ME/AER 676 Robot Modeling & Control
Spring 2026

Dynamics

Hasan A. Poonawala

Department of Mechanical Engineering
University of Kentucky

Email: hasan.poonawala@uky.edu
Web: https://www.engr.uky.edu/~hap

Introduction

Two related problems for robotics:

- Forward Dynamics: given τ, calculate $\ddot{q}$ (for simulation)
- Inverse Dynamics: given $\ddot{q}$, what τ produces it? (for control)

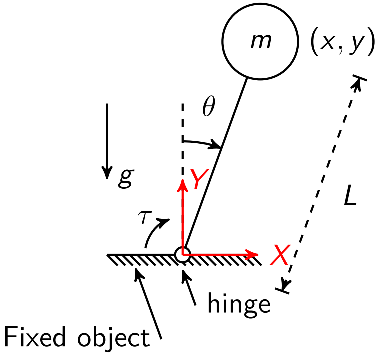
$\ddot{q}$: joint accelerations
τ: motor torques

Introduction

- Since our robot is a powered mechanism, we will obtain models for the motion that are generalizations of the Newton's Second Law $F = ma$
- a can be joint acceleration $\ddot{q}$ or task-variable accelerations $\ddot{\chi}$
- There are largely two approaches :
  - Apply Newton's law to every rigid body (need to know contact forces)
  - Use an energy-based formation, which can ignore contact forces

Simple Pendulum

- Mass suspended by rigid massless rod
- Upward position is $\theta = 0$ $(q \to \theta)$
- Force due to gravity



Standard Form

$$mL^2\ddot{\theta} = \tau + mgL\sin\theta$$

Applying Newton's laws to this system, we obtain

$$m\ddot{x} = R_x + f_x \tag{1}$$

$$m\ddot{y} = R_y + f_y - mg \tag{2}$$

The Euler equation boils down to zero net torque about mass m at (x, y):

$$yR_x - xR_y = 0 \tag{3}$$

We have three equations in four unknowns: $\ddot{x}$, $\ddot{y}$, R_x, and R_y. The constraint $x^2 + y^2 = L^2$ may be differentiated twice to obtain

$$x\ddot{x} + y\ddot{y} = -\dot{x}^2 - \dot{y}^2 \tag{4}$$

With this fourth equation, we may solve the linear system consisting of four equations in four unknowns $\ddot{x}$, $\ddot{y}$, R_x, and R_y:

$$\begin{array}{cccccl} m\ddot{x} & & -R_x & & = & f_x \\ & m\ddot{y} & & -R_y & = & f_y - mg \\ & & +yR_x & -xR_y & = & 0 \\ x\ddot{x} & +y\ddot{y} & & & = & -\dot{x}^2 - \dot{y}^2 \end{array} \tag{5}$$

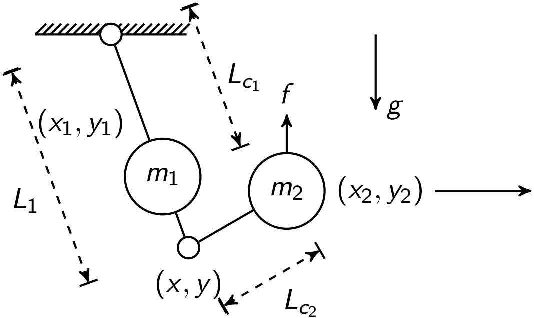
$$\begin{bmatrix} m & 0 & -1 & 0 \\ 0 & m & 0 & -1 \\ 0 & 0 & y & -x \\ x & y & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ R_x \\ R_y \end{bmatrix} = \begin{bmatrix} f_x \\ f_y - mg \\ 0 \\ -\dot{x}^2 - \dot{y}^2 \end{bmatrix} \tag{6}$$

$\downarrow$ Block form

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} a \\ R \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix} \tag{7}$$

$$a = A^{-1}f - A^{-1}B\left(D - CA^{-1}B\right)^{-1}\left(b - CA^{-1}f\right) \tag{8}$$

Double Pendulum



$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} a \\ R \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix}$$

(See notes)

Dynamics Problems

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} a \\ R \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix} \tag{9}$$

There are two standard problems we solve using (9).

