Visual Navigation for Flying Robots

3D Geometry and Sensors

Dr. Jürgen Sturm

Organization: Lab Course

- Robot lab: room 02.05.14 (different room!)
- Exercises: room 02.09.23 (here)
- You have to sign up for a team before May 2nd (team list in student lab)
- After May 2nd, remaining places will be given to students on waiting list
- First exercise sheet is due **next Tuesday 10am**

Today's Agenda

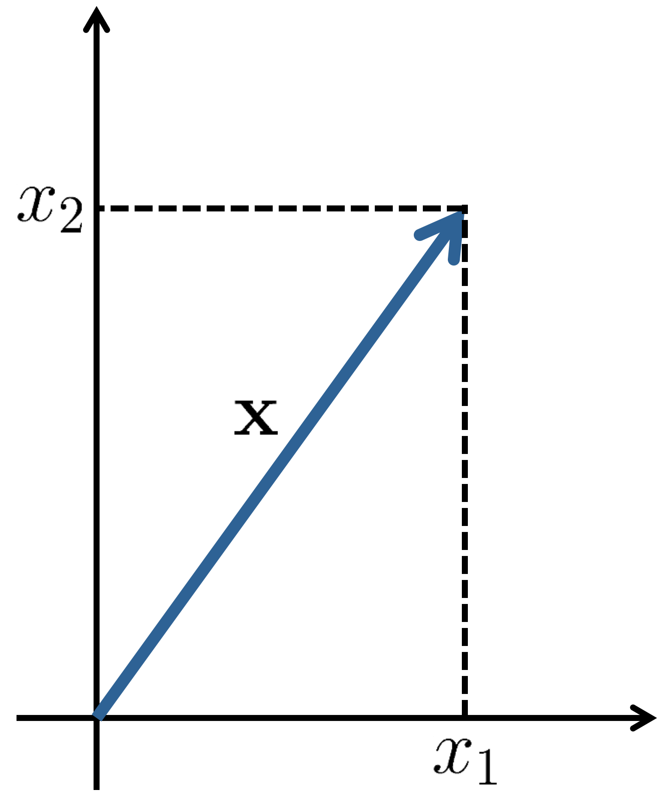
- Linear algebra
- 2D and 3D geometry
- Sensors
- First exercise sheet

Vectors

- Vector and its coordinates

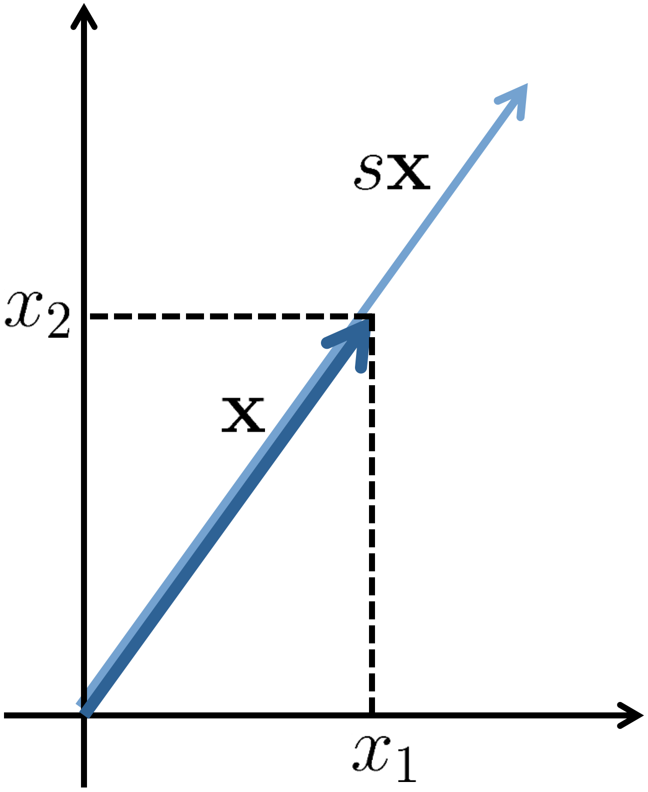
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

