南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Robot Modeling & Control ME331

Section 15： Dynamics V

Chenglong Fu（付成龙）
Dept. of MEE， SUSTech

Initiative project

Project Proposal （项目开题）

PPT Presentation Date: May 12，2025

（5 minutes PPT Group Report + 4 minutes discussion）

1. 小组成员（Team member）
2. 研究动机（Motivation）
3. 研究内容（Research contents）
4. 大致思路（General idea）
5. 分工与工作计划（Working plan）

Outline

- Review
  - Using MATLAB to Obtain the Model
  - Generalized Forces
- Examples: dynamic biped walking
- Newton-Euler Method
  - Newton-Euler Formulation
  - Forward-backward Recursion
  - Planar Elbow Manipulator Revisited
  - Examples： Double Pendulum

Equations of Motion

- **Derive dynamic equations - Lagrangian method**
  - **Coordinate system, generalized coordinates, generalized forces**
  - **Calculate the position and velocity of CoM for each link**
  - **Calculate the total kinetic energy of robot** $K(q, \dot{q})$
  - **Calculate the total potential energy of robot** $P(q)$
  - **Lagrange function** $L = K - P$
  - **Substituting into dynamic equation**

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i \quad \Rightarrow \quad \boldsymbol{D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau}$$

Using MATLAB to Obtain the Model

- **Using Matlab Symbolic Toolbox to Derive the Model**

- **%% Calculate the potential energy**
- PE=g*m*p1(2)+g*Mh*p2(2)+g*m*p3(2);

- **%% Calculate the CoM velocity for each link**
- v1=jacobian(p1,q)*dq;
- v2=jacobian(p2,q)*dq;
- v3=jacobian(p3,q)*dq;

- **%% Calculate the kinematic energy**
- KE1=simplify((1/2)*m*v1.'*v1);
- KE2=simplify((1/2)*Mh*v2.'*v2);
- KE3=simplify((1/2)*m*v3.'*v3);
- KE=KE1+KE2+KE3;

Using MATLAB to Obtain the Model

- **Using Matlab Symbolic Toolbox to Derive the Model**

- % D(q)ddq + C(q,dq)*dq + G(q) = tau

- **%% Calculate the inertial matrix**
- D=jacobian(KE,dq).';
- D=jacobian(D,dq);

- **%% Calculate the gravity vector**
- G=jacobian(PE,q).';

Using MATLAB to Obtain the Model

- **Using Matlab Symbolic Toolbox to Derive the Model**
- %% D(q)ddq + C(q,dq)*dq + G(q) = tau
- **%% Calculate the matrix $C(q,\dot{q})$**
- syms C real
- n=max(size(q));
- for k=1:n
- for j=1:n
- C(k,j)=0;
- for i=1:n
- C(k,j)=C(k,j)+(1/2)*(diff(D(k,j),q(i))+diff(D(k,i),q(j))-diff(D(i,j),q(k)))*dq(i);
- end
- end
- end
- C

$$c_{kj} = \sum_{i=1}^{n} c_{ijk}(q)\dot{q}_i = \sum_{i=1}^{n} \frac{1}{2}\left\{\frac{\partial d_{kj}}{\partial q_i} + \frac{\partial d_{ki}}{\partial q_j} - \frac{\partial d_{ij}}{\partial q_k}\right\}\dot{q}_i$$

Using MATLAB to Obtain the Model

- **Very convenient to derive model, compute control laws, and perform simulations in a common environment:**

– we have been using MATLAB

– model derived using SYMBOLIC TOOLBOX

– m-files for ODE45 or Simulink are automatically generated from the symbolic computations

– relevant files can be downloaded at

http://www.eecs.umich.edu/~grizzle/papers/Three_link_walker.zip

J.W. Grizzle, Gabriel Abba and Franck Plestan, Asymptotically Stable Walking for Biped Robots: Analysis via Systems with Impulse Effects, IEEE T-AC, Volume 46, No. 1, January 2001, pp. 51-64

http://www.eecs.umich.edu/~grizzle/papers/Westervelt_Grizzle_five_link_walking_simulator.zip

E.R. Westervelt, J.W. Grizzle, and D.E. Koditschek, Hybrid Zero Dynamics of Planar Biped Walkers. Published, IEEE-TAC, Vol. 48, No. 1, January 2003, pp. 42-56

Generalized Forces

- **Actuation torques（驱动转矩）**

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{B}\boldsymbol{\tau}$$

- **Viscous damping（阻尼力）**

$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{B}\boldsymbol{\tau} - \boldsymbol{F}_v\dot{\boldsymbol{q}}$$

where $\boldsymbol{F}_v$ is the coefficient of damping matrix（n×n, diagonal）.

- **Coulomb friction（摩擦力）**

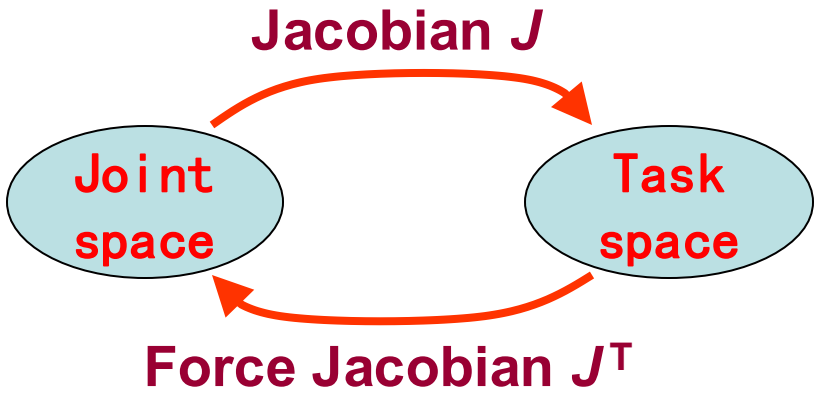
$$\boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{B}\boldsymbol{\tau} - \boldsymbol{F}_v\dot{\boldsymbol{q}} - \boldsymbol{F}_s\mathrm{sgn}(\dot{\boldsymbol{q}})$$

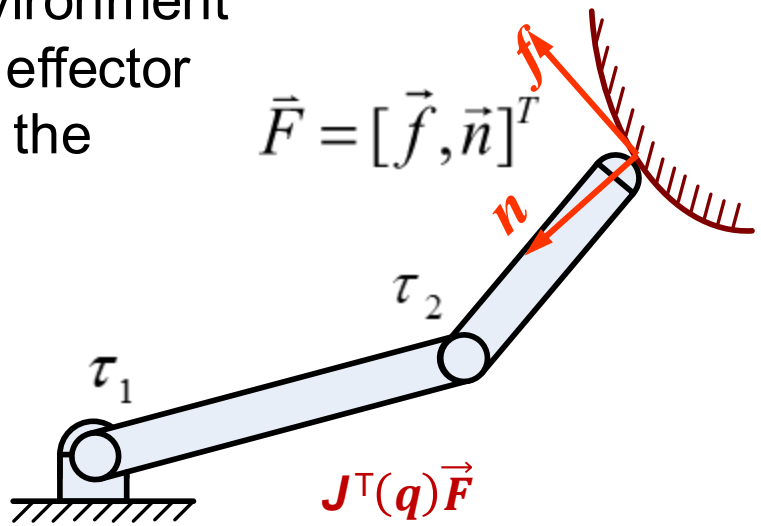
where $\boldsymbol{F}_s$ is the coefficient of friction matrix（n×n, diagonal）, and $\mathrm{sgn}(\dot{q})$ is the vector of sign function (n× 1)，determined by the sign of the joint velocity.

Generalized Forces

- **Contact forces**

Interaction of the manipulator with the environment produces forces and moments at the end effector or tool. These, in turn, produce torques at the joints of the robot.



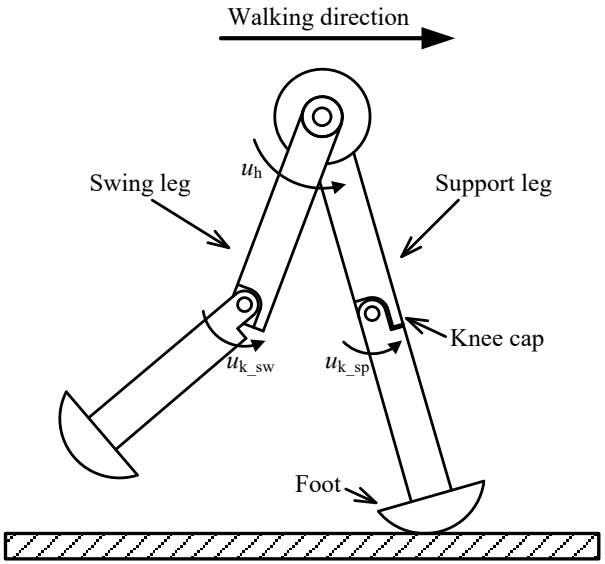


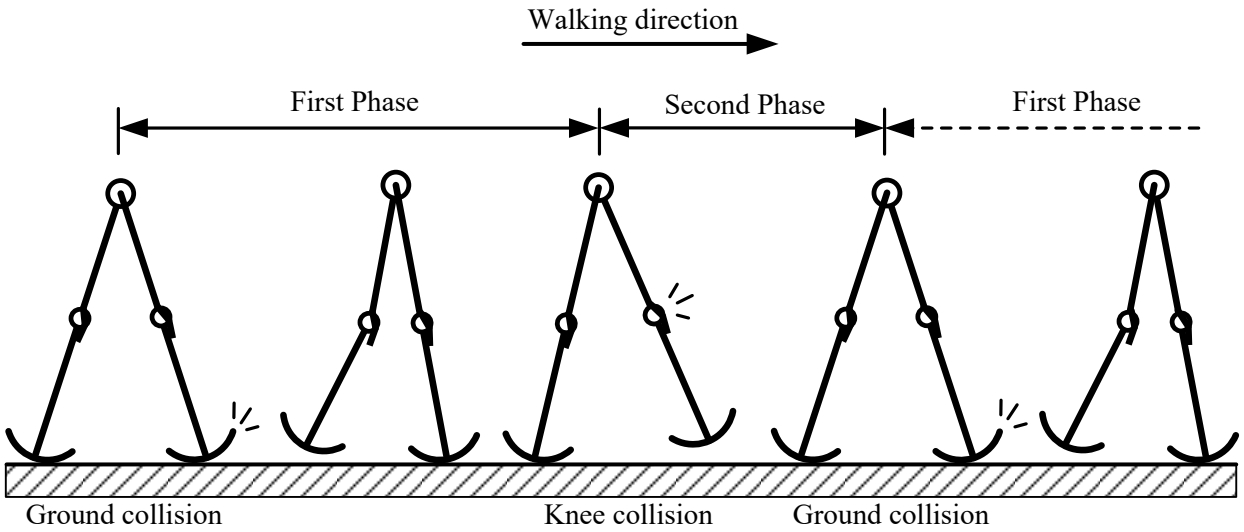
$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = B\tau - F_v\dot{q} - F_s\mathrm{sgn}(\dot{q}) + J^T(q)\vec{F}$$

