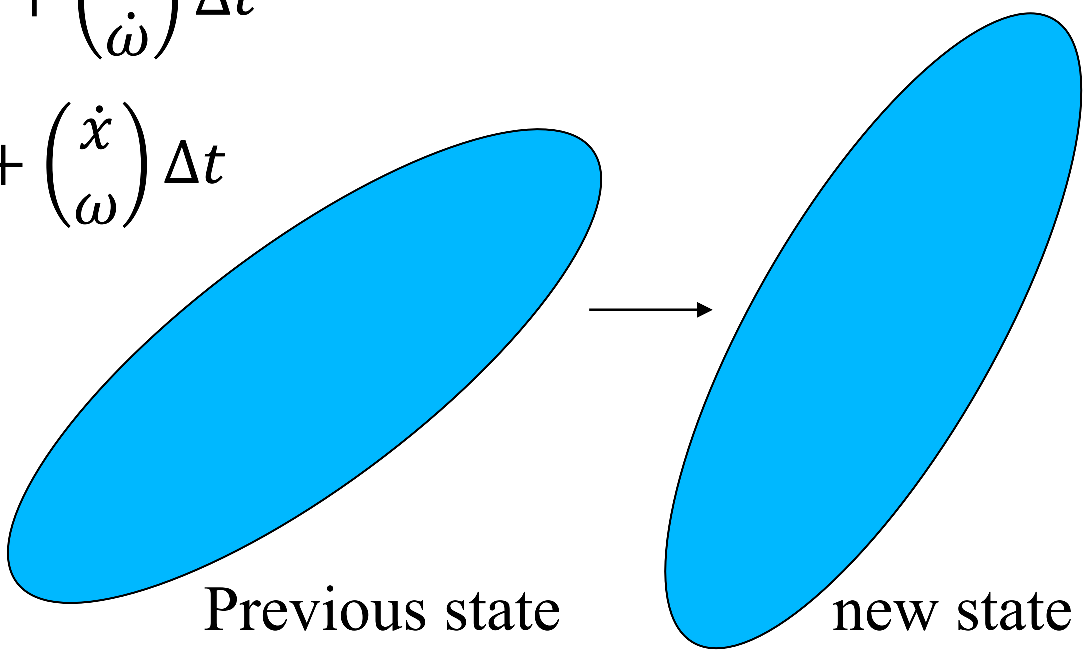
How to do the physics computation?

- For every rigid body, its motion follows Newton's law
- $F_{ext} = M\ddot{x}$ (linear motion)
- $r \times F_{ext} + \tau = I\dot{\omega} + \omega \times I\omega$(angular motion)