- Vectors represent points in an n-dimensional space



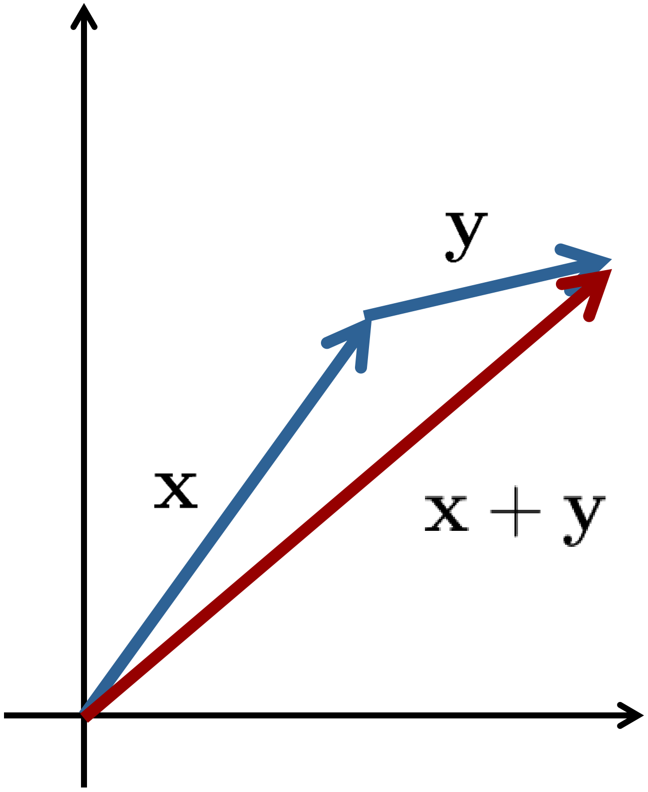
Vector Operations

- **Scalar multiplication**
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product



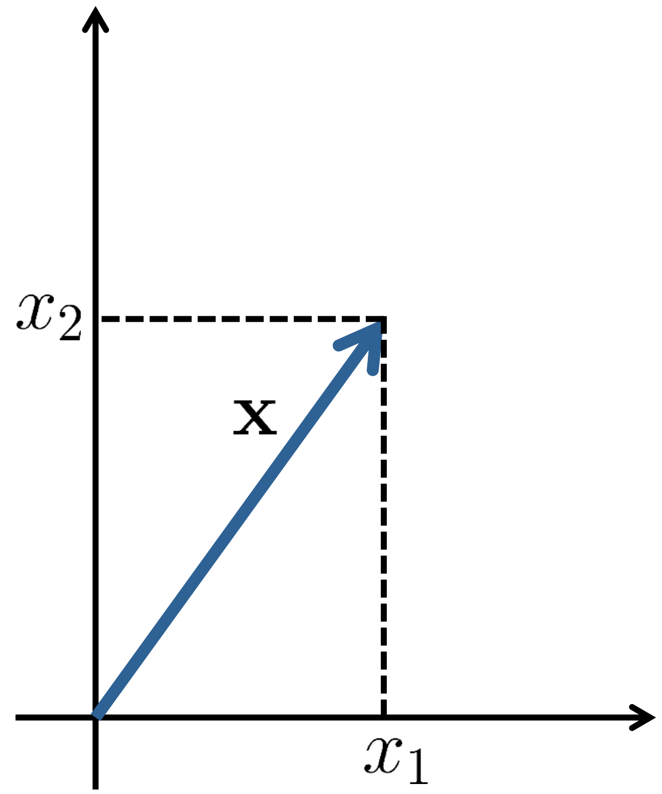
Vector Operations

- Scalar multiplication
- **Addition/subtraction**
- Length
- Normalized vector
- Dot product
- Cross product



Vector Operations

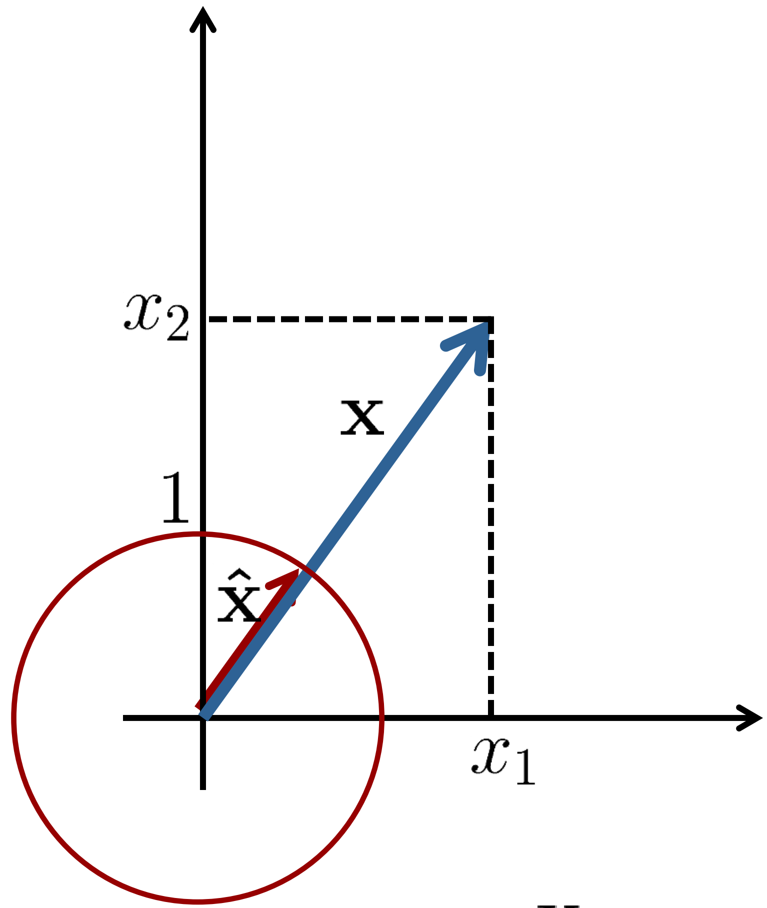
- Scalar multiplication
- Addition/subtraction
- **Length**
- Normalized vector
- Dot product
- Cross product