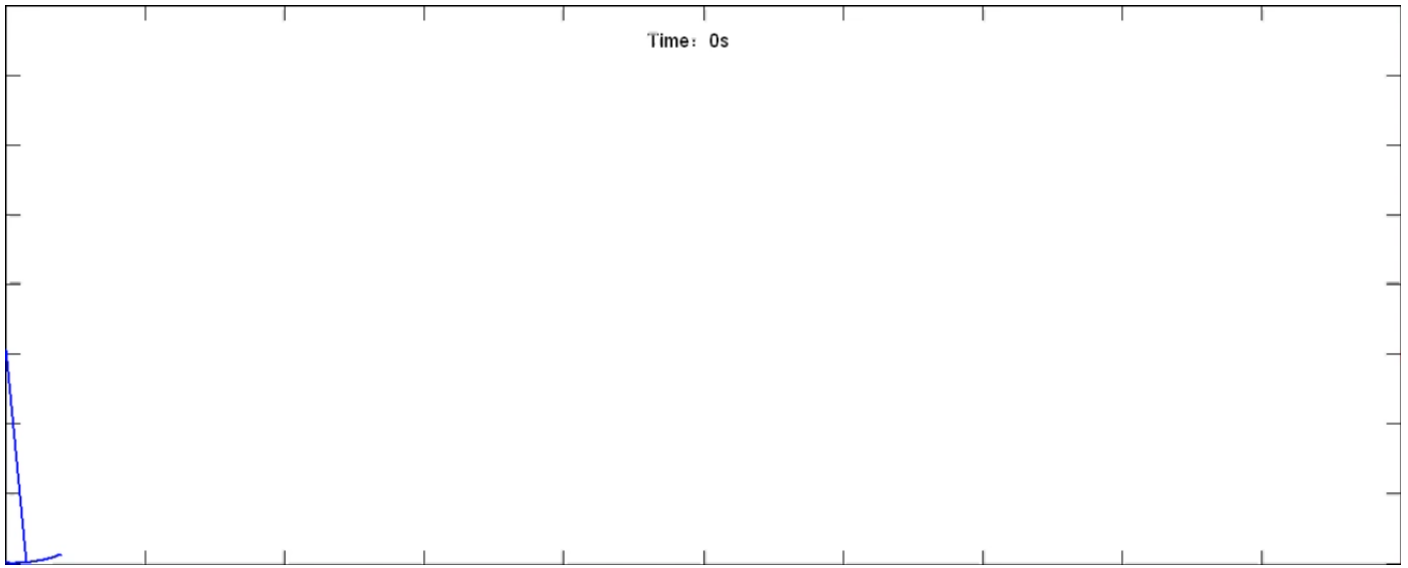
where $\vec{F}$ represent the vector of forces and moments at the end effector,

$J(q)$ is the transpose of the manipulator Jacobian。

Example: dynamic walking





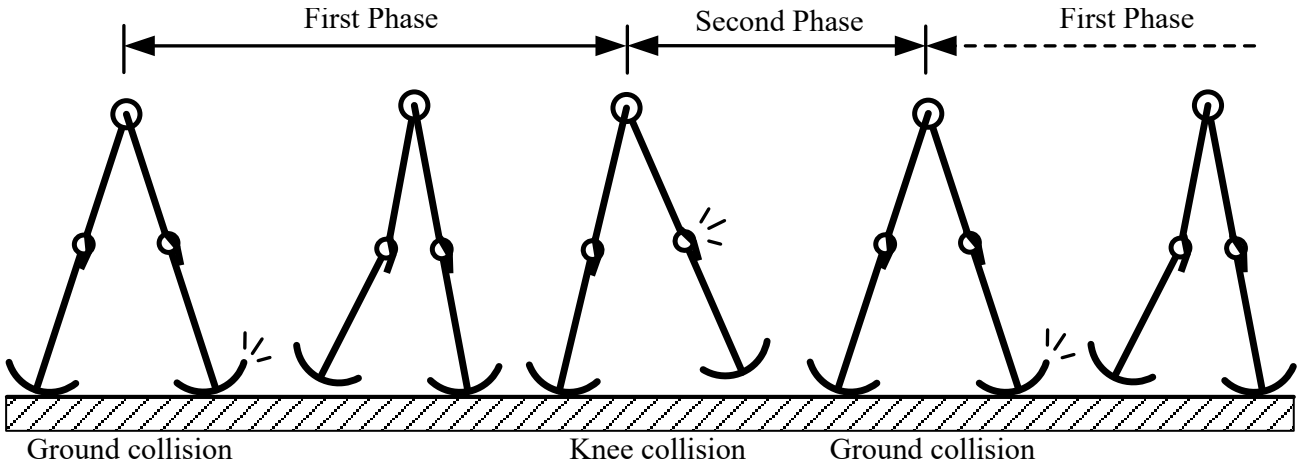


Fu, C., Tan, F., Chen, K. (2010). A simple walking strategy for biped walking based on an intermittent sinusoidal oscillator. ***Robotica***, *28*(6), 869-884.

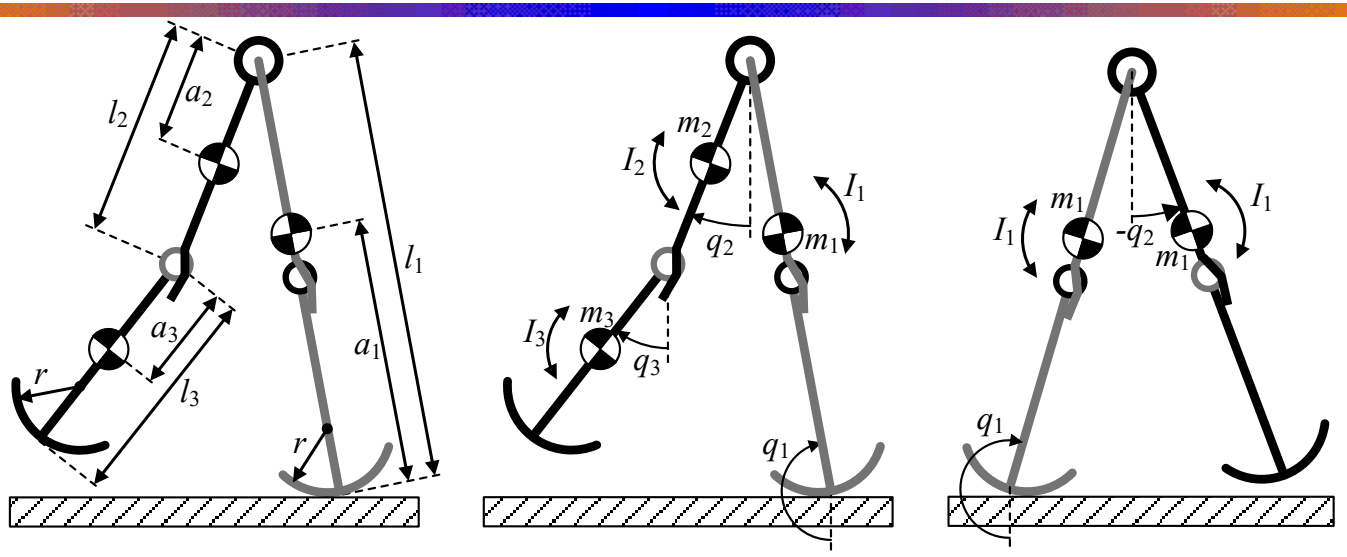
Example: dynamic walking

The hypotheses assumed for desired walking gait is enumerated:

1. **Walking consists of a single support phase and an instantaneous double support phase.**
2. **The support leg is always straight during walking.**
3. **The swing leg is fully extended before ground collision.**
4. **The contact of the swing shank with the knee cap is assumed to be a completely inelastic collision.**
5. **The contact of the swing leg with the ground is also assumed to be a completely inelastic collision.**



Example: dynamic walking

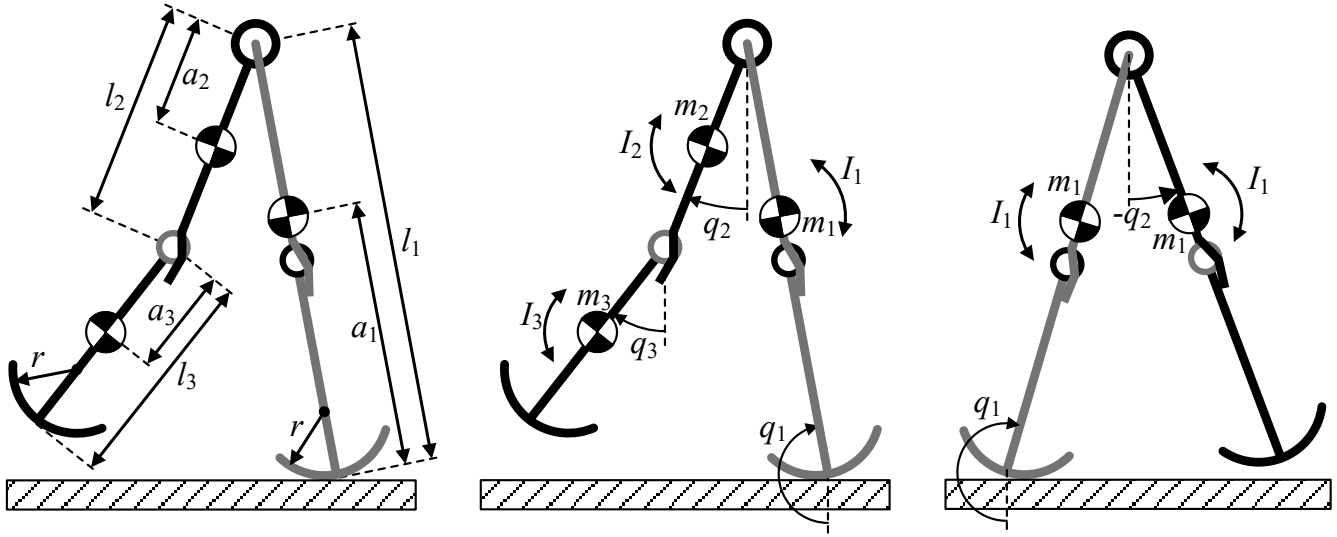


(a) Link parameters (b) Coordinates for phase 1 (c) Coordinates for phase 2

MODEL PARAMETERS	UNIT	LINK	LABEL
Mass	kg	Leg	m_1
		Thigh	m_2
		Shank	m_3
Length	m	Leg	l_1
		Thigh	l_2
		Shank	l_3
		Foot radius	r
Inertia at mass center	m^2kg	Leg	I_1
		Thigh	I_2
		Shank	I_3
Center of mass	m	Leg	a_1
		Thigh	a_2
		Shank	a_3

