Feedforward Control

- Feedforward control assumes \( q \approx q_d \) and \( \dot{q} \approx \dot{q}_d \)

- Hence, we have

\[
    u = u_{\text{FF}}(q_d, \dot{q}_d, \ddot{q}_d) + u_{\text{FB}}
\]

with feedforward torque prediction using an inverse dynamics model

\[
    u_{\text{FF}} = M(q)\ddot{q} + c(q, \dot{q}) + g(q)
\]

and a linear PD control law for feedback

\[
    u_{\text{FB}} = K_P(q_{\text{des}} - q) + K_D(\dot{q}_{\text{des}} - \dot{q})
\]
Feedforward Control

\[ T \]

\[ q_d, \dot{q}_d, \ddot{q}_d \]

\[ \text{Inverse Dynamics} \]

\[ q, \dot{q} \]

\[ u \]

\[ \text{Dynamics} \]
Feedforward Control

Key on feedforward control (FF) …

- FF can be done with less real-time computation as feedforward terms can often be pre-computed.

- FF is generally more stable - even with bad models or approximate models

- Only when you have a very good model, you should prefer Model-based Feedback Control.

- In practice, FF is often more important…
Content of this Lecture

1. What is a robot?

2. Modeling Robots
   - Kinematics
   - Dynamics

3. Representing Trajectories
   - Splines

4. Control in Joint Space
   - Linear Control
   - Model-based Control

5. Control in Task Space
   - Inverse Kinematics
   - Differential Inverse Kinematics
Assume your plan is in a task space...

I.e., we want the end-effector to follow a specific trajectory \( \mathbf{x}(t) \)

- Typically given in Cartesian coordinates
- Eventually also orientation

\[
x_d, \dot{x}_d, \ddot{x}_d \\
\]

\[
q, \dot{q}, \ddot{q} \\
\]

\[
x, \dot{x}, \ddot{x} \\
\]
Why don’t we try it this way?
Inverse Kinematics (IK)

What do we want to have?

- Inverse Kinematics: A mapping from task space to configuration

If I want my center of gravity in the middle what joint angles do I need?

$$q = f^{-1}(x)$$
Example 1 - revisited

As $x = q_1 + q_2$

we have

$q_1 = h$
$q_2 = x - h$

for any $h \in \mathbb{R}$

⇒ We have infinitely many solutions!!! Yikes!
Example 2 - revisited

We can solve for $\theta_1$ and $\theta_2$ and get

$$\theta_2 = \cos^{-1}\left( \frac{x^2 + y^2 - \alpha_1^2 - \alpha_2^2}{2\alpha_1\alpha_2} \right)$$

$$\theta_1 = \tan^{-1}\left( \frac{y}{x} \right) - \tan^{-1}\left( \frac{\alpha_2 \sin \theta_2}{\alpha_1 + \alpha_2 \cos \theta_2} \right)$$

$\Rightarrow$ **BUT:** There is more than one solution!

$\Rightarrow$ **This is not a function!**
Problems with Inverse Kinematics

Multiple solutions even for non-redundant robots (Example 2)

Redundancy results in infinitely many solutions.

» Often only numerical solutions are possible!

» **Note:** Industrial robots are often built to have invertible kinematics!

» Block diagram in the start is among the most common approaches.
Content of this Lecture

1. What is a robot?

2. Modeling Robots
   - Kinematics
   - Dynamics

3. Representing Trajectories
   - Splines

4. Control in Joint Space
   - Linear Control
   - Model-based Control

5. Control in Task Space
   - Inverse Kinematics
   - Differential Inverse Kinematics
Differential Inverse Kinematics

Inverse kinematics:
\[ q_d = f^{-1}(x_d) \]
- Not computable as we have an infinite amount of solutions

Differential inverse kinematics:
\[ \dot{q}_t = h(x_d, q_t) \]
- Given current joint positions, compute joint velocities that minimizes the task space error
- Computable
Differential Inverse Kinematics

Differential inverse kinematics:

\[ \dot{q}_t = h(x_d, q_t) \]

How can we use this for control?