$$\|x\|_2 = \|x\| = \sqrt{x_1^2 + x_2^2 + \ldots}$$

Vector Operations

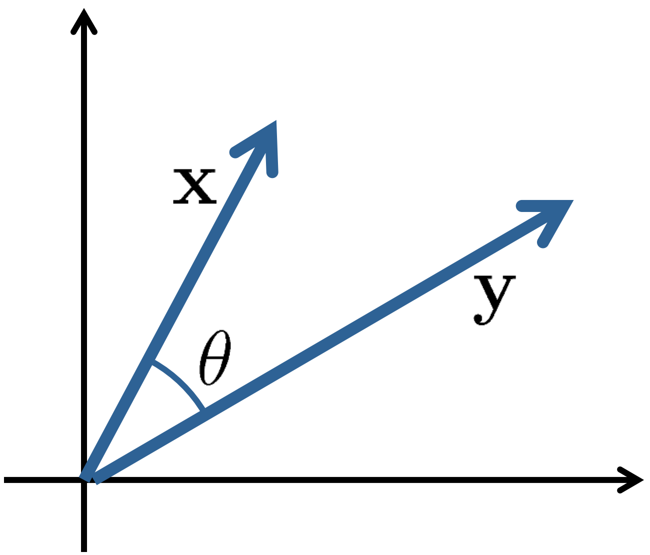
- Scalar multiplication
- Addition/subtraction
- Length
- **Normalized vector**
- Dot product
- Cross product



$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- **Dot product**
- Cross product



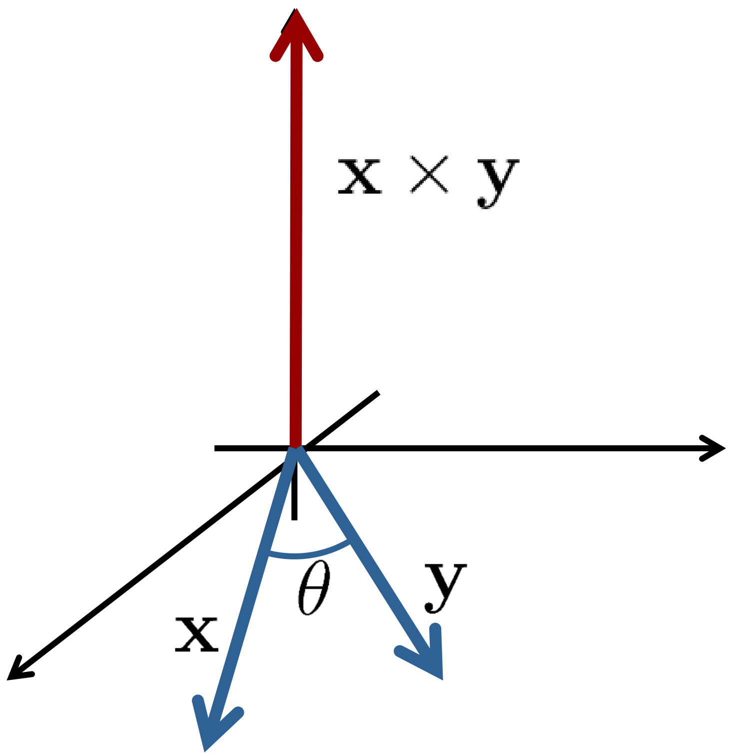
$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$

$\mathbf{x}, \mathbf{y}$ are orthogonal if $\mathbf{x} \cdot \mathbf{y} = 0$

$\mathbf{y}$ is lin. dependent from $\{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ if $\mathbf{y} = \sum_i k_i \mathbf{x}_i$

