COM SCI 188

Intro to Robotics

Lecture 4

Yuchen Cui

Winter 2026

Agenda

- Announcements

- Quaternions
- Forward Kinematics (MR 4, Appendix C)
- Inverse Kinematics (MR 6)

- Robot Control (MR 11)

Announcements

- Problem Set 1 due Friday (you will need today's lecture)
- Coding Assignment 1 due next Friday
  - Discussion session tomorrow will help you get a head start!

3D Rotation Representation

There are many ways to specify rotation

- Rotation matrix
- Euler angles: 3 angles about 3 axes
- Axis-angle representation
- Quaternions

Euler's rotation theorem: any rotation or sequence of rotations of a rigid body in a three-dimensional space is equivalent to a single rotation about a single fixed axis.

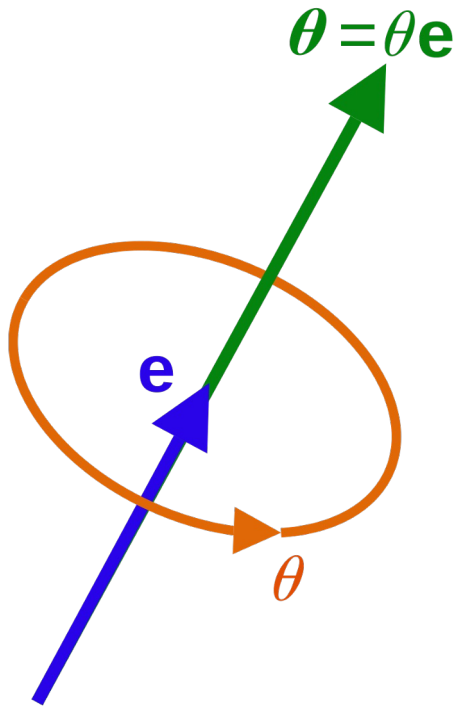
Who's Euler?

Known as the "master of us all," **Leonhard Euler** was the ultimate math **G.O.A.T.** who invented the symbols we use today—like $f(x)$, e, and π—and was so legendary that he founded entire fields like **graph theory** while writing up to **one-third of all 18th-century research** in mathematics.

Axis Angle

Euler's rotation theorem: any rotation or sequence of rotations of a rigid body in a three-dimensional space is equivalent to a single rotation about a single fixed axis.

Axis-angle representation:
Parameterize a 3D rotation by two quantities: **a unit vector e** indicating the direction of an axis of rotation, and **an angle θ** describing the magnitude of the rotation about the axis.



Quaternion:

Quaternions: uses a unit four dimensional vector **(x,y,z,w)** to represent rotation.
If the rotation is (v1,v2,v3, θ) in angle-axis representation. It can be written in quaternion as:

$$x = v_1 \sin\frac{\theta}{2}$$
$$y = v_2 \sin\frac{\theta}{2}$$
$$z = v_3 \sin\frac{\theta}{2}$$
$$w = \cos\frac{\theta}{2}$$

$$x^2 + y^2 + z^2 + w^2 = 1$$

4-dimensional vector on a 4D sphere

Note: here x,y,z, w is just variable name, not coordinate

Quaternion:

Quaternions: uses a unit four dimensional vector **(x,y,z,w)** to represent rotation.
If the rotation is (v1,v2,v3, θ) in angle-axis representation. It can be written in quaternion as:

$$x = v_1 \sin\frac{\theta}{2}$$
$$y = v_2 \sin\frac{\theta}{2}$$
$$z = v_3 \sin\frac{\theta}{2}$$
$$w = \cos\frac{\theta}{2}$$

Quaternions are a very popular parametrization due to the following properties:

- More compact than the matrix representation (4 numbers instead of 9 numbers)
- The quaternion elements vary continuously over the unit sphere in $\mathbb{R}^4$ as the orientation changes, avoiding discontinuous jumps *(it is important for many optimization or learning algorithms)*

Quaternion:

Quaternions: examples

$$x = v_1 \sin\frac{\theta}{2}$$

$$y = v_2 \sin\frac{\theta}{2}$$

$$z = v_3 \sin\frac{\theta}{2}$$

$$w = \cos\frac{\theta}{2}$$

(0,0,0,1)	**Identity**
(1,0,0,0)	**Rotate along x by π, (*cos(θ/2)=0, θ/2=π/2*)**
(0,1,0,0)	**Rotate along y by π**
(0,0,1,0)	**Rotate along z by π**

Quaternion:

Quaternions: examples

$$x = v_1 \sin\frac{\theta}{2}$$

$$y = v_2 \sin\frac{\theta}{2}$$

$$z = v_3 \sin\frac{\theta}{2}$$

$$w = \cos\frac{\theta}{2}$$

(0,0,0,1)	**Identity**
(1,0,0,0)	**Rotate along x by π**
(0,1,0,0)	**Rotate along y by π**
(0,0,1,0)	**Rotate along z by π**

Rotation axis

Rotation angle cos(θ/2)

Quaternion:

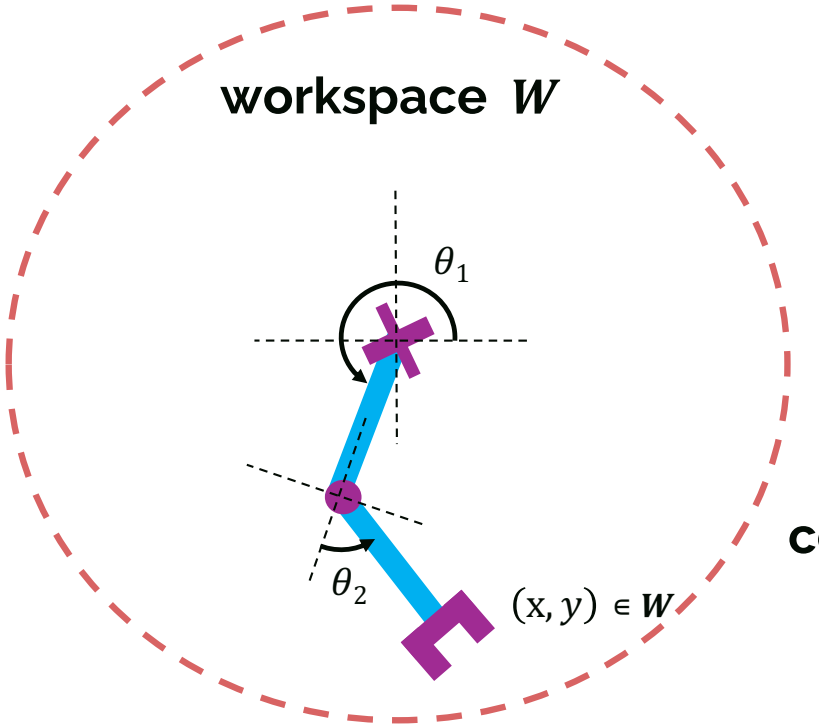
Quaternions: examples

$$x = v_1 \sin\frac{\theta}{2}$$
$$y = v_2 \sin\frac{\theta}{2}$$
$$z = v_3 \sin\frac{\theta}{2}$$
$$w = \cos\frac{\theta}{2}$$

(0,0,0,1)	**Identity**
(1,0,0,0)	**Rotate along x by π**
(0,1,0,0)	**Rotate along y by π**
(0,0,1,0)	**Rotate along z by π**
$(\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}})$	**Rotate along x by $\pi/2$ (90˚)**

Recap

task space: 2D plane



Forward Kinematics

FK: $C \rightarrow W$

$$FK((\theta_1, \theta_2)) = (x, y)$$

Inverse Kinematics

IK: $W \rightarrow C$

$$IK((x, y)) = (\theta_1, \theta_2)$$

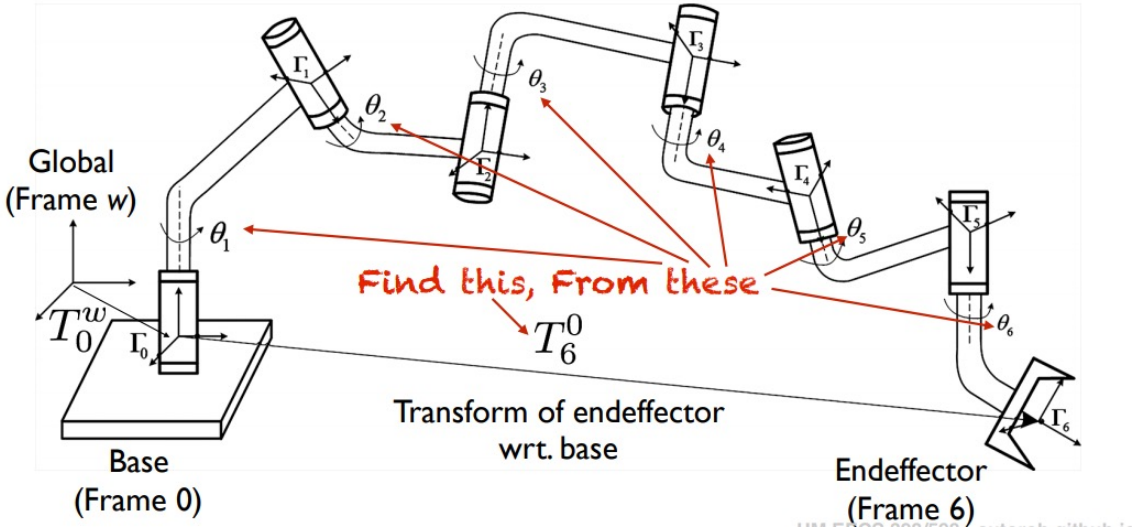
This is not a proper function.
Many configuration can lead to the same end-effector pose.

