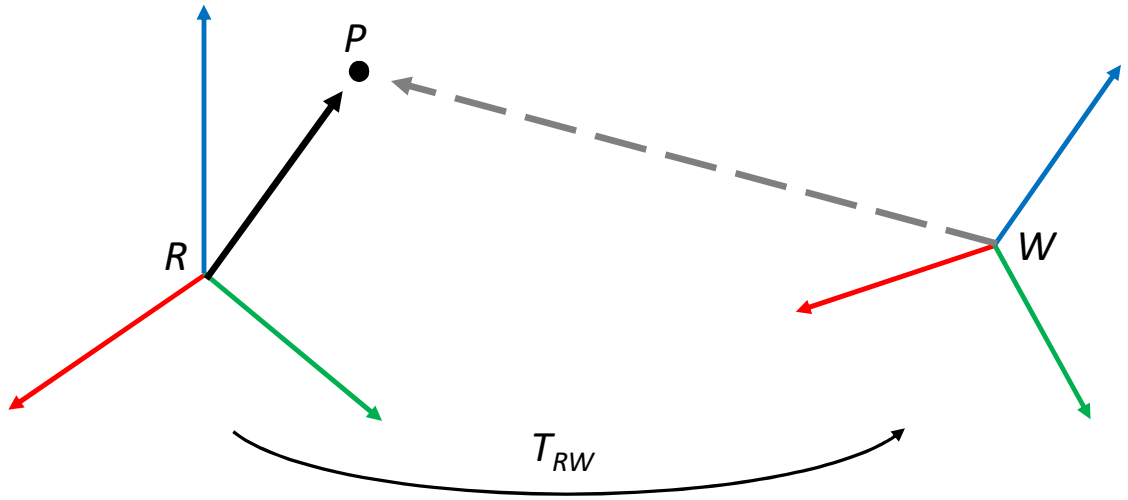
EECE 5550: Mobile Robotics



Lecture 1: Coordinate Systems and Geometry

Plan of the day

- Coordinate systems
- Coordinate transformations and transformation groups
- Rotation representations in 2 and 3 dimensions

Sources & references for this lecture:

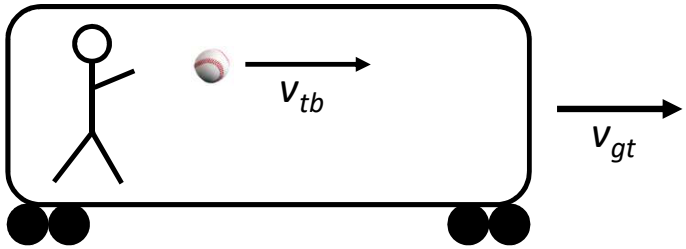
- Handbook of Robotics, Sec. 1.1-1.2
- “Math Fundamentals” – A. Kelly
- “Representing Robot Pose – The Good, the Bad, and the Ugly” – P. Furgale

Modeling geometric / physical relations

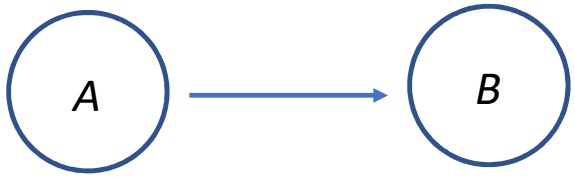
Key (deep!) insight: almost all physically-meaningful quantities are *relative*.

Examples

- **Position**: I am 2 m *in front of* the whiteboard
- **Velocity**: The velocity of the ball *with respect to* the train
- **Potential energy**: Gravitational potential *above ground level*



Implication: We are typically interested in modeling *geometric relations* of the form:



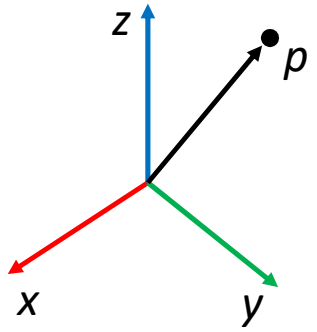
"Property P of object B with respect to object A"

Coordinate systems

Main idea: Coordinate systems enable us to model *geometry* using *algebra*

- We can represent geometric objects using algebraic ones.
  Ex: 3d points ⇔ 3-dimensional vectors
- We can model geometric operations using algebraic ones.
  Ex: translation ⇔ vector subtraction, rotation ⇔ rotation matrix multiplication
- Unlike geometry, computers (and robots) "natively" understand (at least basic) algebra ☺

$$\vec{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

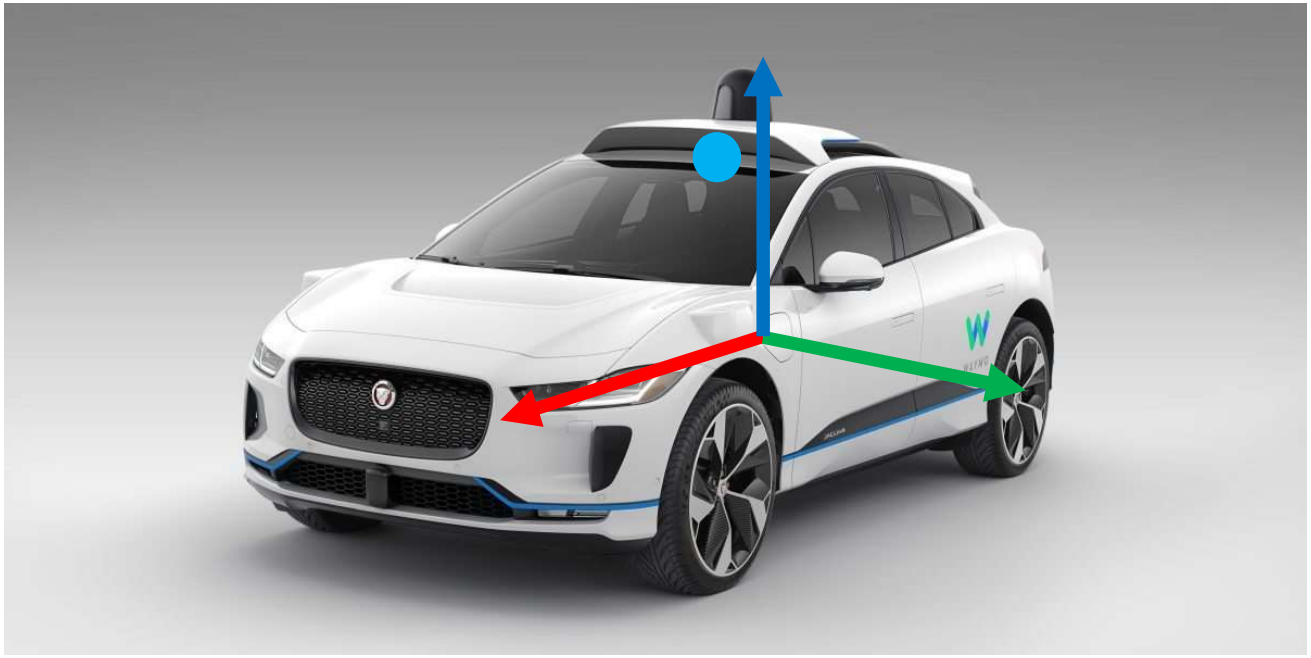


Coordinate systems: The Bane of All Roboticists

Question: Where is the point x = [1, 2.5, 2.5] with respect to the car?