Example: dynamic walking

Motion equations for phase 1 :



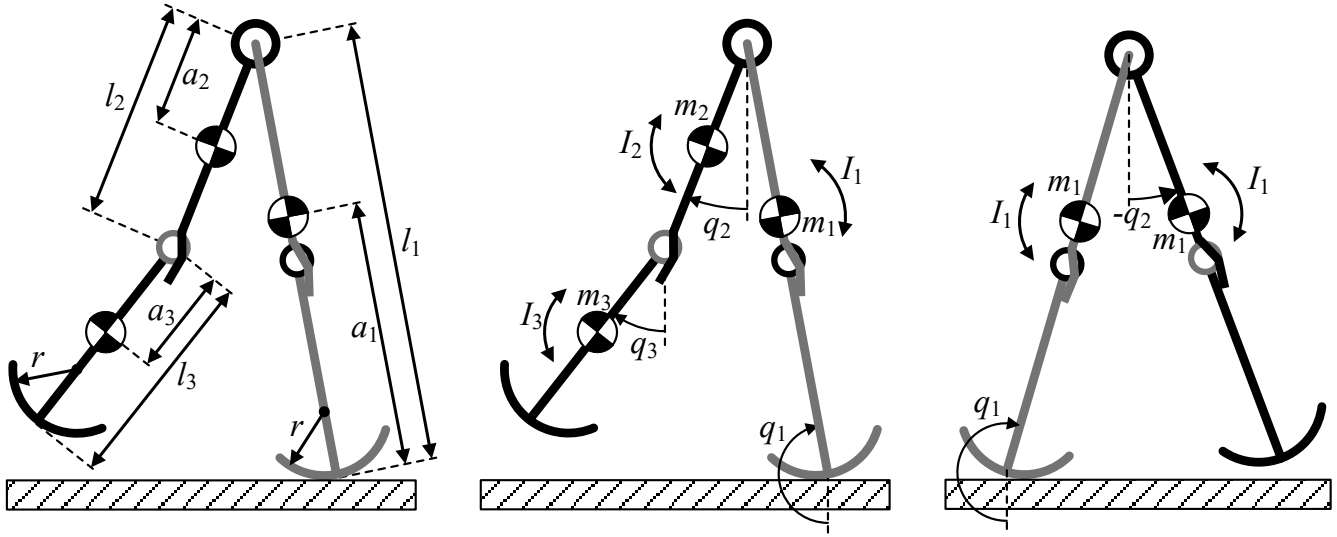
(a) Link parameters (b) Coordinates for phase 1 (c) Coordinates for phase 2

Phase 1 can be represented as a 3-DOF link system. The generalized coordinates for phase 1 are defined as (b). The equation of motion for phase 1 can be written as

$$\begin{bmatrix} D1_{11} & D1_{12} & D1_{13} \\ & D1_{22} & D1_{23} \\ sym & & D1_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} C1_{11} & C1_{12} & C1_{13} \\ C1_{21} & 0 & C1_{23} \\ C1_{31} & C1_{32} & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} + \begin{bmatrix} G1_1 \\ G1_2 \\ G1_3 \end{bmatrix} = \begin{bmatrix} u_h \\ -u_h + u_{k\_sw} \\ -u_{k\_sw} \end{bmatrix}$$

Example: dynamic walking

Motion equations for phase 2 :



(a) Link parameters (b) Coordinates for phase 1 (c) Coordinates for phase 2

Phase 2 can be represented as a 2-DOF link system. The generalized coordinates for phase 1 are defined as (c) . The equation of motion for phase 1 can be written as

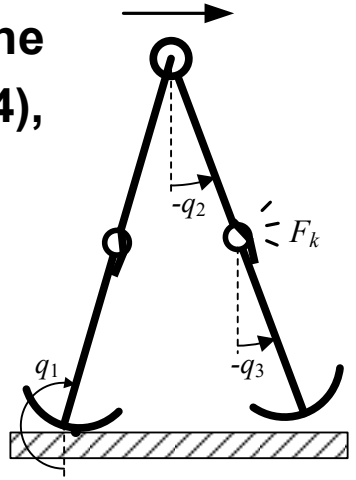
$$\begin{bmatrix} D2_{11} & D2_{12} \\ sym & D2_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \begin{bmatrix} C2_{11} & C2_{12} \\ C2_{21} & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \begin{bmatrix} G2_1 \\ G2_2 \end{bmatrix} = \begin{bmatrix} u_h \\ -u_h \end{bmatrix}$$

Example: dynamic walking

Equations governing knee collision:

After the end of the first phase knee collision occurs when the shank comes in line with the thigh. Under Hypothesis H4), according to the angular momentum theorem, we can obtain

$$\boldsymbol{D1}(q^+)\dot{q}^+ - \boldsymbol{D1}(q^-)\dot{q}^- = \begin{bmatrix} 0 \\ -\hat{F}_{\mathrm{k}} \\ \hat{F}_{\mathrm{k}} \end{bmatrix}$$



(a) Knee collision

where $\hat{F}_{\mathrm{k}}$ is the integral of the torque acting on the shank during knee collision.

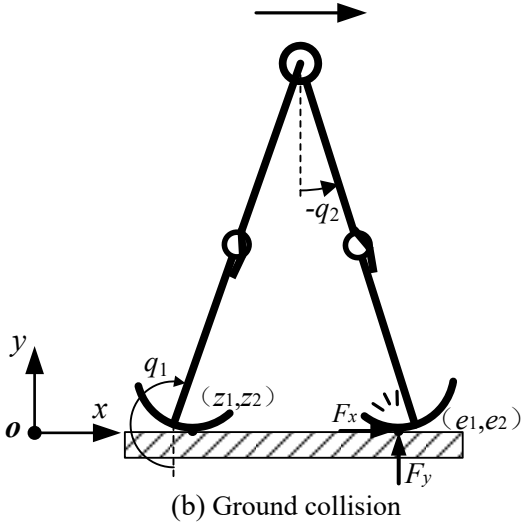
After the collision, the angular velocities of the shank and thigh become same and therefore $[0 \quad -1 \quad 1]\dot{q}^+ = 0.$

Solving above two Eqs. yields

$$\begin{bmatrix} \dot{q}^+ \\ \hat{F}_{\mathrm{k}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{D1}(q^+) & \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} \\ \begin{bmatrix} 0 & -1 & 1 \end{bmatrix} & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{D1}(q^-)\dot{q}^- \\ 0 \end{bmatrix}$$

Example: dynamic walking

Equations governing ground collision :



(b) Ground collision

After the end of the second phase ground collision occurs when the swing foot touches the ground. The contact model of ground collision requires the four degrees of freedom of the robot. Select the generalized coordinates to be $\boldsymbol{q}_{\mathrm{f}} = [q_1, q_2, z_1, z_2]'$

This gives once again a model of the form

$$\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}})\ddot{\boldsymbol{q}}_{\mathrm{f}} + \boldsymbol{C}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}, \dot{\boldsymbol{q}}_{\mathrm{f}})\dot{\boldsymbol{q}}_{\mathrm{f}} + \boldsymbol{G}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}) = \boldsymbol{B}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}})\boldsymbol{u} + \left(\frac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right)' \begin{bmatrix} F_x \\ F_y \end{bmatrix}$$

According to the angular momentum theorem, we can obtain

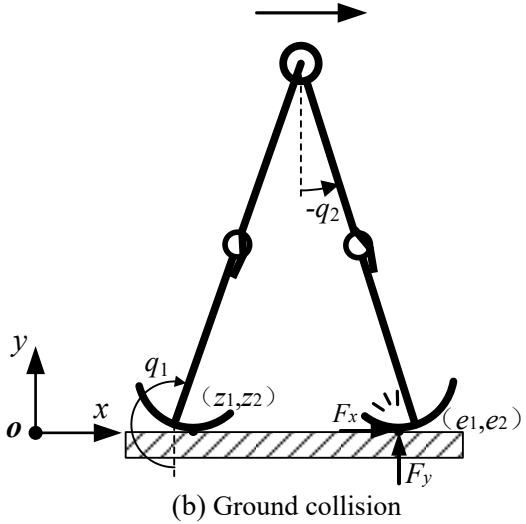
$$\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{+}) \cdot \dot{\boldsymbol{q}}_{\mathrm{f}}^{+} - \boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{-}) \cdot \dot{\boldsymbol{q}}_{\mathrm{f}}^{-} = \left(\frac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right)' \begin{bmatrix} \hat{F}_x \\ \hat{F}_y \end{bmatrix}$$

$$\hat{F}_x = \int_{t^-}^{t^+} F_x(\tau)\mathrm{d}\tau \qquad \hat{F}_y = \int_{t^-}^{t^+} F_y(\tau)\mathrm{d}\tau$$

$$\left.\frac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right|_{\boldsymbol{q}_{\mathrm{f}} = \boldsymbol{q}_{\mathrm{f}}^{-}} \cdot \dot{\boldsymbol{q}}_{\mathrm{f}}^{+} = 0$$

Example: dynamic walking

Equations governing ground collision :



(b) Ground collision

$$\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{+})\cdot\dot{\boldsymbol{q}}_{\mathrm{f}}^{+}-\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{-})\cdot\dot{\boldsymbol{q}}_{\mathrm{f}}^{-}=\left(\frac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right)'\begin{bmatrix}\hat{F}_x\\ \hat{F}_y\end{bmatrix}$$

$$\left.\frac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right|_{\boldsymbol{q}_{\mathrm{f}}=\boldsymbol{q}_{\mathrm{f}}^{-}}\cdot\dot{\boldsymbol{q}}_{\mathrm{f}}^{+}=0$$

Solving above equations yields

$$\begin{bmatrix}\dot{\boldsymbol{q}}_{\mathrm{f}}^{+}\\ \begin{bmatrix}\hat{F}_x\\ \hat{F}_y\end{bmatrix}\end{bmatrix}=\begin{bmatrix}\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{-}) & -\left(\dfrac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right)'\Bigg|_{\boldsymbol{q}_{\mathrm{f}}=\boldsymbol{q}_{\mathrm{f}}^{-}}\\ \left.\left(\dfrac{\partial \boldsymbol{E}(\boldsymbol{q}_{\mathrm{f}})}{\partial \boldsymbol{q}_{\mathrm{f}}}\right)\right|_{\boldsymbol{q}_{\mathrm{f}}=\boldsymbol{q}_{\mathrm{f}}^{-}} & \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\end{bmatrix}^{-1}\begin{bmatrix}\boldsymbol{D}_{\mathrm{f}}(\boldsymbol{q}_{\mathrm{f}}^{-})\cdot\dot{\boldsymbol{q}}_{\mathrm{f}}^{-}\\ \begin{bmatrix}0\\ 0\end{bmatrix}\end{bmatrix}$$

Fu, C., Tan, F., Chen, K. (2010). A simple walking strategy for biped walking based on an intermittent sinusoidal oscillator. ***Robotica***, *28*(6), 869-884.

