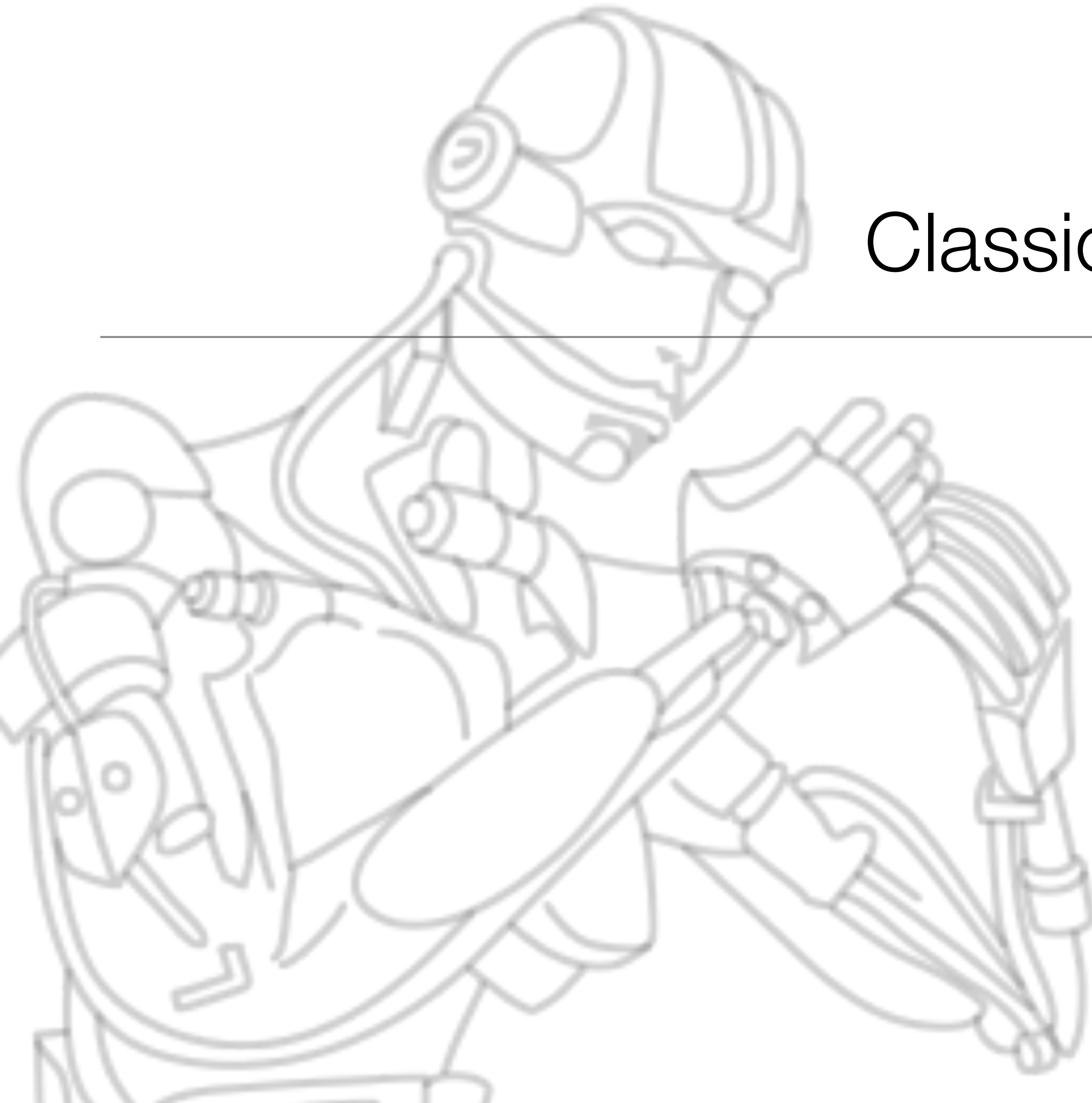
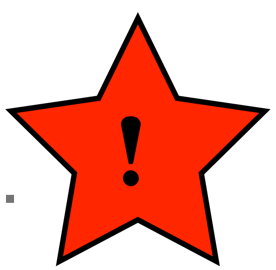


Classical Robotics in Nutshell

Jan Peters
Gerhard Neumann

Purpose of this Lecture

- What you need to know about robotics!
- Important robotics background in a nutshell!
- In order to understand robot learning, we have to understand the problems first
- Essentials are starred...



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

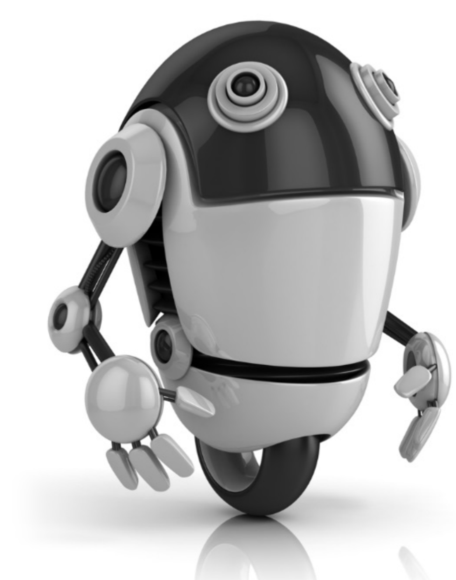
What is a Robot?

A robot is a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks.

Robotics Institute of America

A computer is just amputee robot

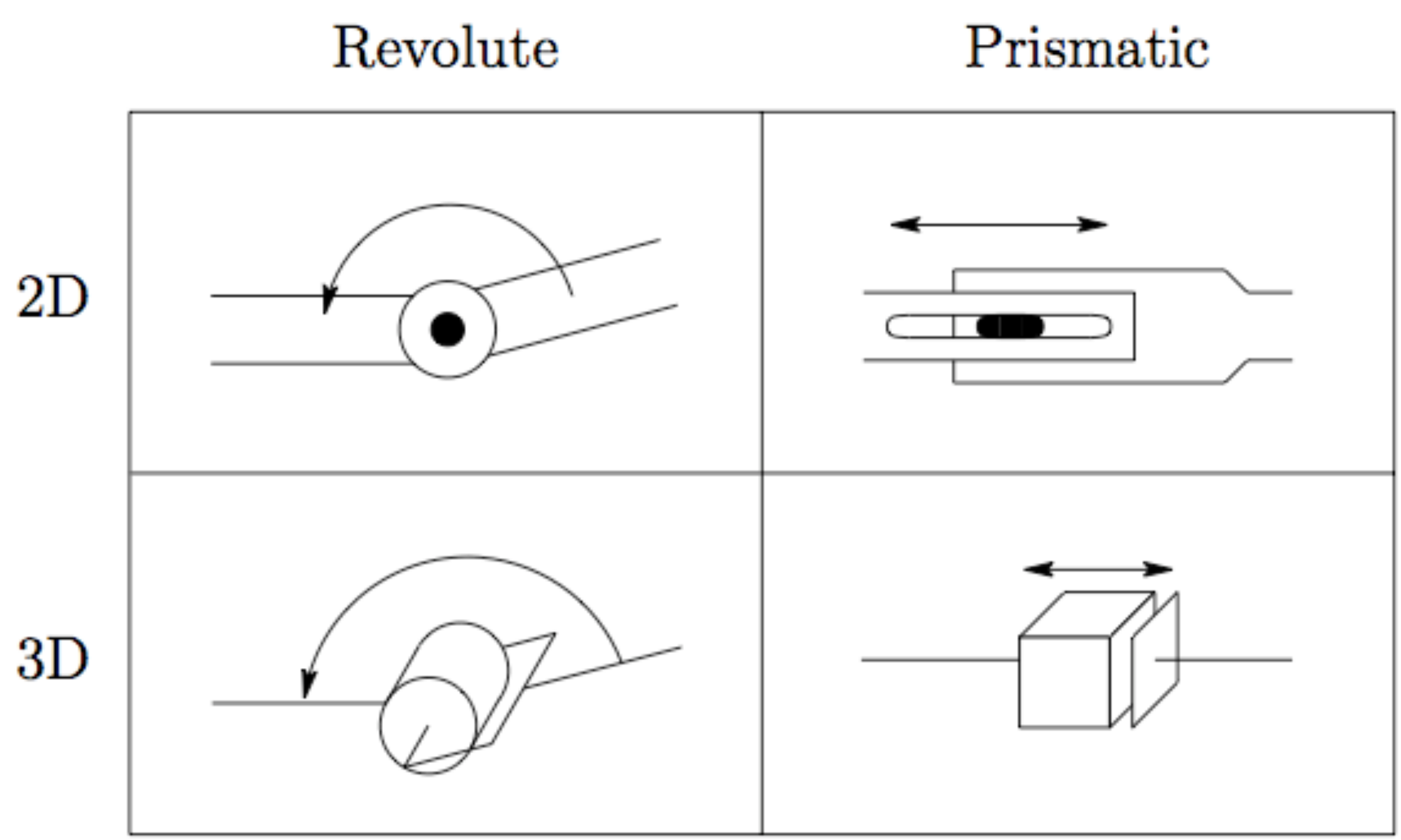
G. Randlov



Modeling: What are the Degrees of Freedom?

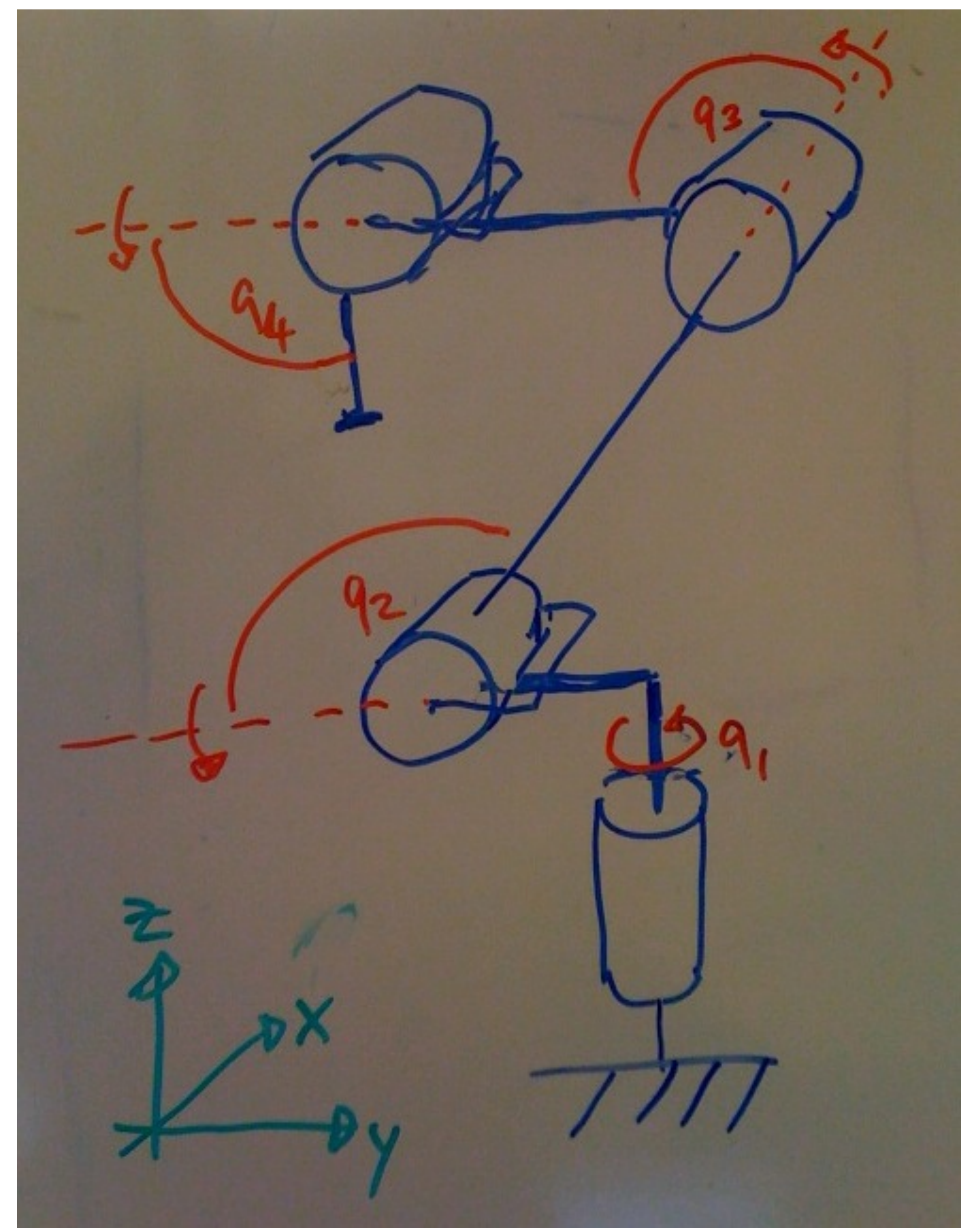
2 types of joints:

- revolute
- prismatic



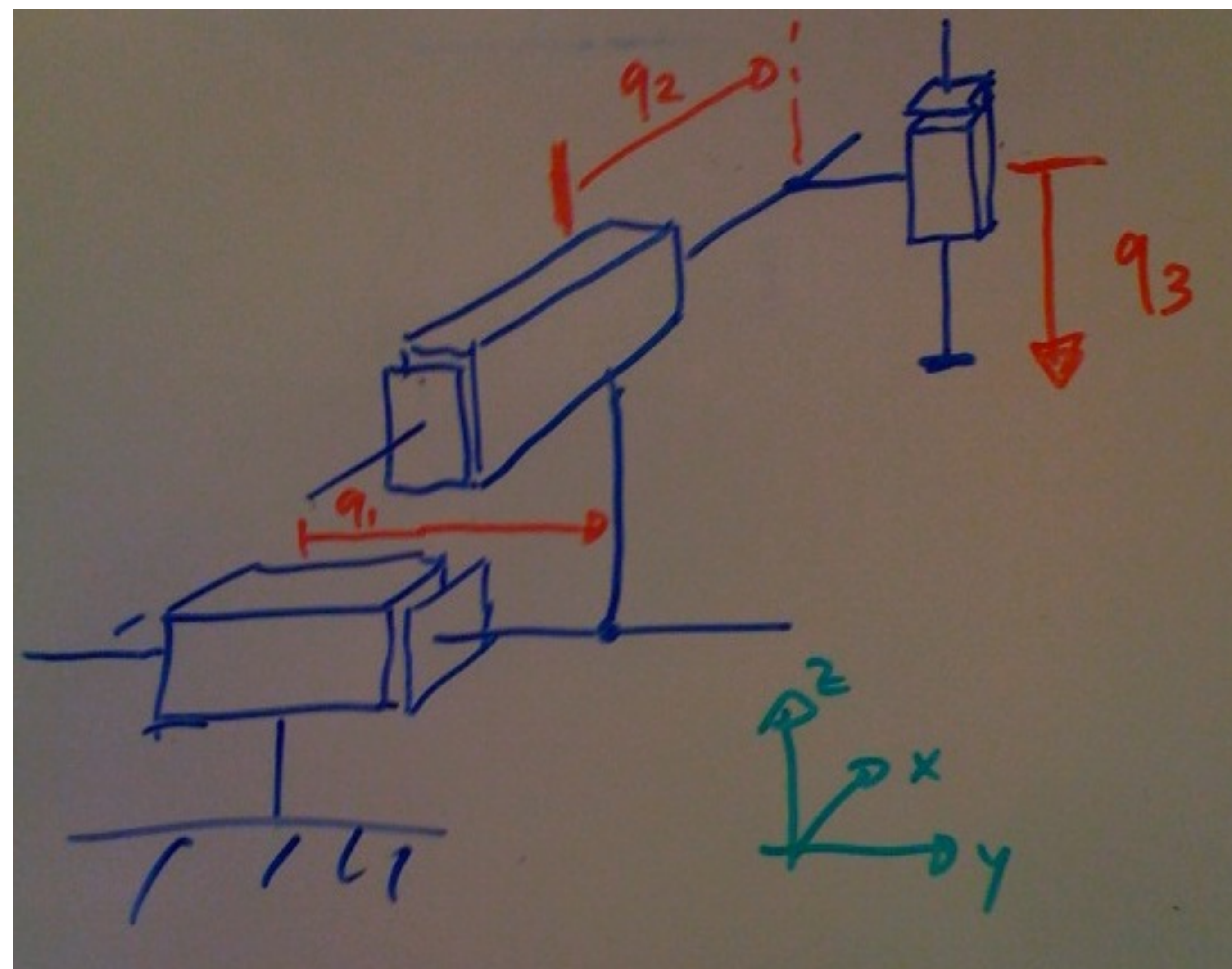
Modeling: What are the Degrees of Freedom?

Revolute joints



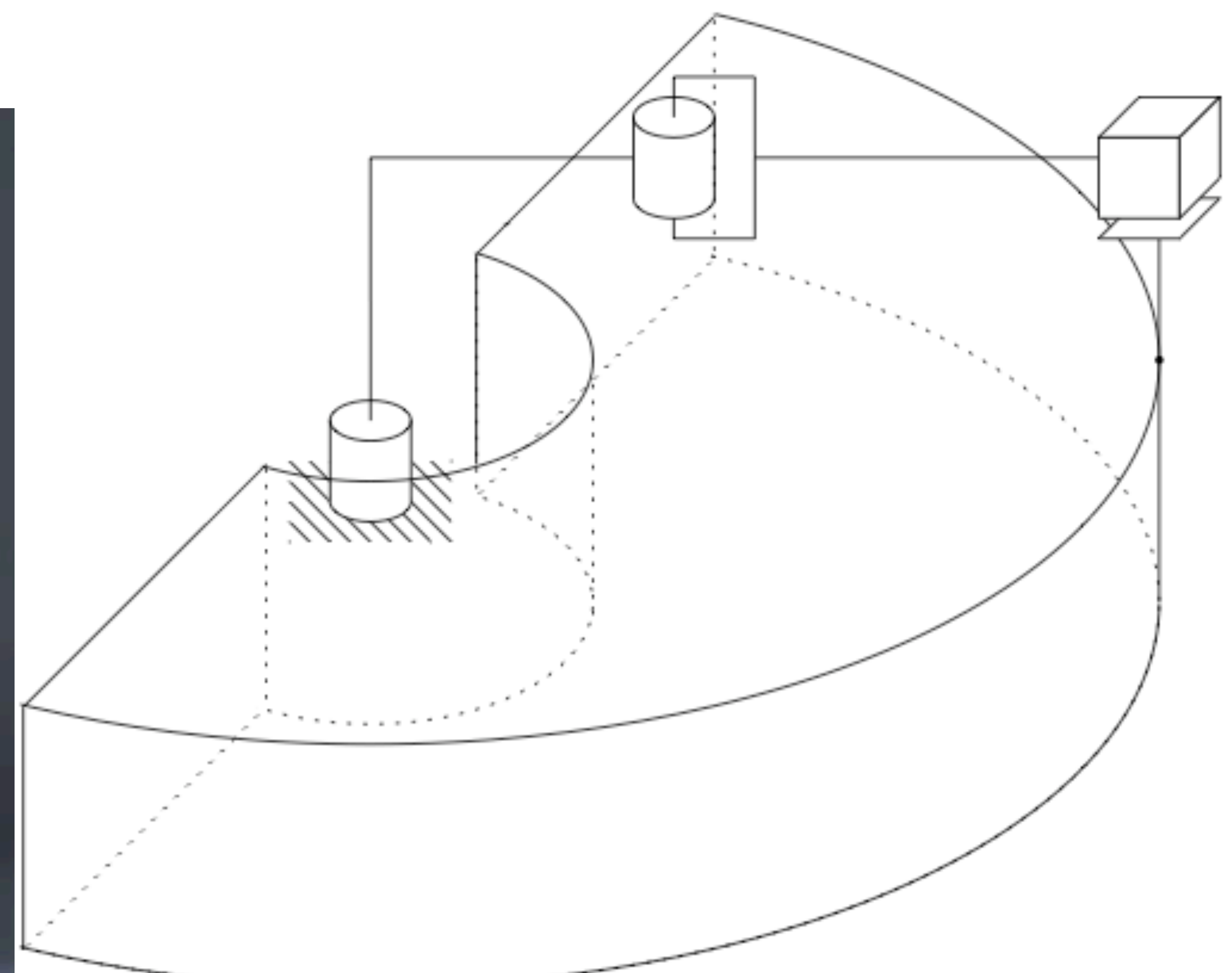
Modeling: What are the Degrees of Freedom?

Prismatic Joints



Workspace

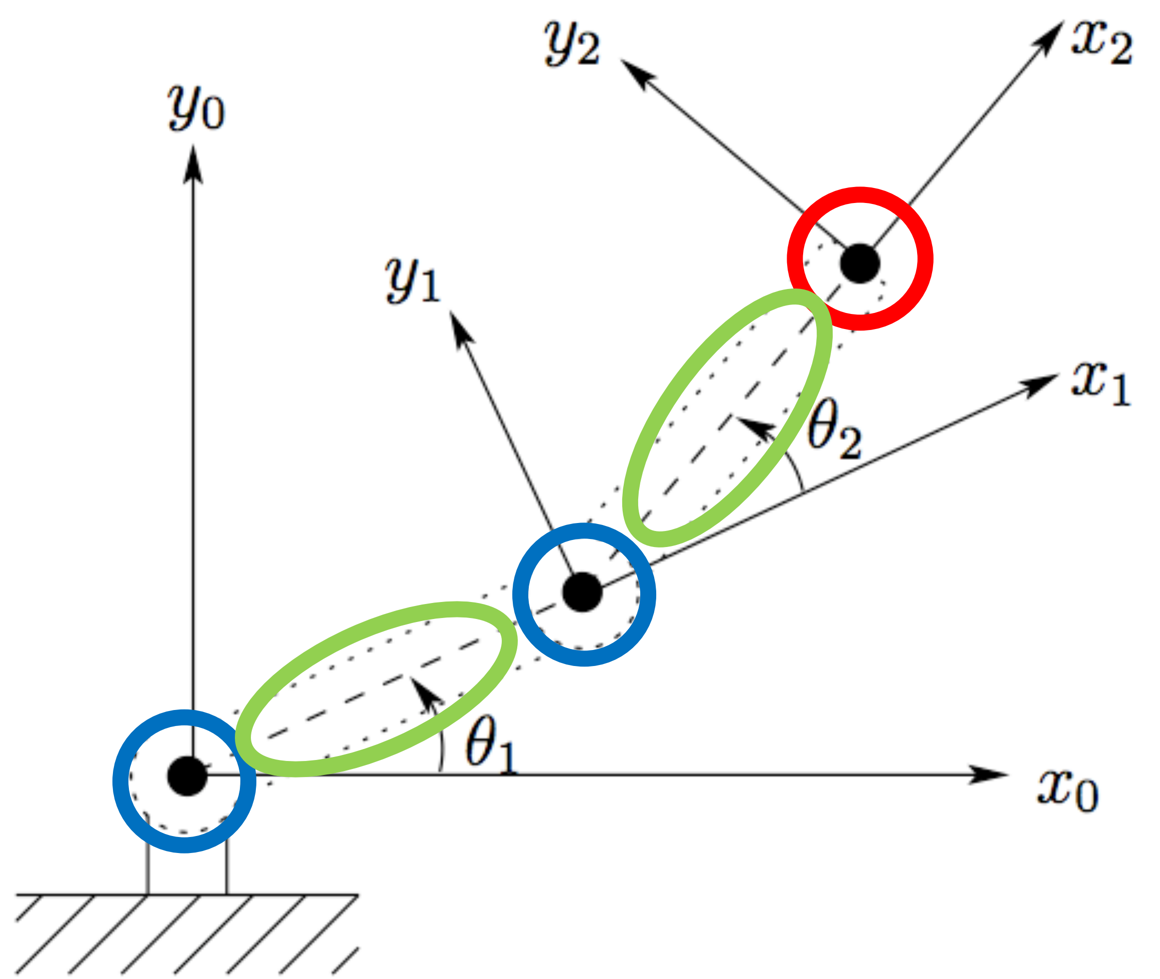
The workspace is the reachable space with the end-effector



Basic Terminology

Link

Joints: $\boldsymbol{q}$ [rad]



Task/Endeffector space: $\boldsymbol{x}$[m]

State (robot and environment): $\boldsymbol{s}$

Basic Terminology

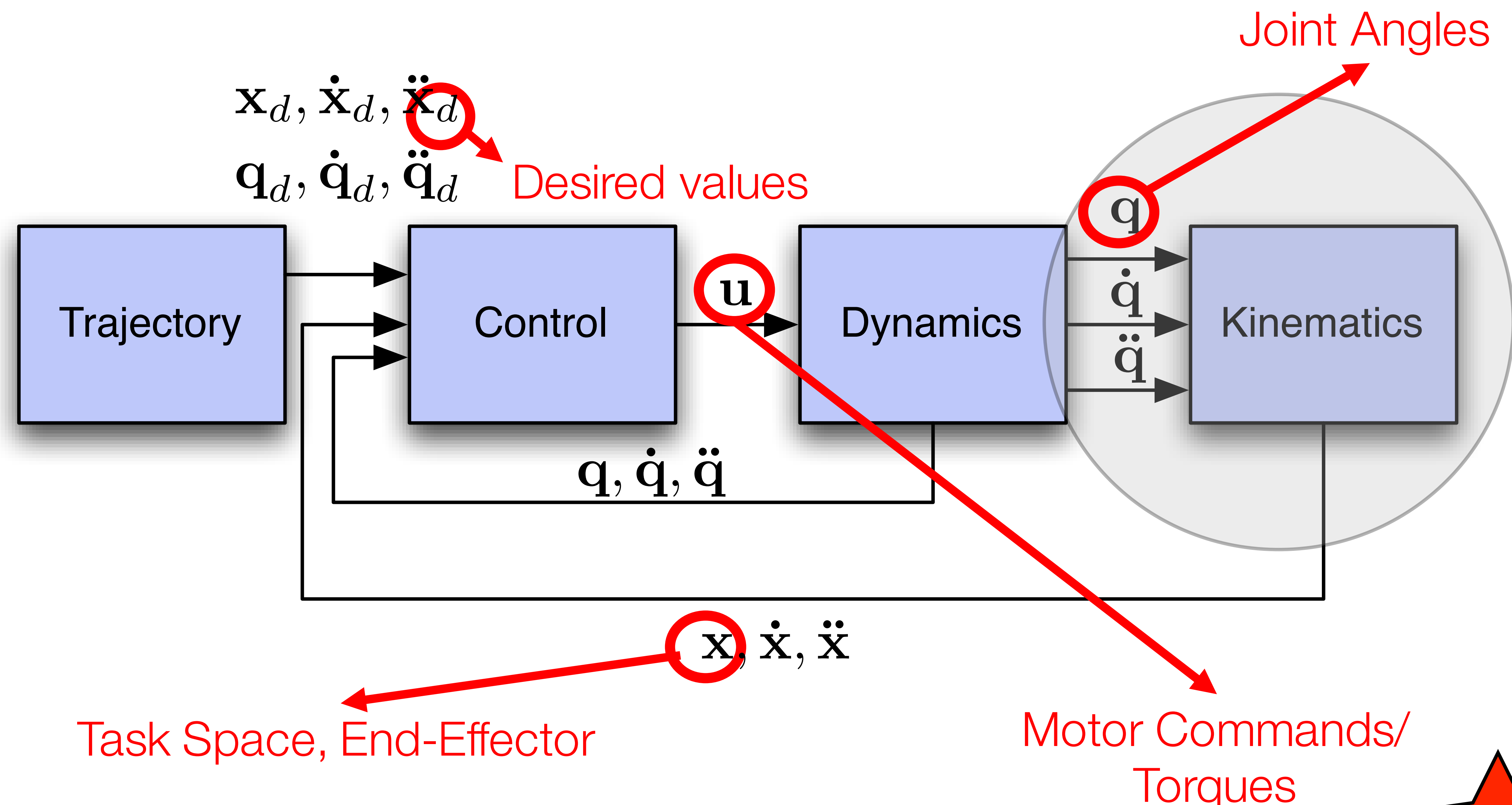
Actions: $\boldsymbol{u}/\boldsymbol{a}$

- In general: Can be velocities, accelerations or torques
- In robotics: they are always in some way mapped to torques

(Control) Policy:

- Deterministic $\boldsymbol{u} = \pi(\boldsymbol{s})$
- Stochastic $\boldsymbol{u} \sim \pi(\boldsymbol{u}|\boldsymbol{s})$

Block Diagram of Complete System



Content of this Lecture

1. What is a robot?

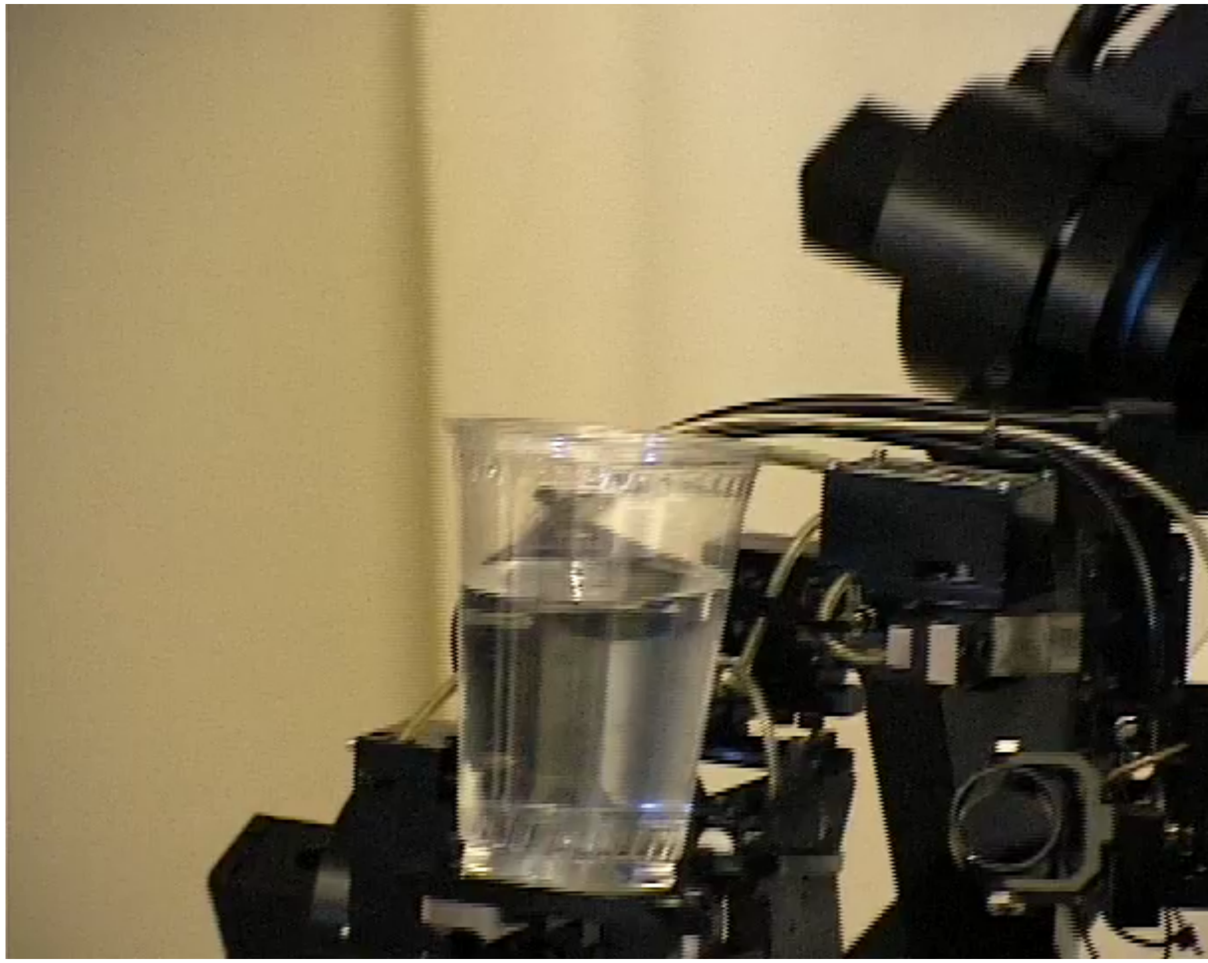
2. **Modeling Robots**
   - **Kinematics**
   - Dynamics

3. Representing Trajectories
   - Splines

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

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

Kinematics



Where is my hand/endeffector & what is it's orientation?

