COM SCI 188

Intro to Robotics

Lecture 3

Yuchen Cui

Winter 2026

Agenda

- Announcements
- Recap: Configuration Space (MR Chapter 2)

- C-Space Topology & Representation (MR Chapter 2)
- Rigid Body Motions (MR Chapter 3)
- Forward Kinematics (MR Chapter 4)
- Inverse Kinematics (MR Chapter 6)

Announcements

- Problem Set 1 due Friday
  - Covering materials in this lecture

- Coding Assignment 1 out, due next Friday
  - We will learn PID control next lecture
  - Discussion session will help you to get started

Announcements

- All PTEs have been assigned
- You are allowed to miss 1 discussion section without penalty

Announcements

- Final project
  - **Two Options**:
    - Default project in Robosuite (TBD)
    - Open project: propose your own!
  - **Open project requirement**:
    - Involves a robot (that senses, plans, and acts; sim or real)
    - Has verifiable success criteria

  - In teams of 2-4 (default project is 2)

Announcements

- Video supplements of Modern Robotics: https://modernrobotics.northwestern.edu/nu-gm-book-resource/foundations-of-robot-motion/

Recap

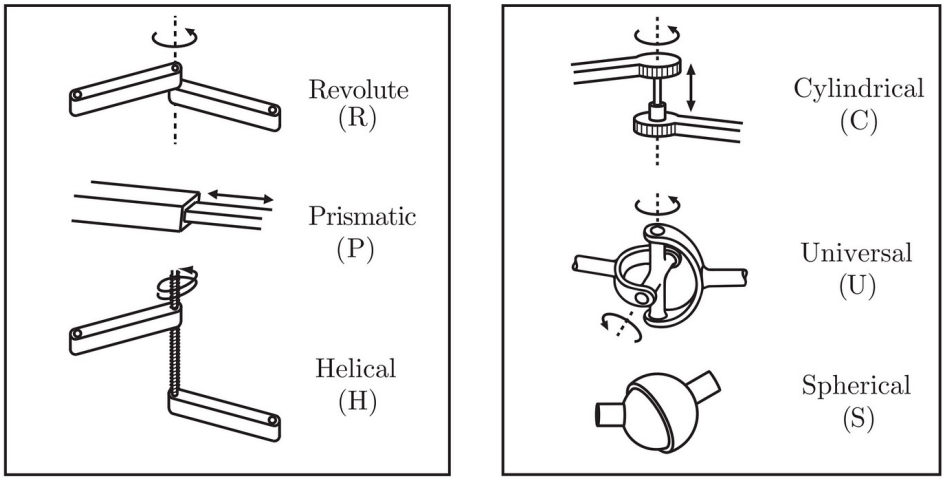
Definition of a robot: Sense, Plan, Act

Skeleton of a robot: Rigid Bodies + Joints

Muscles of a robot: Geared Motors

Recap

Common Joints



Joint type	dof f	Constraints c between two planar rigid bodies (2D robot)	Constraints c between two spatial rigid bodies (3D robot)
Revolute (R)	1	2	5
Prismatic (P)	1	2	5
Helical (H)	1	N/A	5
Cylindrical (C)	2	N/A	4
Universal (U)	2	N/A	4
Spherical (S)	3	N/A	3

MODERN ROBOTICS
Kevin M. Lynch and Frank C. Park May 3, 2017

Recap

Grübler's formula

$$\text{dof} = \underbrace{m(N-1)}_{\text{rigid body freedoms}} - \underbrace{\sum_{i=1}^{J} c_i}_{\text{joint constraints}}$$

$$= m(N-1) - \sum_{i=1}^{J}(m - f_i)$$

$$= m(N-1-J) + \sum_{i=1}^{J} f_i.$$

m=3 for planar mechanism (2D)
m=6 for a spatial mechanism (3D)

MODERN ROBOTICS
Kevin M. Lynch and Frank C. Park May 3, 2017

Recap

task space: 2D plane

Forward Kinematics

FK: $\boldsymbol{C} \rightarrow \boldsymbol{W}$

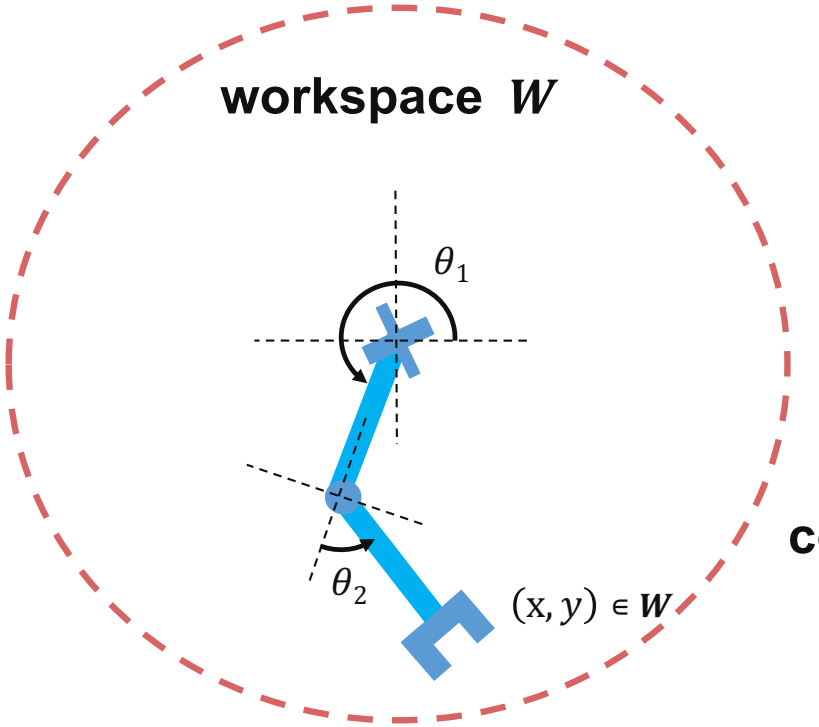
$$FK\big((\theta_1, \theta_2)\big) = (\mathrm{x}, y)$$

Inverse Kinematics

IK: $\boldsymbol{W} \rightarrow \boldsymbol{C}$

$$IK\big((\mathrm{x}, y)\big) = (\theta_1, \theta_2)$$

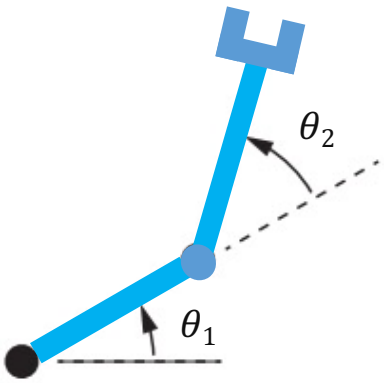
This is not a proper function.
Many configuration can lead to the same end-effector pose.



configuration space C

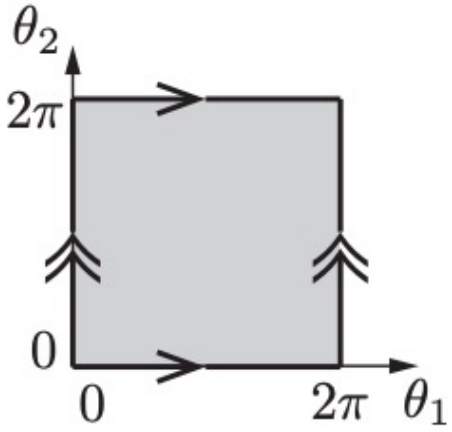
$(\theta_1, \theta_2) \in C$

C-Space Topology & Representation



2R robot arm

$T^2 = S^1 \times S^1$



$[0, 2\pi) \times [0, 2\pi)$

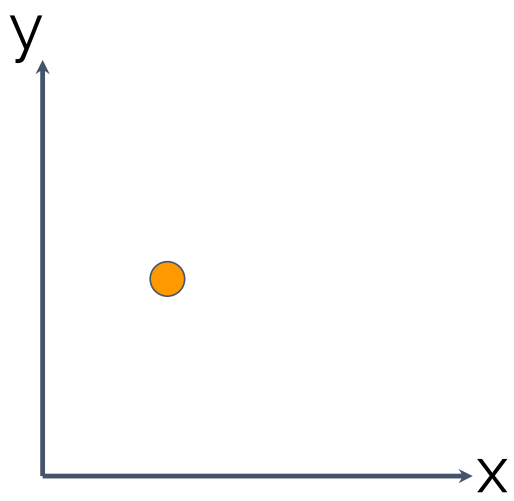
Rigid Body Motions

Reference Frames & Transformations

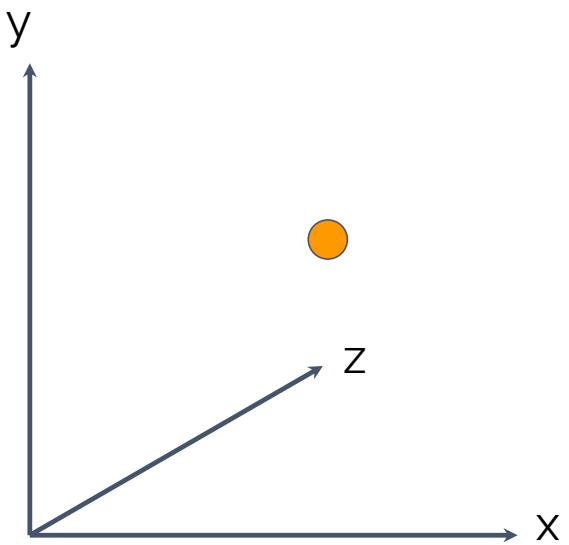
Slides adapted from Stanford CS 227A

Representing Position

A point in 2D: p = (x,y)



A point in 3D: p = (x,y,z)