Newton-Euler Method

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

    = - applied by body i+1 to body i
- **Newton dynamic equation**
  - balance: sum of forces = variation of linear momentum

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

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

$$\tau_0 = \frac{d}{dt}(I_0\omega_0)$$

where $I_0 = RIR^{\mathrm{T}}$.

expressed in inertial frame（absolute frame）

Newton-Euler Method

- **Euler dynamic equation expressed in attached frame**

  The angular momentum expressed in the inertial frame, is

$$h = I_0 \omega_0$$
$$= RIR^{\mathrm{T}} R\omega$$
$$= RI\omega$$

  The rate of change of the angular momentum expressed in the inertial frame, is

$$\dot{h} = \dot{R}I\omega + RI\dot{\omega} \qquad \dot{R} = S(\omega_0)\, R$$
$$= S(\omega_0)\, RI\omega + RI\dot{\omega}$$

  The rate of change of the angular momentum expressed in the body-attached frame, is

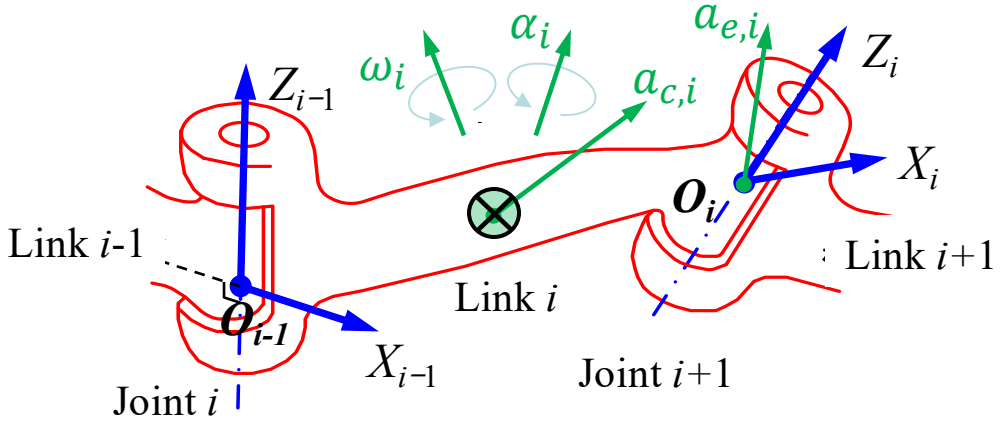
$$\tau = R^{\mathrm{T}}\dot{h} = R^{\mathrm{T}} S(\omega_0)\, RI\omega + I\dot{\omega}$$
$$= S(R^{\mathrm{T}}\omega_0)\, I\omega + I\dot{\omega}$$
$$= S(\omega)\, I\omega + I\dot{\omega}$$
$$= \omega \times (I\omega) + I\dot{\omega}$$

$$\tau = I\dot{\omega} + \omega \times (I\omega)$$

Newton-Euler Formulation

- **Euler Newton–Euler formulation of an n-link manipulator**
  - ❑ Choose frames 0, ... , n, where frame 0 is an inertial frame, and frame i is rigidly attached to link i for $i \geq 1$.
  - ❑ Introduce several vectors, which are all expressed in frame i.

    The first set of vectors pertains to the velocities and accelerations.



$a_{c,i}$ = the acceleration of the center of mass of link i

$a_{e,i}$ = the acceleration of the end of link i (i. e., joint $i+1$)

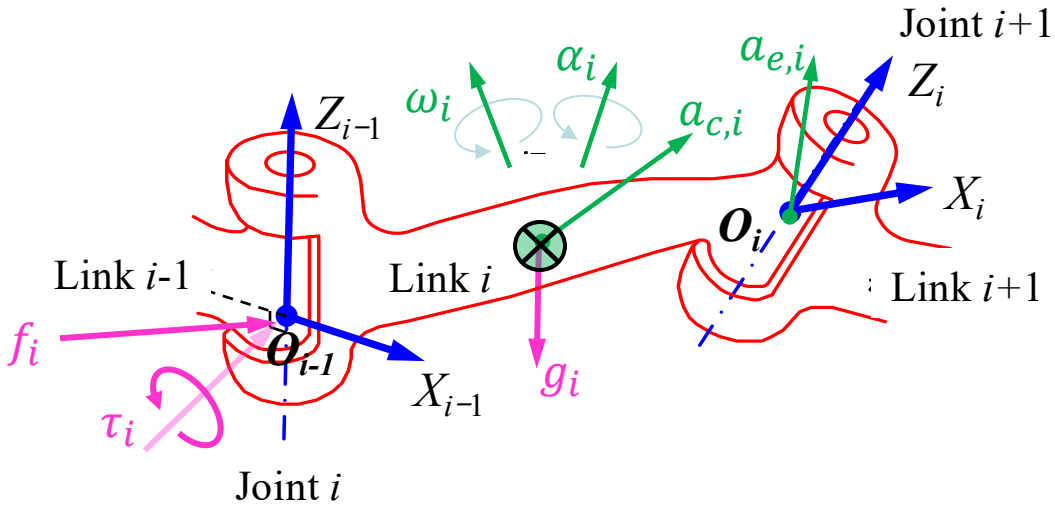
ω_i = the angular velocity of frame i w.r.t. frame 0 ($= \omega_{i,0}^{i}$)

α_i = the angular acceleration of frame i w.r.t. frame 0 ($= \alpha_{i,0}^{i}$)

Newton-Euler Formulation

- **Euler Newton–Euler formulation of an n-link manipulator**
  - ❑ Introduce several vectors, which are all expressed in frame i.

    The next several vectors pertain to forces and torques.



g_i = the acceleration due to gravity (expressed in frame i)

f_i = the force exerted by link $i-1$ on link i

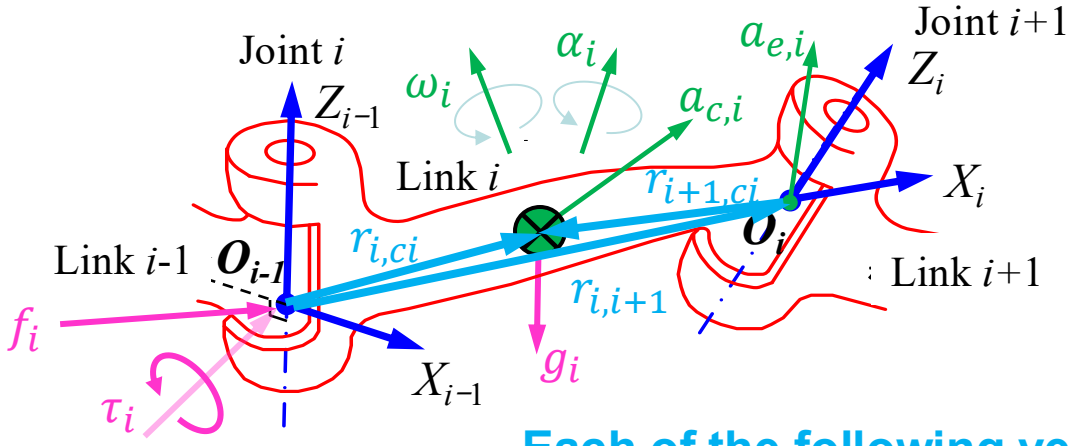
τ_i = the torque exerted by link $i-1$ on link i

R_{i+1}^{i} = the rotation matrix from frame $i+1$ to frame i

Newton-Euler Formulation

- **Euler Newton–Euler formulation of an n-link manipulator**
  - ❑ Introduce several vectors, which are all expressed in frame i.

    The final set of vectors pertain to physical features of the manipulator.