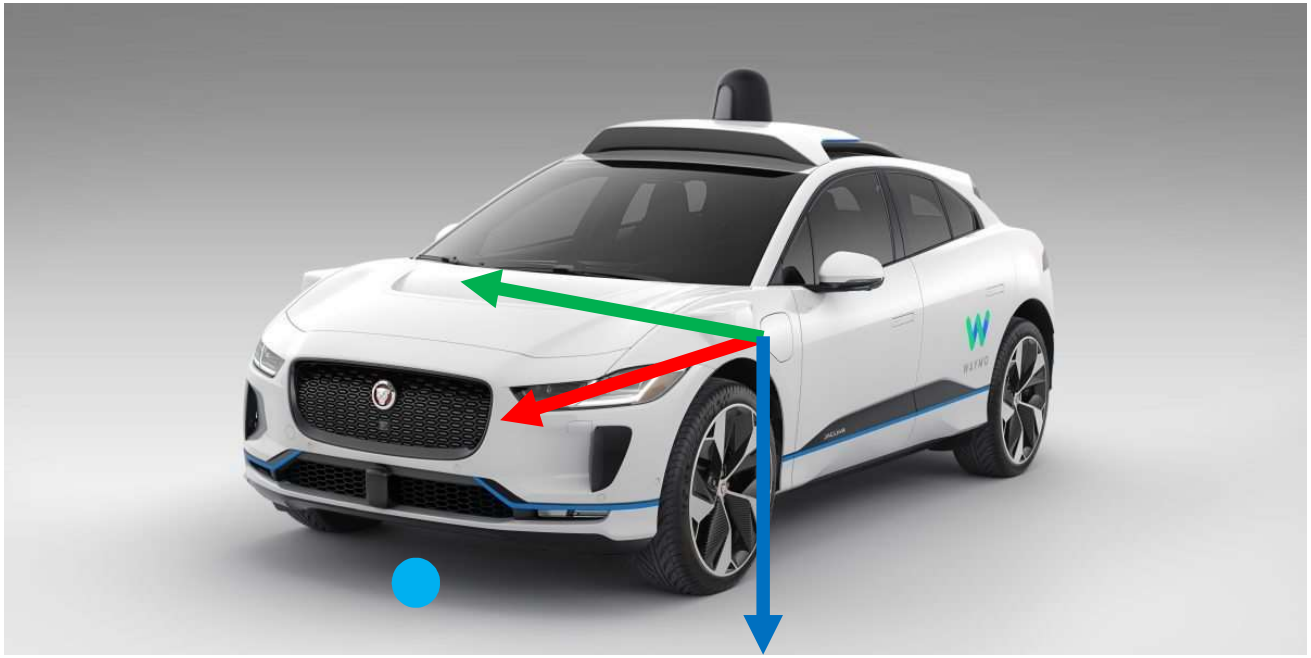
Coordinate systems: The Bane of All Roboticists

Question: Where is the point x = [1, 2.5, 2.5] with respect to the car?



Coordinate systems: The Bane of All Roboticists

Question: Where is the point x = [1, 2.5, 2.5] with respect to the car?



Coordinate systems: The Bane of All Roboticists

Question: Where is the point x = [1, 2.5, 2.5] with respect to the car?

Coordinate systems: The Bane of All Roboticists

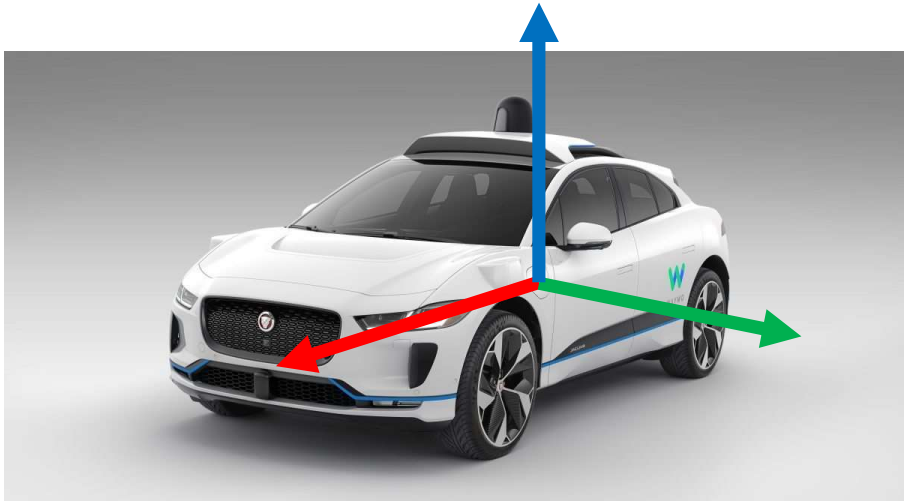
Question: Where is the point x = [1, 2.5, 2.5] with respect to the car?

Key Point: Coordinate expressions are only meaningful *if you know the reference frame*.