configuration space C

$(\theta_1, \theta_2) \in C$

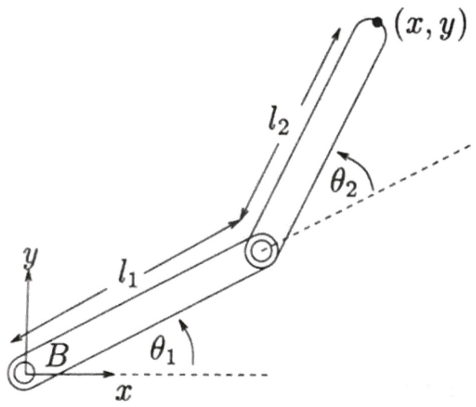
Recap

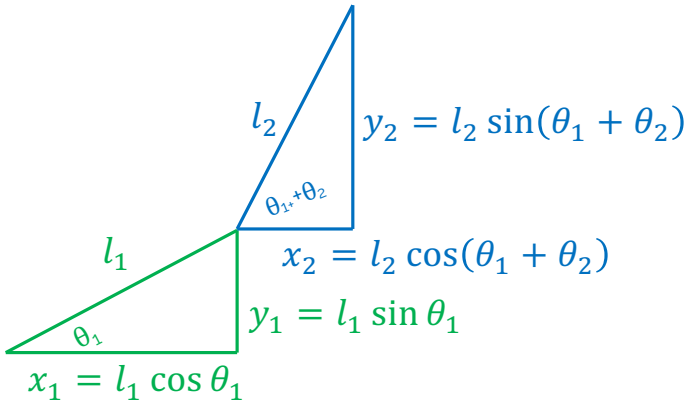
Forward Kinematics



Recap: **Forward Kinematics of 2-link Manipulator**

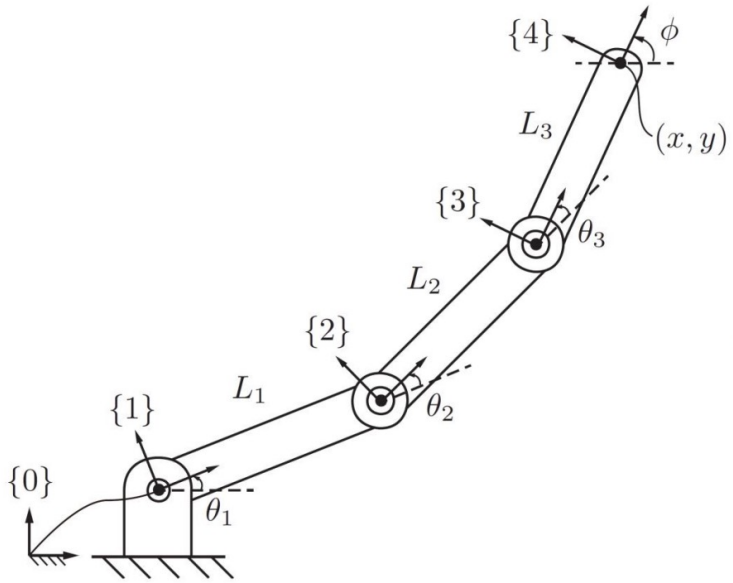
Given joint angles, calculate position of end-effector





$$x = l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2)$$
$$y = l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2)$$

Recap: **Forward Kinematics of RRR open-chain**



Forward kinematics of a 3R planar open chain.

$$T_{04} = T_{01}T_{12}T_{23}T_{34}$$

$$T_{01} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{12} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & L_1 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

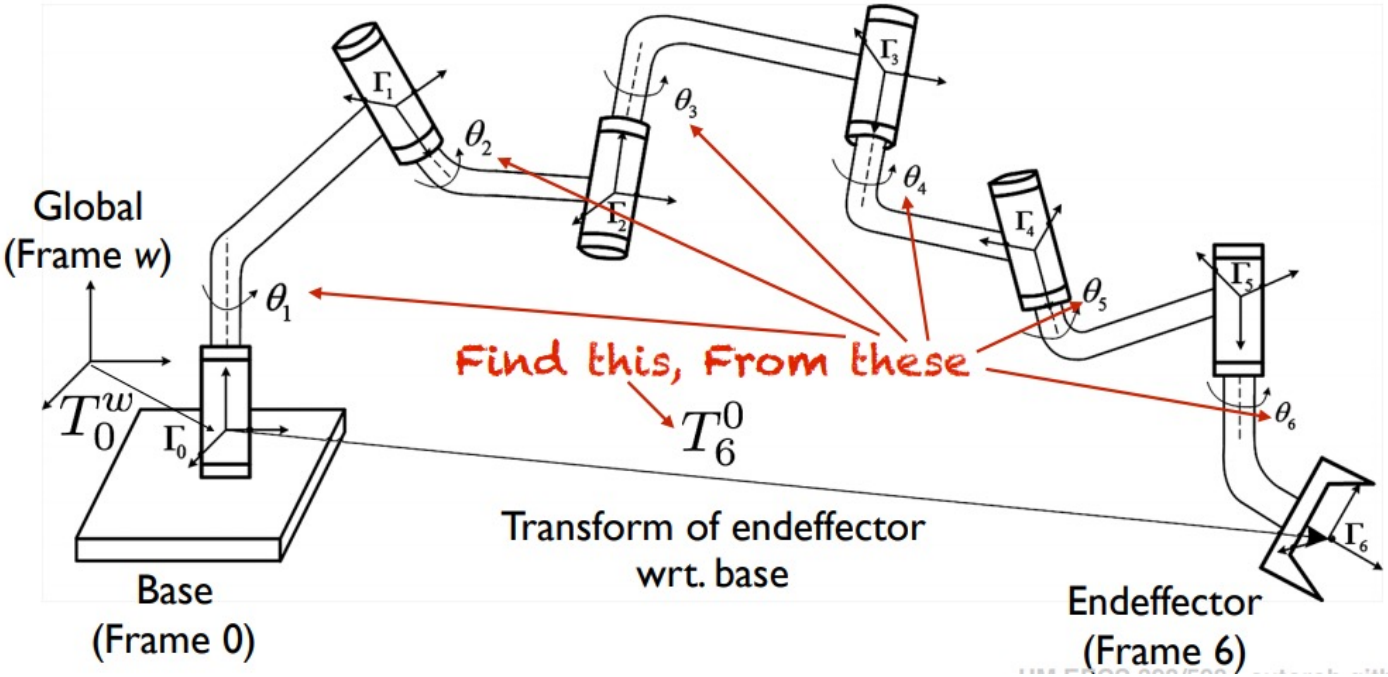
$$T_{23} = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{34} = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$T_{i-1,i}$ Depends only on the joint variable θ_i

Denavit–Hartenberg (DH) parameters

(MR Appendix C)

Purpose: To derive the forward kinematics (end-effector position) of a robot arm by defining transformations between link coordinate frames.



$$T_{0n}(\theta_1, \ldots, \theta_n) = T_{01}(\theta_1)T_{12}(\theta_2)\cdots T_{n-1,n}(\theta_n)$$

DH Parameters

$$T_{0n}(\theta_1, \ldots, \theta_n) = T_{01}(\theta_1) T_{12}(\theta_2) \cdots T_{n-1,n}(\theta_n)$$

Link twist Link length Link offset Joint angle

$$T_{i-1,i} = \mathrm{Rot}(\hat{\mathrm{x}}, \alpha_{i-1}) \mathrm{Trans}(\hat{\mathrm{x}}, a_{i-1}) \mathrm{Trans}(\hat{\mathrm{z}}, d_i) \mathrm{Rot}(\hat{\mathrm{z}}, \phi_i)$$

DH Parameters

- The length of the mutually perpendicular line, denoted by the scalar a_{i-1}, is called the **link length** of link $i-1$. Despite its name, this link length does not necessarily correspond to the actual length of the physical link.
- The **link twist** α_{i-1} is the angle from $\hat{z}_{i-1}$ to $\hat{z}_i$, measured about $\hat{x}_{i-1}$.
- The **link offset** d_i is the distance from the intersection of $\hat{x}_{i-1}$ and $\hat{z}_i$ to the origin of the link-i frame (the positive direction is defined to be along the $\hat{z}_i$-axis).
- The **joint angle** ϕ_i is the angle from $\hat{x}_{i-1}$ to $\hat{x}_i$, measured about the $\hat{z}_i$-axis.