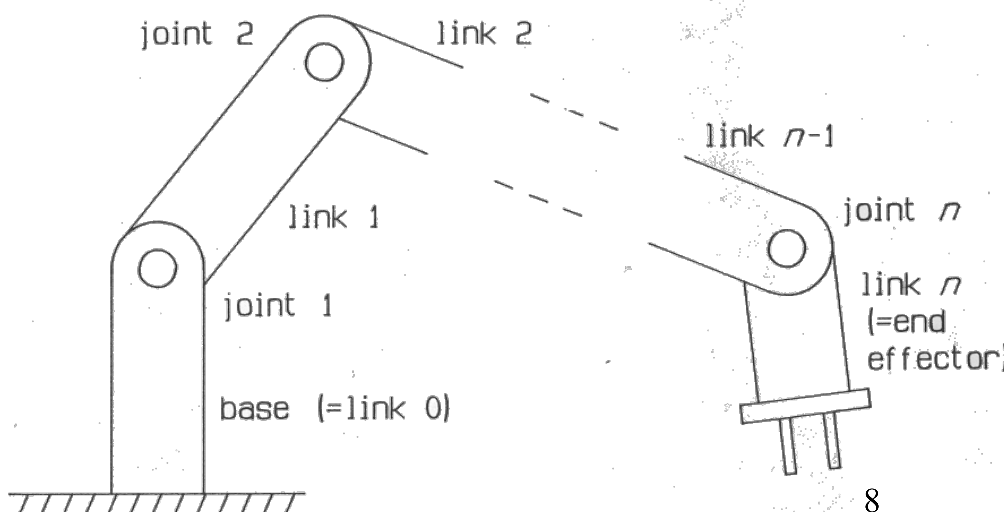
Physical Simulation

- Using the computed $\ddot{x}, \dot{\omega}$, update the position and orientation by integration
  - $\begin{pmatrix} \dot{x} \\ \omega \end{pmatrix} = \begin{pmatrix} \dot{x}_{prev} \\ \omega_{prev} \end{pmatrix} + \begin{pmatrix} \ddot{x} \\ \dot{\omega} \end{pmatrix} \Delta t$
  - $\begin{pmatrix} x \\ \theta \end{pmatrix} = \begin{pmatrix} x_{prev} \\ \theta_{prev} \end{pmatrix} + \begin{pmatrix} \dot{x} \\ \omega \end{pmatrix} \Delta t$



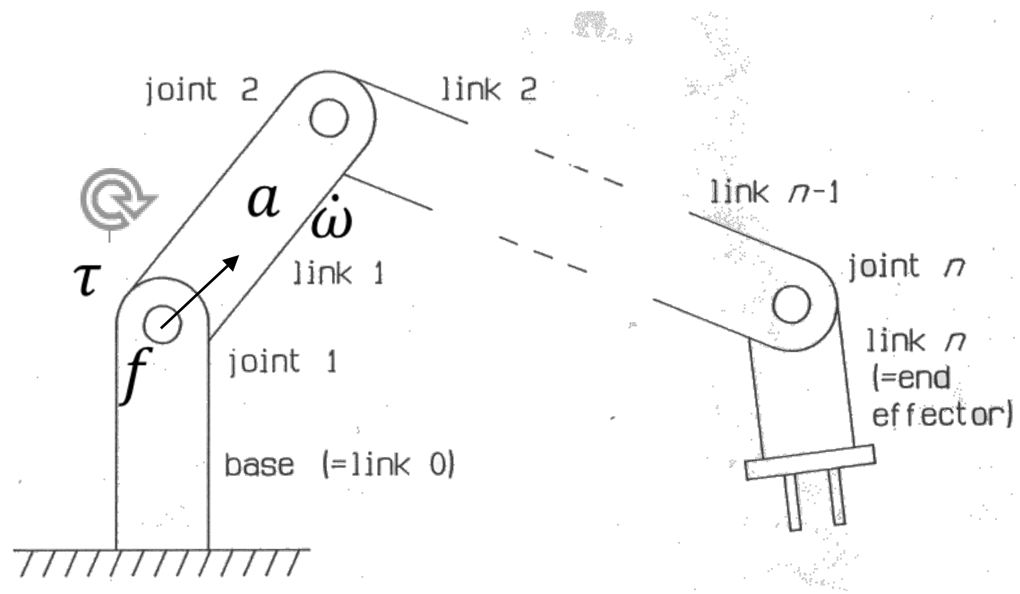
Articulated Body

- An articulated body is a group of rigid bodies (called links) connected by joints
- Multiple types of joints
  - Hinge/Revolute (1 degree of freedom)
  - Ball joint (3 degrees of freedom)
  - Prismatic, screw, etc.



