Lecture Note:
Robot Kinematics & Dynamics

Marc Toussaint

Learning & Intelligent Systems Lab, TU Berlin

May 1, 2024

This is meant as essentials on robotic kinematics and dynamics – developed as background for the *Robot Learning* course.

Articulated Multibody System

A robot is a multibody system. Each body has a **pose** $x_i \in \text{SE}(3)$, an inertia (m_i, I_i) with mass $m_i \in \mathbb{R}$ and inertia tensor $I_i \in \mathbb{R}^{3\times3}$ sym.pos.def., and a shape s_i (any shape representation that defines a pairwise signed-distance $d(s_i, s_j)$ is sufficient).

We assume this multibody system is tree-structured, i.e., every body is linked to a parent body or the world. Body i has a relative transformations $Q_i \in \text{SE}(3)$ from its parent (or world) $\text{par}(i)$. Given the tree structure, we can compute the pose x_i of each body simply by forward chaining all relative transformations all Q_j from world to i. Some robotic systems might not really be tree-structured – these will first be represented as a tree plus additional constraints to describe loops.

What is special about robots is that some of the relative transformations have degrees of freedom (dofs) that are *articulated* (i.e., are motorized/movable). Let Q_i have 1 dofs $q_i \in \mathbb{R}^1$, then $Q_i(q_i)$ is a function of this dof. We stack all dofs of the whole multibody tree into a vector $q \in \mathbb{R}^n$, which is called **joint vector**. We will discuss in more depth the concepts of generalized and minimal coordinates below.

Forward Kinematics & Jacobian

We have defined $x_i \in \text{SE}(3)$ as the pose of body i, and $q \in \mathbb{R}^n$ as the dofs of the full multibody system. The mappings $\phi_i : q \mapsto x_i$ (for any i) is what so-called "forward kinematics" is concerned about – and we already explained that $x_i = \phi_i(q)$ can be computed simply by chaining all relative transformations. (Very often we are not interested in really computing all poses x_i of the multibody system, but only the pose of one relevant body i (esp. the so-called endeffector or manipulator of a robot), or only the position x_i^{pos} or some rotated vector $x_i^v = R_i v$ (with $R_i \in \text{SO}(3)$ its orientation) for body i. In robotics, the word forward kinematics is often used to refer only to computing the endeffector pose, position, or orientation.)

The derivative $J_i(q) = \partial_q \phi_i(q) \in \text{se}(3) \otimes \mathbb{R}^n$ is of central importance to later solve constraint problems, or also to relate joint space velocities $\dot{q}$ to the velocity $\dot{x}_i \in \text{se}(3)$ of the ith body. Note that elements $(v, w) \in \text{se}(3) \equiv \mathbb{R}^6$ are 6-vectors composed of the linear velocity $v \in \mathbb{R}^3$ and angular velocity $w \in \mathbb{R}^3$ (always in world coordinates). Therefore, we can write $J_i(q) \in \mathbb{R}^{6\times n}$ as a matrix, called **Jacobian**, and

$$(v_i, w_i) = J_i(q)\ \dot{q} \tag{1}$$

gives us the linear and angular velocity of body i when joints have velocities $\dot{q}$. In practice, code typically returns separate positional Jacobian $J_i^{\text{pos}} \in \mathbb{R}^{3\times n}$ and angular Jacobian $J_i^{\text{ang}} \in \mathbb{R}^{3\times n}$. In fact, the core

job of a kinematics engine is (1) to represent the articulated multibody tree, (2) to forward compute $\phi_i(q)$, and (3) to compute $J_i^{\mathsf{pos}}, J_i^{\text{ang}}$ for any i.

Since we know how to compute $\phi_i(q)$, we could use "autodiff" to also compute the derivative $J_i(q)$. However, the columns of the positional and angular Jacobians can actually be computed very easily and more efficiently by simple insight of how the local translational/angular velocity of each joint dof translates to the translational/angular velocity of body i. The footnote[1] provides details on how to derive the Jacobian for rotational (hinge), translational (prismatic), and quaternion (ball) joints. The Jacobians are typically sparse for large robotic systems (e.g., multi-robot systems): Every column of $J_i \in \mathbb{R}^{6\times n}$ describes how dof $j \in \{1, .., n\}$ influences body i. This column will be zero if dof j is not between the world and body i in the tree.