1. **Forward Dynamics** (FD): Solve for a given f (simulate)
2. **Inverse Dynamics** (ID): Solve for f given a (control)

Dynamics Problems

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} a \\ R \end{bmatrix} = \begin{bmatrix} f \\ b \end{bmatrix} \tag{9}$$

There are two standard problems we solve using (9).

1. **Forward Dynamics** (FD): Solve for a given f (simulate)
2. **Inverse Dynamics** (ID): Solve for f given a (control)

Forward Dynamics: use the Articulated Body Algorithm

Inverse Dynamics: use the Recursive Newton-Euler Algorithm

Recursive Newton-Euler Methods

- Apply Newton-Euler equations to a link in the frame attached to it's center of mass (easy to encode)
- Forward pass: Since the frames are not inertial, propagate the coriolis accelerations from the base to the end-effector
- Backward pass: propagate the torques that achieve the accelerations, and contact forces they imply, from end-effector frame to the base

Recursive Newton-Euler Methods

- Most simulators implement the RNE algorithm also for simulation
- $\mathcal{O}(n)$ complexity, which is fast
- Method generalizes to all kinematic trees
- Closed chains / parallel mechansisms can be handled by additional steps
- Screw-theory-based approaches may be better for parallel mechanisms

Euler-Lagrangian Models

- Derive's equations from the total energy of the system
- Avoids needing to account for internal joint forces
- Difficult to automate, at least so far
- Deep structural insights into robot dynamics

Euler-Lagrangian Models

- Define the minimal coordinates q of the system
- Define the Lagrangian $\mathcal{L}(q, \dot{q})$ = Kinetic Energy - Potential Energy
- For each DoF q_i:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \sum \text{ Generalized Forces}$$

- The robot equations are

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau + \tau_{friction} + \tau_e,$$

Euler-Lagrangian Models: Simple Pendulum

$$\begin{aligned}\mathcal{L}(q,\dot{q}) &= \text{K.E - P.E} \\ &= \frac{1}{2}m(L\dot{\theta})^2 - mgL\cos\theta\end{aligned}$$

Apply the Euler-Lagrange Equation to $q = \theta$:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \sum \text{ Generalized Forces}$$

$$\implies \frac{d}{dt}(mL^2\dot{\theta}) - mgL\sin\theta = \tau$$

$$\implies mL^2\ddot{\theta} = mgL\sin\theta + \tau$$

$$D(q) = mL^2 \quad C(q,\dot{q}) = 0 \quad G(q) = -mgL\sin\theta$$

Properties

- The matrix $\dot{D}(Q) - 2C$ is skew symmetric.
- Bounded Inertia: For a system with revolute joints, there exist λ_m and λ_M such that

$$\lambda_m I_{n\times n} \leq D(q) \leq \lambda_M I_{n\times n} < \infty \qquad (10)$$

- Linearity in Parameters: We can derive a function $Y(q, \dot{q}, \ddot{q})$ and parameter set θ such that

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = Y(q, \dot{q}, \ddot{q})\theta \qquad (11)$$

Dynamics Including Actuators

- For torque-controlled robots, use

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau + \tau_{friction} + \tau_e,$$

- When torque is created by voltage-controlled geared motors, we often use

$$\underbrace{M(q)}_{+\text{motor inertia}} \ddot{q} + C(q,\dot{q})\dot{q} + \underbrace{B\dot{q}}_{+\text{motor friction}} + G(q) = \underbrace{u}_{\text{voltage}} + \tau_e,$$

Practice

We sometimes *design* algorithms using the Euler-Lagrange equations, then *implement* them in software by exploiting RNEA functions:

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

Question

How would you use function RNEA to compute $D(q)$, $C(q, \dot{q})$ or $G(q)$ in

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \tau + \tau_{friction} + \tau_e?$$

(assume $\tau_e = \tau_{friction} = 0$)