Robotics

Dynamics

1D point mass, damping & oscillation, PID, dynamics of mechanical systems, Euler-Lagrange equation, Newton-Euler recursion, general robot dynamics, joint space control, reference trajectory following, operational space control

Marc Toussaint
U Stuttgart

Kinematic

instantly change joint velocities $\dot{q}$:

$$\delta q_t \overset{!}{=} J^\sharp \, (y^* - \phi(q_t))$$

accounts for kinematic coupling of joints but **ignores inertia, forces, torques**

gears, **stiff**, all of industrial robots

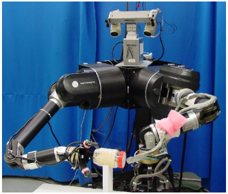
Dynamic

instantly change joint torques u:

$$u \overset{!}{=} ?$$

accounts for dynamic coupling of joints and full Newtonian physics

future robots, **compliant**, few research robots



When velocities cannot be changed/set arbitrarily

- Examples:
  – An air plane flying: You cannot command it to hold still in the air, or to move straight up.
  – A car: you cannot command it to move side-wards.
  – Your arm: you cannot command it to throw a ball with arbitrary speed (force limits).
  – A *torque controlled* robot: You cannot command it to instantly change velocity (infinite acceleration/torque).

- What all examples have in comment:
  – One can set **controls** u_t (air plane's control stick, car's steering wheel, your muscles activations, torque/voltage/current send to a robot's motors)
  – But these controls only indirectly influence the **dynamics of state**, $x_{t+1} = f(x_t, u_t)$

Dynamics

- The dynamics of a system describes how the controls u_t influence the change-of-state of the system

$$x_{t+1} = f(x_t, u_t)$$

– The notation x_t refers to the *dynamic state* of the system: e.g., joint positions *and velocities* $x_t = (q_t, \dot{q}_t)$.
– f is an arbitrary function, often smooth

Outline

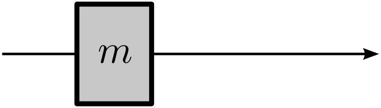
- We start by discussing a **1D point mass** for 3 reasons:
  - The most basic force-controlled system with inertia
  - We can introduce and understand **PID control**
  - The behavior of a point mass under PID control is a *reference* that we can also follow with arbitrary dynamic robots (if the dynamics are known)

- We discuss computing the dynamics of general robotic systems
  - Euler-Lagrange equations
  - Euler-Newton method
- We derive the dynamic equivalent of inverse kinematics:
  - operational space control

PID and a 1D point mass

The dynamics of a 1D point mass

- Start with simplest possible example: 1D point mass
  (no gravity, no friction, just a single mass)



- The state $x(t) = (q(t), \dot{q}(t))$ is described by:
  – position $q(t) \in \mathbb{R}$
  – velocity $\dot{q}(t) \in \mathbb{R}$

- The controls $u(t)$ is the force we apply on the mass point

- The system dynamics is:

$$\ddot{q}(t) = u(t)/m$$

1D point mass – proportional feedback

- Assume current position is q.
  The goal is to move it to the position q^*.

  What can we do?

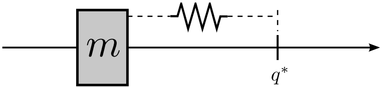
1D point mass – proportional feedback

- Assume current position is q.
  The goal is to move it to the position q^*.

  What can we do?

- **Idea 1:**
  "Always pull the mass towards the goal q^*:"

$$u = K_p\,(q^* - q)$$



1D point mass – proportional feedback

- What's the effect of this control law?

$$m\,\ddot{q} = u = K_p\,(q^* - q)$$

$q = q(t)$ is a function of time, this is a second order differential equation

1D point mass – proportional feedback

- What's the effect of this control law?

$$m\,\ddot{q} = u = K_p\,(q^* - q)$$

  $q = q(t)$ is a function of time, this is a second order differential equation

- Solution: assume $q(t) = a + be^{\omega t}$
  (a "non-imaginary" alternative would be $q(t) = a + b\,\epsilon^{-\lambda t}\,\cos(\omega t)$)

1D point mass – proportional feedback

- What's the effect of this control law?

