



上海科技大学  
ShanghaiTech University

## CS283: Robotics Spring 2026: Kinematics

---

Sören Schwertfeger / 师泽仁

ShanghaiTech University

# KINEMATICS

---

# Motivation

- Autonomous mobile robots move around in the environment.

Therefore **ALL** of them:

- They need to know **where** they **are**.
- They need to know **where** their **goal** is.
- They need to know **how** to get there.

- **Odometry!**

- Robot:

- I know how fast the wheels turned =>
- I know how the robot moved =>
- I know where I am 😊

# Odometry

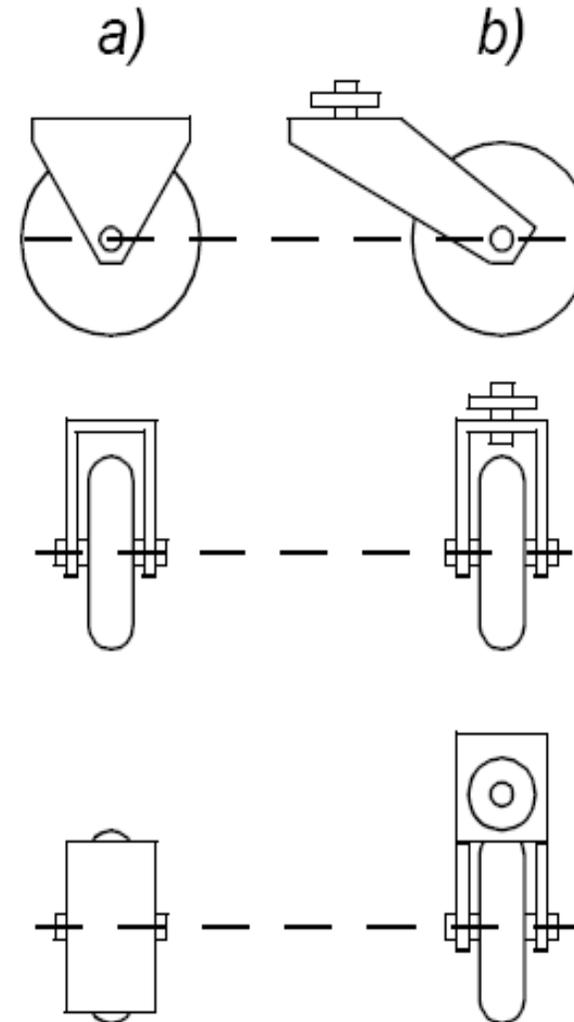
- Robot:
  - I know how fast the wheels turned =>
  - I know how the robot moved =>
  - I know where I am 😊
- Marine Navigation: Dead reckoning (using heading sensor)
- Sources of error (AMR pages 269 - 270):
  - Wheel slip
    - Uneven floor contact (non-planar surface)
    - Robot kinematic: tracked vehicles, 4 wheel differential drive..
  - Integration from speed to position: Limited resolution (time and measurement)
  - Wheel misalignment
  - Wheel diameter uncertainty
  - Variation in contact point of wheel

# Mobile Robots with Wheels

- Wheels are the most appropriate solution for most applications
- Three wheels are sufficient to guarantee stability
- With more than three wheels an appropriate suspension is required
- Selection of wheels depends on the application

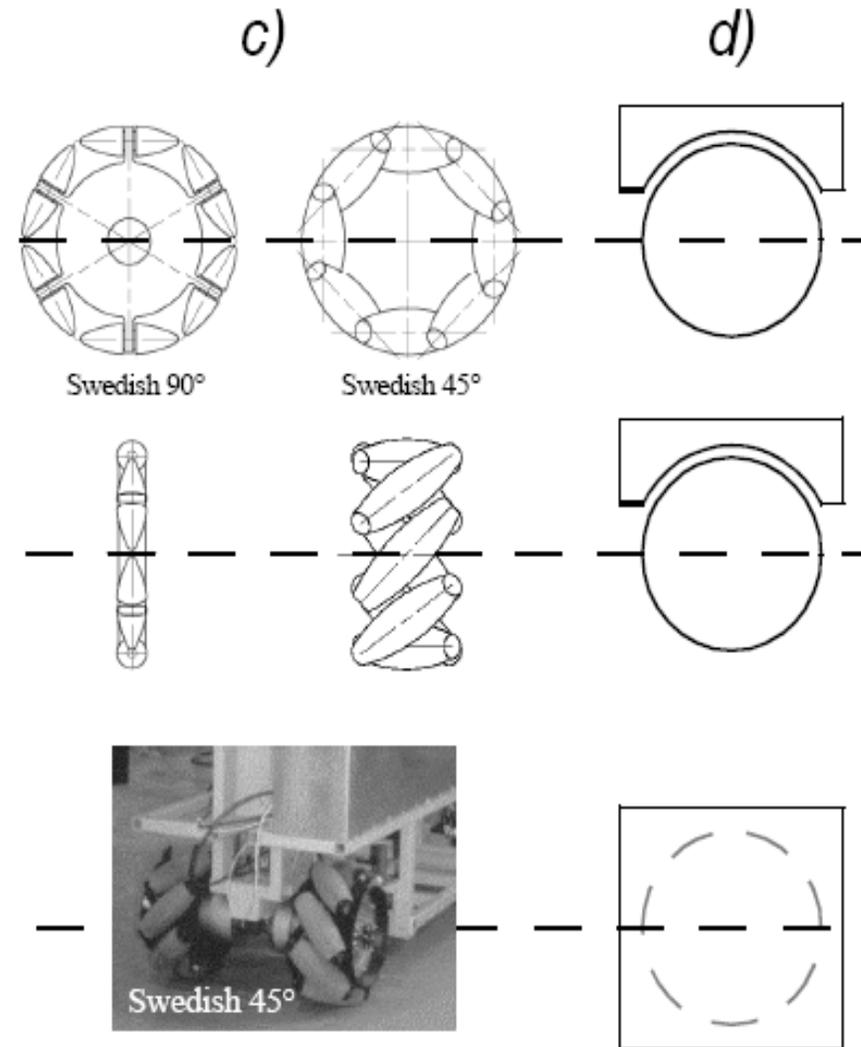
# The Four Basic Wheels Types I

- a) Standard wheel: Two degrees of freedom; rotation around the (motorized) wheel axle and the contact point
- b) Castor wheel: Three degrees of freedom; rotation around the wheel axle, the contact point and the castor axle



# The Four Basic Wheels Types II

- c) Swedish wheel: Three degrees of freedom; rotation around the (motorized) wheel axle, around the rollers and around the contact point
- d) Ball or spherical wheel: Suspension technically not solved

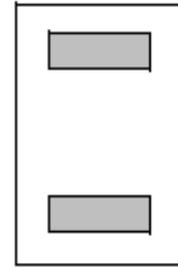


# Characteristics of Wheeled Robots and Vehicles

- Stability of a vehicle is be guaranteed with 3 wheels
  - center of gravity is within the triangle with is formed by the ground contact point of the wheels.
- Stability is improved by 4 and more wheel
  - however, this arrangements are hyperstatic and require a flexible suspension system.
- Bigger wheels allow to overcome higher obstacles
  - but they require higher torque or reductions in the gear box.
- Most arrangements are non-holonomic (see chapter 3)
  - require high control effort
- Combining actuation and steering on one wheel makes the design complex and adds additional errors for odometry.

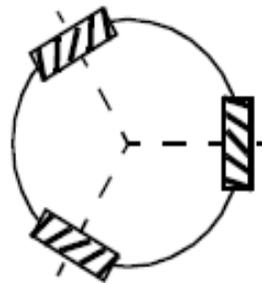
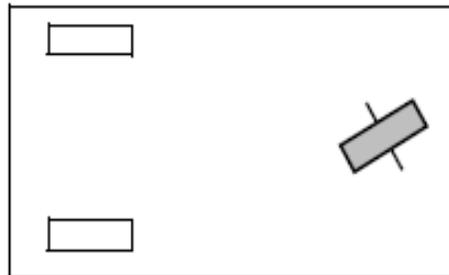
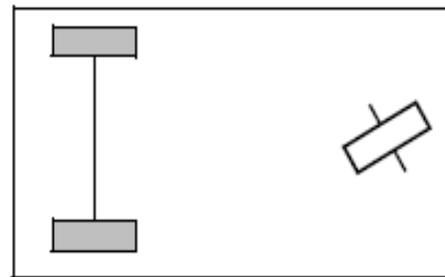
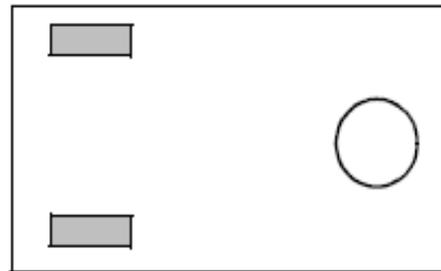
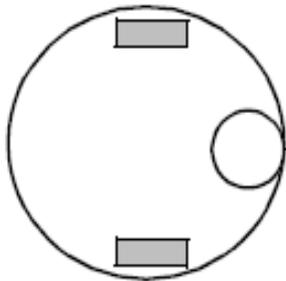
# Different Arrangements of Wheels I

- Two wheels

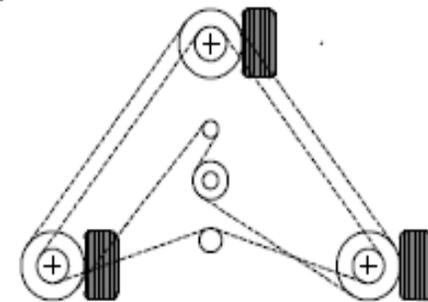


*Center of gravity below axle*

- Three wheels



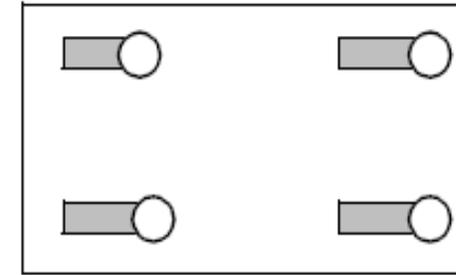
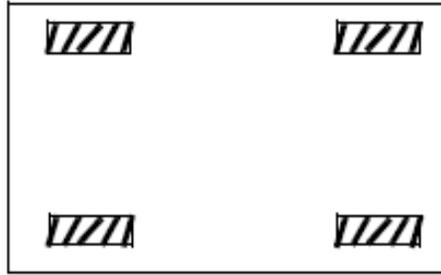
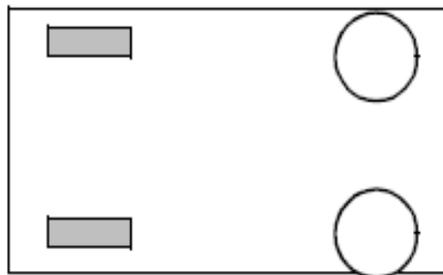
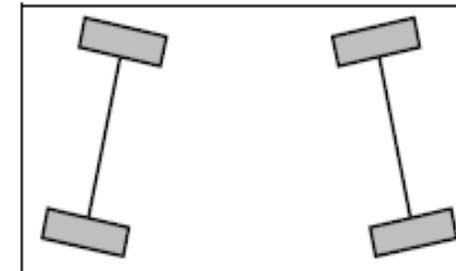
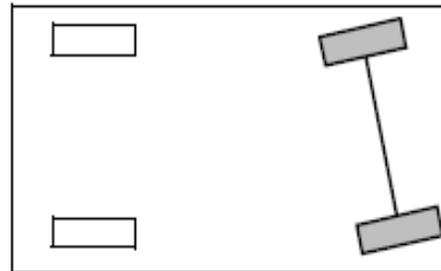
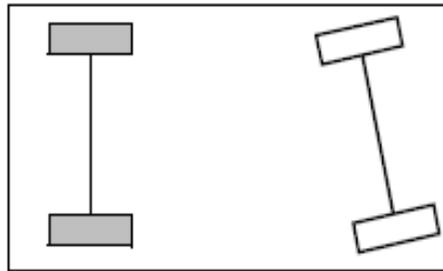
Omnidirectional Drive