Articulated Bodys

- For articulated bodies, there will be
  - Force between the joints
  - Torque generated by the motors at the joints
  - These forces and torques will result in a motion of the entire model

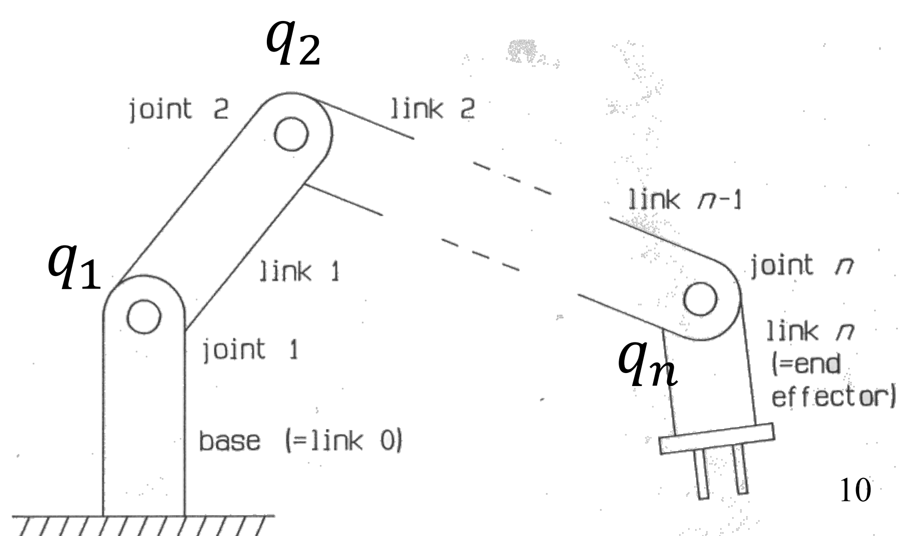


Notation : generalized coordinates

- Transitioning this to articulated bodies

$$\mathbf{q} = (q_1, q_2, \dots, q_n)$$

  - Generalized coordinates
  - All degrees of freedom put into a single vector

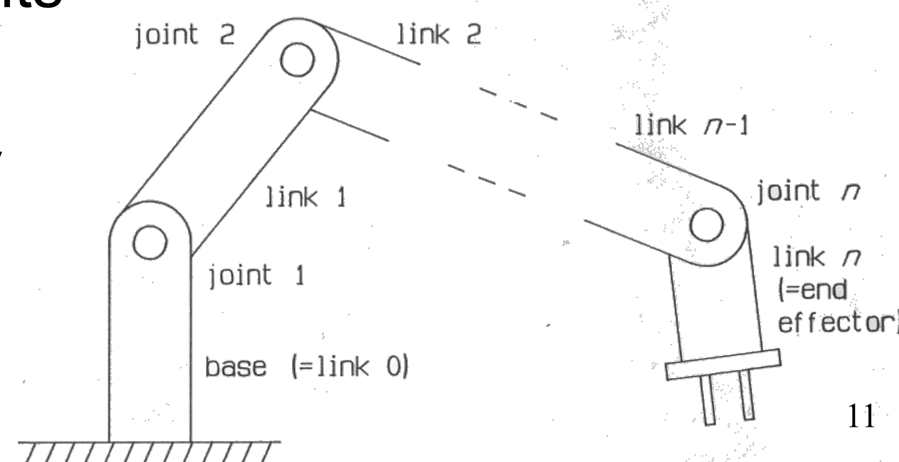


Notation

- Transitioning this to articulated bodies

$$\boldsymbol{\tau} = \mathbf{H}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$$

  - $\boldsymbol{\tau}$ is the force on link i
  - **H** is the joint-space inertia matrix (n x n)
  - $\mathbf{q}, \dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are the coordinates, velocities and accelerations of the joints
  - **C** are Coriolis force
  - $\boldsymbol{\tau}_g$: force due to gravity



Forward vs. Inverse Dynamics

- Inverse Dynamics
  - The calculation of forces given a set of accelerations
- Forward Dynamics
  - The calculation of accelerations given a set of forces