Wikipedia definition is **different**

$$
\begin{aligned}
\mathrm{Rot}(\hat{\mathrm{x}}, \alpha_{i-1}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
\mathrm{Trans}(\hat{\mathrm{x}}, a_{i-1}) &= \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
\mathrm{Trans}(\hat{\mathrm{z}}, d_i) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \\
\mathrm{Rot}(\hat{\mathrm{z}}, \phi_i) &= \begin{bmatrix} \cos\phi_{i-1} & -\sin\phi_{i-1} & 0 & 0 \\ \sin\phi_{i-1} & \cos\phi_{i-1} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.
\end{aligned}
$$

DH Parameters

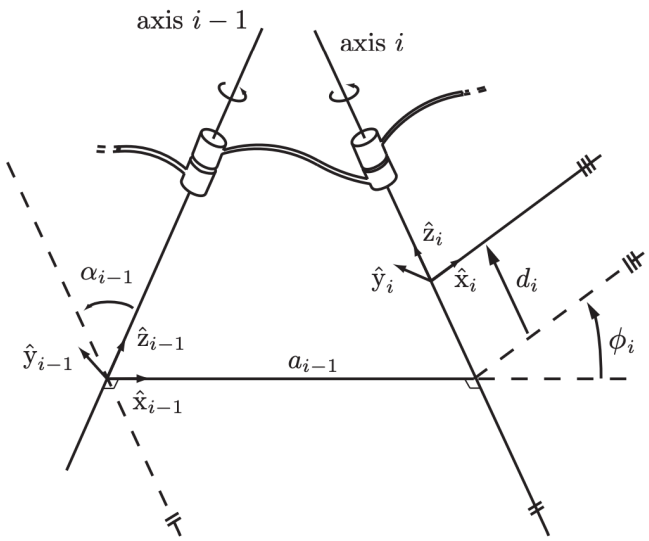
$$T_{0n}(\theta_1, \ldots, \theta_n) = T_{01}(\theta_1) T_{12}(\theta_2) \cdots T_{n-1,n}(\theta_n)$$

Link twist Link length Link offset Joint angle

$$T_{i-1,i} = \mathrm{Rot}(\hat{\mathrm{x}}, \alpha_{i-1}) \mathrm{Trans}(\hat{\mathrm{x}}, a_{i-1}) \mathrm{Trans}(\hat{\mathrm{z}}, d_i) \mathrm{Rot}(\hat{\mathrm{z}}, \phi_i)$$

$$= \begin{bmatrix} \cos\phi_i & -\sin\phi_i & 0 & a_{i-1} \\ \sin\phi_i \cos\alpha_{i-1} & \cos\phi_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i \sin\alpha_{i-1} \\ \sin\phi_i \sin\alpha_{i-1} & \cos\phi_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DH Parameters

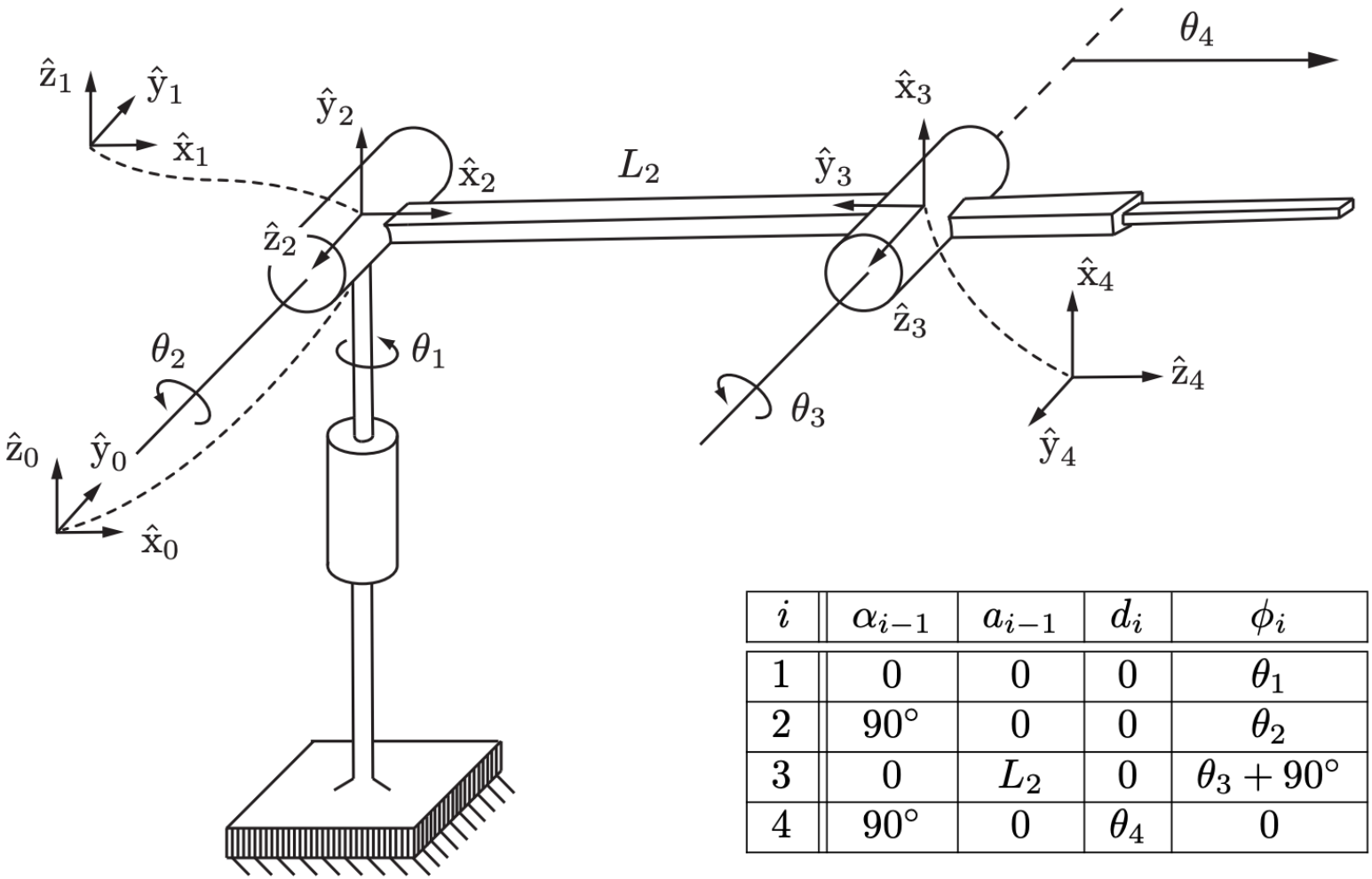


The first rule is that the $\hat{z}_i$-axis coincides with joint axis i and the $\hat{z}_{i-1}$-axis coincides with joint axis $i-1$. The direction of positive rotation about each link's $\hat{z}$-axis is determined by the right-hand rule.

Once the $\hat{z}$-axis direction has been assigned, the next rule determines the origin of the link reference frame. First, find the line segment that orthogonally intersects both the joint axes $\hat{z}_{i-1}$ and $\hat{z}_i$. For now let us assume that this line segment is unique; the case where it is not unique (i.e., when the two joint axes are parallel), or fails to exist (i.e., when the two joint axes intersect), is addressed later. Connecting joint axes $i-1$ and i by a mutually perpendicular line, the origin of frame $\{i-1\}$ is then located at the point where this line intersects joint axis $i-1$.

Determining the remaining $\hat{x}$- and $\hat{y}$-axes of each link reference frame is now straightforward: the $\hat{x}$-axis is chosen to be in the direction of the mutually perpendicular line pointing from the $(i-1)$-axis to the i-axis. The $\hat{y}$-axis is then uniquely determined from the cross product $\hat{x} \times \hat{y} = \hat{z}$. Figure C.1 depicts the link frames $\{i\}$ and $\{i-1\}$ chosen according to this convention.

You won't be asked to assign the frames!

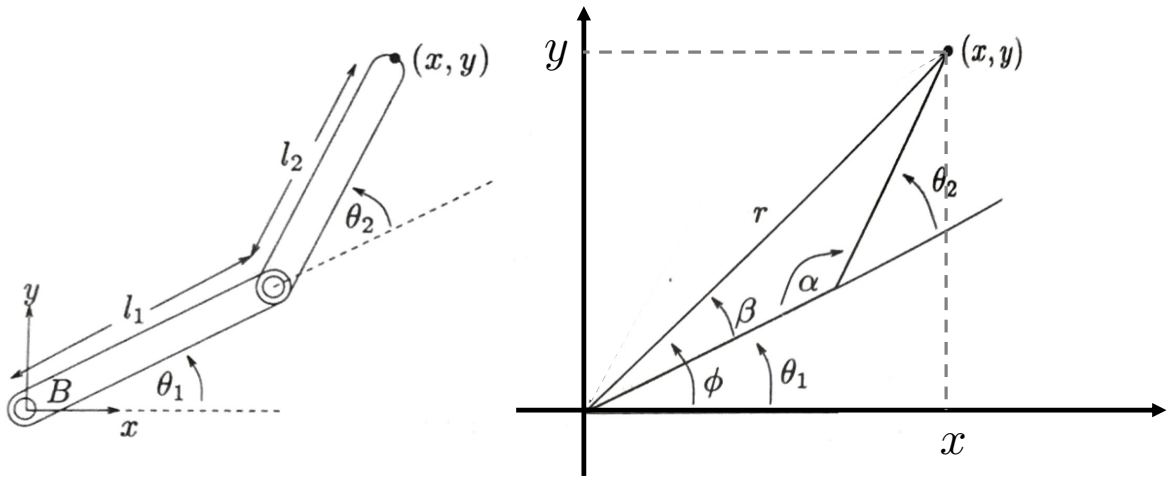


i	α_{i-1}	a_{i-1}	d_i	ϕ_i
1	0	0	0	θ_1
2	90°	0	0	θ_2
3	0	L_2	0	$\theta_3 + 90°$
4	90°	0	θ_4	0