Each of the following vectors is constant !

m_i = the mass of link i

I_i = the inertia matrix of link i about a frame parallel to frame i whose origin is at the center of mass of link i

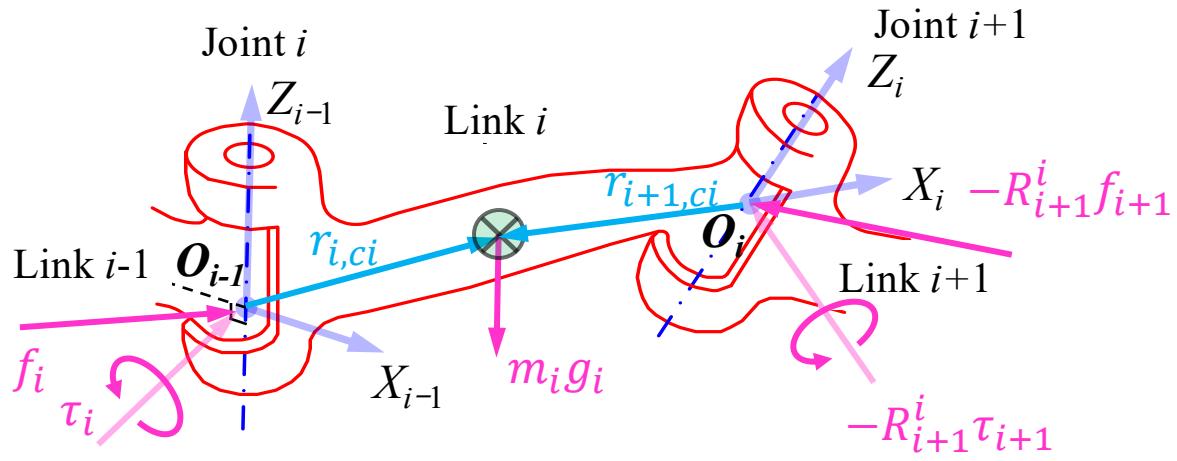
$r_{i,ci}$ = the vector from joint i to the center of mass of link i

$r_{i+1,ci}$ = the vector from joint i+1 to the center of mass of link i

$r_{i,i+1}$ = the vector from joint i to joint i+1

Newton-Euler Formulation

- **Euler Newton–Euler formulation of an n-link manipulator**
  - ❑ Link i together with all forces and torques acting on it

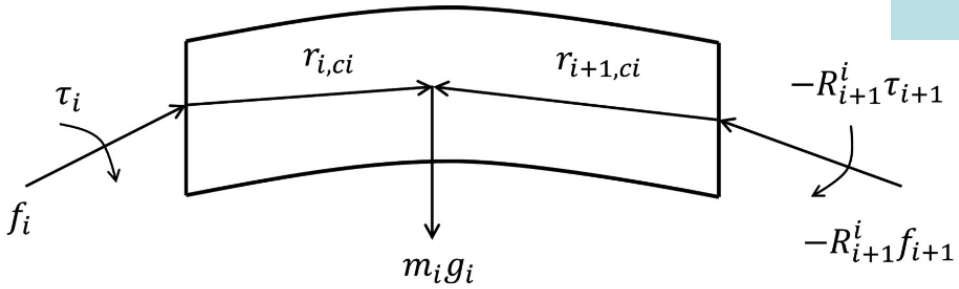


force balance equation

$$f_i - R^i_{i+1} f_{i+1} + m_i g_i = m_i a_{c,i}$$

moment balance equation

$$\tau_i - R^i_{i+1}\tau_{i+1} + f_i \times r_{i,ci} - \left(R^i_{i+1} f_{i+1}\right) \times r_{i+1,ci} = I_i \alpha_i + \omega_i \times (I_i \omega_i)$$

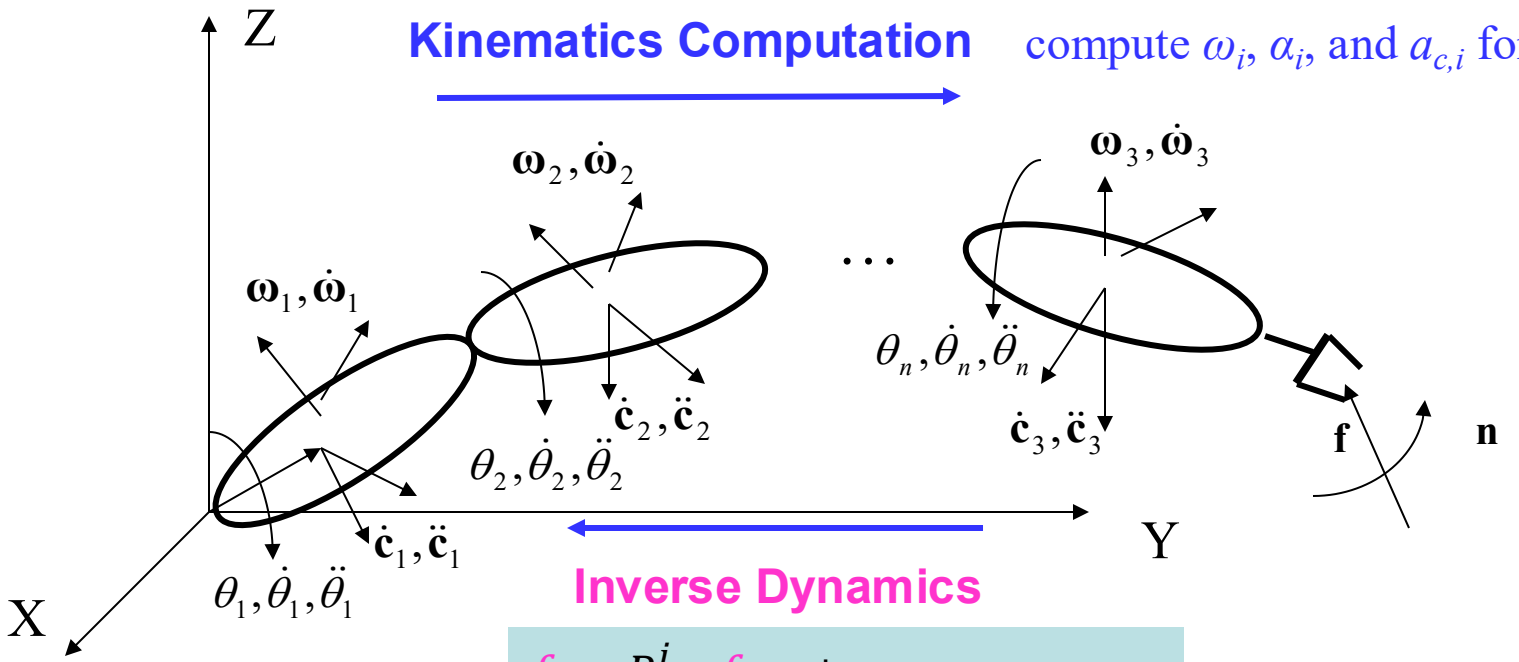


Newton–Euler formulation:
Find the f and τ corresponding to a **particular trajectory** $q(t)$.

Forward-backward Recursion

Inverse Dynamics: robot motion → joint torque

Algorithm implementation: write Newton-Euler equations from terminal to base.



$$f_i = R_{i+1}^{i} f_{i+1} + m_i a_{c,i} - m_i g_i$$

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

Forward-backward Recursion

1. Start with the initial conditions

$$\omega_0 = 0,\ \alpha_0 = 0,\ a_{c,0} = 0,\ a_{e,0} = 0$$

Compute ω_i, α_i, $a_{c,i}$ for i increasing from 1 to n.

$$\omega_i = (R_i^{i-1})^T \omega_{i-1} + b_i \dot{q}_i \quad \text{where } b_i = (R_i^0)^T z_{i-1}$$

$$\alpha_i = (R_i^{i-1})^T \alpha_{i-1} + b_i \ddot{q}_i + \omega_i \times b_i \dot{q}_i$$

$$a_{e,i} = (R_i^{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i,i+1} + \omega_i \times (\omega_i \times r_{i,i+1})$$

$$a_{c,i} = (R_i^{i-1})^T a_{e,i-1} + \dot{\omega}_i \times r_{i,ci} + \omega_i \times (\omega_i \times r_{i,ci})$$

2. Start with the terminal conditions

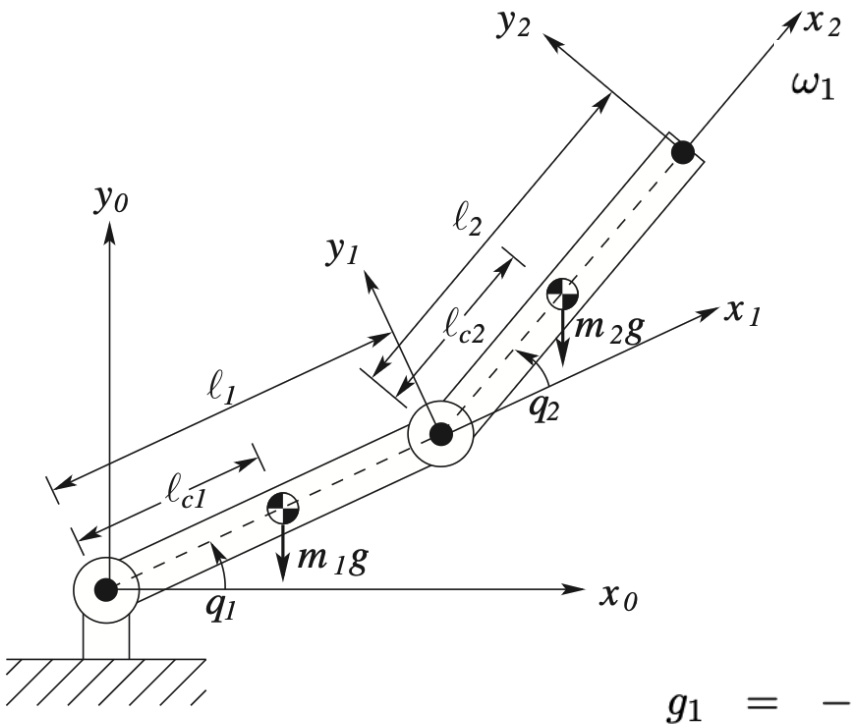
$$f_{n+1} = 0,\ \tau_{n+1} = 0$$

Compute f_i, τ_i for i decreasing from n to 1.

$$f_i = R_{i+1}^i f_{i+1} + m_i a_{c,i} - m_i g_i$$

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

Planar Elbow Manipulator Revisited



$$\omega_1 = \dot{q}_1 k,\ \alpha_1 = \ddot{q}_1 k,\ \omega_2 = (\dot{q}_1 + \dot{q}_2)k,\ \alpha_2 = (\ddot{q}_1 + \ddot{q}_2)k$$

$$r_{1,c1} = \ell_{c1} i,\ r_{2,c1} = (\ell_{c1} - \ell_1)i,\ r_{1,2} = \ell_1 i$$

$$r_{2,c2} = \ell_{c2} i,\ r_{3,c2} = (\ell_{c2} - \ell_2)i,\ r_{2,3} = \ell_2 i$$

Forward Recursion : Link 1

$$a_{c,1} = \ddot{q}_1 k \times \ell_{c1} i + \dot{q}_1 k \times (\dot{q}_1 k \times \ell_{c1} i)$$

$$= \ell_{c1}\ddot{q}_1 j - \ell_{c1}\dot{q}_1^2 i = \begin{bmatrix} -\ell_{c1}\dot{q}_1^2 \\ \ell_c \ddot{q}_1 \\ 0 \end{bmatrix}$$

$$g_1 = -(R_1^0)^T g j = g \begin{bmatrix} -\sin q_1 \\ -\cos q_1 \end{bmatrix} \qquad a_{e,1} = \begin{bmatrix} -\ell_1 \dot{q}_1^2 \\ \ell_1 \ddot{q}_1 \end{bmatrix}$$