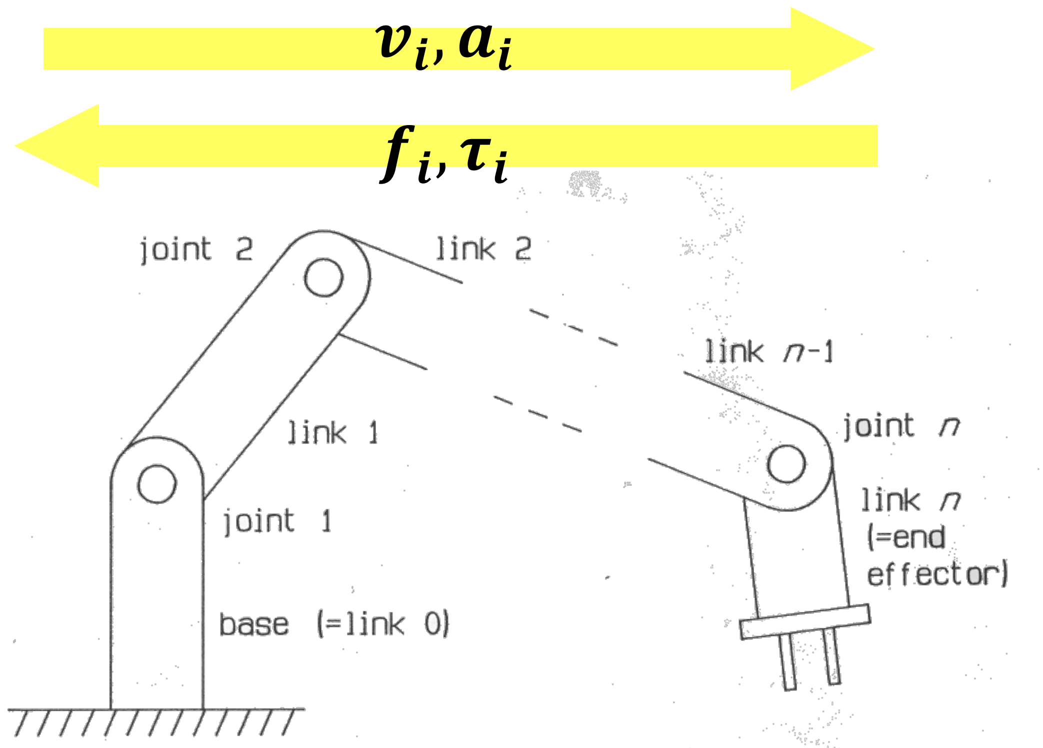
$$\boldsymbol{\tau} = \mathbf{H}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$$

Algorithms

- Inverse Dynamics
  - Newton-Euler Algorithm
- Forward Dynamics
  - Recursive Newton-Euler
  - Articulated-Body Algorithm

Inverse Dynamics: Newton-Euler Algorithm

- Goal
  - Given the accelerations and velocities at the joints, find the forces/torques required at the joints to generate those accelerations
- Recursive approach



- Compute $\boldsymbol{v}_i, \boldsymbol{a}_i$ from root to leaf using $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$
- Compute the force $\boldsymbol{f}_i, \boldsymbol{\tau}_i$ from leaves to root using the equation of motion

Spatial Notation

- Spatial notation combines linear and angular quantities – extending the 3x3 notion to 6x6
- $\mathbf{v} = \begin{bmatrix} \boldsymbol{\omega} \\ \dot{\mathbf{x}} \end{bmatrix}$ $\boldsymbol{\omega}$ angular velocity, $\dot{\mathbf{x}}$ linear velocity
- $\mathbf{a} = \begin{bmatrix} \dot{\boldsymbol{\omega}} \\ \ddot{\mathbf{x}} \end{bmatrix}$ $\dot{\boldsymbol{\omega}}$ angular acceleration, $\ddot{\mathbf{x}}$ linear acceleration
- $\mathbf{f} = \begin{bmatrix} \boldsymbol{n} \\ \boldsymbol{f} \end{bmatrix}$ $\boldsymbol{n}$ momentum, $\boldsymbol{f}$ force
- $\mathbf{I} = \begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & m\mathbf{1} \end{bmatrix}$ $\boldsymbol{I}$ moment of inertia around center of mass, m is mass