Inverse Kinematics

Inverse Kinematics

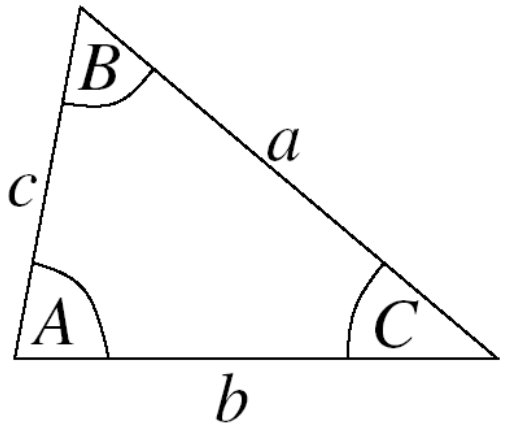
Given end-effector position, calculate joint angles



$$x = l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2)$$
$$y = l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2)$$

$$r^2 = x^2 + y^2$$

sin and cos rules:



Sine Rule

$$\frac{a}{\sin(A)} = \frac{b}{\sin(B)} = \frac{c}{\sin(C)} \quad \text{or} \quad \frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c}$$

(for finding sides) (for finding angles)

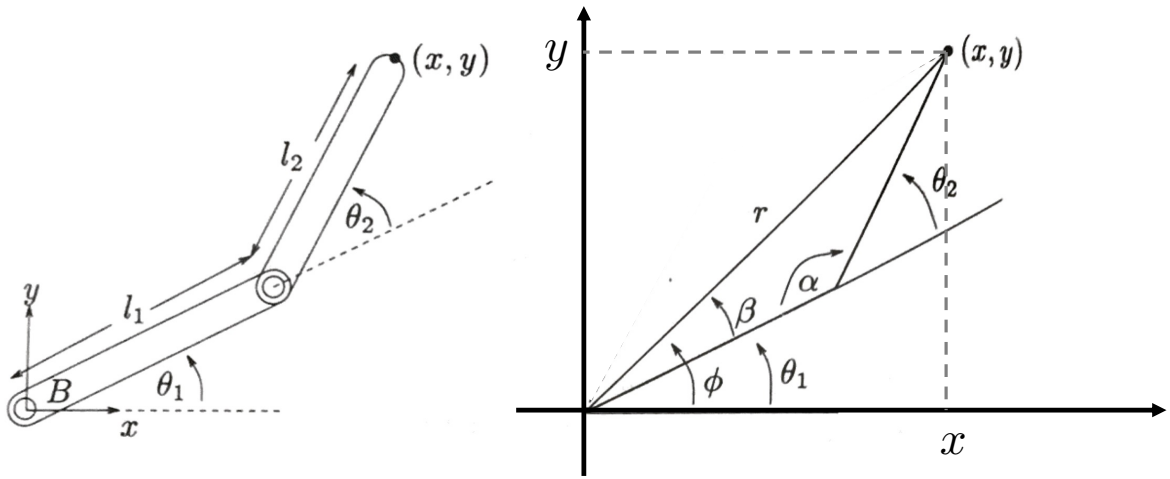
Cosine Rule

$$a^2 = b^2 + c^2 - 2bc\cos(A) \quad \text{or} \quad \cos(A) = \frac{b^2 + c^2 - a^2}{2bc}$$

(for finding sides) (for finding angles)

Inverse Kinematics

Given end-effector position, calculate joint angles

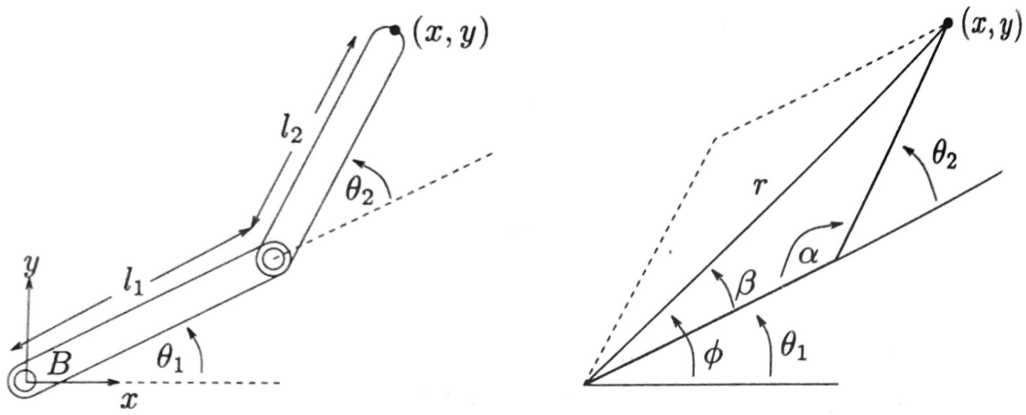


$$x = l_1 cos\theta_1 + l_2 cos(\theta_1 + \theta_2)$$
$$y = l_1 sin\theta_1 + l_2 sin(\theta_1 + \theta_2)$$

$$r^2 = x^2 + y^2$$
$$r^2 = l_1^2 + l_2^2 - 2l_1 l_2 \cos(\alpha)$$
$$l_2^2 = l_1^2 + r^2 - 2l_1 r \cos(\beta)$$

Inverse Kinematics

Given end-effector position, calculate joint angles



$$\theta_2 = \pi \pm \alpha \qquad \alpha = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - r^2}{2 l_1 l_2}\right)$$

If $a \neq 0$, there are two distinct values of q_2 which give the appropriate radius — the *flip solution* is shown dashed above

$$\theta_1 = \operatorname{atan2}(y, x) \pm \beta \qquad \beta = \cos^{-1}\left(\frac{r^2 + l_1^2 - l_2^2}{2 l_1 r}\right)$$

Solve for f and use this to get q_1 for **both** possible q_2 values

Inverse Kinematics

Inverse kinematics: given end-effector position, calculate joint angles

Inverse kinematics for robots with more than two joints generally does not admit a unique closed-form solution

Closed-form solutions are possible when:

- The robot has a simple structure (like planar arms).
- The robot meets certain geometric conditions (e.g., all joints aligned in a certain way).
- The number of degrees of freedom (DoF) matches the task space dimensions exactly.

What to do instead?

Numerical IK

- Iterative solvers (like Newton-Raphson, Jacobian pseudo-inverse).
- Solves for joint angles by minimizing position/orientation error.
- Used in most general-purpose robot controllers.

Optimization-Based IK

- Define an objective function (like minimizing joint torque or staying within limits).
- Add constraints (collision, joint bounds, etc.).
- Solvers: gradient descent, SQP, or even MPC.

Learning-Based IK

- Neural networks or reinforcement learning.
- Especially helpful in high-DoF or redundant robots (like humanoids or octopuses).

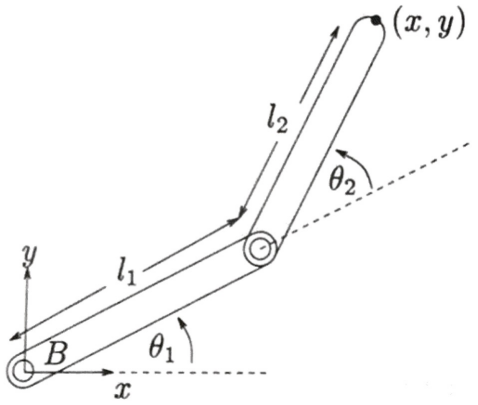
Inverse Dynamics

Given joint velocities, find end-effector velocity

$$x = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)$$
$$y = L_2 \sin\theta_1 + L_2 \sin(\theta_1 + \theta_2)$$

First, differentiate with respect to joint angles

$$\frac{\partial x}{\partial \theta_1} = -L_1 \sin\theta_1 - L_2 \sin(\theta_1 + \theta_2)$$
$$\frac{\partial x}{\partial \theta_2} = -L_2 \sin(\theta_1 + \theta_2)$$
$$\frac{\partial y}{\partial \theta_1} = L_1 \cos\theta_1 + L_2 \cos(\theta_1 + \theta_2)$$
$$\frac{\partial y}{\partial \theta_2} = L_2 \cos(\theta_1 + \theta_2)$$