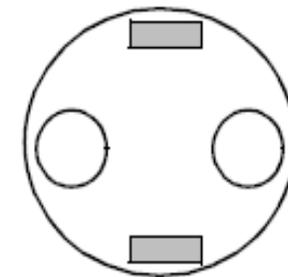
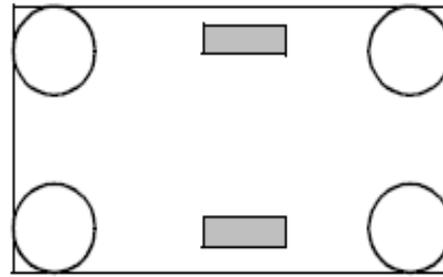
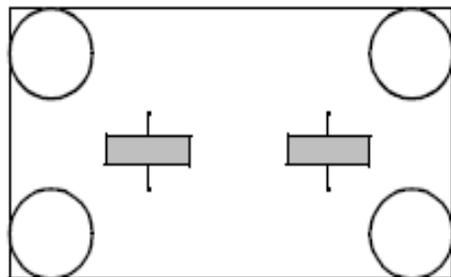
Synchro Drive

# Different Arrangements of Wheels II

- Four wheels

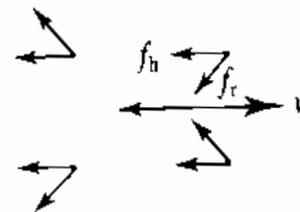
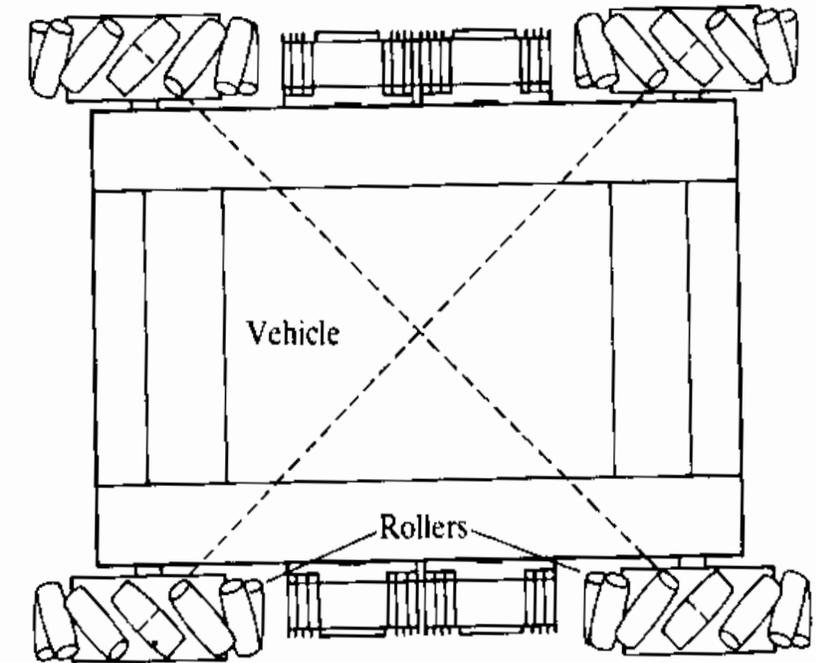
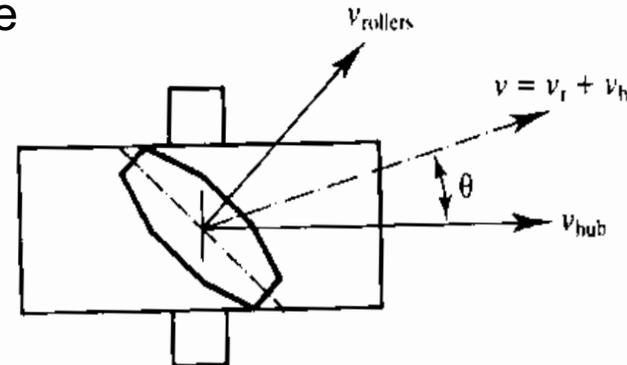
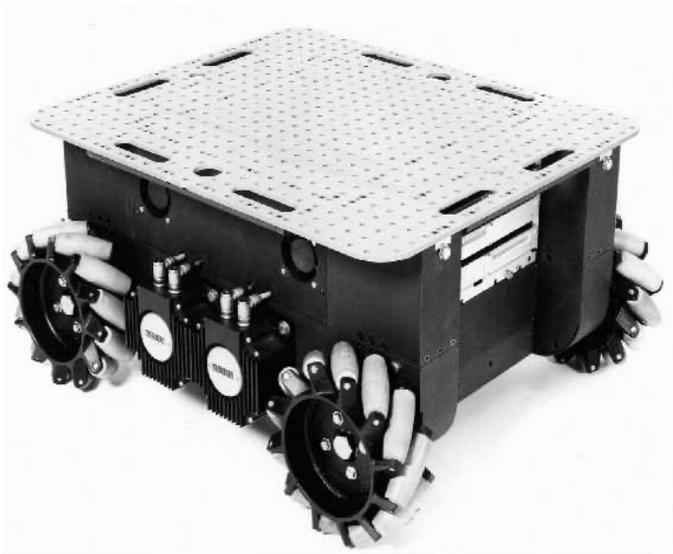


- Six wheels

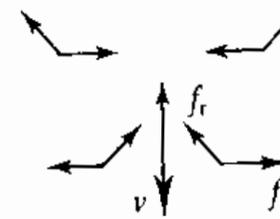


# Uranus, CMU: Omnidirectional Drive with 4 Wheels

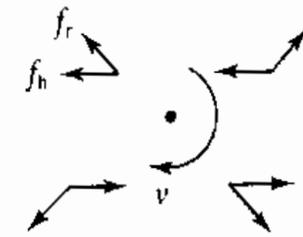
- Movement in the plane has 3 DOF
  - thus only three wheels can be independently controlled
  - It might be better to arrange three swedish wheels in a triangle



Forward



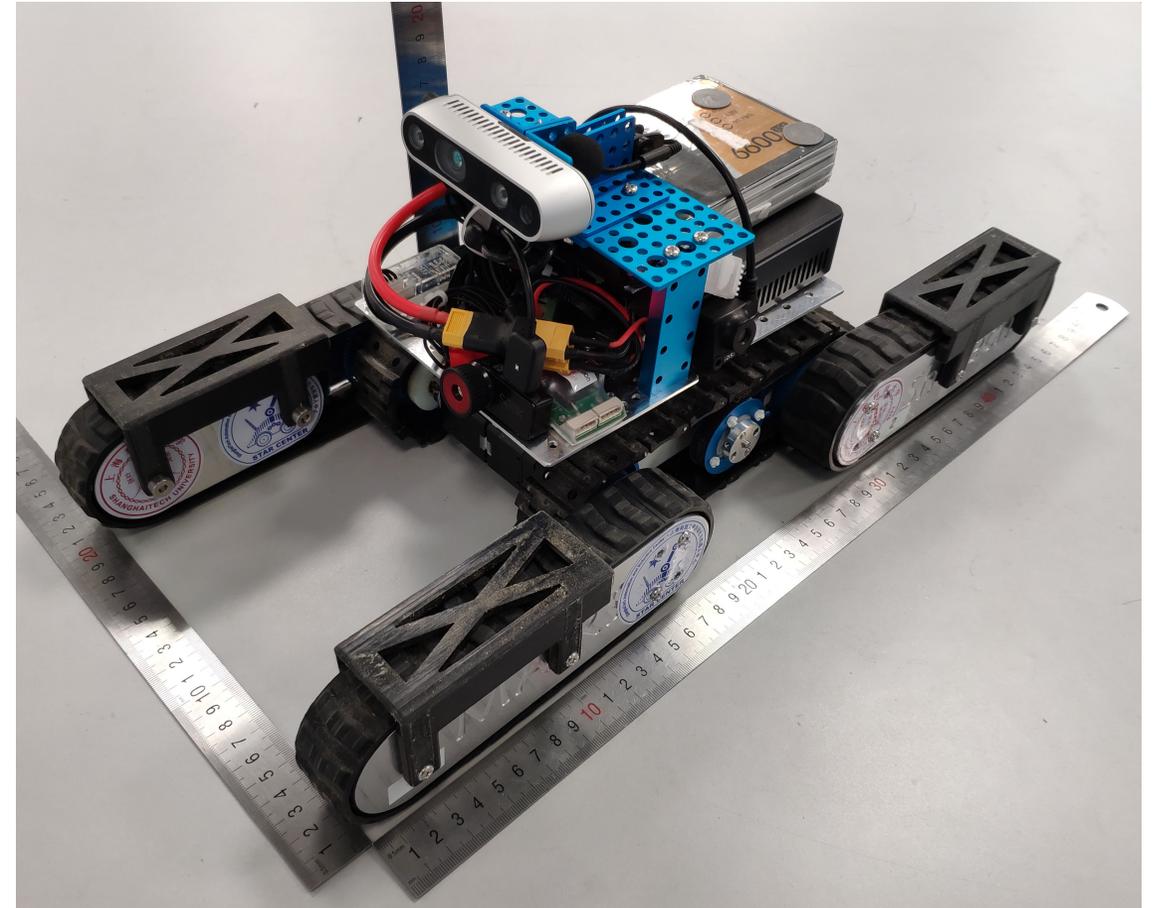
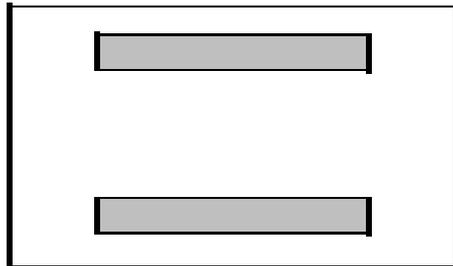
Right



Clockwise

# MARS Rescue Robot: Tracked Differential Drive

- Kinematic Simplification:
  - 2 Wheels, located at the center

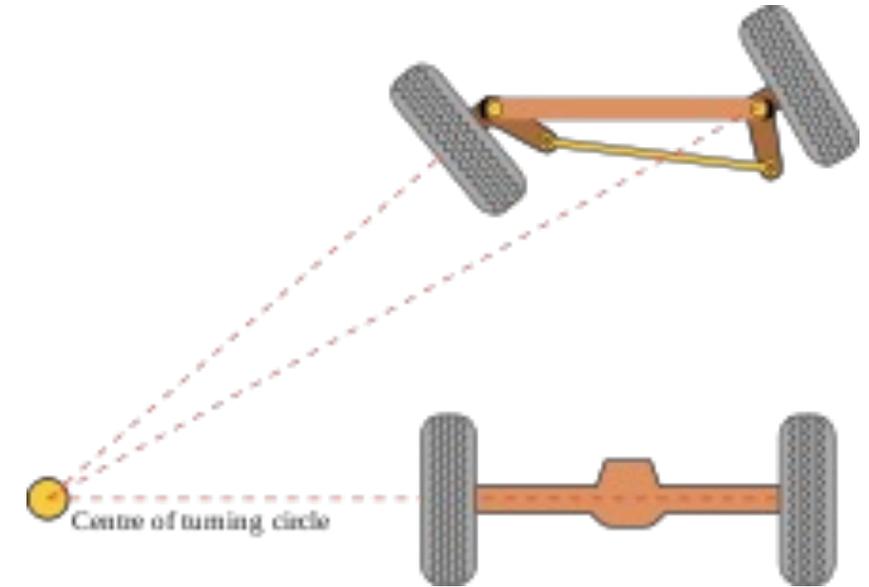


# Differential Drive Robots



# Ackermann Robot

- No sideways slip than differential drive during turning 😊
- Cannot turn on the spot 😞



