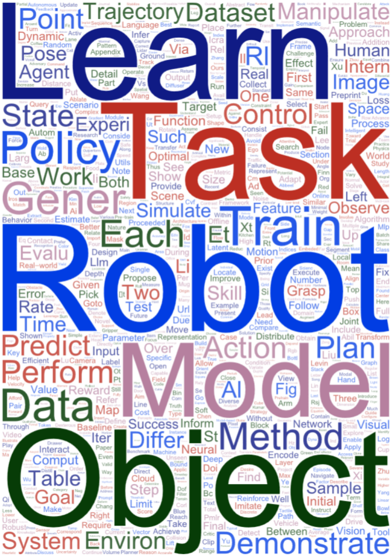


Robot Learning

Robotics Essentials

Marc Toussaint
Technical University of Berlin
Summer 2024

Robotics Essentials Outline

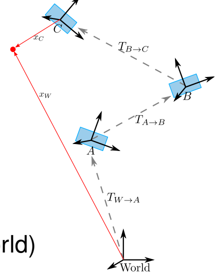
- A robot is an articulated multi-body system: kinematics & dynamics
- Standard Control: IK, path finding & traj. opt, PD & MPC

Robot as Articulated Multibody System

- A robot is a multibody system. Each body
  - has a pose $x_i \in \mathsf{SE}(3)$
  - has inertia (m_i, I_i) with mass $m_i \in \mathbb{R}$ and inertia tensor $I_i \in \mathbb{R}^{3\times 3}$ sym.pos.def.
  - has a shape s_i (formally: any representation that defines a pairwise signed-distance $d(s_i, s_j)$)

[Useful: “multibody system” on Wikipedia]

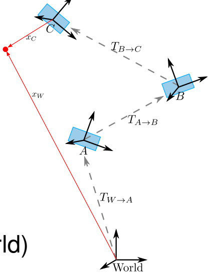
Robot as Articulated Multibody System



- **Tree structure:**
  - Every body is linked to a parent body or the world
  - We have relative transformations $Q_i \in \text{SE}(3)$ from parent (or world)

[If not tree-structured, we only represent a tree and use additional constraints to describe loops $\to$ more involved, but doable]

Robot as Articulated Multibody System

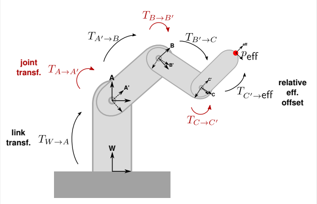


- **Tree structure:**
  - Every body is linked to a parent body or the world
  - We have relative transformations $Q_i \in \mathrm{SE}(3)$ from parent (or world)

[If not tree-structured, we only represent a tree and use additional constraints to describe loops $\to$ more involved, but doable]

- **Articulated Degrees of Freedom (dofs):**
  - Some of the relative transformations Q_i may have articulated (=motorized) **dofs** q so that $Q_i(q)$

    [Different types of joints (hinge, prismatic, universal, ball) have different # dofs and different mapping from dofs $q \mapsto Q_i(q)$]
  - We stack all dofs of all relative transformations into a single **joint vector** $q \in \mathbb{R}^n$



$x \in \mathsf{SE}(3)^m$: all body poses, $\qquad q \in \mathbb{R}^n$: joint vector

– Forward kinematics: $q \mapsto x,\ \dot q \mapsto \dot x,\ \ddot q \mapsto \ddot x$
– Forward dynamics: $u \mapsto \ddot q$, inverse dynamics: $\ddot q \mapsto u$ $\quad$ ($u \in \mathbb{R}^n$: joint torques)

Forward Kinematics $q \mapsto x$

- Given q, what is the pose of any body i?

$$q \;\mapsto\; \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} = \phi(q) \quad \in \mathsf{SE}(3)^m$$

  - *Algorithm:* First determine all rel. trans. $Q_i(q)$, then forward chain them
  - Often one cares only about position/orientation of one particular body x_i: the **“endeffector”**

Forward Velocities & Jacobian $\dot q \mapsto \dot x$

- Given $\dot q$, what is the linear and angular velocity (v_i, w_i) of any body i?

$$\dot q \mapsto \begin{pmatrix} v_1, w_1 \\ v_2, w_2 \\ \vdots \\ v_m, w_m \end{pmatrix} = J(q)\ \dot q \quad \in \mathbb{R}^{m\times 6}$$

  – with **Jacobian** $J(q) = \partial_q \phi(q) \ \in \mathbb{R}^{m\times 6\times n}$.