Jacobian Matrix

Form *Jacobian* matrix:

the matrix of all first-order partial derivatives of a vector-valued function

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix}$$

Velocity of end-effector:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

Inverse of Jacobian

Jacobian is used for inverse dynamics

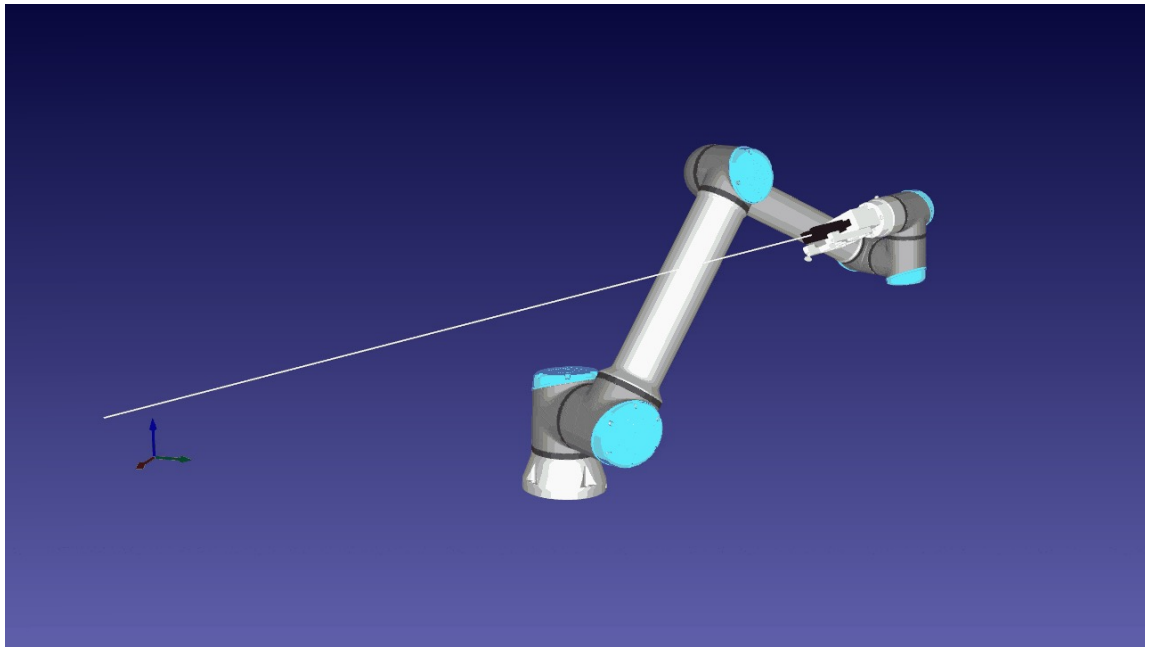
$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix}^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}$$

Can be used for closed-loop control

Inverse of Jacobian

Manipulator has **singularity** when determinant of Jacobian is zero

Difficult to control around singularity



A singularity is a configuration where the robot temporarily loses (or gains) control degrees of freedom due to geometry, making IK and control unstable.

Control

slides adapted from USC CSCI 445

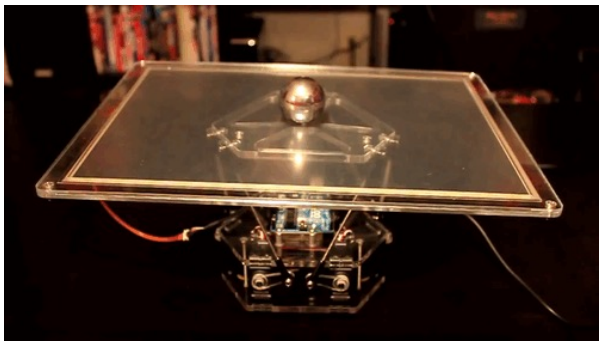
The Role of Control

- Many tasks in robotics are defined by *achievement goals*:
  - Make a cup of coffee
  - Drive to a delivery point

Typically AI (e.g. search, supervised learning)

- Other tasks in robotics are defined by *maintenance goals*:
  - Drive at 75 mph
  - Maintain a distance to front car at 2-car length
  - Keep balance

Typically Control Theory

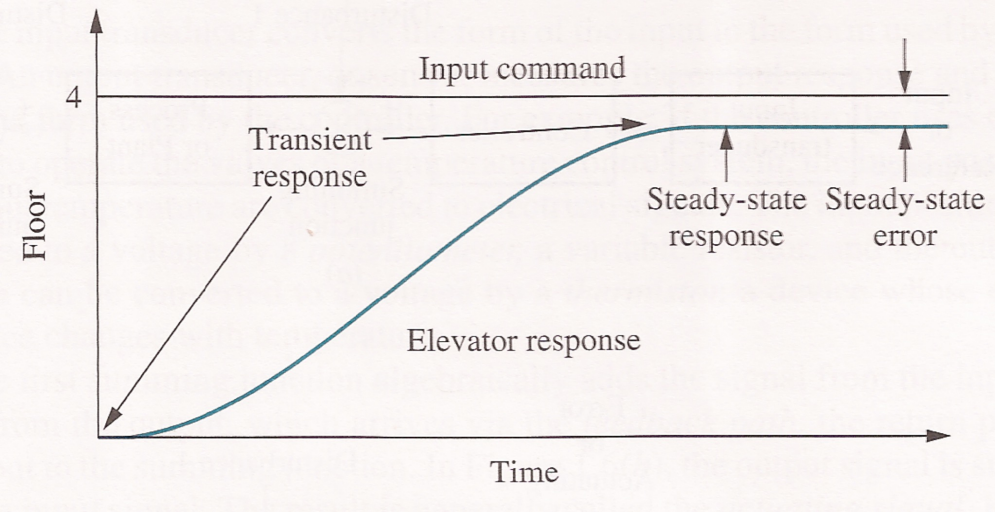


The Role of Control

- Control theory is generally used for low-level maintenance goals
- General notions:
  - **output** = *Controller*(**input**)
  - output is control signal to actuator (e.g., motor voltage/current)
  - input is either goal state or goal state error (e.g., desired motor velocity)

Control Systems

- Provides an output or response for a given input or stimulus
  - Input: desired response
  - Output: actual response
  - E.g. pressing 4th floor button on an elevator



Output ≠ Input

Transient Response:
Instantaneous change of input but gradual change of output

Steady-state Error:
Accuracy of leveling

Steady state error is inherent!

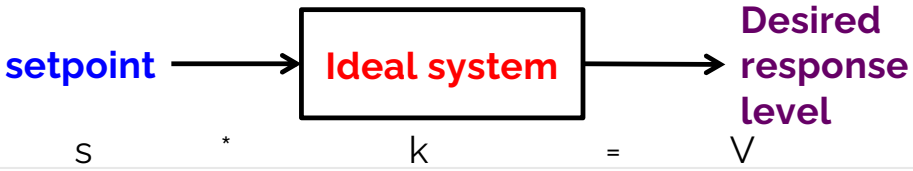
Open loop (feedforward) control

- Open loop controller:
  - **output** = *FF*(**goal**)
- E.g.: motor speed controller (linear):
  - V is applied voltage on motor
  - s is goal speed
  - k is gain term (from calibration)
- How do we know we've reached the goal?
- Weakness:
  - Varying load on motor: *motor may not maintain goal speed*

For what types of tasks/systems is this useful?

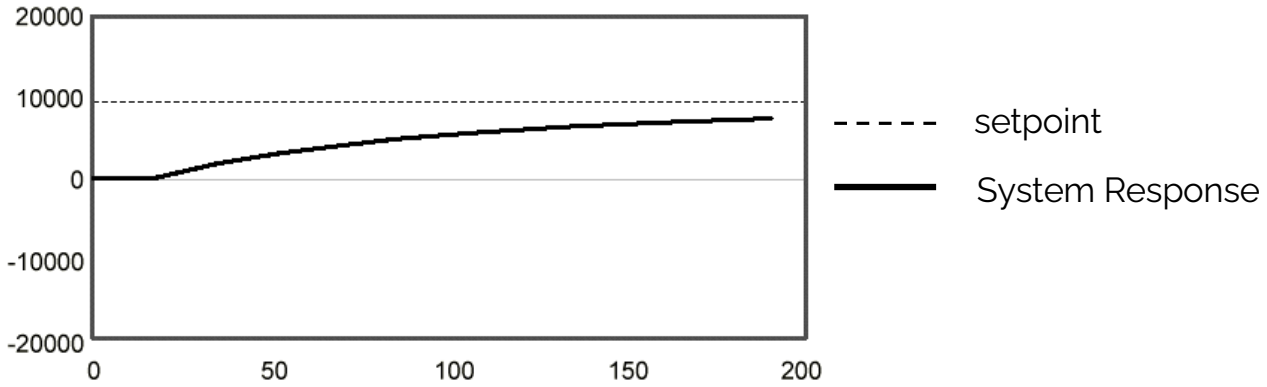
$$V = ks$$

When can feedforward be useful for robotics?



Uses of Open Loop Control

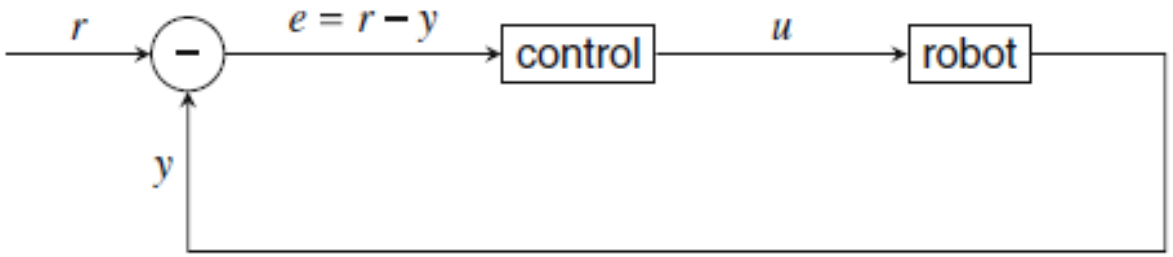
- Industrial machines may use open loop control
  - Biological systems use it, in various movements
  - These are called ballistic movements
  - Ballistic movements cannot be corrected while they are executed
  - E.g., pouncing, reflex reaching & withdrawal, etc.



Definition of Feedback Control

- Feedback control is a means of getting a system to achieve & maintain a desired state by *continuously feeding back* the current state and comparing it to the desired state, then adjusting the current state to minimize the difference.

- Feedback control systems are drawn in a traditional diagrammatic way



Comparing Methods

- **Feedforward**
  - Anticipative
  - Previous plan
  - Doesn't wait
- **Feedback**
  - Reactive / Responsive
  - Seeks to correct errors
  - Always too late!