# Introduction: Mobile Robot Kinematics

- Aim
  - Description of mechanical behavior of the robot for *design* and *control*
  - Similar to robot manipulator kinematics
  - However, mobile robots can move unbound with respect to its environment
    - there is no direct way to measure the robot's position
    - Position must be integrated over time
    - Leads to inaccuracies of the position (motion) estimate
      - > *the number 1 challenge in mobile robotics*

# Kinematics vs. Kinetics

## Kinematics:

- ▶ Greek origin: “motion”, “moving”
- ▶ Describes motion of points and bodies
- ▶ Considers position, velocity, acceleration, ..
- ▶ Examples: Celestial bodies, particle systems, robotic arm, human skeleton

## Kinetics:

- ▶ Describes causes of motion
- ▶ Effects of forces/moments
- ▶ Newton’s laws, e.g.,  $F = ma$

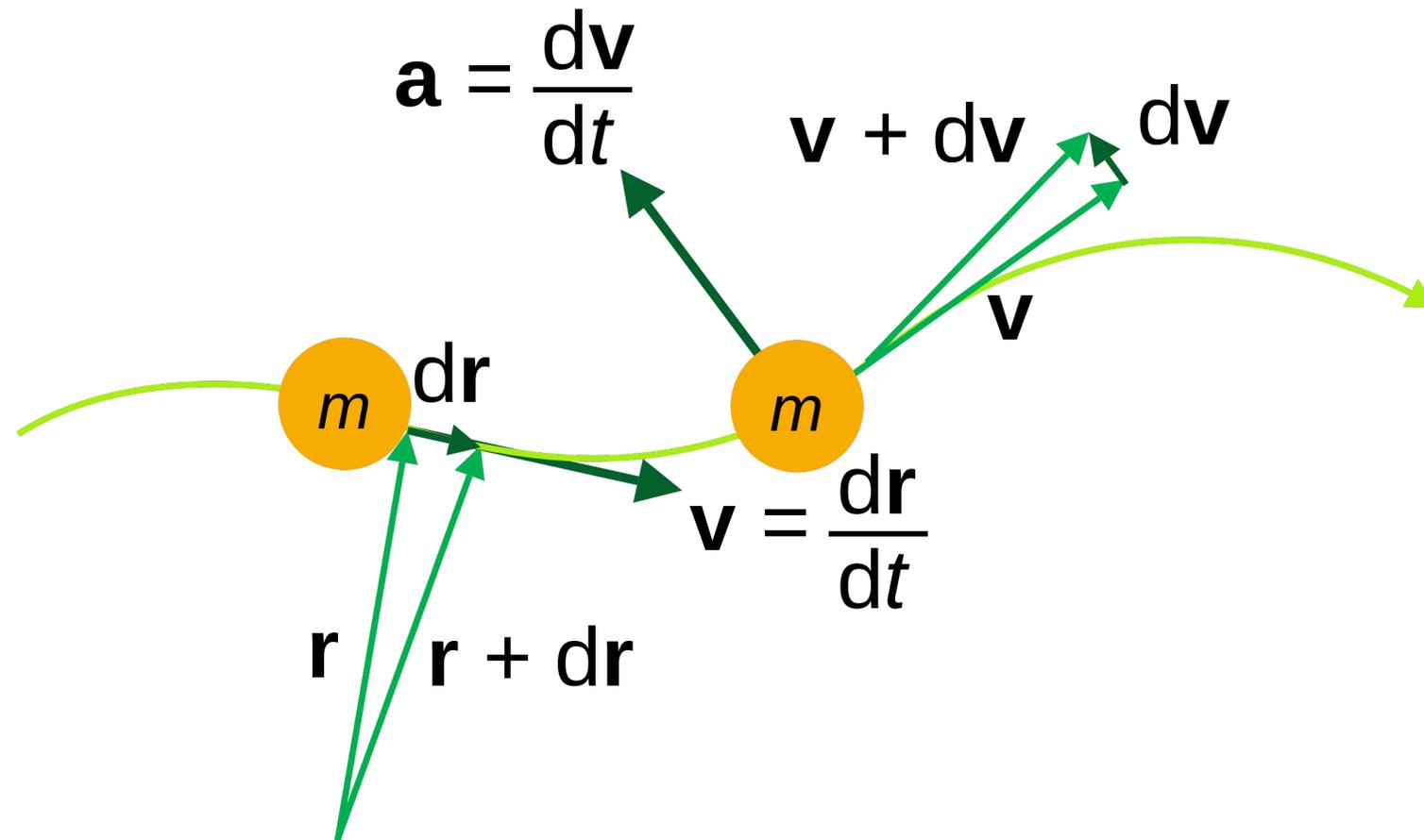
Kinematics and Control Slides:  
Andreas Geiger

<https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/self-driving-cars/>

# What are kinematics?

- Describes the motion of points, bodies (objects), and systems of objects
  - Does not consider the forces that cause them (that would be kinetics)
  - Also known as “the geometry of motion”
- For robotics:
  - Describes the motion of the vehicle
  - Puts position/orientation in relation with translational/angular velocities and accelerations
  - Used for regularization, prediction, etc.

# What are kinematics?



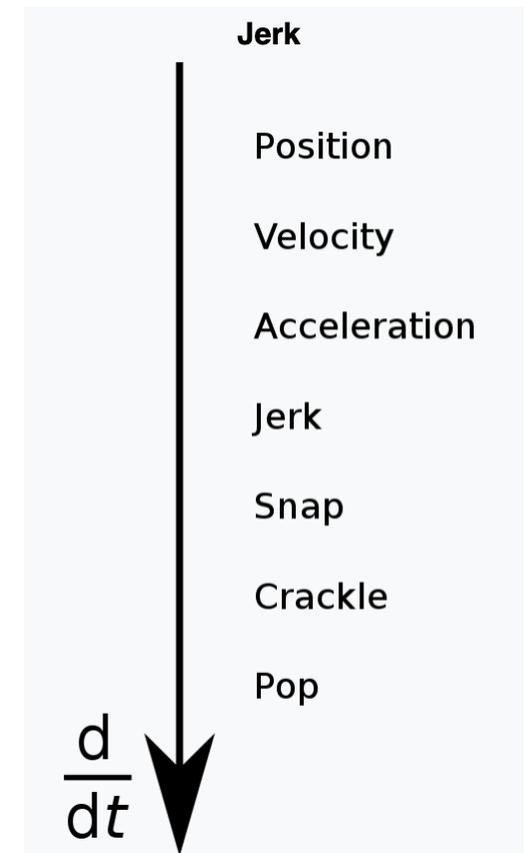
# What are kinematics?

- It does not stop at acceleration, but theory involves an arbitrarily high number of derivatives:

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t + \frac{1}{2} \mathbf{a}_0 t^2 + \frac{1}{3} \mathbf{j} t^3 + \dots$$

$$\boldsymbol{\theta}(t) = \boldsymbol{\theta}_0 + \boldsymbol{\omega}_0 t + \frac{1}{2} \boldsymbol{\alpha}_0 t^2 + \frac{1}{3} \boldsymbol{\zeta} t^3 + \dots$$

Jerk equations: minimal setting for solutions showing chaotic behavior!



# In practice

- Often we use finite models to simplify/smoothify the system
  - Locally constant acceleration

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t + \frac{1}{2} \mathbf{a}_0 t^2$$

- Locally constant velocity

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}_0 t$$

# Why do we want to introduce kinematic models

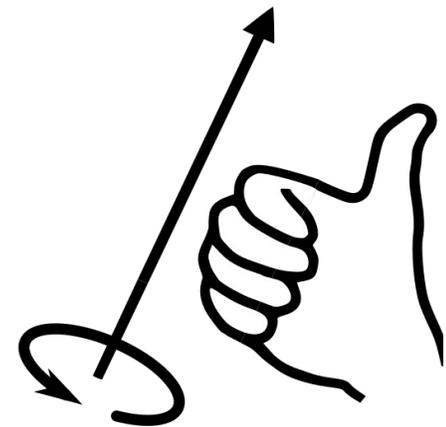
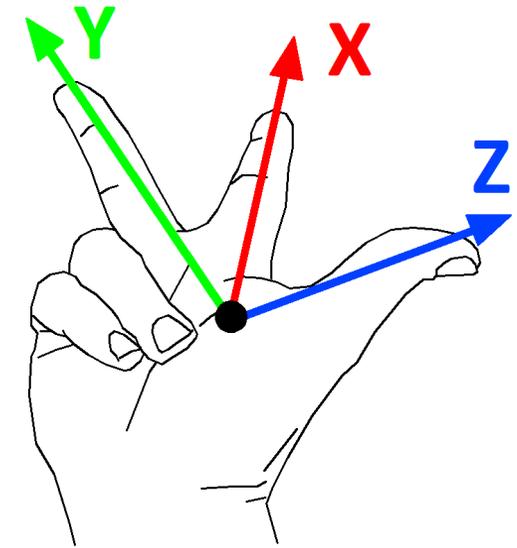
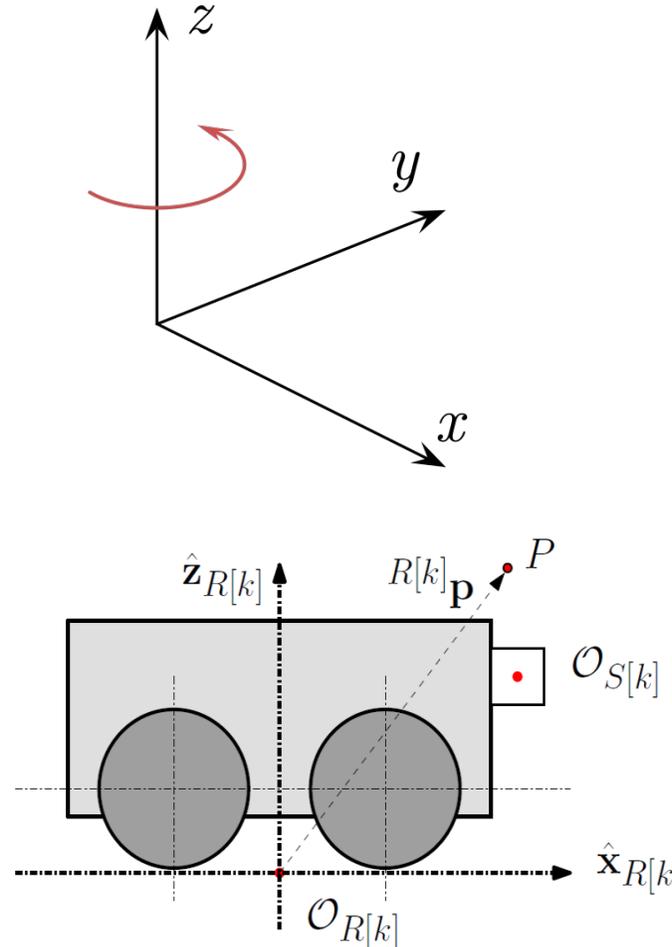
- For prediction
  - E.g.: If we have an initial estimate, we can use a kinematic model to generate a prior pose at a later point
- For smoothness
  - E.g.: If we estimate poses, we may constrain their difference to be consistent with some prior or measured velocity
- To impose constraints
  - E.g.: The motion may be more specific and include kinematic constraints
- For control
  - E.g.: Knowledge of how the system is moving is beneficial for reaching the goal pose