$$m\,\ddot{q} = u = K_p\,(q^* - q)$$

$q = q(t)$ is a function of time, this is a second order differential equation

- Solution: assume $q(t) = a + be^{\omega t}$
(a "non-imaginary" alternative would be $q(t) = a + b\,\epsilon^{-\lambda t}\,\cos(\omega t)$)

$$m\,b\,\omega^2\,e^{\omega t} = K_p\,q^* - K_p\,a - K_p\,b\,e^{\omega t}$$

$$(m\,b\,\omega^2 + K_p\,b)\,e^{\omega t} = K_p\,(q^* - a)$$

$$\Rightarrow (m\,b\,\omega^2 + K_p\,b) = 0\ \wedge\ (q^* - a) = 0$$

$$\Rightarrow\ \omega = i\sqrt{K_p/m}$$

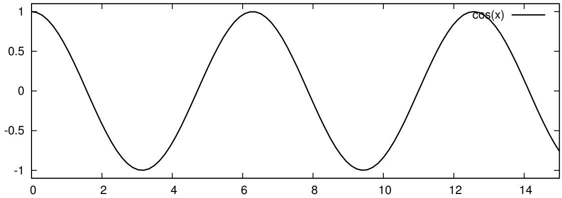
$$q(t) = q^* + b\,e^{i\sqrt{K_p/m}\ t}$$

This is an oscillation around q^* with amplitude $b = q(0) - q^*$ and frequency $\sqrt{K_p/m}$!

1D point mass – proportional feedback

$$m\,\ddot q = u = K_p\,(q^* - q)$$
$$q(t) = q^* + b\,e^{i\sqrt{K_p/m}\,t}$$

Oscillation around q^* with amplitude $b = q(0) - q^*$ and frequency $\sqrt{K_p/m}$

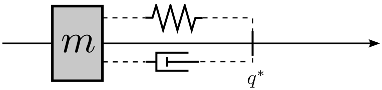


1D point mass – derivative feedback

- **Idea 2**
  "Pull less, when we're heading the right direction already:"
  "Damp the system:"

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q})$$

$\dot{q}^*$ is a desired goal velocity
For simplicity we set $\dot{q}^* = 0$ in the following.



1D point mass – derivative feedback

- What's the effect of this control law?

$$m\ddot{q} = u = K_p(q^* - q) + K_d(0 - \dot{q})$$

- Solution: again assume $q(t) = a + be^{\omega t}$

$$\begin{aligned}
&m\ b\ \omega^2\ e^{\omega t} = K_p\ q^* - K_p\ a - K_p\ b\ e^{\omega t} - K_d\ b\ \omega e^{\omega t} \\
&(m\ b\ \omega^2 + K_d\ b\ \omega + K_p\ b)\ e^{\omega t} = K_p\ (q^* - a) \\
&\Rightarrow (m\ \omega^2 + K_d\ \omega + K_p) = 0\ \wedge\ (q^* - a) = 0 \\
&\Rightarrow \omega = \frac{-K_d \pm \sqrt{K_d^2 - 4mK_p}}{2m} \\
&q(t) = q^* + b\ e^{\omega\ t}
\end{aligned}$$

The term $-\frac{K_d}{2m}$ in ω is real $\leftrightarrow$ exponential decay (damping)

1D point mass – derivative feedback

$$q(t) = q^* + b\, e^{\omega\, t}\,, \quad \omega = \frac{-K_d \pm \sqrt{K_d^2 - 4mK_p}}{2m}$$

- Effect of the second term $\sqrt{K_d^2 - 4mK_p}/2m$ in ω:

$K_d^2 < 4mK_p \quad \Rightarrow$ ω has imaginary part
oscillating with frequency $\sqrt{K_p/m - K_d^2/4m^2}$
$q(t) = q^* + be^{-K_d/2m\ t}\ e^{i\sqrt{K_p/m - K_d^2/4m^2}\ t}$

$K_d^2 > 4mK_p \quad \Rightarrow$ ω real
strongly damped

$K_d^2 = 4mK_p \quad \Rightarrow$ second term zero
only exponential decay

1D point mass – derivative feedback

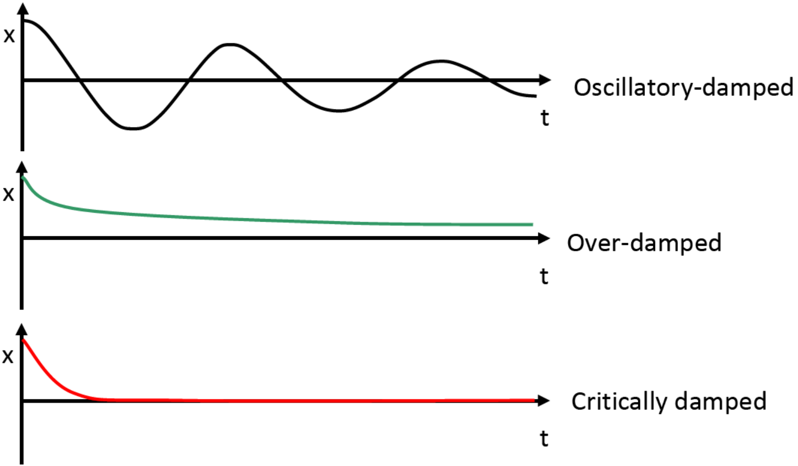


illustration from O. Brock's lecture

1D point mass – derivative feedback

Alternative parameterization:

Instead of the *gains* K_p and K_d it is sometimes more intuitive to set the

- wave length $\lambda = \frac{1}{\omega_0} = \frac{1}{\sqrt{K_p/m}}$, $\quad K_p = m/\lambda^2$
- damping ratio $\xi = \frac{K_d}{\sqrt{4mK_p}} = \frac{\lambda K_d}{2m}$, $\quad K_d = 2m\xi/\lambda$

$\xi > 1$: over-damped
$\xi = 1$: critically dampled
$\xi < 1$: oscillatory-damped

$$q(t) = q^* + be^{-\xi\ t/\lambda}\ e^{i\sqrt{1-\xi^2}\ t/\lambda}$$

1D point mass – integral feedback

- **Idea 3**
  "Pull if the position error accumulated large in the past:"

$$u = K_p(q^* - q) + K_d(\dot q^* - \dot q) + K_i \int_{s=0}^{t} (q^*(s) - q(s))\, ds$$

- This is not a linear ODE w.r.t. $x = (q, \dot q)$.
  However, when we extend the state to $x = (q, \dot q, e)$ we have the ODE

$$\begin{aligned}
\dot q &= \dot q \\
\ddot q &= u/m = K_p/m(q^* - q) + K_d/m(\dot q^* - \dot q) + K_i/m\, e \\
\dot e &= q^* - q
\end{aligned}$$

  (no explicit discussion here)

1D point mass – PID control

$$u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q}) + K_i \int_{s=0}^{t} (q^* - q(s))\, ds$$

- **PID control**
  - Proportional Control ("Position Control")
    $f \propto K_p(q^* - q)$
  - Derivative Control ("Damping")
    $f \propto K_d(\dot{q}^* - \dot{q})$ ($\dot{x}^* = 0 \to$ damping)
  - Integral Control ("Steady State Error")
    $f \propto K_i \int_{s=0}^{t} (q^*(s) - q(s))\, ds$

Controlling a 1D point mass – lessons learnt

- Proportional and derivative feedback (PD control) are like adding a spring and damper to the point mass

- PD control is a *linear control law*

$$(q, \dot{q}) \mapsto u = K_p(q^* - q) + K_d(\dot{q}^* - \dot{q})$$

  (linear in the *dynamic system state* $x = (q, \dot{q})$)

- With such linear control laws we can design approach trajectories (by tuning the gains)
  – but no optimality principle behind such motions

Dynamics of mechanical systems

Two ways to derive dynamics equations for mechanical systems

- The Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot q} - \frac{\partial L}{\partial q} = u$$

Used when you want to derive analytic equations of motion (“on paper”)

- The Newton-Euler recursion (and related algorithms)

$$f_i = m\dot v_i \;, \quad u_i = I_i \dot w + w \times I w$$

Algorithms that “propagate” forces through a kinematic tree and numerically compute the *inverse* dynamics $u = \text{NE}(q, \dot q, \ddot q)$ or *forward* dynamics $\ddot q = f(q, \dot q, u)$.

The Euler-Lagrange equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u$$

- $L(q, \dot{q})$ is called **Lagrangian** and defined as

  $$L = T - U$$

  where T=kinetic energy and U=potential energy.

- q is called generalized coordinate – any coordinates such that $(q, \dot{q})$ describes the state of the system. Joint angles in our case.

- u are external forces

The Euler-Lagrange equation

- How is this typically done?
- **First**, describe the *kinematics and Jacobians* for every link i:

$$(q, \dot q) \mapsto \{T_{W\to i}(q), v_i, w_i\}$$

Recall $T_{W\to i}(q) = T_{W\to A}\ T_{A\to A'}(q)\ T_{A'\to B}\ T_{B\to B'}(q)\ \cdots$
Further, we know that a link's velocity $v_i = J_i \dot q$ can be described via its position Jacobian.
Similarly we can describe the link's *angular velocity* $w_i = J_i^w \dot q$ as linear in $\dot q$.