Goal/Desired State

- The desired state is also called the **set point** or the **goal state** of the system.

- Goal state can be:
  - *external*
    (e.g., a thermostat monitors and controls the temperature of the house)
  - *internal*
    (e.g., a robot can monitor its battery and control its energy usage)

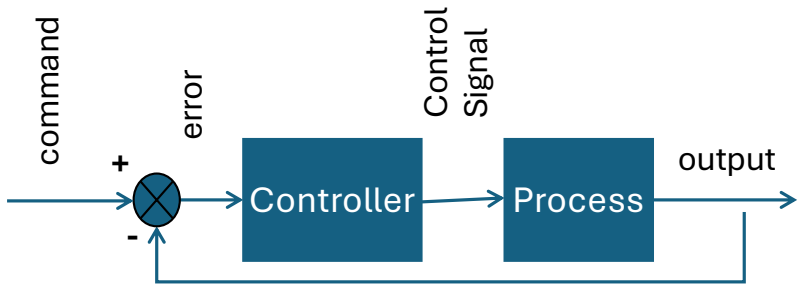
- *If the desired and current states are the same, then what?*

  Then there is nothing to do!

- *What if they are not?*

Measuring Error

- The controller first computes the difference between the current and desired states.
- The difference is called **error**.
- *What is the controller's job?*
  - To minimize the error at all times.



- Depending on the type of sensors, the error may be measured with different amounts of information.

Zero/non-zero Error

- The least information we can have about the error is
  whether it exists
  i.e., whether current state ≡ desired state

- This is called **zero/non-zero error**

- **Zero/non-zero error** provides very little information, yet useful control systems can be constructed with it. (E.g., some forms of reinforcement learning work this way)
- *What other information would be helpful?*

Error Magnitude & Direction

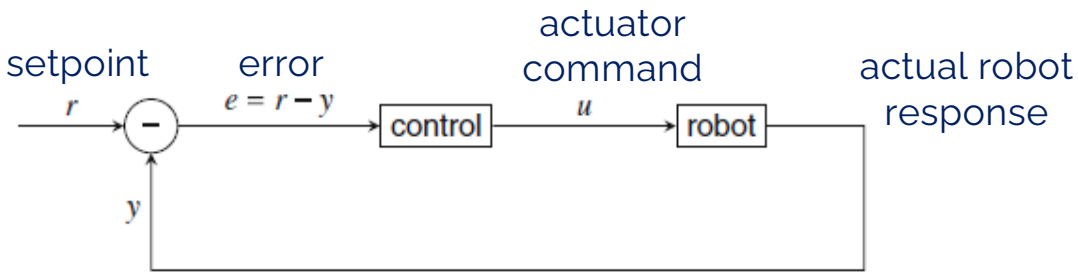
- Additional information about the error is its *magnitude*, i.e., the absolute difference (distance) between the current state and the desired state.

- The last part of the error information is its *direction*, i.e., whether the difference is positive or negative.

- Control is easiest if frequent **feedback** provides both magnitude and direction.

Closed loop (feedback) control

- Feedback controller:
  - output = FB(error)
  - error = goal state - measured state
  - controller attempts to minimize error
- Feedback control requires sensors:
  - Binary (at goal/not at goal)
  - Direction (less than/greater than)
  - Magnitude (very bad, bad, good)
- Control is easiest when direction and magnitude are available



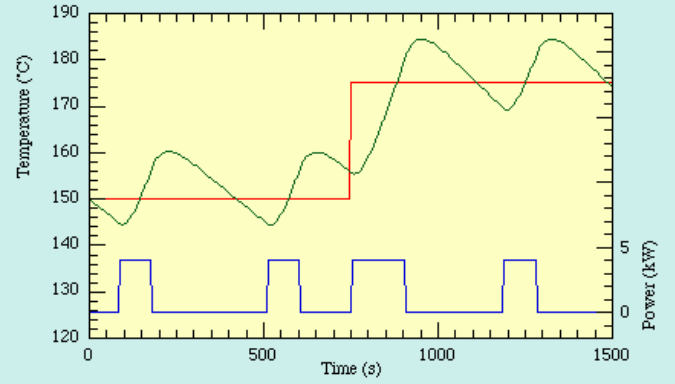
An Example

- Using feedback control to implement a wall-following behavior
  - What type of goal is this? Achievement or Maintenance?
  - What would the error be?
- What sensors could you use?
  - Would the sensor provide magnitude and direction of the error?
- How can we word the controller?
- What will this robot's behavior look like?

Simplest control: ON/off or bang/bang

- If *error* > 0, turn on actuators
- If error ≤ 0, turn off actuators

on/off control



AKA Bang-Bang or Hysteresis control

setpoint

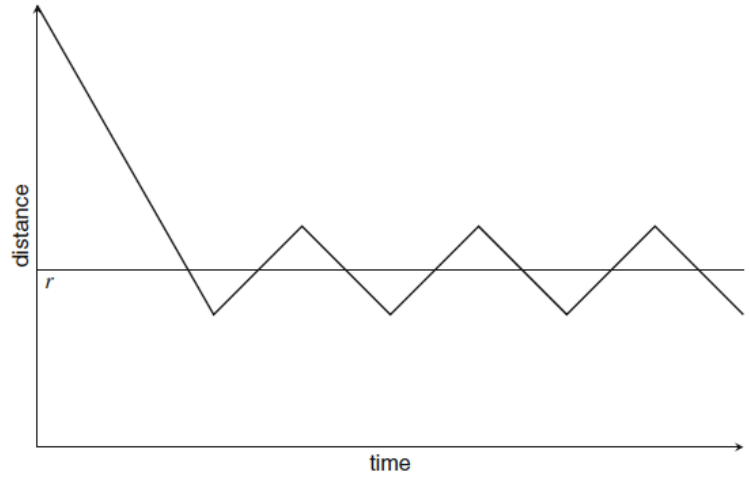
System Response

Control Output

Imagine the elevator…

Oscillation & the Set Point

- The behavior of a feedback system oscillates around the desired state
- E.g., the robot's movement will oscillate around the desired distance from the wall
- *How can we decrease oscillation?*



Control Theory

- *Control theory is the science that studies the behavior of control systems*

- Three main types of simple linear controllers:
  - P: proportional control
  - PD: proportional derivative control
  - PID: proportional integral derivative control

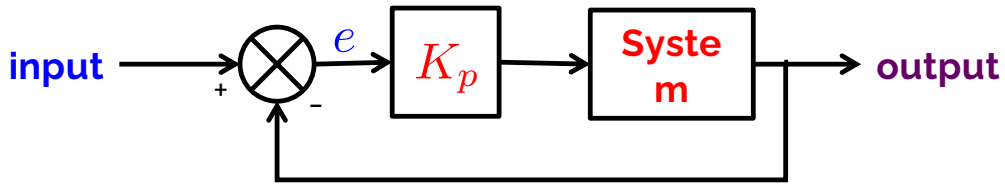
- All use direction and magnitude of error

Proportional Control

- Act in proportion to the error
- A proportional controller has an output *o* proportional to its input i:

$$o = K_p e$$

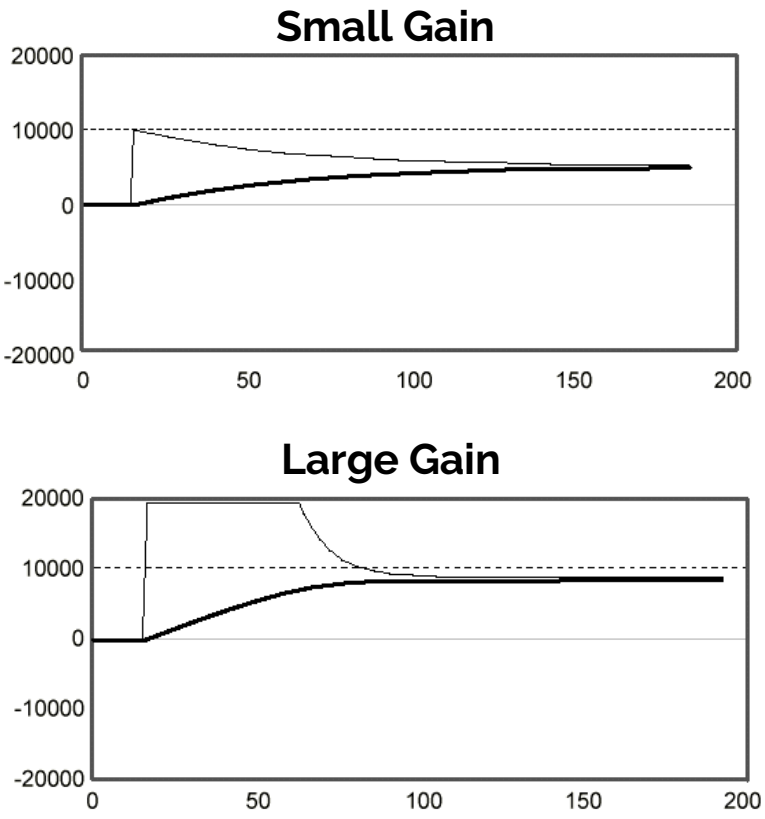
- K_p is a proportionality constant (gain)



What is a Gain?