Fundamental Kinematics Concepts

The word "kinematics" more generally refers to the mathematical description of the possible motions of a (potentially constrained) multibody system or mechanism *without considering the forces.*

For a multibody system, the poses $x_{1:m}$ fully describe the *configuration* of the system. When x is some (potentially redundant) description of system state, we generally call its embedding space the **configuration space** $\mathcal{X}$. E.g., for our multibody system $\mathcal{X} = \text{SE}(3)^m$ is a generic embedding space. However, not all configurations may be feasible, e.g. because bodies are linked in a multibody system, the relative transformations Q_i have limits, or body shapes cannot penetrate. We can imagine the **feasible configuration space**

$$\mathcal{X}_{\text{fea}}$$

as a manifold of feasible configurations, potentially with disconnected components or holes. (Actually also trans-dimensionality, where some parts have different dimensionality is possible, but we neglect this here.) When introducing coordinates $q \in \mathbb{R}^n$ for $\mathcal{X}$ or $\mathcal{X}_{\text{fea}}$ these are called **generalized coordinates**. (This should not be confused with the word **canonical coordinates**, which is used for coordinates in the phase space of a system, and usually denoted (q, p).)

We discussed the manifold of feasible configurations, however kinematics is about describing feasible *motions* on that manifold. Therefore, formally, kinematics describes which $\dot{q} \in T_q\mathcal{X}$ (in the tangent space of $\mathcal{X}$) are feasible, and therefore which paths of motion on the manifold.

A holonomic constraint is of the form $h(q, t) = 0$, with h a d-dimensional function (where t allows a dependence on absolute time, which is hardly ever relevant in our field). In such a system, the generalized coordinates are not minimal and provide only an embedding space for the true, lower-dimensional feasible manifold. The true **degrees of freedom** of the system are $p = n - d$. **Minimal coordinates** are defined to be generalized coordinates of dimension $n = p$ (where no further holonomic constraints exist). However, sometimes it may be convenient to stick with non-minimal generalized coordinates: E.g. when the feasible manifold is S^1, it might be convenient to use redundant $q = (x, y)$ coordinates and add the constraint $x^2 + y^2 = 1$ rather than introducing an angle coordinate and running into the annoying modulo issue. Analogous for SO(3) and embedding quaternion coordinates $\mathbb{R}^4$ may be more convenient than minimal coordinates for SO(3) (see the Quaternion Lecture Notes). Further, when we have a closed loop robotic system, it is convenient to use non-minimal joint coordinates along a tree and profit from the efficiency of tree-based kinematics engines, and handle the closed loop constraint otherwise.

[1] For instance, if j is a rotational ("hinge") joint around axis e in the joint's origin frame $x_{\text{par}(j)}$, and body i is downstream at current pose x_i, then the jth column of

$$J_i^{\text{ang}}$$

is $a = (R_{\text{par}(j)}e)$ (rotates about the axis of j in world coordinates), and the jth column of J_i^{pos} is

$$a \times (x_i^{\mathsf{pos}} - x_{\text{par}(j)}^{\mathsf{pos}})$$

(translates with a lever around the axis). Similar arguments can be made if j is a translational ("prismatic") joint, and a bit more complicated arguments if j is a ball joint parameterized by a quaternion $q_j \in \mathbb{R}^4$ (see the Quaternion Lecture Notes). Thinking of other joint types as compositions of these basic joints, this covers all cases.