    [Since, ϕ is SE(3)-valued, the Jacobian actually has output in its tangent space $se(3) \equiv \mathbb{R}^6$. In practise, code typically provides separate positional Jacobian $J^{\text{pos}} \in \mathbb{R}^{m\times 3\times n}$ and angular Jacobian $J^{\text{ang}} \in \mathbb{R}^{m\times 3\times n}$.]

Forward Velocities & Jacobian $\dot q \mapsto \dot x$

- Given $\dot q$, what is the linear and angular velocity (v_i, w_i) of any body i?

$$\dot q \mapsto \begin{pmatrix} v_1, w_1 \\ v_2, w_2 \\ \vdots \\ v_m, w_m \end{pmatrix} = J(q)\ \dot q \quad \in \mathbb{R}^{m\times 6}$$

  - with **Jacobian** $J(q) = \partial_q \phi(q) \ \in \mathbb{R}^{m\times 6\times n}$.

    [Since, ϕ is SE(3)-valued, the Jacobian actually has output in its tangent space $se(3) \equiv \mathbb{R}^6$. In practise, code typically provides separate positional Jacobian $J^{\text{pos}} \in \mathbb{R}^{m\times 3\times n}$ and angular Jacobian $J^{\text{ang}} \in \mathbb{R}^{m\times 3\times n}$.]
  - Since we know how to compute $\phi(q)$, we can think of $J(q)$ as the "autodiff" of it
  - However, positional/angular Jacobians are really very easy to provide without expensive autodiff

    [In practise, one only needs to figure out the J^{pos}, J^{ang} for a rotational and translational joint – all others follow from this.]

Forward Accelerations $\ddot{q} \mapsto \ddot{x}$

- Given $\ddot{q}$, what is the linear and angular acceleration $(\dot{v}_i, \dot{w}_i)$ of any body i?

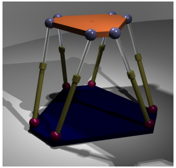
$$\ddot{x} = \dot{J}(q)\ \dot{q} + J(q)\ \ddot{q}\ \approx\ J(q)\ \ddot{q}$$

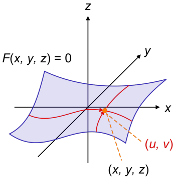
  – One typically approximates $\dot{J} = 0$

The word “kinematics”

[in parts from Wikipedia]

– Mathematical description of possible motions of a (constrainted/multibody) system/mechanism *without considering the forces*
– “geometry of [possible] motions”
– Formally: Describe the space (manifold) of possible system poses and all possible paths in that space
– Read **generalized coordinates** on wikipedia: Understanding motion in terms of coordinates and (non-)holonomic constraints:





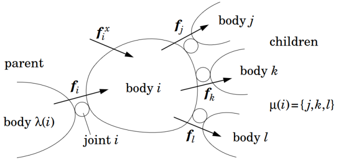
Inverse dynamics $\ddot{q} \mapsto u$

- Given $\ddot{q}$, what joint torques u do we need to generate this $\ddot{q}$ (accounting for gravity)?

Inverse dynamics $\ddot{q} \mapsto u$

- Given $\ddot{q}$, what joint torques u do we need to generate this $\ddot{q}$ (accounting for gravity)?

- Coupled Newton-Euler equations: For each body:



from Featherstone'14

$$F_i = \begin{pmatrix} f_i \\ \tau_i \end{pmatrix} = \begin{pmatrix} m_i \dot{v}_i \\ I_i \dot{w}_i + w_i \times I_i w_i \end{pmatrix}$$

$$F_i^{\text{back}} = F_i - F_i^{\text{ext}} + \sum_{j=\text{child(i)}} F_j^{\text{back}} \;, \qquad u_i = h_i^\top F_i^{\text{back}}$$

[where F_i^{ext} are external (e.g. gravity) forces; and F_i^{back} is the force "send back through the joint to the parent of i"; h_i is the joint axis (picking up the torque)]

[Can also be written as linear equation system between $\ddot{q}$, F, F^{back}, and u (with sparse matrices only) – and solved/inverted in $O(m)$.]

solved!   We can accelerate the thing as we like solved!   We can accelerate the thing as we like

the rest is planning: How should I accelerate to reach some future goals?