- How do we decide how much to turn, or how fast to go? (i.e. the magnitude of the system's response)
- Those parameters (K_p) are called **gains**, and are very important in control
- Determining the right gains is difficult
- It can be done
  - analytically (mathematics)
  - empirically (trial and error)

Effect of Gains



Physics and Gains

- The physical properties of the system directly affect the gain values.
  - E.g. the velocity profile of a motor (how fast it can accelerate and decelerate),
  - the backlash and friction in the gears,
  - the friction on the surface, in the air, etc.

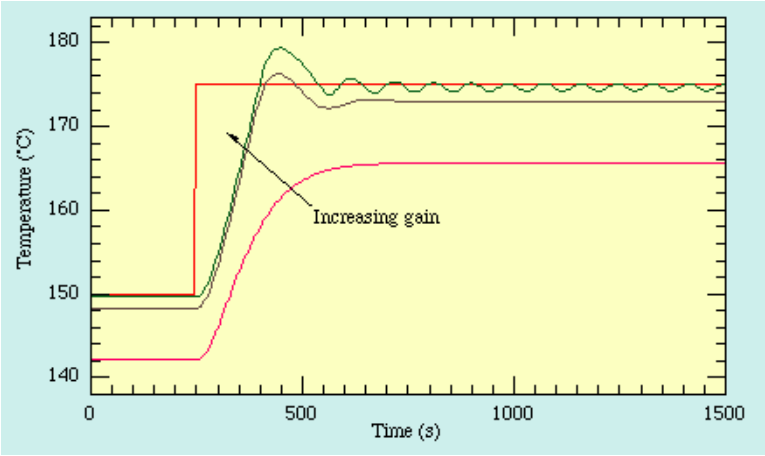
- All of these influence what the system actually does in response to a command

Setting Gains

- Analytical approaches
  - require that the system be well understood and characterized mathematically
- Trial and error (ad hoc, system-specific) approaches
  - require that the system be tested extensively
- Gains can also be tuned by the system itself, automatically, by trying different values at run-time

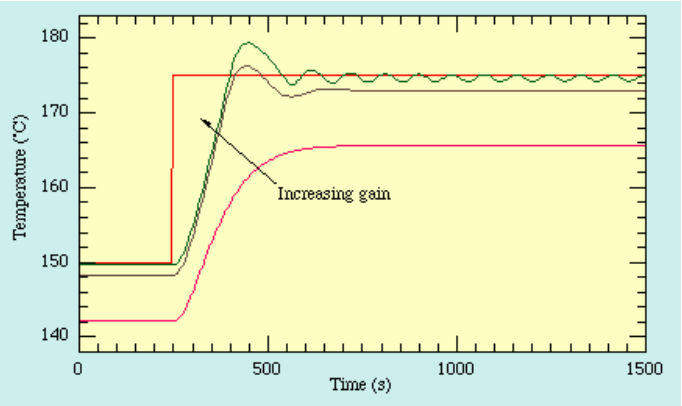
Gains and Oscillations

- Real systems have *momentum*
  - System takes some time to respond to commands
- Large gains can lead to *oscillations*:
  - System overshoots, recovers, overshoots again
- Very large gains can lead to *divergent* oscillations:
  - Overshoot gets larger and larger over time



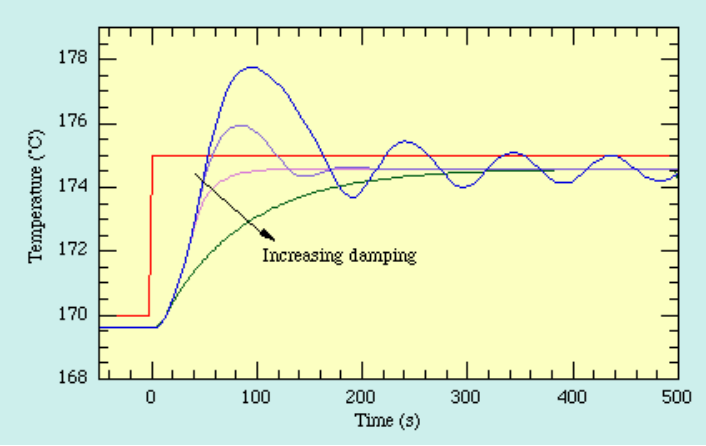
Control Near the Set Point

- Setting gains is difficult, and simply increasing the proportional gain does not remove oscillations.
- While at low values this may work, as the gain increases, the oscillations increase as well.
- The problem has to do with the distance from the set point…
- *How can we solve this?*



Damping

- **Damping** is the process of systematically decreasing oscillations
- *What do you think it means to be properly damped?*
- A system is *properly damped* if it does not oscillate with increasing magnitude, i.e., if its oscillations are either avoided, or decrease to the desired set point within a reasonable time period

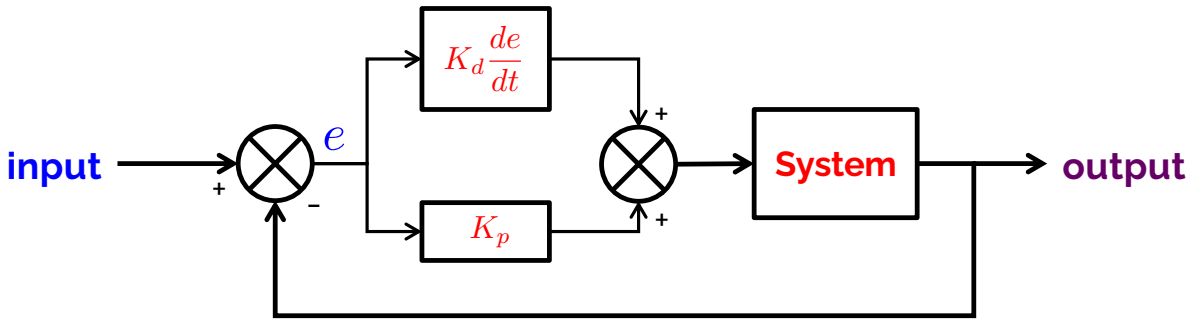


Derivative Control

- Act in proportion to the *rate of change* of the error
  - Allows us to treat the system differently when close to the goal
- A derivative controller has an output *o* proportional to the *derivative* of its input:

$$o = K_d \frac{de}{dt}$$

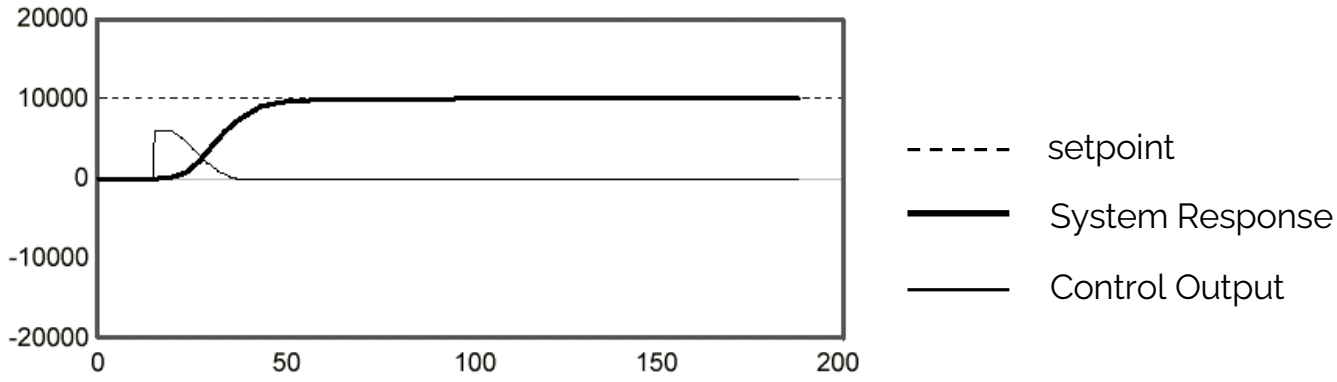
- K_d is a proportionality constant



PD Control

- PD control combines P and D control:
- P component minimizes error
- D component provides damping
- Gains K_p and K_d must be tuned together

$$o = K_p e + K_d \frac{de}{dt}$$



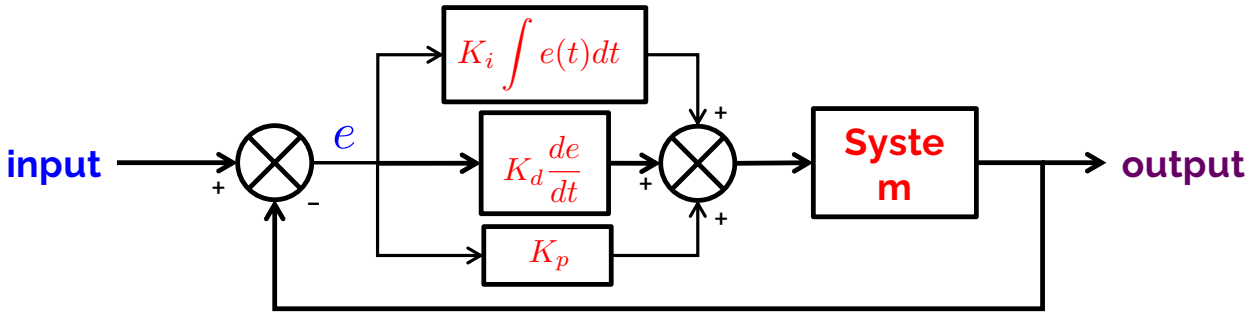
Integral Control

Remember the elevator error?

