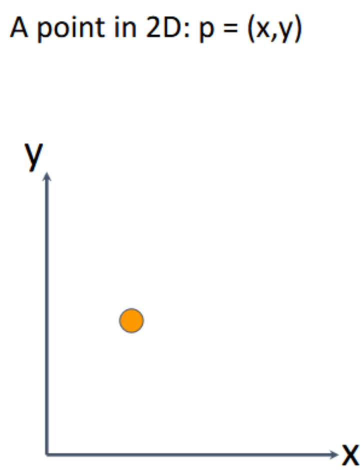
CS 188 Robotics Week 2

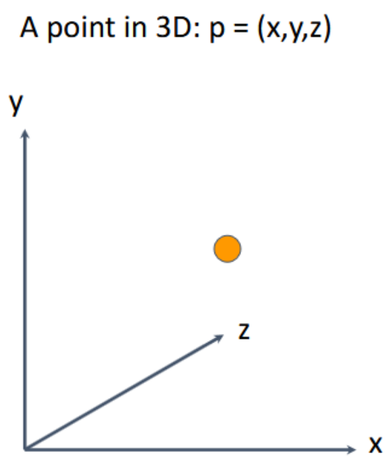
Aidan Jan

April 10, 2025

Rigid Body Motions

Representing Position



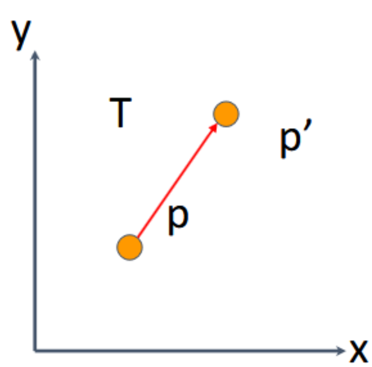


2D Transformation: Translation

Translate the point p to p' with $T = (\mathrm{d}x, \mathrm{d}y)$:

$$p' = T + p$$

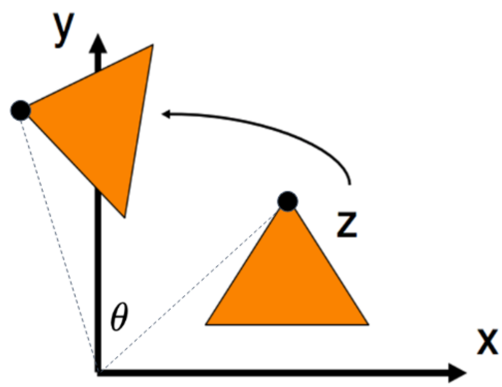
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Transformation: Rotation

$$p' = R \cdot p$$

Here we are doing a counter-clockwise rotation



The triangle here helps us visualize the rotation. However, we are still considering one 2D point p.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Combining Rotation and Transformation

$$p' = R \cdot p + T$$

In general, a matrix multiplication lets us linearly combine components of a vector.

- It is sufficient for representing rotation, but we can't add a constant :(

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Homogeneous Coordinates

- The solution? Stick a "1" at the end of every vector.
- Now, we can do rotation AND translation
- This is called "homogeneous coordinates"

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$$

- Our old way of representing point is called "Cartesian coordinate system"

Cartesian and Homogeneous Coordinate

- A point in cartesian coordinate $\langle x, y\rangle$ can be represented by $\langle sx, sy, s\rangle$ in homogeneous coordinate, where s is any scalar number.
  - For example, $\langle 2, 3\rangle$ in cartesian coordinate can be represented as $\langle 2, 3, 1\rangle$ or $\langle 4, 6, 2\rangle$, or $\langle 1, 1.5, 0.5\rangle$, etc. in homogeneous coordinates
  - A point in homogeneous coordinate $\langle x, y, z\rangle$ can be converted to cartesian coordinates by dividing the last element $\langle x/z, y/z\rangle$
  - Similarly for higher dimensions

Transformation Matrices

Representing rotation and translation homogeneous coordinates

- 2D Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 2D Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Now we can represent both the rotation and translation operation with one transformation matrix.

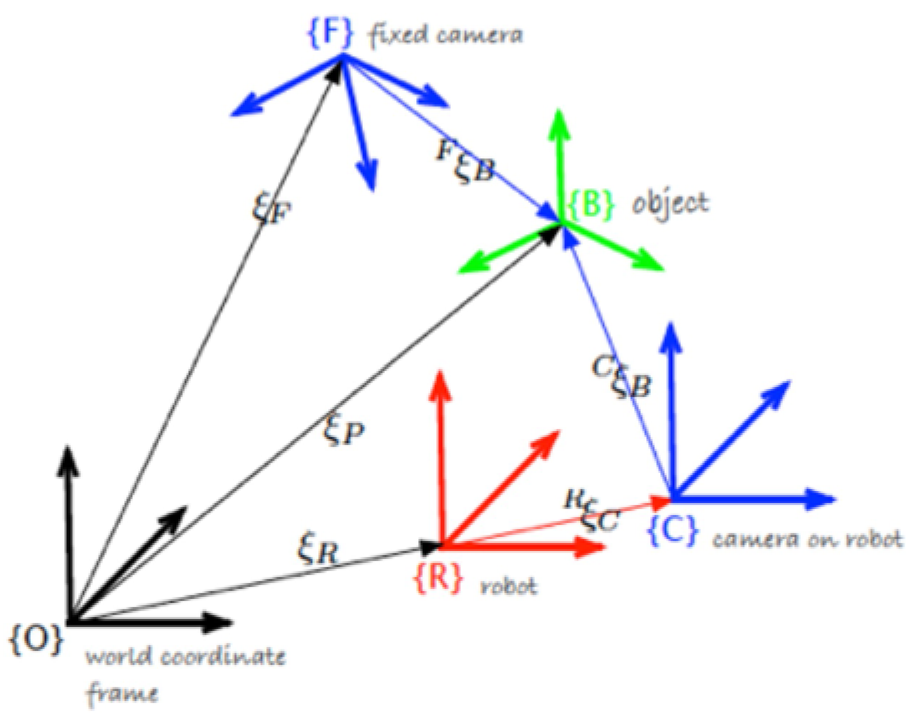
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Note: Following the matrix multiplication rule, a transformation matrix always apply rotation first, then translation.

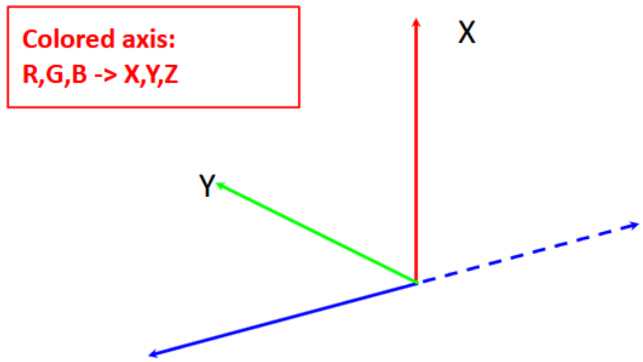
- Matrix multiplication is *not* commutative.