Standard Template: Waypoint + Reference Motion + Controller

Standard Template: Waypoint + Reference Motion + Controller

- Standard problem setting: Control motors, so that at $t = T$ seconds the endeffector x_i is at desired position $y^* \in \mathbb{R}^3$, i.e., $\phi(q_{t=T}) = y^*$

Standard Template: Waypoint + Reference Motion + Controller

- Standard problem setting: Control motors, so that at $t = T$ seconds the endeffector x_i is at desired position $y^* \in \mathbb{R}^3$, i.e., $\phi(q_{t=T}) = y^*$
- Problem decomposition:
  - Find a final robot pose q_T that fulfills constraint $\phi(q_{t=T}) = y^*$ – **inverse kinematics**
  - Find a nice *reference* motion from current robot pose q_0 to q_T – **path finding, trajectory optimization, or trivial interpolation/PD**
  - Find a control policy $\pi : x_t \mapsto u_t$ that reactively sends motor commands to follow the reference motion – **inverse dynamics, PD control, Riccati**

[You could think of this as three different time scales: rough future waypoint(s)/goal(s), continuous motion to next waypoint, short-term controls.]

[There are other ways to approach this: You could remove step (1) and shift that issue into (2), or remove (1 2) and shift all issues into (3) - morphing this into other approaches. E.g. directly defining a desired force/acceleration behavior in “task space” (=operational space control).]

[continuous replanning/re-estimation can also make (1) and (2) reactive.]

Inverse Kinematics

- Find q to fulfill $\phi(q) = y^*$ for differentiable fwd kinematics ϕ.

$$\min_{q \in \mathbb{R}^n} \|q - q_0\|^2 \text{ s.t. } \phi(q) = y^*$$

$$\text{or} \quad \min_{q \in \mathbb{R}^n} \|q - q_0\|^2 + \mu \|\phi(q) - y^*\|^2 \quad \text{for large } \mu$$

- Solution for linearized ϕ:

$$q^* = q_0 + J^\top (J J^\top + \tfrac{1}{\mu} \mathbf{I})^{-1} (y^* - \phi(q_0))$$

Python Package: `https://marctoussaint.github.io/robotic/`

Path Finding & Trajectory Optimization

- Given current q_0 and future q^*, find a collision free **path**
  - Wolfgang Hönig's & Andreas Orthey's lecture
  - RRTs, PRMs, under constraints (kinodynamic)

Path Finding & Trajectory Optimization

- Given current q_0 and future q^*, find a collision free **path**
  - Wolfgang Hönig's & Andreas Orthey's lecture
  - RRTs, PRMs, under constraints (kinodynamic)

- **Trajectory** opimization
  - Time continuous formulation:

$$\min_{q(t)} \int_0^T c(q(t), \dot q(t), \ddot q(t)) \, dt \quad \text{s.t.} \quad q(0) = q_0,\ q(T) = q^*, \dot q(0) = \dot q(T) = 0 \ , \forall_{t \in [0,T]} : \bar\phi(q(t), \dot q(t), \ddot q(t)) \le 0 \ .$$

  - Time-discretized, assuming k-order Markov coupling terms (KOMO):

    A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference: *Marc Toussaint*. Springer 2017

Control around a Reference

- Use **Inverse Dynamics** directly
  - We have $\ddot{q}^*(t) \to$ map it to controls u directly
  - But what if you're off the reference a bit? *How to steer back?*

Control around a Reference

- Use **Inverse Dynamics** directly
  - We have $\ddot{q}^*(t) \to$ map it to controls u directly
  - But what if you're off the reference a bit? *How to steer back?*
- Use **PD law** to accelerate back to reference:
  - Define a PD law $\ddot{q}^{\text{desired}} = \ddot{q}^*(t) + k_p(q^*(t) - q) + k_d(\dot{q}^*(t) - \dot{q})$ with desired PD behavior back to reference
  - Then use Inv dynamics $\ddot{q}^{\text{desired}} \mapsto u$
  - (Also ok, but needs severe tuning: directly define a PD controller $\ddot{u} = M\ddot{q}^*(t) + K_p(q^*(t) - q) + K_d(\dot{q}^*(t) - \dot{q})$.)