Where is my center of gravity?

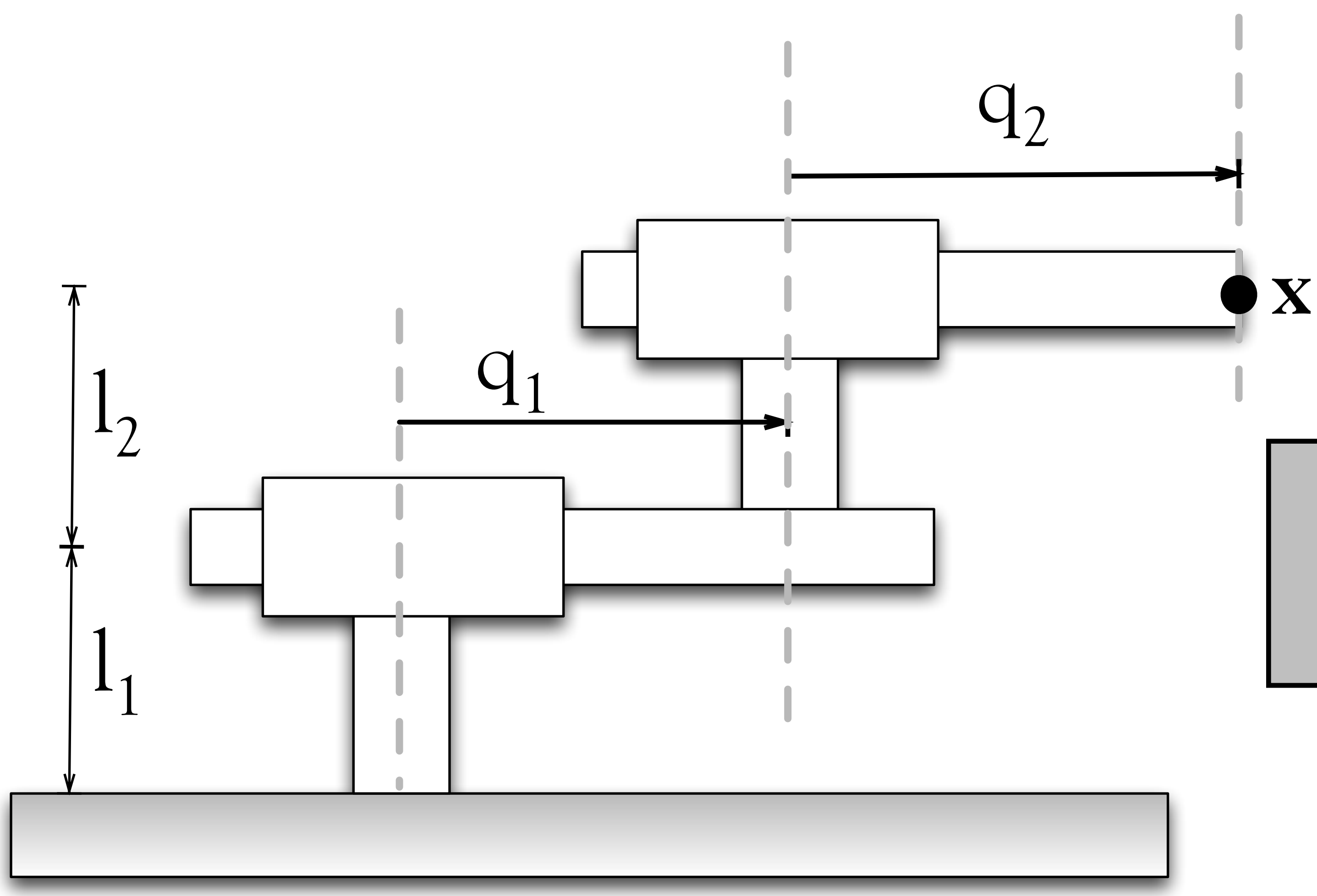
What do we want to have?

Forward Kinematics: A mapping from joint space to task space

$$\mathbf{x} = f(\mathbf{q})$$

Example 1: Prismatic Robot with 2 DoF

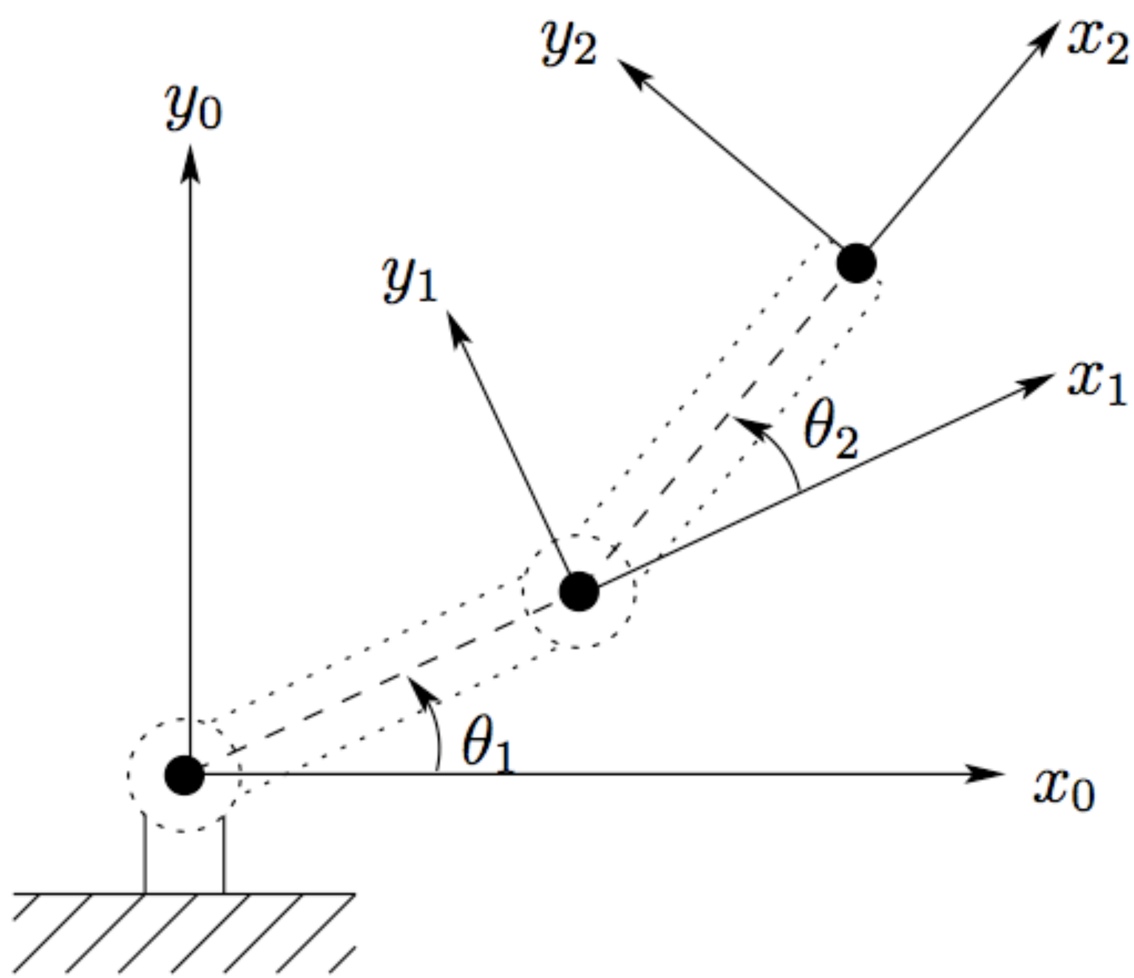
What are the forward kinematics $\mathbf{x} = f(\mathbf{q})$ **?**



$$x = q_1 + q_2$$

Example 2: Rotary Robot with 2 DoF

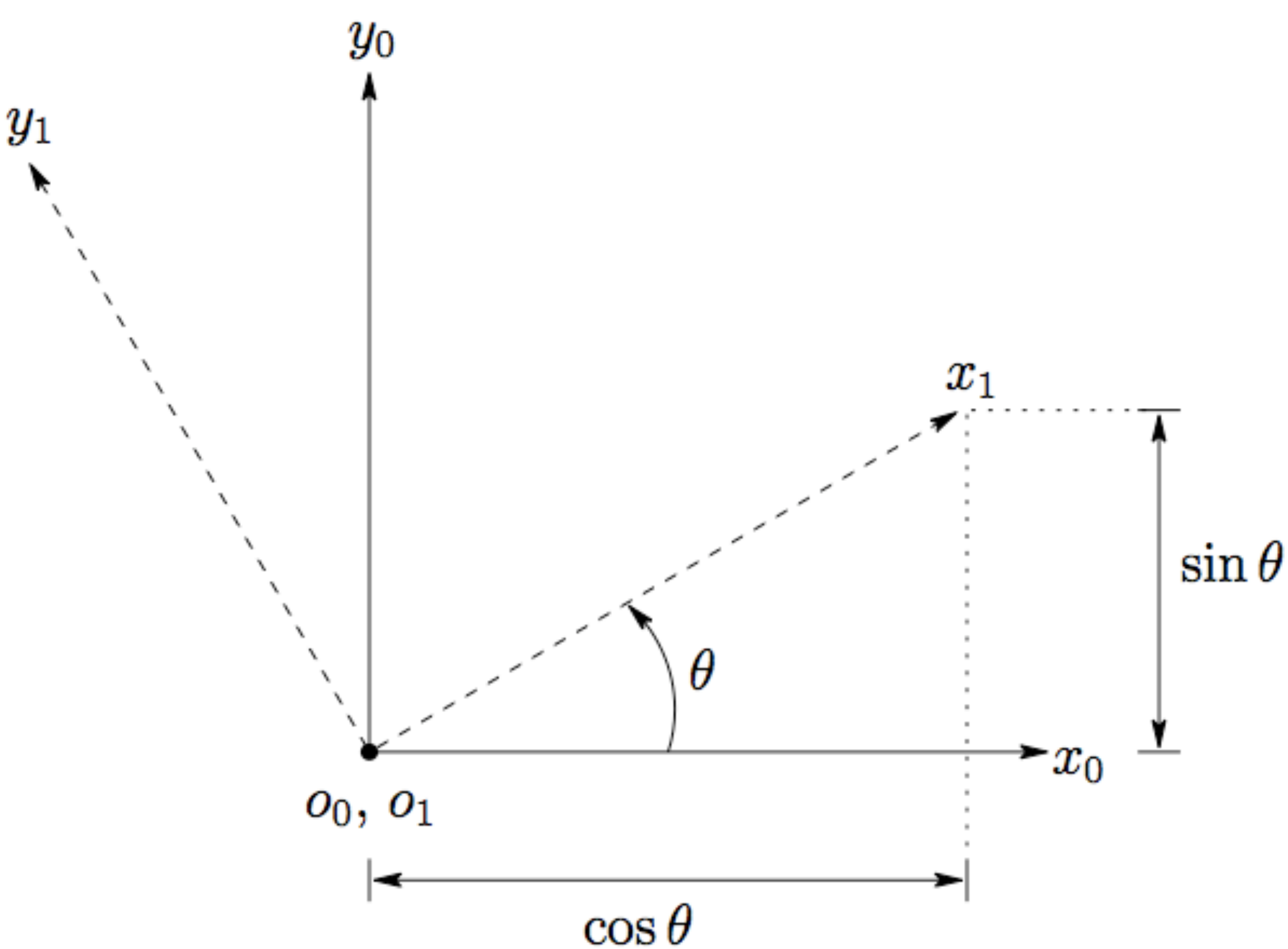
What are the forward kinematics $\mathbf{x} = f(\mathbf{q})$ **?**



$$
\boxed{
\begin{aligned}
x &= x_2 = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \\
y &= y_2 = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2)
\end{aligned}
}
$$

What does a “Rotation” mean?

A rotation is a transformation of coordinate frames



Can we write the transformation as matrix multiplication?

- We want a matrix such that

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \mathbf{R}(\theta) \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Which matrix fulfills this?

- We know that:

$$\mathbf{e}_x^1 = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} = \mathbf{R}(\theta)\mathbf{e}_x^0$$

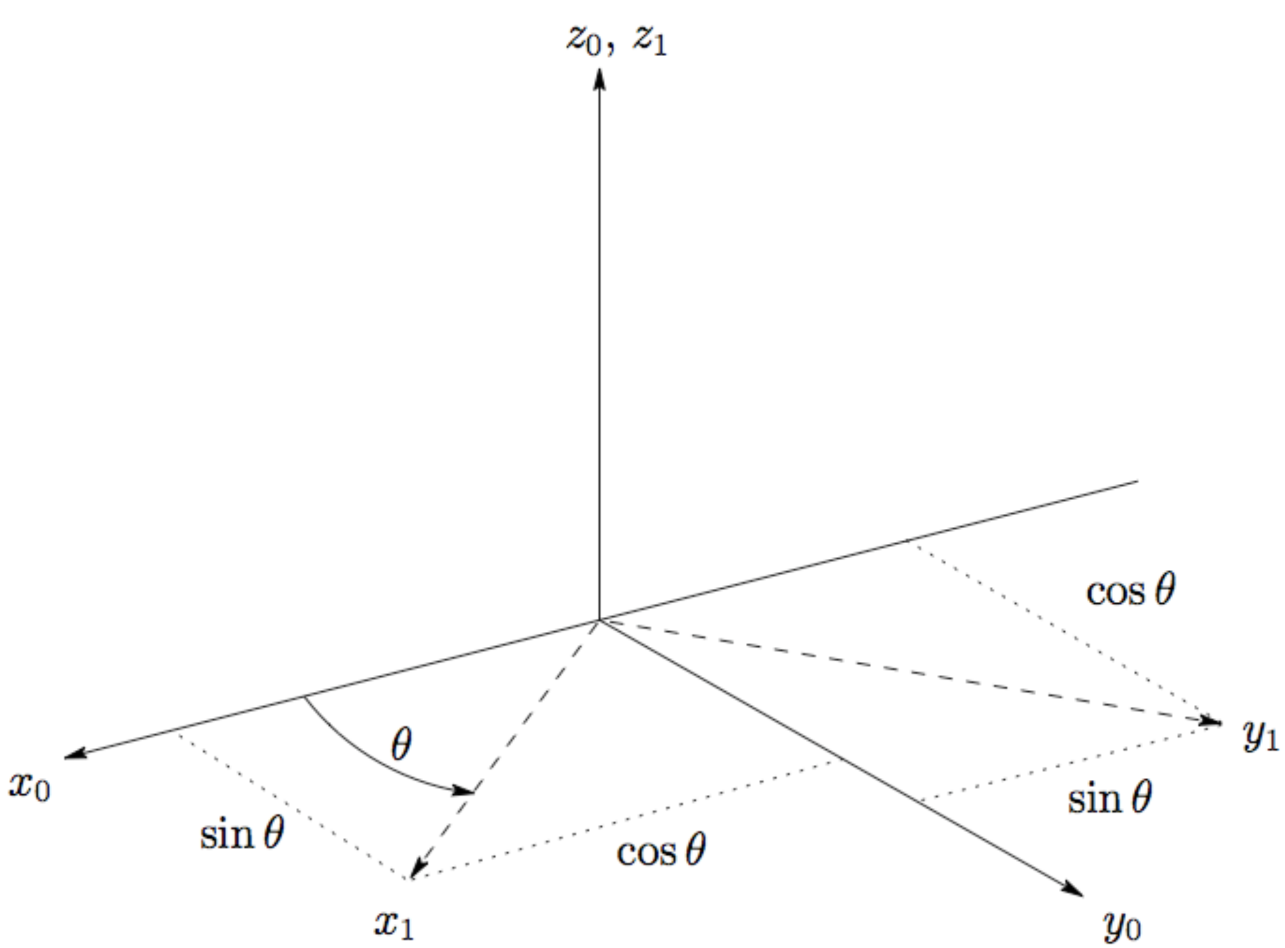
$$\mathbf{e}_y^1 = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} = \mathbf{R}(\theta)\mathbf{e}_y^0$$

- Hence, we have

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Rotations in 3D

Rotations in 3D require rotating about any axis:



$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

It's just like 2D, just add an identity for the axis around which you are rotating.

More about Rotations ...

Rotations can be stacked:

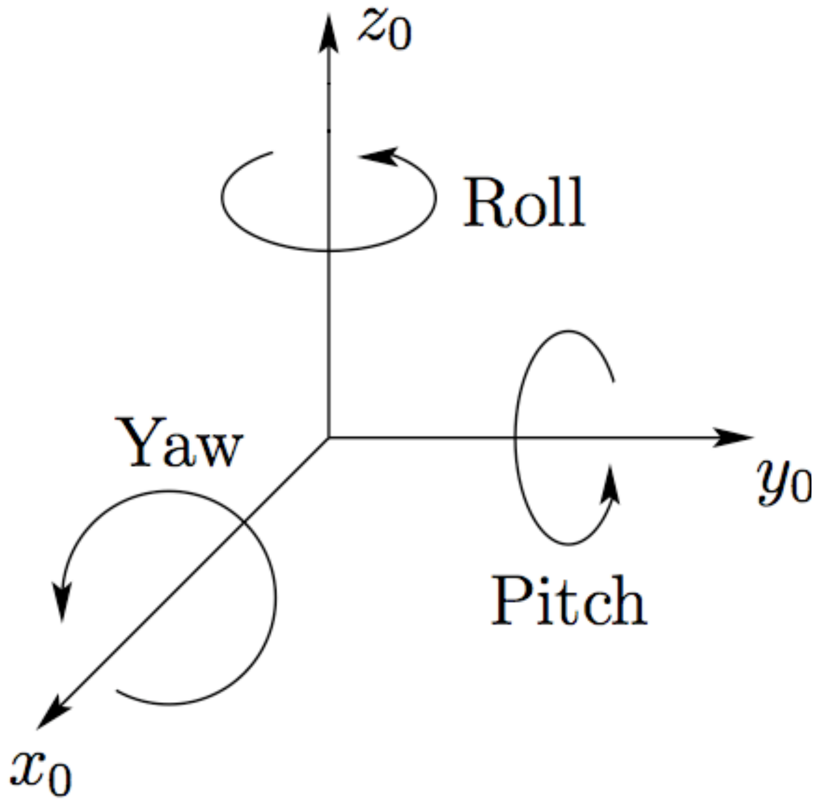
$$\begin{aligned} p^0 &= R_1^0 p^1 \\ p^1 &= R_2^1 p^2 \end{aligned} \quad \Rightarrow \quad \begin{aligned} p^0 &= R_2^0 p^2 = R_1^0 R_2^1 p^2 \\ R_2^0 &= R_1^0 R_2^1 \end{aligned}$$

Other basic facts: Orthonormality!

$$R^{-1} = R^T \qquad \det\{R\} = 1$$

Representation of Rotations

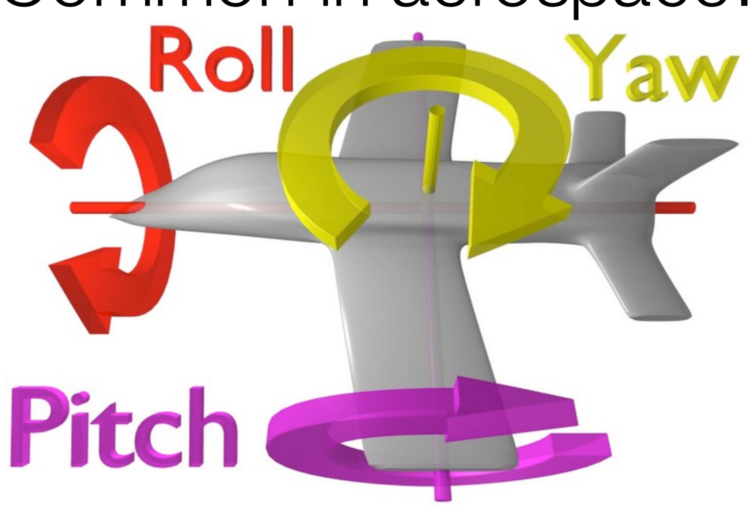
Euler Angles: Roll-Pitch-Yaw Representation



$$
\begin{aligned}
R_1^0 &= R_{z,\phi} R_{y,\theta} R_{x,\psi} \\
&= \begin{bmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{bmatrix} \\
&= \begin{bmatrix} c_\phi c_\theta & -s_\phi c_\psi + c_\phi s_\theta s_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ s_\phi c_\theta & c_\phi c_\psi + s_\phi s_\theta s_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi \\ -s_\theta & c_\theta s_\psi & c_\theta c_\psi \end{bmatrix}.
\end{aligned}
$$

c_ϕ, s_ϕ… short form for $\sin(\phi), \cos(\phi)$

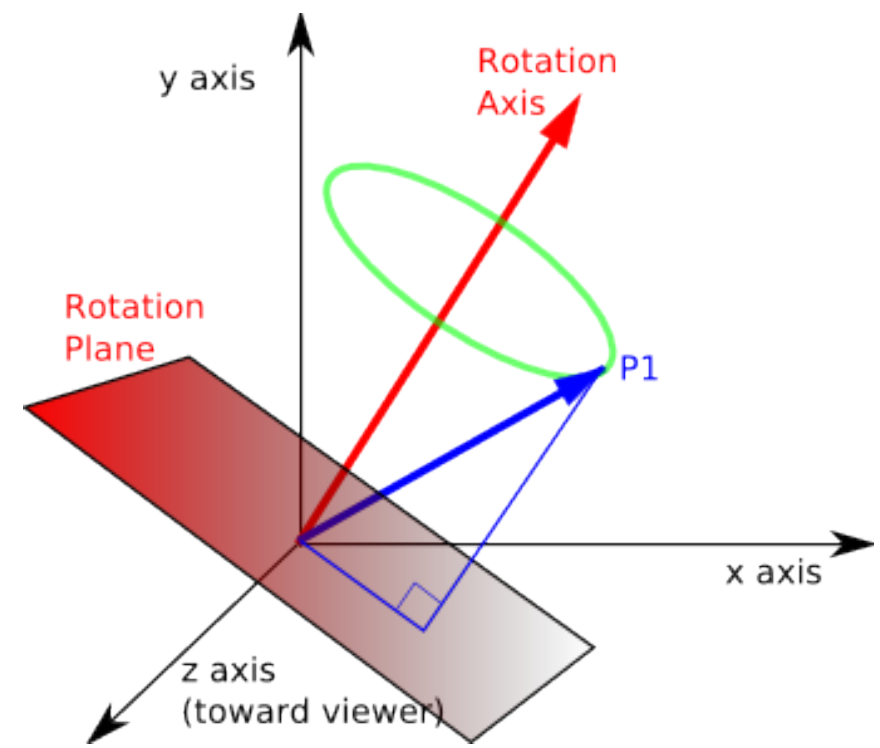
Common in aerospace…



Problems with Euler Angles:

- Not Unique: Many angles result in the same rotation
- Hard to quantify differences between two Euler Angles

Representation of Rotations



Other Types of Representations:

- Angle-Axis
- Unit-Quaternion

Solves the problems of singularities with the Euler Angles

- Easier to compute differences of orientations
- Important if we want to control the orientation of the end-effector

See Siciliano or Spong Textbook!

Homogeneous Transformations

- Translations alone are easy $\mathbf{p}^0 = \boldsymbol{\delta}^0 + \mathbf{p}^1$

- Combining Translation and Rotation is a mess...

$$\boldsymbol{p}^0 = \boldsymbol{\delta}^0 + \boldsymbol{R}_1^0(\boldsymbol{\delta}^1 + \boldsymbol{R}_2^1(\boldsymbol{\delta}^2 + \boldsymbol{R}_3^2\boldsymbol{p}^3)))$$

- ...but a trick solves this mess: Homogeneous Transformations!

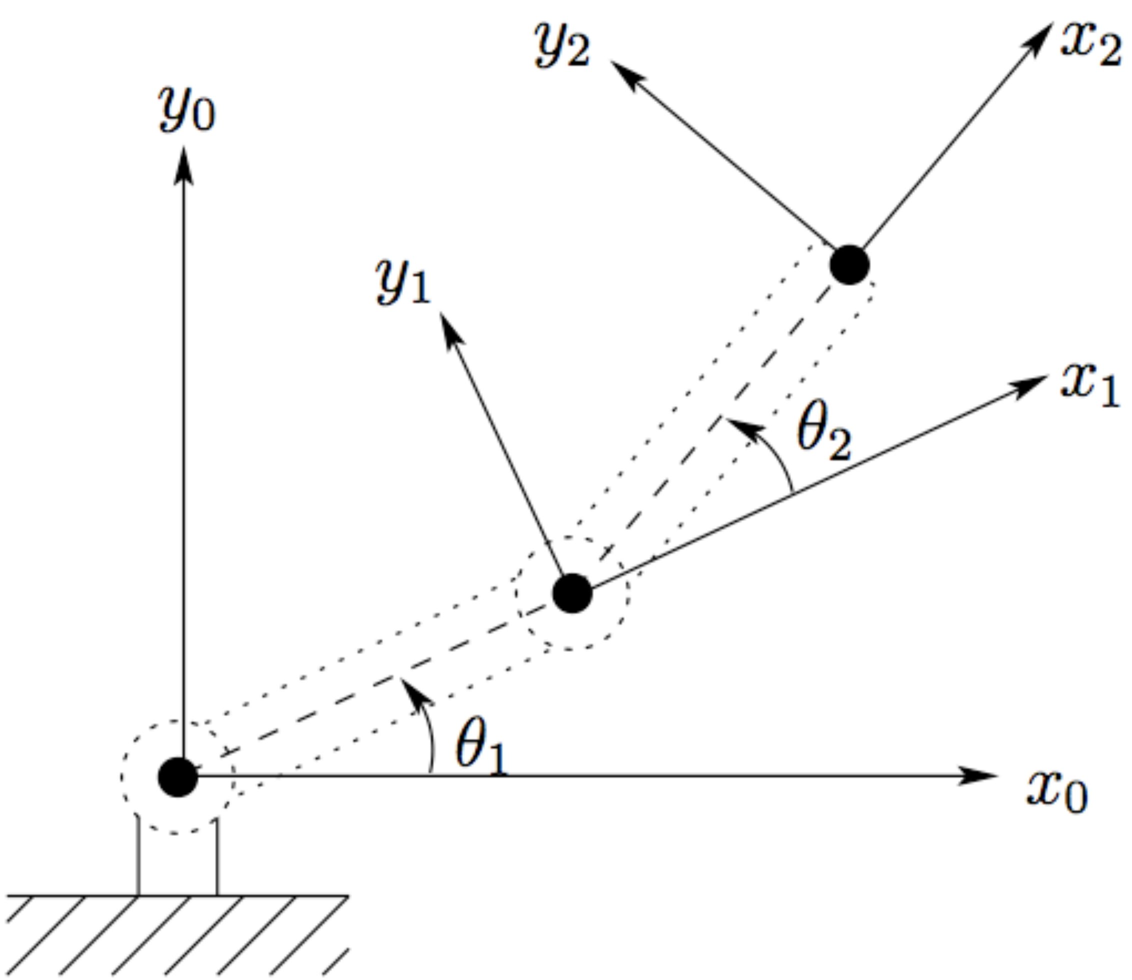
$$\boldsymbol{p}^0 = \boldsymbol{\delta}^0 + \boldsymbol{R}_1^0\boldsymbol{p}^1 \quad \Rightarrow \quad \begin{bmatrix} \boldsymbol{p}^0 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_1^0 & \boldsymbol{\delta}^0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{p}^1 \\ 1 \end{bmatrix}$$

$$= \boldsymbol{H}_1^0 \tilde{\boldsymbol{p}}^1$$

4x4 Transformationmatrix

- Hence, we have: $\tilde{\boldsymbol{p}}^0 = \boldsymbol{H}_1^0 \boldsymbol{H}_2^1 \ldots \boldsymbol{H}_n^{n-1} \tilde{\boldsymbol{p}}^n$

Example 2 - *revisited!*



Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1^*
2	a_2	0	0	θ_2^*

$$\mathbf{A}_1 = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

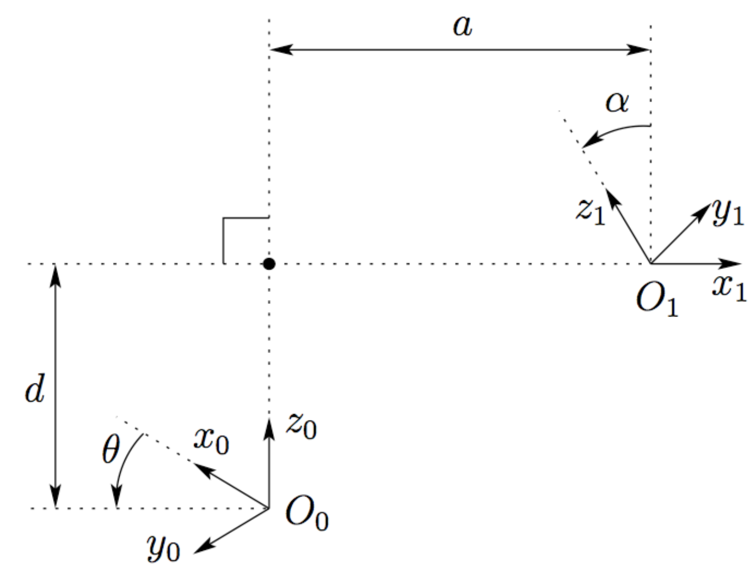
$$\mathbf{A}_2 = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_1^0 = \boldsymbol{A}_1$$

$$\boldsymbol{H}_2^0 = \boldsymbol{A}_1\boldsymbol{A}_2$$

Typical Robot Description: Denavit Hartenberg

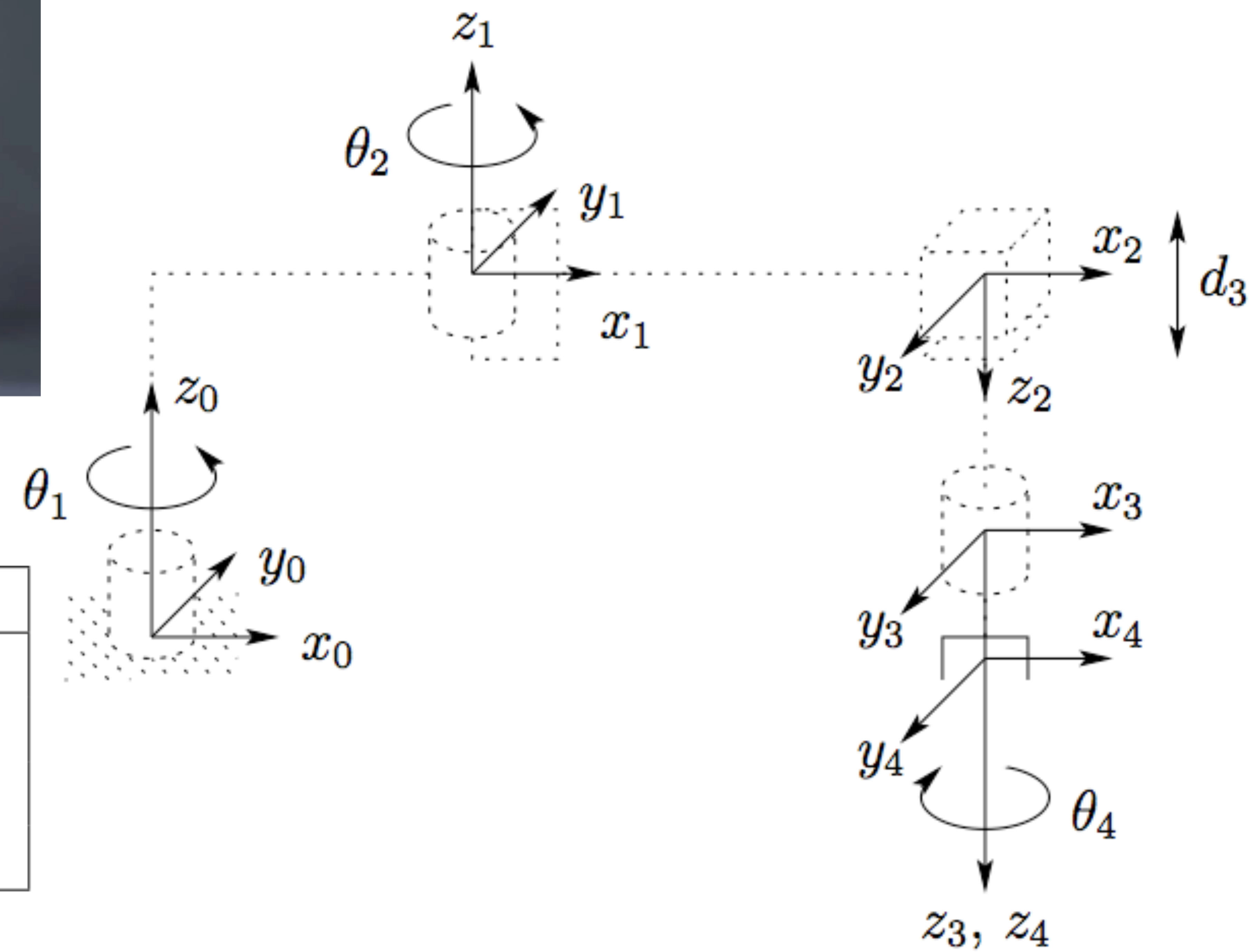
Denavit-Hartenberg Description:



➡ Just four steps with Homogeneous Transformations!

$$
\begin{aligned}
A_i &= Rot_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}Rot_{x,\alpha_i} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Excercise: SCARA



Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	$\star$
2	a_2	180	0	$\star$
3	0	0	$\star$	0
4	0	0	d_4	$\star$

Differential Forward Kinematics

Sometimes, we are interested in the velocity $\dot{\mathbf{x}}$ or acceleration $\ddot{\mathbf{x}}$

Remember chain rule from high school?