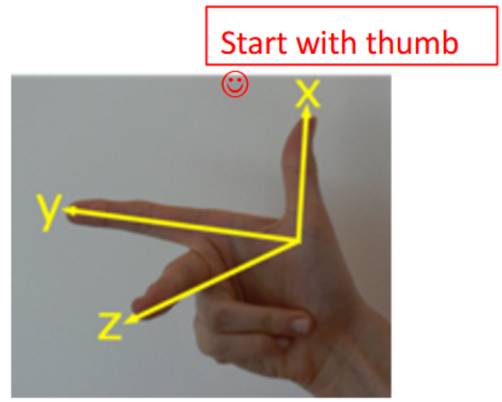
3D Transformation

Our examples so far were all in 2D, but we often want a 3D representation



Right Hand Rule





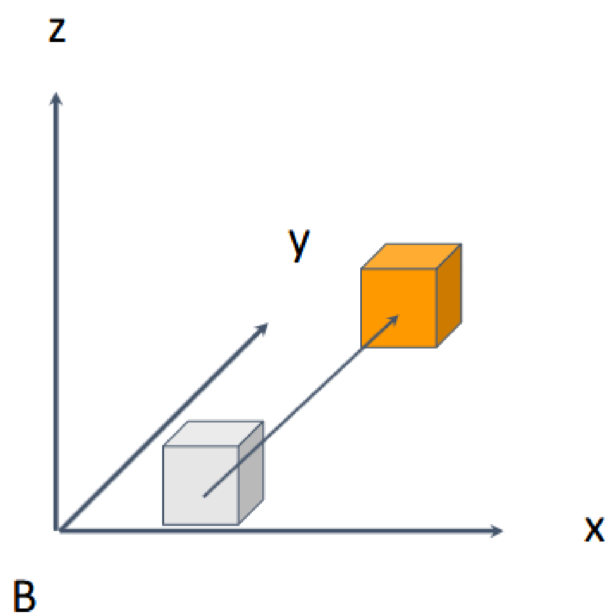
Most of robotics system's coordinate system follows the right hand rule

- Not always true (e.g., in some graphics and physics engine directX Unity)
- Therefore, be careful!

3D Transformation: Translation

A 3D point (x, y, z), translation by t_x, t_y, t_z:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



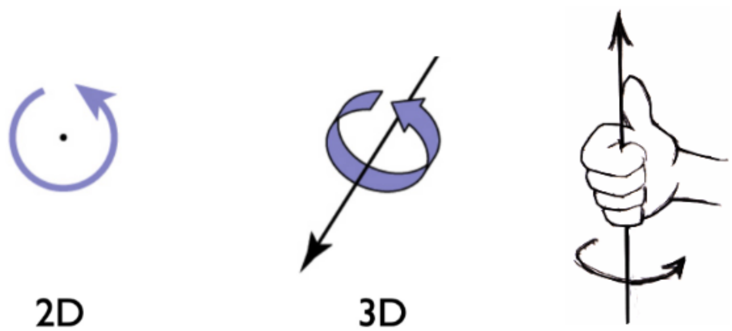
3D Transformation: Rotation

- A rotation in 2D is around a point
- A rotation in 3D is around an <u>axis</u> (a line with direction)

– rotation direction also follows right hand rule (thumb points to the axis direction, other fingers points towards the **positive** rotation direction)

– It is a 3D space, not just 1D

– most common choices for rotation axes are the x, y, z-axes (Euler angle representation)



3D Rotation Matrices

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

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

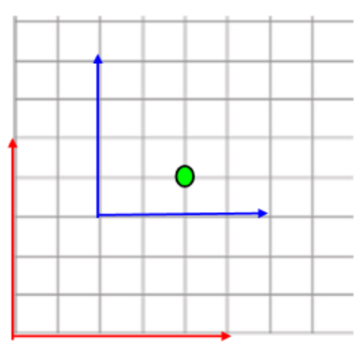
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reference Frames (Coordinate System)

- Up to now we have look at transformation in a single reference frame. However, in a complex robotic system we often need to define many reference frames.

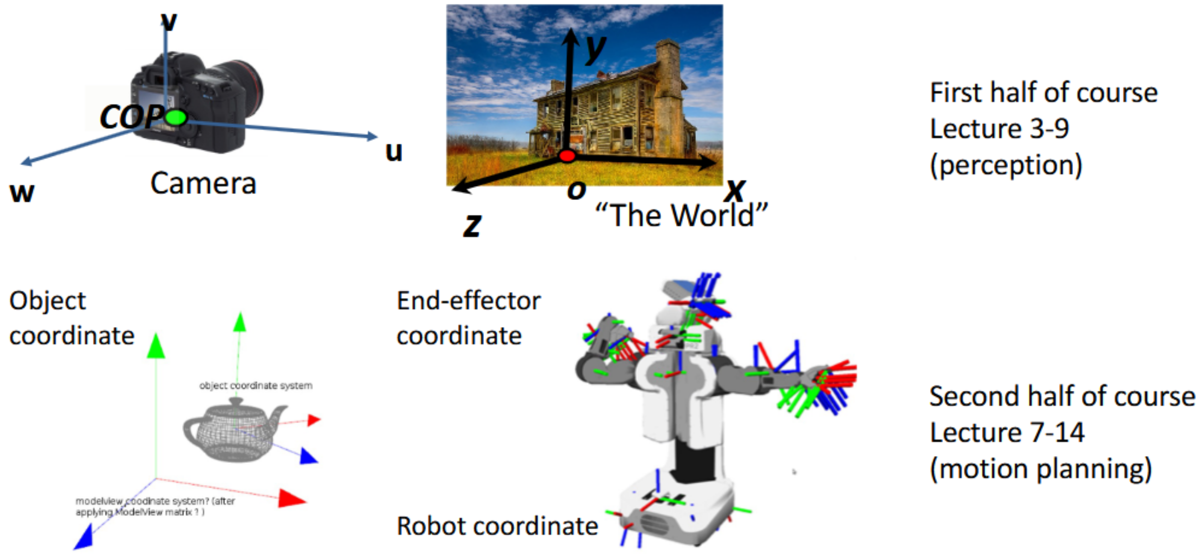
- The same 3D point might have different coordinate if we use different reference frames, next we will learn how to transform between different reference frames.

Example: green dot's coordinate is $(2, 1)$ in blue reference frame, but its coordinate is $(4, 4)$ in red reference frame.



Changing coordinate frame is like translating between two different languages that describes the same thing.