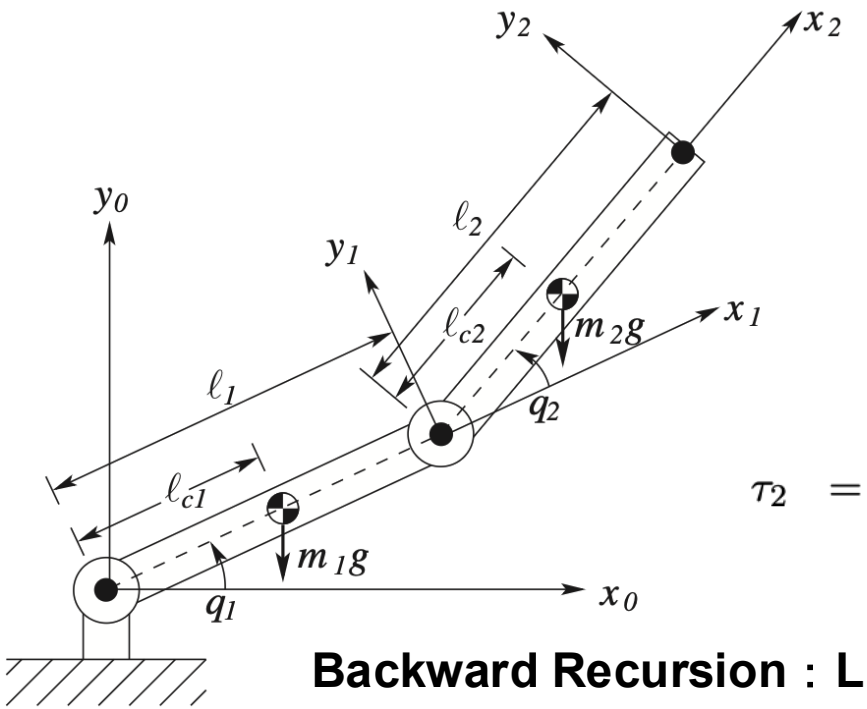
Forward Recursion : Link 2

$$a_{c,2} = (R_2^1)^T a_{e,1} + [(\ddot{q}_1 + \ddot{q}_2)k] \times \ell_{c2} i + (\dot{q}_1 + \dot{q}_2)k \times [(\dot{q}_1 + \dot{q}_2)k \times \ell_{c2} i]$$

$$= \begin{bmatrix} -\ell_1 \dot{q}_1^2 \cos q_2 + \ell_1 \ddot{q}_1 \sin q_2 - \ell_{c2}(\dot{q}_1 + \dot{q}_2)^2 \\ \ell_1 \dot{q}_1^2 \sin q_2 + \ell_1 \ddot{q}_1 \cos q_2 - \ell_{c2}(\ddot{q}_1 + \ddot{q}_2) \end{bmatrix}$$

$$g_2 = g \begin{bmatrix} \sin(q_1 + q_2) \\ -\cos(q_1 + q_2) \\ 0 \end{bmatrix}$$

Planar Elbow Manipulator Revisited



Backward Recursion : Link 2

$i = 2$ and note that $f_3 = 0$. This results in

$$\begin{aligned} f_2 &= m_2 a_{c,2} - m_2 g_2 \\ \tau_2 &= I_2\alpha_2 + \omega_2 \times (I_2\omega_2) - f_2 \times \ell_{c2} i \end{aligned}$$

The final result is

$$\begin{aligned} \tau_2 = {} & I_2(\ddot{q}_1 + \ddot{q}_2)k + [m_2\ell_1\ell_{c2}\sin q_2\dot{q}_1^2 + m_2\ell_1\ell_{c2}\cos q_2\ddot{q}_1 \\ & + m_2\ell_{c2}^2(\ddot{q}_1 + \ddot{q}_2) + m)2\ell_{c2}g\cos(q_1 + q_2)]k \end{aligned}$$

Backward Recursion : Link 1

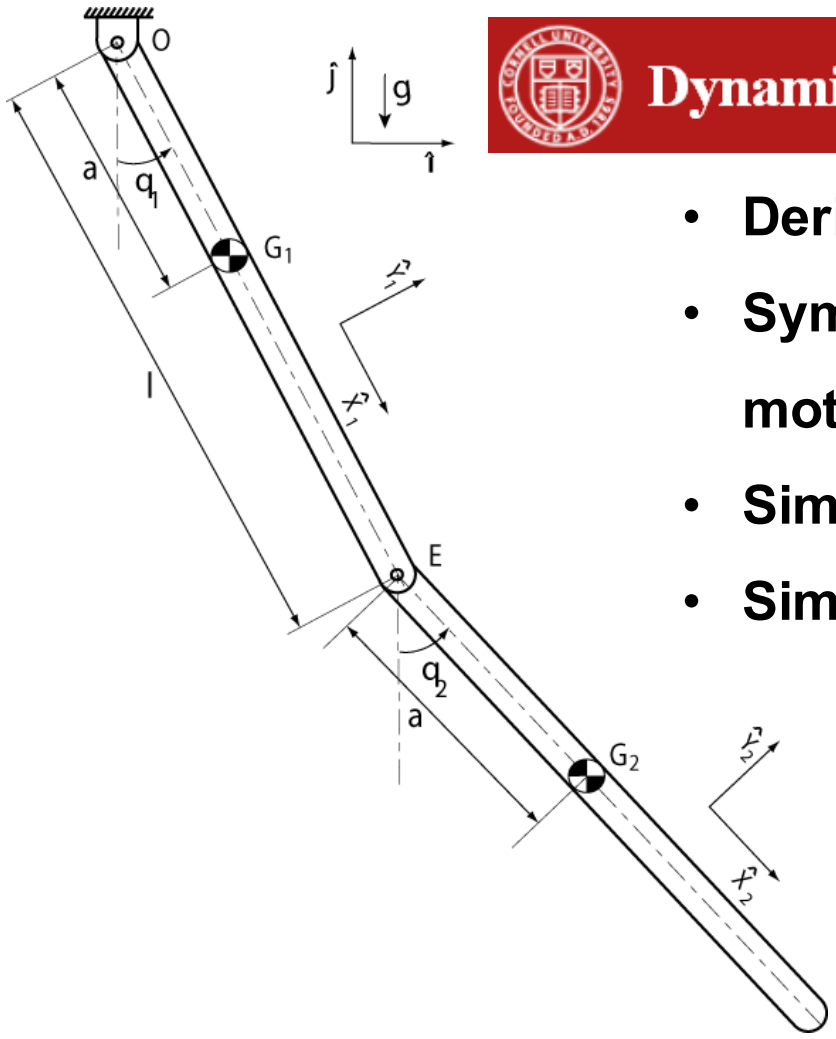
$$f_1 = m_1 a_{c,1} + R_2^1 f_2 - m_1 g_1$$

$$\begin{aligned} \tau_1 = {} & R_2^1\tau_2 - f_1 \times \ell_{c,1} i - (R_2^1 f_2) \times (\ell_1 - \ell_{c1})i \\ & + I_1\alpha_1 + \omega_1 \times (I_1\omega_1) \end{aligned}$$

The final result is

$$\begin{aligned} \tau_1 = {} & \tau_2 + m_1\ell_{c1}^2 + m_1\ell_{c1}g\cos q_1 + m_2\ell_1 g\cos q_1 + I_1\ddot{q}_1 \\ & + m_2\ell_1^2\ddot{q}_1 - m_1\ell_1\ell_{c2}(\dot{q}_1 + \dot{q}_2)^2\sin q_2 + m_2\ell_1\ell_{c2}(\ddot{q}_1 + \ddot{q}_2)\cos q_2 \end{aligned}$$

Newton-Euler： Double Pendulum



Dynamic Walking MATLAB Simulation Guide

- **Derivation of equations of motion by hand;**
- **Symbolic derivation of the equations of motion in MATLAB;**
- **Simulation of the equations of motion;**
- **Simulation checks.**

Fig. Double pendulum with the assumed coordinate systems, dimensions and angles.

http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/ranger_paper/Reports/Ranger_Robot/control/simulator/doublependulum.html

Newton-Euler: Double Pendulum

Finding the Equations of Motion

- To find the equations of motion for a dynamic system, we use the Newton-Euler method.
- This method involves balancing the linear and angular momentum of a system.
- For our example, we will only perform angular momentum balances. Angular momentum balances must take place about a point and can be expressed as,

$$\sum M_{/C} = \dot{H}_{/C}$$

(Sum of the moments is equal to the rate of change of angular momentum.)

Newton-Euler: Double Pendulum

Finding the Equations of Motion

Angular Momentum balance about point O:

$$\vec{M}_{/O} = \dot{\vec{H}}_{/O}$$

$$\vec{M}_{/O} = \vec{r}_{G1/O} \times -m_1 g\hat{j} + \vec{r}_{G2/O} \times -m_2 g\hat{j} + T_1\hat{k}$$