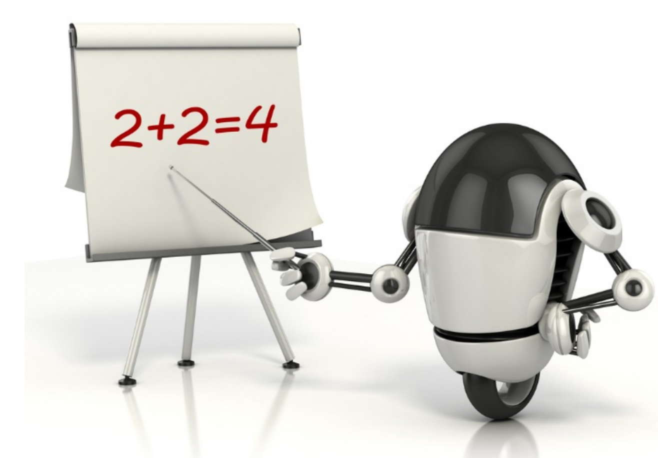
Velocity:

$$\dot{\boldsymbol{x}} = \frac{d}{dt} f(\boldsymbol{q}) = \frac{df(\boldsymbol{q})}{d\boldsymbol{q}} \frac{d\boldsymbol{q}}{dt} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$

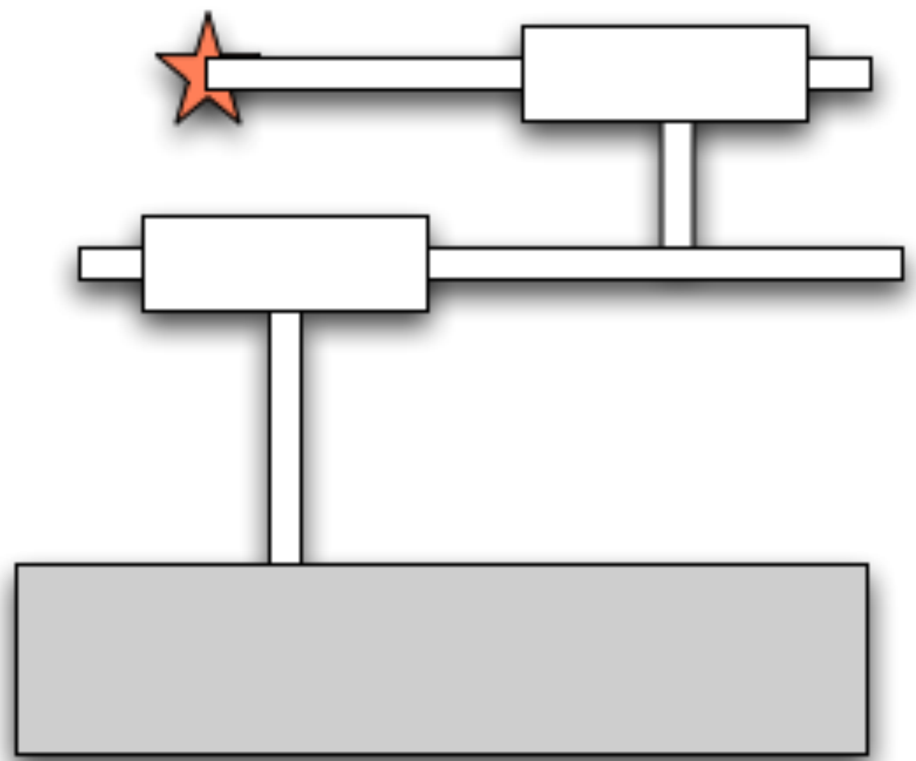
$$\boldsymbol{J}(\boldsymbol{q}) = \frac{df(\boldsymbol{q})}{d\boldsymbol{q}} \ \ldots \ \text{Jacobian}$$

Acceleration:

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

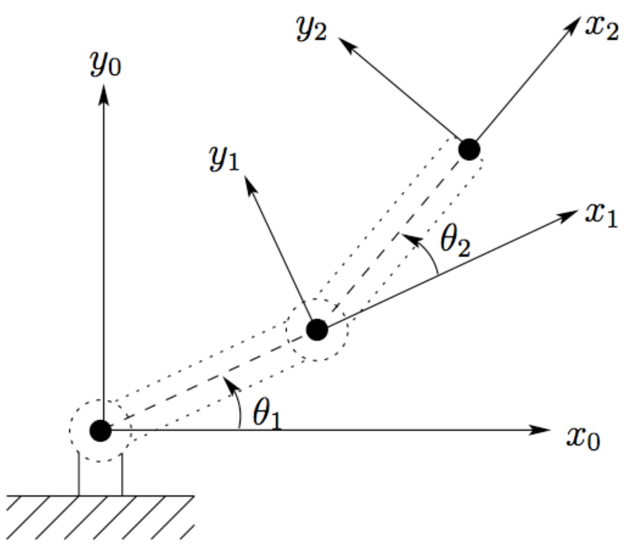


Example 1 - *revisited*



$$
\begin{aligned}
x &= q_1 + q_2 \\
\dot{x} &= \dot{q}_1 + \dot{q}_2 \\
&= [1, 1] \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \mathbf{J}\dot{\mathbf{q}}
\end{aligned}
$$

Examples 2 - *revisited*



$$\boxed{\begin{aligned} x &= x_2 = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \\ y &= y_2 = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) \end{aligned}}$$

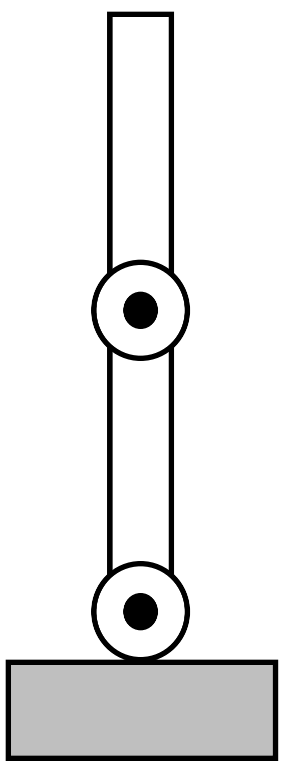
$$\begin{aligned} \dot{x} &= -a_1 \sin\theta_1 \dot{\theta}_1 - a_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ \dot{y} &= a_1 \cos\theta_1 \dot{\theta}_1 + a_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \end{aligned}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -a_1 \sin(\theta_1) - a_2 \sin(\theta_1 + \theta_2) & -a_2 \sin(\theta_1 + \theta_2) \\ a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) & +a_2 \cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$

Singularities

- What happens when I stretch out my arm?

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -(a_1 + a_2)\sin(\theta_1) & -a_2 \sin(\theta_1) \\ (a_1 + a_2)\cos(\theta_1) & +a_2 \cos(\theta_1) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$



- The columns of the Jacobian get linearly dependent
- I lose a degree of freedom and

$$\det \mathbf{J} = 0$$

- These positions are called ***Singularities***!

Computing the Jacobians

Two ways are common:

- **Analytical Jacobians** are easier to understand (as before) and can be derived by symbolic differentiation. However, the representation of the rotation matrix can cause “representational singularities”

- **Geometric Jacobians** are derived from geometric insight (more contrived), can be implemented easier and do not have “representational singularities”.

- **Main difference:** How the Jacobian for the orientation is represented

See the Spong or Siciliano Textbook...

Content of this Lecture

1. What is a robot?

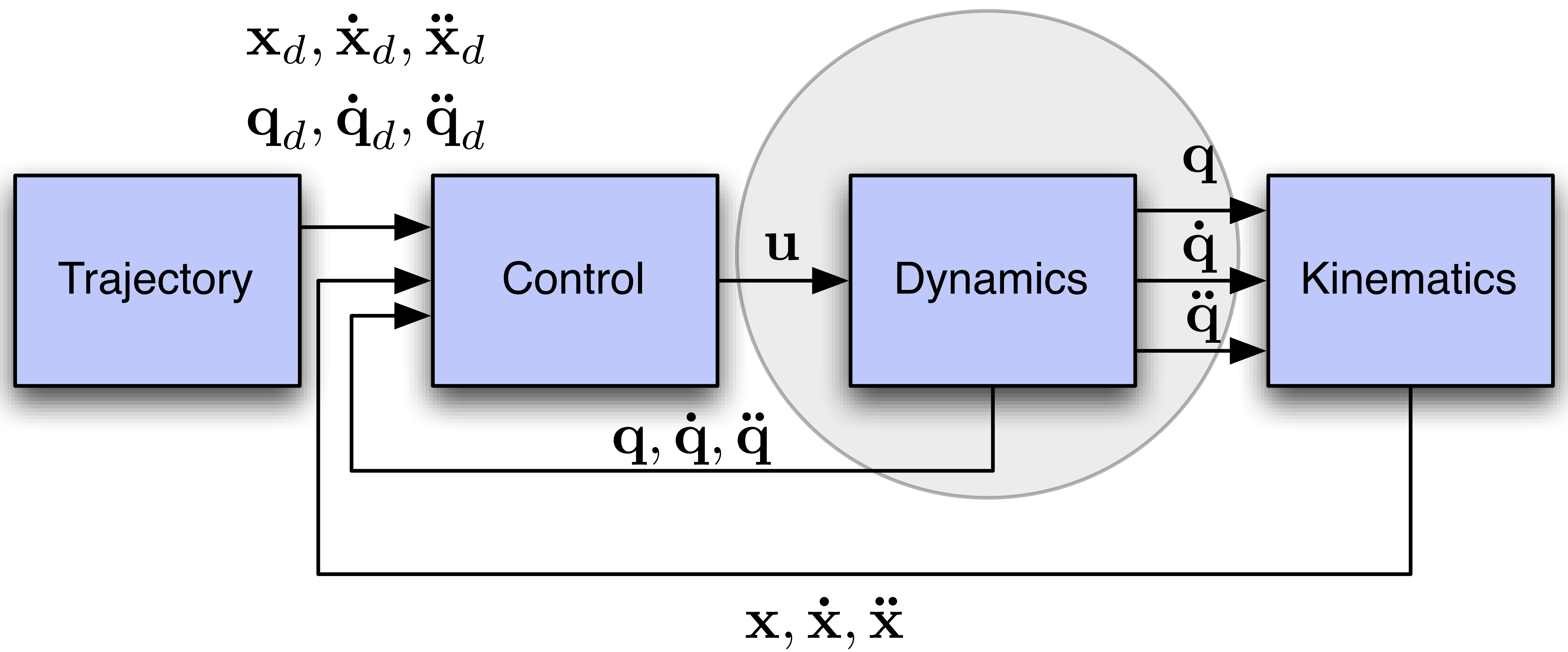
2. Modeling Robots
- **Kinematics**
- **Dynamics**

3. Representing Trajectories
- Splines

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

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

Block Diagram of Complete System



Dynamics

Goal: Obtain a forward dynamics model

$$\ddot{\boldsymbol{q}} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$$

Essential equations:

1. Forces F_i (Kraft):

$$\text{mass} \leftarrow m\ddot{x} = \sum_i F_i$$

1. Torques τ_i (Drehmoment):

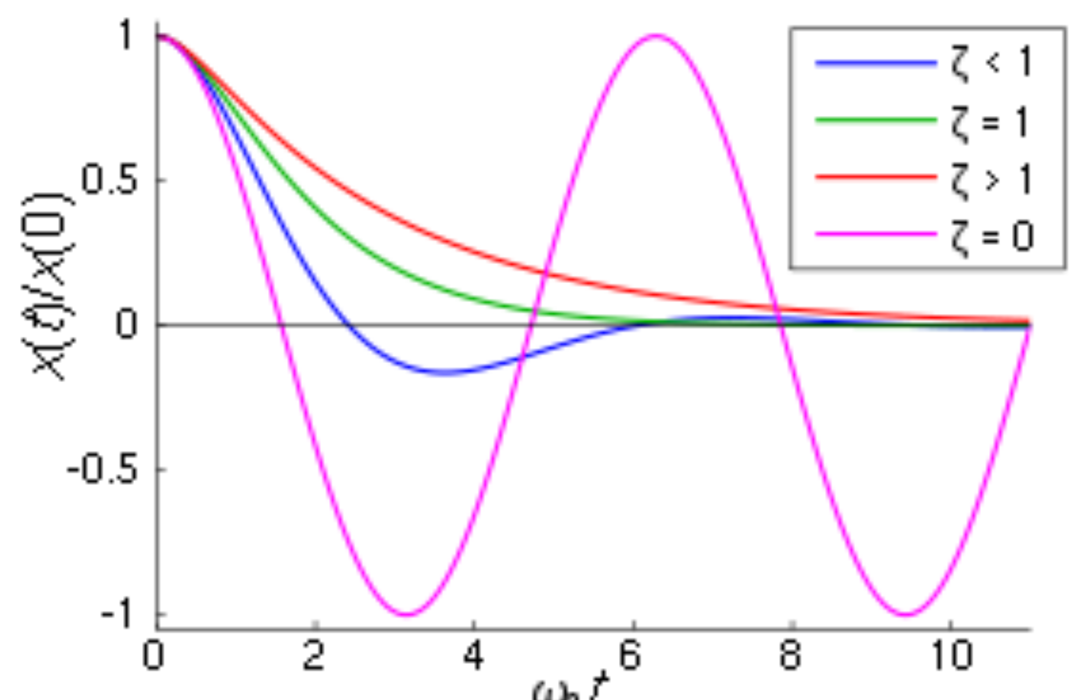
$$\text{Inertia} \leftarrow I\ddot{q} = \sum_i \tau_i$$

!

What forces are there?

- **Gravity:** $F_{\text{grav}} = mg$
- **Friction**
  - Stiction: $F_{\text{stiction}} = -c_s \text{sgn}(\dot{x})$
  - Damping (Viscous Friction): $F_{\text{damping}} = -D\dot{x}$
- **Springs:**
- **Example:** Spring-Damper System

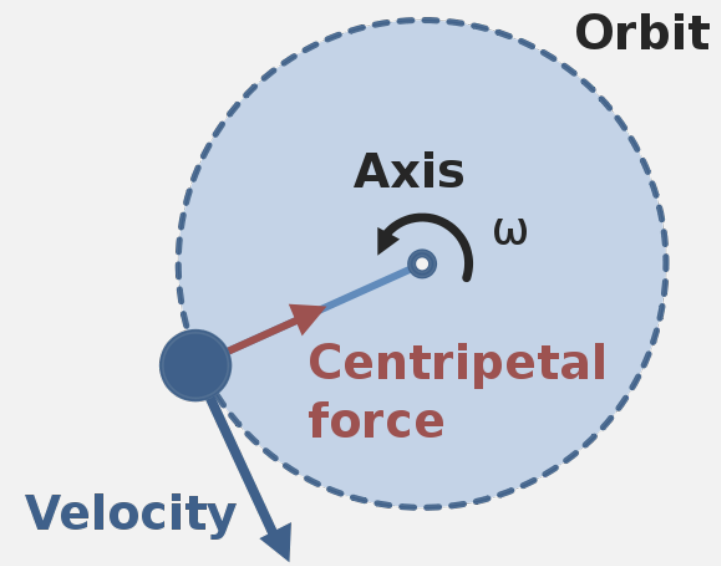
$$m\ddot{x} = K(x_{\text{eq}} - x) - D\dot{x}$$



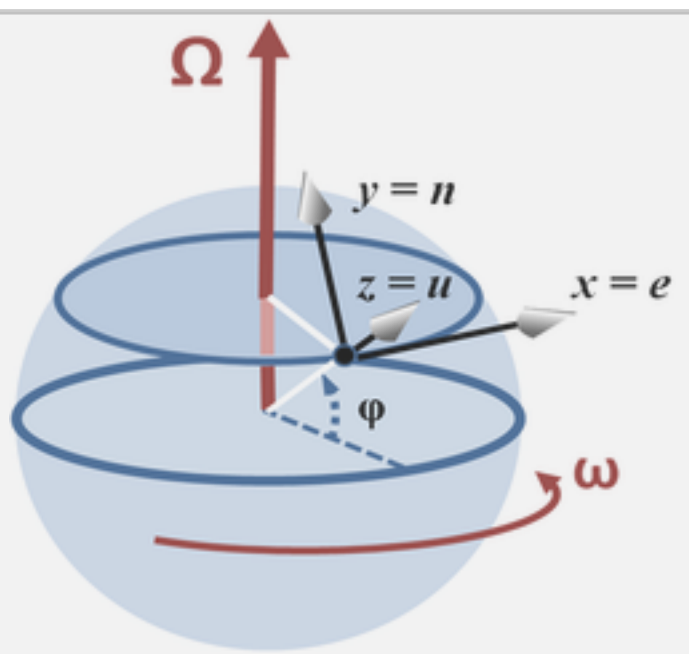
What torques are there?

- Gravity $\boldsymbol{\tau}_{\mathbf{gravity}} = mgl$
- Friction just as before.
- **Virtual Forces:**
  - Centripetal
  - Coriolis forces

Centripetal Forces



Coriolis Forces



General Form

Dynamics are usually denoted in this form:

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

- Motor commands: $\boldsymbol{u}$
- Joint positions, velocities and accelerations: $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$
- Mass matrix: $\boldsymbol{M}(\boldsymbol{q})$
- Coriolis forces and Centripetal forces: $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$
- Gravity: $\boldsymbol{g}(\boldsymbol{q})$

Where do I get these Forces/Torques from?

Friction? No general recipe!

Rigid body forces $\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$?

- Newton-Euler's Method

  1. Manually by Force Dissection ("Freischneiden", see Technical Mechanics 1)

  2. Can be formalized nicely! See Oskar's class for details…

- Lagrangian Method

!

Short break - time for feedback?

I appreciate FEEDBACK!

Too slow?

Too fast?

Too much fun?

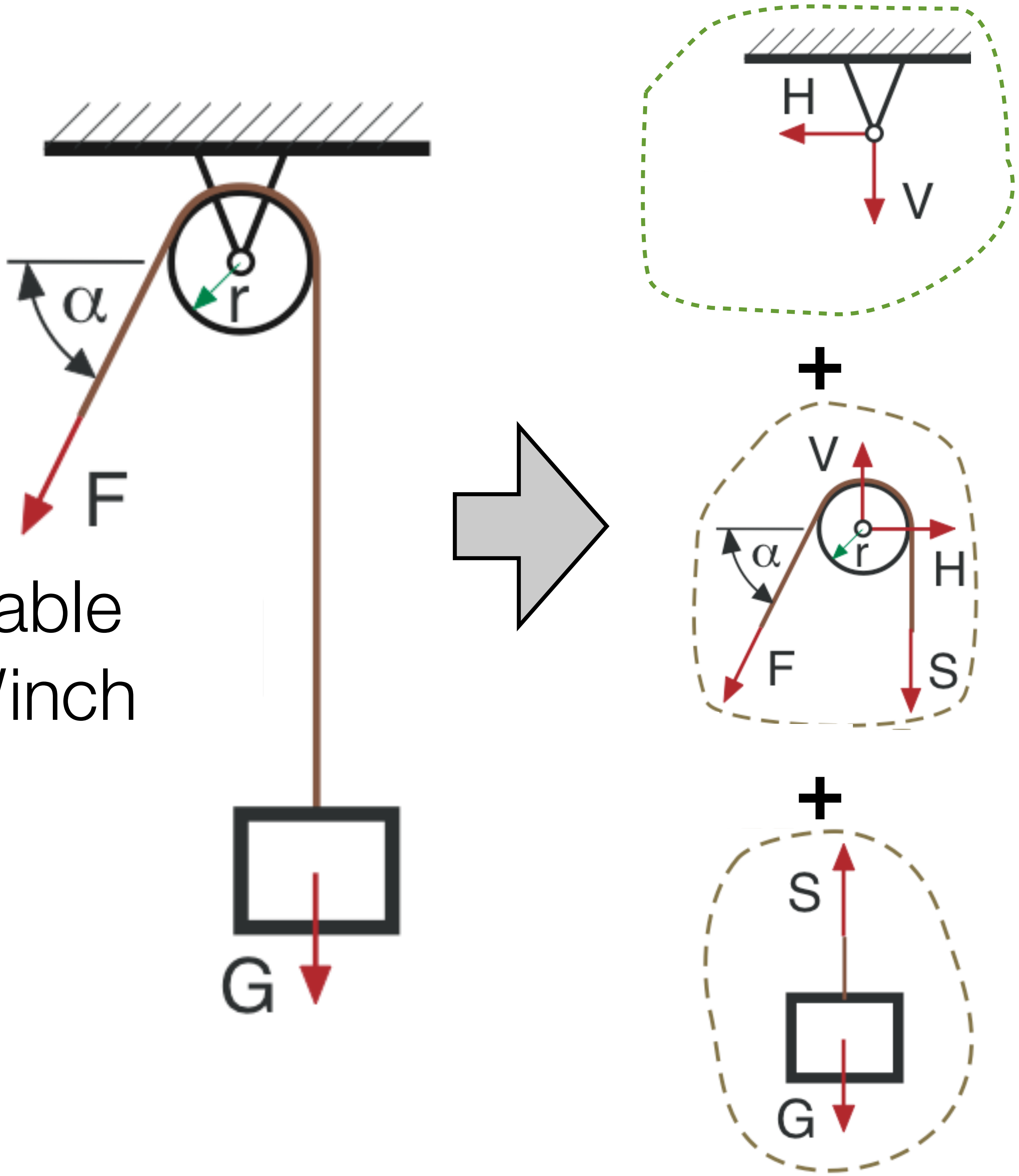
Not enough

Who is that guy in the front and why is he talking so

Jeder Prof hat ‘ne Meise. Meine duerfen Sie fuettern!

Newton-Euler's Method manually: Force Dissection ("Freischneiden")

Cable Winch



Environment is static

$$m\ddot{x} = 0$$

$$J\ddot{\theta} = 0$$

Disk rolls

$$m\ddot{x} = 0$$

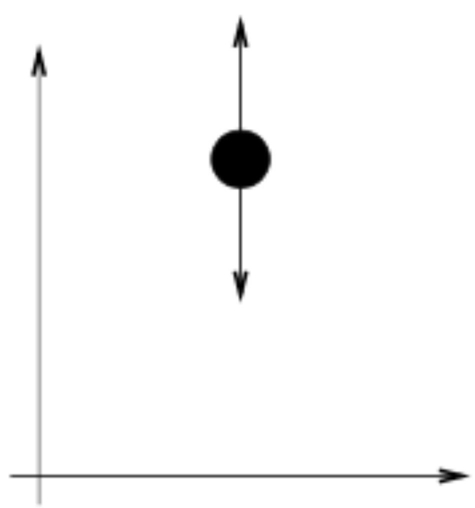
$$J\ddot{\theta} = rF\sin\alpha - S$$

Mass is pulled

$$m\ddot{x} = G - S$$

$$J\ddot{\theta} = 0$$

Intuition: Lagrangian Method



For a Single Particle System:

- Dynamics $m\ddot{y} = f - mg$
- Kinetic Energy $\mathcal{K} = \frac{1}{2}m\dot{y}^2$
- Potential Energy $\mathcal{P} = mgy$

We define the Lagrangian $\mathcal{L} = \mathcal{K} - \mathcal{P}$ and note

$$m\ddot{y} = \frac{d}{dt}(m\dot{y}) = \frac{d}{dt}\frac{\partial}{\partial \dot{y}}\left(\frac{1}{2}m\dot{y}^2\right) = \frac{d}{dt}\frac{\partial \mathcal{K}}{\partial \dot{y}} = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}}$$

$$mg = \frac{\partial}{\partial y}(mgy) = \frac{\partial \mathcal{P}}{\partial y} = -\frac{\partial \mathcal{L}}{\partial y}$$

Lagrange’s Approach $$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{y}} - \frac{\partial \mathcal{L}}{\partial y} = f.$$

This can be done for any robot!

Lagrangian for Robots

For robots?

1. Determine the Kinetic Energy

$$\begin{aligned}\mathcal{K} &= \frac{1}{2}mv^Tv + \frac{1}{2}\boldsymbol{\omega}^T\mathcal{I}\boldsymbol{\omega}. \\ &= \frac{1}{2}\dot{\boldsymbol{q}}^T\sum_{i=1}^{n}\left[m_iJ_{v_i}(\boldsymbol{q})^TJ_{v_i}(\boldsymbol{q}) + J_{\omega_i}(\boldsymbol{q})^TR_i(\boldsymbol{q})I_iR_i(\boldsymbol{q})^TJ_{\omega_i}(\boldsymbol{q})\right]\dot{\boldsymbol{q}}\end{aligned}$$

2. Determine the Potential Energy

$$P = \sum_{i=1}^{n}P_i = \sum_{i=1}^{n}g^Tr_{ci}m_i.$$

3. Use Lagrange's Approach

Problem? Very expensive O(n^3)!

Newton-Euler vs. Lagrange

When should I use Newton-Euler vs. Lagrange?

- Newton-Euler manually? For complex systems with pulleys, etc.
- Lagrange manually? Best for most robots?
- Lagrange computationally? It’s $O(n^3)$, so no!
- Newton-Euler computationally? It’s $O(n)$, so yeah!

General Form

- Dynamics are usually denoted in this form:

$$\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

  ➡ **Inverse dynamics model** $\boldsymbol{u} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$

- From this equation we can already build a robot simulator

  ➡ **Forward dynamics model** $\ddot{\boldsymbol{q}} = f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$

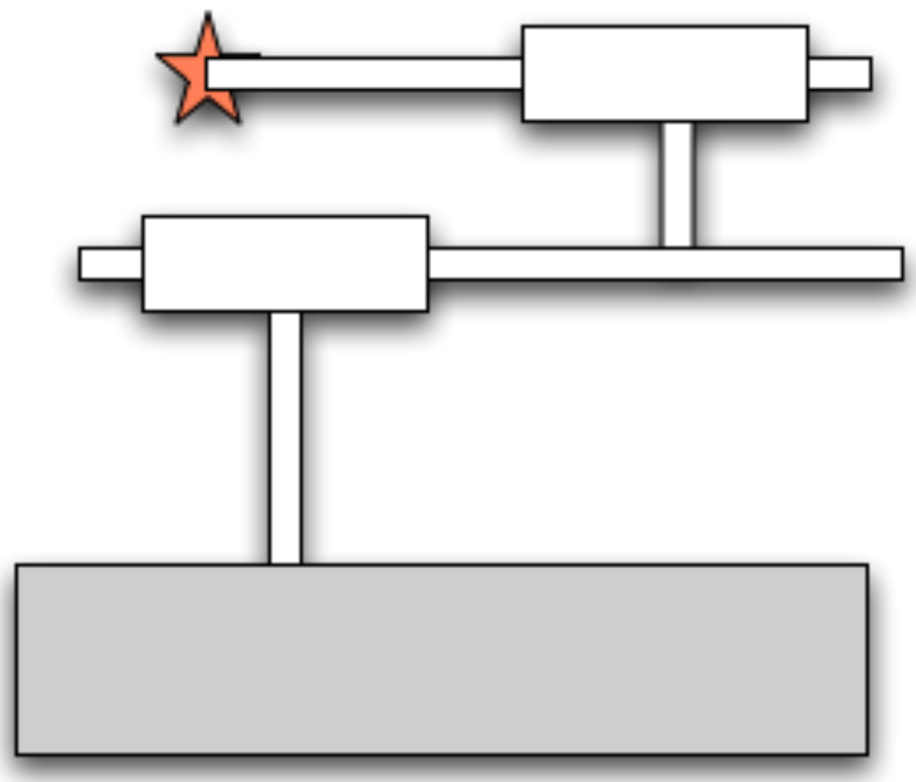
Compute accelerations $\quad \ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\mathbf{u} - \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q}))$

Integrate $\quad \dot{\mathbf{q}} = \int_0^t \ddot{\mathbf{q}} d\tau, \qquad \mathbf{q} = \int_0^t \dot{\mathbf{q}} d\tau$

How to integrate?

How can we integrate $\dot{\mathbf{q}} \quad = \quad \int_0^t \ddot{\mathbf{q}} d\tau, \qquad \mathbf{q} = \int_0^t \dot{\mathbf{q}} d\tau$?