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- **Cross product**



$$\mathbf{x} \times \mathbf{y} = \|\mathbf{x}\| \, \|\mathbf{y}\| \sin(\theta) \mathbf{n}$$

Cross Product

- Definition

$$\mathbf{x} \times \mathbf{y} = \begin{pmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$$

- Matrix notation for the cross product

$$[\mathbf{x}]_\times = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix}$$

- Verify that $\mathbf{x} \times \mathbf{y} = [\mathbf{x}]_\times \mathbf{y}$

Matrices

- Rectangular array of numbers

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{pmatrix} \in \mathbb{R}^{\overset{\text{rows}}{n} \times \overset{\text{columns}}{m}}$$

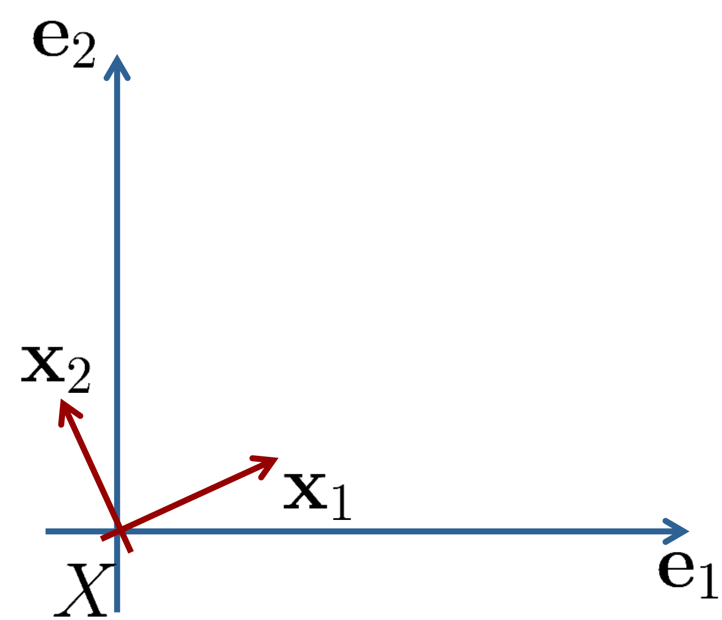
- First index refers to row
- Second index refers to column

Matrices

- Column vectors of a matrix

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{x}_{*1} & \mathbf{x}_{*2} & \dots & \mathbf{x}_{*m} \end{pmatrix}$$



- Geometric interpretation: for example, column vectors can form basis of a coordinate system

Matrices

- Row vectors of a matrix

$$X = \begin{pmatrix} \boxed{x_{11} \quad x_{12} \quad \ldots \quad x_{1m}} \\ \boxed{x_{21} \quad x_{22} \quad \ldots \quad x_{2m}} \\ \vdots \\ \boxed{x_{n1} \quad x_{n2} \quad \ldots \quad x_{nm}} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{1*}^\top \\ \mathbf{x}_{2*}^\top \\ \vdots \\ \mathbf{x}_{n*}^\top \end{pmatrix}$$

Matrices

- Square matrix
- Diagonal matrix
- Upper and lower triangular matrix
- Symmetric matrix
- Skew-symmetric matrix
- (Semi-)positive definite matrix
- Invertible matrix
- Orthonormal matrix
- Matrix rank

Matrices

- Square matrix
- Diagonal matrix
- Upper and lower triangular matrix
- Symmetric matrix $X = X^\top$
- Skew-symmetric matrix $X = -X^\top (= \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix})$
- (Semi-)positive definite matrix $\mathbf{a}^\top X \mathbf{a} \geq 0$
- Invertible matrix
- Orthonormal matrix
- Matrix rank

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

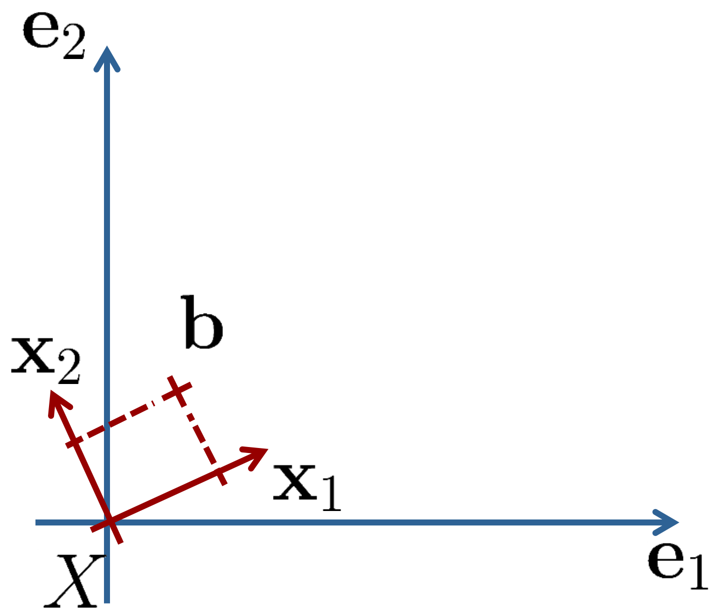
Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- **Matrix-vector multiplication** $X\mathbf{b}$
- Matrix-matrix multiplication
- Inversion