# COORDINATE SYSTEM

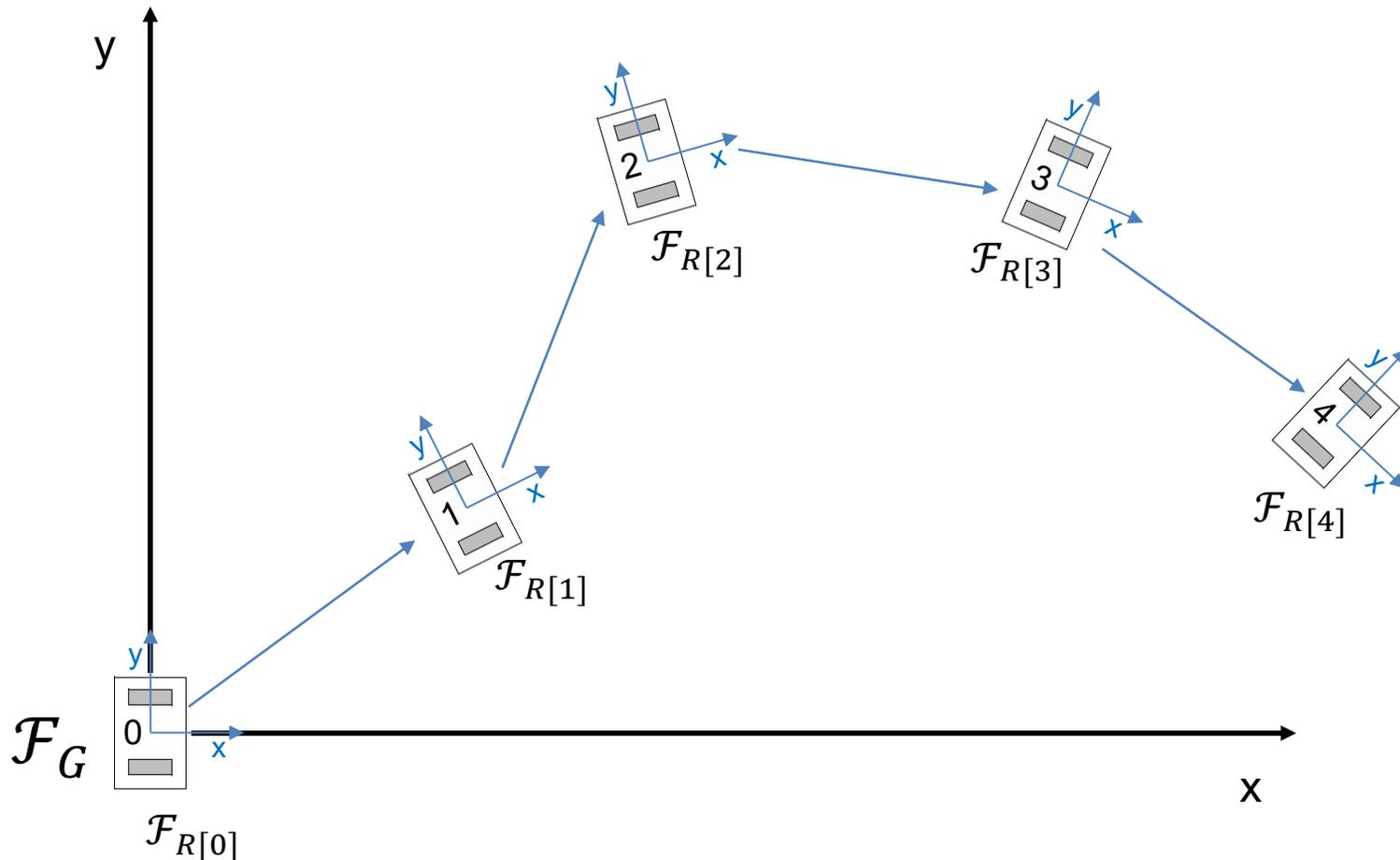
---

# Right Hand Coordinate System

- Standard in Robotics
- Positive rotation around X is anti-clockwise
- Right-hand rule mnemonic:
  - Thumb: z-axis
  - Index finger: x-axis
  - Second finger: y-axis
  - Rotation: Thumb = rotation axis, positive rotation in finger direction
- Robot Coordinate System:
  - X front
  - Z up (Underwater: Z down)
  - Y ???



# Odometry



With respect to the robot start pose:

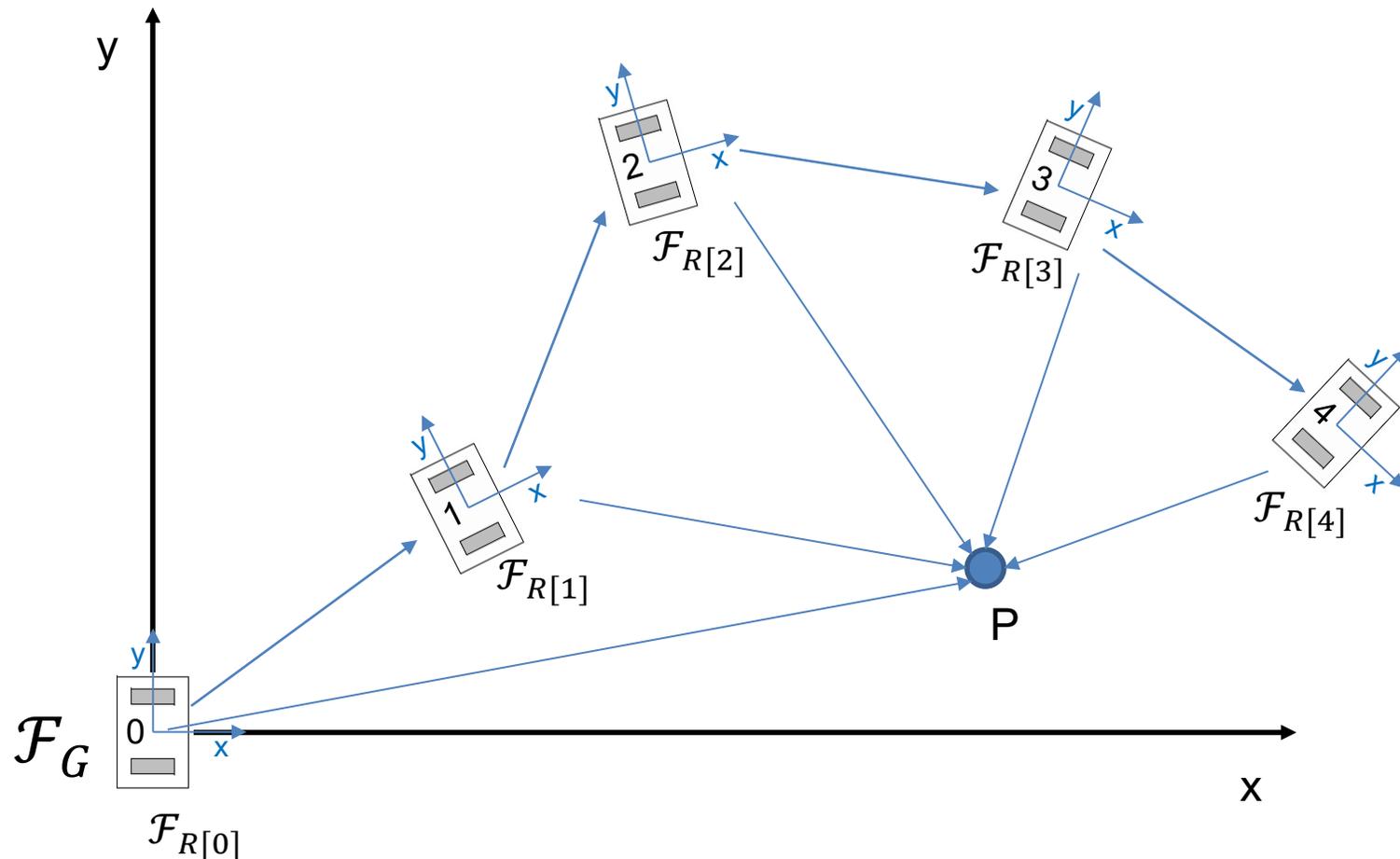
Where is the robot now?

Two approaches – same result:

- Geometry (easy in 2D)
- Transforms (better for 3D)

$\mathcal{F}_{R[X]}$  : The **F**rame of reference (the local coordinate system) of the **R**obot at the time **X**

# Use of robot frames $\mathcal{F}_{R[X]}$

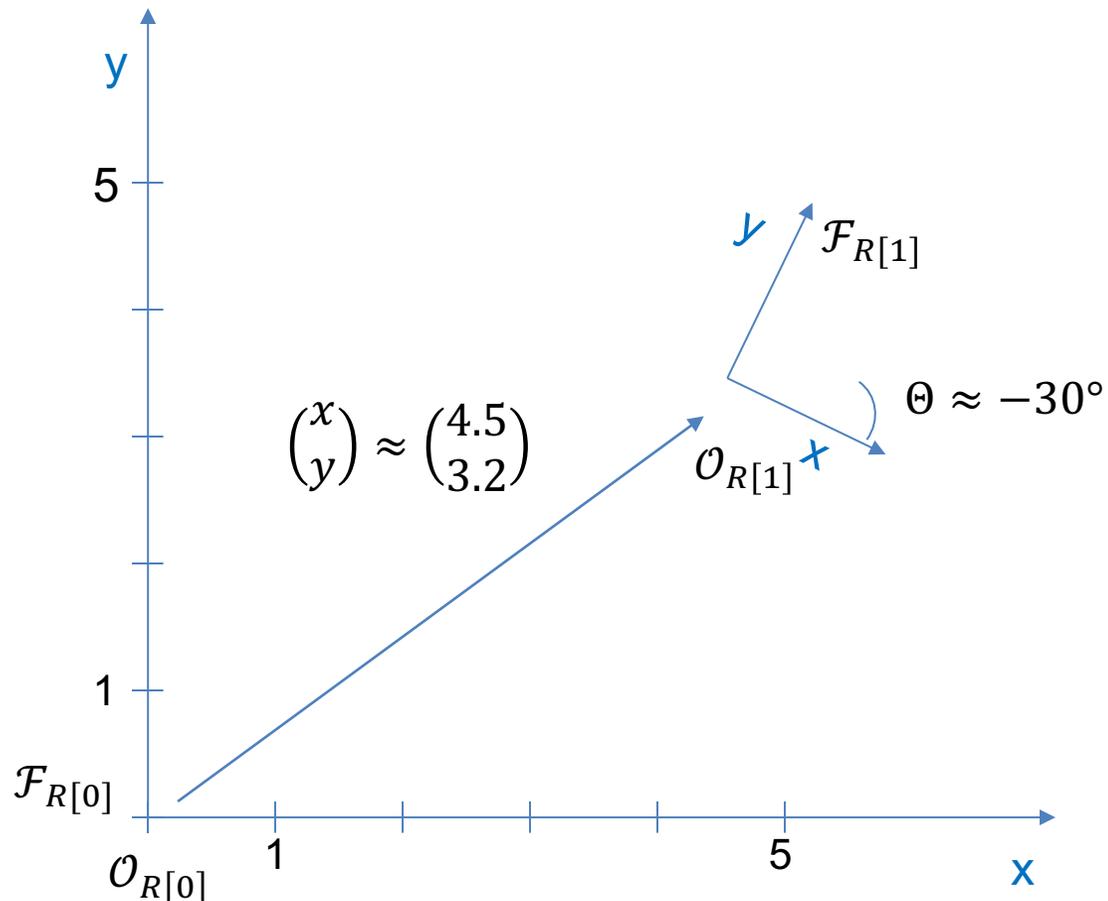


$\mathcal{O}_{R[X]}$  : Origin of  $\mathcal{F}_{R[X]}$   
(coordinates (0, 0))

$\overrightarrow{\mathcal{O}_{R[X]}P}$  : position vector from  $\mathcal{O}_{R[X]}$  to point P -  $\begin{pmatrix} x \\ y \end{pmatrix}$