Example 1 - *revisited*



Acting Force

$$
\begin{aligned}
m_1\ddot{x}_1 &= u_1 - u_2 \\
m_2\ddot{x}_2 &= u_2
\end{aligned}
$$

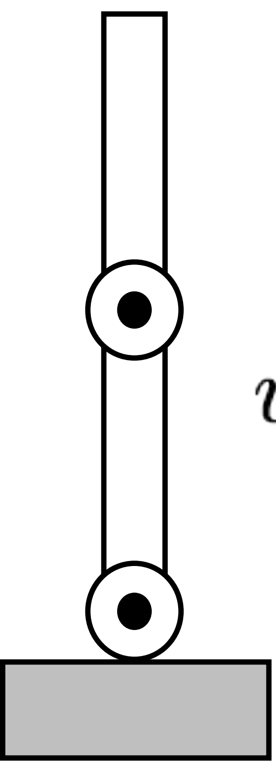
Joints Position

$$
\begin{aligned}
x_1 &= q_1 \\
x_2 &= q_1 + q_2
\end{aligned}
$$

Dynamics

$$
\left[\begin{array}{cc} m_1 + m_2 & m_2 \\ m_2 & m_2 \end{array}\right]
\left[\begin{array}{c} \ddot{q}_1 \\ \ddot{q}_2 \end{array}\right]
=
\left[\begin{array}{c} u_1 \\ u_2 \end{array}\right]
$$

Example 2 - *revisited*



$$
\begin{aligned}
u_1 \quad = \quad & [m_1 l_{g1}^2 + J_1 + m_2(l_1^2 + l_{g2}^2 + 2 l_1 l_{g2} \cos\theta_2) + J_2]\ddot{\theta}_1 \\
+ \quad & [m_2(l_{g2}^2 + l_1 l_2 \cos\theta_2) + J_2]\ddot{\theta}_2 \\
- \quad & 2 m_2 l_1 l_{g2} \dot{\theta}_1 \dot{\theta}_2 \sin\theta_2 \\
- \quad & 2 m_2 l_1 l_{g2} \dot{\theta}_1^2 \sin\theta_2 \\
+ \quad & m_1 g l_{g1} \cos\theta_1 + m_2 g (l_1 \cos\theta_1 + l_{g2} \cos(\theta_1 + \theta_2))
\end{aligned}
$$

Inertial Forces

Coriolis Forces

Centripetal Forces

Gravity

$$
\begin{aligned}
u_2 \quad = \quad & [m_2(l_{g2}^2 + l_1 l_{g2} \cos\theta_2) + J_2]\ddot{\theta}_1 \\
+ \quad & (m_2 l_{g2}^2 + J_2)\ddot{\theta}_2 \\
- \quad & m_2 l_1 l_{g2} \dot{\theta}_1^2 \sin\theta_2 \\
+ \quad & m_2 g l_{g2} \cos(\theta_1 + \theta_2)
\end{aligned}
$$

Inertial Forces

Centripetal Forces

Gravity

Content of this Lecture

1. What is a robot?

2. Modeling Robots
- **Kinematics**
- **Dynamics**

3. Representing Trajectories
- **Splines**

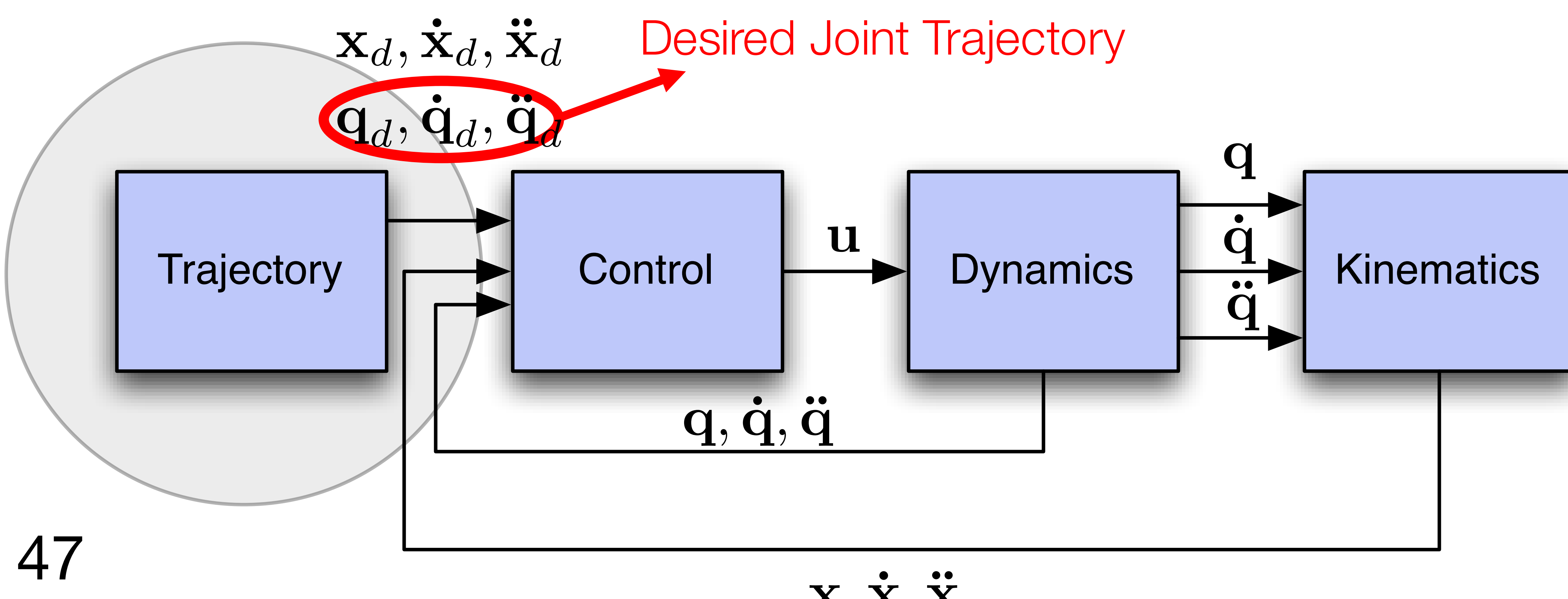
4. Control in Joint Space
- Linear Control
- Model-based Control

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

Block Diagram of Complete System

Trajectory $\mathbf{q}_d(t), \dot{\mathbf{q}}_d(t), \ddot{\mathbf{q}}_d(t)$

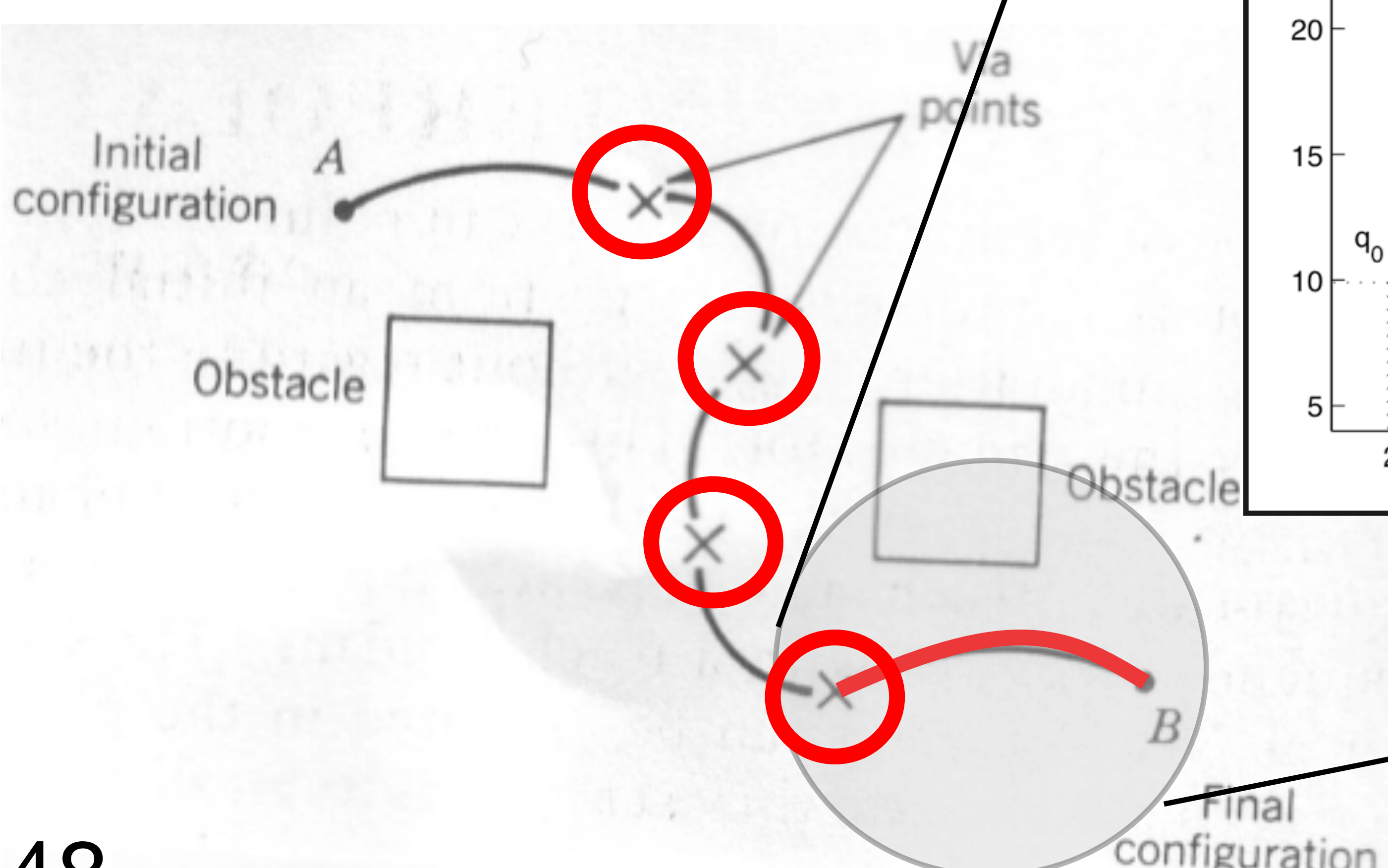
- Specifies the joint positions, velocities and accelerations for each instant of time *t*
- Used to specify the desired movement plan
- Inherently includes velocities and accelerations

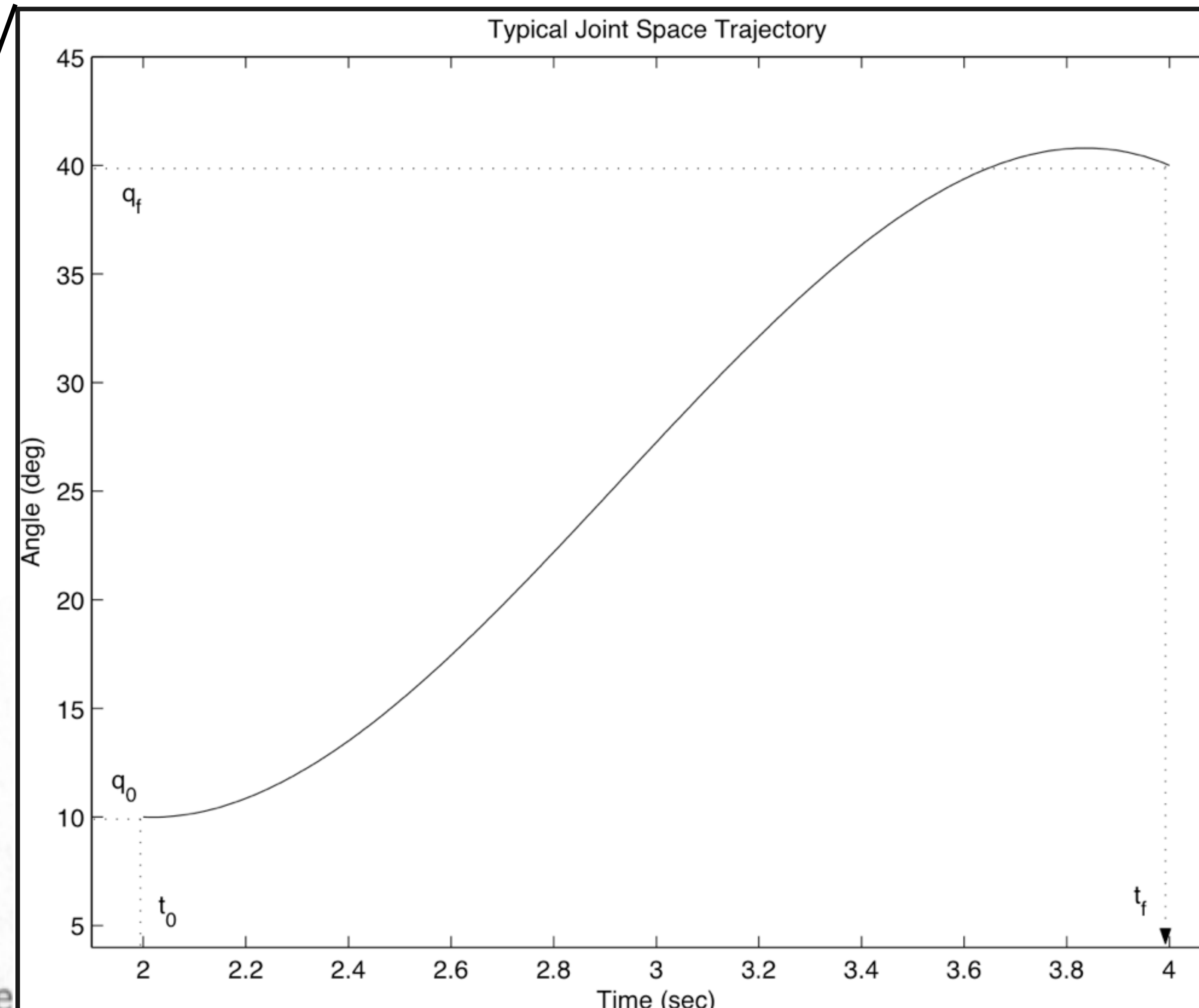


Movement Plans

How to represent trajectories ?

- ➡ Representation with via-points





Trajectory of a single segment

What do we need?

Look once again at the mathematical model of a robot:

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\mathbf{u}$$

$$\dot{\mathbf{q}} = \int_0^t \ddot{\mathbf{q}} d\tau, \qquad \mathbf{q} = \int_0^t \dot{\mathbf{q}} d\tau$$

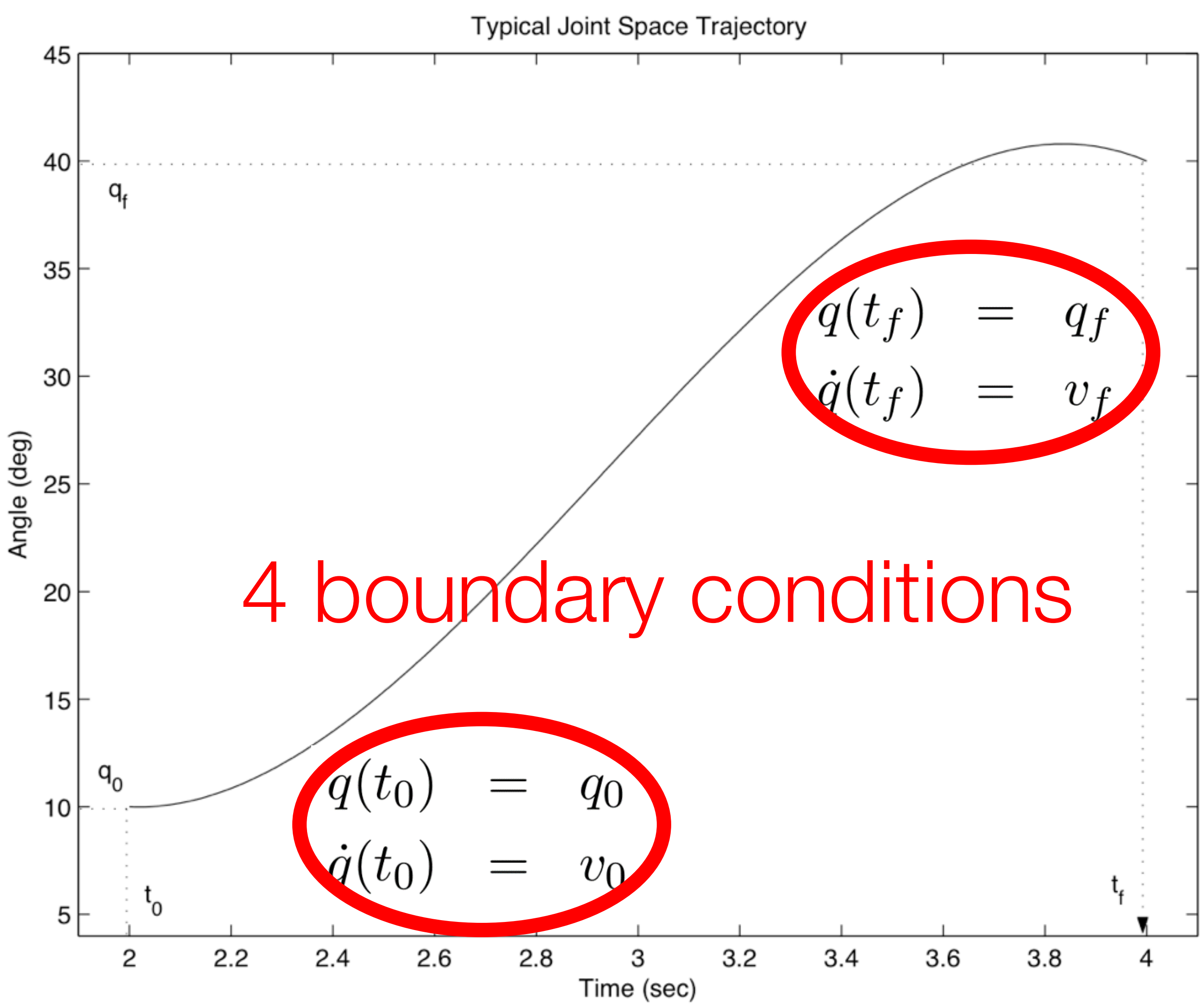
- ➡ Our motor commands can only **influence the acceleration!**
- ➡ The velocities and positions are just integrals of the acceleration.

➡ Any trajectory representation must be **twice differentiable!** The positions and velocities cannot jump.

➡ We can use **polynomials**!

Cubic Splines

How do guarantee no jumps in pos. and vel.?



$$\begin{aligned} q(t_f) &= q_f \\ \dot{q}(t_f) &= v_f \end{aligned}$$

4 boundary conditions

$$\begin{aligned} q(t_0) &= q_0 \\ \dot{q}(t_0) &= v_0 \end{aligned}$$

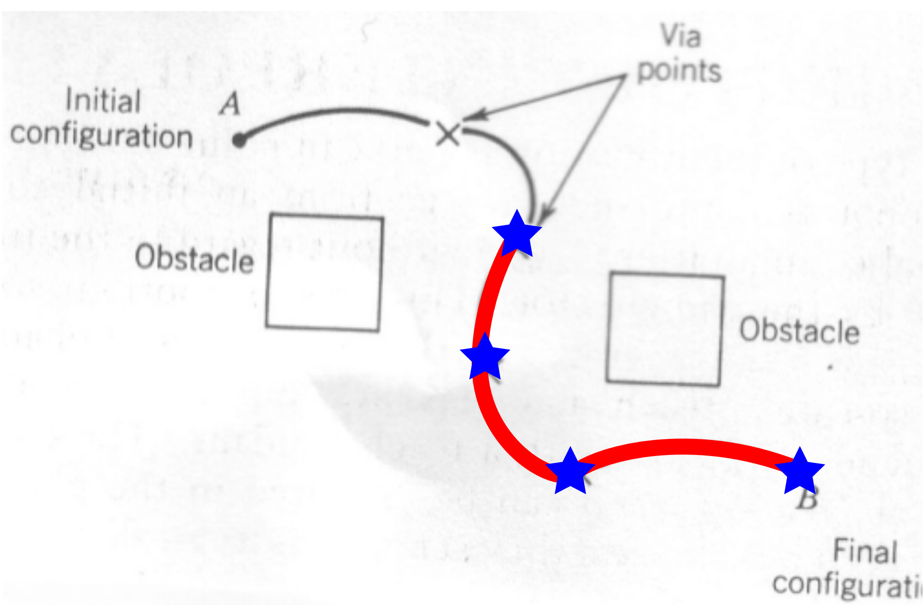
4 free parameters

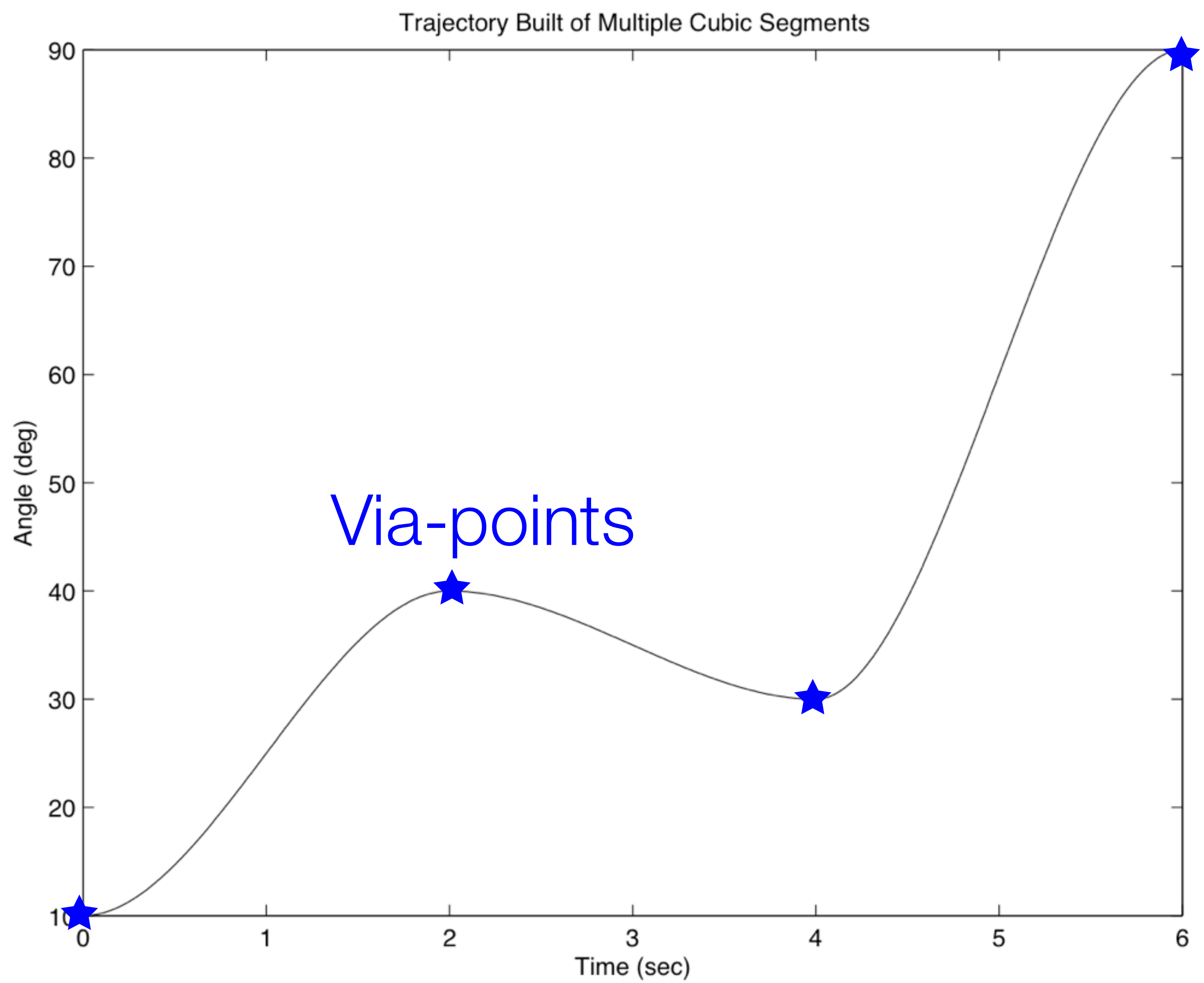
$$\begin{aligned} q(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \dot{q}(t) &= a_1 + 2a_2 t + 3a_3 t^2 \end{aligned}$$

Solve using Boundary Conditions

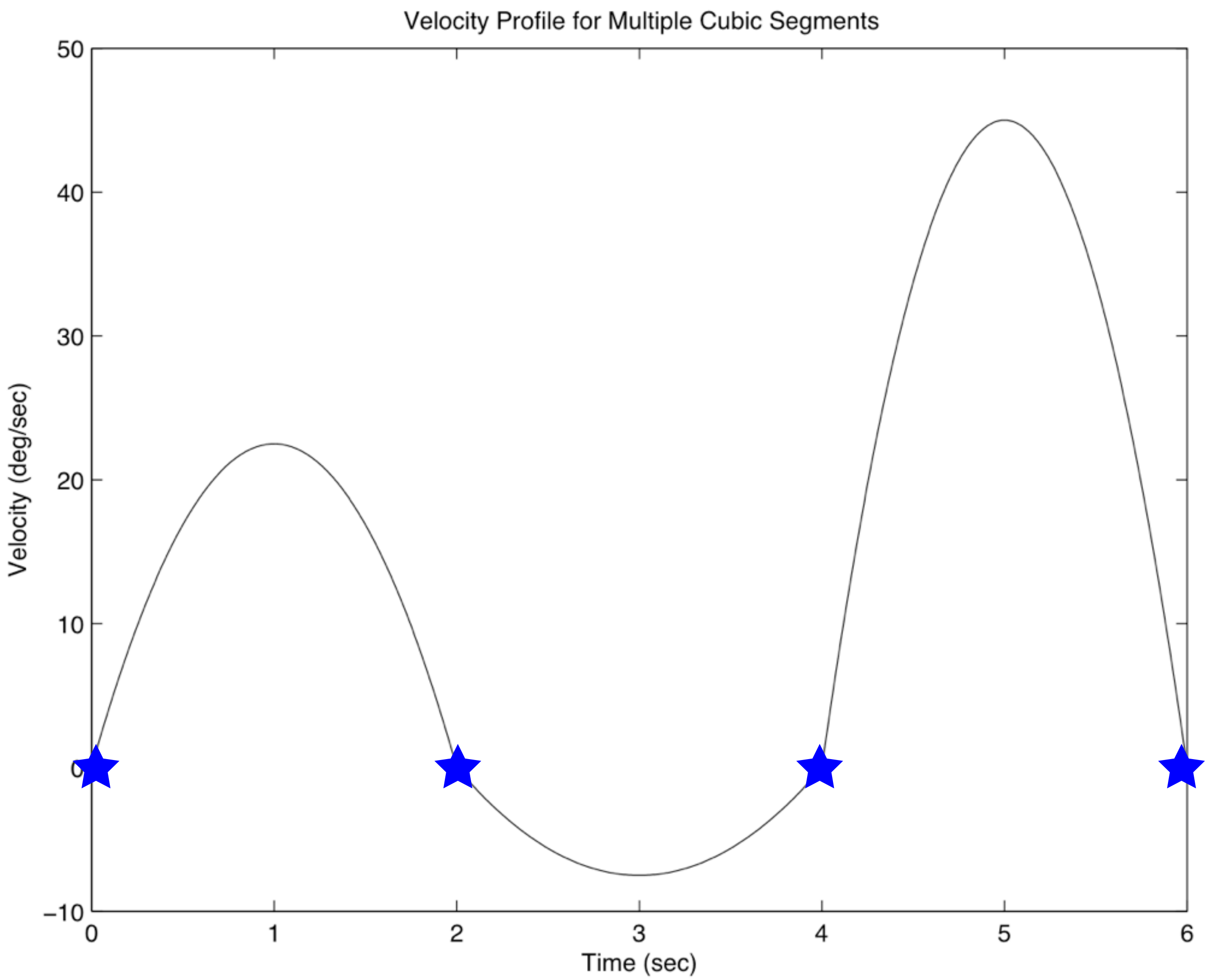
$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix}$$

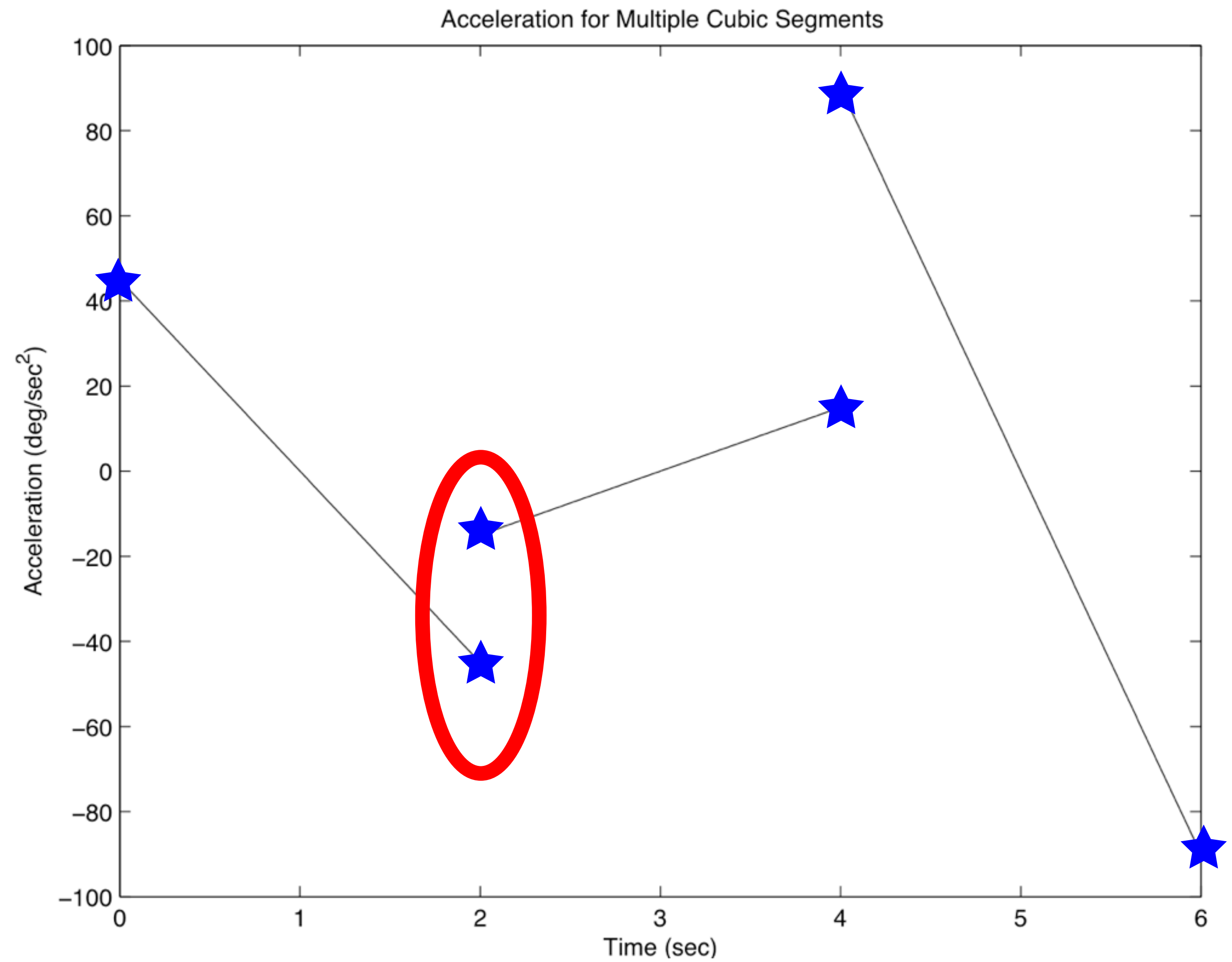
Problems with Cubic Splines





Problems with Cubic Splines



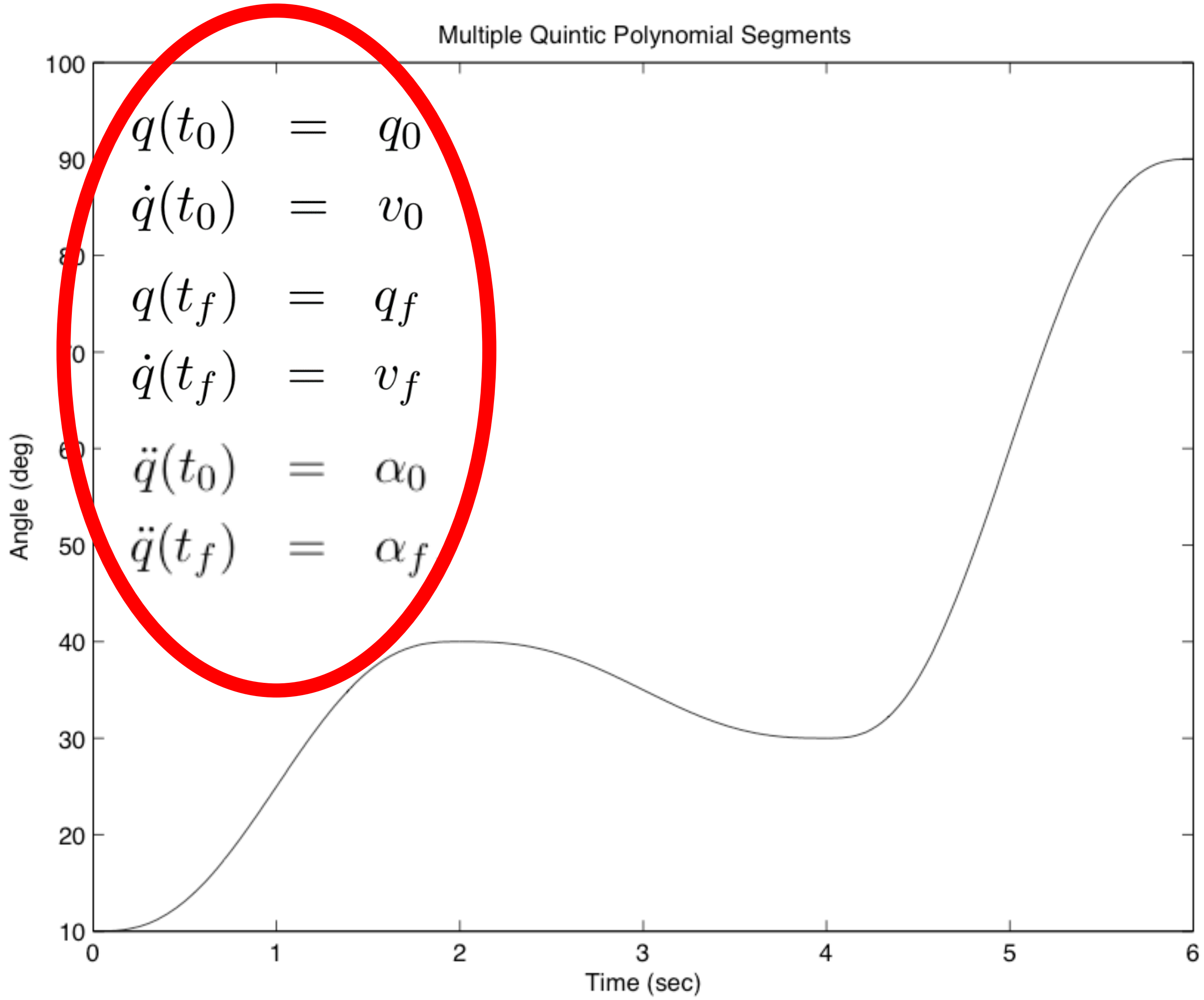


We still get jumps in the acceleration!