Examples:



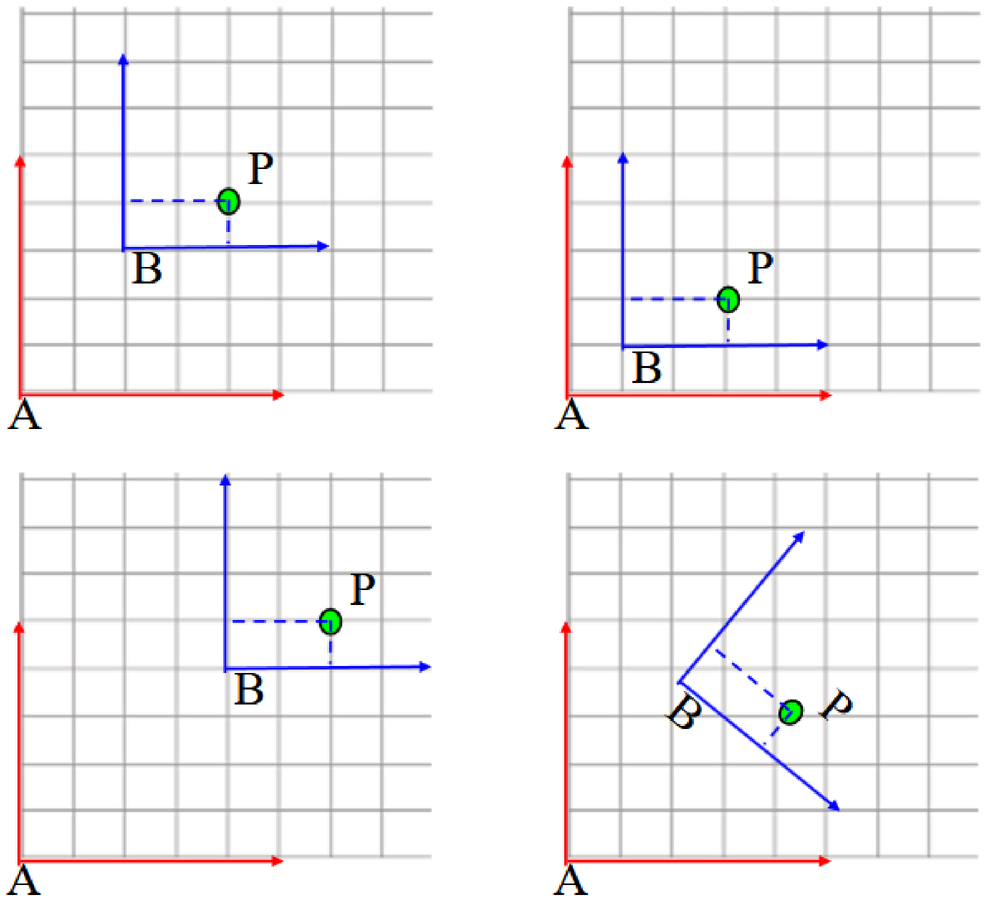
Changing Reference Frames

- We define two coordinate frames A and B
- A Point P:
  - P's coordinate in Frame A is ${}^AP = (4, 4)$
  - P's coordinate in Frame B is ${}^BP = (2, 1)$
- Transformations between reference frames we will use the notation ${}^AT_B$ (FROM frame is in the bottom right and the TO frame is in the top left.)
- To transform ${}^BP$'s reference frame from B to A, we just need to apply ${}^AT_B$ to ${}^BP$.

$$ {}^AP = {}^AT_B \cdot {}^BP $$

How do we compute ${}^AT_B$?

- Suppose the point P is rigidly attached to reference Frame B.
- No matter where the reference B, point P is its coordinates with respect to Frame B is always given by ${}^BP = (2, 1)$.

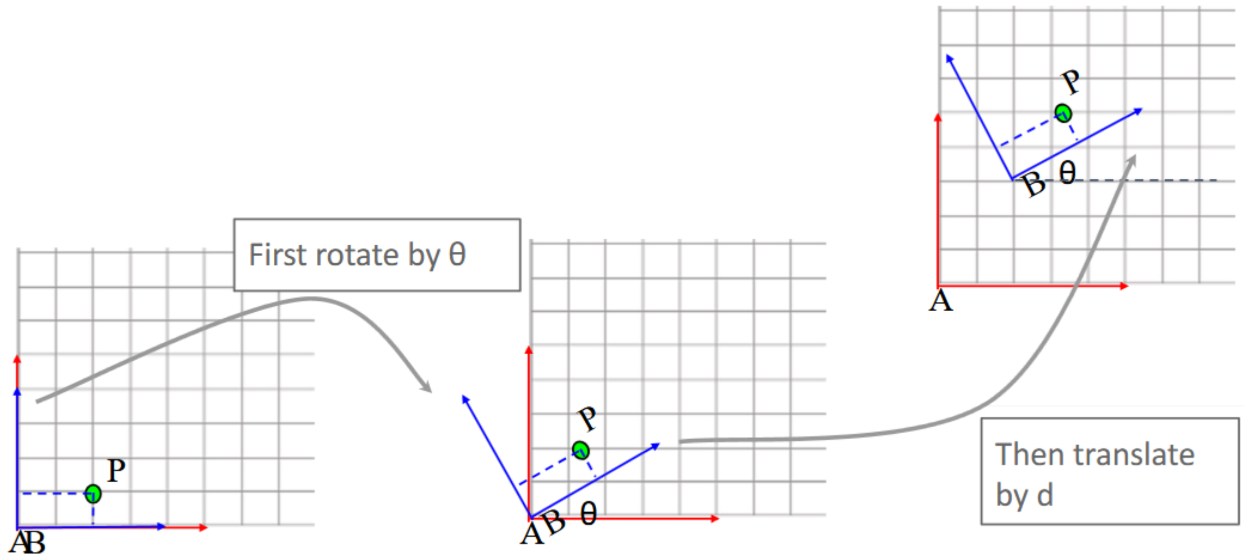


First, let's make Frame B identical to Frame A. Now, ${}^{A}P = {}^{B}P = (2, 1)$. Now, simply <u>translate</u> Frame B together with $d = (2, 3)$, we will get the ${}^{A}P = {}^{B}P + d$.
Therefore in this case,

$$
{}^{A}T_{B} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}
$$

(There is no rotation in this case, only translation)

- If there is a rotation, first rotate the frame so it is aligned with the target, then do a translation.