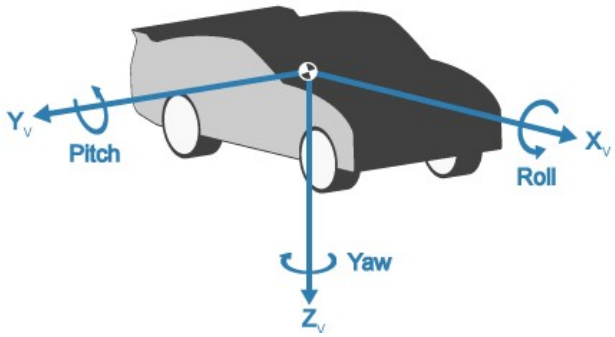
Pitfalls

- **LOTS** of different frame conventions
- **LOTS** of different frames to keep track of

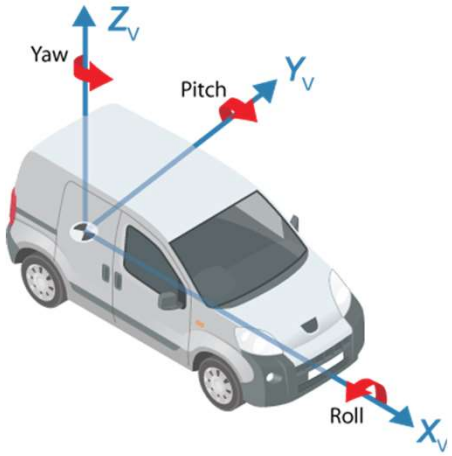
⇒ **LOTS** of opportunities for mistakes!



Examples of common frame conventions



Vehicle *z*-down (robotics)



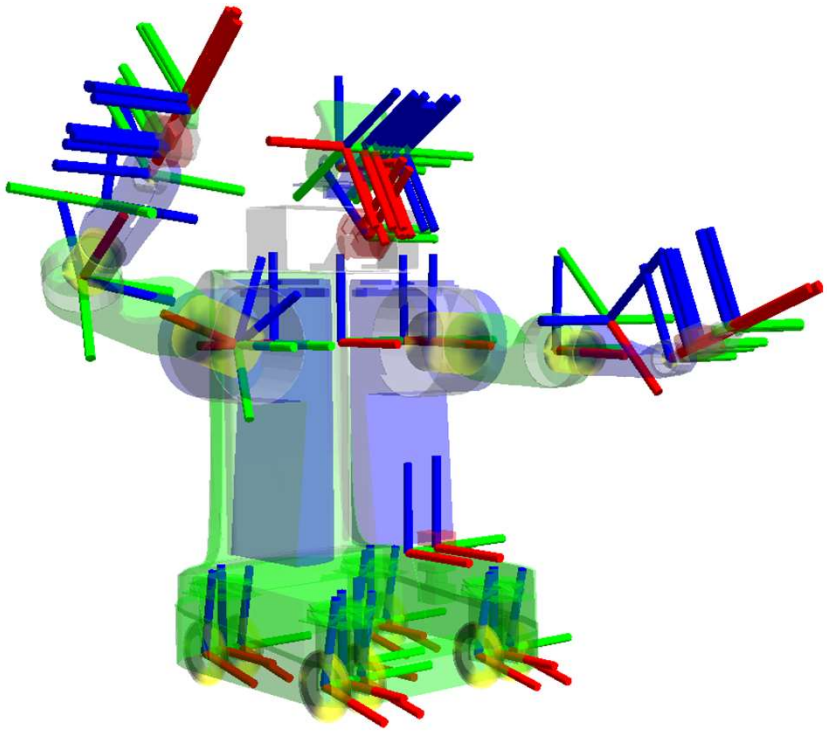
Vehicle *z*-up (robotics)

Camera body-fixed (computer vision)

These are **both** used in MATLAB *in the same toolbox* (!)

Examples of coordinate frames on robots

Willow Garage PR2 robot



PR2 coordinate frames

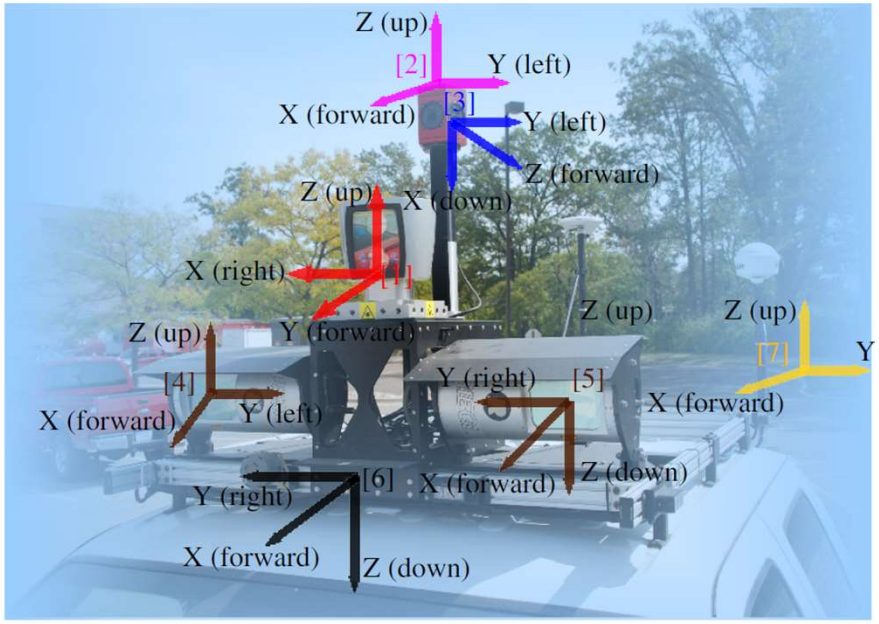
Examples of coordinate frames on robots

Main point: Note that

- every sensor
- every rigid link / moving part

has its own associated coordinate frame.

That is **A LOT** to keep track of …
=> **LOTS** of opportunities for mistakes!



[1] Velodyne, [2] Ladybug3 (actual location: center of camera system),
[3] Ladybug3 Camera 5, [4] Right Riegl, [5] Left Riegl,
[6] Body Frame (actual location: center of rear axle)
[7] Local Frame (Angle between the X-axis and East is known)

Pandey et al.: "Ford Campus Vision and Lidar Data Set"

Los Angeles Times

Mars Probe Lost Due to Simple Math Error

BY ROBERT LEE HOTZ
OCT. 1, 1999 12 AM PT

TIMES SCIENCE WRITER

NASA lost its $125-million Mars Climate Orbiter because spacecraft engineers failed to convert from English to metric measurements when exchanging vital data before the craft was launched, space agency officials said Thursday.

A navigation team at the Jet Propulsion Laboratory used the metric system of millimeters and meters in its calculations, while Lockheed Martin Astronautics in Denver, which designed and built the spacecraft, provided crucial acceleration data in the English system of inches, feet and pounds.

As a result, JPL engineers mistook acceleration readings measured in English units of pound-seconds for a metric measure of force called newton-seconds.

In a sense, the spacecraft was lost in translation.

Don't let this happen to you!!