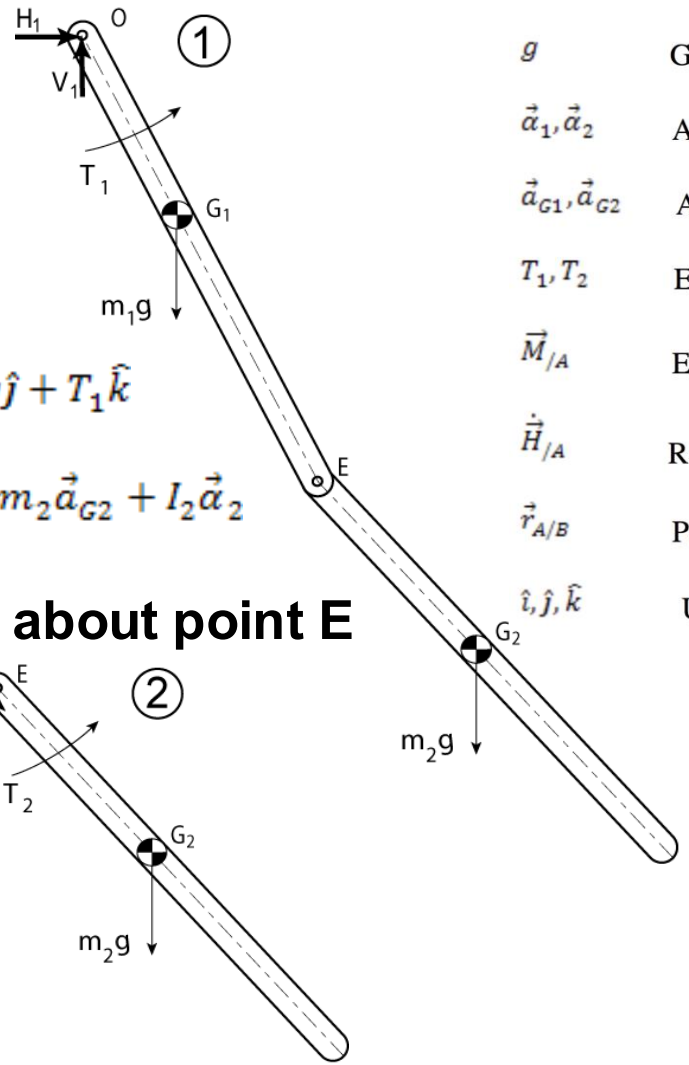
$$\dot{\vec{H}}_{/O} = \vec{r}_{G1/O} \times m_1\vec{a}_{G1} + I_1\vec{\alpha}_1 + \vec{r}_{G2/O} \times m_2\vec{a}_{G2} + I_2\vec{\alpha}_2$$

Angular Momentum balance about point E

$$\vec{M}_{/E} = \dot{\vec{H}}_{/E}$$

$$\vec{M}_{/E} = \vec{r}_{G2/E} \times -m_2 g\hat{j} + T_2\hat{k}$$

$$\dot{\vec{H}}_{/E} = \vec{r}_{G2/E} \times m_2\vec{a}_{G2} + I_2\vec{\alpha}_2$$



m_1, m_2	Masses of links 1 and 2
I_1, I_2	Inertia of links 1 and 2
g	Gravitational acceleration
$\vec{\alpha}_1, \vec{\alpha}_2$	Angular accelerations of links 1 and 2
$\vec{a}_{G1}, \vec{a}_{G2}$	Accelerations of links 1 and 2
T_1, T_2	External torques on links 1 and 2
$\vec{M}_{/A}$	External moment about point A
$\dot{\vec{H}}_{/A}$	Rate of change of angular momentum
$\vec{r}_{A/B}$	Position vector from point B to A
$\hat{\imath}, \hat{\jmath}, \hat{k}$	Unit vectors along x, y and z axes

Newton-Euler： Double Pendulum

Symbolic Deviation in MATLAB

- **Determine the symbolic variables that will be required to solve the problem.**
- **Any variable such angles or distances and its derivatives should be defined as symbolic.**
- **Any parameter in the problem such as length or radius should also be symbolic.**

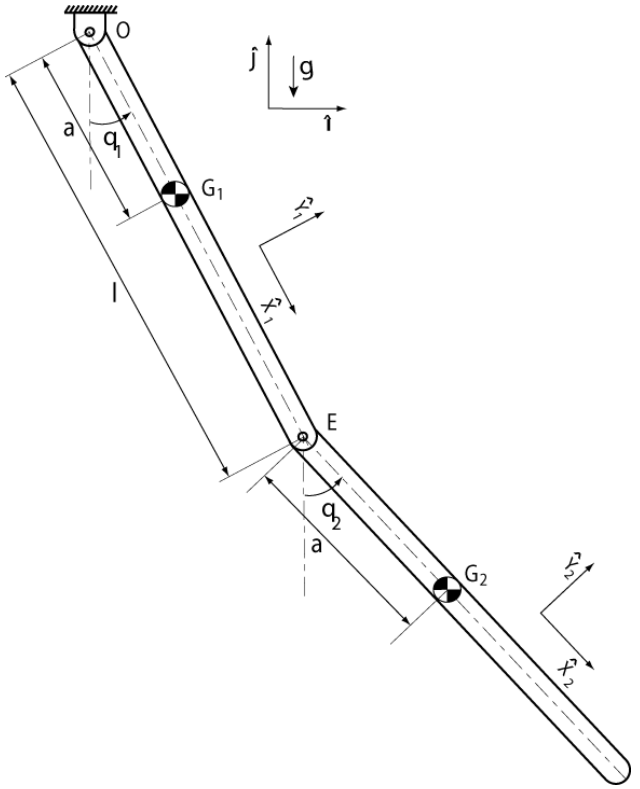
```
%% parameters%%
syms a l m1 m2 I1 I2 g T1 T2
%% angles, angular velocities, angular accelerations%%
syms q1 q2 u1 u2 ud1 ud2
```

Newton-Euler: Double Pendulum

Symbolic Deviation in MATLAB

- **Define the coordinate frames we set up in the solution to our problem and any position vectors that will be needed for plotting and/or solving the problem.**

```
%%%%%%%% Reference frames %%%%%%%%
i = [1 0 0]; j = [0 1 0]; k = [0 0 1];
X1 = sin(q1)*i - cos(q1)*j; Y1 = cos(q1)*i + sin(q1)*j;
X2 = sin(q2)*i - cos(q2)*j; Y2 = cos(q2)*i + sin(q2)*j;
%%%%%%%% Position vectors %%%%%%%%
r_O_G1 = a*X1;
r_O_E = l*X1;
r_E_G2 = a*X2;
r_O_G2 = r_O_E + r_E_G2;
```

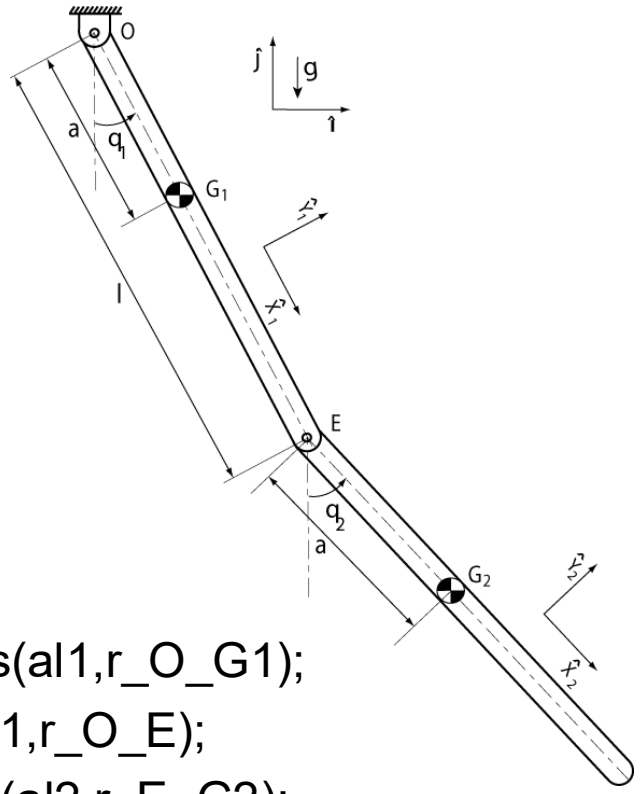


Newton-Euler： Double Pendulum

Symbolic Deviation in MATLAB

- **Define the angular velocities, angular accelerations, mass velocities and mass accelerations in terms of the symbolic variables and position vectors.**

```
%% Angular velocities and accelerations %%
om1 = u1*k; om2 = u2*k;
al1 = ud1*k; al2 = ud2*k;
%%Velocties and acelerations of masses %%
v_O = 0;
v_G1 = v_O + cross(om1,r_O_G1);
v_E = v_O + cross(om1,r_O_E);
v_G2 = v_E + cross(om2,r_E_G2);
a_O = 0;
a_G1 = a_O + cross(om1,cross(om1,r_O_G1)) + cross(al1,r_O_G1);
a_E = a_O + cross(om1,cross(om1,r_O_E)) + cross(al1,r_O_E);
a_G2 = a_E + cross(om2,cross(om2,r_E_G2)) + cross(al2,r_E_G2);
```



Newton-Euler: Double Pendulum

Symbolic Deviation in MATLAB

- **Code the statements for the angular momentum balance. Each angular momentum balance will result in one equation of motion. Equations of motion will be in the *k* direction of resulting vector.**

```
%%%%% Angular Momentum %%%%%%%%%%%%%%%%%%%%%%%
M_O = cross(r_O_G1,-m1*g*j) + cross(r_O_G2,-m2*g*j) + T1*k;
Hdot_O = cross(r_O_G1,m1*a_G1) + I1*al1 + cross(r_O_G2,m2*a_G2) + I2*al2;
M_E = cross(r_E_G2, -m2*g*j) + T2*k;
Hdot_E = cross(r_E_G2, m2*a_G2) + I2*al2;
%%%%% Equations of motion %%%%%%%%%%%%%%%%%%%%%
AMB_O = M_O - Hdot_O;
AMB_E = M_E - Hdot_E;
%%% The k component has the equations of motion %%%
eqn1 = AMB_O(3);
eqn2 = AMB_E(3);
eqn1 = eqn1 - eqn2; %%There is no need to do this but we do it to make the
resulting M matrix symmetric
```

Newton-Euler： Double Pendulum

Symbolic Deviation in MATLAB

- **Manipulate** $\begin{bmatrix} \text{eqn}_1(q, \dot{q}, \ddot{q}) \\ \text{eqn}_2(q, \dot{q}, \ddot{q}) \end{bmatrix} = 0$ **to fit the form of** $\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix}$

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} - \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} = \begin{bmatrix} \text{eqn}_1(q, \dot{q}, \ddot{q}) \\ \text{eqn}_2(q, \dot{q}, \ddot{q}) \end{bmatrix}$$

$$\begin{cases} \ddot{q}_1 = 0 \\ \ddot{q}_2 = 0 \end{cases} \Rightarrow \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} = -\left.\begin{bmatrix} \text{eqn}_1 \\ \text{eqn}_2 \end{bmatrix}\right|_{\substack{\ddot{q}_1=0, \\ \ddot{q}_2=0}}$$

$$\begin{cases} \ddot{q}_1 = 1 \\ \ddot{q}_2 = 0 \end{cases} \Rightarrow \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} = \left.\begin{bmatrix} \text{eqn}_1 \\ \text{eqn}_2 \end{bmatrix}\right|_{\substack{\ddot{q}_1=1, \\ \ddot{q}_2=0}}$$

$$\begin{bmatrix} M_{11} \\ M_{21} \end{bmatrix} = \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} + \left.\begin{bmatrix} \text{eqn}_1 \\ \text{eqn}_2 \end{bmatrix}\right|_{\substack{\ddot{q}_1=1, \\ \ddot{q}_2=0}}$$

$$\begin{cases} \ddot{q}_1 = 0 \\ \ddot{q}_2 = 1 \end{cases} \Rightarrow \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} = \left.\begin{bmatrix} \text{eqn}_1 \\ \text{eqn}_2 \end{bmatrix}\right|_{\substack{\ddot{q}_1=0, \\ \ddot{q}_2=1}}$$

$$\begin{bmatrix} M_{12} \\ M_{22} \end{bmatrix} = \begin{bmatrix} \text{RHS}_1 \\ \text{RHS}_2 \end{bmatrix} + \left.\begin{bmatrix} \text{eqn}_1 \\ \text{eqn}_2 \end{bmatrix}\right|_{\substack{\ddot{q}_1=0, \\ \ddot{q}_2=1}}$$