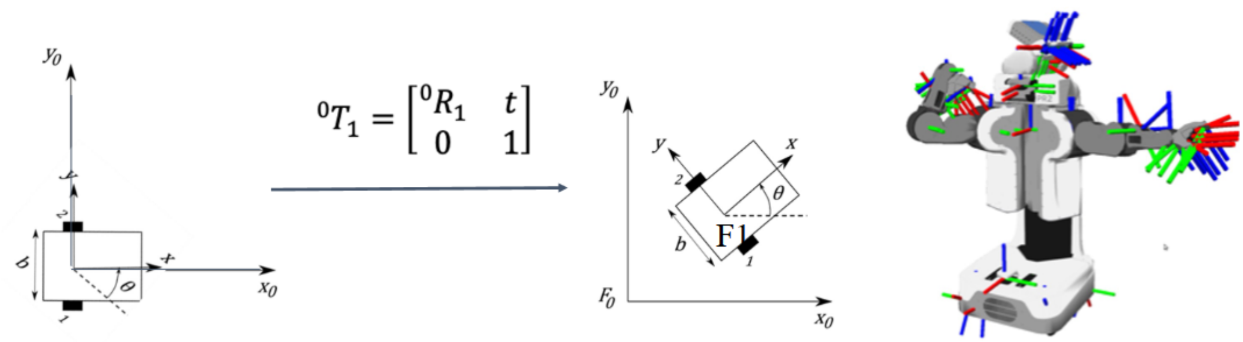


If we combine this rotation and translation into one transformation matrix, we get:

$$T = \begin{bmatrix} R_\theta & d \\ 0_n & 1 \end{bmatrix}$$

This is the transformation ${}^AT_B$ that change the coordinate frame from B to A.

- However, geometrically it describes the motion from Frame A to B.
- ${}^AT_B$ also describes Frame B's "pose" in Frame A, where the rotation component R describes the B's orientation in Frame A, and the translation represents B's position in Frame A.

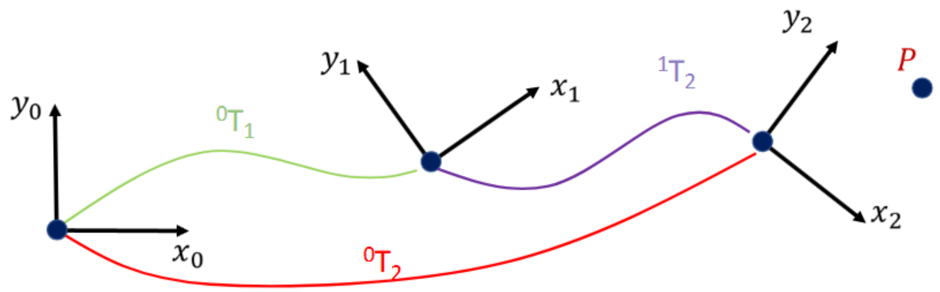


Change of Basis Summary

What is ${}^AT_B$?

- ${}^AT_B$ is a rigid transformation matrix (3x3 matrix in 2D, 4x4 in 3D)
- ${}^AT_B$ represents the transform that **change the coordinate frame from B to A:** ${}^AP = {}^AT_B\, {}^BP$
- ${}^AT_B$ geometrically describes the motion from Frame A to B.
- ${}^AT_B$ is also the pose of coordinate frame (B) in the coordinate frame (A); that describes the position and orientation of Frame B in Frame A.

Composing Transformation



From our previous results, we know:

${}^0P = {}^0T_1 {}^1P$

${}^1P = {}^1T_2 {}^2P$

$\rightarrow$ ${}^0P = {}^0T_1 \, {}^1T_2 \, {}^2P$

But we also know: ${}^0P = {}^0T_2 \, {}^2P$

$\rightarrow$ ${}^0T_2 = {}^0T_1 \, {}^1T_2$

This is the composition law for homogeneous transformations.

Chained 3D Rotation

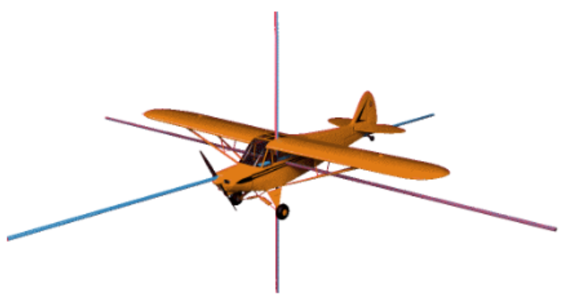
We can chain a sequence of Euler angle rotations (multiple sequence of rotation matrix) to get a general 3D rotation.

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha\cos\alpha & 0 & \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

There are a few things to note when writing down the sequence of rotation:

1. Rotation matrix is non-commutative - order matters!
2. Be aware of which sequence convention you are using when describing the 2nd and 3rd rotations: **extrinsic** rotation (fixed global frame), or **intrinsic** rotation? (last rotated coordinate system) - they are different.

Extrinsic vs. Intrinsic Rotation



Extrinsic: all rotation are described with respect to fixed global frame (red frame)

$$R = R_z(90°) \cdot R_y(45°) \cdot R_x(180°)$$

First, rotate about the global *x*-axis, 180
then, rotate about the global *y*-axis, 45
finally rotate about the global *z*-axis, 90

Intrinsic: a rotation is described to the last rotated coordinate system (blue, airplane's body frame)

$$R = R_z(90°) \cdot R_{y'}(45°) \cdot R_{x''}(180°)$$

1) rotate about the global *z*-axis, 90
2) rotate about the new *y'*-axis, 45
3) rotate about the new *x''*-axis, 180

The final rotation R is the same. However, the order of describing rotation sequence is opposite in each convention.

- (Use premultiply!)

Rotation Matrix

Rotation matrix has a number of highly useful properties:

- R is an orthonormal matrix: Its columns are orthogonal unit vectors. ($R^{-1} = R^T$)
  - This does not apply to general transformation matrices.
- determinant of the matrix $|R| = 1$
- The length of the vector is unchanged after transformation

Other 3D Rotation Representations

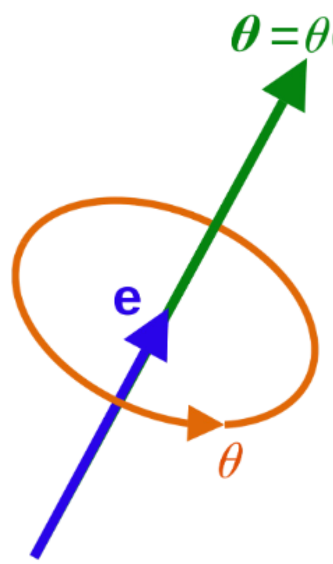
There are many ways to specify rotation

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

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.

- 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.



Quaternions

Uses a unit four-dimensional vector (x, y, z, w) to represent rotation.

- If the rotation is (v_1, v_2, v_3, θ) in angle-axis representation, it can be written in quaternion as:

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

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

- the above is a 4-dimensional vector on a 4D sphere.

