Introduction to Robotics
Lecture 12: Numerical Inverse Kinematics

Inverse kinematics

- Forward kinematics: compute the end-effector position (as an element of $SE(3)$) from joint angles θ_i: compute the function

$$T : \text{joint space} \to SE(3) : \theta \mapsto T(\theta)$$

- Inverse kinematics: compute the (possible) joint angles from the position of the end-effector: compute the function

$$T^{-1} : SE(3) \to \text{joint space} : X \mapsto \theta.$$

- The inverse kinematics function is often *multi-valued*.
- When analytic solutions are are or impossible to come by, we can solve $T(\theta) - X = 0$ for θ numerically.
- We write the previous equation as $f(\theta) - x = 0$, where $x \in \mathbb{R}^m$ and $f : \mathbb{R}^n \to \mathbb{R}^m$.

Newton-Raphson method

- Let x_d be the desired end-effector coordinates (the ones we want to find joint angles θ_i's for). Define $g(\theta) := f(\theta) - x_d$. We need to find a *zero* of $g(\theta)$, that is θ_d so that $g(\theta_d) = 0$.
- Start with an initial *guess* θ^0 for θ_d. Using a Taylor expansion, we can write

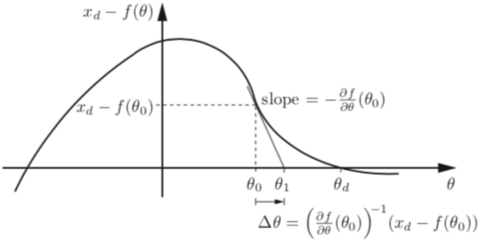
$$x_d = f(\theta_d) = f(\theta^0) + \underbrace{\frac{\partial f}{\partial \theta}|_{\theta^0}}_{J(\theta^0)} \underbrace{(\theta_d - \theta^0)}_{\Delta\theta} + h.o.t,$$

  where we see that the Jacobian evaluated at θ^0, $J(\theta^0)$, appears.
- Truncating the expansion, we get

$$J(\theta^0)\Delta\theta = x_d - f(\theta^0).$$

  We can use this equation to get an approximation to $\Delta\theta$!

Newton-Raphson method



- Assuming that $J(\theta^0)$ is invertible, we get

$$\Delta\theta = J^{-1}(\theta^0)(x_d - f(\theta^0)).$$

- We can then set

$$\theta^1 := \theta^0 + \Delta\theta$$

and iterate the process to obtain a sequence $\{\theta^0, \theta^1, \theta^2, \ldots\}$ converging to θ_d.

The case of non-invertible Jacobian: pseudo-inverse

- The Jacobian $J(\theta^0)$ can fail to be invertible for 2 reasons: either it is singular (i.e. with $\det J(\theta^0) = 0$), or it is non-square.
- In both cases, we can replace the inverse of J by its *pseudo-inverse*.
- For $J \in \mathbb{R}^{m\times n}$, we denote by $J^\dagger \in \mathbb{R}^{n\times n}$ its *Moore-Penrose pseudo-inverse*, or simply pseudo-inverse.
- Consider the linear equation

$$Jy = z.$$

  It either has *many* solutions (e.g. if $n < m$), exactly one solution (e.g. if $m = n$ and J is full rank), or no solutions (e.g. if $n > m$ and z is not in the column span of J.)

The case of non-invertible Jacobian: pseudo-inverse

- The solution
$$y^* = J^\dagger z$$
is so that
    1. If J is square and invertible, $y^* = J^{-1}z$.
    2. If there are many solutions to $Jy = z$, then y^* is the one of *minimal norm*. That is, for any other solution $\tilde{y}$, $J\tilde{y} = z$, we have $\|y^*\| \leq \|\tilde{y}\|$.
    3. If there are no solutions, then y^* minimizes the norm of the error
    $$\|Jy^* - z\| \leq \|J\tilde{y} - z\|$$
    for all $\tilde{y}$

The case of non-invertible Jacobian: pseudo-inverse

- When J is of full column rank (for $m > n$, tall matrix), we have
$$J^\dagger = (J^\top J)^{-1} J^\top$$
- When J is of full row rank (for $n < m$, wide matrix), we have
$$J^\dagger = J^\top (J J^\top)^{-1}$$
- When $n = m$ and J is of full rank, $J^\dagger = J^{-1}$.
- If the matrix is not of full rank, remove redundant columns or rows and apply above formulas

Numerical inverse kinematics

- When J is of full column rank (for $m > n$, tall matrix), we have
$$J^\dagger = (J^\top J)^{-1} J^\top$$
- When J is of full row rank (for $n < m$, wide matrix), we have
$$J^\dagger = J^\top (J J^\top)^{-1}$$
- When $n = m$ and J is of full rank, $J^\dagger = J^{-1}$.
- If the matrix is not of full rank, remove redundant columns (or rows)

Numerical inverse kinematics

- The Newton-Raphson algorithm needs to be modified in order to take into account that $X \in SE(3)$, which comes with some constraints, and is not a general matrix in $\mathbb{R}^{4\times 4}$.
- Deriving the algorithm exactly requires more advanced mathematics, which is outside the scope of this course.
- Intuitively, the error vector $x_d - f(\theta^i)$ represents the update needed to go from the current guess to the desired end-effector configuration (after being multiplied by the inverse Jacobian).
- Said otherwise, following the direction $(x_d - f(\theta^i))$ for one second, starting from $f(\theta^i)$, should send us to x_d (but only does it approximately, because of the truncation of Taylor series).
- In our case, we are given $X \in SE(3)$, and instead of computing $X - T(\theta^i)$, we should compute the *twist* which, if followed for one second, sends us from $T(\theta^i)$ to X.