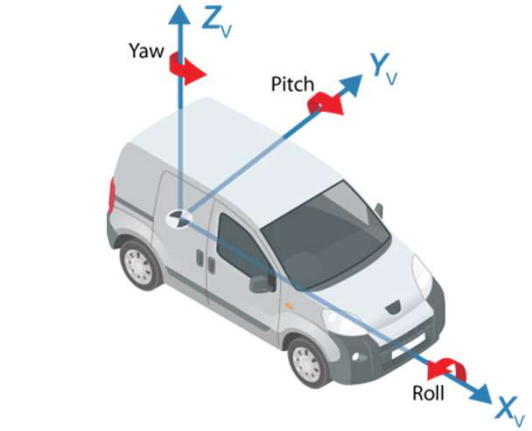
3 Simple Rules for Minimizing Needless Coordinate-Based Suffering

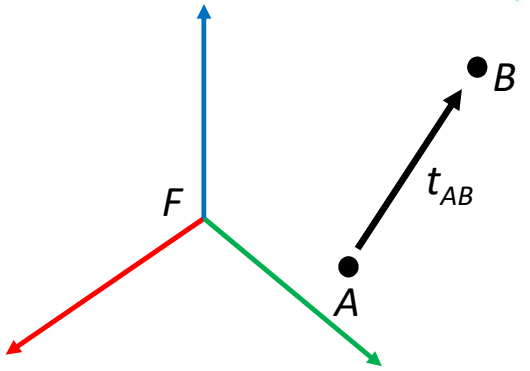
1. **Have Fear*:** if it is not *absolutely clear* what the reference frame is.

   ⇒ You can (and should!) always check this using a 3D plotting tool

2. Always draw a **frame diagram**!

3. Use **explicit notation** for coordinate expressions that captures:
   - The object of interest (to which the property attaches)
   - The datum (reference) for the property
   - The coordinate frame

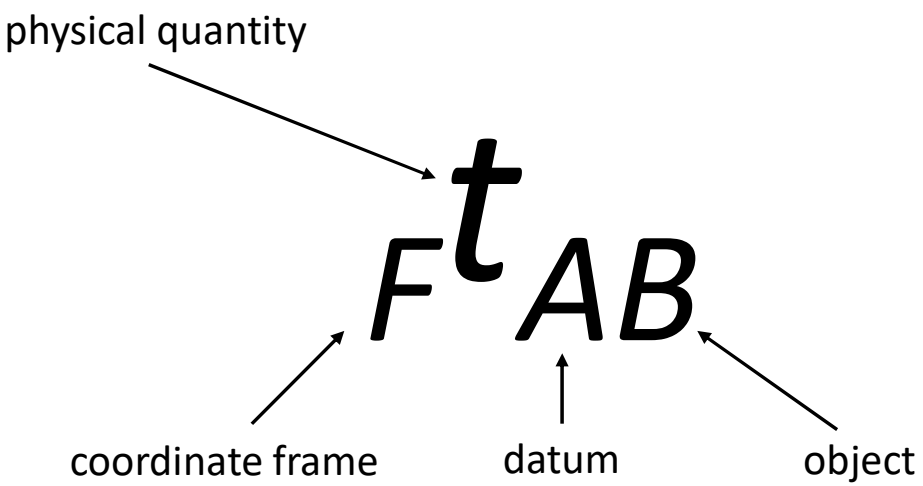


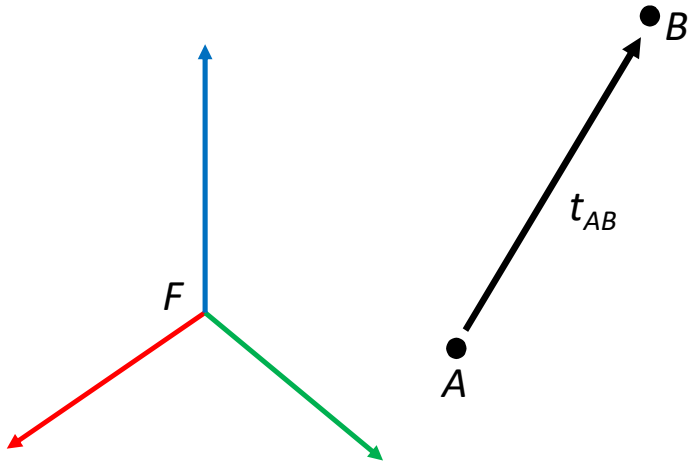


*h/t Paul Furgale

Notational conventions for coordinate expressions

Common **explicit** notational convention for a physical quantity:



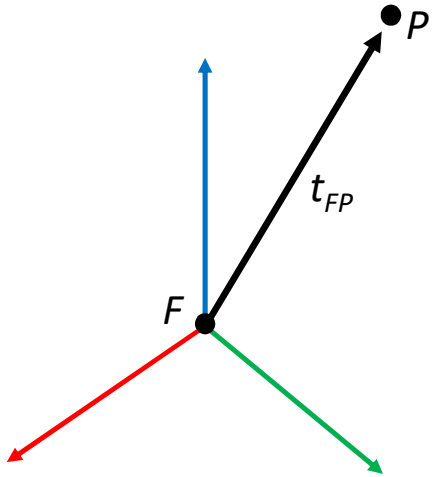


Example: $_F t_{AB}$ means "the translation of *B* relative to *A*, expressed in coordinate frame *F*"

Notational conventions for coordinate expressions

Special case: We may drop explicit reference to the datum if we are expressing a quantity with respect to the origin of the coordinate frame.

$$ {}_{F}p := {}_{F}t_{FP} $$



Example: ${}_{F}p$ means "the position of point *P* expressed in frame *F*"

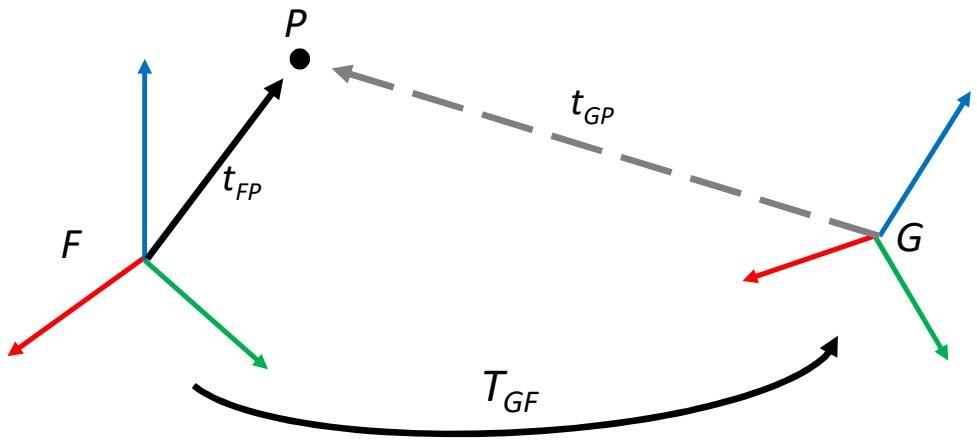
Transformations between coordinate frames

Two coordinate frames *F* and *G* are related by an *affine transformation*.