Quaternions are a very popular parameterization 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).

For example:

- $(0, 0, 0, 1)$ is the identity quaternion.
- $(1, 0, 0, 0)$ rotates along x-axis by π. (Since $w = 0$, therefore $\theta = \pi$).

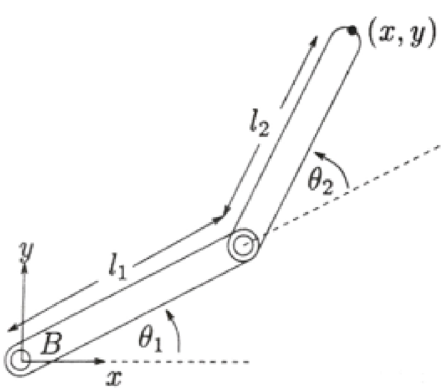
To inverse a quaternion:

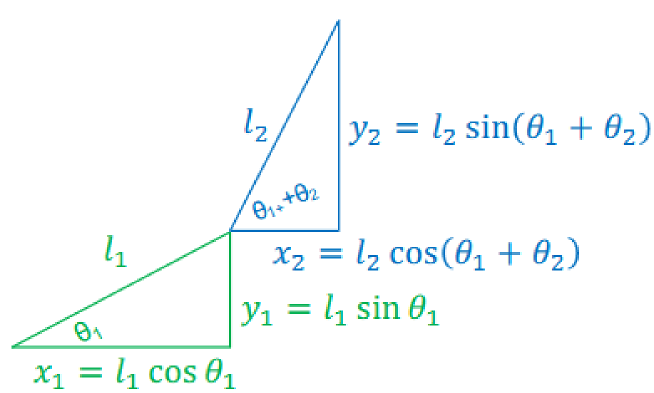
- keep the rotation axis, rotate backward
- Inverse of (x, y, z, w) is $(x, y, z, -w)$
- (x, y, z, w) is equivalent to $(-x, -y, -z, -w)$

Forward Kinematics

Forward Kinematics of 2-link Manipulator

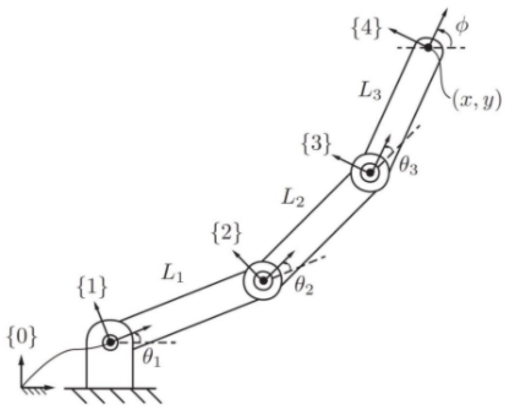
Given joint angles, calculate position of end-effector





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

Forward Kinematics of RRR open-chain



Forward kinematics of a 3R planar open chain.

- General cases
  - Attaching frames to links
  - Using homogeneous transformations

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

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

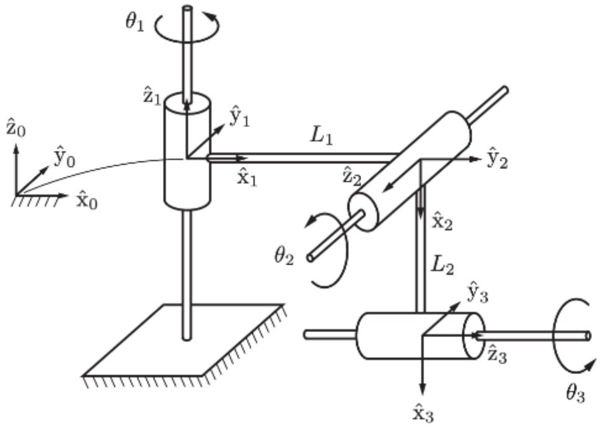
$$T_{23} = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & L_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_{34} = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Denavit-Hartenberg (DH) parameters

$$T_{0n}(\theta_1, \ldots, \theta_n) = T_{01}(\theta_1)T_{12}(\theta_2)\cdots T_{n-1,n}(\theta_n)T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 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 $\hat{z}_{i-1}$, 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-0i 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.

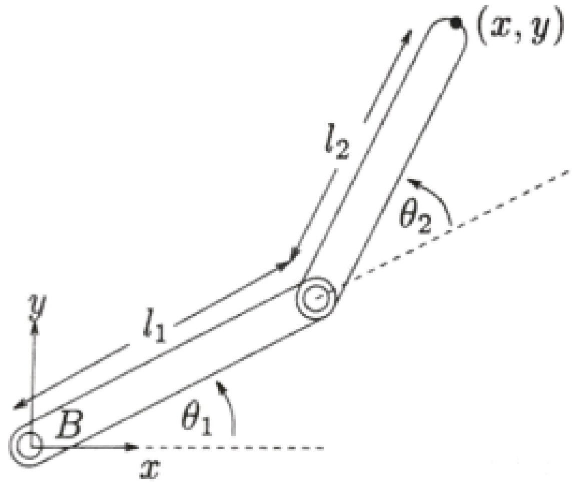


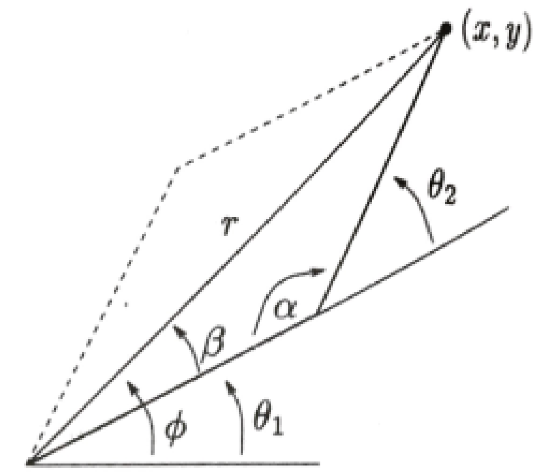
Parameter	Symbol	Meaning
Link length	a_i	Distance from Z_{i-1} to Z_i along X_i
Link twist	α_i	Angle from Z_{i-1} to Z_i around X_i
Link offset	d_i	Distance from X_{i-1} to X_i along Z_{i-1}
Joint angle	ϕ_i	Angle from X_{i-1} to X_i around Z_i

i	α_{i-1}	a_{i-1}	d_i	ϕ_i
1	0	0	0	θ_1
2	$90°$	L_1	0	$\theta_2 - 90°$
3	$-90°$	L_2	0	θ_3

Inverse Kinematics

Given the end-effector position, calculate joint angles





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

If $\alpha \neq 0$, there are two distinct values of θ_2 which give the appropriate radius - the *flip solution* is shown dashed above.