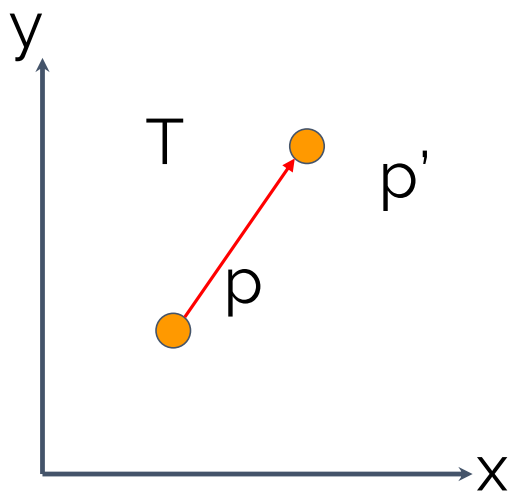


2D Transformation: Translation

Translate the point p to p' with T = (dx,dy)

$$p' = T + p$$

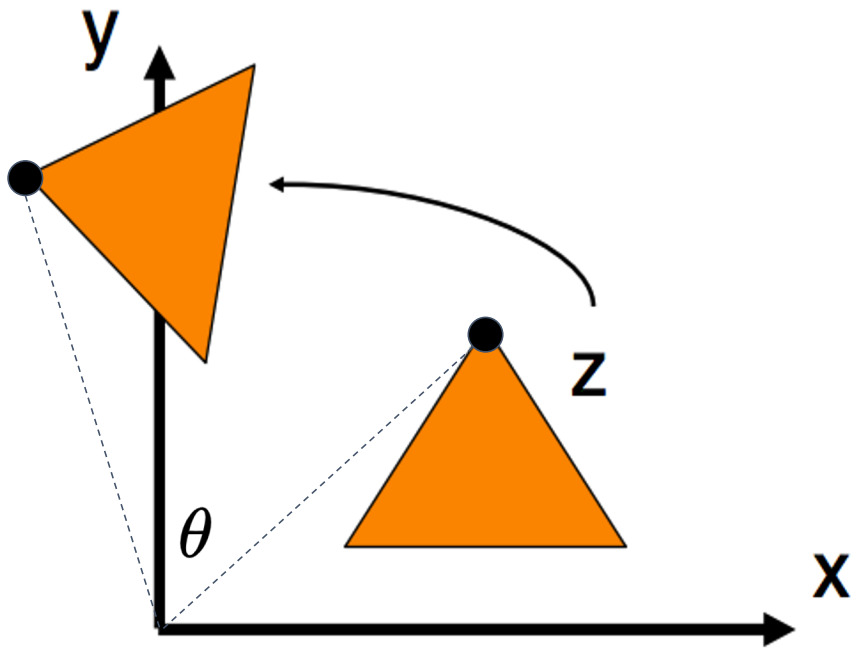
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} d_x \\ d_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Transformation: Rotation

$$p' = R \bullet p$$

Counter-clockwise



Triangle here helps us visualize the rotation.
But we are still considering one 2D point p

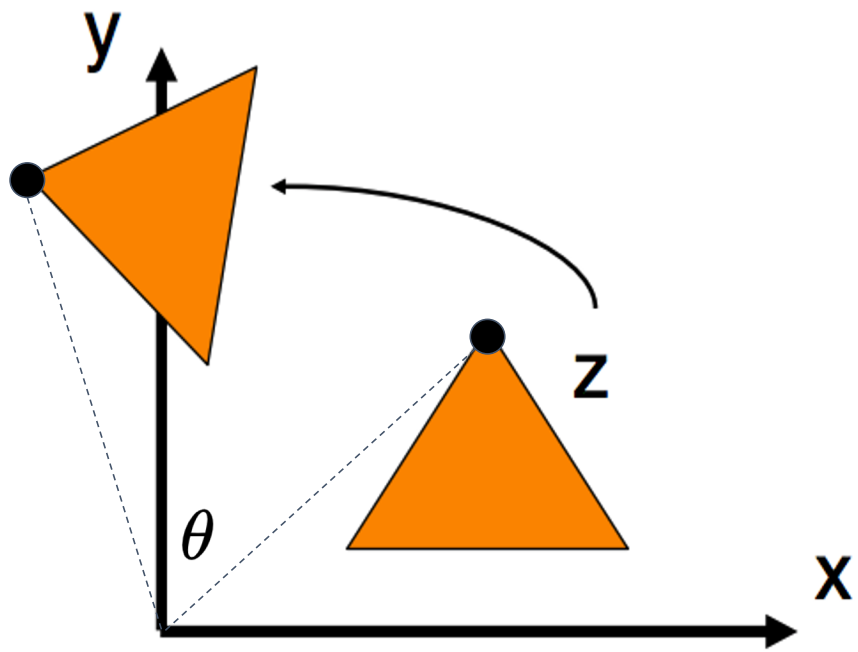
2D Transformation: Rotation

$$p' = R \bullet p$$

2D Rotation Matrix

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

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

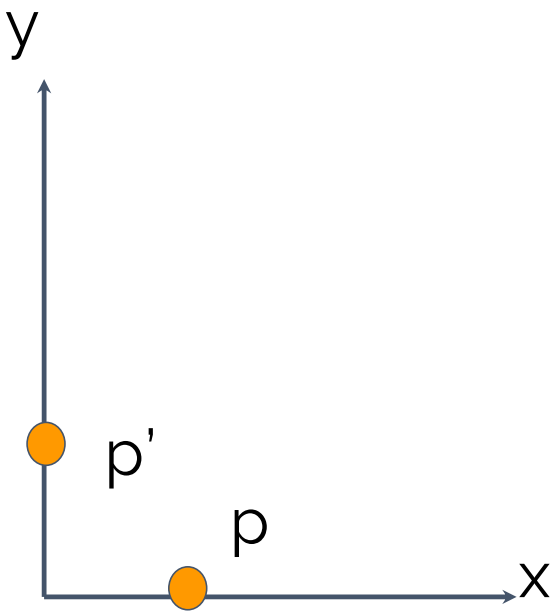


2D Transformation: Rotation

$$p' = R \bullet p$$

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

Quick Excise:
What's the coordinate of P = [1,0] rotate by 90 degree counterclockwise?



Combining rotation and translation

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

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 (somewhat hacky) 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 & Homogeneous Coordinate

- A point in cartesian coordinate <x,y> can be represented by <sx, sy, s > in homogeneous coordinate, where s is any scalar number.

E.g.: <2,3> in cartesian coordinate can be represented as <2,3,1> or <4,6,2> or <1,1.5,0.5>… in homogeneous coordinate

- A point in homogeneous coordinate <x,y,z> converted to cartesian coordinate by dividing the last element <x/z, y/z>.

- 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

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

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

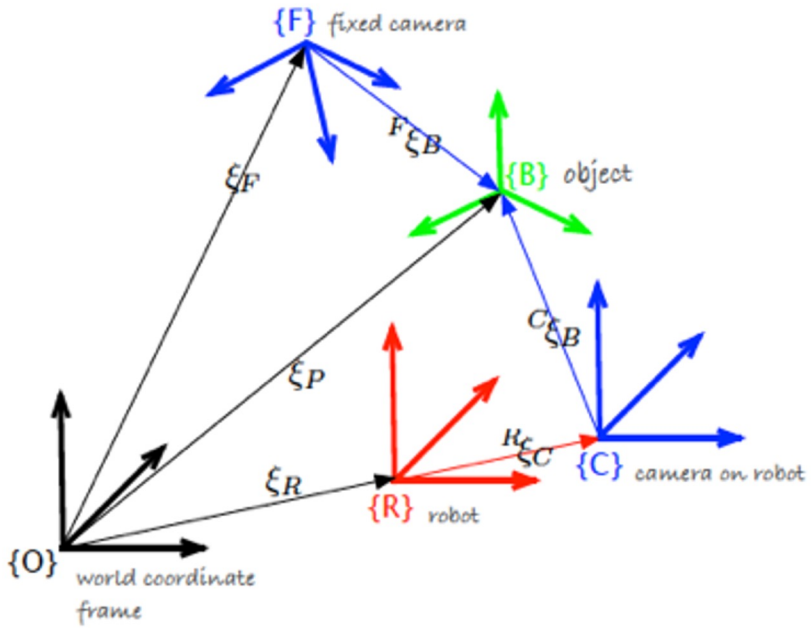
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} P' \\ 1 \end{bmatrix} = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P \\ 1 \end{bmatrix}$$

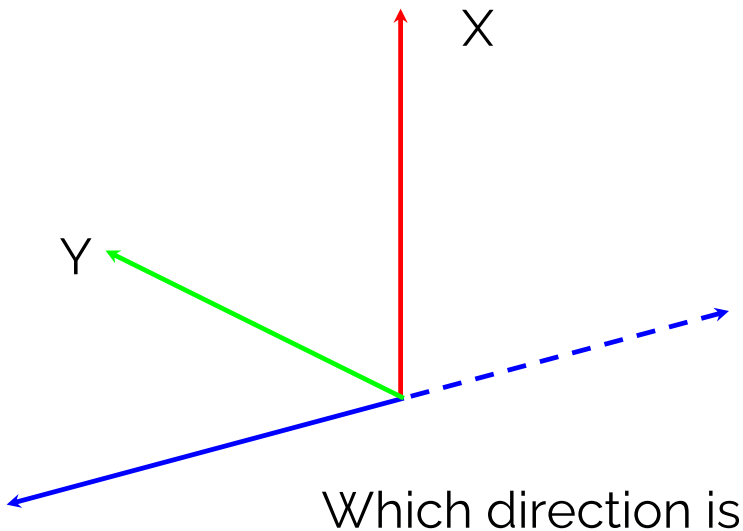
Note: Following the matrix multiplication rule. A transformation matrix always apply rotation first then translation

3D Transformation

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



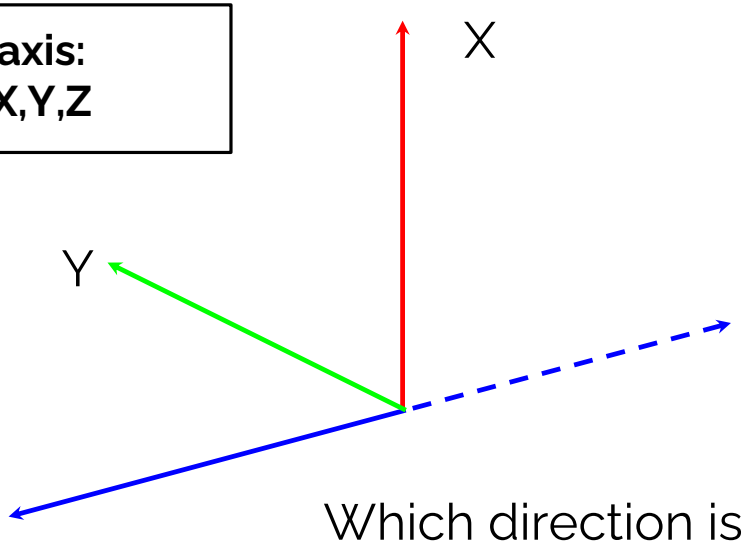
Define coordinate frame in 3D



Which direction is Z?