Matrix-Vector Multiplication

$$X \cdot \mathbf{b} = \sum_{k=1}^{n} \mathbf{x}_{*k} \cdot b_k$$

column vectors



- Geometric interpretation:
  A linear combination of the columns of A scaled by the coefficients of **b**
  → coordinate transf. from local to global frame

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- **Matrix-matrix multiplication**
- Inversion

Matrix-Matrix Multiplication

- Operator $\mathbb{R}^{n\times m} \times \mathbb{R}^{m\times p} \to \mathbb{R}^{n\times p}$
- Definition

$$
\begin{aligned}
C &= AB \\
&= A \begin{pmatrix} \mathbf{b}_{*1} & \mathbf{b}_{*2} & \cdots \mathbf{b}_{*p} \end{pmatrix}
\end{aligned}
$$

- Interpretation: transformation of coordinate systems
- Can be used to concatenate transforms

Matrix-Matrix Multiplication

- Not commutative (in general)

$$AB \neq BA$$

- Associative

$$A(BC) = (AB)C$$

- Transpose

$$(AB)^\top = B^\top A^\top$$

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- **Inversion**

Matrix Inversion

- If A is a square matrix of full rank, then there is a unique matrix $B = A^\top$ such that $AB = I$.
- Different ways to compute, e.g., Gauss-Jordan elimination, LU decomposition, …
- When A is orthonormal, then

$$A^{-1} = A^\top$$

Recap: Linear Algebra

- Vectors
- Matrices
- Operators

- Now let’s apply these concepts to 2D+3D geometry

Geometric Primitives in 2D

- 2D point $$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2$$
- Augmented vector $$\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3$$
- Homogeneous coordinates $$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2$$

Geometric Primitives in 2D

- Homogeneous vectors that differ only be scale represent the same 2D point
- Convert back to inhomogeneous coordinates by dividing through last element

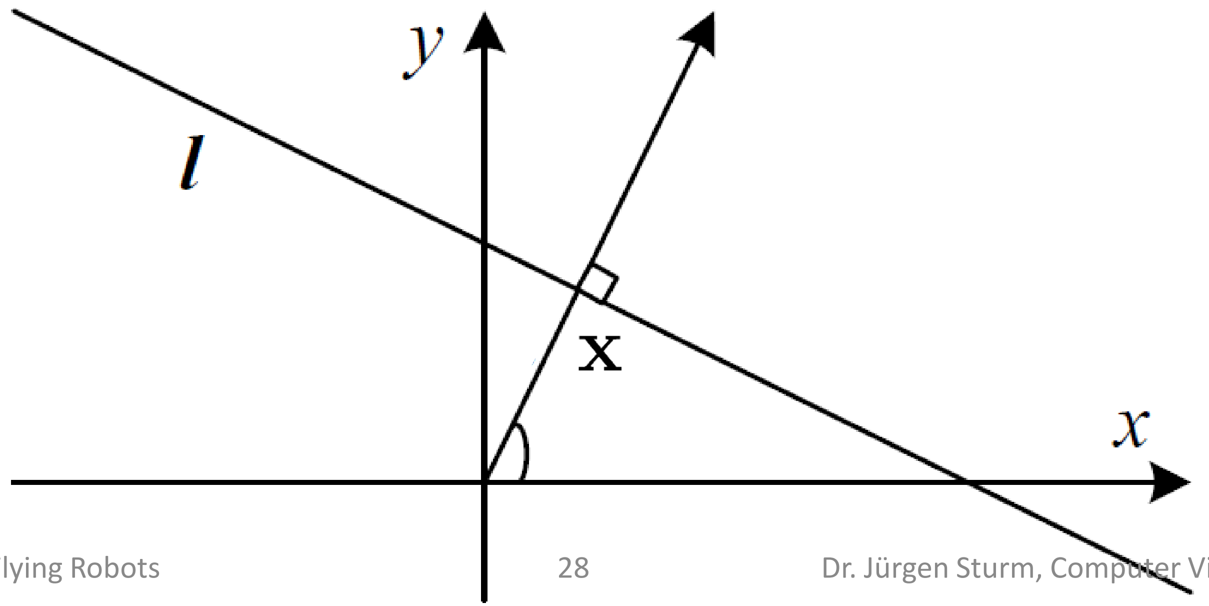
$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{w}\bar{\mathbf{x}}$$