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

Solve for ϕ and use this to get θ_1 for **both** possible θ_2 values

- Inverse kinematics for joints > 2 is generally not solvable (no closed-form solution)
- More than one solution (redundancy)
- A hard (and well-studied problem)

"Solving" Inverse Kinematics

Inverse kinematics for joints > 2 is generally not solvable. In this case, what do we do?

1. 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.
2. 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
3. Learning-Based IK
   - Neural networks or reinforcement learning.
   - Especially helpful in high-DoF or redundant robots (like humanoids or octopuses).

Dynamics

Given joint velocities, find the 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

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

Aside: Jacobian Matrix

For *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:

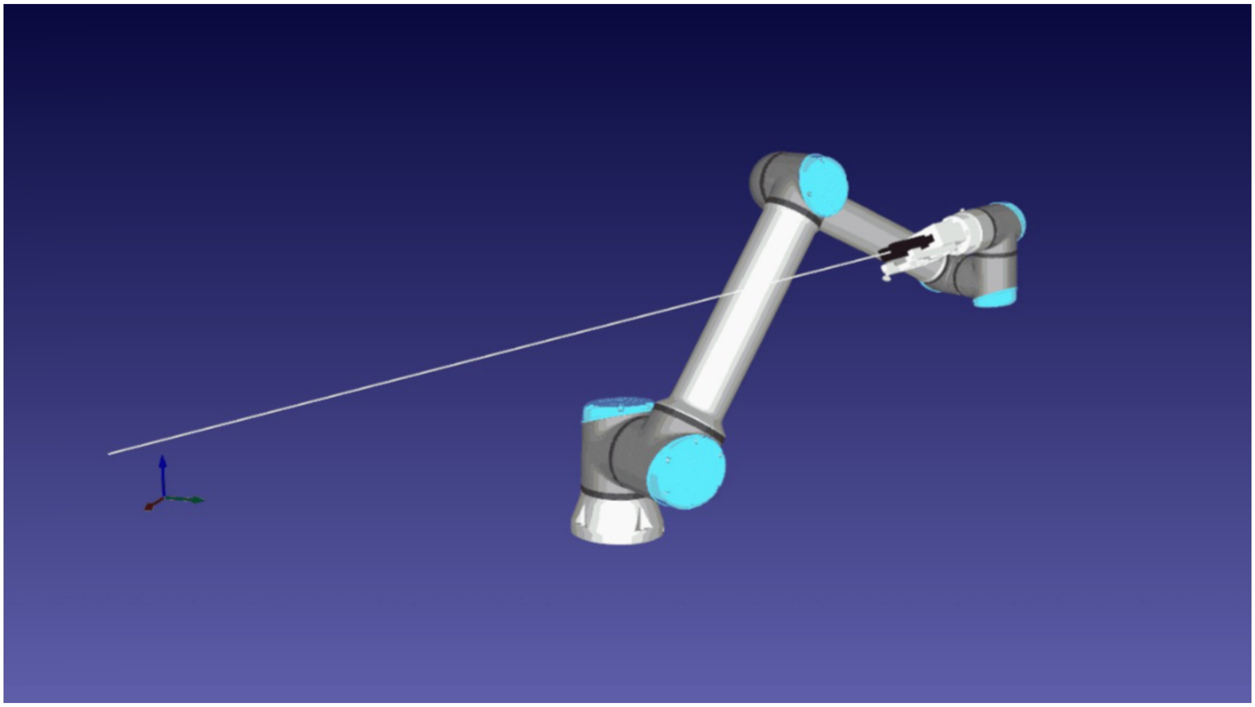
$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

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
- Manipulator has singularity when determinant of Jacobian is zero
- Difficult to control around singularity

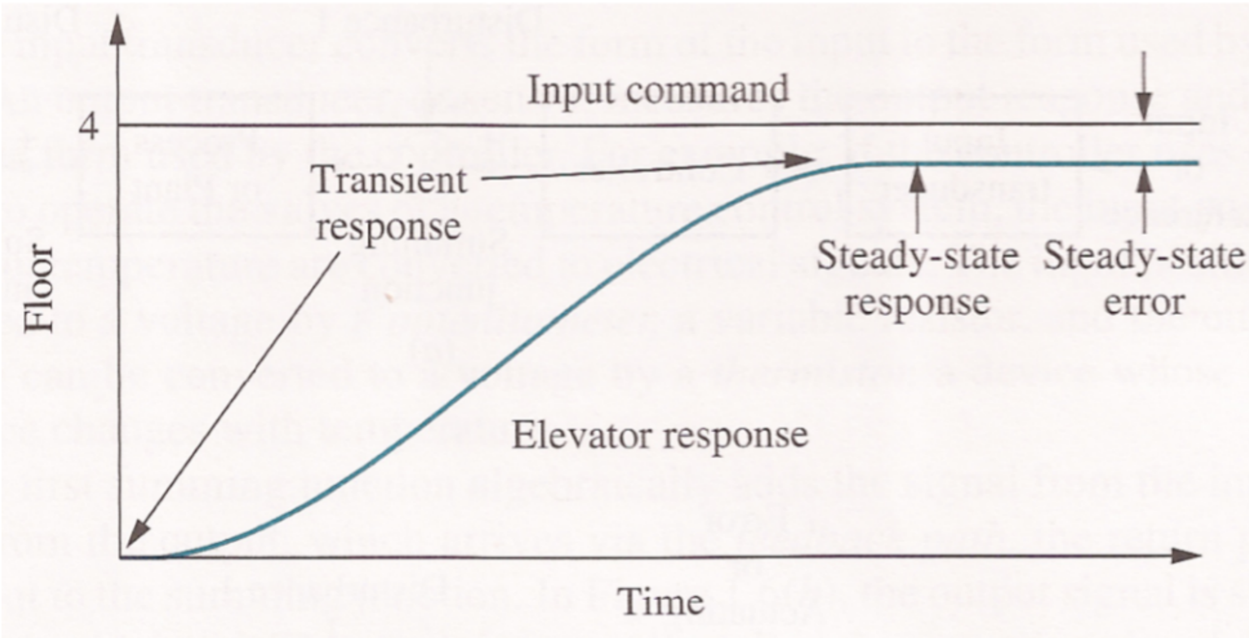


Control

- Many tasks in robotics are defined by *achievement goals*
  - Go to the end of the maze
  - Push that box over here
  - Typically AI (search, ML) algorithms
- Other tasks in robotics are defined by *maintenance goals*
  - Drive at 0.5 m/s
  - Balance on one leg
- 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



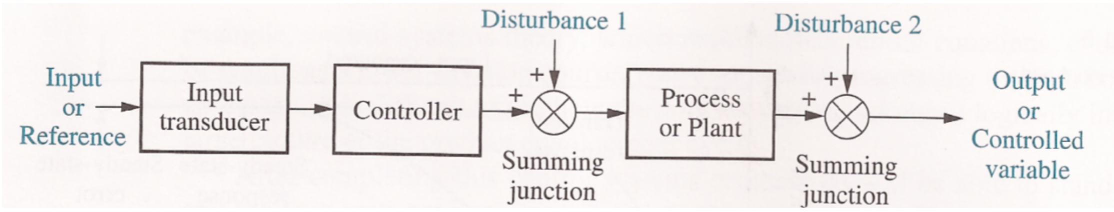
Output $\neq$ Input

- Transient Response:
  - Instantaneous change of input but gradual change of output
- State-state Error:
  - Accuracy of leveling
  - Steady state error is inherent!