A non-holonomic constraint is of the form $h(q, \dot{q}, t) = 0$, which is a general description of any constraints on possible motions on the feasible manifold. A typical example is a car or wheel in 2D: We describe the configuration naturally with $x = (p, \varphi)$ with 2D position $p \in \mathbb{R}^2$ and heading angle $\varphi \in \mathbb{R}$. Note that (without obstacles) any configuration is feasible, so the full configuration space is feasible. As there is no better alternative, we choose generalized coordinates equally as $q = x$. However, at any q, the positional velocity is constraint to aligned with the heading direction, $\dot{p}^\top(\sin\phi, \cos\phi) = 0$, which is a non-holonomic constraint.

There are cases where a constraint is naturally expressed as $h(q, \dot{q}, t) = 0$ and "looks" non-holonomic, but actually it is so-called integratable. This mean that, by analyzing integrals of trajectories of feasible velocities $\dot{q}(t)$, we understand that the actually reachable q all lie on a sub-manifold which we could more directly be described by a holonomic constraint $h(q, t) = 0$ (here, the absolute time t dependence really is important). So, such "integratable non-holonomic constraints" can be reformulated to become holonomic and still describe a holonomic system. The literature describes elaborate maths (Pfaffian form of constraints) to uniquely decide whether a system is truly holonomic or non-holonomic. But this is rarely relevant for typical robotic systems in our field.

"Force Kinematics"

We learned that with $(v_i, w_i) = J_i\dot{q}$ the Jacobian relates joint velocities to body velocities. Let's do the analogous for forces: Assume we have a wrench (f_i, τ_i) directly acting on body i, what "do we feel" in the joints, i.e., what torques $u \in \mathbb{R}^n$ are propagated into the joints? The answer is the Jacobian transpose: $u = J^\top(f_i, \tau_i)$. We can derive this by the conservation of work, or better, power (=work/time): For joint torques u and velocities $\dot{q}$ we would consume power $u^\top\dot{q}$, which needs to equal the power received at the body, $(f_i, \tau_i)^\top(v_i, w_i) = (f_i, \tau_i)^\top J\dot{q}$. As this holds for any $\dot{q}$ we have $u^\top = (f_i, \tau_i)^\top J$.

Dynamics

While kinematics describes which q and $\dot{q}$ are feasible, dynamics describes which $\ddot{q}$ are feasible. For a passive dynamical system there is just one $\ddot{q}$ feasible: the one that follows from the laws of physics. For an articulated robotic system we can choose to exert torques u in each joint (or some joints), and thereby create various $\ddot{q}$. For standard, fully actuated robotic systems we can command torques $u \in \mathbb{R}^n$ in all dofs and create arbitrary accelerations in them. However, when our multibody system description includes passive objects of the environment, or describes a free-floating (walking/running) robot, the feasible accelerations are highly constrained, depending esp. on contacts and possibilities of force transmission to objects or the ground through contacts.

Assume we know how we want to accelerate $\ddot{q}$ the system, and want to compute the necessary joint torques u to achieve this acceleration. That is, we want to derive the mapping $(q, \dot{q}, \ddot{q}) \mapsto u$. Given the Jacobians described above, this is easy to derive: For each body, we can compute $(v_i, w_i) = J_i\dot{q}$ and $(\dot{v}_i, \dot{w}_i) = J_i\ddot{q} + \dot{J}_i\dot{q}$. I.e., we know how we want each body to accelerate. The Newton-Euler equation tells us that such an acceleration would raise the following inertia forces at the body:

$$\begin{pmatrix} f_i \\ \tau_i \end{pmatrix} = \begin{pmatrix} m_i\dot{v}_i \\ \bar{I}_i\dot{w}_i + w_i \times \bar{I}_i w_i \end{pmatrix} = M_i J_i\ddot{q} + c_i \ , \tag{2}$$

with $M_i = \mathrm{diag}(m\mathbf{I}_3, \bar{I}_i)$, $c_i = M_i\dot{J}_i\dot{q} + (0_3, w_i \times \bar{I}_i w_i)$, where $\bar{I}_i = R_i I_i R_i^\top$ and I_i the inertia tensor in body coordinates. Note that for brevity we dropped dependencies $M_i = M_i(q), J_i = J_i(q)$, and $c_i = c_i(q, \dot{q})$. (When implementing this, note that $\dot{J}_i\dot{q}$ is the frame acceleration for $\ddot{q} = 0$, which can best be computed by forward propagation of accelerations over the full tree and captures the Coriolis effects.[2])