- Dangerous at high speed and damage the robot
- This requires higher order splines...

Quintic Splines

No jumps in the acceleration ➡ 6 boundary conditions



$$
\begin{aligned}
q(t_0) &= q_0 \\
\dot{q}(t_0) &= v_0 \\
q(t_f) &= q_f \\
\dot{q}(t_f) &= v_f \\
\ddot{q}(t_0) &= \alpha_0 \\
\ddot{q}(t_f) &= \alpha_f
\end{aligned}
$$

Replace Cubic Polynomials
by Quintic Polynomials

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

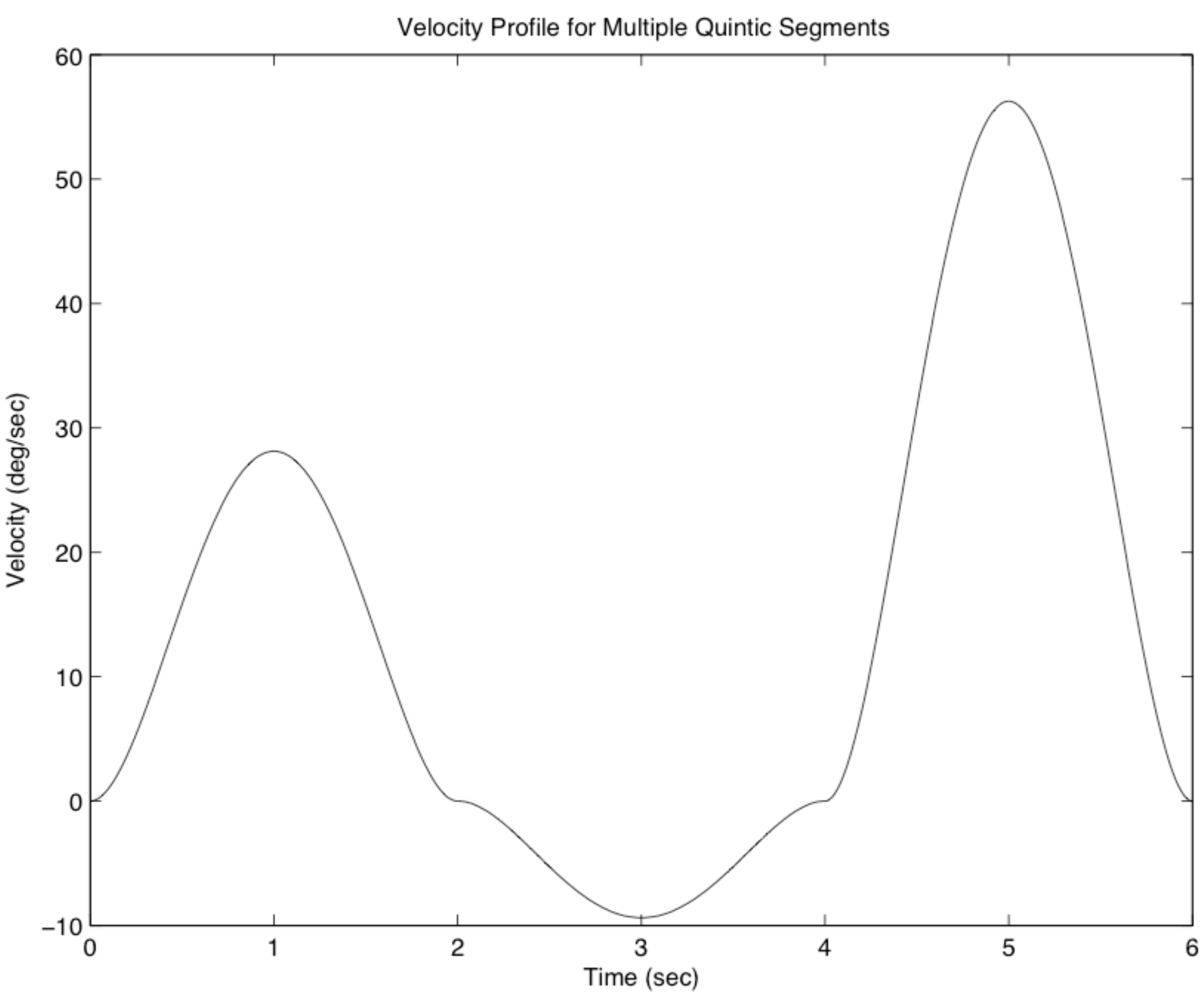
6 free parameters

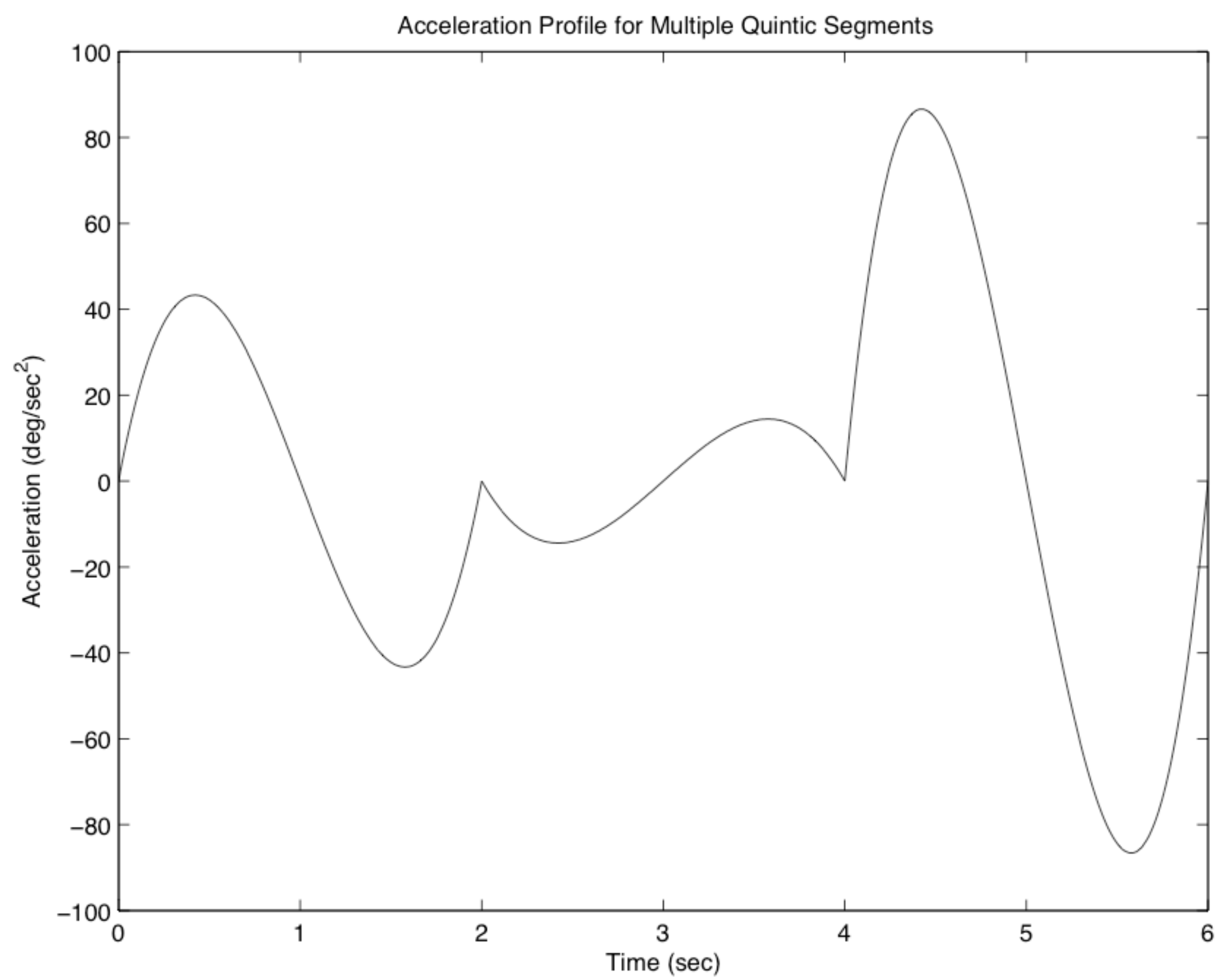
Use new boundary conditions

$$
\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\
0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\
0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\
1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\
0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\
0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}
=
\begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix}
$$

Quintic Splines

Smooth velocity and acceleration profiles with quintic splines





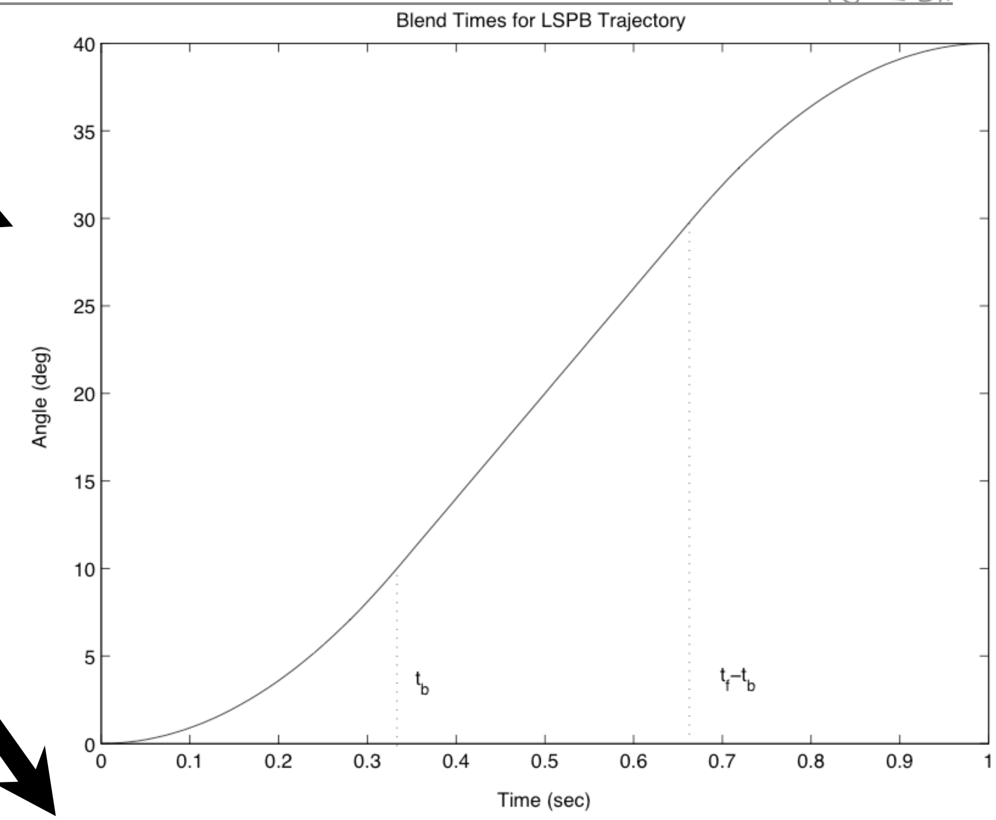
Alternatives to Splines

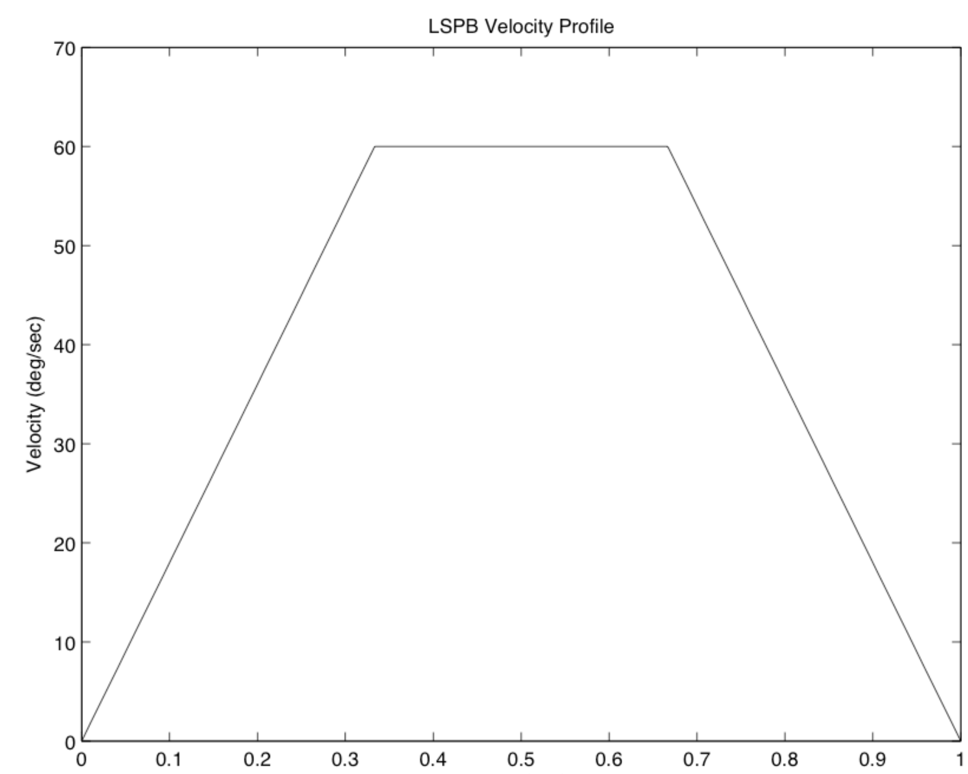
- Linear Segments with Parabolic Blends!
- Trapezoidal Minimum Time Trajectories
- Potential Fields $V(\mathbf{q})$

$$\dot{\mathbf{q}} = \frac{dV(\mathbf{q})}{d\mathbf{q}}$$

- Nonlinear Dynamical Systems

$$\ddot{\mathbf{q}} = f(\mathbf{q}, \dot{\mathbf{q}}, \theta)$$





Ask questions...

Ask questions...

Q & A?

Content of this Lecture

1. What is a robot?

2. Modeling Robots
  Kinematics
  Dynamics

3. Representing Trajectories
  Splines

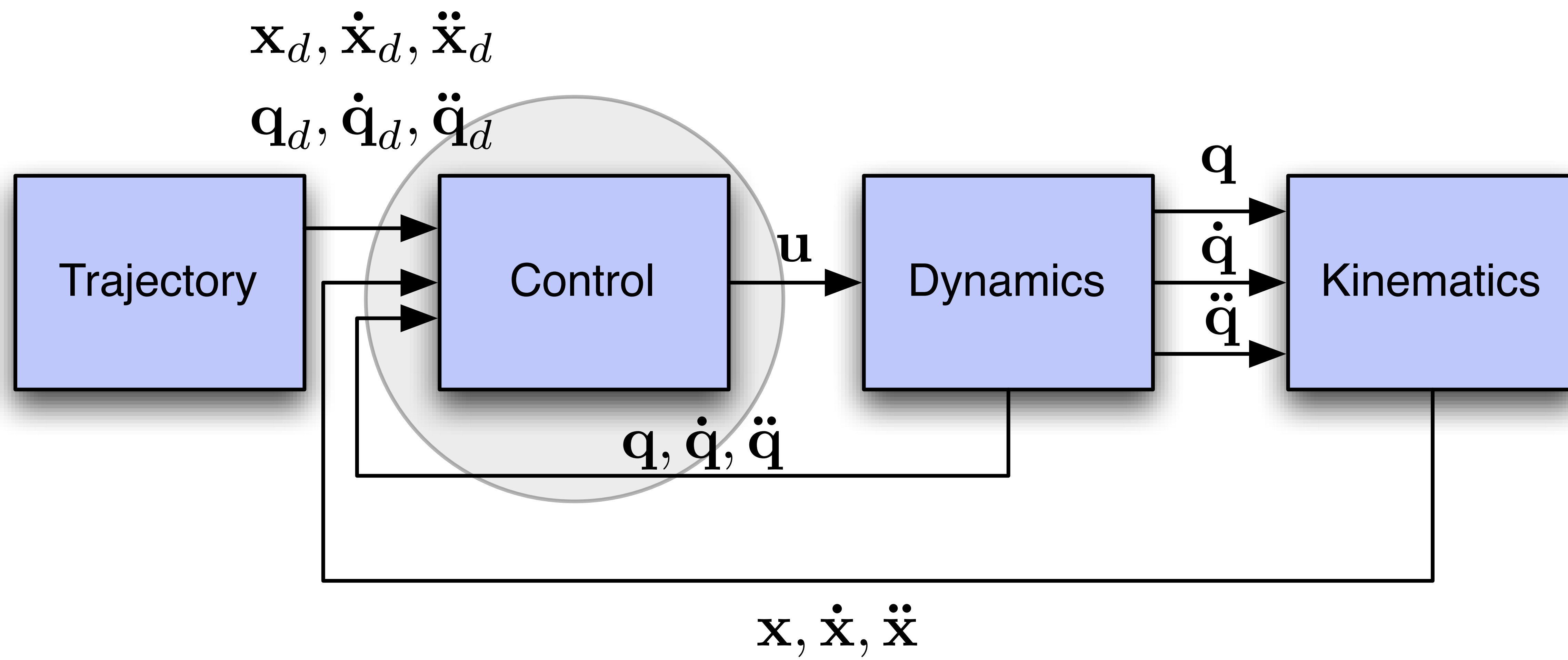
4. Control in Joint Space
  Linear Control
  Model-based Control

5. Control in Task Space
  Inverse Kinematics
  Differential Inverse Kinematics

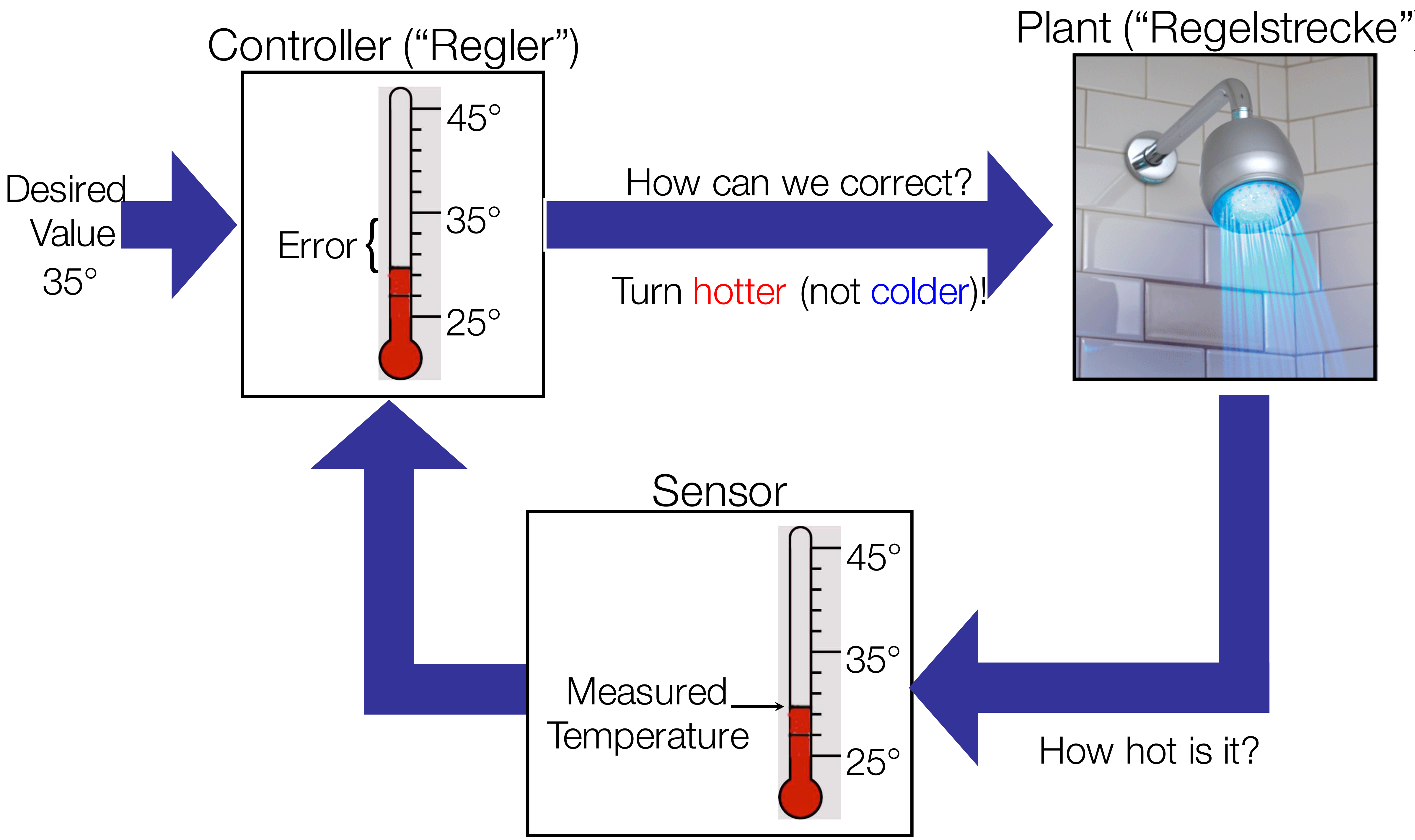
Control

Why do we need control?

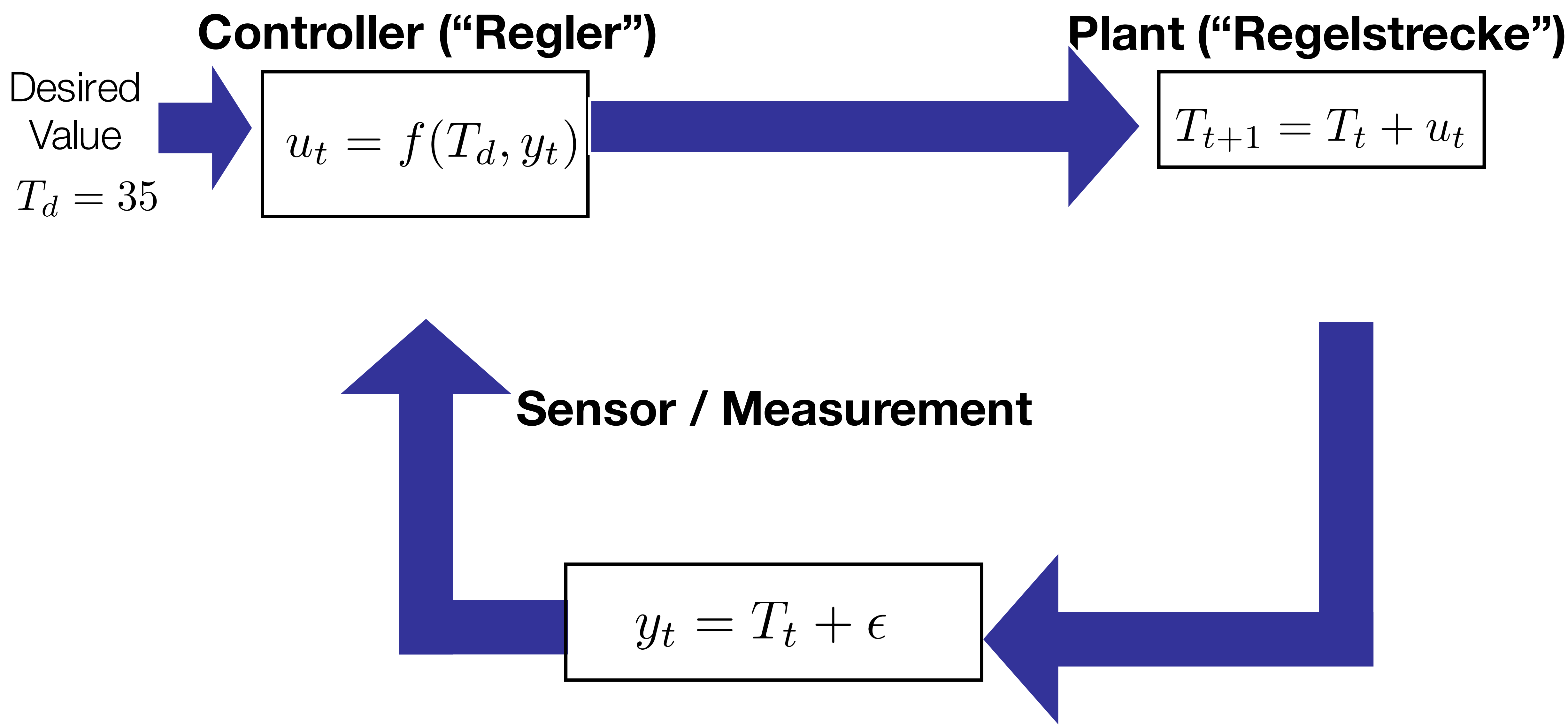
➡ Given a desired trajectory like $\mathbf{q}_d(t), \dot{\mathbf{q}}_d(t), \ddot{\mathbf{q}}_d(t)$, we still need to find the controls $\boldsymbol{u}$ to follow this trajectory



