Robotics algorithms

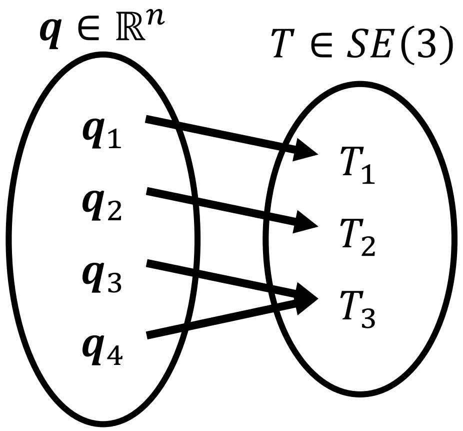
Lee Yongseok, Jung Hyunseo

dldydtjr2000@postech.ac.kr

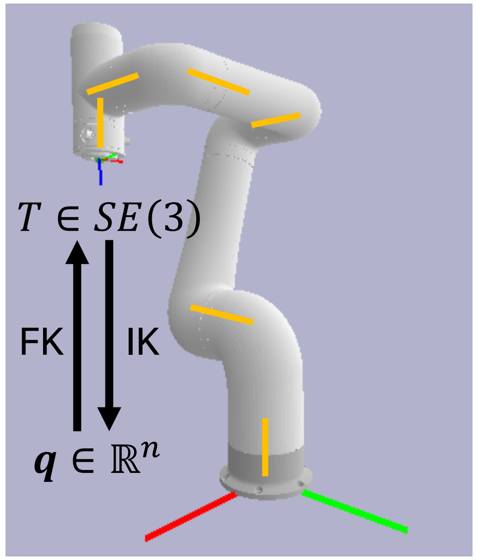
1. Numerical IK

Inverse kinematics (IK)

The inverse kinematics (IK) is to find the joint position (called as joint variables) $\boldsymbol{q} \in \mathbb{R}^n$ for a given transformation matrix $T \in SE(3)$.
The IK problem is a nonlinear problem; the existence and uniqueness of the solution are not always guaranteed, while the forward kinematics (FK) is surjective.



Forward kinematics (FK)



Inverse kinematics has two approaches: closed-form solutions and numerical methods. The closed-form solution uses geometry and polynomial equations, but there is no general solution for a 6-dof robotic system due to its complex structure. The numerical method has a computational burden but is easy to apply, and we will deal with the numerical methods in this lecture.

1. Numerical IK

Numerical IK: Newton-Raphson method

For numerical solutions, we can resort to the Newton-Raphson method. Let the target task pose is $\boldsymbol{p}^{des}$ and current joint pose is $\boldsymbol{q}^{(k)}$. Here, our objective is to find a desired joint pose $\boldsymbol{q}^{des}$ in next step $\boldsymbol{q}^{(k+1)}$, i.e.

$$\boldsymbol{p}^{des} = f\left(\boldsymbol{q}^{(k+1)}\right) = f\left(\boldsymbol{q}^{(k)}\right) + \left.\frac{\partial f}{\partial \boldsymbol{q}^{\top}}\right|_{\boldsymbol{q}^{(k)}} \cdot \left(\boldsymbol{q}^{(k+1)} - \boldsymbol{q}^{(k)}\right) + \cdots$$

Here, the partial derivative of $f(\boldsymbol{q})$ is a jacobian matrix $J(\boldsymbol{q})$, so

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + J^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \left(\boldsymbol{p}^{des} - f\left(\boldsymbol{q}^{(k)}\right)\right)$$

where J^{+} is a pseudo-inverse of jacobian (ex. Moore-Penrose, Damped least-square).

The convergence rate of the Newton-Raphson method can be adjusted by multiplying scaling constant κ

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + \kappa J^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \left(\boldsymbol{p}^{des} - f\left(\boldsymbol{q}^{(k)}\right)\right)$$

march lab

1. Numerical IK

Numerical IK: Newton-Raphson method

However, the desired target pose is given in 4x4 homogeneous transformation matrix $T \in SE(3)$, so the following part cause a problem.