- **Second**, formulate the kinetic energy

$$T = \sum_i \frac{1}{2} m_i v_i^2 + \frac{1}{2} w_i^\top I_i w_i = \sum_i \frac{1}{2} \dot q^\top M_i \dot q\ , \quad M_i = \begin{pmatrix} J_i \\ J_i^w \end{pmatrix}^\top \begin{pmatrix} m_i \mathbf{I}_3 & 0 \\ 0 & I_i \end{pmatrix} \begin{pmatrix} J_i \\ J_i^w \end{pmatrix}$$

where $I_i = R_i \bar I_i R_i^\top$ and $\bar I_i$ the inertia tensor in link coordinates

- **Third**, formulate the potential energies (typically independent of $\dot q$)

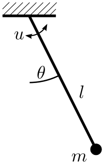
$$U = g m_i \text{height}(i)$$

- **Fourth**, compute the partial derivatives analytically to get something like

$$\underbrace{u}_{\text{control}} = \frac{d}{dt}\frac{\partial L}{\partial \dot q} - \frac{\partial L}{\partial q} = \underbrace{M}_{\text{inertia}} \ddot q + \underbrace{\dot M \dot q - \frac{\partial T}{\partial q}}_{\text{Coriolis}} + \underbrace{\frac{\partial U}{\partial q}}_{\text{gravity}}$$

which relates accelerations $\ddot q$ to the forces

Example: A pendulum



- Generalized coordinates: angle $q = (\theta)$
- Kinematics:
  - velocity of the mass: $v = (l\dot\theta\cos\theta, 0, l\dot\theta\sin\theta)$
  - angular velocity of the mass: $w = (0, -\dot\theta, 0)$
- Energies:

$$T = \frac{1}{2}mv^2 + \frac{1}{2}w^\top I w = \frac{1}{2}(ml^2 + I_2)\dot\theta^2 \;, \quad U = -mgl\cos\theta$$

- Euler-Lagrange equation:

$$\begin{aligned} u &= \frac{d}{dt}\frac{\partial L}{\partial \dot q} - \frac{\partial L}{\partial q} \\ &= \frac{d}{dt}(ml^2 + I_2)\dot\theta + mgl\sin\theta = (ml^2 + I_2)\ddot\theta + mgl\sin\theta \end{aligned}$$

Newton-Euler recursion

- An algorithms that compute the *inverse dynamics*

$$u = \mathsf{NE}(q, \dot q, \ddot q^*)$$

  by recursively computing force balance at each joint:
  - **Newton's equation** expresses the force acting at the center of mass for an accelerated body:

$$f_i = m \dot v_i$$

  - **Euler's equation** expresses the torque (=control!) acting on a rigid body given an angular velocity and angular acceleration:

$$u_i = I_i \dot w + w \times I w$$

- **Forward recursion:** ($\approx$ kinematics)
  Compute (angular) velocities (v_i, w_i) *and* accelerations $(\dot v_i, \dot w_i)$ for every link (via forward propagation; see geometry notes for details)

- **Backward recursion:**
  For the leaf links, we now know the desired accelerations q^* and can compute the necessary joint torques. Recurse backward.

Numeric algorithms for forward and inverse dynamics

- **Newton-Euler recursion**: very fast ($O(n)$) method to compute *inverse* dynamics

$$u = \mathsf{NE}(q, \dot q, \ddot q^*)$$

  Note that we can use this algorithm to also compute
  - gravity terms: $u = \mathsf{NE}(q, 0, 0) = G(q)$
  - Coriolis terms: $u = \mathsf{NE}(q, \dot q, 0) = C(q, \dot q)\ \dot q$
  - column of Intertia matrix: $u = \mathsf{NE}(q, 0, e_i) = M(q)\ e_i$

- **Articulated-Body-Dynamics**: fast method ($O(n)$) to compute *forward* dynamics $\ddot q = f(q, \dot q, u)$

Some last practical comments

- `[demo]`
- Use *energy conservation* to measure dynamic of physical simulation
- Physical simulation engines (developed for games):
  - ODE (Open Dynamics Engine)
  - Bullet (originally focussed on collision only)
  - Physx (Nvidia)

  Differences of these engines to Lagrange, NE or ABD:
  - Game engine can model much more: Contacts, tissues, particles, fog, etc
  - (The way they model contacts looks ok but is somewhat fictional)
  - On kinematic trees, NE or ABD are much more precise than game engines
  - Game engines do not provide *inverse* dynamics, $u = \text{NE}(q, \dot{q}, \ddot{q})$

- Proper modelling of contacts is really really hard

Dynamic control of a robot

- We previously learnt the effect of PID control on a 1D point mass

- Robots are not a 1D point mass
  - Neither is each joint a 1D point mass
  - Applying separate PD control in each joint neglects force coupling (Poor solution: Apply very high gains separately in each joint $\leftrightarrow$ make joints stiff, as with gears.)