Notation convention:

$$T_{GF}$$

Transform to Transform from



Example: T_{GF} means "the affine transformation sending a point *from frame F to frame G*."

Payoff: Using this convention, coordinate transformations satisfy simple "cancellation laws":

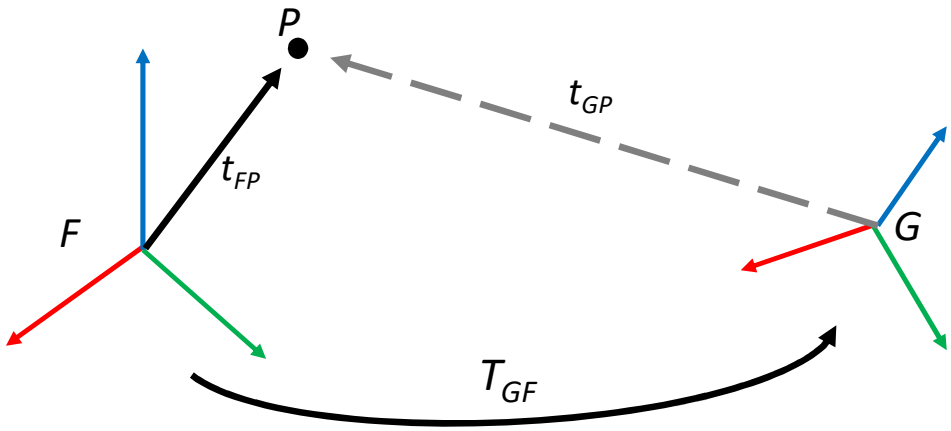
$$_G p = T_{GF}(\ _F p) \qquad T_{HF} = T_{HG}\, T_{GF}$$

Homogeneous representations of coordinate transformations

Two coordinate frames *F* and *G* are related by an *affine transformation*.

⇒There are $b_{GF} \in \mathbb{R}^d$ and $A_{GF} \in GL(d)$ so that:

$$T_{GF}(x) = A_{GF}x + b_{GF}$$



Observation:

$$\begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} A_{GF}x + b_{GF} \\ 1 \end{pmatrix} = \begin{pmatrix} T_{GF}(x) \\ 1 \end{pmatrix}$$

Implication: If we write a vector *y* in homogeneous coordinates as $\hat{y} \coloneqq \begin{pmatrix} y \\ 1 \end{pmatrix}$, then:

$$\widehat{T_{GF}(x)} = \begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix} \hat{x}$$

Therefore: We can identify the affine linear transformation T_{GF} with the following matrix:

$$T_{GF} \coloneqq \begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix} \in GL(d+1)$$

Exercise

Given frame transformations T_{GF} and T_{HG}, with:

$$T_{GF}(x) = A_{GF}x + b_{GF} \qquad T_{HG}(x) = A_{HG}x + b_{HG}$$

Find:

1. The frame transformation T_{HF}.

2. The frame transformation T_{FG}.

Solution

Given frame transformations T_{GF} and T_{HG}, with:

$$T_{GF}(x) = A_{GF}x + b_{GF} \qquad T_{HG}(x) = A_{HG}x + b_{HG}$$

Find:

1. The frame transformation T_{HF}. Using homogeneous representation:

$$T_{HF} = T_{HG}\,T_{GF} = \begin{pmatrix} A_{HG} & b_{HG} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} A_{HG}A_{GF} & A_{HG}b_{GF} + b_{HG} \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow T_{HF}(x) = A_{HG}A_{GF}x + (A_{HG}b_{GF} + b_{HG})$$

Note that once again the subscripts "cancel" in the right way.

2. The frame transformation T_{FG}. Again using homogeneous representation:

$$T_{FG} = T_{GF}^{-1} = \begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} A_{GF}^{-1} & -A_{GF}^{-1}b_{GF} \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow T_{FG}(x) = A_{GF}^{-1}x - A_{GF}^{-1}b_{GF}$$

The affine group

We have seen that every coordinate transformation between frames in $\mathbb{R}^d$ can be represented by a (d+1) x (d+1) matrix in the set:

$$Aff(d) \coloneqq \left\{ \begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix} \mid A \in GL(d), b \in \mathbb{R}^d \right\}$$

In the exercise, we also showed that:

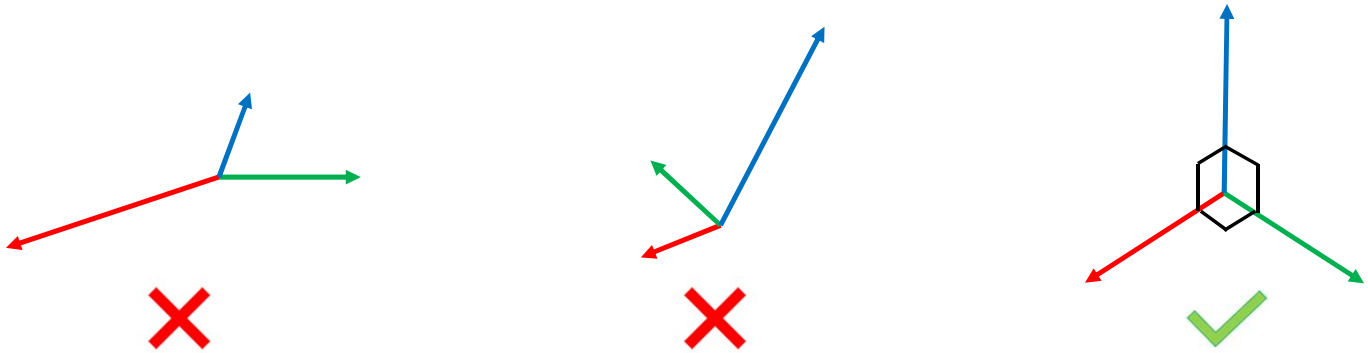
- XY ∈ Aff(d) for all X, Y∈ Aff(d)
- X^{-1}∈ Aff(d) for all X ∈ Aff(d)

⇒This proves that Aff(d) is a subgroup of the general linear group GL(d+1).
We call this group the **affine group** – this is the group of *coordinate transformations in* $\mathbb{R}^d$.