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + \kappa J^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \left(\boldsymbol{p}^{des} - f\left(\boldsymbol{q}^{(k)}\right)\right)$$

This part means the error between desired pose and current pose, but $SE(3)$ is not closed under the ‘matrix summation’. Therefore, we need to define a new representation of error between two transformation matrices.

Recall the jacobian matrix $J_r \in \mathbb{R}^{6\times n}$ in the lecture.

$$\left[\dot{x}\ \ \dot{y}\ \ \dot{z}\ \ \omega_x\ \ \omega_y\ \ \omega_z\right]^{\mathsf{T}} = J_r(\boldsymbol{q}) \cdot \dot{\boldsymbol{q}}$$

For a numerical IK, we can use the pose error term as follow:

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + \kappa J_r^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \left[(position\ error)^{\mathsf{T}}\ \ (orientation\ error)^{\mathsf{T}}\right]^{\mathsf{T}}$$

It is clear that the position error is $\boldsymbol{r}^{err} = \left[x^{des} - x^{(k)}\ \ \ y^{des} - y^{(k)}\ \ \ z^{des} - z^{(k)}\right]^{\mathsf{T}}$. However, how can we define the orientation error?

1. Numerical IK

Error in SO(3): matrix exponential/logarithm (Out of the course)

The error between two rotation matrices R_1 and R_2 can be defined as follow:

$$R^{err} = R_1^{\top} R_2$$

Also, this error matrix R^{err} can be mapped into error vector $\boldsymbol{\xi}^{err} \in \mathbb{R}^{3\times1}$

$$\boldsymbol{\xi}^{err} = \log(R^{err}) \quad \text{and} \quad R^{err} = \exp(\boldsymbol{\xi}^{err})$$

For more details, please refer [1], [2] or take an advanced robotics lecture. (The python code will be provided!)

Therefore, the numerical IK can be formulated as follow:

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + \kappa J_r^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \begin{bmatrix} \boldsymbol{r}^{err} \\ \boldsymbol{\xi}^{err} \end{bmatrix}$$

[1] Lynch, Kevin M., and Frank C. Park. Modern robotics. Cambridge University Press, 2017.
[2] Murray, Richard M., Zexiang Li, and S. Shankar Sastry. A mathematical introduction to robotic manipulation. CRC press, 2017.