Open Loop (feedforward) Control

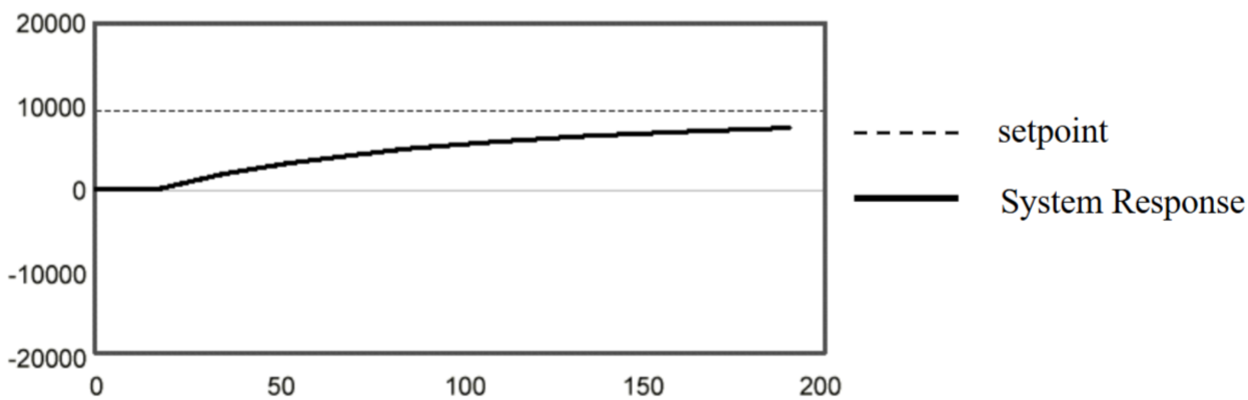
- Open loop controller:
  - output = FF(goal)
- E.g.: motor speed controller (linear):
  - Is applied voltage on motor: V
  - Is goal speed: s
  - Is gain term (from calibration): k
  - $V = ks$
- How do we know we've reached the goal?
  - Weakness:

  - ∗ Varying load on motor: *motor may not maintain goal speed*
- – More likely scenario in robotics:



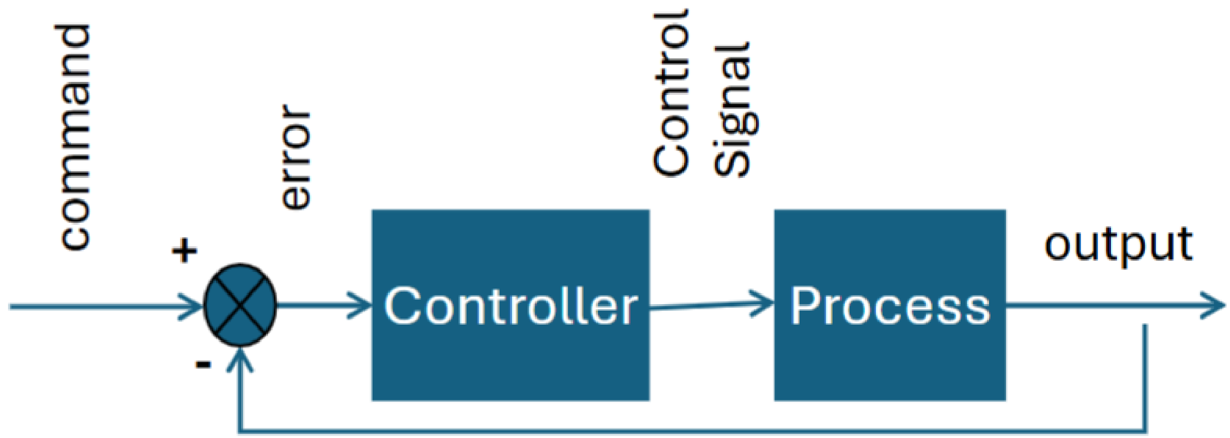
Uses of Open Loop Control

- 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 and withdrawal, etc.



Definition of Feedback Control

- Feedback control is a means of getting a system to achieve and 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

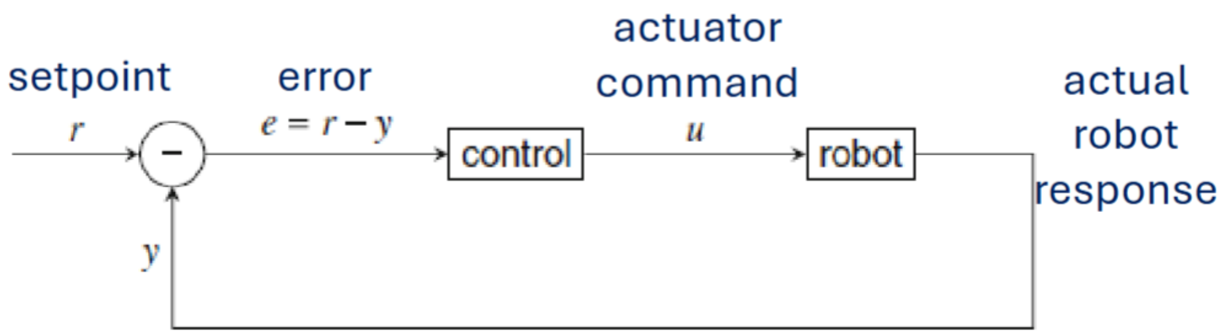
- Feedforward
  - Anticipative
  - Previous plan
  - Doesn't wait
- Feedback
  - Reactive / Responsive
  - Seeks to correct errors
  - Always too late!