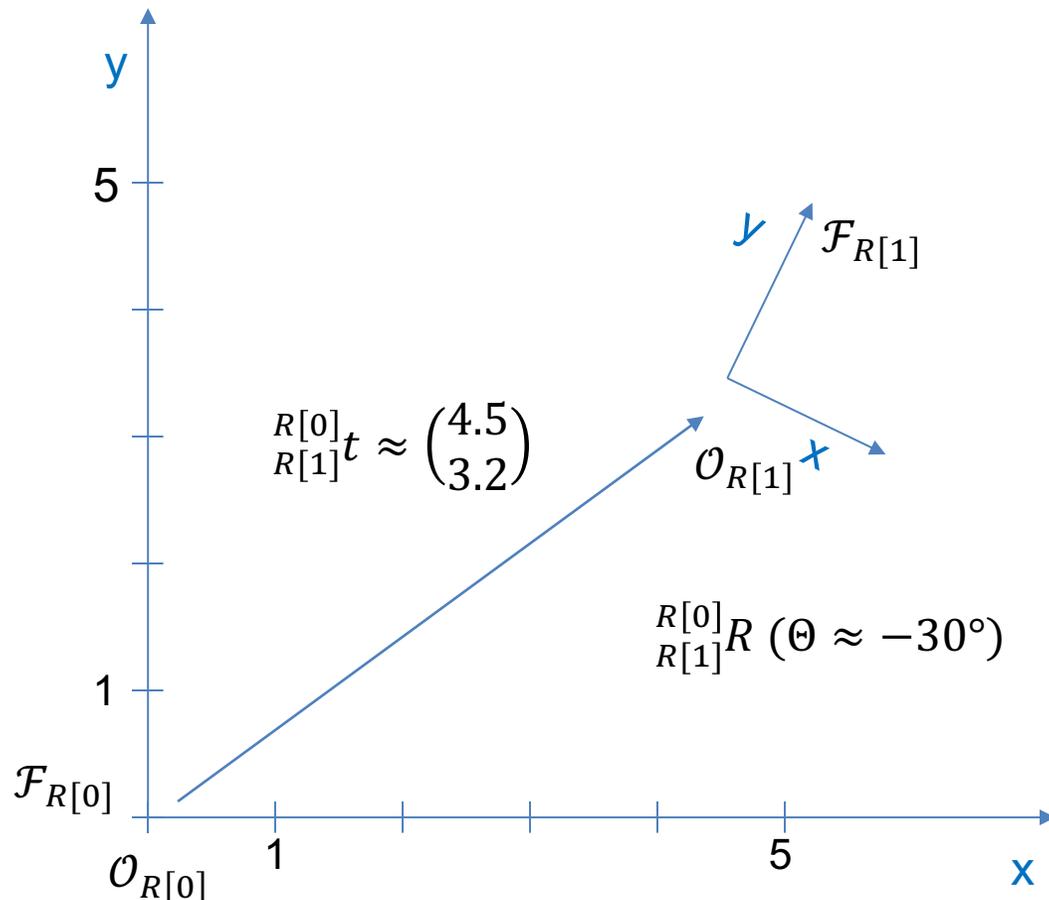
- Object P is observed at times 0 to 4
- Object P is static (does not move)
- The Robot moves (e.g.  $\mathcal{F}_{R[0]} \neq \mathcal{F}_{R[1]}$ )
- $\Rightarrow$  (x, y) coordinates of P are different in all frames, for example:
  - $\overrightarrow{\mathcal{O}_{R[0]}P} \neq \overrightarrow{\mathcal{O}_{R[1]}P}$

# Position, Orientation & Pose



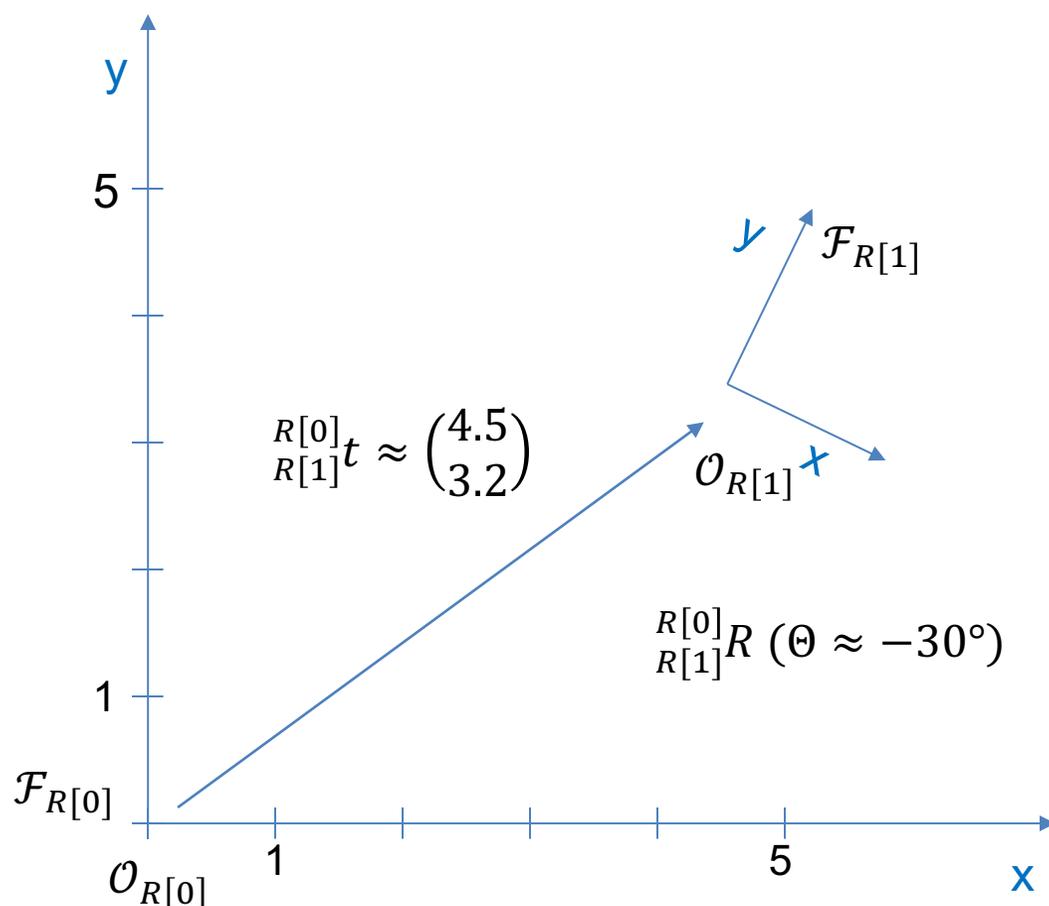
- **Position:**
  - $\begin{pmatrix} x \\ y \end{pmatrix}$  coordinates of any object or point (or another frame)
  - with respect to (wrt.) a specified frame
- **Orientation:**
  - $(\theta)$  angle of any oriented object (or another frame)
  - with respect to (wrt.) a specified frame
- **Pose:**
  - $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$  position and orientation of any oriented object
  - with respect to (wrt.) a specified frame

# Translation, Rotation & Transform



- **Translation:**
  - $\begin{pmatrix} x \\ y \end{pmatrix}$  difference, change, motion from one reference frame to another reference frame
- **Rotation:**
  - $(\Theta)$  difference in angle, rotation between one reference frame and another reference frame
- **Transform:**
  - $\begin{pmatrix} x \\ y \\ \Theta \end{pmatrix}$  difference, motion between one reference frame and another reference frame

# Position & Translation, Orientation & Rotation



- $\mathcal{F}_{R[X]}$  : Frame of reference of the robot at time X
- Where is that frame  $\mathcal{F}_{R[X]}$  ?
  - Can only be expressed with respect to (wrt.) another frame (e.g. global Frame  $\mathcal{F}_G$ ) =>
  - Pose of  $\mathcal{F}_{R[X]}$  wrt.  $\mathcal{F}_G$

- $O_{R[X]}$  : Origin of  $\mathcal{F}_{R[X]}$ 
  - $\overrightarrow{O_{R[X]}O_{R[X+1]}}$  : **Position** of  $\mathcal{F}_{R[X+1]}$  wrt.  $\mathcal{F}_{R[X]}$

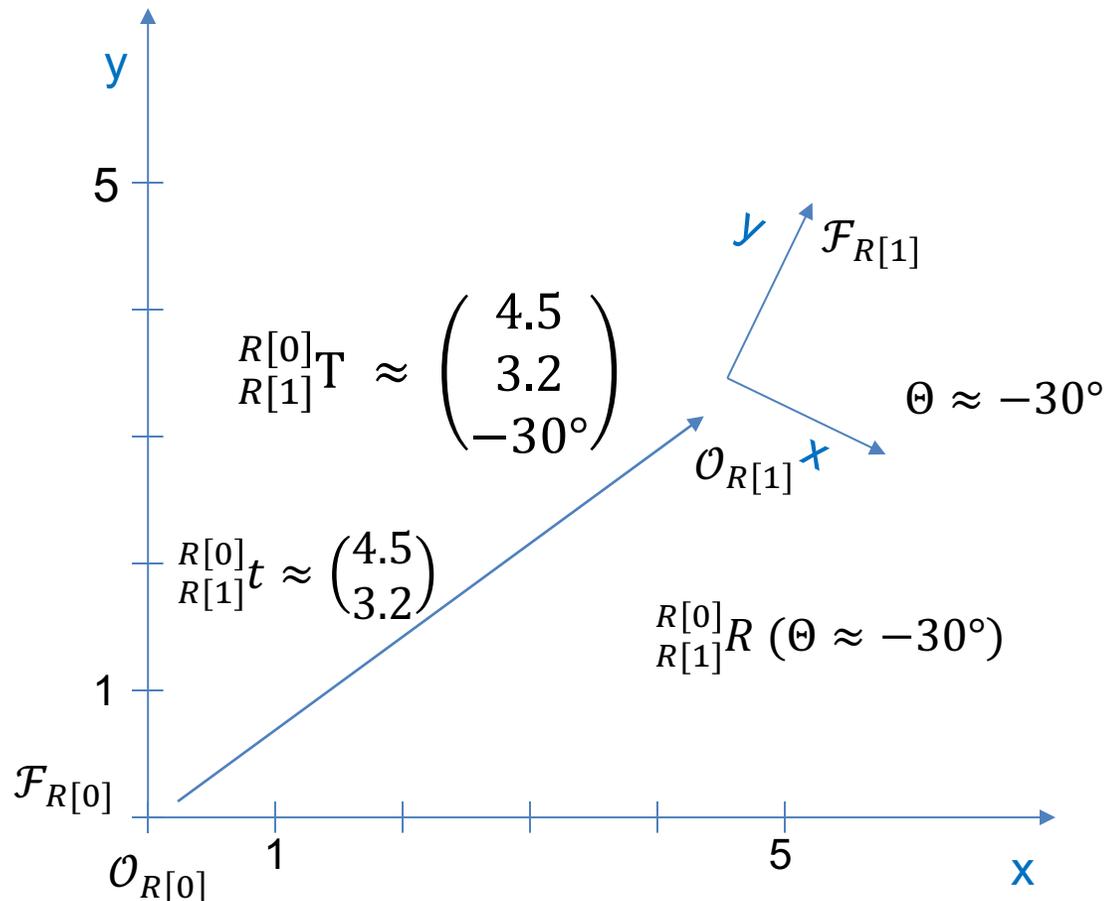
so  $O_{R[X+1]}$  wrt.  $\mathcal{F}_{R[X]}$

$\triangleq$   ${}^{R[X]}_{R[X+1]}t$  : **Translation**

- The angle  $\theta$  between the  $x$ -Axes:
  - **Orientation** of  $\mathcal{F}_{R[X+1]}$  wrt.  $\mathcal{F}_{R[X]}$

$\triangleq$   ${}^{R[X]}_{R[X+1]}R$  : **Rotation** of  $\mathcal{F}_{R[X+1]}$  wrt.  $\mathcal{F}_{R[X]}$