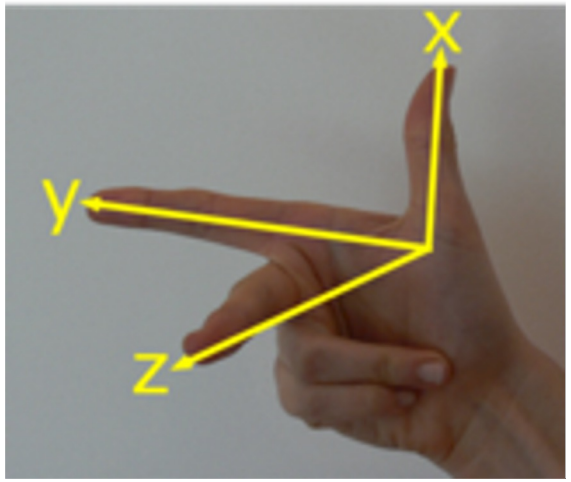
Right hand rule

Colored axis:
R,G,B -> X,Y,Z



Which direction is Z?

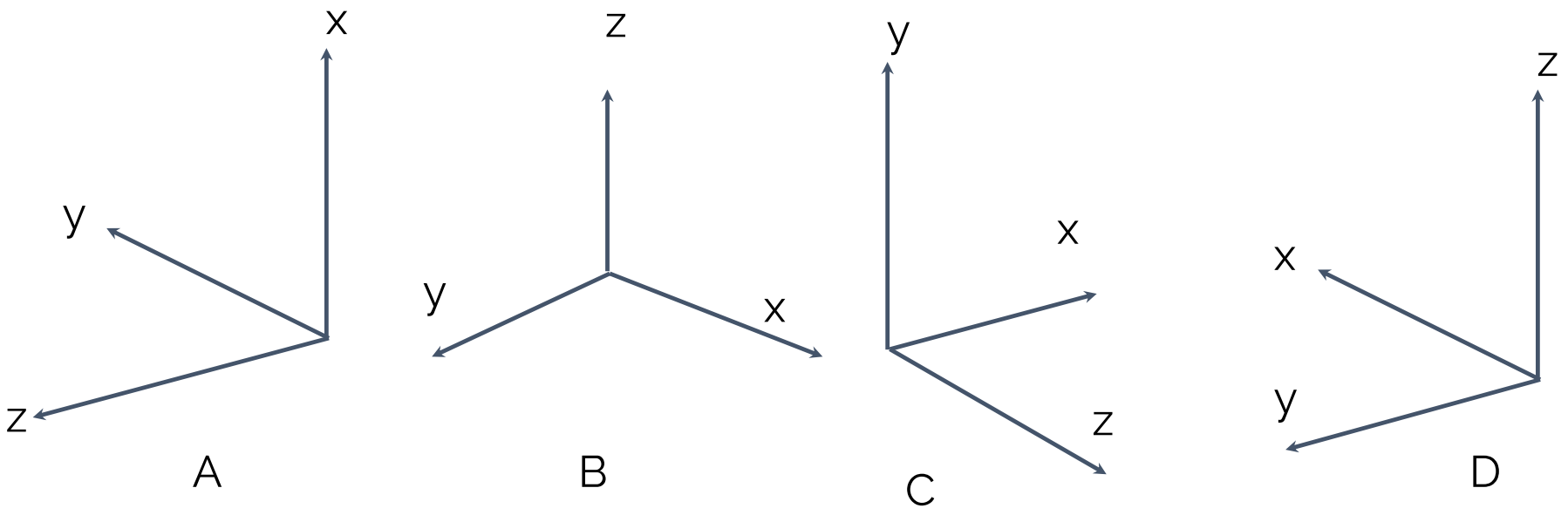
Start with thumb ☺



Most of robotics system's coordinate system follows right hand rule
-- not always true (e.g., in some graphics and physic engine directX Unity)
-- so, be careful!

Right hand rule

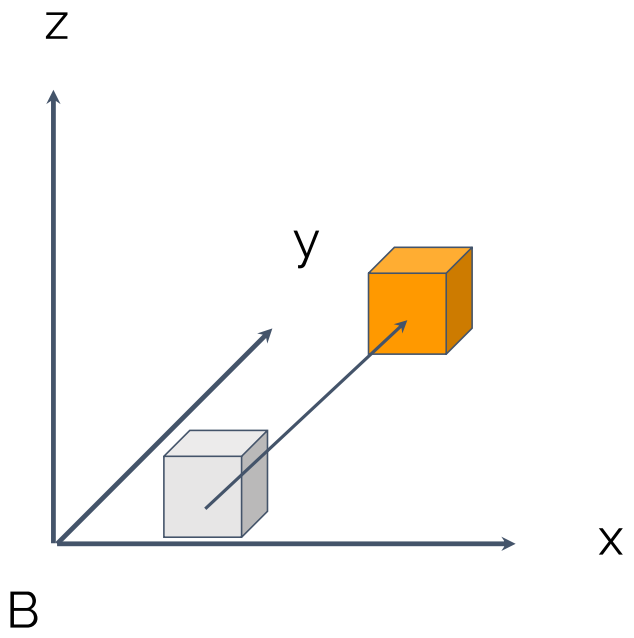
Which of following coordinate frame does not follow right hand rule?



3D Transformation: Translation

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

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

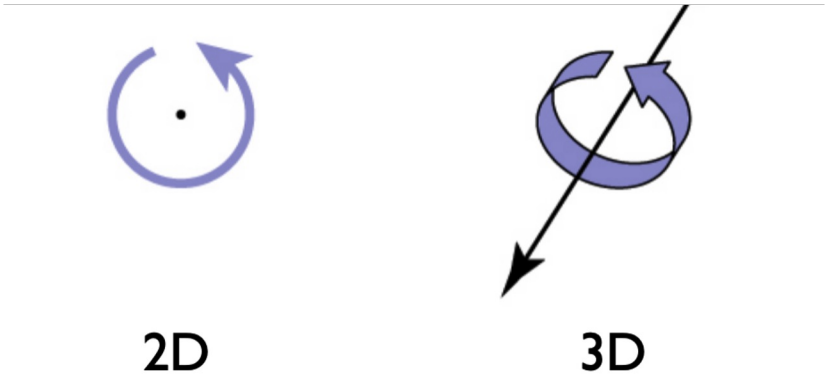


3D Transformation: Rotation

A rotation in 2D is around a point

A rotation in 3D is around an axis (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
- the most common choices for rotation axis are the X Y Z-axis (Euler angle representation)



2D

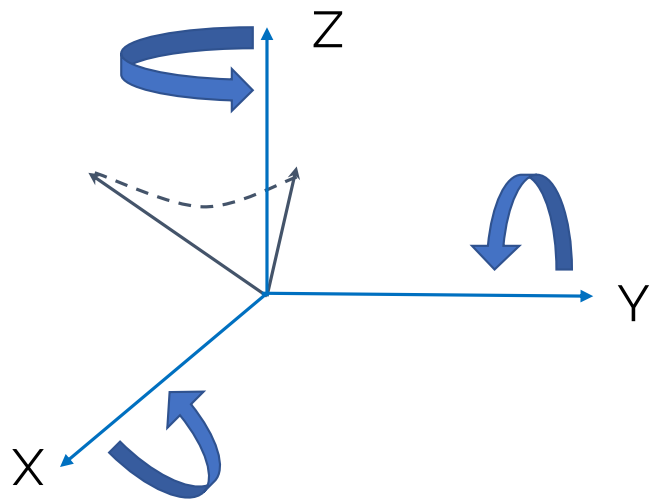
3D

3D Transformation: Rotation

First let's look at rotation along z-axis: the 3D points Z value won't change, equivalent to 2D rotation and keep z the same

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

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + ycos\theta$$
$$z' = z$$



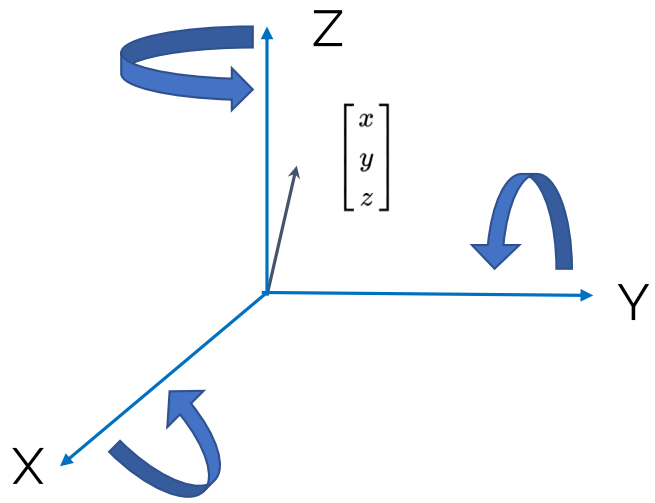
Basic 3D Rotation:

We can do the same operation for rotation along x or y

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

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

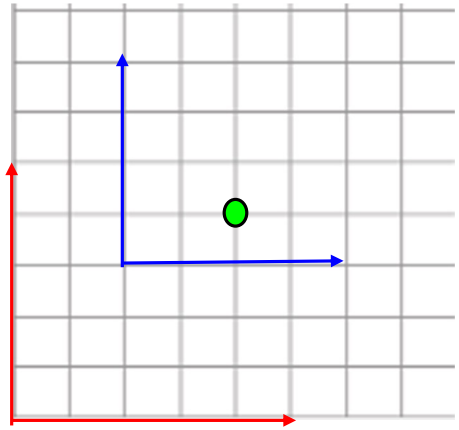
$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



You don't need to remember this…

Reference Frames (Coordinate System)

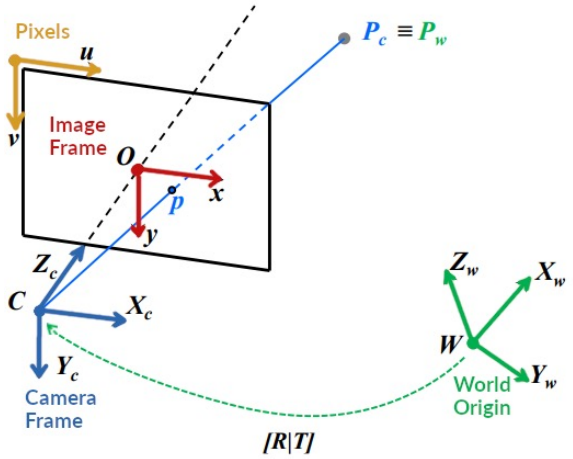
- Up to now we have look at transformation in a single reference frames. However, in a complex robotic system we often need to define **many reference frames**.
- The same 3D point might have different coordinate if we use different reference frames, next we will learn how to transform between different reference frames.



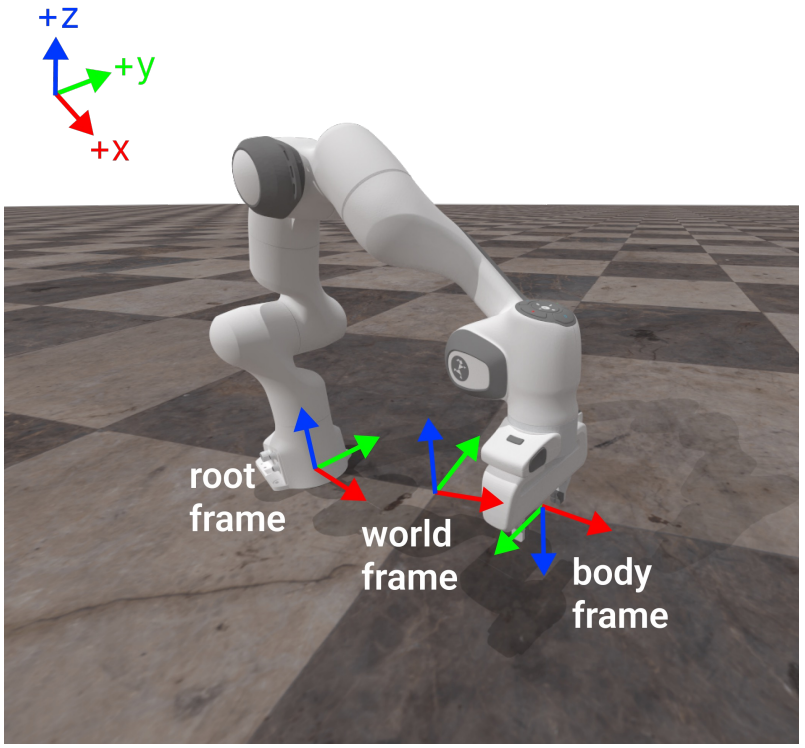
Example: green dot's coordinate is (2,1) in blue reference frame, but its coordinate is (4,4) in red reference frame.

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

Reference Frames: Examples



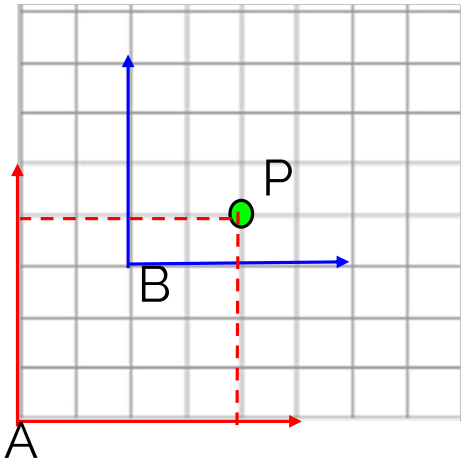
camera frame



End-effector frame

Robot frame

Changing Reference Frames



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

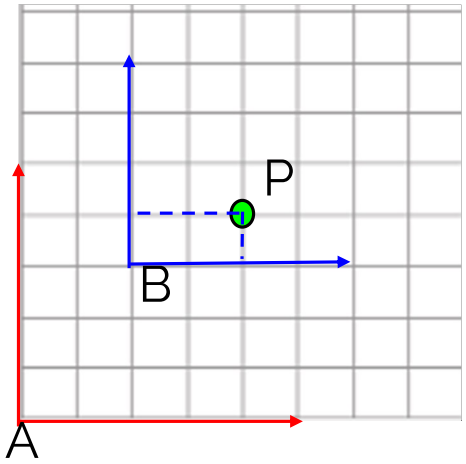
$$ {}^{A}P = {}^{A}T_{B} \cdot {}^{B}P $$

How to compute ${}^{A}T_{B}$?

"Canceled" out

Changing Reference Frames:

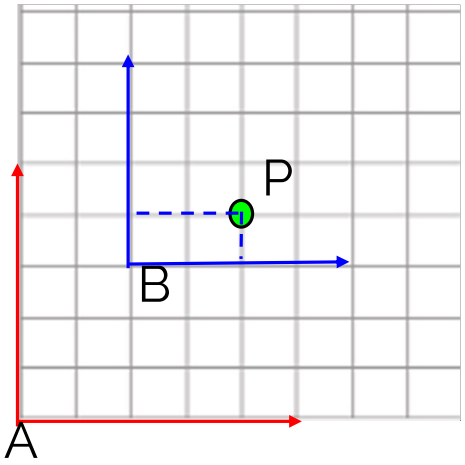
How to compute ${}^{A}T_{B}$? **(${}^{A}P = {}^{A}T_{B} \cdot {}^{B}P$)**

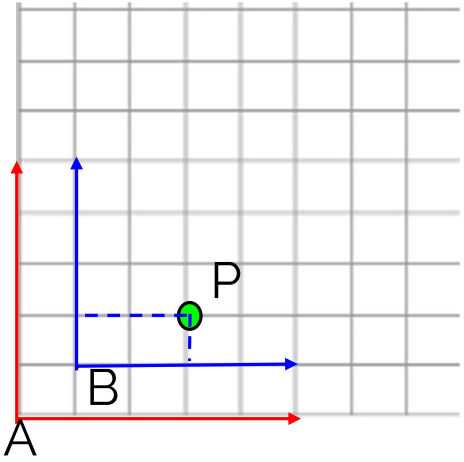


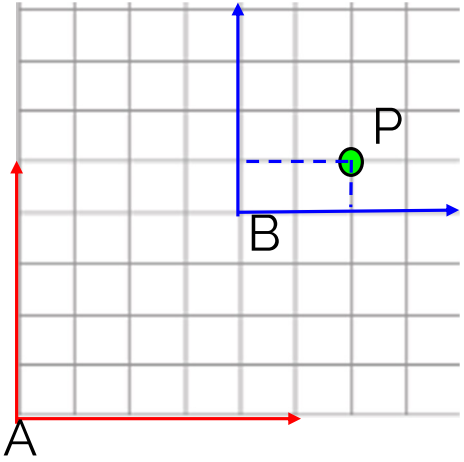
Changing Reference Frames:

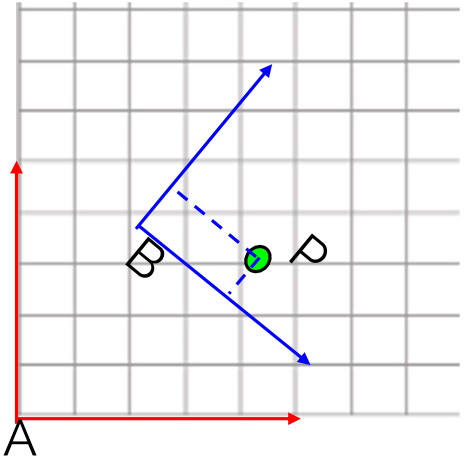
How to compute ${}^{A}T_{B}$?

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





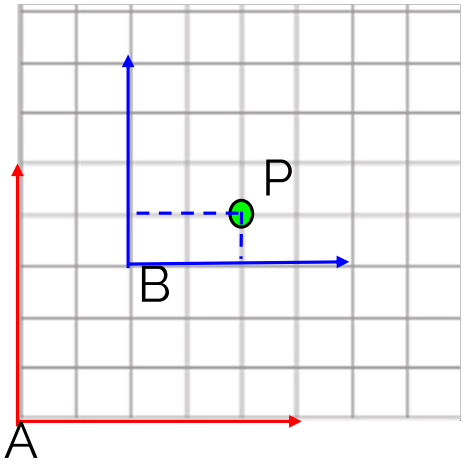




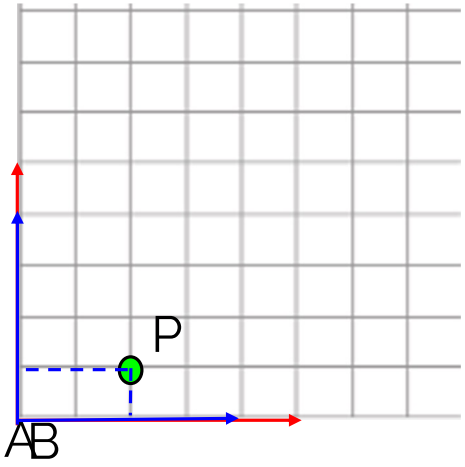
Changing Reference Frames:

How to compute ${}^{A}T_{B}$?