- However, knowing the robot dynamics we can transfer our understanding of PID control of a point mass to general systems

General robot dynamics

- Let $(q, \dot q)$ be the dynamic state and $u \in \mathbb{R}^n$ the controls (typically joint torques in each motor) of a robot
- Robot dynamics can generally be written as:

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

$M(q) \in \mathbb{R}^{n \times n}$	is positive definite intertia matrix (can be inverted $\to$ 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

(cf. to the Euler-Lagrange equation on slide 22)

- We often write more compactly:

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

Controlling a general robot

- From now on we jsut assume that we have algorithms to efficiently compute $M(q)$ and $F(q, \dot q)$ for any $(q, \dot q)$
- **Inverse dynamics:** If we know the desired $\ddot q^*$ for each joint,

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

  gives the necessary torques
- **Forward dynamics:** If we know which torques u we apply, use

$$\ddot q^* = M(q)^{\text{-1}} (u - F(q, \dot q))$$

  to simulate the dynamics of the system (e.g., using Runge-Kutta)

Controlling a general robot – joint space approach

- 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

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

  However, tiny errors in acceleration will accumulate greatly over time! This is Instable!!

Controlling a general robot – joint space approach

- 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

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

  However, tiny errors in acceleration will accumulate greatly over time! This is Instable!!

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

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

  This is a standard and very convenient heuristic to track a reference trajectory when the robot dynamics are known: *All joints will exactly behave like a 1D point particle around the reference trajectory!*

Controlling a robot – operational space approach

- Recall the inverse kinematics problem:
  - We know the desired step δy^* (or velocity $\dot y^*$) of the *endeffector*.
  - Which step δq (or velocities $\dot q$) should we make in the joints?

- Equivalent dynamic problem:
  - We know how the desired acceleration $\ddot y^*$ of the *endeffector*.
  - What controls u should we apply?

Operational space control

- Inverse kinematics:

$$q^* = \underset{q}{\text{argmin}} \, \|\phi(q) - y^*\|_C^2 + \|q - q_0\|_W^2$$

- Operational space control (one might call it “Inverse task space dynamics”):

$$u^* = \underset{u}{\text{argmin}} \, \|\ddot{\phi}(q) - \ddot{y}^*\|_C^2 + \|u\|_H^2$$

Operational space control

- We can derive the optimum perfectly analogous to inverse kinematics
  We identify the minimum of a locally squared potential, using the local linearization (and approx. $\ddot J = 0$)

  $$\ddot\phi(q) = \frac{d}{dt}\dot\phi(q) \approx \frac{d}{dt}(J\dot q + \dot J q) \approx J\ddot q + 2\dot J\dot q = JM^{\text{-}1}(u - F) + 2\dot J\dot q$$

  We get

  $$u^* = T^\sharp(\ddot y^* - 2\dot J\dot q + TF)$$
  $$\text{with } T = JM^{\text{-}1} \;, \quad T^\sharp = (T^\top CT + H)^{\text{-}1}T^\top C$$

  ($C \to \infty \;\Rightarrow\; T^\sharp = H^{\text{-}1}T^\top(TH^{\text{-}1}T^\top)^{\text{-}1}$)

Controlling a robot – operational space approach

- Where could we get the desired $\ddot{y}^*$ from?
  - **Reference trajectory** $y^{\text{ref}}_{0:T}$ in operational space
  - **PD-like behavior** in each operational space:

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

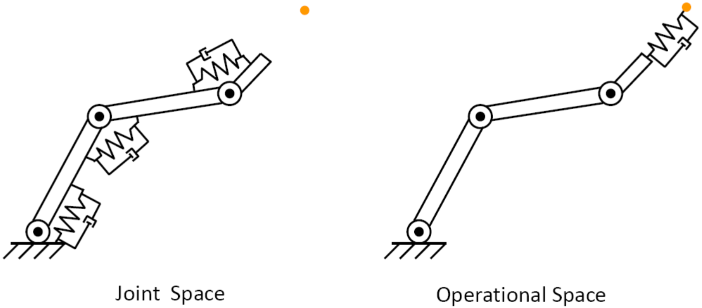


illustration from O. Brock's lecture

- Operational space control: *Let the system behave as if we could directly "apply a 1D point mass behavior" to the endeffector*

Multiple tasks

- Recall trick last time: we defined a "big kinematic map" $\Phi(q)$ such that

$$q^* = \underset{q}{\operatorname{argmin}} \|q - q_0\|_W^2 + \|\Phi(q)\|^2$$

- Works analogously in the dynamic case:

$$u^* = \underset{u}{\operatorname{argmin}} \|u\|_H^2 + \|\Phi(q)\|^2$$