# Transform



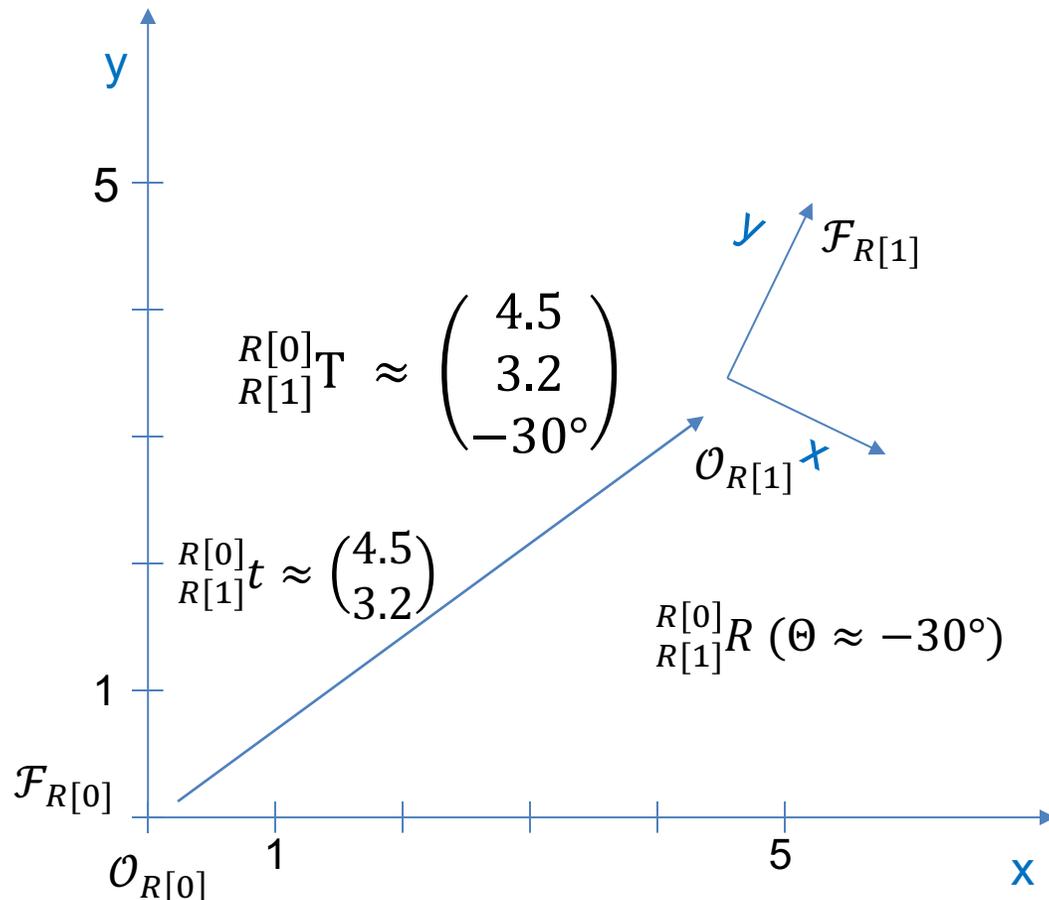
- $\begin{matrix} R[X] \\ R[X+1] \end{matrix} \mathbf{t}$  : **Translation**
  - Position vector  $(x, y)$  of  $R[X + 1]$  wrt.  $R[X]$
- $\begin{matrix} R[X] \\ R[X+1] \end{matrix} \mathbf{R}$  : **Rotation**
  - Angle  $(\theta)$  of  $R[X + 1]$  wrt.  $R[X]$
- **Transform:**  $\begin{matrix} R[X] \\ R[X+1] \end{matrix} \mathbf{T} \equiv \left\{ \begin{matrix} R[X] \\ R[X+1] \end{matrix} \mathbf{t} \\ \begin{matrix} R[X] \\ R[X+1] \end{matrix} \mathbf{R} \right\}$

# Geometry approach to Odometry

We want to know:

- Position of the robot ( $x, y$ )
- Orientation of the robot ( $\theta$ )
- => together: Pose  $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$

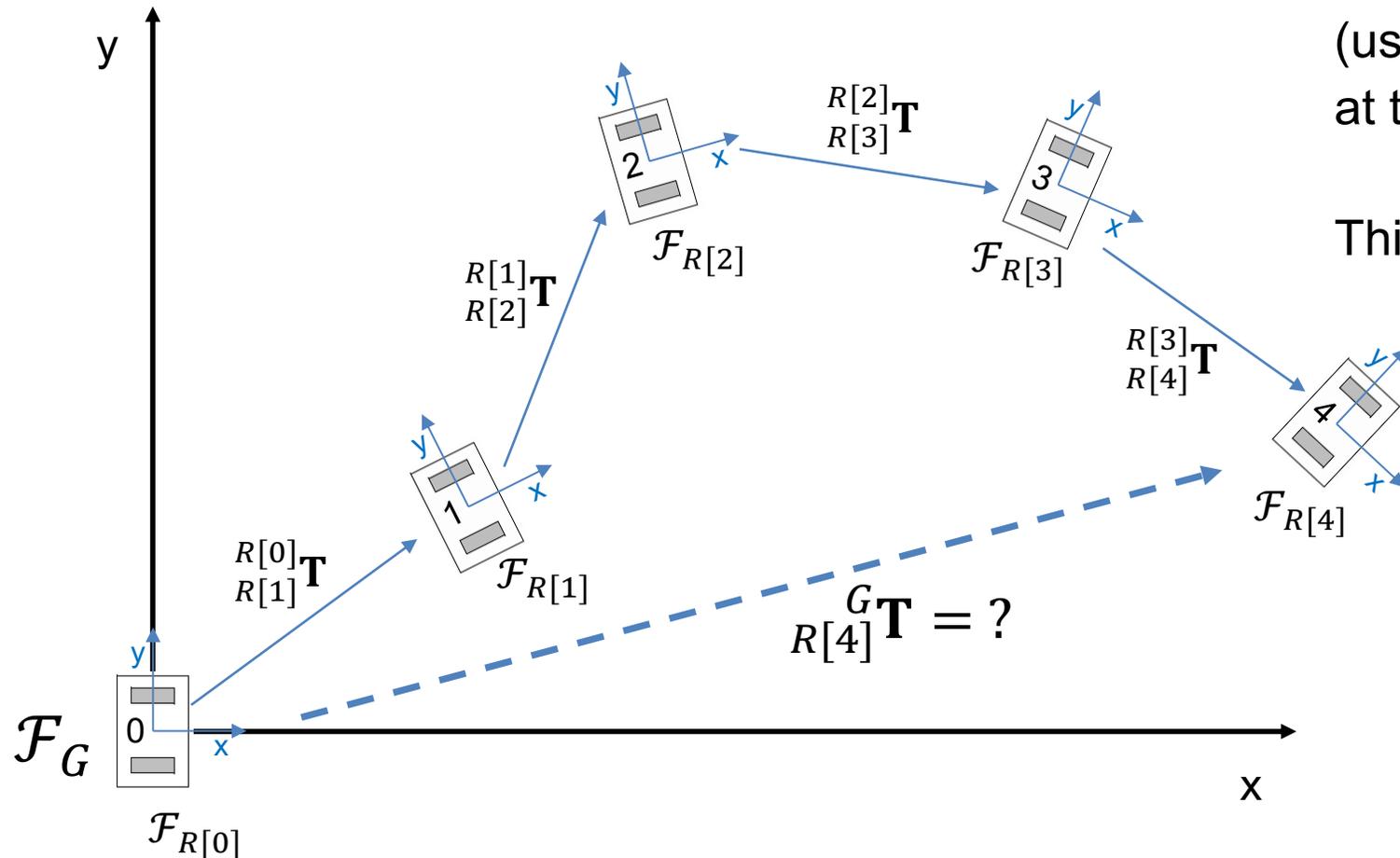
With respect to (wrt.)  $\mathcal{F}_G$  : The global frame; global coordinate system



$$\mathcal{F}_{R[0]} = \mathcal{F}_G \Rightarrow {}^G \mathcal{F}_{R[0]} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$${}^G \mathcal{F}_{R[1]} = R_{R[1]}^{R[0]} T \approx \begin{pmatrix} 4.5 \\ 3.2 \\ 30^\circ \end{pmatrix}$$

# Mathematical approach: Transforms



## Where is the Robot now?

The pose of  $\mathcal{F}_{R[X]}$  with respect to  $\mathcal{F}_G$  (usually =  $\mathcal{F}_{R[0]}$ ) is the pose of the robot at time X.

This is equivalent to  ${}^G R[X] \mathbf{T}$

## Chaining of Transforms

$${}^G R[X+1] \mathbf{T} = {}^G R[X] \mathbf{T} {}^{R[X]} R[X+1] \mathbf{T}$$

often:  $\mathcal{F}_G \equiv \mathcal{F}_{R[0]} \Rightarrow {}^G R[0] \mathbf{T} = id$

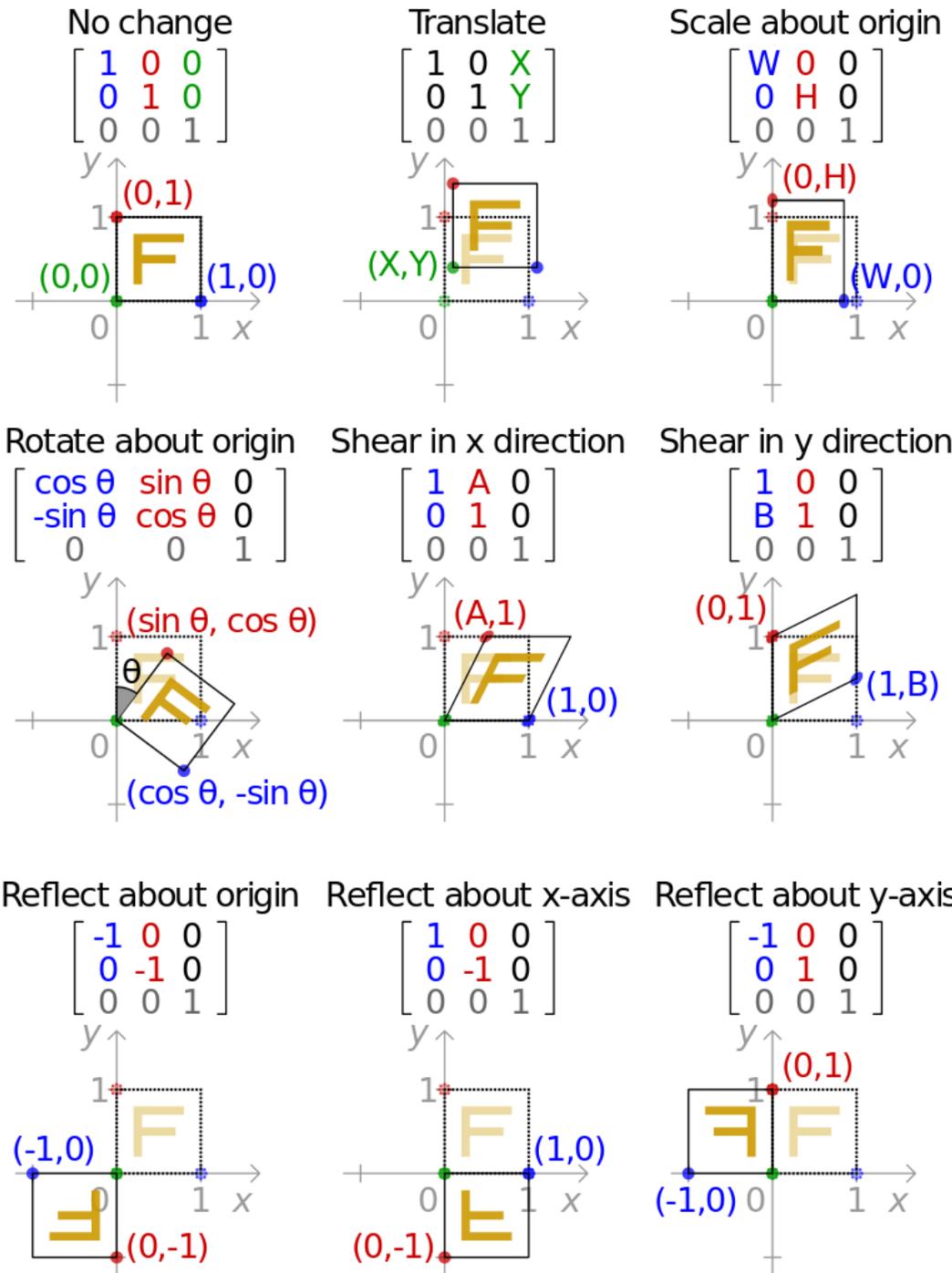
# TRANSFORMS & STUFF 😊

---

# Affine Transformation

- Function between affine spaces. Preserves:
  - points,
  - straight lines
  - planes
  - sets of parallel lines remain parallel
- Allows:
  - Interesting for Robotics: translation, rotation, (scaling), and chaining of those
  - Not so interesting for Robotics: reflection, shearing, homothetic transforms