We'll hear more about matrix groups next time …

Structure-preserving subgroups of the affine group

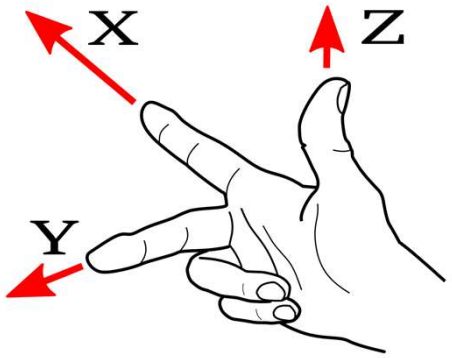
So far we have considered *completely generic* coordinate systems and transformations.



But: often we would like our coordinate frames & transformations to have some "nice" additional structure.

In particular, we often take our coordinate systems to be:

- Orthonormal
- Right-handed



Structure-preserving subgroups of the affine group

This restriction imposes additional constraints on the transformations *T* that relate these frames.

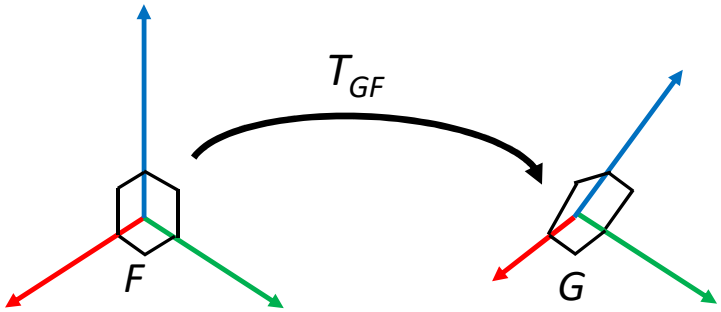
- If *F* and *G* are both **orthonormal**, then T_{GF} *preserves distances* between mapped points:

$$d\big(T_{GF}(x), T_{GF}(y)\big) = d(x, y)$$

- In this case, A_{GF} belongs to the **orthogonal group**:

$$O(d) \coloneqq \{R \in \mathbb{R}^{d\times d} | R^T R = I\}$$

This is the group of *linear* transformations of $\mathbb{R}^d$ that *preserve the inner product*.



$$T_{GF} = \begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix}$$

- The group of all *distance-preserving transformations* of $\mathbb{R}^d$ is called the **Euclidean group:**

$$E(d) \coloneqq \left\{ \begin{pmatrix} R & b \\ 0 & 1 \end{pmatrix} \mid R \in O(d), b \in \mathbb{R}^d \right\}$$

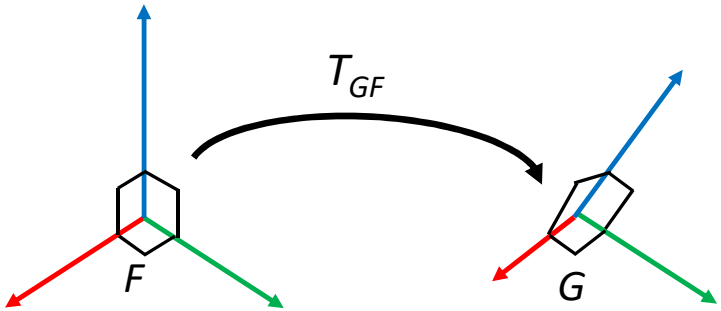
Structure-preserving subgroups of the affine group

This restriction imposes additional constraints on the transformations *T* that relate these frames.

- If *F* and *G* are both **orthonormal *and* right-handed**, then T_{GF} preserves *distance and orientation*.
- In this case, A_{GF} belongs to the **special orthogonal group**, or the group of **rotations**:

$$SO(d) \coloneqq \{R \in \mathbb{R}^{d\times d} | R^T R = I, \det(R) = 1\}$$

This is the group of *linear* transformations of $\mathbb{R}^d$ that *preserve the inner product and orientation*.



$$T_{GF} = \begin{pmatrix} A_{GF} & b_{GF} \\ 0 & 1 \end{pmatrix}$$

- The group of all *distance- and orientation-preserving transformations* of $\mathbb{R}^d$ is called the **special Euclidean group**, or the group of ***rigid motions:***

$$SE(d) \coloneqq \left\{ \begin{pmatrix} R & b \\ 0 & 1 \end{pmatrix} \mid R \in SO(d), b \in \mathbb{R}^d \right\}$$

A taxonomy of spatial transformation groups

- **Affine group:** group of transformations between *general coordinate frames* in $\mathbb{R}^d$:

$$Aff(d) \coloneqq \left\{ \begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix} \mid A \in GL(d), b \in \mathbb{R}^d \right\}$$

- **Euclidean group:** *distance-preserving* transformations of $\mathbb{R}^d$:

$$E(d) \coloneqq \left\{ \begin{pmatrix} R & b \\ 0 & 1 \end{pmatrix} \mid R \in O(d), b \in \mathbb{R}^d \right\}$$

- **Special Euclidean group (rigid motions):** *distance- and orientation-preserving* transformations of $\mathbb{R}^d$

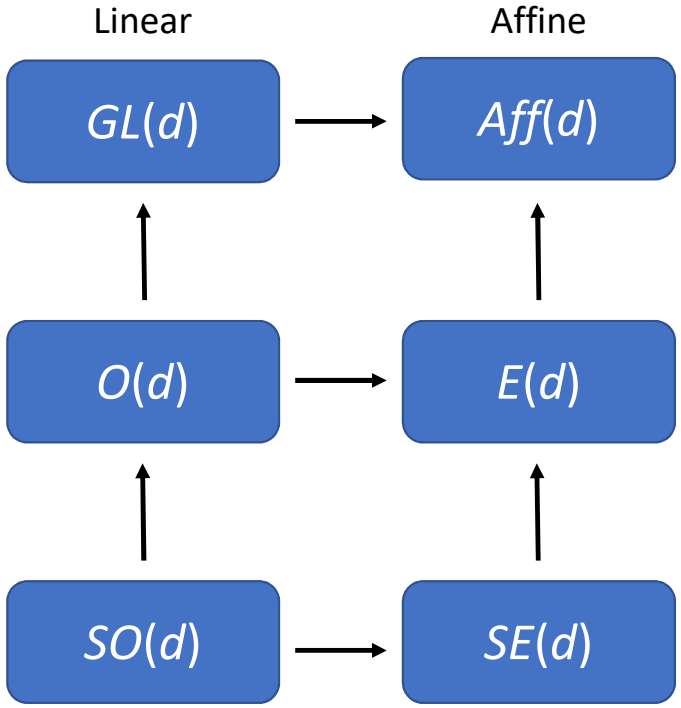
$$SE(d) \coloneqq \left\{ \begin{pmatrix} R & b \\ 0 & 1 \end{pmatrix} \mid R \in SO(d), b \in \mathbb{R}^d \right\}$$