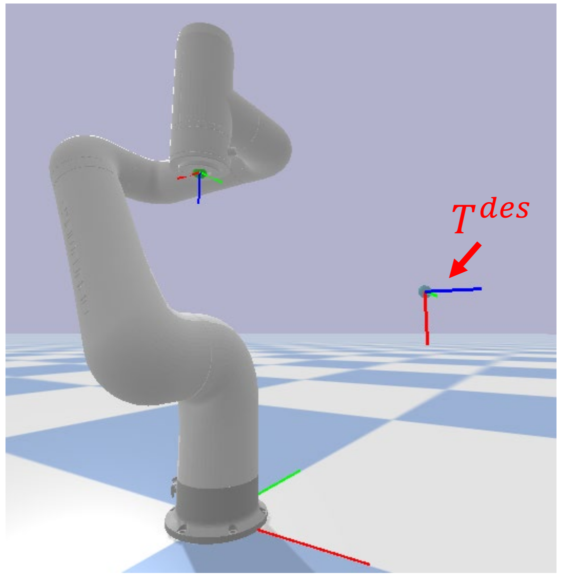
march lab

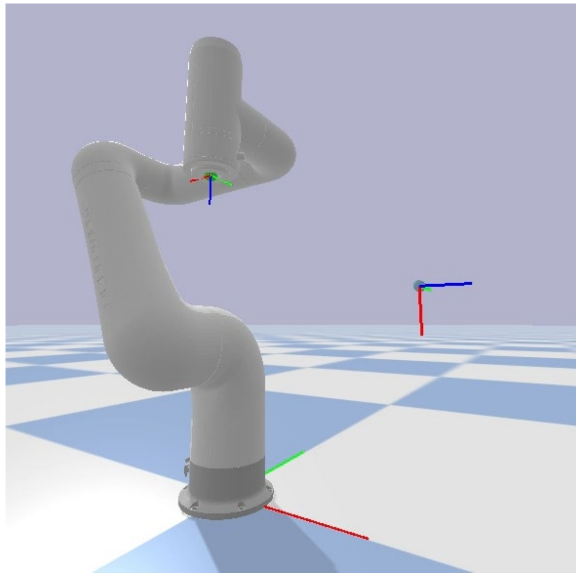
1. Numerical IK

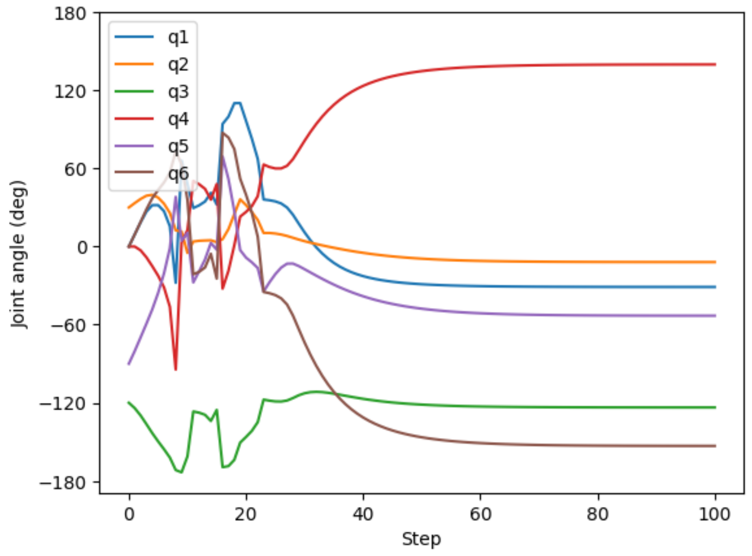
Example

$$\boldsymbol{q}^{(0)} = [0 \;\; \pi/6 \;\; -2\pi/3 \;\; 0 \;\; -\pi/2 \;\; 0]^\top \qquad T^{des} = \begin{bmatrix} 0 & 0 & 1 & 0.4 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{q}^{(k+1)} \leftarrow \boldsymbol{q}^{(k)} + \kappa J_r^{+}\left(\boldsymbol{q}^{(k)}\right) \cdot \begin{bmatrix} \boldsymbol{r}^{err} \\ \boldsymbol{\xi}^{err} \end{bmatrix} \qquad \kappa = 0.1$$







2. Robot Dynamics

Lagrangian approach

The dynamics of the mechanical system can be derived through Lagrangian $L = T - U$.

$$L = T - U = \frac{1}{2}\dot{\boldsymbol{q}}^\top M(\boldsymbol{q})\dot{\boldsymbol{q}} - U(\boldsymbol{q})$$

$$\begin{aligned}
\boldsymbol{\tau} &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial L}{\partial \boldsymbol{q}} \\
&= \frac{d}{dt}\left(M(\boldsymbol{q})\dot{\boldsymbol{q}}\right) - \frac{1}{2}\dot{\boldsymbol{q}}^\top \frac{\partial M}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial U}{\partial \boldsymbol{q}} \\
&= M(\boldsymbol{q})\ddot{q} + \left(\dot{M}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \frac{1}{2}\dot{\boldsymbol{q}}^\top \frac{\partial M}{\partial \boldsymbol{q}}\right)\dot{\boldsymbol{q}} + \frac{\partial U}{\partial \boldsymbol{q}} \\
&= \underbrace{M(\boldsymbol{q})}_{\text{Mass matrix}}\ddot{\boldsymbol{q}} + \underbrace{C(\boldsymbol{q},\dot{\boldsymbol{q}})}_{\text{Coriolis matrix}}\dot{\boldsymbol{q}} + \underbrace{\boldsymbol{g}(\boldsymbol{q})}_{\substack{\text{Gravity vector} \\ \text{(Gravity compensate term)}}}
\end{aligned}$$