Conversely, to counteract these inertia forces we have to apply joint torques $u = J_i^\top\Big[M_i J_i\ddot{q} + c_i\Big]$ – this is how "we feel" the inertia in the joints. We can separately consider gravity: By the same argument we

[2]Forward propagation of velocities and accelerations: If parent frame has $(p, R, v, w, \dot{v}, \dot{w})$, the relative transform is

need joint torques $u = J_i^\top g_i$ to counteract gravity, where $g_i = (mge_z, 0_3) \in \mathbb{R}^6$ has only a single entry in z-direction. As we have many bodies that are accelerated and need gravity compensation, we overall have

$$u = \sum_{i=1}^{m} J_i^\top \Big[M_i J_i \ddot{q} + c_i + g_i \Big] \ . \tag{4}$$

Other texts provide derivations either via the general Euler-Lagrange equations, or recursive Newton-Euler equations. I find the above very concise and easy to implement, and efficient with sparse J_i. The first term $\sum_{i=1}^m J_i^\top M_i J_i \ddot{q}$ can elegantly also be found in the Euler-Lagrange derivation,[3] but the Coriolis terms c_i are less obvious. Recursive Newton-Euler can be tuned to be numerically faster, but does not provide this nice general and invertible form.

In standard notation, general robot dynamics are written in the form

$$u = M(q)\ \ddot{q} + F(q, \dot{q}) \tag{7}$$

where, for multi-rigid-body systems, we derived $M(q) = \sum_i J_i^\top M_i J_i$ and $F(q, \dot{q}) = \sum_i J_i^\top (c_i + g_i)$.

Keep in mind that only when $M(q)$ is invertible we have a one-to-one relation between our controls and the system acceleration, and we have the guarantee that our system accelerates with $\ddot{q} = M(q)^{-1}u - F(q, \dot{q})$ as desired. $M(q)$ is not invertible if J_i do not have full rank, e.g., if some body i is not articulated at all and J_i is zero. In this case the equation $u = J_i^\top \Big[M_i J_i \ddot{q} + c_i \Big]$ says that we feel none of the accelerations of i in our joints – and conversely cannot induce any accelerations of i. That's the case when the robot is not in contact with body i.

Standard Usage: Waypoint + Reference Motion + Controller

With the above equations we can accelerate the system in any way we like – at least those dofs that are currently articulable. In this view, the rest is planning: We need to decide how we want to accelerate the system right now in order to reach some future goal.

There are a myriad of opinions on how robotic control systems and middleware should be structured. Here is just one version, which I consider a baseline.

Consider that we want to have the robot fulfill a kinematic constraint

$$\phi(q_{t=T}) = y^* \tag{8}$$

at time $t = T$, where ϕ is a d-dimensional constraint function that typically depends on some poses x_i of some bodies, and $y^* \in \mathbb{R}^d$ is called a setpoint. A standard control stack could address this problem as follows:

$(p', R', v', w', 0, 0)$, then child frame has

$$\begin{pmatrix} \bar{p} \\ \bar{R} \\ \bar{v} \\ \bar{w} \\ \bar{\dot{v}} \\ \bar{\dot{w}} \end{pmatrix} = \begin{pmatrix} p + Rp' \\ RR' \\ v + Rv' + w^\times Rp' \\ w + Rw' \\ \dot{v} + w^\times Rv' + w^\times w^\times Rp' + \dot{w}^\times Rp' \\ \dot{w} + w^\times Rw' \end{pmatrix} = \begin{pmatrix} p + P' \\ RR' \\ v + V' + w^\times P' \\ w + W' \\ \dot{v} + w^\times V' + (w^\times w^\times + \dot{w}^\times)P' \\ \dot{w} + w^\times W' \end{pmatrix} \tag{3}$$

where capital letters are relative vectors rotated to world coordinates.