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



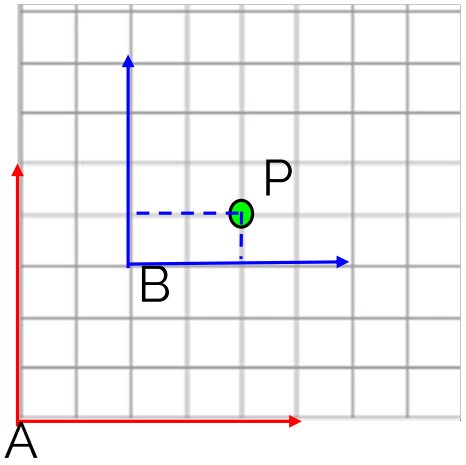
Let's first make Frame B identical to Frame A.
Now ${}^{A}P = {}^{B}P = (2,1)$



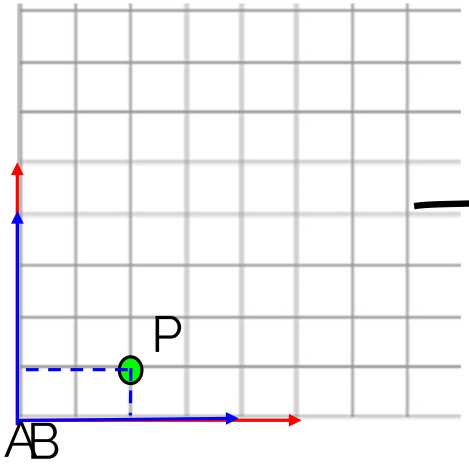
Changing Reference Frames:

How to compute ${}^{A}T_{B}$?

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



Let's first make Frame B identical to Frame A.
Now ${}^{A}P = {}^{B}P = (2,1)$

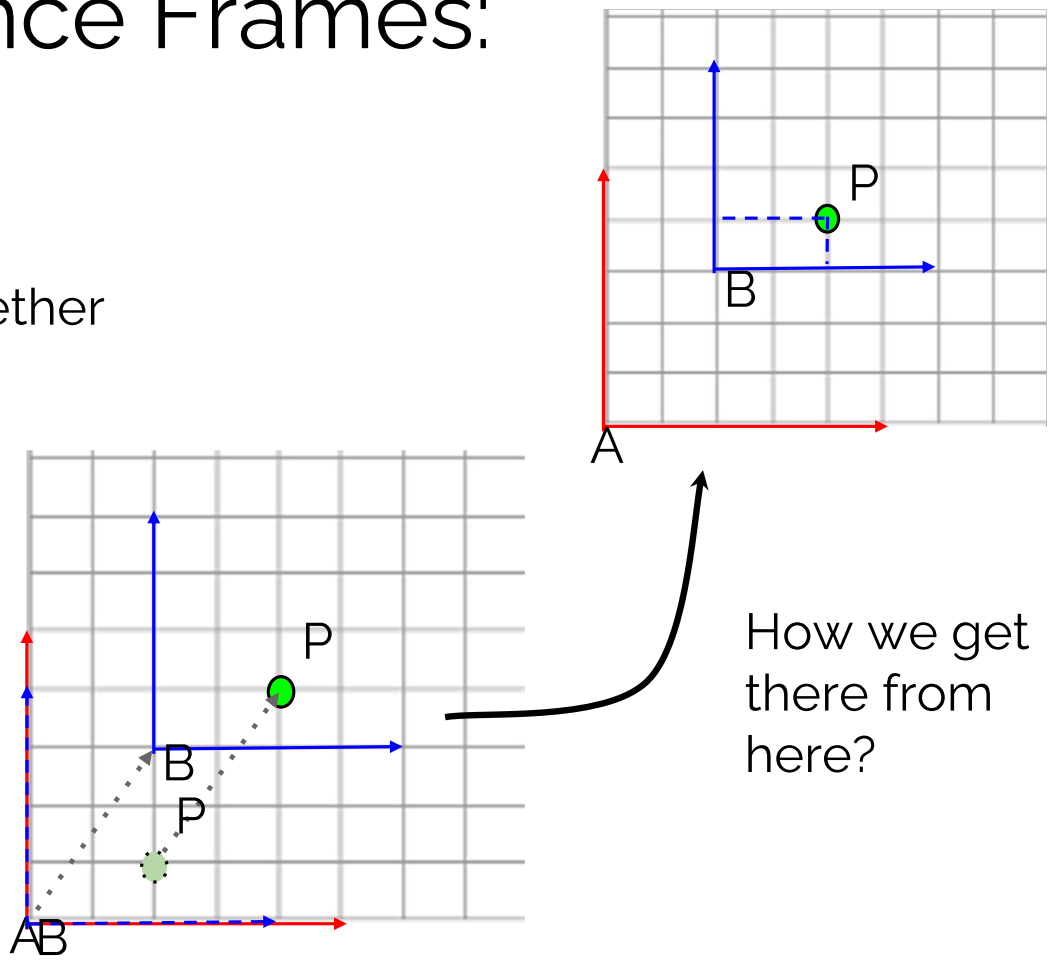


How we get there from here?

Changing Reference Frames:

How to compute ${}^{A}T_{B}$?

Simply translate Frame B together with d = (2,3), we will get the
${}^{A}P = {}^{B}P + d$



Changing Reference Frames:

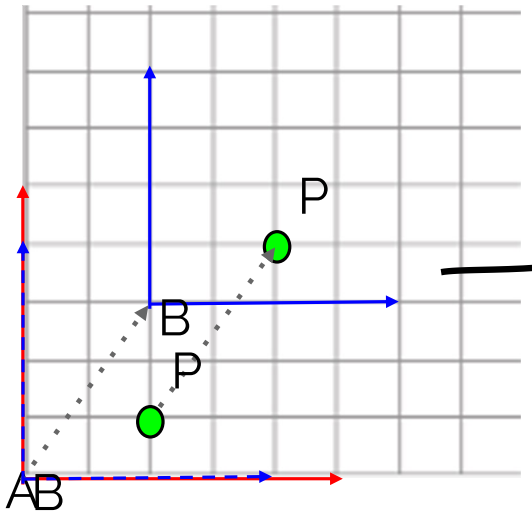
How to compute ${}^{A}T_{B}$?

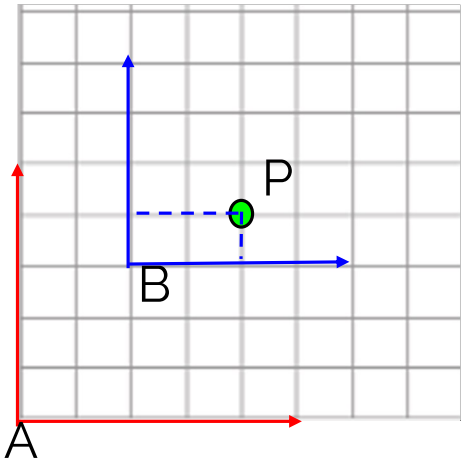
Simply translate Frame B together with d = (2,3), we will get the
${}^{A}P = {}^{B}P + d$

Therefore in this case

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

(No rotation only translation by d)





How we get there from here?

Change coordinate system

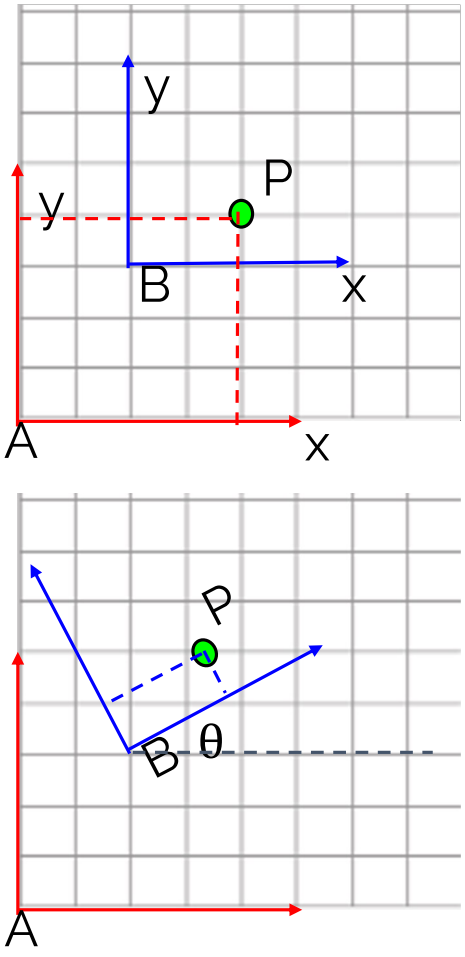
Let's check:

${}^{B}P$ = (2,1)

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

$${}^{A}T_{B}\,{}^{B}P = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+2 \\ 1+3 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

How about more general case with rotation?

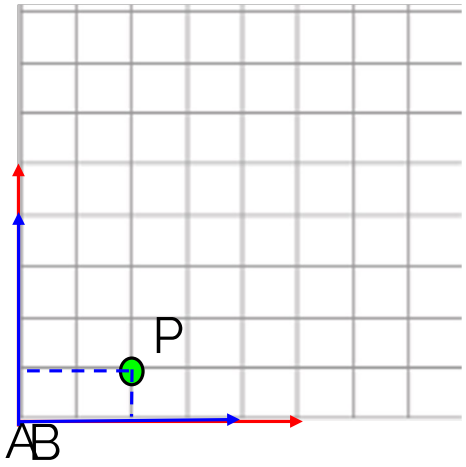


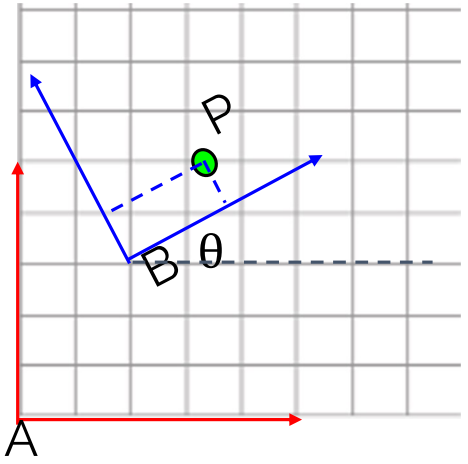
Change coordinate system (general case)

How to compute ${}^{A}T_{B}$?