- Rotation and Translation: 
$$\begin{bmatrix} \cos \theta & \sin \theta & X \\ -\sin \theta & \cos \theta & Y \\ 0 & 0 & 1 \end{bmatrix}$$



# Math: Rigid Transformation

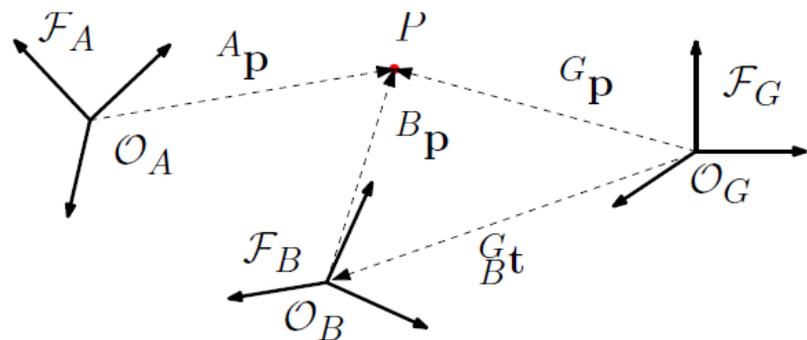
- Geometric transformation that preserves Euclidean distance between pairs of points.
- Includes reflections (i.e. change from right-hand to left-hand coordinate system and back)
- Just rotation & translation: rigid motions or proper rigid transformations:
  - Decomposed to rotation and translation
  - => subset of Affine Transformations
- In Robotics: Just use term **Transform** or **Transformation** for rigid motions (without reflections)

# Lie groups for transformations

- Smoothly differentiable Group
- No singularities
- Good interpolation
- SO: Special Orthogonal group
- SE: Special Euclidian group
- Sim\_ilarity transform group

Group	Description	Dim.	Matrix Representation
SO(3)	3D Rotations	3	3D rotation matrix
SE(3)	3D Rigid transformations	6	Linear transformation on homogeneous 4-vectors
SO(2)	2D Rotations	1	2D rotation matrix
SE(2)	2D Rigid transformations	3	Linear transformation on homogeneous 3-vectors
Sim(3)	3D Similarity transformations (rigid motion + scale)	7	Linear transformation on homogeneous 4-vectors

# Transform



Notation	Meaning
$\mathcal{F}_{R[k]}$	Coordinate frame attached to object 'R' (usually the robot) at sample time-instant $k$ .
$O_{R[k]}$	Origin of $\mathcal{F}_{R[k]}$ .
${}^{R[k]}p$	For any general point $P$ , the position vector $\overrightarrow{O_{R[k]}P}$ resolved in $\mathcal{F}_{R[k]}$ .
${}^H\hat{x}_R$	The x-axis direction of $\mathcal{F}_R$ resolved in $\mathcal{F}_H$ . Similarly, ${}^H\hat{y}_R$ , ${}^H\hat{z}_R$ can be defined. Obviously, ${}^R\hat{x}_R = \hat{e}_1$ . Time indices can be added to the frames, if necessary.
${}^{R[k]}S[k']\mathbf{R}$	The rotation-matrix of $\mathcal{F}_{S[k']}$ with respect to $\mathcal{F}_{R[k]}$ .
${}^R_S t$	The translation vector $\overrightarrow{O_R O_S}$ resolved in $\mathcal{F}_R$ .

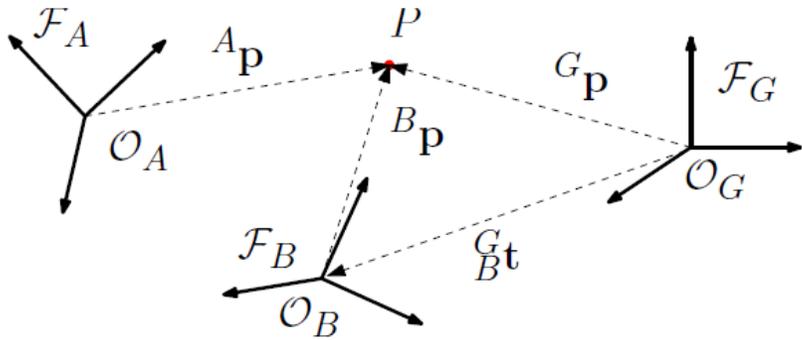
Transform  
between two  
coordinate frames

$${}^G_A t \triangleq \overrightarrow{O_G O_A} \text{ resolved in } \mathcal{F}_G \quad \begin{pmatrix} {}^G p \\ 1 \end{pmatrix} \equiv \begin{pmatrix} {}^G_A \mathbf{R} & {}^G_A t \\ \mathbf{0}_{1 \times [2,3]} & 1 \end{pmatrix} \begin{pmatrix} {}^A p \\ 1 \end{pmatrix} \quad {}^G_A \mathbf{T} \equiv \left\{ \begin{matrix} {}^G_A t \\ {}^G_A \mathbf{R} \end{matrix} \right\}$$

$$\begin{aligned} {}^G p &= {}^G_A \mathbf{R} \, {}^A p + {}^G_A t \\ &\triangleq {}^G_A \mathbf{T} ({}^A p). \end{aligned}$$

$$\begin{bmatrix} \cos \theta & -\sin \theta & {}^G_A t_x \\ \sin \theta & \cos \theta & {}^G_A t_y \\ 0 & 0 & 1 \end{bmatrix}$$

# Transform: Operations



Transform between two coordinate frames (chaining, compounding):

$${}^G\mathbf{T} = {}^G\mathbf{T} {}^A\mathbf{T} \equiv \begin{Bmatrix} {}^G\mathbf{R}^A \mathbf{t} + {}^G\mathbf{t} \\ {}^G\mathbf{R}^A \end{Bmatrix}$$

Inverse of a Transform :

$${}^B\mathbf{T} = {}^A\mathbf{T}^{-1} \equiv \begin{Bmatrix} -{}^A\mathbf{R}^B \mathbf{t} \\ {}^A\mathbf{R}^B \end{Bmatrix}$$

Relative (Difference) Transform :  ${}^B\mathbf{T} = {}^G\mathbf{T}^{-1} {}^G\mathbf{T}$

See: **Quick Reference to Geometric Transforms in Robotics** by Kaustubh Pathak on the webpage!

**Chaining :** 
$${}_{R[X+1]}^G \mathbf{T} = {}_{R[X]}^G \mathbf{T} {}_{R[X+1]}^{R[X]} \mathbf{T} \equiv \begin{Bmatrix} {}_{R[X]}^G \mathbf{R} & {}_{R[X+1]}^{R[X]} \mathbf{t} + {}_{R[X]}^G \mathbf{t} \\ {}_{R[X]}^G \mathbf{R} & {}_{R[X+1]}^{R[X]} \mathbf{R} \end{Bmatrix} = \begin{Bmatrix} {}_{R[X+1]}^G \mathbf{t} \\ {}_{R[X+1]}^G \mathbf{R} \end{Bmatrix}$$

**In 2D Translation:** 
$$\begin{bmatrix} {}_{R[X+1]}^G t_x \\ {}_{R[X+1]}^G t_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos {}_{R[X]}^G \theta & -\sin {}_{R[X]}^G \theta & {}_{R[X]}^G t_x \\ \sin {}_{R[X]}^G \theta & \cos {}_{R[X]}^G \theta & {}_{R[X]}^G t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}_{R[X+1]}^{R[X]} t_x \\ {}_{R[X+1]}^{R[X]} t_y \\ 1 \end{bmatrix}$$

**In 2D Rotation:**

$${}_{R[X+1]}^G \mathbf{R} = \begin{bmatrix} \cos {}_{R[X+1]}^G \theta & -\sin {}_{R[X+1]}^G \theta \\ \sin {}_{R[X+1]}^G \theta & \cos {}_{R[X+1]}^G \theta \end{bmatrix} = \begin{bmatrix} \cos {}_{R[X]}^G \theta & -\sin {}_{R[X]}^G \theta \\ \sin {}_{R[X]}^G \theta & \cos {}_{R[X]}^G \theta \end{bmatrix} \begin{bmatrix} \cos {}_{R[X+1]}^{R[X]} \theta & -\sin {}_{R[X+1]}^{R[X]} \theta \\ \sin {}_{R[X+1]}^{R[X]} \theta & \cos {}_{R[X+1]}^{R[X]} \theta \end{bmatrix}$$

**In 2D Rotation (simple):** 
$${}_{R[X+1]}^G \theta = {}_{R[X]}^G \theta + {}_{R[X+1]}^{R[X]} \theta$$

# In ROS: nav\_2d\_msgs/Pose2DStamped

- First Message at time 97 : G
- Message at time 103 : X
- Next Message at time 107 : X+1

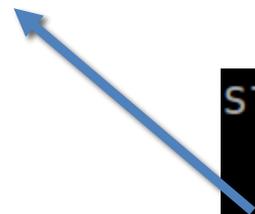
$$R_{[X+1]}^G \mathbf{T} = R_{[X]}^G \mathbf{T} R_{[X+1]}^{R[X]} \mathbf{T}$$