See Roy Featherstone’s materials for the details

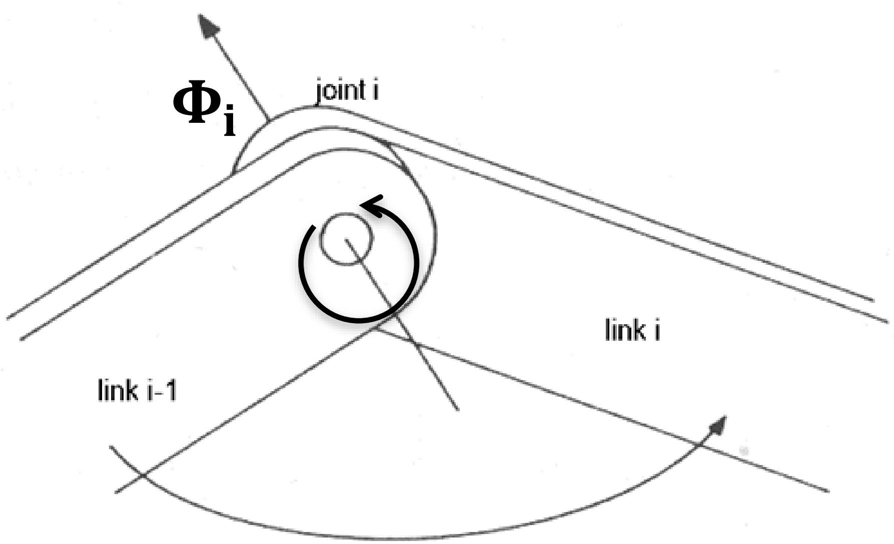
Newton-Euler Algorithm

1. Find the velocities and accelerations of the links

$$\mathbf{v_i} = \mathbf{v_{i-1}} + \mathbf{\Phi_i \dot{q}_i} \quad (\mathbf{v_0} = \mathbf{0})$$

$$\mathbf{a_i} = \mathbf{a_{i-1}} + \mathbf{\Phi_i \ddot{q}_i} + \mathbf{\dot{\Phi}_i \dot{q}_i} \; (\mathbf{a_0} = \mathbf{0})$$

where $\mathbf{\Phi_i}$ is the axis of rotation of joint i,
q, $\dot{q}$ and $\ddot{q}$ are the joint angle, rotation speed and acceleration
and
$\boldsymbol{v_i}$ and $\boldsymbol{a_i}$ are the velocity and acceleration of the link in the world