Same steps:

1. Suppose the point P is rigidly attached to ref Frame B
2. Suppose Frame B is aligned with Frame A



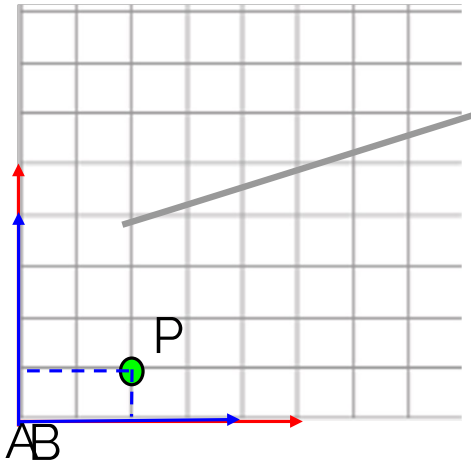


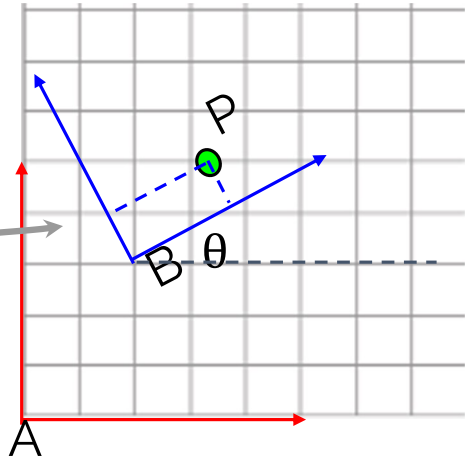
Change coordinate system (general case)

How to compute ${}^{A}T_{B}$?

Same steps:

1. Suppose the point P is rigidly attached to ref Frame B
2. Suppose Frame B is aligned with Frame A





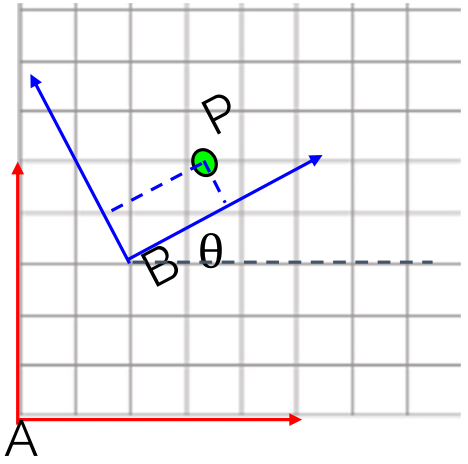
Finding ${}^{A}T_{B}$ is same as find the transformation that move the current Frame B and P to the target location

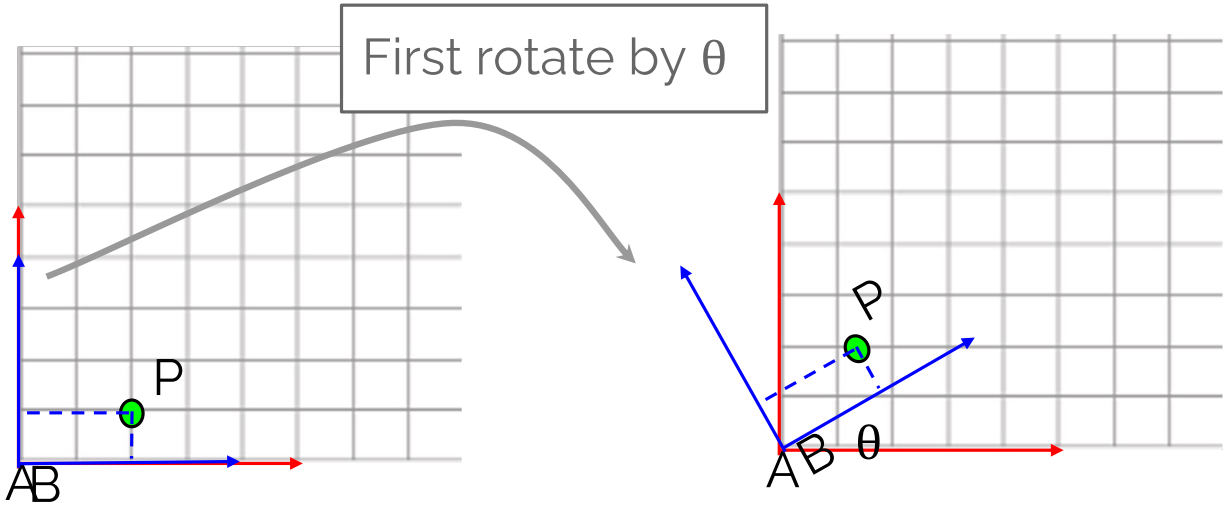
Change coordinate system (general case)

How to compute ${}^{A}T_{B}$?

Same steps:

1. Suppose the point P is rigidly attached to ref Frame B
2. Suppose Frame B is aligned with Frame A



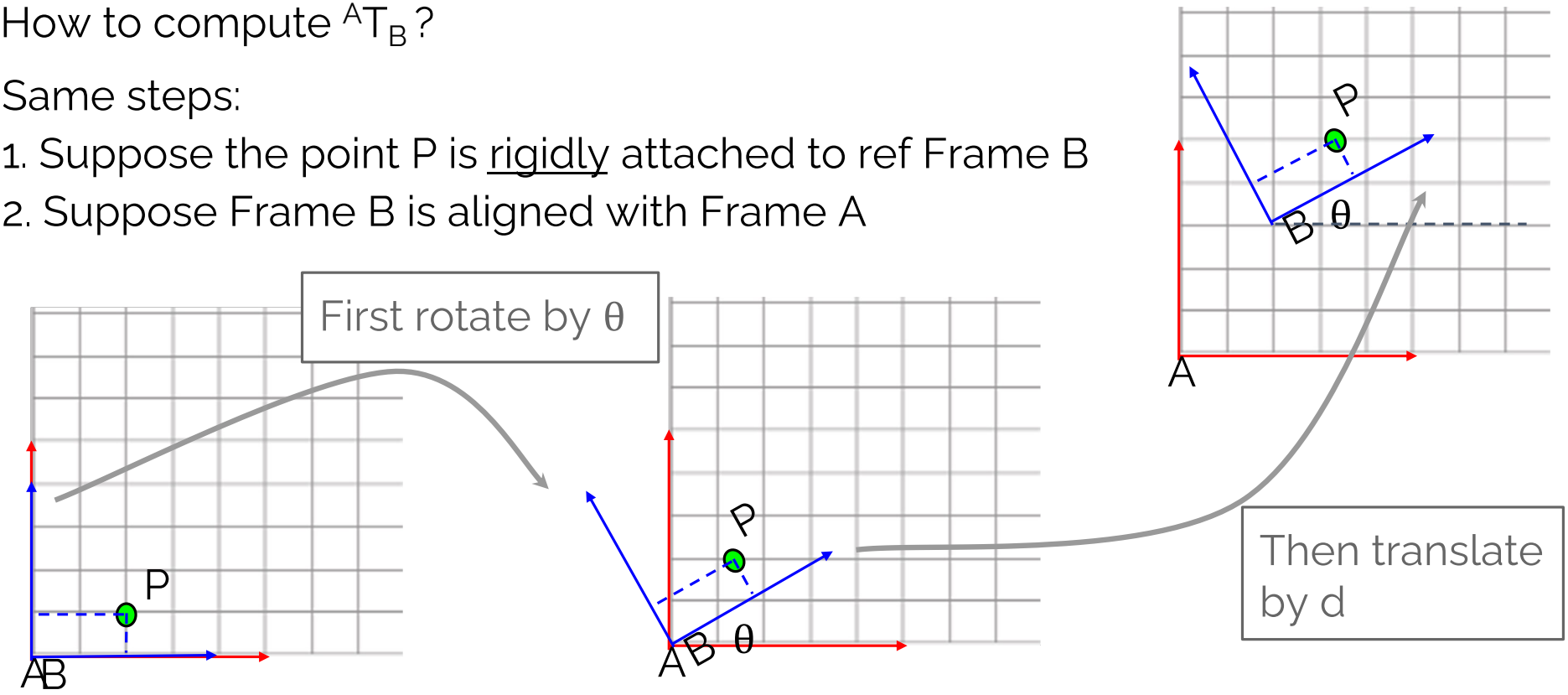


Change coordinate system (general case)

How to compute ${}^{A}T_{B}$?