Feedback Control: Generic Idea

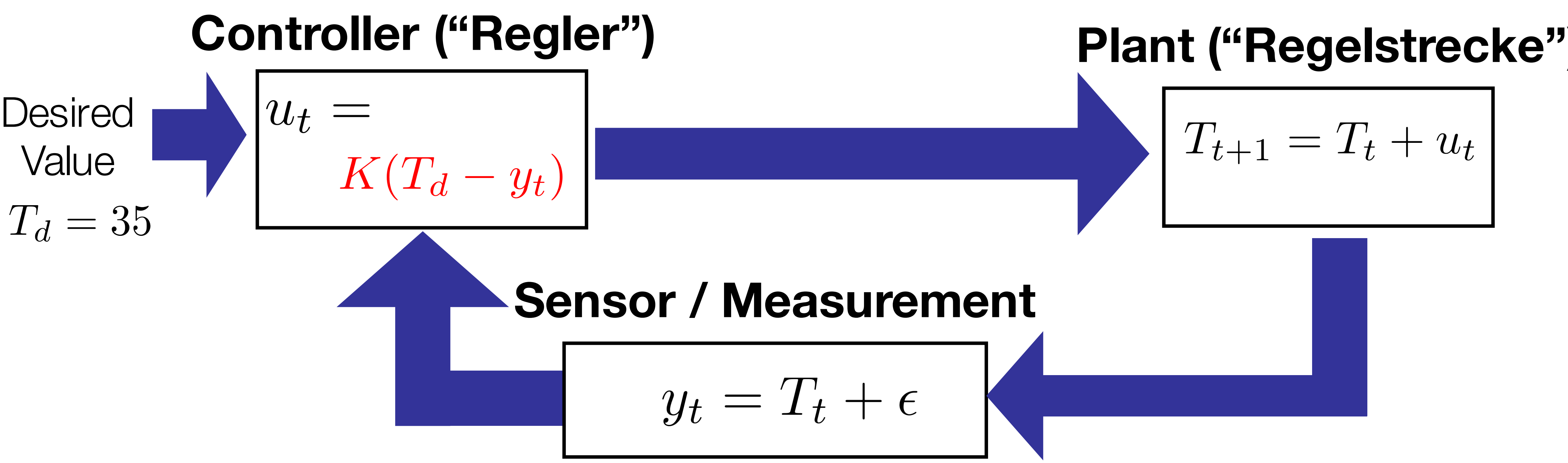


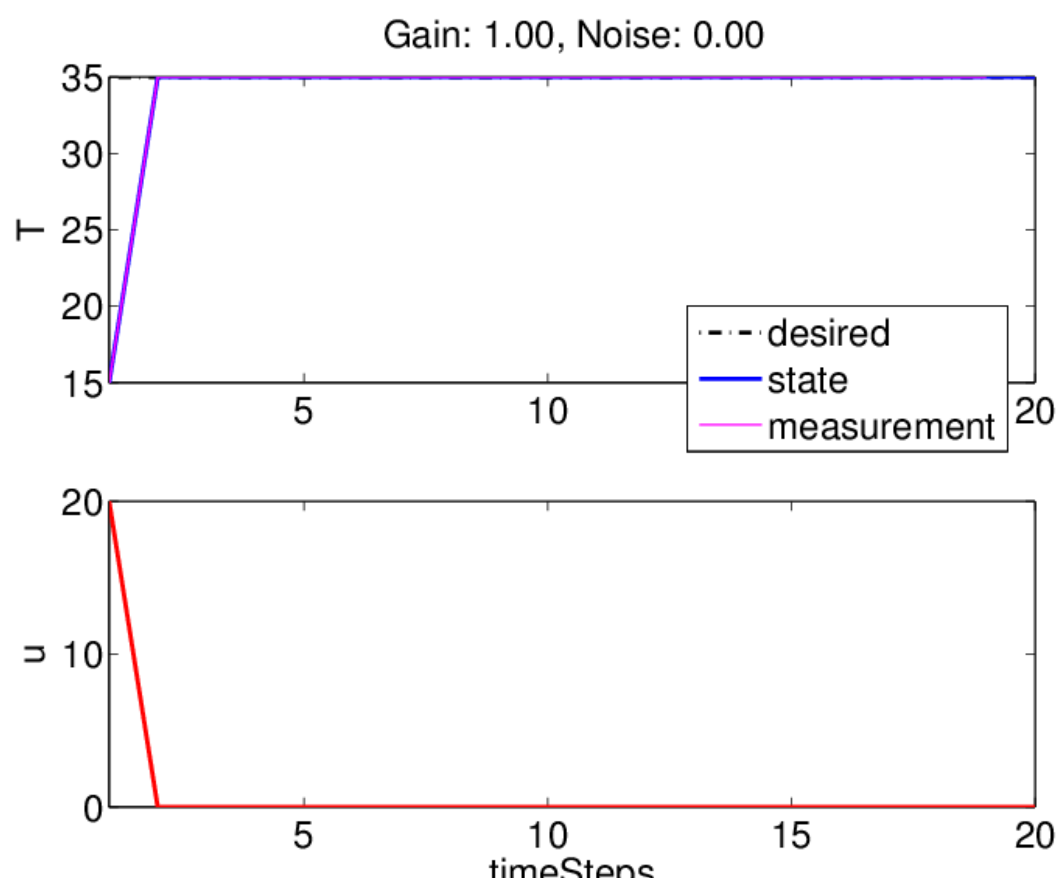
Feedback Control: Generic Idea



ϵ Measurement errors

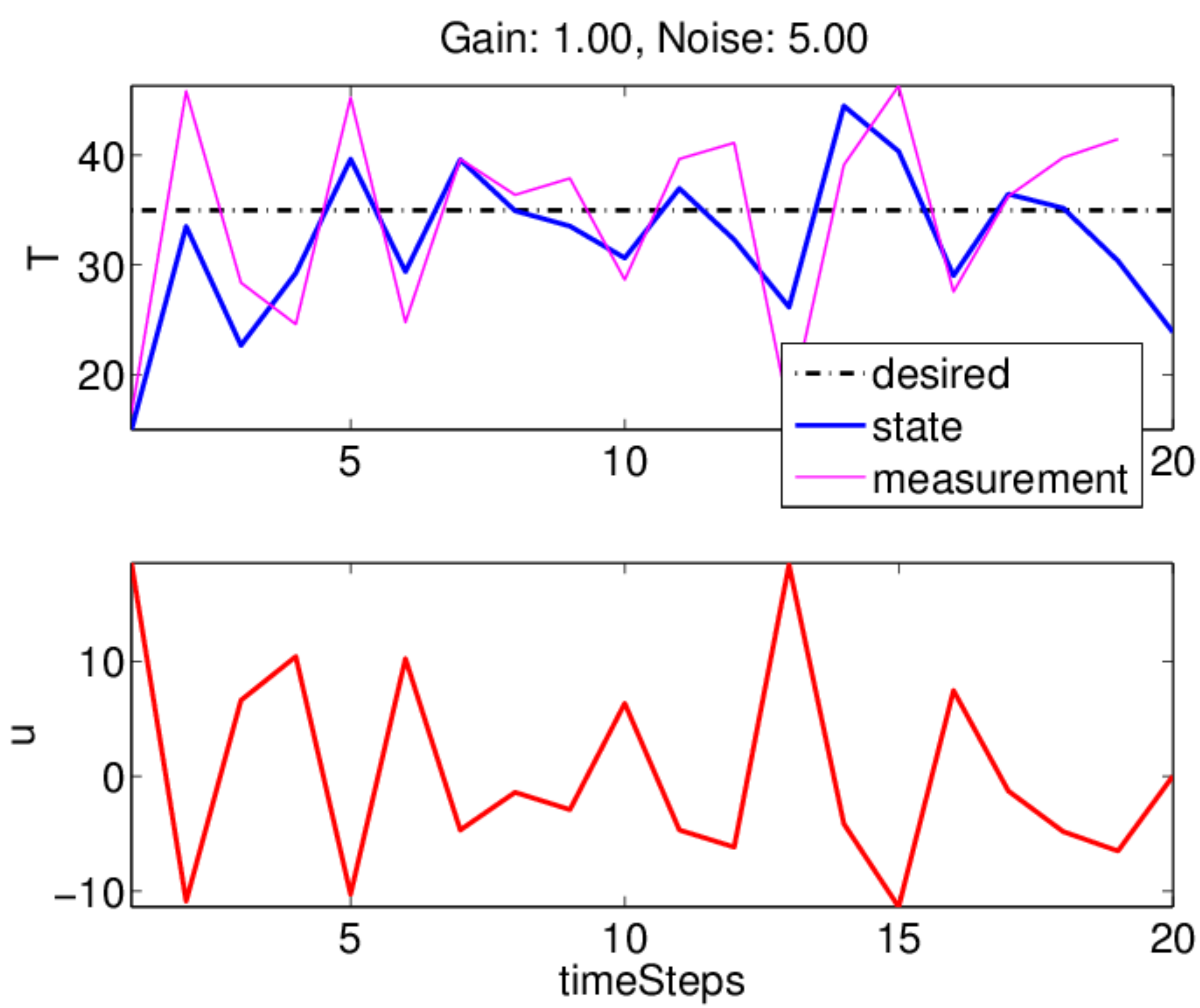
Linear Feedback Control





Measurement Errors

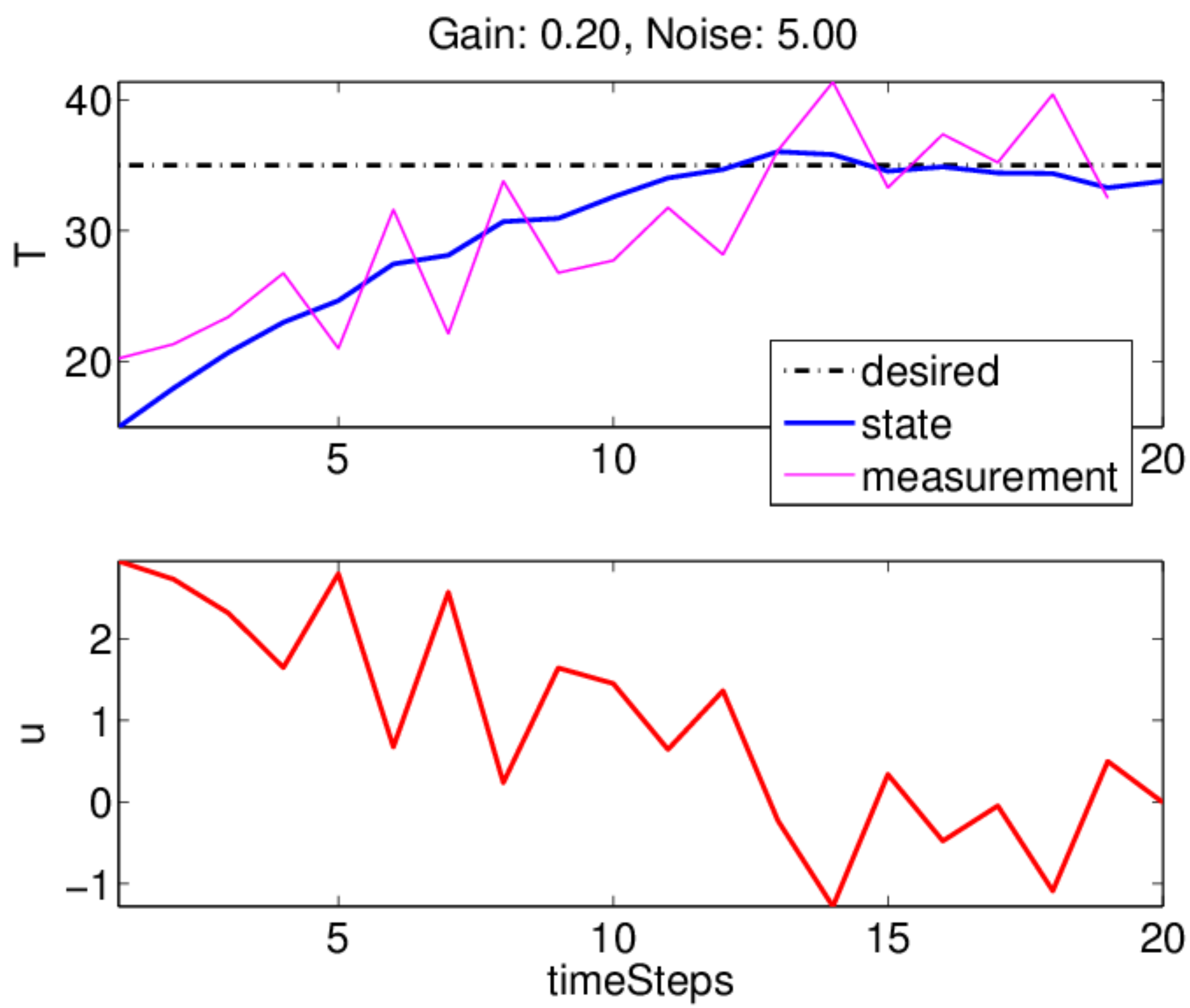
What effect do measurement errors have?



➡ High Motor Commands, that's not a comfortable way to shower

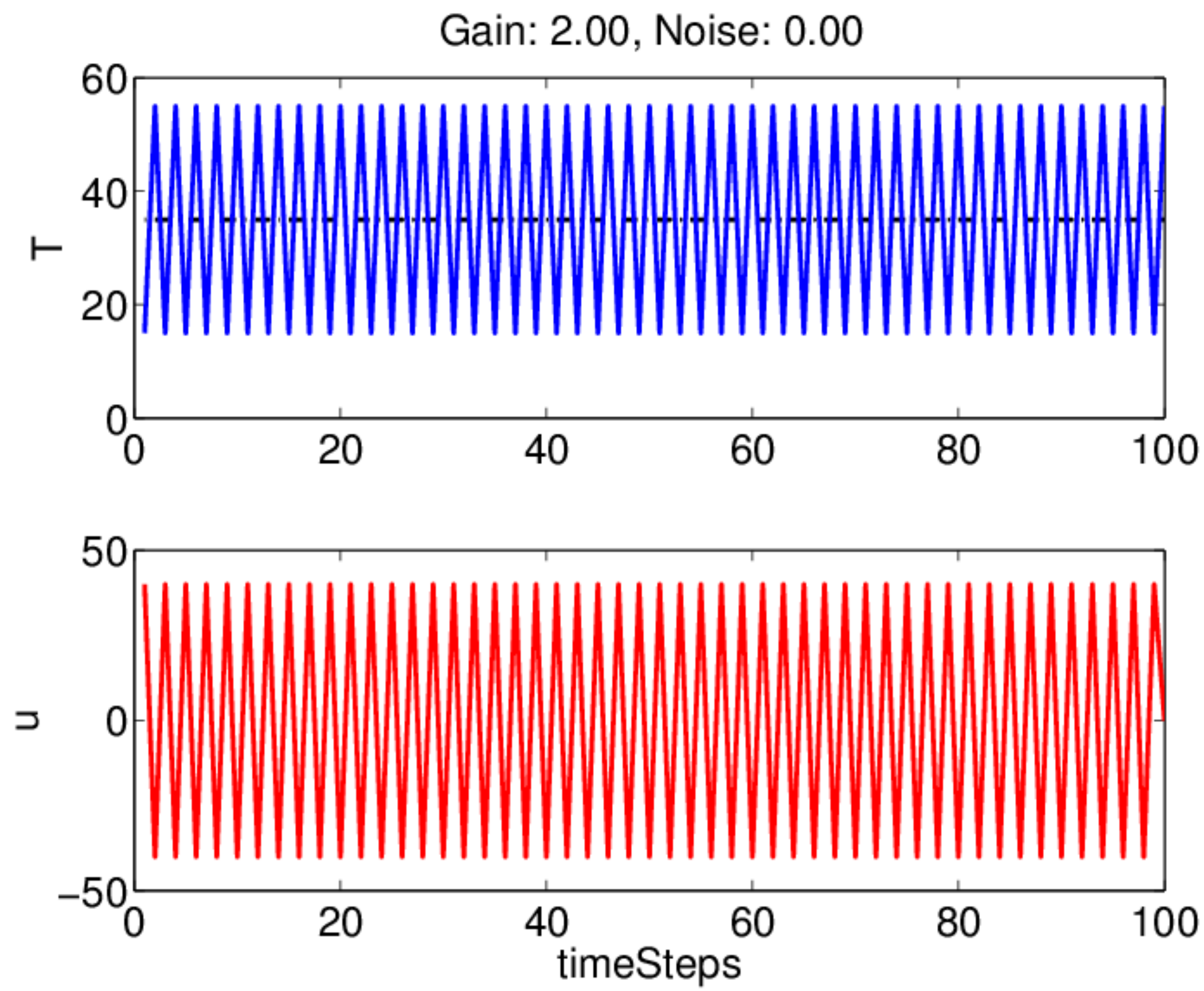
Proper Control with Measurement Errors

Lower our gains!!!



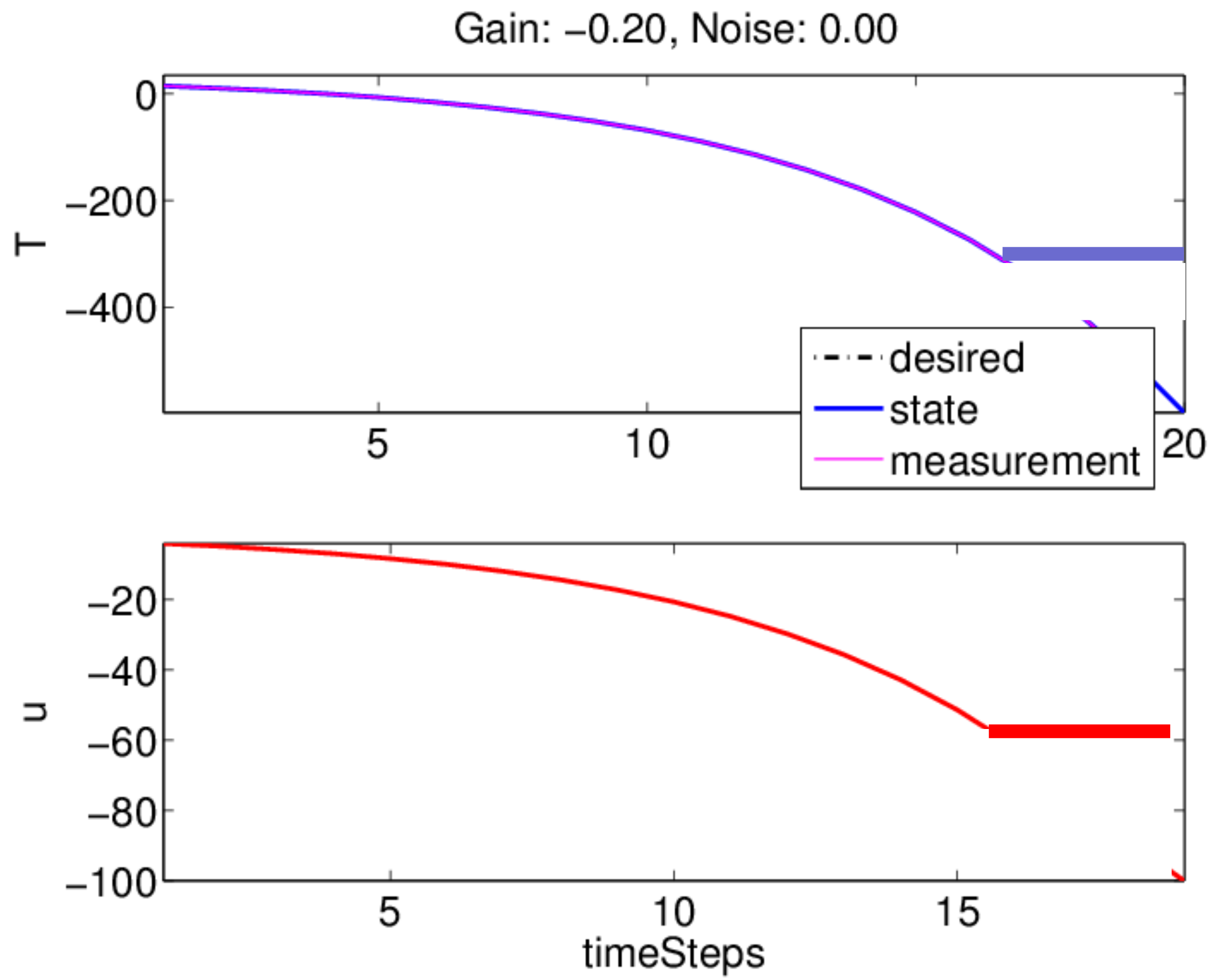
What do High Gains do?

High gains are always problematic!!!! Check K = 2 !



What happens if the sign is messed up?

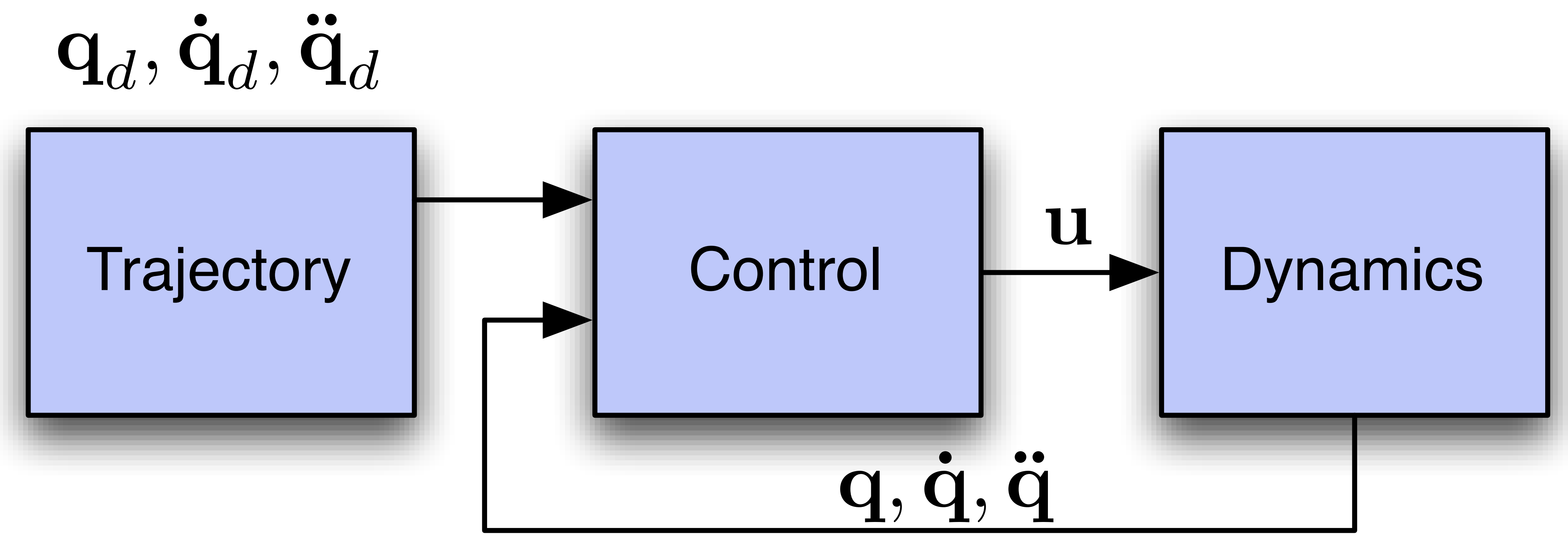
Check K = -0.2.



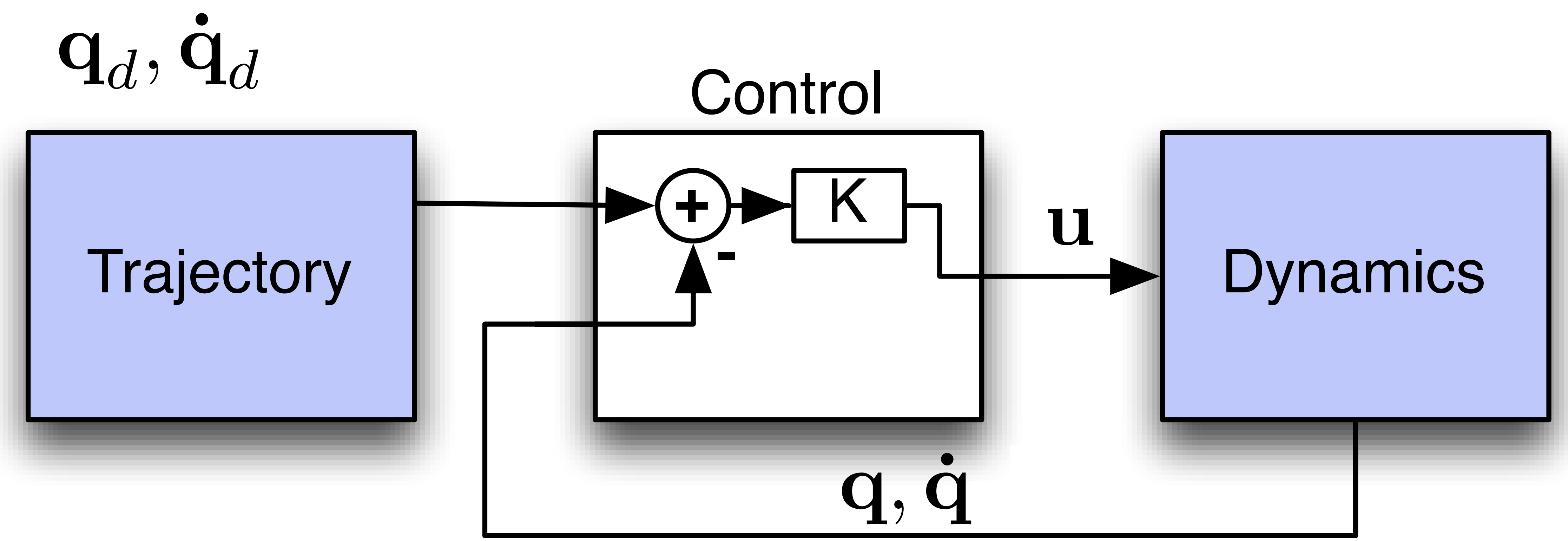
Zero Kelvin

No growing error

Control in Robotics



Linear Control in Robotics?



Linear Controllers:

- P-Controller (only $\mathbf{q}_d$ in the diagram above)
- PD-Controller
- PID-Controller (different from above's block diagram)

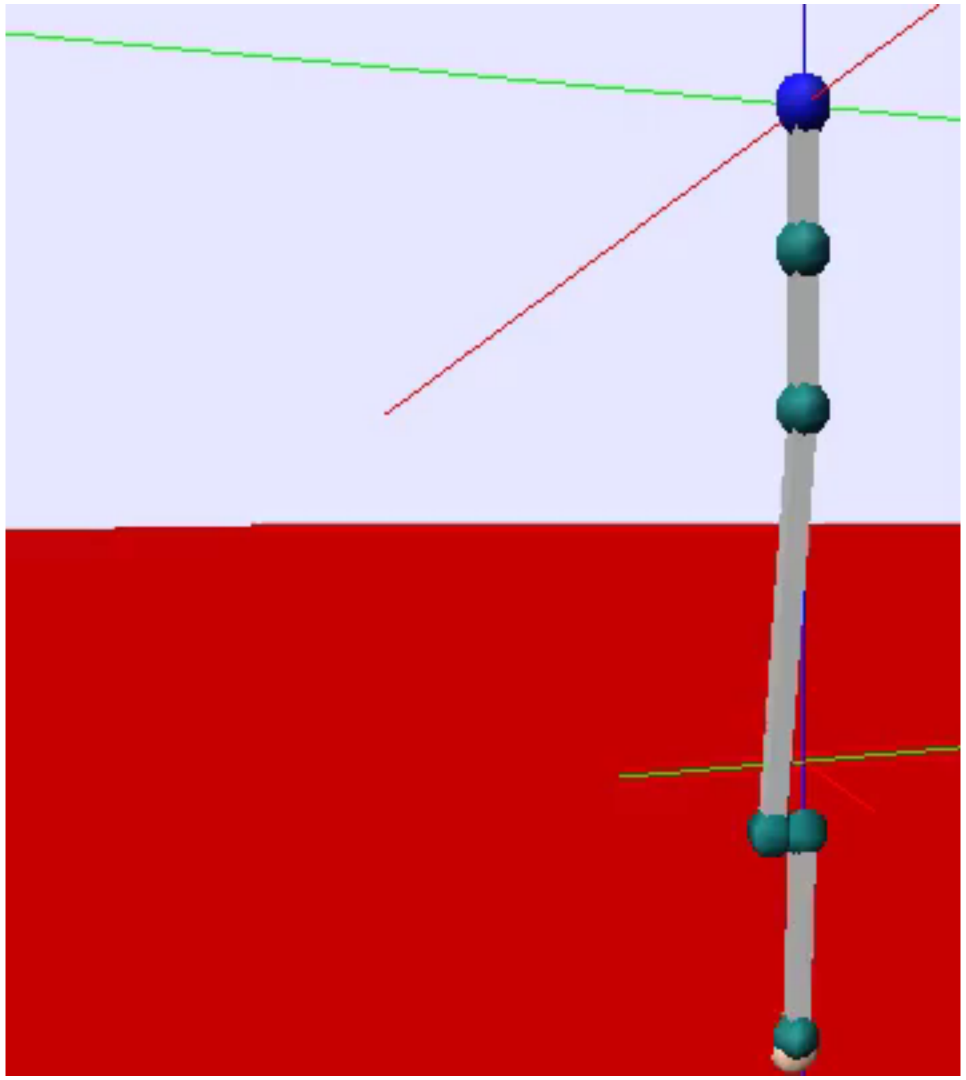
Linear Control: “P-Regler”

P-Controller:

based on position error

$$\boldsymbol{u}_t = \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}_t)$$

$$\mathbf{q}_d \quad = \quad \begin{bmatrix} 0 \\ 0.9 \\ 0 \\ 0.9 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \dot{\mathbf{q}}_d \quad = \quad 0$$



What happens for this control law?

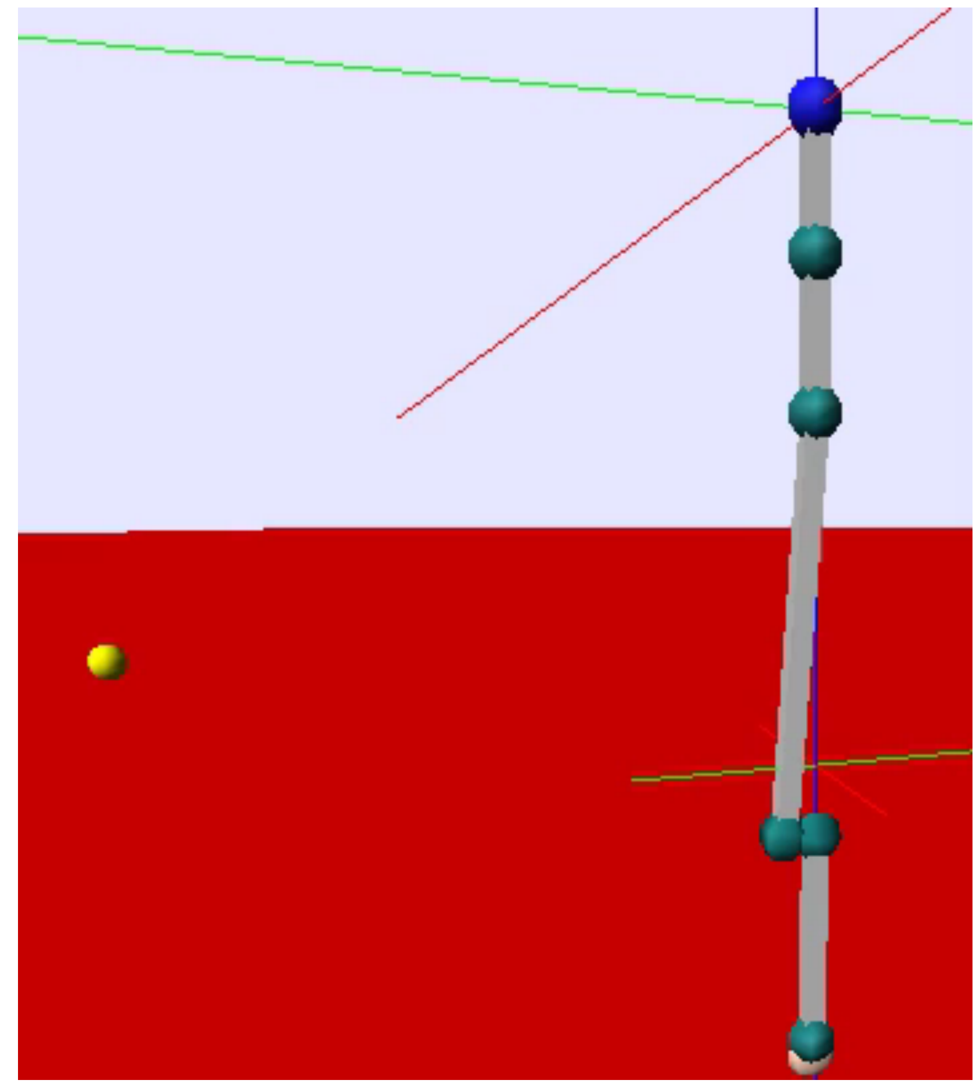
Oscillations, mean position error

Linear Control: “PD-Regler”

PD-Controller:

based on position and velocity errors

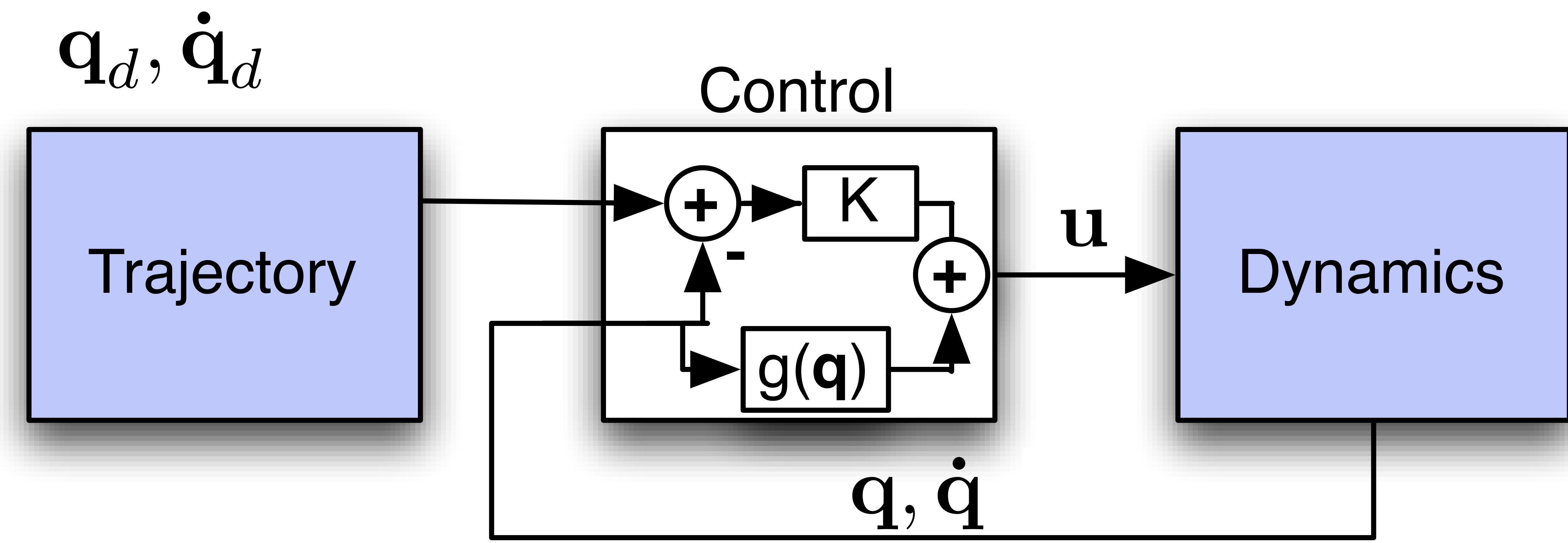
$$\boldsymbol{u}_t = \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}_t) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}_t)$$



What happens for this control law?