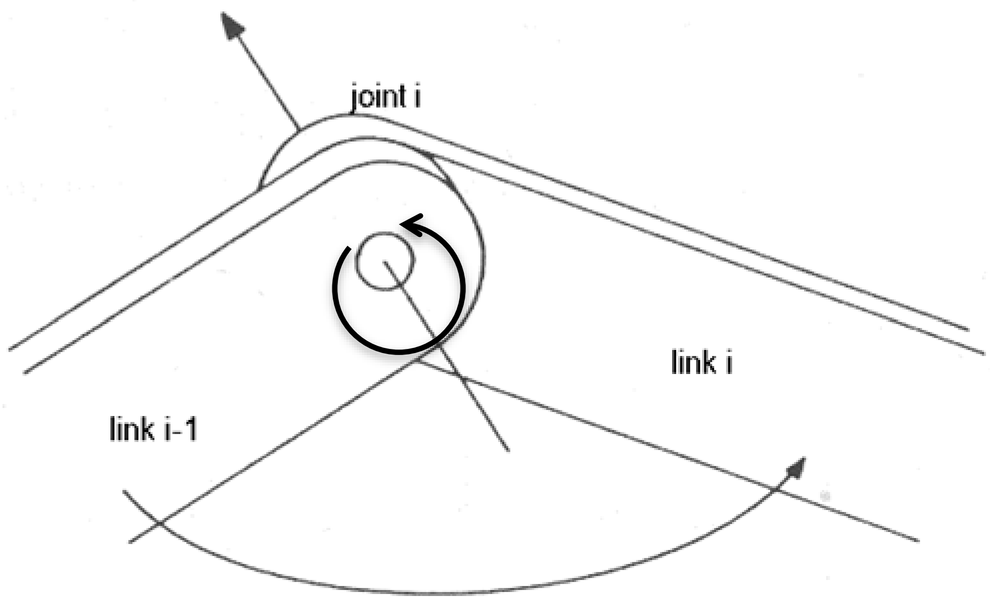


Newton-Euler Algorithm

2. Calculate the equation of motion for each link

$$\mathbf{f}_{\mathrm{i}}^{\mathrm{a}} = \mathbf{I}_{\mathbf{i}}\mathbf{a}_{\mathbf{i}} + \mathbf{v}_{\mathbf{i}} \times \mathbf{I}_{\mathbf{i}}\mathbf{v}_{\mathbf{i}}$$

Computing $\mathbf{f}_{\mathrm{i}}^{\mathrm{a}}$, the total amount of force needed to do the motion

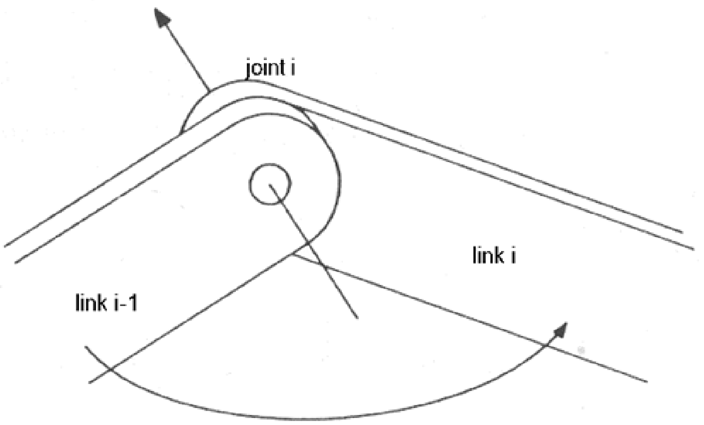


Newton-Euler Algorithm

3. Calculate the joint forces (from leaves towards the root)

$$\mathbf{f_i}^{a} = \mathbf{f_i} - \mathbf{f}_{i+1} + \mathbf{f_i}^{c}$$

$$\mathbf{f_i} = \mathbf{f_i}^{a} - \mathbf{f_i}^{c} + \mathbf{f}_{i+1}$$

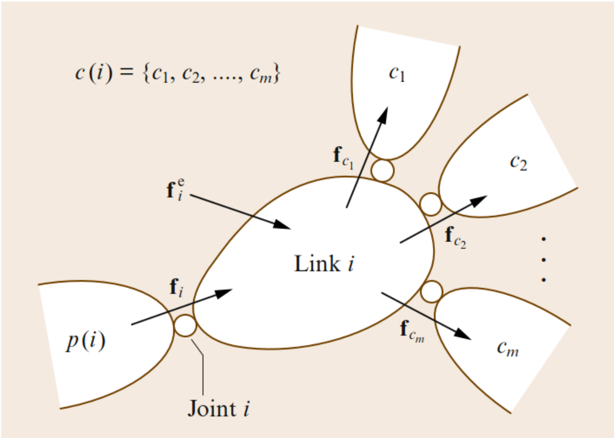


- $\mathbf{f_i}$: force at joint i
- $\mathbf{f_i}^{c}$: external forces

Newton-Euler Algorithm

3. Calculate the joint forces (from leaves towards the root)

$$\mathbf{f_i} = \mathbf{f_i}^a - \mathbf{f_i}^c + \sum_{j \in c(i)} \mathbf{f}_j$$