- **Orthogonal group:** *inner-product-preserving linear maps* of $\mathbb{R}^d$:

$$O(d) \coloneqq \{R \in \mathbb{R}^{d \times d} | R^T R = I\}$$

- **Special orthogonal group (rotations):** *inner-product- and orientation-preserving linear maps* of $\mathbb{R}^d$:

$$SO(d) \coloneqq \{R \in \mathbb{R}^{d \times d} | R^T R = I, \det(R) = +1\}$$



Transformation group containment

Rigid body motion

We can model the *pose* (*position* and *orientation*) of a rigid body (i.e. robot) by attaching a fixed coordinate frame *B* to its body.

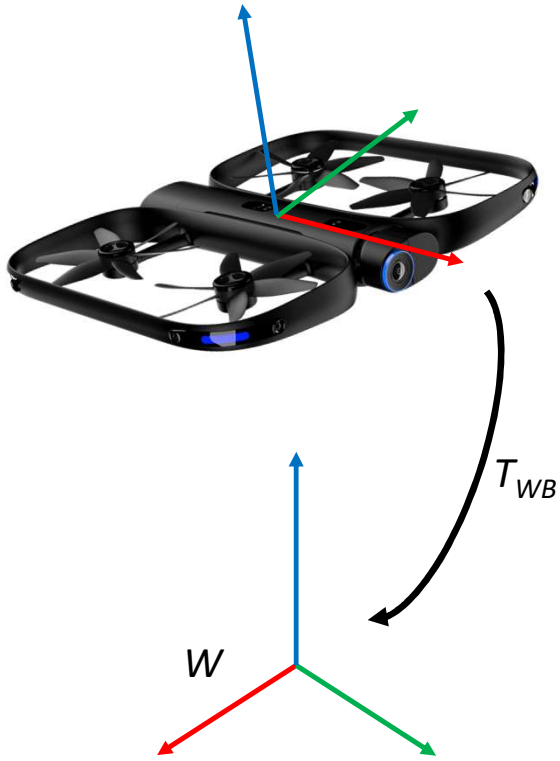
- We represent the pose of *B* with respect to a reference frame *W* using T_{WB}, the transformation that sends *B* to *W*:

$$T_{WB} = \begin{pmatrix} R_{WB} & t_{WB} \\ 0 & 1 \end{pmatrix}$$

We call T_{WB} the "pose of B with respect to W".

- We typically take *W* and *B* to be orthonormal and right-handed, so that $T_{WB} \in \mathrm{SE(d)}$ and $R_{WB} \in \mathrm{SO(d)}$.

- t_{WB} gives the *position* of *B* with respect to *W.*
- R_{WB} gives the *orientation* (or *attitude*) of *B* with respect to *W.*



Rotation representations in 2 & 3 dimensions

Recall that the group of rotations is:

$$SO(d) \coloneqq \{R \in \mathbb{R}^{d\times d} | R^T R = I, \det(R) = 1\}$$

NB: A matrix $R \in$ SO(d) has d^2 elements, and satisfies $d(d+1)/2$ constraints (from upper triangle of symmetric matrix equation $R^T R = I$). Therefore:

$$\dim\big(SO(d)\big) = \frac{d^2 - d}{2}$$

In particular:

- $\dim(SO(2)) = 1 < 2^2 = 4$
- $\dim(SO(3)) = 3 < 3^2 = 9$

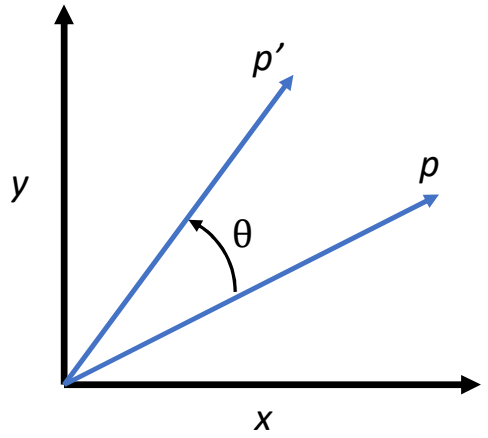
Punchline: Maybe we can save some storage & compute by using a more clever representation for rotations than a matrix …

Rotations in 2D

Recall that we can describe a counterclockwise rotation in the plane using a single parameter: the *rotation angle* θ

The corresponding rotation matrix is then:

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



Rotations in 3D

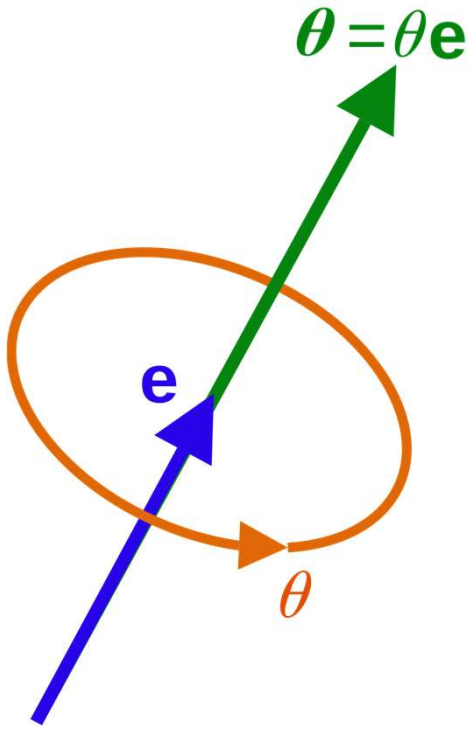
Euler's rotation theorem: Every 3D rotation is equivalent to an elementary right-handed rotation through an angle θ about an axis $\hat{e}$.