1. Integrate \( \dot{q}_t \) and directly use it for joint space control

2. Iterate differential IK algorithm to find \( q_d \)

\[ q_{k+1} = q_k + h(x_d, q_k) \]

and plan trajectory to reach \( q_d \)
Numerical Solution: Jacobian Transpose

⇒ Minimize the task-space error

\[ E = \frac{1}{2} (\mathbf{x} - f(\mathbf{q}))^T (\mathbf{x} - f(\mathbf{q})) \]

⇒ Gradient always points in the direction of steepest ascent

\[
\frac{dE}{dq} = - (\mathbf{x} - f(\mathbf{q}))^T \frac{df(\mathbf{q})}{dq} \\
= - (\mathbf{x} - f(\mathbf{q}))^T \mathbf{J}(\mathbf{q})
\]
Minimize error per gradient descent

- Follow negative gradient with a certain step size $\gamma$

$$\dot{q} = -\gamma \left( \frac{dE}{dq} \right)^T = \gamma J(q)^T(x - f(q))$$

$$= \gamma J(q)^T e$$

- Known as Jacobian Transpose Method
Note:
- This diagram is limited to joint space controllers that require no accelerations (e.g., PD control with gravity compensation).
- If you add additional differentiation (less pleasant than integration), you can use other joint space control laws.
Assume that we are not so far from our solution manifold.

Take smallest step $\dot{q}$ that has a desired task space velocity

$$\dot{x} = \eta(x_d - f(q)) = \eta e$$

Yields the following optimization problem

$$\min \dot{q}^T \dot{q} \quad \text{s.t.} \quad J(q)\dot{q} = \dot{x}$$

Solution: (right) pseudo-inverse

$$\dot{q} = J(q)^T(J(q)J(q)^T)^{-1}\dot{x}$$

$$= \eta J(q)^\dagger e$$
Task-Prioritization with Null-Space Movements

Execute another task $\dot{q}_0$ simultaneously in the “Null-Space”

- For example, “push” robot to a rest-posture
  
  $$\dot{q}_0 = K_P (q_{\text{rest}} - q)$$

- Take step that has smallest distance to “base” task
  
  $$\min_{\dot{q}} (\dot{q} - \dot{q}_0)^T (\dot{q} - \dot{q}_0), \quad \text{s.t.} \quad \dot{x} = J(q)\dot{q}$$

- Solution: 
  
  $$\dot{q} = J^\dagger \dot{x} + (I - J^\dagger J) \dot{q}_0$$

- Null-Space: 
  
  $$(I - J^\dagger J)$$

- All movements $\dot{q}_{\text{null}}$ that do not contradict the constraint
  
  $$\dot{x} = J(q)(\dot{q} + \dot{q}_{\text{null}}) \text{ or } J(q)\dot{q}_{\text{null}} = 0$$
More advanced solutions

Similarly, we can also use a acceleration formulation

**Solution:**  \( \ddot{q} = J^+(\ddot{x} - \dot{J}\dot{q}) + (I - J^+J)\ddot{q}_0 \)

There is a whole class of operational space control laws that can be derived from

\[
\begin{align*}
\min \quad & (u - u_0)^T(u - u_0) \\
\text{s.t.} \quad & A(q, \dot{q}, t)\ddot{q} = \dot{b}(q, \dot{q}, t) \\
& u_0 = g(q, \dot{q}, t) \\
& M(q)\ddot{q} = u + c(q, \dot{q}) + g(q)
\end{align*}
\]

- The resolved acceleration control law with a model-based control law can be derived from this framework.
- For an up-to-date and conclusive treatment, see
Singularity Problems

**Problem:** However, the inversion in the pseudo-inverse

\[ J^\dagger = J^T (JJ^T)^{-1} \]

can be problematic.

In the case of singularities, \( JJ^T \) cannot be inverted!
Damped Pseudo Inverse

Numerically more stable solution:

- Find a tradeoff between minimizing the error and keeping the joint movement small
  \[
  \min_q (\dot{x} - J(q)\dot{q})^T (\dot{x} - J(q)\dot{q}) + \lambda \dot{q}^T \dot{q}
  \]

- Regularization constant \( \lambda \)

- Damped Pseudo Inverse Solution
  \[
  \dot{q} = J^T (J J^T + \lambda I)^{-1} \dot{x} = J^{\dagger}(\lambda) \dot{x}
  \]

- Works much better for singularities
Ask questions...

Q & A?