$$L = \frac{1}{2}\sum_{j,k} m_{jk}\dot{q}_j\dot{q}_k - U(\boldsymbol{q})$$

$$\begin{aligned}
\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} &= \frac{d}{dt}\left(\frac{1}{2}\sum_{i,k} m_{ik}\dot{q}_k + \frac{1}{2}\sum_{j,i} m_{ji}\dot{q}_j\right) - \frac{1}{2}\sum_{j,k}\frac{\partial m_{jk}}{\partial q_i}\dot{q}_j\dot{q}_k + \frac{\partial U}{\partial q_i} \\
&= \sum_j m_{ij}\ddot{q}_j + \sum_{j,k}\frac{\partial m_{ij}}{\partial q_k}\dot{q}_j\dot{q}_k - \frac{1}{2}\sum_{j,k}\frac{\partial m_{jk}}{\partial q_i}\dot{q}_j\dot{q}_k + \frac{\partial U}{\partial q_i} \\
&= \sum_j m_{ij}\ddot{q}_j + \frac{1}{2}\sum_{j,k}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i}\right)\dot{q}_j\dot{q}_k + \frac{\partial U}{\partial q_i} \\
&= \sum_j m_{ij}\ddot{q}_j + \sum_{j,k}\Gamma_{ijk}(\boldsymbol{q})\dot{q}_j\dot{q}_k + \frac{\partial U}{\partial q_i}
\end{aligned}$$

$$\tau = M(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{q}}^\top\underbrace{\boldsymbol{\Gamma}(\boldsymbol{q})}_{\substack{\text{Christoffel symbols} \\ (\text{tensor, } \mathbb{R}^{n\times n\times n})}}\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})$$

2. Robot Dynamics

Inverse dynamics & Forward dynamics

Forward dynamics (for a robot simulation): $q, \dot{q}, \tau \rightarrow \ddot{q}$ $\quad$ $\ddot{\boldsymbol{q}} = M(\boldsymbol{q})^{-1}\left(\boldsymbol{\tau} - C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q})\right)$

Inverse dynamics (for a robot control): $q, \dot{q}, \ddot{q} \rightarrow \tau$ $\quad$ $\boldsymbol{\tau} = M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q})$

In practice, the inverse dynamics is computed recursively using recursive Newton-Euler method (RNE). It is out of this course, so please refer the [1], [2] if you want details.

[1] Lynch, Kevin M., and Frank C. Park. Modern robotics. Cambridge University Press, 2017.
[2] Murray, Richard M., Zexiang Li, and S. Shankar Sastry. A mathematical introduction to robotic manipulation. CRC press, 2017.

3. Robot Control

Simple joint compliance control

The goal of joint compliance control is to get $e = q_{des} - q \rightarrow 0$ when $q_{des} = 0$, the controller opertates the robot while preserving the systems inherent passivity at all times.
The control law is very simple.

$$\tau = \underbrace{\hat{g}(q)}_{\text{Gravity compensation}} + \underbrace{Ke - D\dot{e}}_{\text{PD control (joint compliance)}}$$

This is a typical controller in industrial robots, it compensates for gravity and uses simple PD control. This controller, however, works very well, and this controller is exponentially stable when the gravity is perfectly compensated.
(it means the nominal gravity vector $\hat{g}$ equals to the real gravity vector g; $\hat{g} = g$).

When gravity compensation is imperfect or absent, assessing stability becomes more complex. The equilibrium is dislocated away from $e = 0$ (it means there is a steady state error). We can inject integral error feedback to reject constant disturbance due to the imperfect gravity compensation, so the modified control is expressed as

$$\tau = \hat{g}(q) + Ke - D\dot{e} + \int e\, dt$$