Steady state error: It can not reach set-point

Linear PD Control with Gravity Compensation



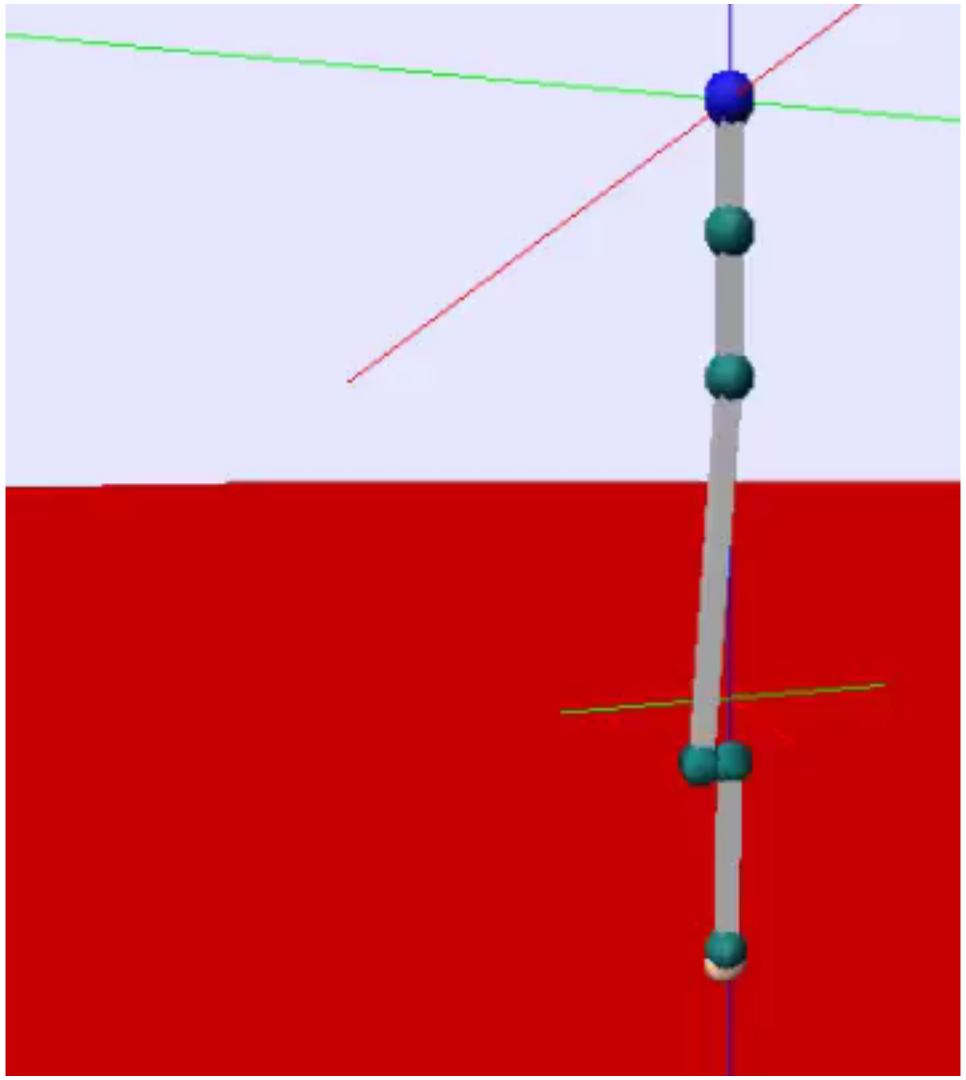
- To reach the set-point, we must compensate for gravity
- Most industrial robots employ this approach

Linear PD Control with Gravity Compensation

PD-Controller with gravity compensation

$$\boldsymbol{u}_t = \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}_t) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}_t) + \boldsymbol{g}(\boldsymbol{q})$$

➡ Requires a model of all steady state components!



Note on PID Control

Alternatively to doing gravity compensation, we could try to estimate the motor command to compensate for the error.

➡ This can be done by integrating the error

$$\mathbf{u} = \mathbf{K}_P(\mathbf{q}_{\text{des}} - \mathbf{q}) + \mathbf{K}_D(\dot{\mathbf{q}}_{\text{des}} - \dot{\mathbf{q}}) + \mathbf{K}_I \int_{-\infty}^{t} (\mathbf{q}_{\text{des}} - \mathbf{q}) d\tau.$$

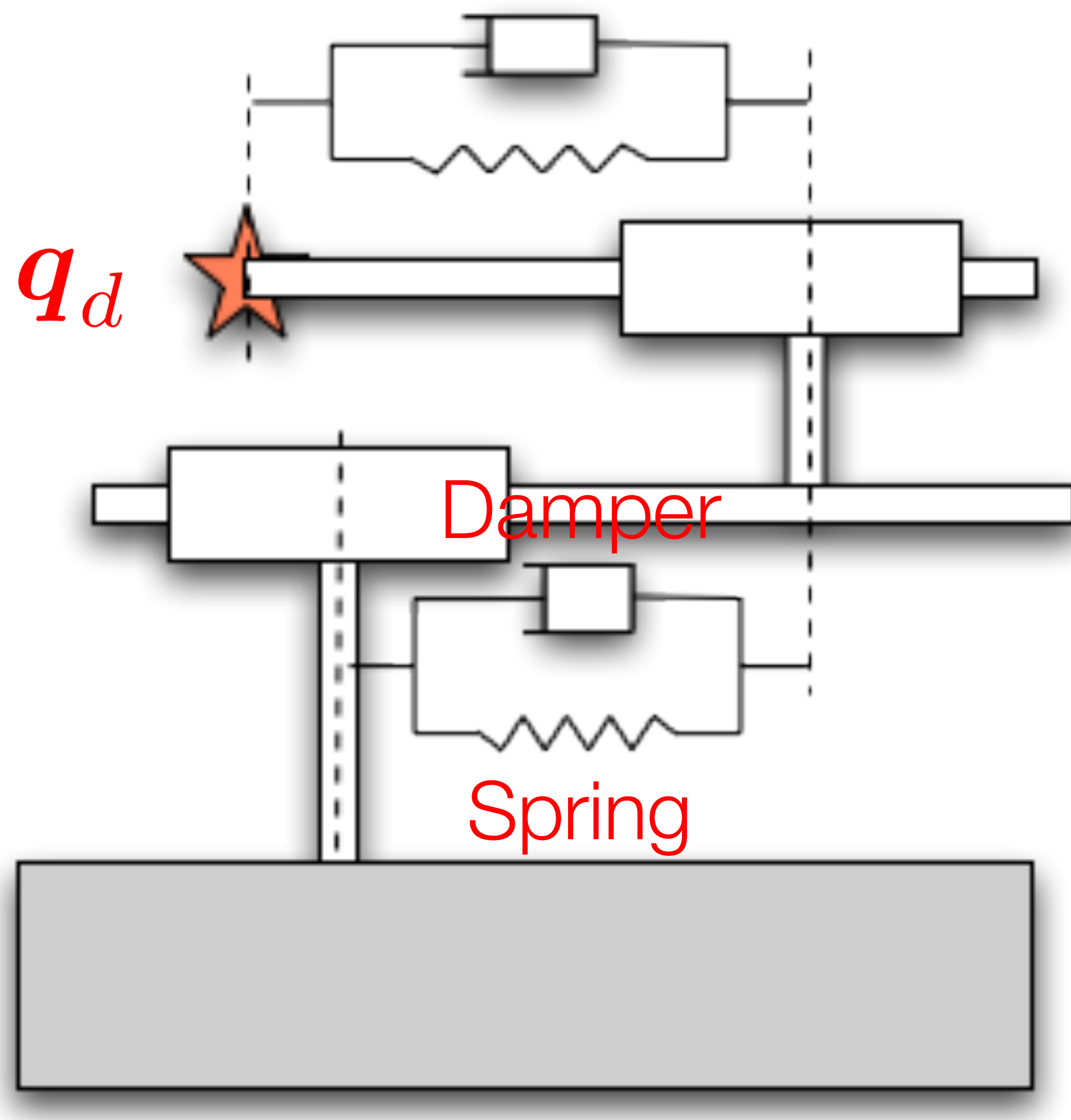
For steady state systems, this approach can be reasonable (e.g., if our shower thermostat has an offset)

- ➢ Useful if no good model is known!

➡ For tracking control, it may create havoc and disaster!

Mechanical Equivalent

PD Control is equivalent to adding spring-dampers between the desired values and the actuated robot parts.



$$\boldsymbol{u}_t = \boldsymbol{K}_P(\boldsymbol{q}_d - \boldsymbol{q}_t) - \boldsymbol{K}_D\dot{\boldsymbol{q}}_t$$

Ask questions...

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

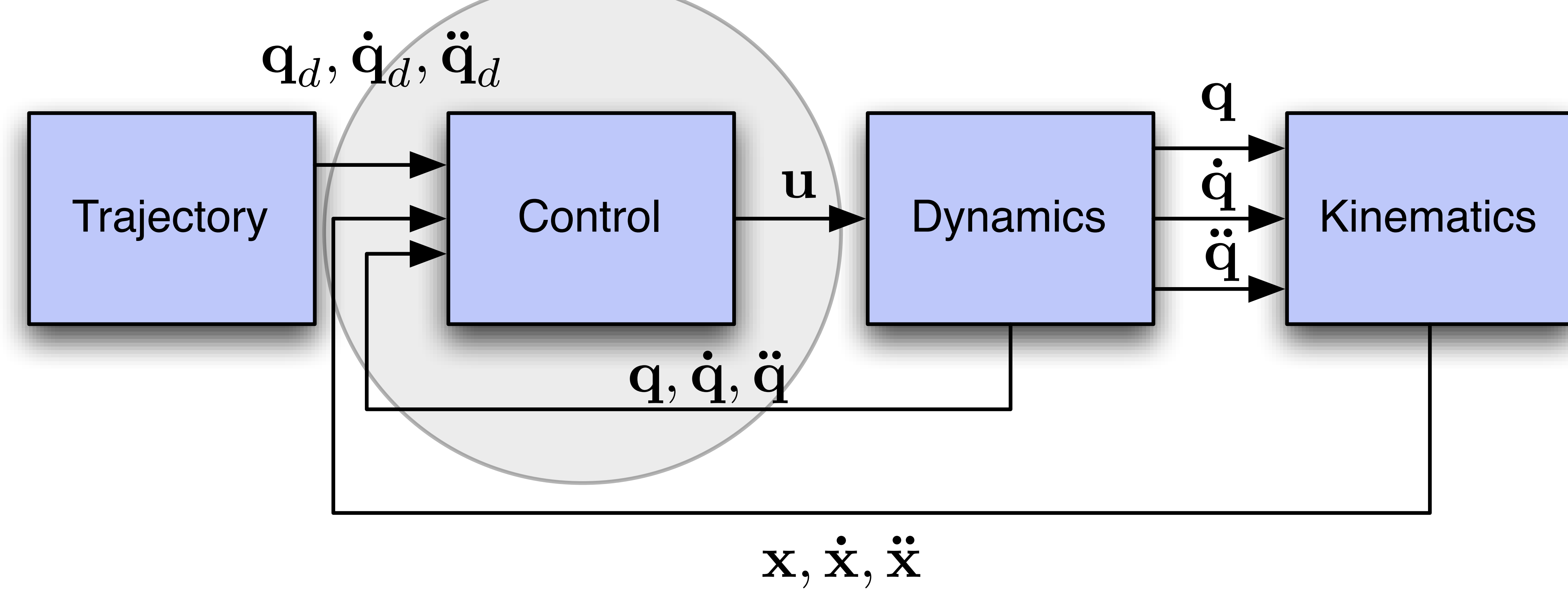
5. Control in Task Space
Inverse Kinematics
Differential Inverse Kinematics

Block Diagram of Complete System

PD with gravity compensation is not a good choice

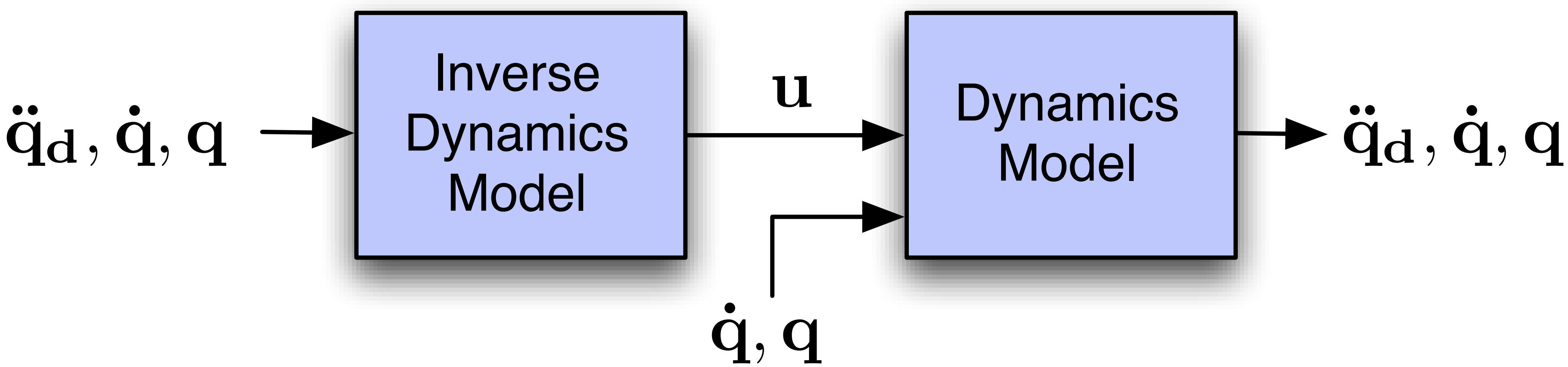
- We need an error to generate a control signal. To be accurate, we need to MAGNIFY a small error, i.e., we have huge gains.
- Huge gains are costly, make the robot very stiff and dangerous.
- Mechanical systems are second order systems, i.e., we can only change the acceleration by inserting torques!

Can we do better with a model?



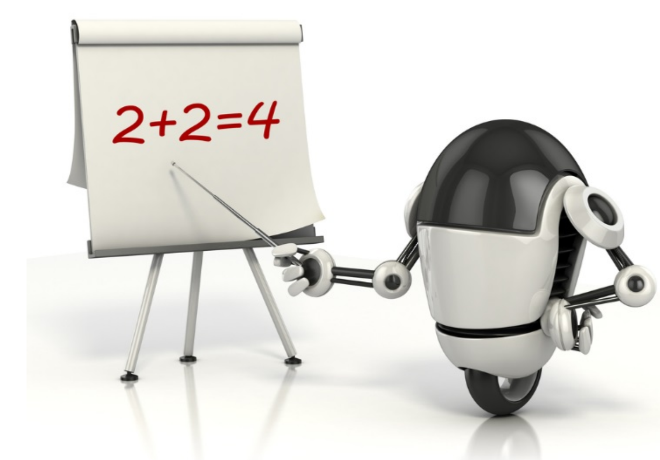
Model-based Control: Key Insight

Forward and inverse dynamics model have a useful property:



- Forward Model: $\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\mathbf{u} - \mathbf{c}(\dot{\mathbf{q}}, \mathbf{q}) - \mathbf{g}(\mathbf{q}))$
- Inverse Model: $\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{c}(\dot{\mathbf{q}}, \mathbf{q}) + \mathbf{g}(\mathbf{q})$

- Thus, we set $\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_d$



Model-based Feedback Control

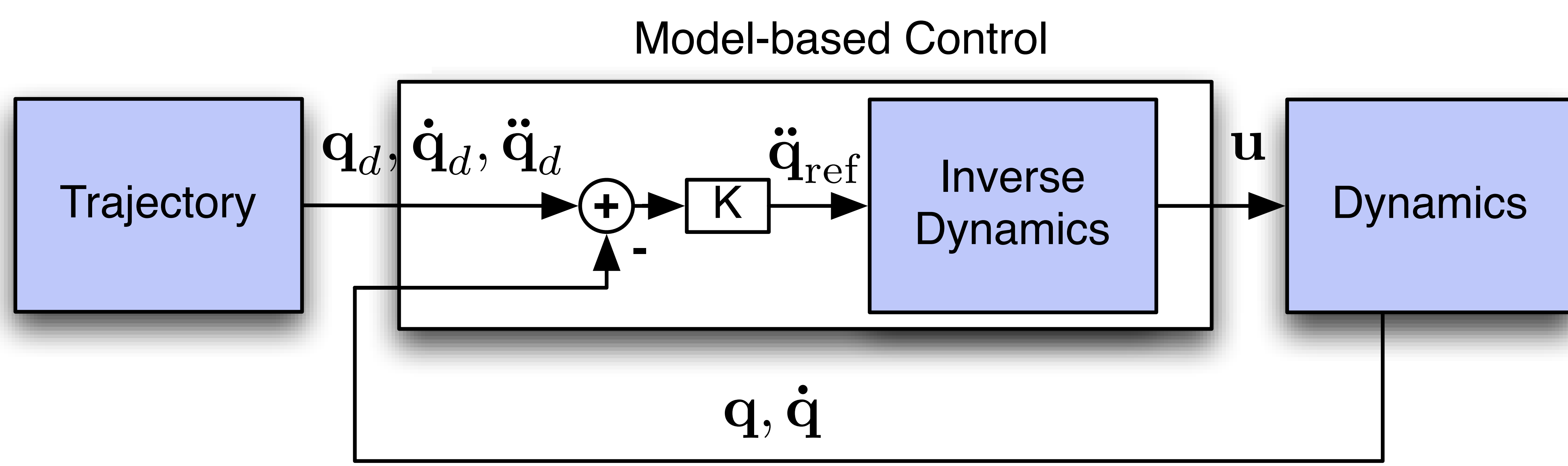
For errors, adapt only reference acceleration

$$\ddot{\mathbf{q}}_{\mathrm{ref}} = \ddot{\mathbf{q}}_{\mathbf{d}} + \mathbf{K}_D(\dot{\mathbf{q}}_{\mathrm{des}} - \dot{\mathbf{q}}) + \mathbf{K}_P(\mathbf{q}_{\mathrm{des}} - \mathbf{q})$$

... and insert it into our model $\mathbf{u} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\mathrm{ref}} + \mathbf{c}(\dot{\mathbf{q}}, \mathbf{q}) + \mathbf{g}(\mathbf{q})$

As $\ddot{q} = \ddot{q}_{\mathrm{ref}}$ the system behaves as linear decoupled system

➡ I.e. it is a decoupled double integrator!



Feedforward Control

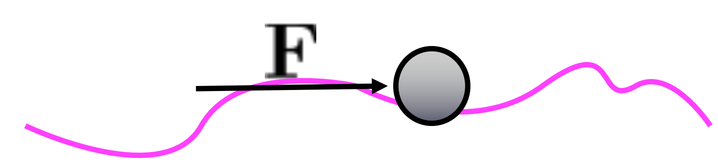
➡ Feedforward control assumes $\mathbf{q} \approx \mathbf{q_d}$ and $\dot{\mathbf{q}} \approx \dot{\mathbf{q}}_\mathbf{d}$

➡ Hence, we have

$$\mathbf{u} = \mathbf{u}_{\mathrm{FF}}(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}, \ddot{\mathbf{q}}_\mathbf{d}) + \mathbf{u}_{\mathrm{FB}}$$

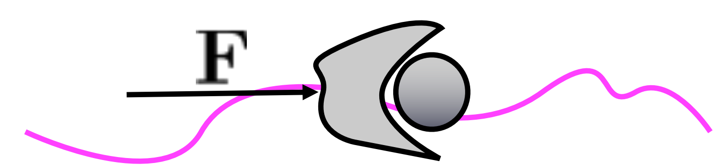
with feedforward torque prediction using an inverse dynamics model

$$\mathbf{u}_{\mathrm{FF}} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})$$

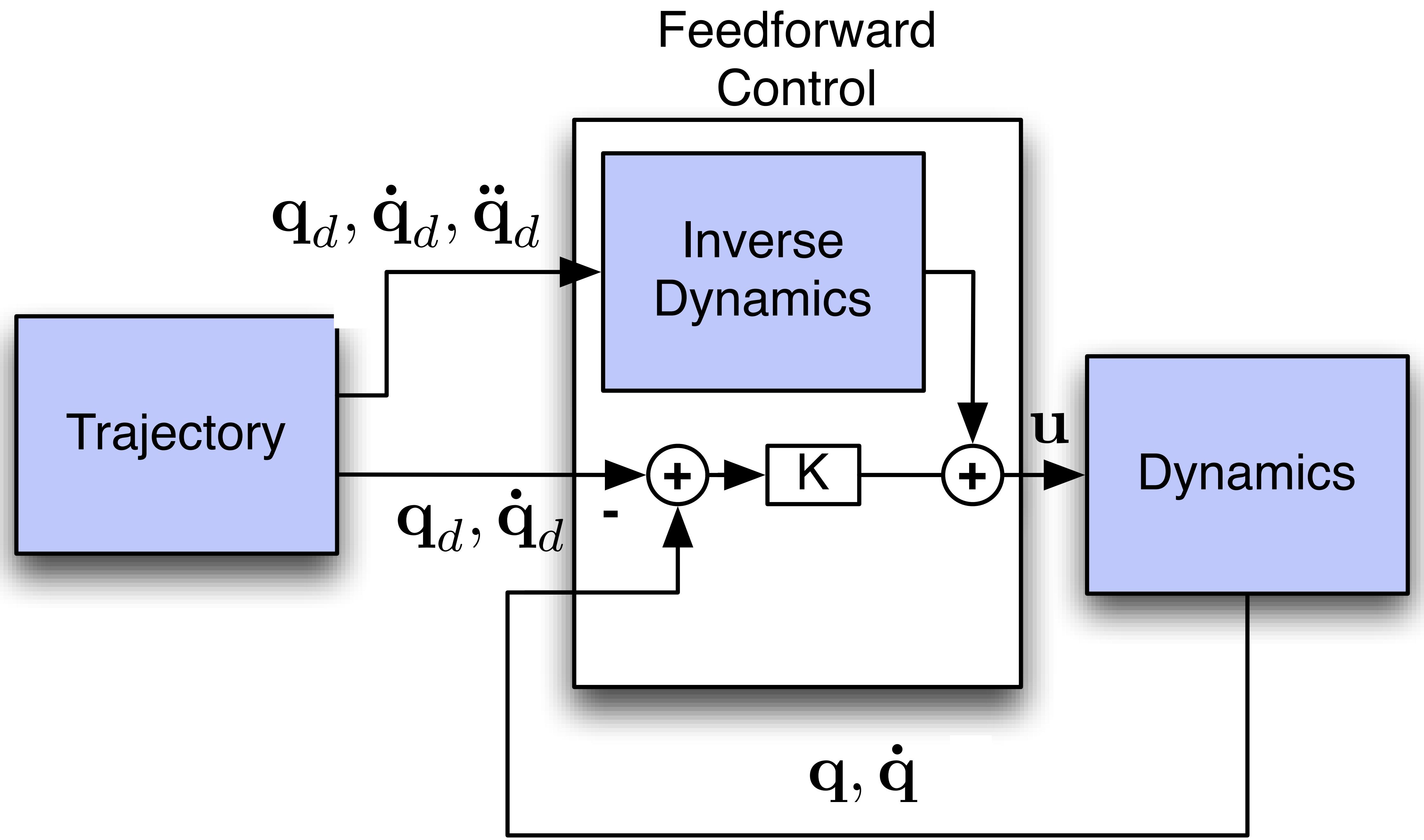


and a linear PD control law for feedback

$$\mathbf{u}_{\mathrm{FB}} = \mathbf{K}_P(\mathbf{q}_{\mathrm{des}} - \mathbf{q}) + \mathbf{K}_D(\dot{\mathbf{q}}_{\mathrm{des}} - \dot{\mathbf{q}})$$



Feedforward Control



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

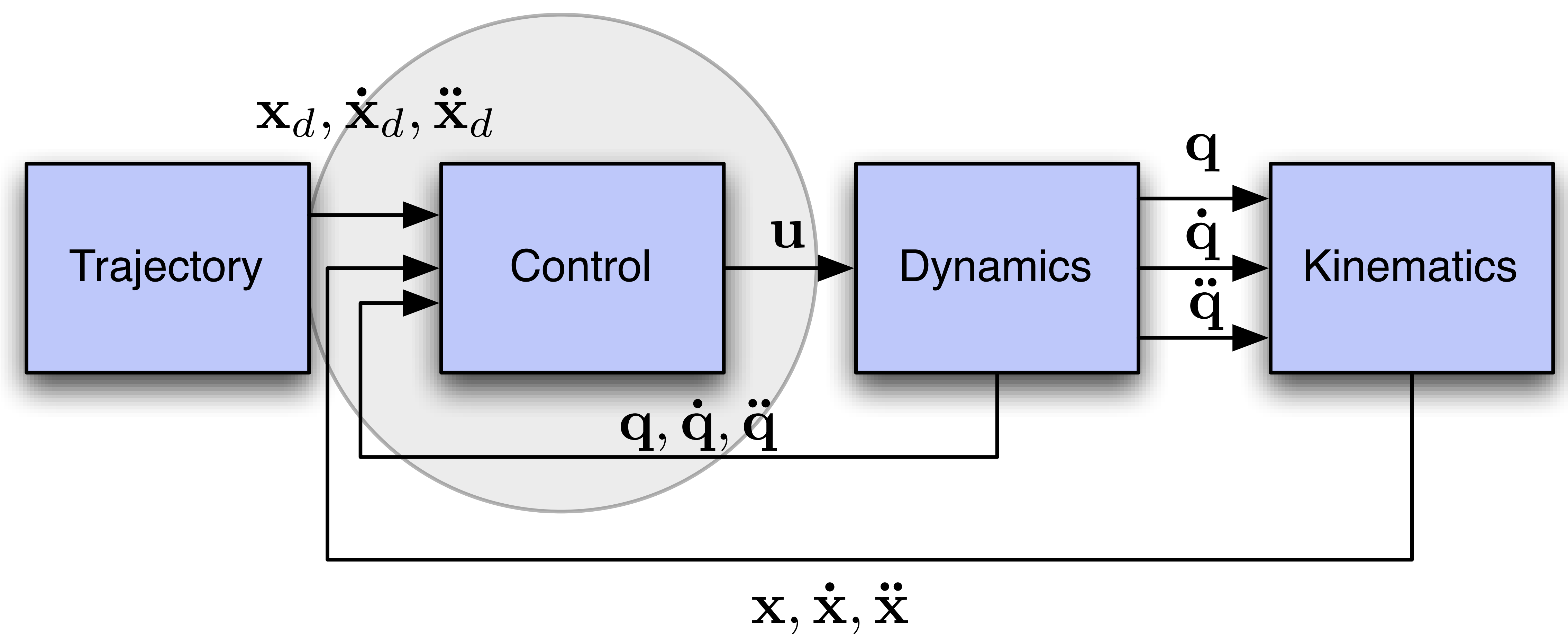
4. **Control in Joint Space**
   Linear Control
   Model-based Control

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

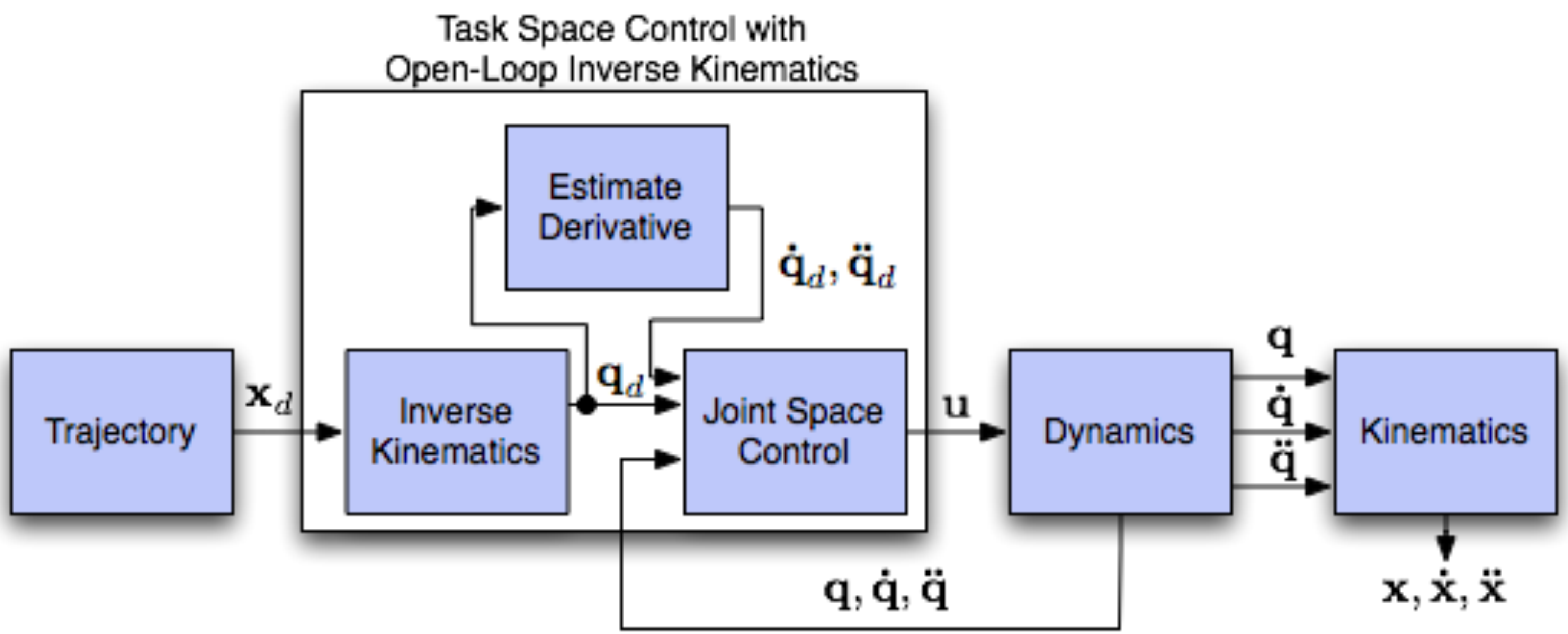
Assume your plan is in a task space...