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 the current state is the 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 and 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

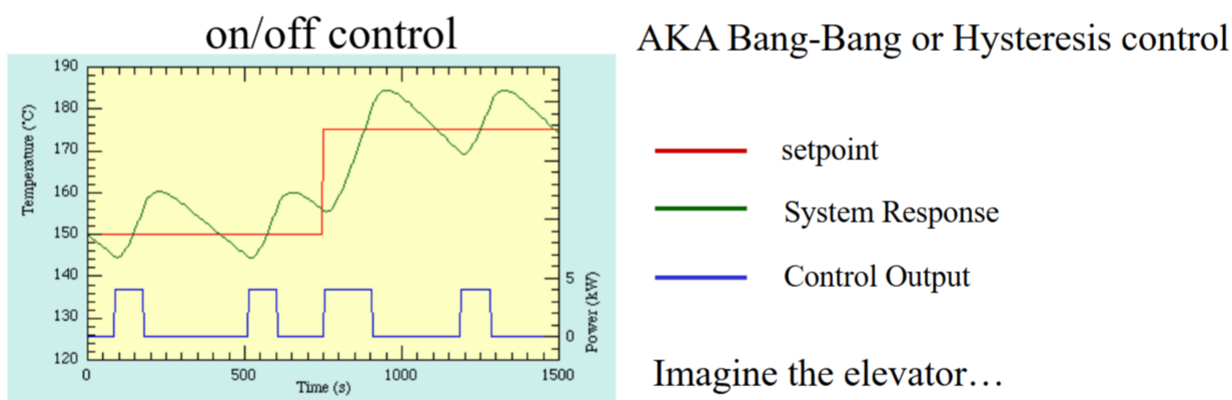
- Feedback controller:
  - output = FB(error)
  - error = goal state - measured state
  - controller attempts to minimize error
- Feedback control requires sensors:
  - Binary (at goal / not at goal)
  - Direction (less than / greater than)
  - Magnitude (very bad, bad, good)
- Control is easiest when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

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?

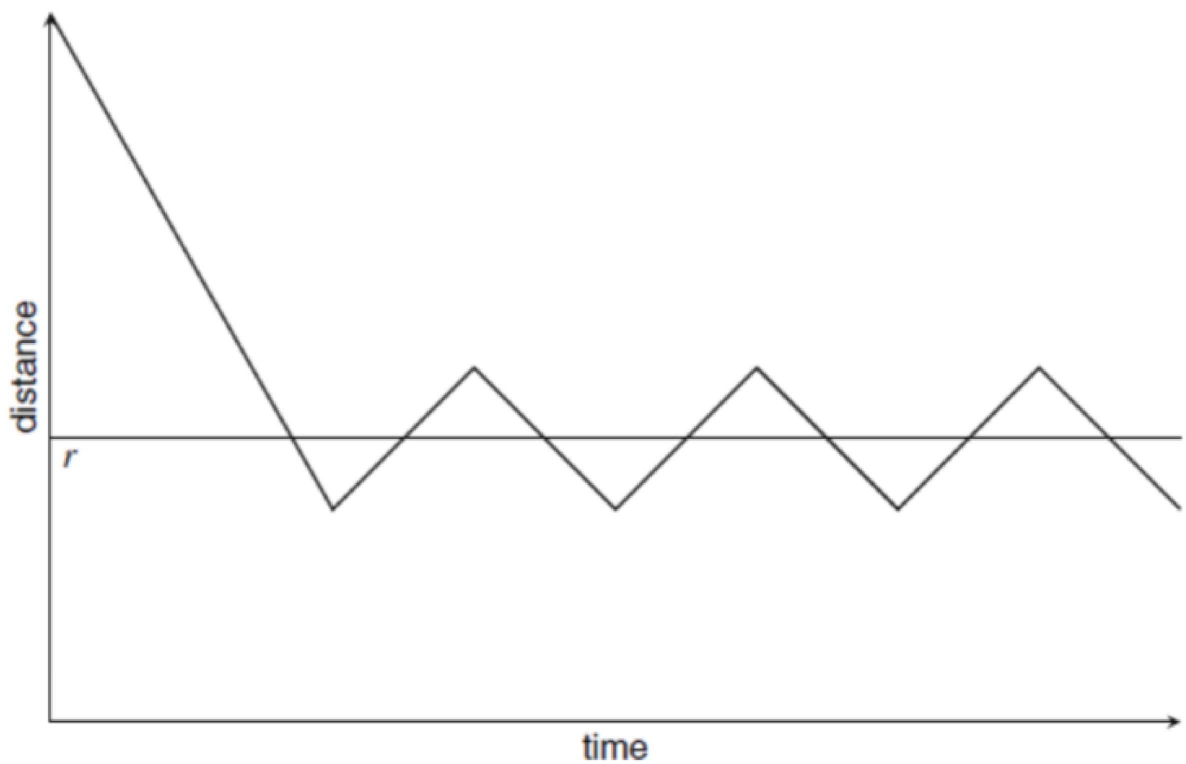
Simplest Control: ON/off or bang/bang

- If error $>$ 0, turn on actuators
- If error $\leq$ 0, turn off actuators



Oscillation and 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

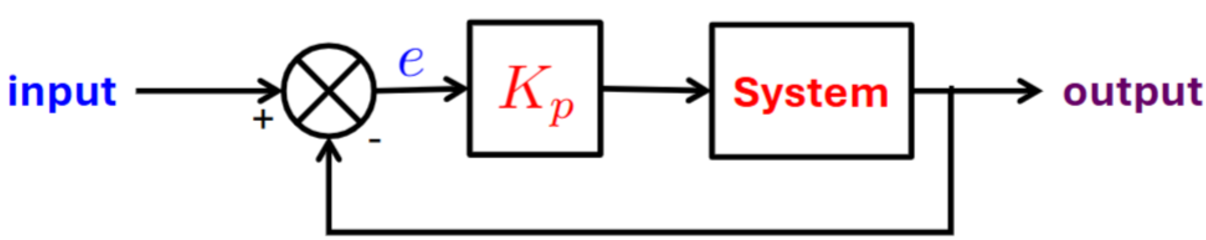
- *Control theory* is the science that studies the behavior of control systems
- Three main types of simple linear controllers:
  - P: proportional control
  - PD: proportional derivative control
  - PID: proportional integral derivative control
- All use direction and magnitude of error

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

[FILL 38, full]

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.

[FILL 41, 43, mix]

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 decreasin 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
- A derivative controller has an output o proportional to the derivative of its input:

$$o = K_d \frac{\mathrm{d}e}{\mathrm{d}t}$$

- K_d is a proportionality constant

[FILL 44]

PD Control

- PD control combined P and D control: $o = K_p e + K_d \frac{\mathrm{d}e}{\mathrm{d}t}$
- P component minimizes error
- D component provides damping
- Gains K_p and K_d must be tuned together

[FILL 45]

Integral Control

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

$$o = K_i \int e(t)\mathrm{d}t$$

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

[FILL 46]
[FILL 47]

PID Control

- PID control combined P, I, and D control:

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

  - P component minimizes instantaneous error
  - I component minimizes cumulative error
  - D component provides damping
- Gains must be tuned together [FILL 48]

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

[FILL 49]