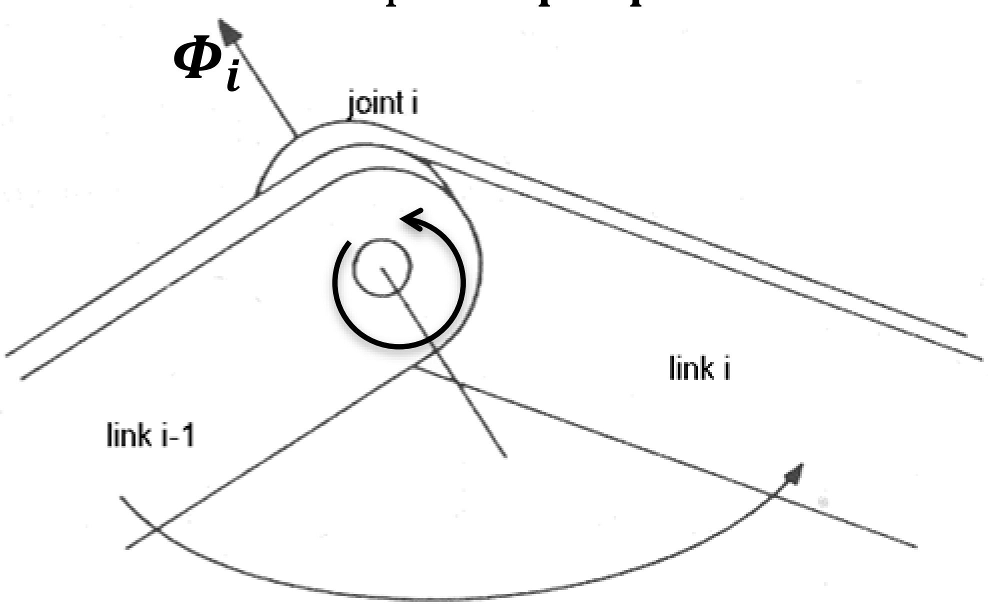


- $\mathbf{f_i}$: force at joint i
- $c(i)$: the child joints of link i
- $\mathbf{f_i}^c$: external forces like gravity

Newton-Euler Algorithm

- Calculate the active forces

$$\tau_{\mathrm{i}} = \mathbf{\Phi_i}^{\mathbf{T}}\, \mathbf{f_i}$$



Inverse Dynamics: Summary

- Inverse Dynamics
  - The calculation of forces given a set of accelerations
  - Newton-Euler : O(N)

- Useful for motion analysis, motion synthesis by optimization (covered later)

Forward Dynamics

- The calculation of accelerations given a set of forces
- Needed for simulating the motion in the virtual environment
- Two approaches described here
  - Recursive Newton-Euler Algorithm
    - A method based on inverse dynamics
    - Easy to understand but slow: $O(N^3)$
  - Articulated Body Inertia
    - Faster to compute: $O(N)$
    - Standard approach in robotics

Recursive Newton-Euler Forward Dynamics

$$\boldsymbol{\tau} = \mathbf{H}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$$

- $\boldsymbol{\tau}$ is the vector of the forces on the links
- $\mathbf{H}$ is the joint-space inertia matrix (n x n)
- $\mathbf{C}$ is the Coriolis force
- $\mathbf{q}, \dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are the coordinates, velocities and accelerations

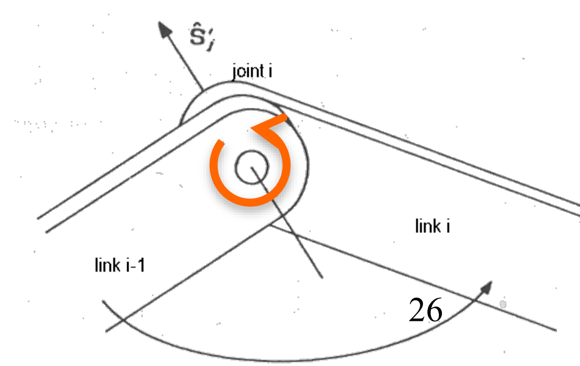
- To compute $\ddot{\mathbf{q}}$ from $\boldsymbol{\tau}$, we need to know $\mathbf{H}$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, $\boldsymbol{\tau}_g$
- Then we can compute $\ddot{\mathbf{q}}$ by solving

$$\mathbf{H}\ddot{\mathbf{q}} = \left(\boldsymbol{\tau} - \boldsymbol{\tau}_g - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\right)$$