Control around a Reference

- Use **Inverse Dynamics** directly
  - We have $\ddot q^*(t) \to$ map it to controls u directly
  - But what if you're off the reference a bit? *How to steer back?*
- Use **PD law** to accelerate back to reference:
  - Define a PD law $\ddot q^{\text{desired}} = \ddot q^*(t) + k_p(q^*(t) - q) + k_d(\dot q^*(t) - \dot q)$ with desired PD behavior back to reference
  - Then use Inv dynamics $\ddot q^{\text{desired}} \mapsto u$
  - (Also ok, but needs severe tuning: directly define a PD controller $\ddot u = M\ddot q^*(t) + K_p(q^*(t) - q) + K_d(\dot q^*(t) - \dot q)$.)
- Use **Riccati** to get an **Optimal Linear Regulator** around reference
  - Define optimal control problem, e.g., $\min_{\pi: q, \dot q \mapsto u} \int_0^T c(q(t), \dot q(t), u(t))\, dt + \phi(x(T))$
  - We can linearize dynamics around reference $\to$ has an analytic solution (Algebraic Riccati eq.)
  - Resulting controller is a “linear regulator”, i.e., a PD law where matrices K_p, K_d depend on t and are chosen optimally.

Model-Predictive Control (MPC)

- When getting far away from the reference, linearization of Riccati might break, and PD is too simple

Model-Predictive Control (MPC)

- When getting far away from the reference, linearization of Riccati might break, and PD is too simple

- Continuously replan ($\sim$ 10-1000Hz): re-solve the optimal control problem
  - Optimal Control problem can also include task constraints directly, not only following a reference
  - As a compromise: typically limit horizon

This is a default way of "thinking control" in robotics

Summary

Summary

- A robot is an articulated multi-body system
  - Fwd kinematics: $q \mapsto x,\ \dot q \mapsto \dot x,\ \ddot q \mapsto \ddot x$
  - Fwd dynamics: $u \mapsto \ddot q$, inv dynamics: $\ddot q \mapsto u$

Summary

- A robot is an articulated multi-body system
  - Fwd kinematics: $q \mapsto x,\ \dot q \mapsto \dot x,\ \ddot q \mapsto \ddot x$
  - Fwd dynamics: $u \mapsto \ddot q$, inv dynamics: $\ddot q \mapsto u$

- Standard Control Template:
  - IK (or constraint solving) to estimate future goal/waypoints
  - Path Finding & Trajectory Optimization to estimate Reference Motion
  - PD, Linear Regulator, or MPC to control (around the reference)

How far can we get with this approach?

How far can we get with this approach?

- What did we assume to *know*?
  - Structure of multi-body system, all shapes, inertias
  - All goals/objectives modelled (=programmed) as differentiable costs/constraints

Challenge 1: Interacting with the environment

- If we only care about the **robot itself** (all goals/objectives/models concern the robot directly) – the above it totally fine

Challenge 1: Interacting with the environment

- If we only care about the **robot itself** (all goals/objectives/models concern the robot directly) – the above it totally fine

- Things get challenging when we care about **interacting with the environment**
  - Models/goals/objectives of interaction (contact, grasp) are more complicated

Challenge 1: Interacting with the environment

- Example: Locomotion
  - Interaction: Making contact with the ground to generate ground forces
  - Robot root is not attached to world, but free floating (complicates dynamics a bit)
  - Dynamics heavily influenced by ground forces, which are *contact complementary* hard on-off switching of forces at contact $\to$ hybrid/discrete structure, makes dynamics and solvers much much more complicated (hybrid control)