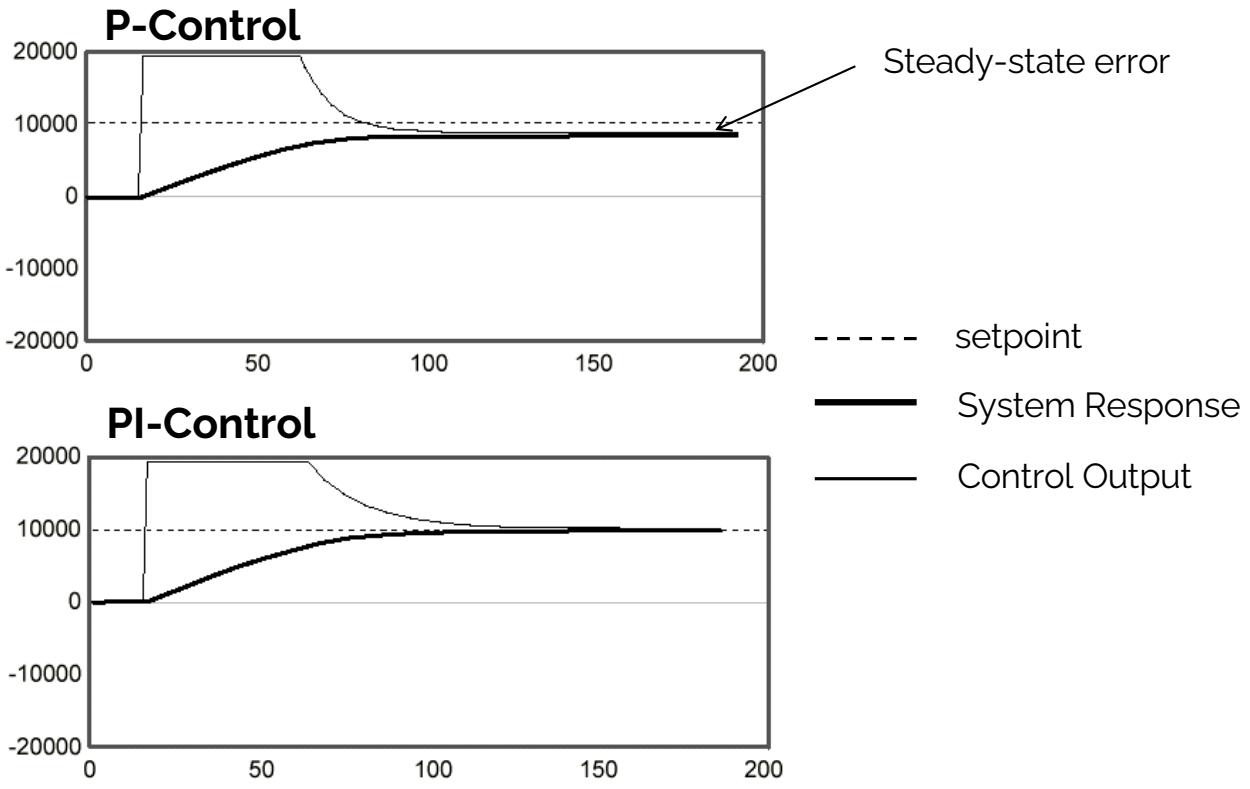
- Act in proportion to the *accumulated* error
- Output proportional to the integral of its input:

$$o = K_i \int e(t)dt$$

- K_i is a proportionality constant
- Integral control is useful for eliminating steady-state errors



Integral Control



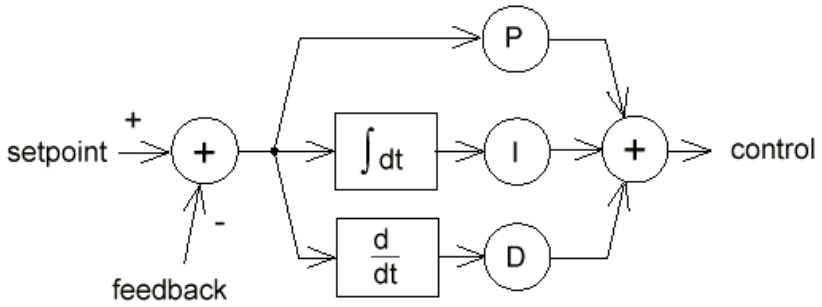
PID control

- PID control combines P, I and D control:

$$o = K_p e + K_i \int e(t)dt + K_d \frac{de}{dt}$$

  - P component minimizes instantaneous error
  - I component minimizes cumulative error
  - D component provides damping

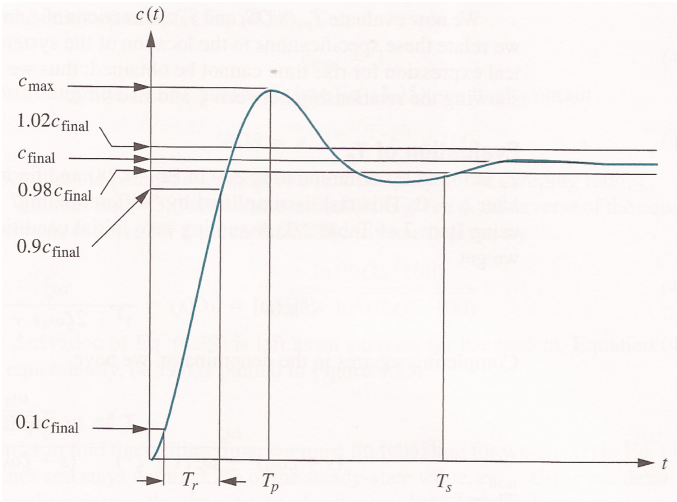
- Gains must be tuned together



System Response Terminology

- **Rise time**
  - Time for the waveform to go from 0.1 to 0.9 of its final value
  - Measures how quickly the controller responds
- **Percent Overshoot**
  - Amount the waveform overshoots the final value, at the peak time
  - Measures how big the oscillations are
- **Settling time**
  - Time for the response to reach, and stay within, 2% of its final value
  - Measures how fast the controller reaches steady-state

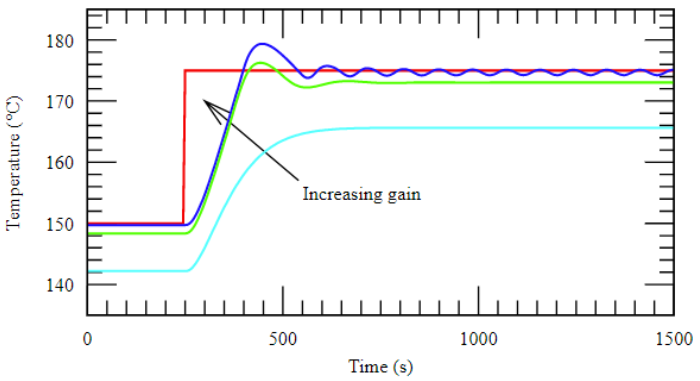
System Response



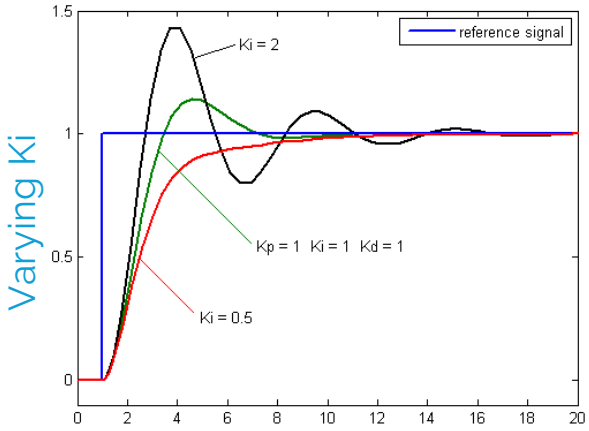
PID control summary

	Rise Time	Overshoot	Settling time
K_p	Decrease	Increase	
K_i	Decrease	Increase	Increase
K_d		Decrease	Decrease

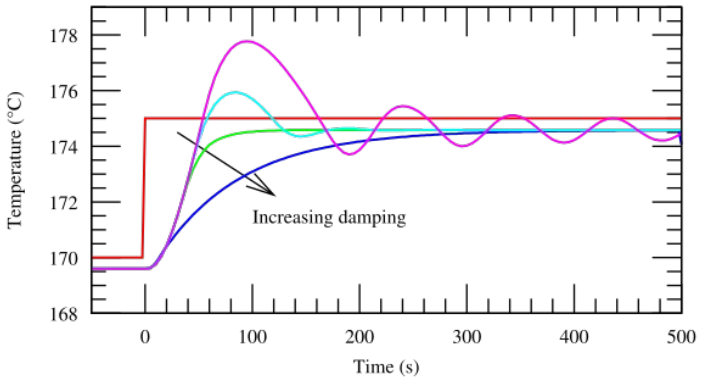
Varying Kp



Varying Ki



Varying Kd



Control is one component…

- Getting robots to do what you want requires many components!
- Feedback Control plays a part at the lower level: turning wheels, moving actuators
- Higher-level goals (coordination, interaction, collaboration) require other approaches
  - These approaches can better represent and handle those challenges
- Control Architectures are the AI portion that determines how the robot makes decisions


That’s it for today!
Questions?