- Points with $\tilde{w} = 0$ are called points at infinity or ideal points

Geometric Primitives in 2D

- 2D line $\tilde{\mathbf{l}} = (a, b, c)^\top$

- 2D line equation $\bar{\mathbf{x}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$

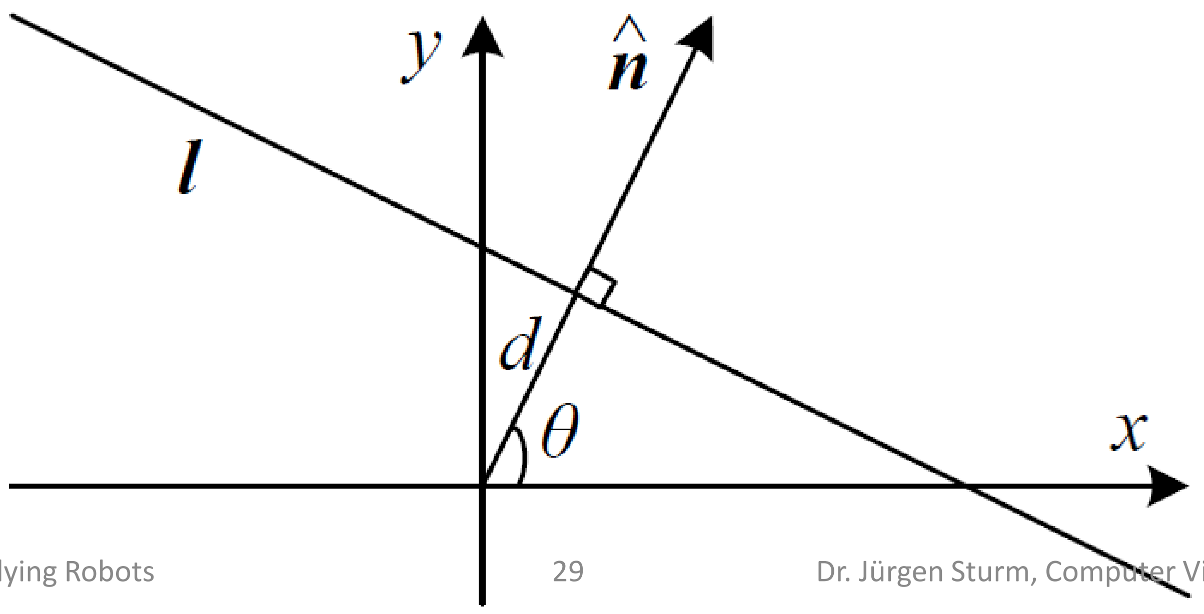


Geometric Primitives in 2D

- Normalized line equation vector

$$\tilde{\mathbf{l}} = (\hat{n}_x, \hat{n}_y, d)^\top = (\hat{\mathbf{n}}, d)^\top \quad \text{with} \quad \|\hat{\mathbf{n}}\| = 1$$

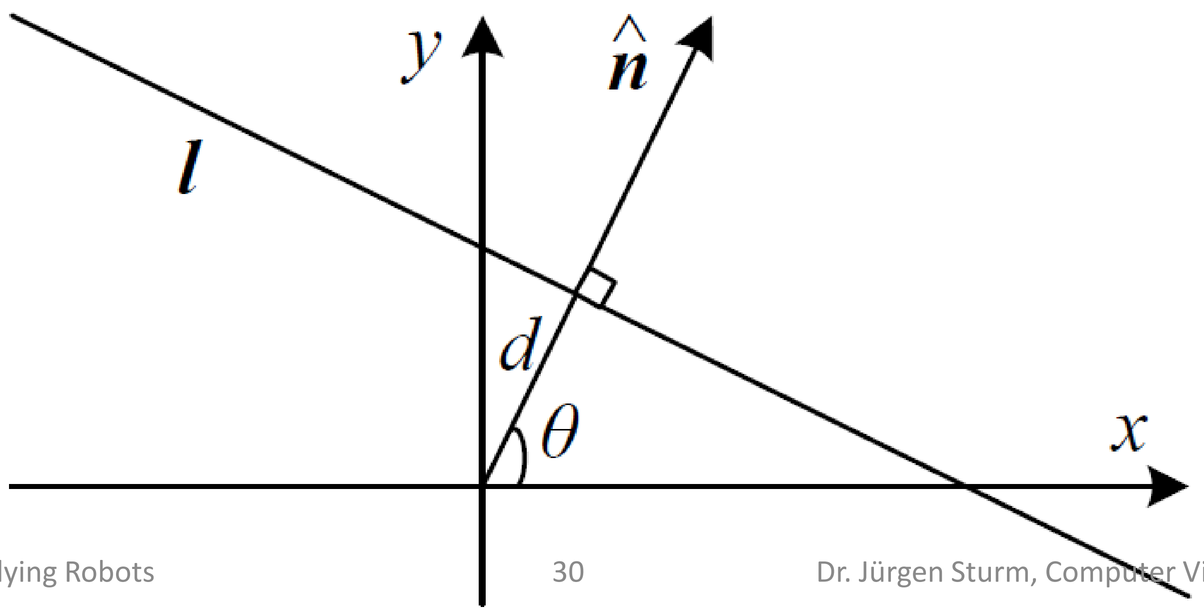
where d is the distance of the line to the origin