Same steps:

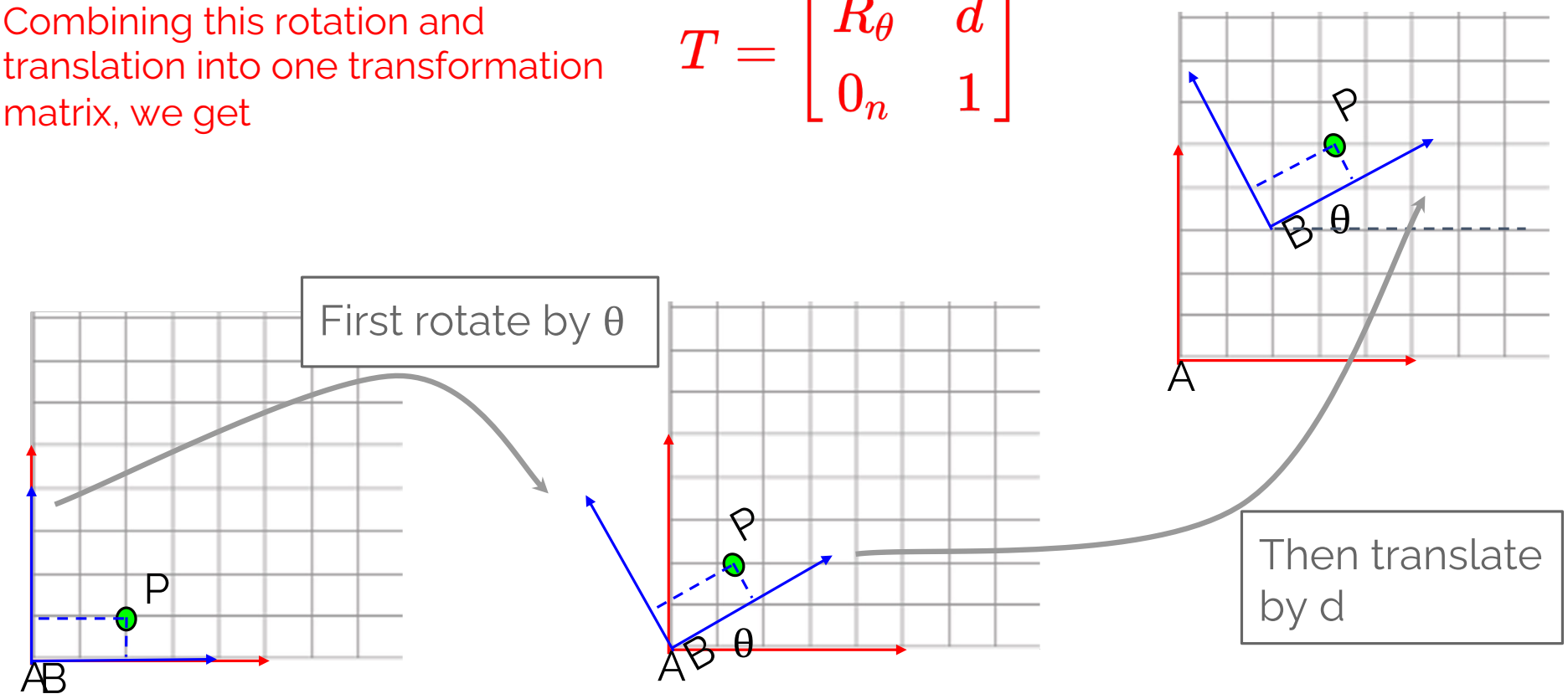
1. Suppose the point P is rigidly attached to ref Frame B
2. Suppose Frame B is aligned with Frame A



Change coordinate system (general case)

Combining this rotation and translation into one transformation matrix, we get

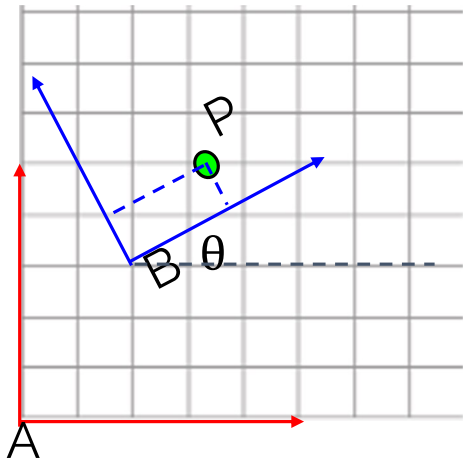
$$T = \begin{bmatrix} R_\theta & d \\ 0_n & 1 \end{bmatrix}$$



Change coordinate system (general case)

Combining this rotation and translation into one transformation matrix, we get

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

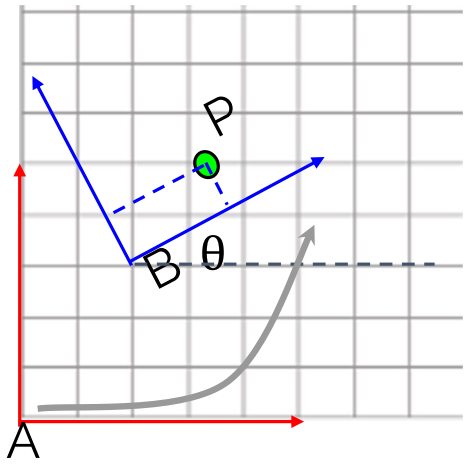


This is the transformation ${}^{A}T_{B}$ that change the coordinate frame from B to A.

Change coordinate system (general case)

Combining 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 ${}^{A}T_{B}$ that change the coordinate frame from B to A.

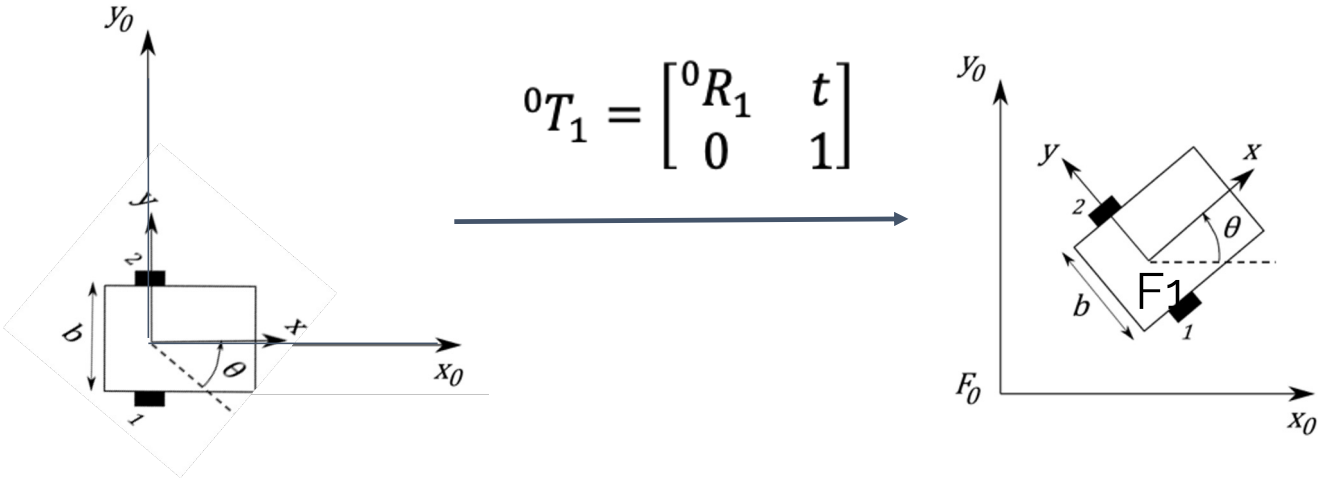
However, geometrically it describes the motion from Frame A to B.

${}^{A}T_{B}$ also describe Frame B's "**pose**" in Frame A, where the rotation represent component R describe the B's orientation in Frame A, and the translation represents B' position in Frame A

Object Pose Definition:

The position and orientation of an object's coordinate frame is referred as its pose.

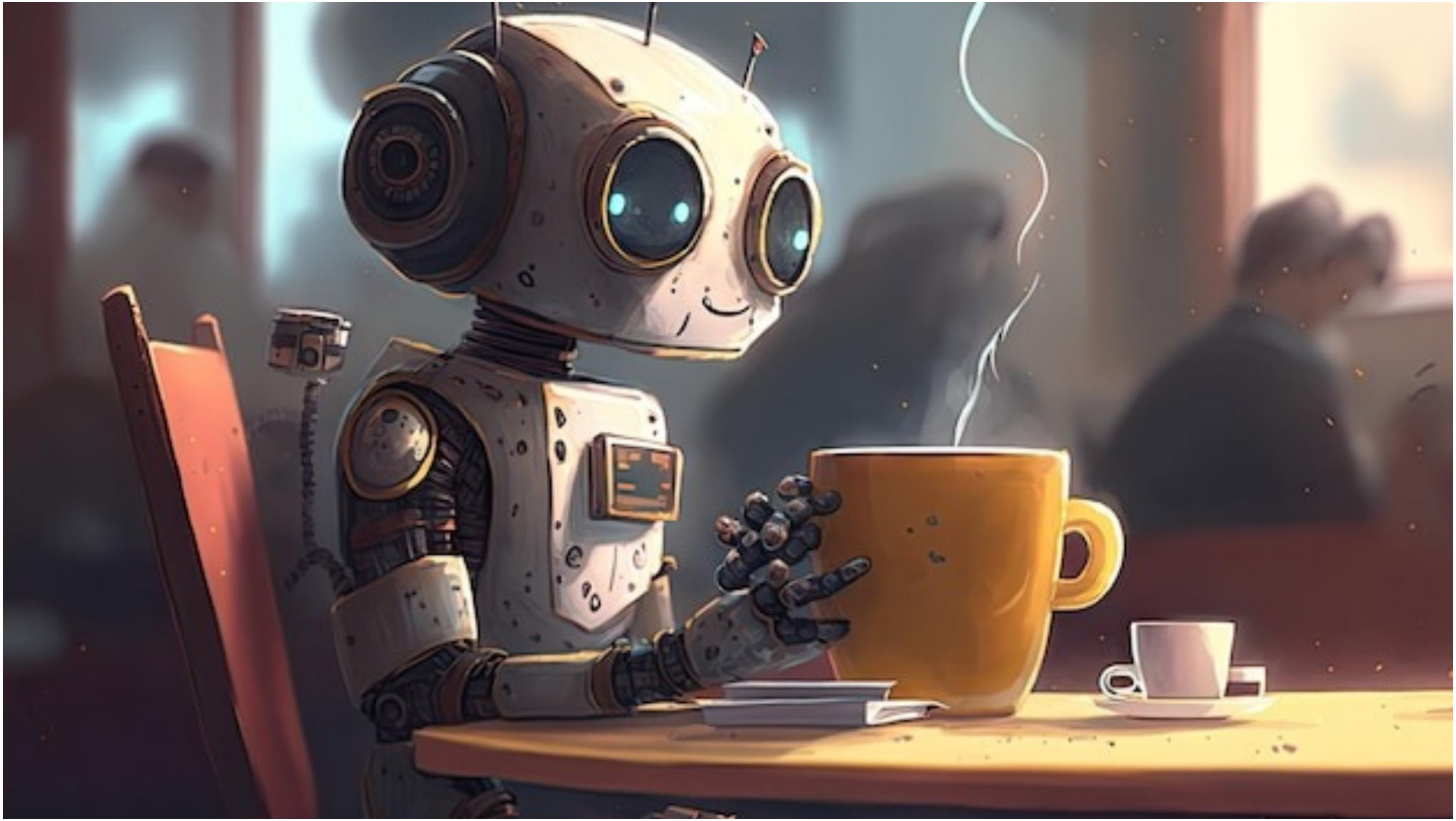
One way to think about it:
We rigidly attached coordinate frames to objects of interest (e.g., F1 on the car). To specify the position and orientation of the object, we merely specify the position and orientation of the attached coordinate frame in the global frame (F0) which is ${}^0T_1$.