Numerical inverse kinematics

- Denote this twist by $\mathcal{V}_b$. Recall that X is the desired configuration, and $T_{sb}(\theta^i)$ the current configuration in the algorithm. The twist that sends us from $T_{sb}(\theta^i)$ to X satisfies by definition
$$T_{sb}(\theta^i)e^{[\mathcal{V}_b]} = X =: T_{sd}.$$
- Hence $e^{[\mathcal{V}_b]} = T_{sb}^{-1}(\theta^i)T_{sd}$ and we obtain
$$[\mathcal{V}_b] = \log(T_{sb}^{-1}(\theta^i)T_{sd})$$

Numerical inverse kinematics: algorithm

Proceeding by analogy with our previous algorithm, we obtain:

1. Given $X = T_{sd}$ a desired position for the end-effector. Given $T_{sb}(\theta)$ the forward kinematics map. Given tolerances ε_w and ε_v. Given an initial guess θ^0.
2. While $\|\omega_b\| > \varepsilon_w$ *or* $\|v_b\| > \varepsilon_v$:
   2.1 Set $[\mathcal{V}_b] = \log(T_{sb}(\theta^i) T_{sd})$
   2.2 Set $\theta^{i+1} := \theta^i + J_b^\dagger(\theta^i)\mathcal{V}_b$
   2.3 Increment i

Numerical inverse kinematics: zero of $SE(3)$-valued functions

- We can derive the algorithm given on the previous slide as follows: our final goal is to find a $\Delta\theta$ so that

$$T_{sb}(\theta^i + \Delta\theta) = T_{sd},$$

  and thus have θ^d. We cannot obtain it at once usually, but we can write a first order approximation to it and iterate.

- Writing the first order expansion of the left-hand-side, we get

$$T_{sb}(\theta^i) + \frac{\partial T}{\partial\theta}\Delta\Theta \simeq T_{sd}.$$

Zeros of $SE(3)$-valued functions

- An alternative approach, that uses the fact that the function T is valued in $SE(3)$, is the following: we write

$$T_{sb}(\theta^i)e^{[\mathcal{V}]} = T_{sd}$$

  which implies that $[\mathcal{V}] = \log(T_{sb}^{-1}(\theta^i)T_{sd})$.

- Now we are after $\Delta\theta$ as described in the previous slide, and we decided to approximate it up to first order. Hence we expand the exponential, up to first order, to get

$$T_{sb}(\theta^i)(I + [\mathcal{V}]) + h.o.t. = T_{sd}$$

Zeros of $SE(3)$-valued functions

- Multiplying the last equation by T_{sb}^{-1} on the left, and similarly for the equation $T_{sb}(\theta^i) + \frac{\partial T}{\partial \theta}\Delta\theta \simeq T_{sd}$, we get

$$I + [\mathcal{V}] = T_{sb}^{-1} T_{sd}$$
$$I + J_b \Delta\theta = T_{sb}^{-1} T_{sd}$$

  where we recall that the body Jacobian is exactly $T_{sb}^{-1}\frac{\partial T}{\partial \theta}$
- We conclude that $J_b\Delta\theta = [\mathcal{V}]$ and thus $\Delta\theta = J_b^{\dagger}[\mathcal{V}]$, where $[\mathcal{V}] = \log(T_{sb}^{-1} T_{sd})$. For a higher order approximation, we should keep more terms in the expansion, but then we need to solve quadratic equations (in $\mathcal{V}$ and $\Delta\theta$).
- This matches the algorithm given earlier.
- Can you write an iterative algorithm that uses the space Jacobian instead of the body Jacobian?

Inverse Velocity Kinematics

- Assume you want a robot's end-effector to follow a trajectory $T_{sb}(t)$.
- One way to do it is to discretize the trajectory $T_{sb}(t_k)$ and compute θ_k to that

  $$T(\theta_k) = T_{sb}(t_k).$$

  If doing so, we need to make sure that θ_k and θ_{k+1} are close to each other, since there may be many solutions to that equation. One possibility is to initialize with the previous value: set $\theta_{k+1}^0 = \theta_k$.
- Equivalently, we can feed velocities to the joints evaluated according to

  $$\dot{\theta}(t_k) \simeq \frac{\theta(t_k) - \theta(t_{k-1})}{(t_k - t_{k-1})}$$
- This approach relies on the previously seen method for computing the inverse kinematics