$$R_{[X]} \mathbf{T} \begin{matrix} t_x \\ t_y \end{matrix}$$

$$R_{[X+1]} \mathbf{T} \begin{matrix} t_x \\ t_y \end{matrix}$$

$$R_{[X]} \Theta$$

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose2D pose2D
  float64 x
  float64 y
  float64 theta
```

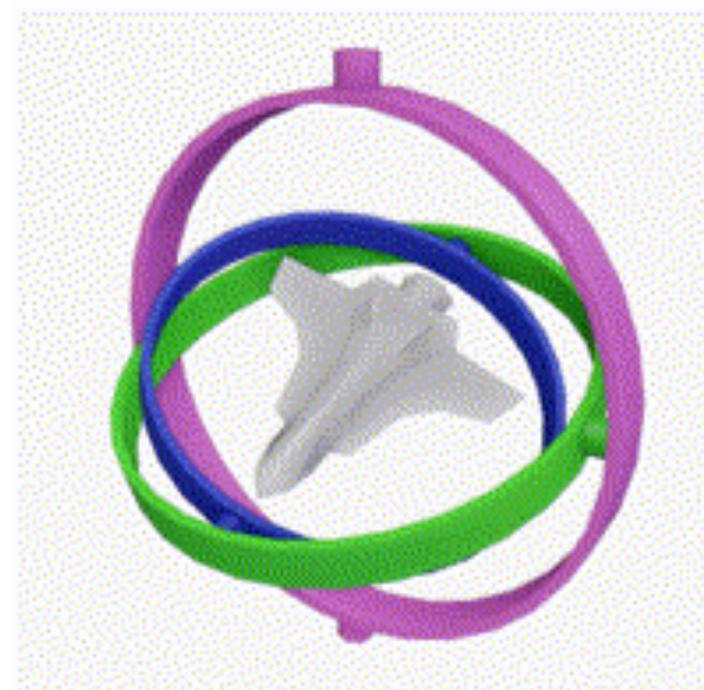
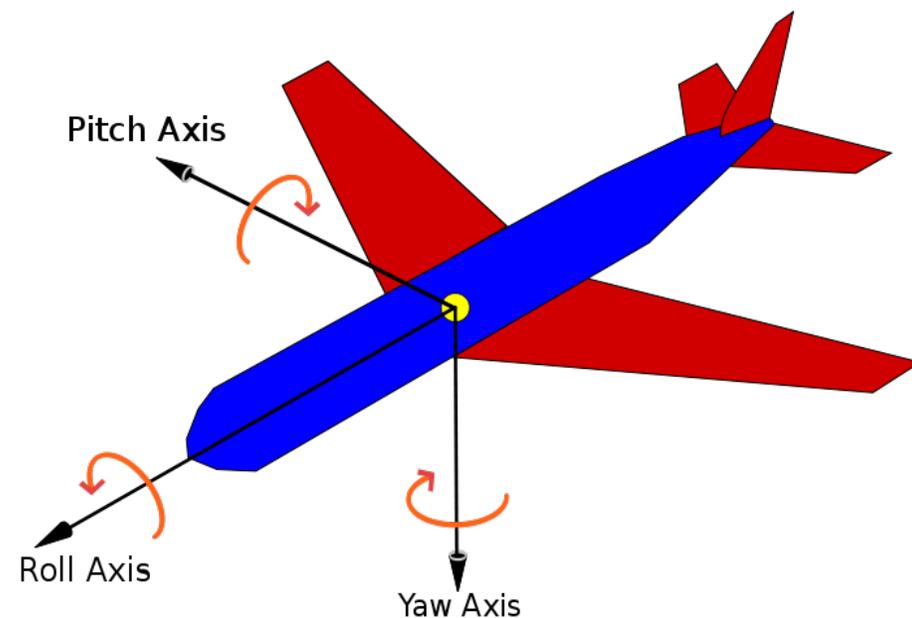


# 3D Rotation

- Many 3D rotation representations:

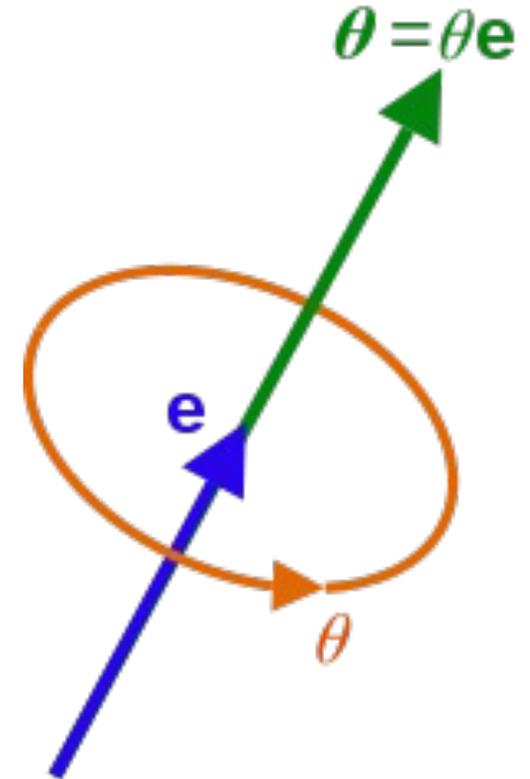
[https://en.wikipedia.org/wiki/Rotation\\_formalisms\\_in\\_three\\_dimensions](https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions)

- Euler angles:
  - Roll: rotation around x-axis
  - Pitch: rotation around y-axis
  - Yaw: rotation around z-axis
  - Apply rotations one after the other...
    - => Order important! E.g.:
      - x-z-x; x-y-z; z-y-x; ...
  - ☹ Singularities
  - Gimbal lock in Engineering
    - "a condition caused by the collinear alignment of two or more robot axes resulting in unpredictable robot motion and velocities"



# 3D Rotation

- Axis Angle
  - Angle  $\theta$  and
  - Axis unit vector  $\mathbf{e}$  (3D vector with length 1)
    - Can be represented with 2 numbers (e.g. elevation and azimuth angles)
- Euler Angles: sequence of 3 rotations around coordinate axes  
equivalent to:
- Axis Angle: pure rotation around a single fixed axis



# 3D Rotation

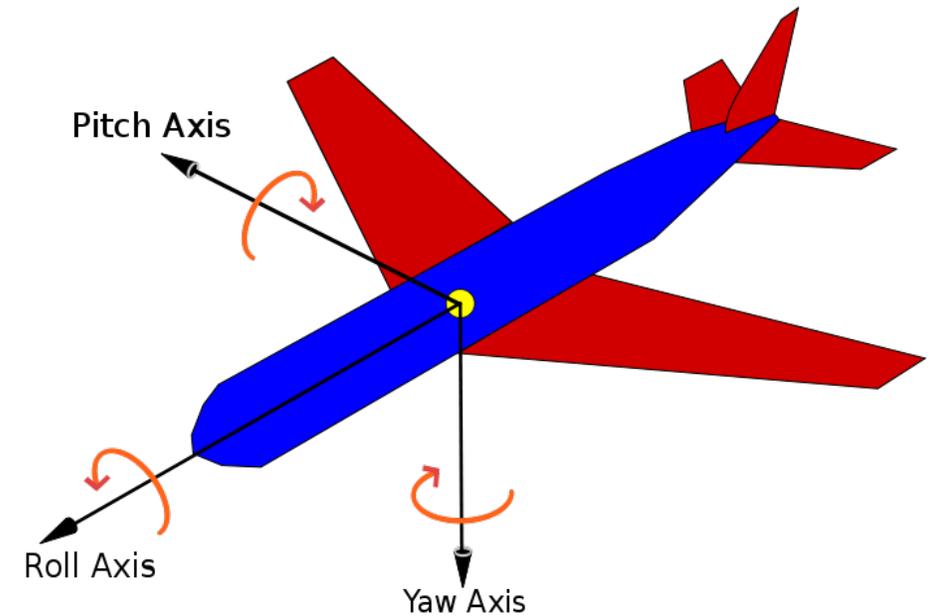
- Quaternions:
  - Concatenating rotations is computationally faster and numerically more stable
  - Extracting the angle and axis of rotation is simpler
  - Interpolation is more straightforward
  - Unit Quaternion: norm = 1

- Versor: <https://en.wikipedia.org/wiki/Versor>

[https://en.wikipedia.org/wiki/Quaternions\\_and\\_spatial\\_rotation](https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation)

- Scalar (real) part:  $q_0$  , sometimes  $q_w$
- Vector (imaginary) part:  $\mathbf{q}$
- Over determined: 4 variables for 3 DoF (but: unit!)

- Check out: <https://eater.net/quaternions> !  
Excellent interactive video...



$$\check{\mathbf{p}} \equiv p_0 + p_x \mathbf{i} + p_y \mathbf{j} + p_z \mathbf{k}$$

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -\mathbf{1}$$

$$\check{\mathbf{q}} = (q_0 \quad q_x \quad q_y \quad q_z)^T \equiv \begin{pmatrix} q_0 \\ \mathbf{q} \end{pmatrix}$$

# Transform in 3D

$${}^G\mathbf{T}_A = \begin{matrix} & \text{Matrix} & \text{Euler} & \text{Quaternion} \\ = & \begin{bmatrix} {}^G\mathbf{R}_A & {}^G\mathbf{t}_A \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} & \begin{pmatrix} {}^G\mathbf{t}_A \\ {}^G\Theta_A \end{pmatrix} & \begin{pmatrix} {}^G\mathbf{t}_A \\ {}^G\check{\mathbf{q}}_A \end{pmatrix} \end{matrix}$$

$${}^G\Theta_A \triangleq (\theta_r, \theta_p, \theta_y)^T$$

In ROS: Quaternions! (w, x, y, z)  
Uses Eigen library for Transforms

## Rotation Matrix 3x3

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

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

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

$$R = R_z(\alpha) R_y(\beta) R_x(\gamma)$$

yaw =  $\alpha$ , pitch =  $\beta$ , roll =  $\gamma$

# Eigen

- Don't have to deal with the details of transforms too much 😊
- Conversions between ROS and Eigen:

[https://docs.ros2.org/foxy/api/tf2\\_eigen/namespaces.html](https://docs.ros2.org/foxy/api/tf2_eigen/namespaces.html)

```
Matrix3f m;
m = AngleAxisf(angle1, Vector3f::UnitZ())
    * AngleAxisf(angle2, Vector3f::UnitY())
    * AngleAxisf(angle3, Vector3f::UnitZ());
```

[https://eigen.tuxfamily.org/dox/group\\_Geometry\\_Module.html](https://eigen.tuxfamily.org/dox/group_Geometry_Module.html)

	void	<code>getEulerYPR (const A &amp;a, double &amp;yaw, double &amp;pitch, double &amp;roll)</code>
	double	<code>getYaw (const A &amp;a)</code>
	A	<code>getTransformIdentity ()</code>
Eigen::Isometry3d		<code>transformToEigen (const geometry_msgs::msg::Transform &amp;t)</code> Convert a timestamped transform to the equivalent <b>Eigen</b> data
Eigen::Isometry3d		<code>transformToEigen (const geometry_msgs::msg::TransformStamped &amp;t)</code> Convert a timestamped transform to the equivalent <b>Eigen</b> data
geometry_msgs::msg::TransformStamped		<code>eigenToTransform (const Eigen::Affine3d &amp;t)</code> Convert an <b>Eigen</b> Affine3d transform to the equivalent geometry_msgs::msg::TransformStamped
geometry_msgs::msg::TransformStamped		<code>eigenToTransform (const Eigen::Isometry3d &amp;t)</code> Convert an <b>Eigen</b> Isometry3d transform to the equivalent geometry_msgs::msg::TransformStamped
	template<>	
	void	<code>doTransform (const Eigen::Vector3d &amp;t_in, Eigen::Vector3d &amp;t_out)</code> Apply a geometry_msgs TransformStamped to an Eigen-specific Vector3d type used in tf2_ros::BufferInterface::transform because this function is not templated
geometry_msgs::msg::Point		<code>toMsg (const Eigen::Vector3d &amp;in)</code> Convert a <b>Eigen</b> Vector3d type to a Point message. This function is not templated
	void	<code>fromMsg (const geometry_msgs::msg::Point &amp;msg, Eigen::Vector3d &amp;out)</code> Convert a Point message type to a Eigen-specific Vector3d type
geometry_msgs::msg::Vector3		<code>toMsg (const Eigen::Vector3d &amp;in, geometry_msgs::msg::Vector3 &amp;out)</code> Convert an <b>Eigen</b> Vector3d type to a Vector3 message. This function is not templated
	void	<code>fromMsg (const geometry_msgs::msg::Vector3 &amp;msg, Eigen::Vector3d &amp;out)</code> Convert a Vector3 message type to a Eigen-specific Vector3d type
	template<>	
	void	<code>doTransform (const tf2::Stamped&lt; Eigen::Vector3d &gt; &amp;t_in, tf2::Stamped&lt; Eigen::Vector3d &gt; &amp;t_out)</code> Apply a geometry_msgs TransformStamped to an Eigen-specific Vector3d type
geometry_msgs::msg::PointStamped		<code>toMsg (const tf2::Stamped&lt; Eigen::Vector3d &gt; &amp;in)</code> Convert a stamped <b>Eigen</b> Vector3d type to a PointStamped message
	void	<code>fromMsg (const geometry_msgs::msg::PointStamped &amp;msg, tf2::Stamped&lt; Eigen::Vector3d &gt; &amp;out)</code> Convert a PointStamped message type to a stamped Eigen-specific Vector3d type