Newton-Euler： Double Pendulum

Symbolic Deviation in MATLAB

$$\begin{bmatrix}\mathrm{RHS}_1\\ \mathrm{RHS}_2\end{bmatrix} = -\left.\begin{bmatrix}\mathrm{eqn}_1\\ \mathrm{eqn}_2\end{bmatrix}\right|_{\substack{\ddot{q}_1=0,\\ \ddot{q}_2=0}}$$

$$\begin{bmatrix}M_{11}\\ M_{21}\end{bmatrix} = \begin{bmatrix}\mathrm{RHS}_1\\ \mathrm{RHS}_2\end{bmatrix} + \left.\begin{bmatrix}\mathrm{eqn}_1\\ \mathrm{eqn}_2\end{bmatrix}\right|_{\substack{\ddot{q}_1=1,\\ \ddot{q}_2=0}}$$

$$\begin{bmatrix}M_{12}\\ M_{22}\end{bmatrix} = \begin{bmatrix}\mathrm{RHS}_1\\ \mathrm{RHS}_2\end{bmatrix} + \left.\begin{bmatrix}\mathrm{eqn}_1\\ \mathrm{eqn}_2\end{bmatrix}\right|_{\substack{\ddot{q}_1=0,\\ \ddot{q}_2=1}}$$

```
RHS1 = -subs(eqn1,[ud1 ud2],[0 0])
RHS2 = -subs(eqn2,[ud1 ud2],[0 0])


M11 = subs(eqn1,[ud1 ud2],[1 0]) + RHS1
M21 = subs(eqn2,[ud1 ud2],[1 0]) + RHS2
M12 = subs(eqn1,[ud1 ud2],[0 1]) + RHS1
M22 = subs(eqn2,[ud1 ud2],[0 1]) + RHS2
%%%%%%% Final system [M] [alpha] = [RHS] %%%%%%%%%%%%%%%%%
M = [M11 M12; M21 M22];
RHS = [RHS1; RHS2];
```

Newton-Euler: Double Pendulum

Symbolic Deviation in MATLAB

$$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} = \begin{bmatrix} \mathrm{RHS}_1 \\ \mathrm{RHS}_2 \end{bmatrix}$$

When we solve for the angular accelerations, we will multiply the inverse of the mass matrix with $[RHS]$. We will not symbolically invert the mass matrix since doing so is computationally expensive compared to inverting it numerically.

In this simpler case, we can copy and paste the equations of motion in the simulation m-file. In more complicated problem with more DOF, the resulting matrices are much larger and it is easier to have MATLAB create a m-file that is already setup for simulation.

We can also derive the kinetic and potential energy equations.

```
%%%% Total energy (as a check on equations) %%%%
KE = simple(0.5*m1*(v_G1(1)*v_G1(1)+v_G1(2)*v_G1(2)) +
0.5*m2*(v_G2(1)*v_G2(1)+v_G2(2)*v_G2(2)) + 0.5*I1*u1*u1 + 0.5*I2*u2*u2)
PE = -m1*g*a*cos(q1) - m2*g*(l*cos(q1) + a*cos(q2))
```

Newton-Euler: Double Pendulum

Creating a Simulation in MATLAB

$$[\boldsymbol{M}][\ddot{\boldsymbol{q}}] = [\mathbf{RHS}]$$

The simulation will be setup into 3 parts: the driver which will integrate the equations of motion and create an animation, the right hand side of the equations of motion, and the energy equations.

First, we need to set up a file for the equations of motion. The m-file will contain a single function that takes in a time, set of angles and their derivatives, and parameters and returns the first and second derivatives of the angles. In this file, we need to put the M and RHS matrices and plug in the parameters, angles and derivatives. We can then find the angular accelerations by,

$$[\ddot{\boldsymbol{q}}] = [\boldsymbol{M}]^{-1}[\mathbf{RHS}]$$

Additionally, we need to set up a m-file for the energy. Similarly to the equation of motion file, this file takes in the time, positions and derivatives, and parameters and returns the kinetic and potential energy at that particular instance.

Newton-Euler： Double Pendulum

Creating a Simulation in MATLAB

1. Define the parameters, initial conditions, and a vector of times

```
%%%%%%%%% INITIALIZE PARAMETERS %%%%%%%
%Mechanical parameters.
m1 = 1; m2 = 1;                 % masses
I1 = 0.5; I2 = 0.5;             % inertias about cms
l = 1;                          % length of links
a = .5;                         % dist. from O to G1 and E to G2 (see figures)
g = 10;
% Initial conditions and other settings.
framespersec=50;                %if view is not speeded or slowed in dbpend_animate
T=10;                           %duration of animation in seconds
tspan=linspace(0, T, T*framespersec);
q1 = pi/2-0.1;                  %angle made by link1 with vertical
u1 = 0;                         %abslolute velocity of link1
q2 = pi ;                       %angle made by link2 with vertical
u2 = 0;                         %abslolute velocity of link2
z0=[q1 u1 q2 u2]';
```

Newton-Euler： Double Pendulum

Creating a Simulation in MATLAB

2. Set the absolute and relative error tolerances for the integration

```
options=odeset('abstol',1e-9,'reltol',1e-9);
```

3. Choose an ode function.

There are three we can choose from ode45, ode113, and ode23. We will choose ode113 because it is relatively faster than the others at higher precisions. Using ode113 and inputting the time span, initial conditions and parameters, we can integrate the equations of motion over the time period allotted.

```
%%%%%%% INTEGRATOR or ODE SOLVER %%%%%%%%
[t  z] = ode113('dbpend_rhs',tspan,z0,options, ...
m1, m2, l1, l2, l,  a,  g);
```

Ode113 will return the same time period and the resulting angles and angular velocities.

Newton-Euler： Double Pendulum

Creating a Simulation in MATLAB

4. Create an animation by repeatedly plotting the positions of the masses in the same plot over the specified time span.

```
figure(1)
for i=1:length(tspan)
    pause(.01)
    xm1=-l*sin(z(i,1));
    ym1=-l*cos(z(i,1));
    xm2=xm1-l*sin(z(i,3));
    ym2=ym1-l*cos(z(i,3));
    plot([0],[0],'ko','MarkerSize',3); %pivot point
    hold on
    plot([0 xm1],[0 ym1],'r','LineWidth',2);% first pendulum
    plot([xm1 xm2],[ym1 ym2],'b','LineWidth',2);% second pendulum
    axis([-2*l 2*l -2*l 2*l]);
    axis square
    hold off
end
```

Newton-Euler： Double Pendulum

Creating a Simulation in MATLAB

5. Plot the time histories of the angles and velocities.

```
figure(2)
plot(t,z(:,1),t,z(:,3));
xlabel('time (s)'); ylabel('position (rad)');
```

Checking the Simulation

One way to do this is to ensure that total energy is conserved. We can calculate the total energy for each time step and plot the difference between each step.

```
for i=1:length(t)
    [KE(i), PE(i)] = dbpend_energy(t(i),z(i,:),m1, m2, l1, l2, l, a, g);
end
TE = KE + PE;
TE_diff = diff(TE);
figure(3)
plot(t(1:end-1),TE_diff)
```

Dynamics

Lagrangian method (energy-based approach)

- 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

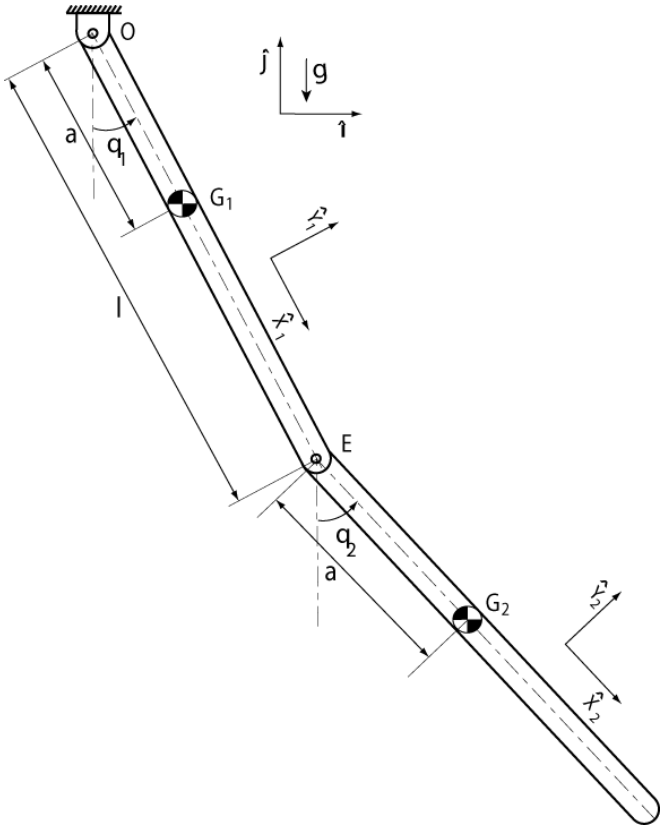
Newton-Euler method (balance of forces/torques)

- 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

Homework 15

Homework 15： http://bb.sustech.edu.cn

Due date: May 6



【Tasks】

1) Read the above document, download the Double Pendulum MATLAB File, run it in the MATLAB, and give a screenshot (print screen) of the result.
2) Read through the MATLAB code, and write code comments line by line.

http://ruina.tam.cornell.edu/research/topics/locomotion_and_robotics/ranger/ranger_paper/Reports/Ranger_Robot/control/simulator/doublependulum.html