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

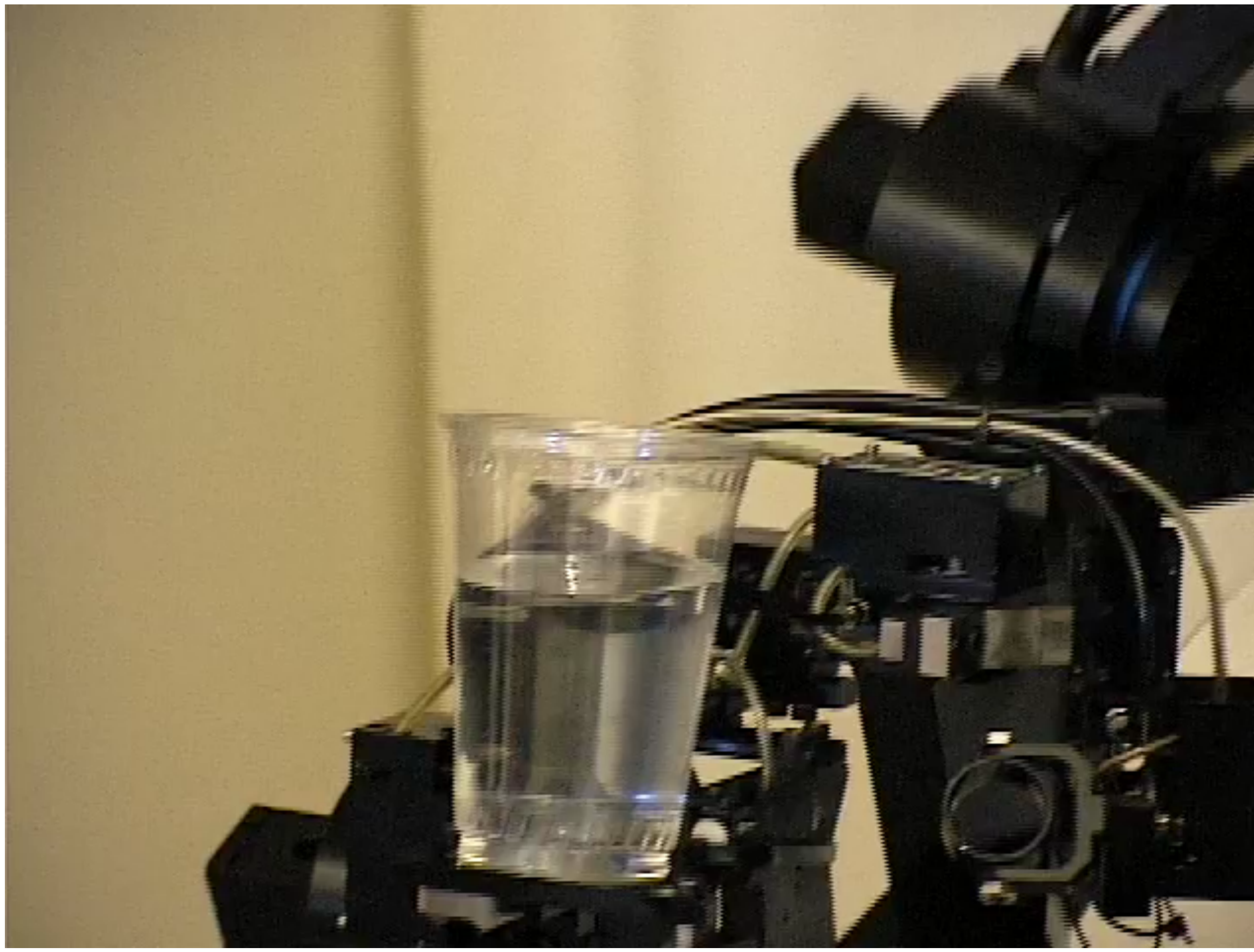
- Typically given in Cartesian coordinates
- Eventually also orientation



Why don't we try it this way?



Inverse Kinematics (IK)



How to move my joints in order to get to a given hand configuration?

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

- **What do we want to have?**
- **Inverse Kinematics:** A mapping from task space to configuration

$$\mathbf{q} = f^{-1}(\mathbf{x})$$

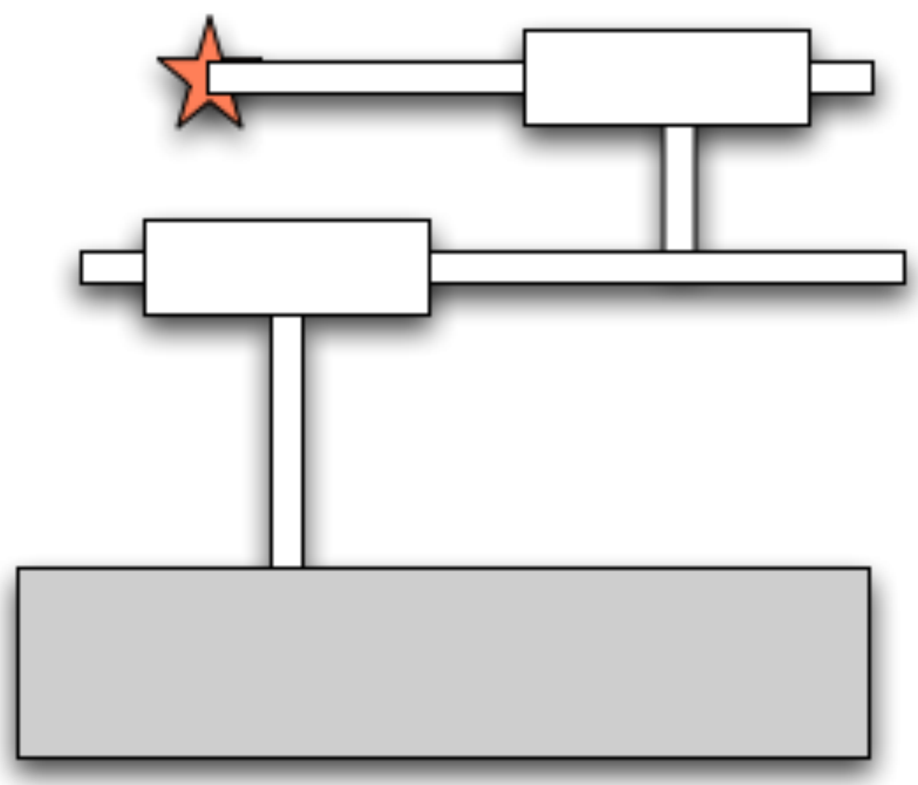
Example 1 - revisited

As $x = q_1 + q_2$

we have

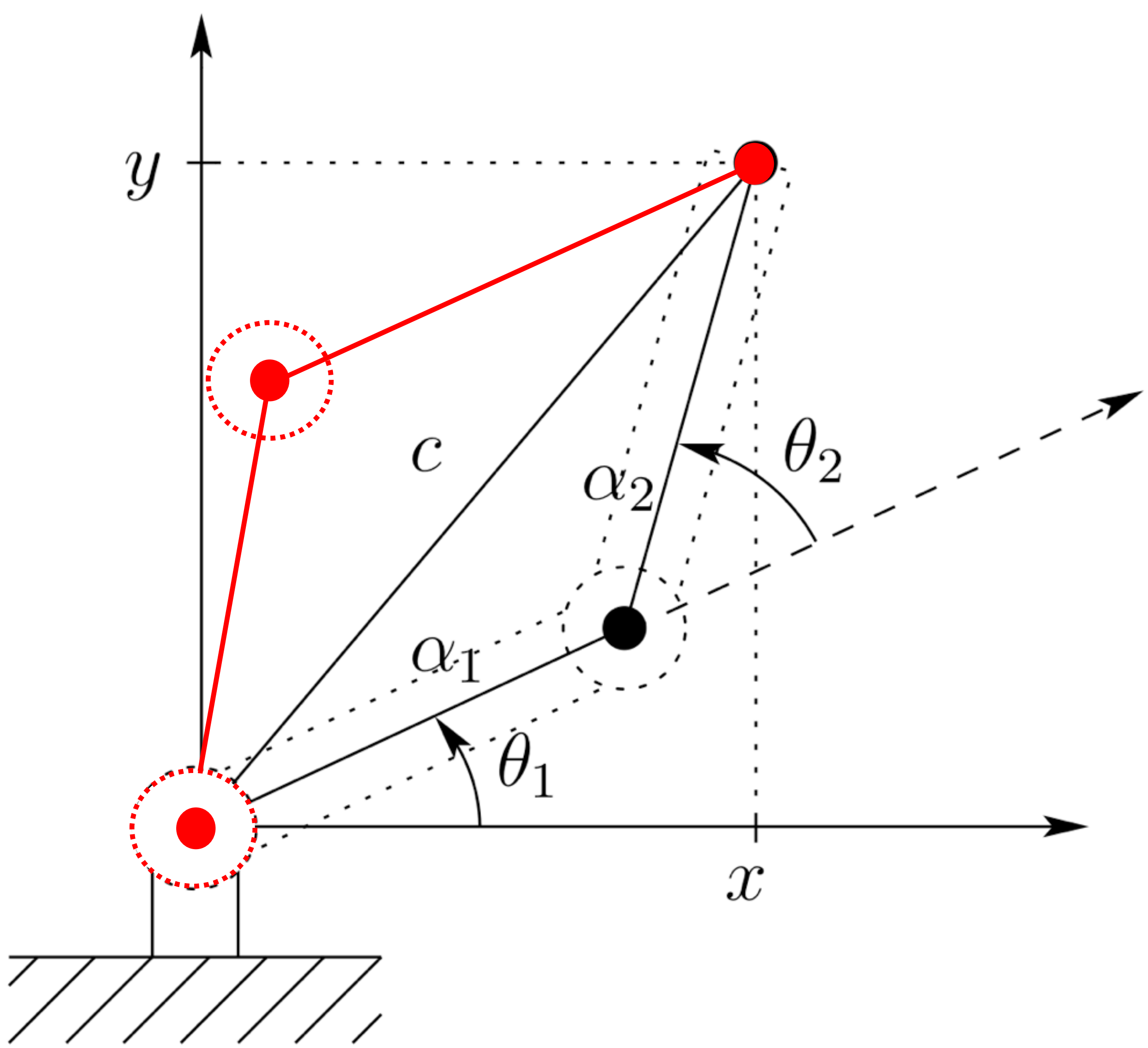
$$\begin{aligned} q_1 &= h \\ q_2 &= x - h \end{aligned}$$

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



➡ We have infinitely many solutions!!! Yikes!

Example 2 - revisited



We can solve for θ_1 and θ_2 and get

$$
\begin{aligned}
\theta_2 &= \cos^{-1}\left(\frac{x^2+y^2-\alpha_1^2-\alpha_2^2}{2\alpha_1\alpha 2}\right) \\
\theta_1 &= \tan^{-1}\left(\frac{y}{x}\right) \\
&- \tan^{-1}\left(\frac{\alpha_2 \sin\theta_2}{\alpha_1+\alpha_2\cos\theta_2}\right)
\end{aligned}
$$

- BUT: There is more than one solution!

- 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?

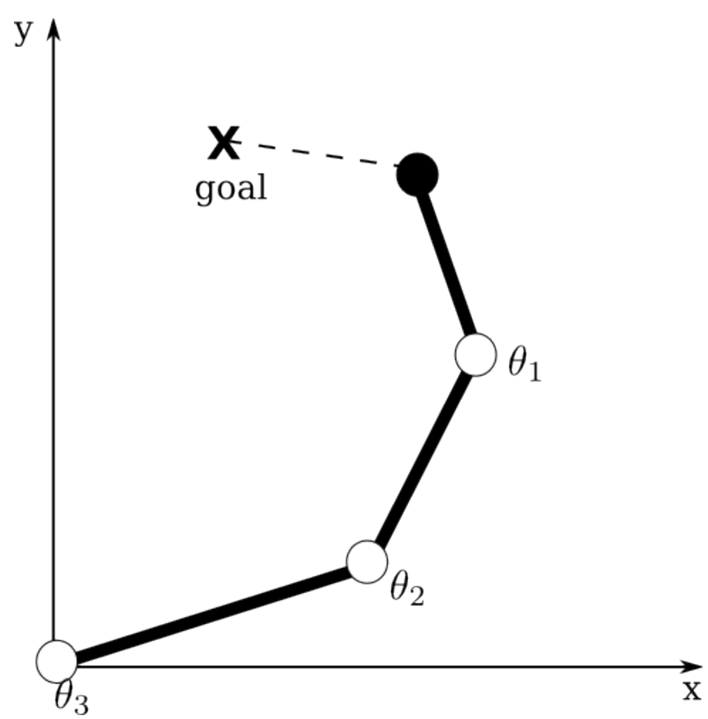
2. Modeling Robots
Kinematics
Dynamics

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:

$$\boldsymbol{q}_d = f^{-1}(\boldsymbol{x}_d)$$

➡ Not computable as we have an infinite amount of solutions

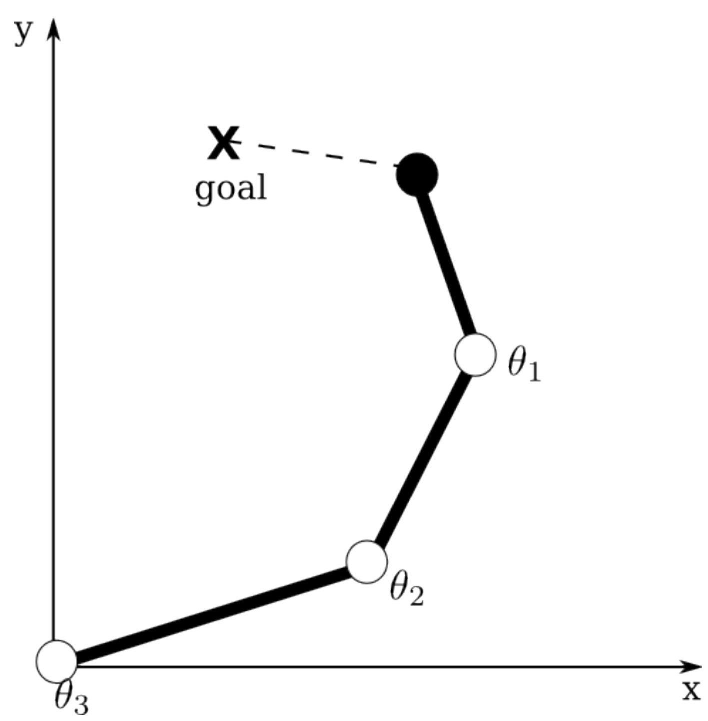
Differential inverse kinematics:

$$\dot{\boldsymbol{q}}_t = \boldsymbol{h}(\boldsymbol{x}_d, \boldsymbol{q}_t)$$

➡ Given current joint positions, compute joint velocities that minimizes the task space error

➡ Computable

Differential Inverse Kinematics



Differential inverse kinematics:

$$\dot{\boldsymbol{q}}_t = \boldsymbol{h}(\boldsymbol{x}_d, \boldsymbol{q}_t)$$

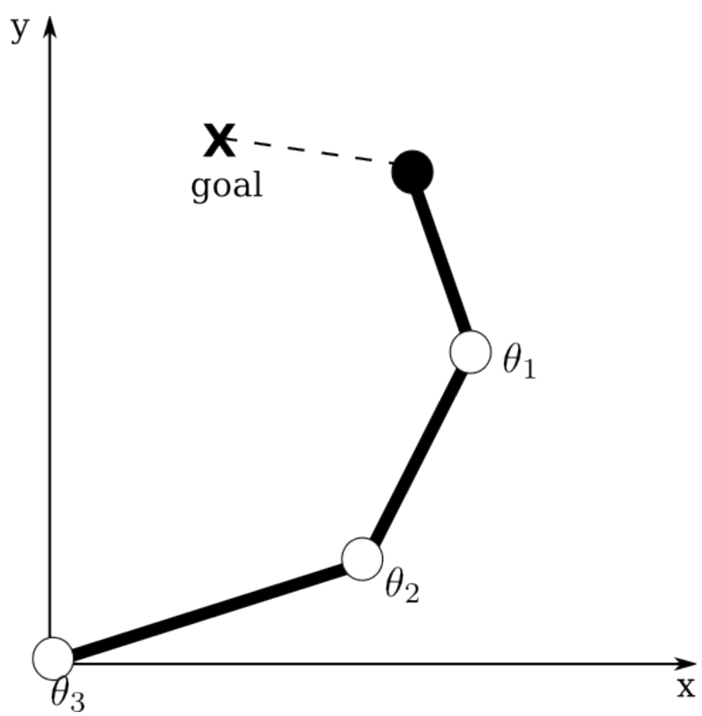
How can we use this for control?

1. Integrate $\dot{\boldsymbol{q}}_t$ and directly use it for joint space control
2. Iterate differential IK algorithm to find $\boldsymbol{q}_d$

   $$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + h(\boldsymbol{x}_d, \boldsymbol{q}_k)$$

   and plan trajectory to reach $\boldsymbol{q}_d$

Numerical Solution: Jacobian Transpose



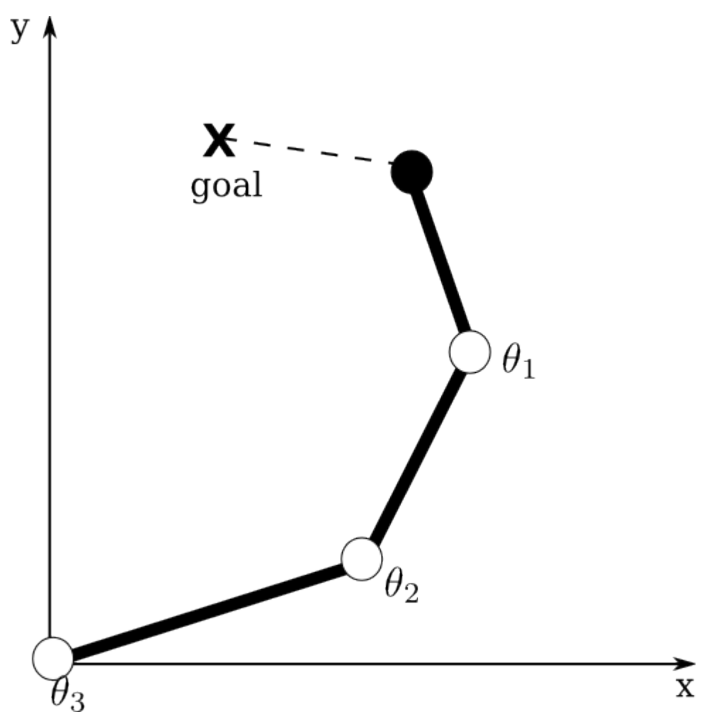
- Minimize the task-space error

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

- Gradient always points in the direction of steepest ascent

$$\begin{aligned} \frac{dE}{d\boldsymbol{q}} &= -(\boldsymbol{x} - f(\boldsymbol{q}))^T \frac{df(\boldsymbol{q})}{d\boldsymbol{q}} \\ &= -(\boldsymbol{x} - f(\boldsymbol{q}))^T \boldsymbol{J}(\boldsymbol{q}) \end{aligned}$$

Jacobian Transpose



Minimize error per gradient descent

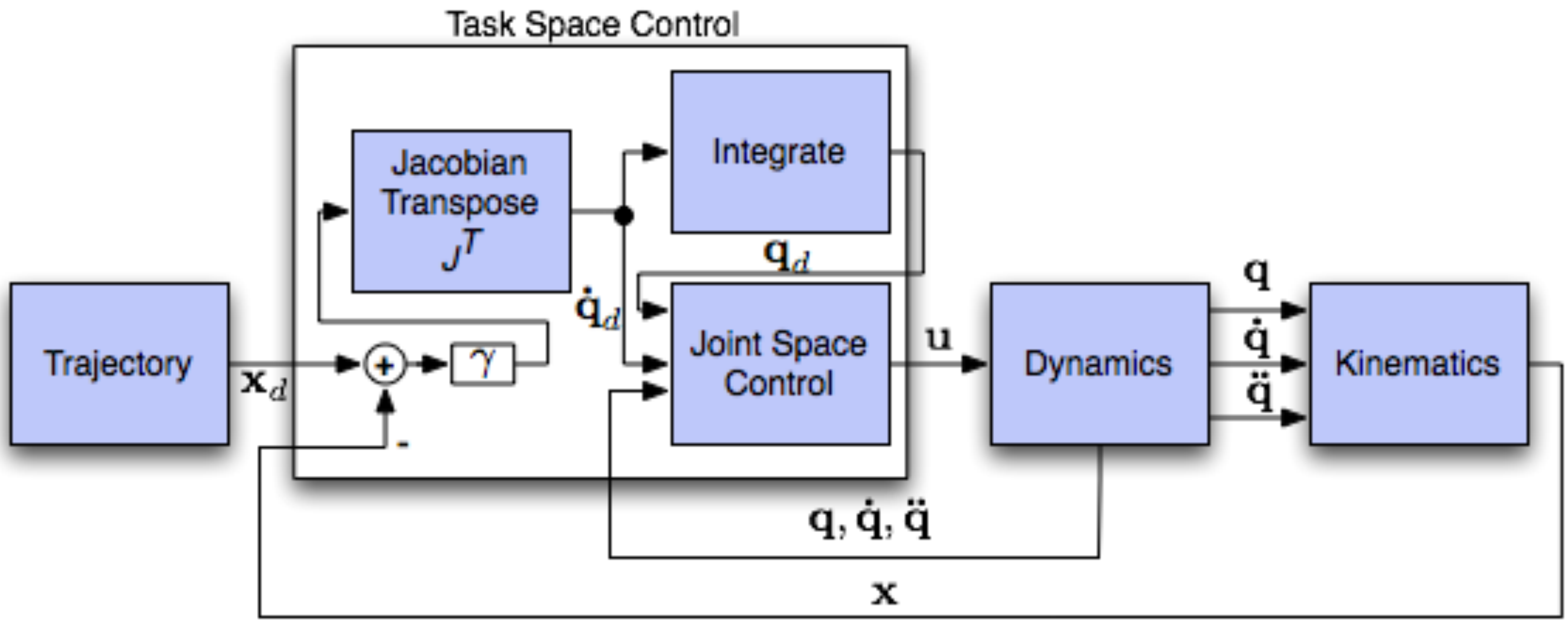
- Follow negative gradient with a certain step size γ

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

$$= \gamma \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{e}$$

- Known as Jacobian Transpose Method

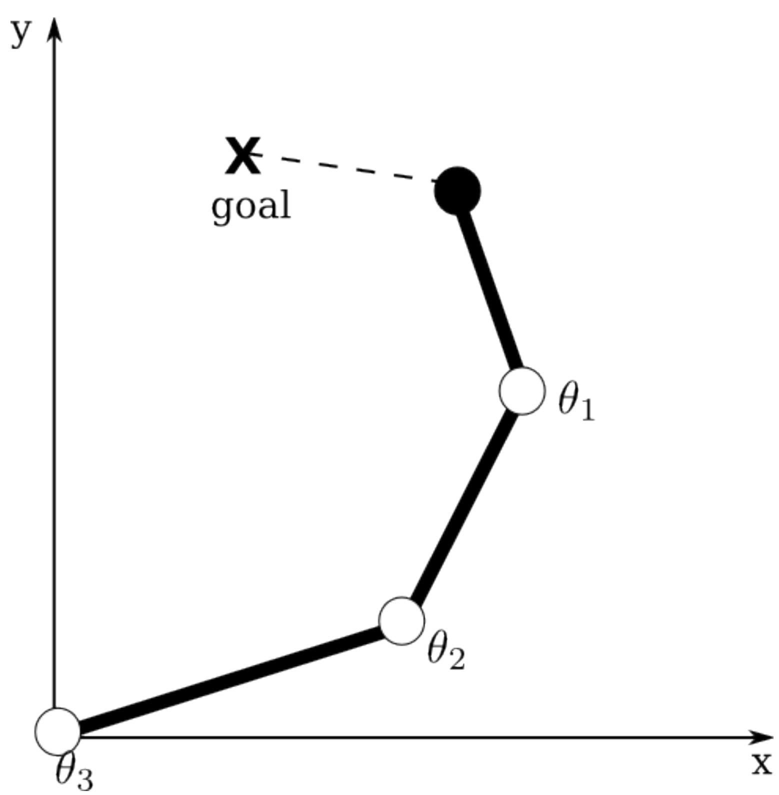
Control often found in robots…



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.

Jacobian Pseudo Inverse



- Assume that we are not so far from our solution manifold.
- Take smallest step $\dot{\boldsymbol{q}}$ that has a desired task space velocity

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

- Yields the following optimization problem

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

- **Solution:** (right) pseudo-inverse

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

  $$= \eta \boldsymbol{J}(\boldsymbol{q})^\dagger \boldsymbol{e}$$

Task-Prioritization with Null-Space Movements

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

- For example, "push" robot to a rest-posture

  $$\dot{\boldsymbol{q}}_0 = \boldsymbol{K}_P(\boldsymbol{q}_{\text{rest}} - \boldsymbol{q})$$

- Take step that has smallest distance to "base" task

  $$\min_{\dot{\boldsymbol{q}}} \; (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0)^T(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_0), \quad \text{s.t. } \dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$$

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

  - **Null-Space:** $(\boldsymbol{I} - \boldsymbol{J}^\dagger \boldsymbol{J})$

  - All movements $\dot{\boldsymbol{q}}_{\text{null}}$ that do not contradict the constraint

    $$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})(\dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}_{\text{null}}) \text{ or } \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}_{\text{null}} = 0$$

More advanced solutions

Similarly, we can also use a acceleration formulation

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

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

$$\begin{aligned} \min \quad & (\mathbf{u} - \mathbf{u}_0)^T(\mathbf{u} - \mathbf{u}_0) \\ s.t. \quad & \mathbf{A}(\mathbf{q}, \dot{\mathbf{q}}, t)\ddot{\mathbf{q}} = \dot{\mathbf{b}}(\mathbf{q}, \dot{\mathbf{q}}, t) \\ & \mathbf{u}_0 = \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t) \\ & \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{u} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) \end{aligned}$$

- 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
  - Nakanishi, J.;Cory, R.;Mistry, M.;Peters, J.;Schaal, S. (2008). Operational space control: A theoretical and emprical comparison, International Journal of Robotics Research, **27**, **6**, pp.737-757.
  - Peters, J.;Mistry, M.;Udwadia, F. E.;Nakanishi, J.;Schaal, S. (2008). A unifying methodology for robot control with redundant DOFs, Autonomous Robots, **24**, **1**, pp.1-12.

Singularity Problems

Problem: However, the inversion in the pseudo-inverse

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1}$$ can be problematic

In the case of singularities, $\boldsymbol{J}\boldsymbol{J}^{T}$ can not be inverted!

Damped Pseudo Inverse

Numerically more stable solution:

- ➡ Find a tradeoff between minimizing the error and keeping the joint movement small

$$\min_{\dot{\boldsymbol{q}}} \; (\dot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}})^T (\dot{\boldsymbol{x}} - \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}) + \lambda \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}}$$

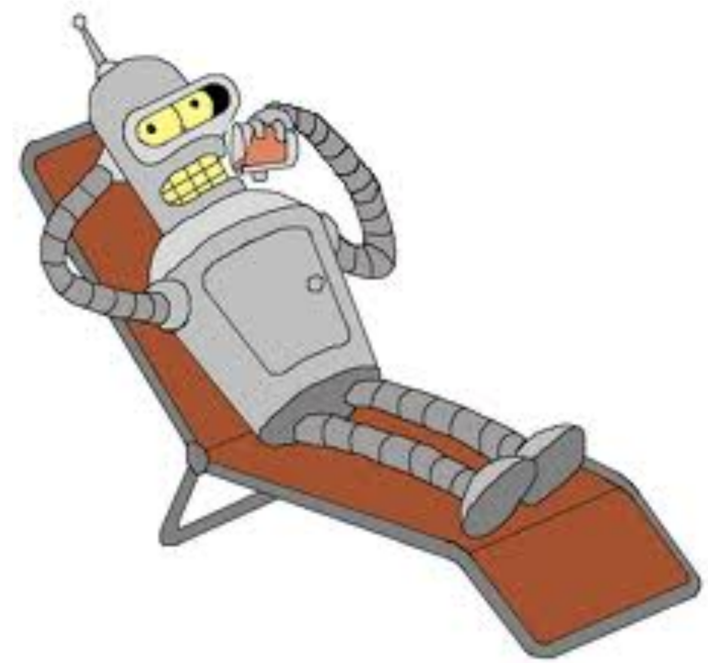
- ➡ Regularization constant λ
- ➡ Damped Pseudo Inverse Solution

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T + \lambda \boldsymbol{I})^{-1} \dot{\boldsymbol{x}} = \boldsymbol{J}^{\dagger(\lambda)} \dot{\boldsymbol{x}}$$

- ➡ Works much better for singularities

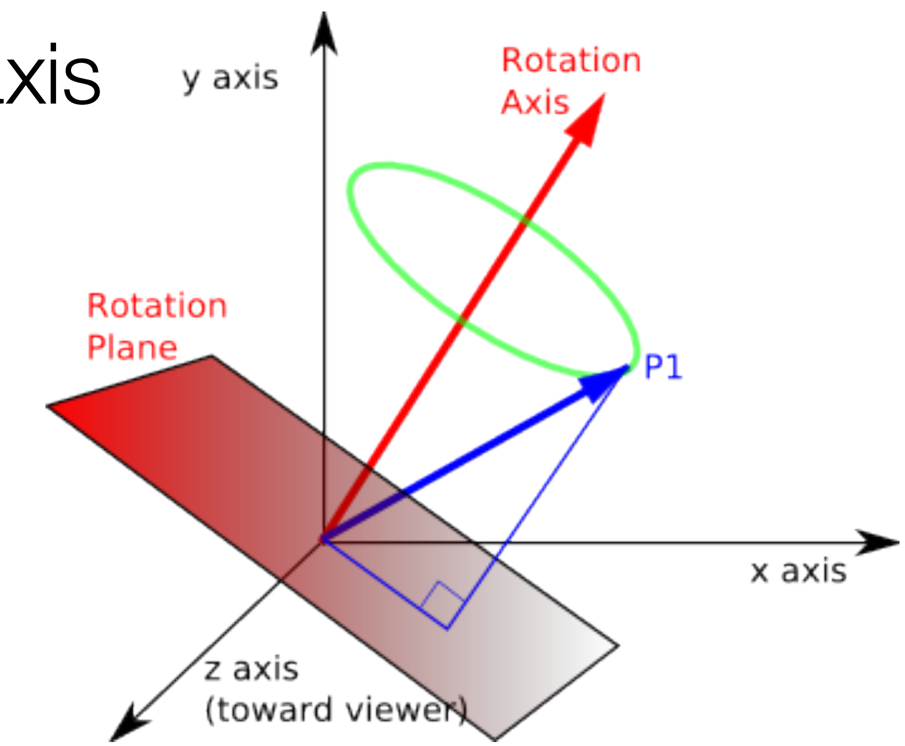
Ask questions...

Q & A?



Unit quaternion

Specify axis $\boldsymbol{r}$ and rotation angle ϑ around axis



- Quaternion is defined by $\mathcal{Q} = \{\eta, \boldsymbol{\epsilon}\}$

$$\eta = \cos\frac{\vartheta}{2} \qquad \boldsymbol{\epsilon} = \sin\frac{\vartheta}{2}\boldsymbol{r}$$

- Always normalized: $\eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1$
- Typically used for inverse kinematics (if we want to control orientation)

Unit quaternion

- Obtain rotation matrix $\boldsymbol{R}(\eta, \boldsymbol{\epsilon})$ from quaternion $\mathcal{Q}$:

$$\boldsymbol{R}(\eta, \boldsymbol{\epsilon}) = \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x \epsilon_y - \eta \epsilon_z) & 2(\epsilon_x \epsilon_z + \eta \epsilon_y) \\ 2(\epsilon_x \epsilon_y + \eta \epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y \epsilon_z - \eta \epsilon_x) \\ 2(\epsilon_x \epsilon_z - \eta \epsilon_y) & 2(\epsilon_y \epsilon_z + \eta \epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix}$$

- Obtain $\mathcal{Q}$ from rotation matrix $\boldsymbol{R}$

$$\eta = \frac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1} \qquad \boldsymbol{\epsilon} = \frac{1}{2}\begin{bmatrix} \mathrm{sgn}(r_{32} - r_{23})\sqrt{r_{11} - r_{22} - r_{33} + 1} \\ \mathrm{sgn}(r_{13} - r_{31})\sqrt{r_{22} - r_{33} - r_{11} + 1} \\ \mathrm{sgn}(r_{21} - r_{12})\sqrt{r_{33} - r_{11} - r_{22} + 1} \end{bmatrix}$$

- Inverse quaternion: $\mathcal{Q}^{-1} = \{\eta, -\boldsymbol{\epsilon}\}$