Challenge 1: Interacting with the environment

- Example: Locomotion
  - Interaction: Making contact with the ground to generate ground forces
  - Robot root is not attached to world, but free floating (complicates dynamics a bit)
  - Dynamics heavily influenced by ground forces, which are *contact complementary* hard on-off switching of forces at contact $\rightarrow$ hybrid/discrete structure, makes dynamics and solvers much much more complicated (hybrid control)

... more complicated than "vanilla robot", but still doable

Challenge 1: Interacting with the environment

- Example: Manipulation
  - Objects in the environment (part of the "multibody system") have their own DOFs, but are NOT "articulated" with motors: if not grasped or touched, they cannot move $\to$ their Jacobian $\partial_q x_i = 0$
  - Hard on-off switching of manipulability; hybrid dynamics & problem
  - Dynamics of object motions can be much more complicated than (also free-floating) robot dynamics: friction, stiction, slip, non-point contacts
  - Waypoint constraints $\phi(x_t)$ much more complicated (correct grasping of complex shape, pushing, throwing)
  - If objects are deformable, their form becomes DOF (e.g. neural latent code) – becomes much much more complicated in above approach

Challenge 1: Interacting with the environment

- Example: Manipulation
  - Objects in the environment (part of the "multibody system") have their own DOFs, but are NOT "articulated" with motors: if not grasped or touched, they cannot move $\to$ their Jacobian $\partial_q x_i = 0$
  - Hard on-off switching of manipulability; hybrid dynamics & problem
  - Dynamics of object motions can be much more complicated than (also free-floating) robot dynamics: friction, stiction, slip, non-point contacts
  - Waypoint constraints $\phi(x_t)$ much more complicated (correct grasping of complex shape, pushing, throwing)
  - If objects are deformable, their form becomes DOF (e.g. neural latent code) – becomes much much more complicated in above approach

- In essence, things become much more complicated, but one still *can* write down essential physics equations of object interaction, and use these equations in above approach

Challenge 2: State Estimation

- All of the above requires to estimate states
  - q_0 (includes pose of a mobile robot)
  - x_i (poses of objects in environment)
  - shapes and inertias in the environment, dynamics parameters (e.g. friction)

[Basic state estimation can often also be formulated as optimization problem (e.g. graph-SLAM) – similar to motion optimization: Find estimates (also of past motion) that is *most consistent* with sensor readings; minimze error between real readings and model-predicted readings. (Or as probabilistic inference.)]

Relation to Robot Learning

- On the formal/theory side, they share foundations:
  - Optimal Control formulation $\leftrightarrow$ Markov Decision Processes & Reinforcement Learning
  - More generally: optimality formulations $\to$ learning/black-box opt. approaches

Relation to Robot Learning

- On the formal/theory side, they share foundations:
  - Optimal Control formulation $\leftrightarrow$ Markov Decision Processes & Reinforcement Learning
  - More generally: optimality formulations $\rightarrow$ learning/black-box opt. approaches

- Components can be *replaced* or *shortcut* by learning:
  - Dynamic modelling $\leftrightarrow$ system identification
  - Optimal Control (e.g., MPC, Riccati) can be shortcut by learning V- or Q-function
  - Need of inverse dynamics can be shortcut by learning Q-function instead of V-function
  - Constraint solving (also IK) can be shortcut by directly learning a policy or sampler that fulfills constraint

Relation to Robot Learning

- On the formal/theory side, they share foundations:
  - Optimal Control formulation $\leftrightarrow$ Markov Decision Processes & Reinforcement Learning
  - More generally: optimality formulations $\to$ learning/black-box opt. approaches

- Components can be *replaced* or *shortcut* by learning:
  - Dynamic modelling $\leftrightarrow$ system identification
  - Optimal Control (e.g., MPC, Riccati) can be shortcut by learning V- or Q-function
  - Need of inverse dynamics can be shortcut by learning Q-function instead of V-function
  - Constraint solving (also IK) can be shortcut by directly learning a policy or sampler that fulfills constraint
  - **Shortcut state estimation:** Avoid all state-based models, learn direct sensor-based models (policies, value functions, planners, dynamics, etc)
  - **End-to-end:** Shortcut the whole approach by learning images $\mapsto$ torques