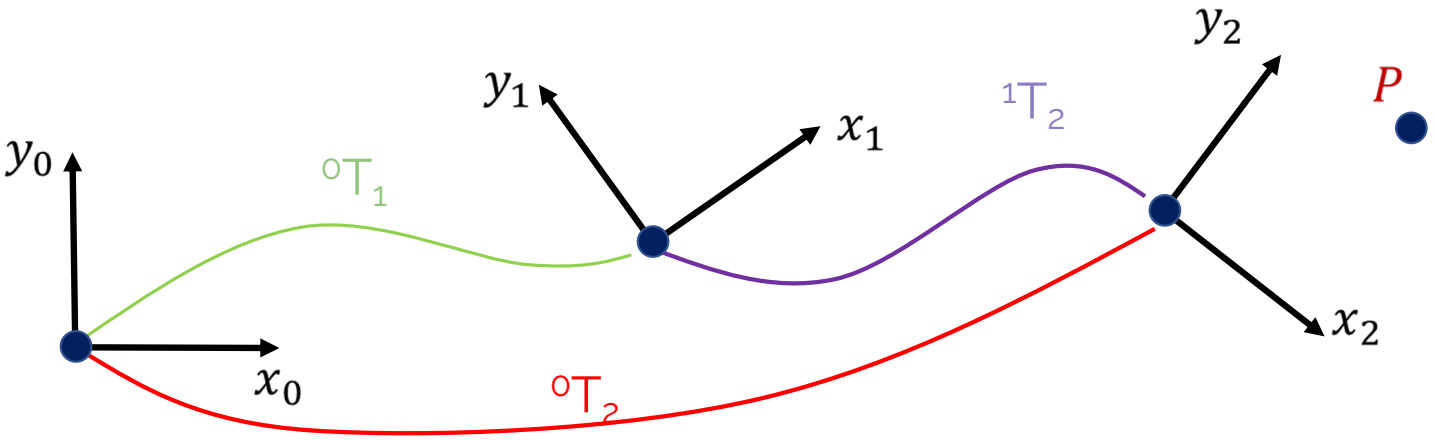
$$ {}^0T_1 = \begin{bmatrix} {}^0R_1 & t \\ 0 & 1 \end{bmatrix} $$

Summary: What is ${}^{A}T_{B}$?

- ${}^{A}T_{B}$ is a rigid transformation matrix (3x3 matrix in 2D, 4x4 in 3D)
- ${}^{A}T_{B}$ represents the transform that change the coordinate frame From B to A: ${}^{A}P = {}^{A}T_{B}\,{}^{B}P$
- ${}^{A}T_{B}$ geometrically describes the motion from Frame A to B.
- ${}^{A}T_{B}$ 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:

<u>This is the composition law for homogeneous transformations.</u>

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

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

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

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

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

Chained 3D Rotation

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

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) = \overset{\text{yaw}}{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}} \overset{\text{pitch}}{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}} \overset{\text{roll}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}$$

Chained 3D Rotation

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

$$R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) = \underbrace{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{yaw}} \underbrace{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}}_{\text{pitch}} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}_{\text{roll}}$$

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.

Rotation Matrix

Rotation matrix has a number of highly useful properties:

- R is an orthonormal matrix: Its columns are orthogonal unit vectors. $R^{-1} = R^T$
- determinant of the matrix |R| = +1
- The length of the vector is unchanged after transformation

3D Transformation

Putting 3D rotation and 3D translation together we get 3D transformation matrix (a.k.a rigid transformation, SE(3) transformation …)

$$\tilde{P} = T \tilde{P} \qquad T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} & R & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^1x \\ {}^1y \\ {}^1z \\ 1 \end{bmatrix}$$

Definition of SE(3) transformation:

$$SE(3) = \left\{ \mathbf{A} \;\middle|\; \mathbf{A} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{r} \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array}\right], \mathbf{R} \in R^{3\times 3}, \mathbf{r} \in R^3, \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}, |\mathbf{R}| = 1 \right\}$$

here |.| is the determinant function

Inverse of a Transformation Matrix

Note: while the inverse of a Rotation matrix is its transpose, it is not true for transformation matrix

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

Instead:

$$T^{-1} = \begin{bmatrix} R & d \\ 0_n & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T d \\ 0_n & 1 \end{bmatrix}$$

To verify:

$$\begin{bmatrix} R & d \\ 0_n & 1 \end{bmatrix} \begin{bmatrix} R^T & -R^T d \\ 0_n & 1 \end{bmatrix} = \begin{bmatrix} RR^T & -RR^T d + d \\ 0_n & 1 \end{bmatrix} = \begin{bmatrix} I_{n\times n} & 0_n \\ 0_n & 1 \end{bmatrix} = I_{(n+1)\times(n+1)}$$

Other 3D Rotation Representation

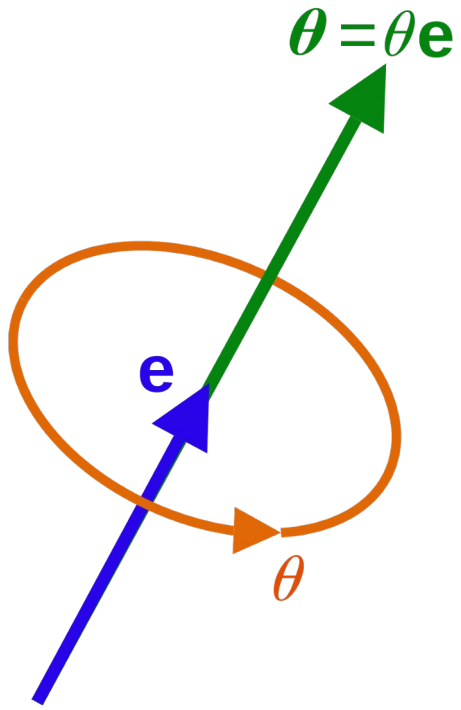
There are many ways to specify rotation

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

Axis Angle

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.



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:

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

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**

$$\left(\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}}\right)$$ **??**

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˚)**

Quaternion:

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

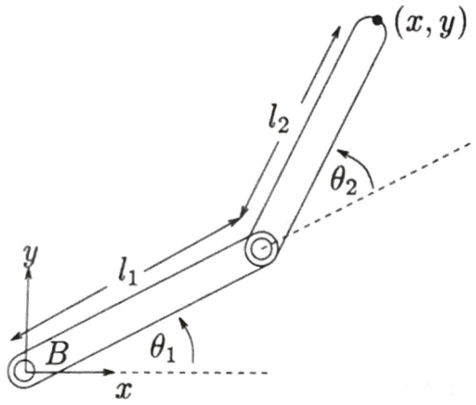
Inverse a quaternion:

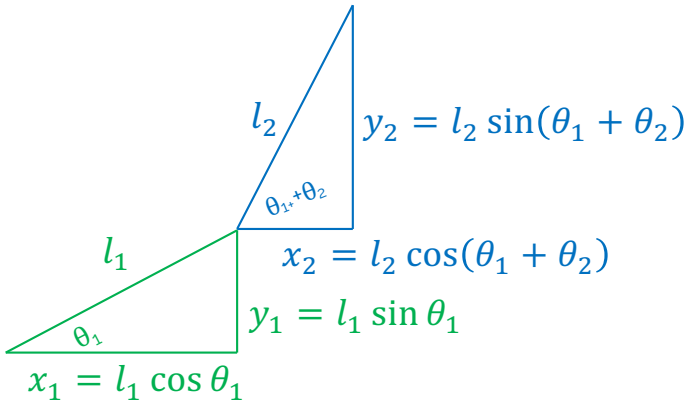
- keep the rotation axis rotate backward
- Inverse of (x,y,z,w) is (x,y,z,-w)
- (x,y,z,w) is equivalent to (-x,-y,-z,-w)

Forward Kinematics

Forward Kinematics of 2-link Manipulator

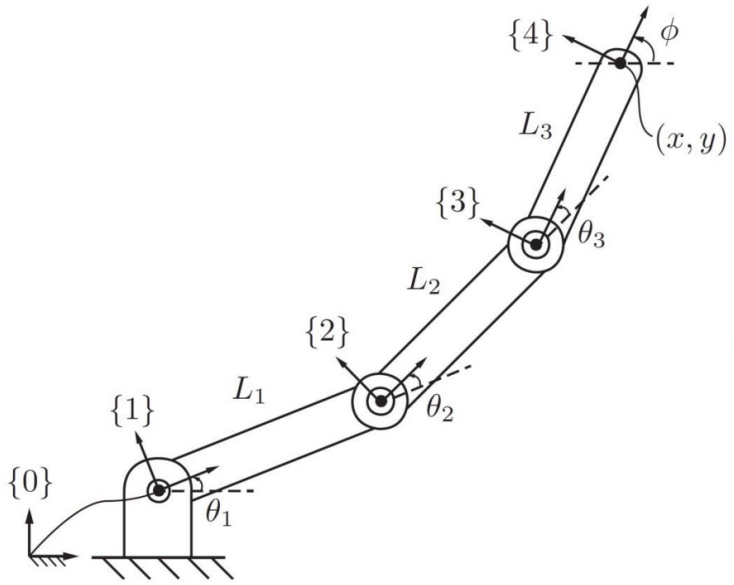
Given joint angles, calculate position of end-effector





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

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


That’s it for today!
Questions?