Axis-angle representation: We can describe a 3D rotation by specifying:

- The rotation axis $\hat{e}$ (3 params)
- The rotation angle θ (1 param)

NB: Letting axis $\hat{e}$ be a unit vector, we can represent a 3D rotation using a single 3D vector as:

$$\boldsymbol{\theta} = \theta\hat{e}$$



Rodrigues' Theorem

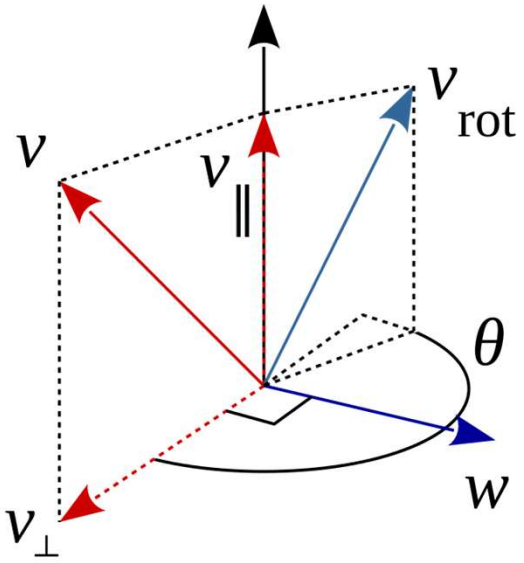
Given: an axis-angle vector $\boldsymbol{\theta} \in \mathbb{R}^3$, let $\theta = \|\boldsymbol{\theta}\|$ and $\hat{e} = \boldsymbol{\theta} / \|\boldsymbol{\theta}\|$.

Then: given any point $v \in \mathbb{R}^3$, the image v_r of v under the rotation determined by $\boldsymbol{\theta}$ is:

$$v_r = \cos(\theta)\, v + \sin(\theta)\, (\hat{e} \times v) + (\hat{e} \cdot v)(1 - \cos(\theta))\hat{e}$$

⇒ The rotation matrix $R(\boldsymbol{\theta})$ corresponding to $\boldsymbol{\theta}$ is:

$$R(\boldsymbol{\theta}) = \cos(\theta)\, I + \sin(\theta)\, [\hat{e}]_\times + (1 - \cos(\theta))\hat{e}\hat{e}^T$$



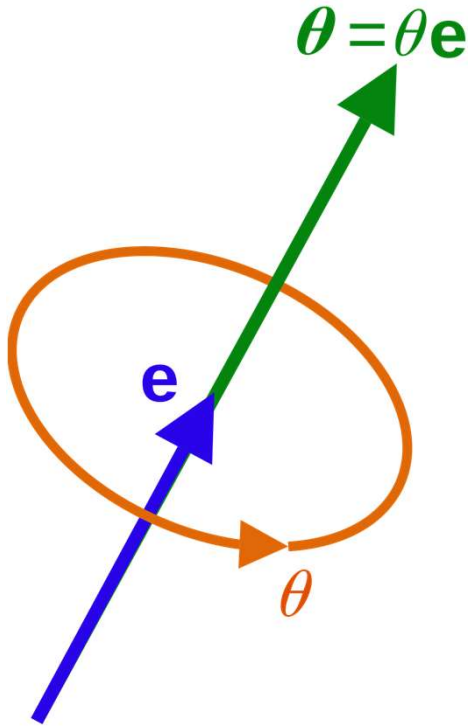
Axis-angle practicalities

Pro:

- Provides a concise [in fact *minimal*] description of a 3D rotation
- Admits an intuitive geometric description

Con:

- Does not admit computing *compositions* (i.e. *products*) of rotations easily
- Acting on (i.e. *rotating*) *points* requires use of trigonometric functions (using Rodrigues' formula)



Quaternions

Quaternions provide a representation of 3D rotations that is closely related to axis-angle, but overcomes some of its weaknesses.

Formally, a **quaternion** is a linear combination of the form:

$$a + bi + cj + dk$$

where *i,j,* and *k* are *elementary quaternions* that satisfy the following multiplication rules:

$$i^2 = j^2 = k^2 = ijk = -1$$

Key facts:

- Quaternions form a 4-dimensional vector space over $\mathbb{R}$
- They are a generalization of the complex numbers
- They are an algebra: a vector space with an *additional* bilinear product (quaternion multiplication)
- They are *non-commutative*: in general $xy \neq yx$ for quaternions x and y.

Unit quaternions and 3D rotation

Given a unit quaternion:

$$q = \underbrace{a}_{\text{real part}} + \underbrace{bi + cj + dk}_{\text{vector part}}$$

let $v = bi + cj + dk$ denote its "vector" (i.e. non-real) part. Then we can write:

$$q = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right)\hat{v}$$

for some angle θ.

⇒ We can associate to each such unit q the rotation R(q) determined by angle θ and axis $\hat{v}$.

Key fact:

- Quaternion multiplication is compatible with this representation: For q1 and q2 unit quaternions:

$$R(q_1 q_2) = R(q_1)R(q_2)$$

- Antipodal quaternions map to the *same* rotation: $R(q) = R(-q)$

Summary of 3D rotation parameterizations

Representation	# Parameters	# Constraints	Pro	Con
Rotation matrix	9	6 (orthonormality)	Fast action on points (mat-vec multiplication)	• Overparameterization • Complex constraints • Composition is expensive (mat-mat multiplication)
Axis-angle	3	0	• Minimal params • Geometrically intuitive	• Action on points is expensive to compute • Can't easily compute compositions
Unit quaternion	4	1 (normalization)	• Slight overparameterization • Simple constraints • Can easily compute compositions	• Action on points is expensive to compute

Key point: Different reps are good for different things! Many software libraries will use *both* for different tasks. (Ex: use quaternions for calculating compositions, and then convert to a rotation matrix to act on points.)

Summary of 3D rotation parameterizations

Representation	# Parameters	# Constraints	Pro	Con
Rotation matrix	9	6 (orthonormality)	Fast action on points (mat-vec multiplication)	• Overparameterization • Complex constraints • Composition is expensive (mat-mat multiplication)
Axis-angle	3	0	• Minimal params • Geometrically intuitive	• Action on points is expensive to compute • Can't easily compute compositions
Unit quaternion	4	1 (normalization)	• Slight overparameterization • Simple constraints • Can easily compute compositions	• Action on points is expensive to compute

NOT SHOWN: Euler angles, because these are an abomination and you should never use them.

Summary

This lecture:

- Coordinate systems and notational conventions
- Coordinate transformations and transformation groups
- How to model the pose of a rigid body
- Representations of rotation in 2D and 3D space

Next time: An introduction to matrix Lie groups