Recursive Newton-Euler Forward Dynamics

$$\boldsymbol{\tau} = \mathbf{H}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$$

- Algorithm (based on Newton-Euler ID)
  - Calculate $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$
  - Calculate **H**
  - Solve for $\ddot{\mathbf{q}}$



Recursive Newton-Euler Forward Dynamics

- Compute $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$
  - Setting the acceleration to zero, we get $\boldsymbol{\tau} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$
  - We can use the inverse dynamics solver (Newton-Euler) to solve for the forces given the position, velocity and an acceleration of zero

Recursive Newton-Euler Forward Dynamics

- How to construct **H**? $\boldsymbol{\tau} = \mathbf{H}\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$
- Again, we use the inverse dynamics solver
- This time we can set $\ddot{\mathbf{q}}$ to (0,..,$\underset{\substack{\uparrow \\ i}}{1}$,…,0) to compute the i-th column of **H**
- By subtracting $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g$ from the computed $\boldsymbol{\tau}$, we get the i-th column of **H**

Recursive Newton-Euler Forward Dynamics

- How to construct **H**? $\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}}$
  - Denote $\mathbf{H} = [\mathbf{H_1 H_2} \cdots \mathbf{H_n}]$, $\mathbf{H_i}$: i-th column of **H**
  - Then $\mathbf{H_i} = ID(\mathbf{e_i}, \mathbf{q}, \dot{\mathbf{q}}) - \left(\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}_g\right)$ (ID :inverse dynamics)
    where $\mathbf{e_i}$ is the i-th basis vector (all zero except i-th element 1)

- <u>Algorithm to construct equations of motion</u>
  - **c=** $ID(\mathbf{0}, \mathbf{q}, \dot{\mathbf{q}})$
  - for i=1 to n
    - M_i=$ID(\mathbf{e_i}, \mathbf{q}, \dot{\mathbf{q}})$-**c**
  - **H=[$\mathbf{H_1}$, $\mathbf{H_2}$ … $\mathbf{H_n}$]**

ID called (N+1) times → O(N^2)

Solving for $\mathbf{H}\ddot{\mathbf{q}} = \mathbf{b}$ O(N^3)

Articulated-Body Algorithm

- Until now, we've seen a forward dynamics procedure

  Step 1: Constructing equations of motion → $O(N^2)$

  Step 2: Computing joint acceleration → $O(N^3)$
- In fact, there's a much faster algorithm
- Articulated-Body Algorithm
  - Developed by Featherstone in 1983
  - It only takes $O(N)$ time for open-loop system!
  - It is a recursive algorithm like Recursive Newton-Euler inverse dynamics
  - It doesn't construct equations of motion explicitly.
  - If interested, read Ch. 2 of *Handbook of Robotics*, Springer
  - Next slides show only the idea of the algorithm

Summary of Forward Dynamics

- Let's define a function FD (forward dynamics)

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

We looked at two approaches to realize $\ddot{q} = FD(q, \dot{q}, \tau)$

1. Construct equations of motion and solve it
    - Typically $O(N^3)$ algorithms
    - Composite-rigid-body method is efficient in this kind
2. Articulated-body algorithm
    - $O(N)$ algorithm

References

- R. Featherstone, The Calculation of Robot Dynamics Using Articulated-Body Inertias, IJRR 1983
- Springer Handbook of Robotics Ch. 2 – Can read online from the library
- Roy Featherstone's home page
  - http://royfeatherstone.org/abstracts.html#RAM-tutorial-1