[3] The Euler-Lagrange derivation starts with $\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = u$, where $L(q, \dot{q}) = T(q, \dot{q}) - U(q)$ with the system kinetic energy

$$T(q, \dot{q}) = \sum_i \frac{1}{2} m_i v_i^2 + \frac{1}{2} w_i^\top \bar{I}_i w_i = \sum_i \frac{1}{2} \dot{q}^\top J_i^\top M_i J_i \dot{q}, \ M_i = \mathrm{diag}(m_i \mathbf{I}_3, \bar{I}_i), \tag{5}$$

and the system potential energy $U(q) = \sum_i g m_i x_i^z$. When computing the partial derivatives analytically we get something of the form

$$u = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = M(q)\ddot{q} + \dot{M}\dot{q} - \frac{\partial T}{\partial q} + \frac{\partial U}{\partial q}, \tag{6}$$

where total inertial $M(q) = \sum_i J_i^\top M_i J_i$ is simple to compute, but the Coriolis terms are more complicated.

(i) First compute a final robot pose q_T that fulfills constraint $\phi(q_{t=T}) = y^*$ – that problem is called **inverse kinematics** and discussed below.

(ii) Next compute a *reference* motion from current robot pose q_0 to q_T – that problem can be addressed with **path finding**, **trajectory optimization**, or basic interpolation with a motion profile or spline (see the Spline Lecture Notes).

(iii) Finally, determine a control policy $\pi : (x, t) \mapsto u$ that reactively computes motor commands u to follow the reference motion – that problem can be addressed using **PD control**, inverse dynamics as derived above, **Riccati**, or **model-predictive control (MPC)**.

You could think of these as three different time scales: First rough future waypoint(s)/goal(s), then continuous motion to next waypoint, then short-term controls. Continuous replanning/re-estimation can also make (1) and (2) reactive.

Of course, robotics systems do not have to be organized in that way: Some approaches skip step (1) and let step (2) also solve for the final configuration (e.g., including the optimization of q_T into the trajectory optimization problem); or one may skip steps (1) and (2) and let step (3) handle the full problem (e.g., including the goal constraint in the MPC formulation, or a basic task-space PD controller ("operational space control"); which typically looses the power of path finding and trajectory optimization).

Inverse Kinematics

Inverse kinematics (IK) simply means computing q to fulfill $\phi(q) = y^*$. A proper approach is to formulate this as an NLP (non-linear mathematical program)

$$\min_{q\in\mathbb{R}^n} \|q - q_0\|^2 \quad \text{s.t.} \quad \phi(q) = y^* \tag{9}$$

$$\text{or} \quad \min_{q\in\mathbb{R}^n} \|q - q_0\|^2 + \mu\|\phi(q) - y^*\|^2 \quad \text{for large } \mu \tag{10}$$

and use an efficient NLP solver (e.g. Augmented Lagrangian, or SQP, exploiting potential sparseness of $\frac{\partial}{\partial q}\phi$). However, typical textbooks at length discuss computing IK more low-level. For instance, when approximating $\phi(q) \approx \phi(q_0) + J(q - q_0)$ as linear with Jacobian J, the analytical solution to (10) can be written as (allowing for $\mu \to \infty$):

$$q^* = q_0 + J^\top (JJ^\top + \tfrac{1}{\mu}\mathbf{I})^{-1}(y^* - \phi(q_0)) \ . \tag{11}$$

In the context of optimization, this is the first Newton step when initializing optimization at q_0. Students sometimes interpret this equation as a tool to directly generate robot motion: They let the robot literally execute these Newton steps (scaled by a small factor α), and then the robot starts moving like the decision variable in a non-linear optimization problem with small-scaled Newton steps. This is not proper! Proper IK should really first compute the solution q_T to (9), and then think about how the robot can actually move to q_T (e.g. using proper optimal control, or reactive spline interpolation, or a basic but nice motion profile).