Geometric Primitives in 2D

- Polar coordinates of a line: $(\theta, d)^\top$
  (e.g., used in Hough transform for finding lines)

$$\hat{\mathbf{n}} = (\cos\theta, \sin\theta)^\top$$



Geometric Primitives in 2D

- Line joining two points $\tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2$

- Intersection point of two lines $\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$

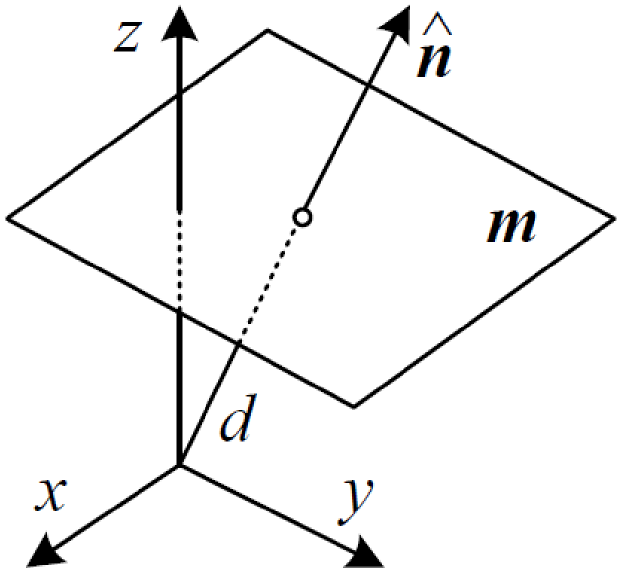
Geometric Primitives in 3D

- 3D point (same as before) $$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$$
- Augmented vector $$\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4$$
- Homogeneous coordinates $$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$$

Geometric Primitives in 3D

- 3D plane $\tilde{\mathbf{m}} = (a, b, c, d)^\top$
- 3D plane equation $\bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$

- Normalized plane
  with unit normal vector
  $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{\mathbf{n}}, d)$
  ($\|\hat{\mathbf{n}}\| = 1$)
  and distance d



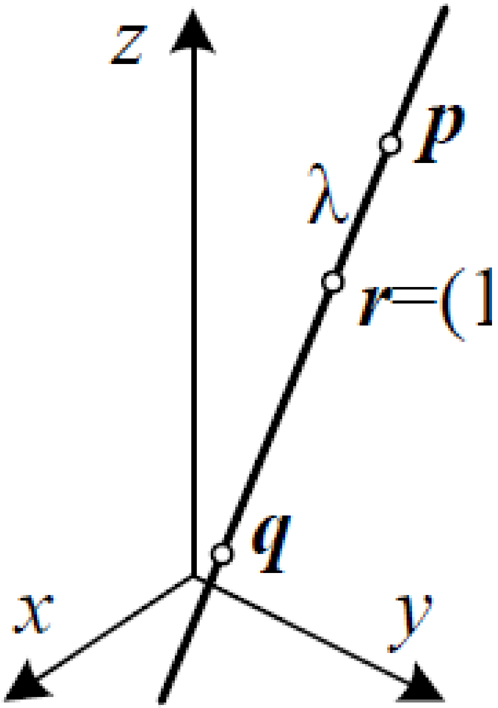
Geometric Primitives in 3D

- 3D line $\mathbf{r} = (1-\lambda)\mathbf{p} + \lambda\mathbf{q}$ through points $\mathbf{p}, \mathbf{q}$
- Infinite line:

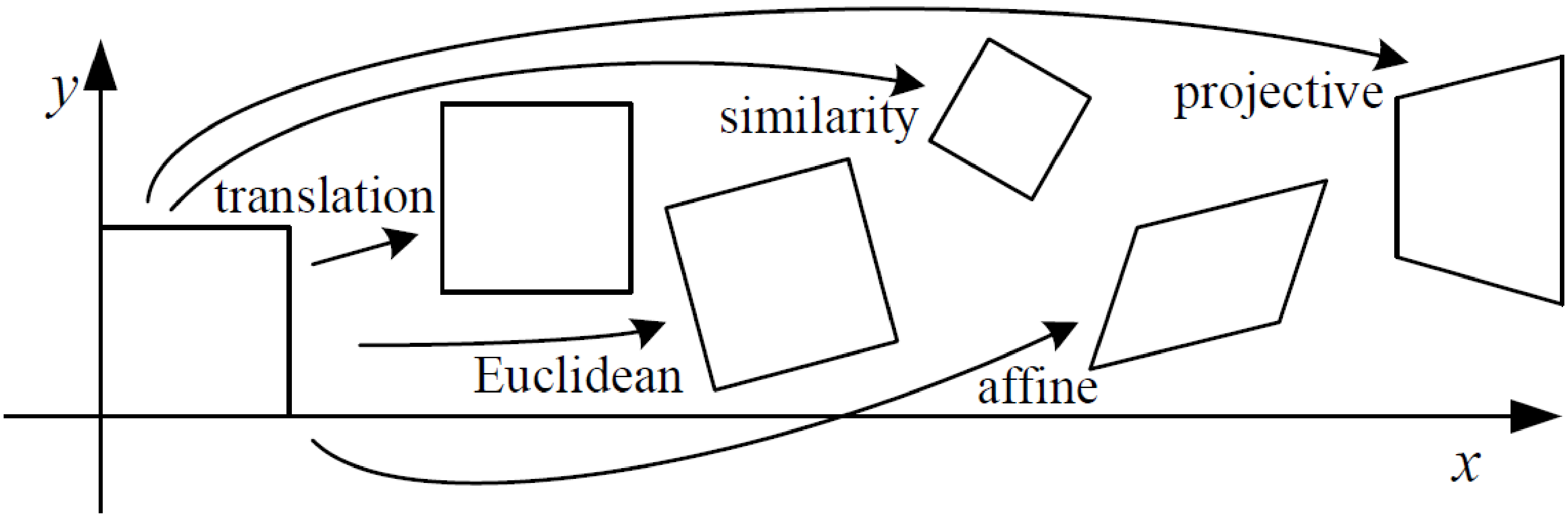
$$\lambda \in \mathbb{R}$$

- Line segment joining $\mathbf{p}, \mathbf{q}$:

$$0 \leq \lambda \leq 1$$



2D Planar Transformations



2D Transformations

- Translation $\mathbf{x}' = \mathbf{x} + \mathbf{t}$

$$\mathbf{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix}}_{2\times 3} \bar{\mathbf{x}}$$

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{3\times 3} \bar{\mathbf{x}}$$

where $\mathbf{t} \in \mathbb{R}^2$ is the translation vector,
$\mathbf{I}$ is the identity matrix, and $\mathbf{0}$ is the zero vector

2D Transformations

- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t}$$

$$\mathbf{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix}}_{2\times 3} \bar{\mathbf{x}}$$

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{3\times 3} \bar{\mathbf{x}}$$

Question: How many DOFs has this transformation?

where $\mathbf{t} \in \mathbb{R}^2$ is the translation vector,
$\mathbf{I}$ is the identity matrix, and $\mathbf{0}$ is the zero vector

2D Transformations

- Rigid body motion or Euclidean transformation (rotation + translation)

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + t \qquad \text{or} \qquad \bar{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

where $\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

is an orthonormal rotation matrix, i.e., $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$

- Distances (and angles) are preserved

2D Transformations

- Scaled rotation/similarity transform

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + t \qquad \text{or} \qquad \bar{\mathbf{x}}' = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Preserves angles between lines

2D Transformations

- Affine transform

$$\bar{\mathbf{x}}' = A\bar{\mathbf{x}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Parallel lines remain parallel

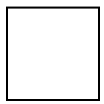
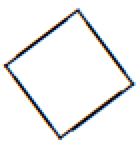
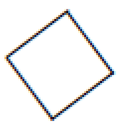
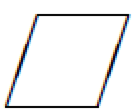
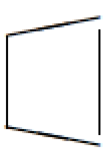
2D Transformations

- Projective/perspective transform

$$\tilde{\mathbf{x}}' = \tilde{H} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tilde{\mathbf{x}}$$

- Note that $\tilde{H}$ is homogeneous (only defined up to scale)
- Resulting coordinates are homogeneous
- Lines remain lines :-)

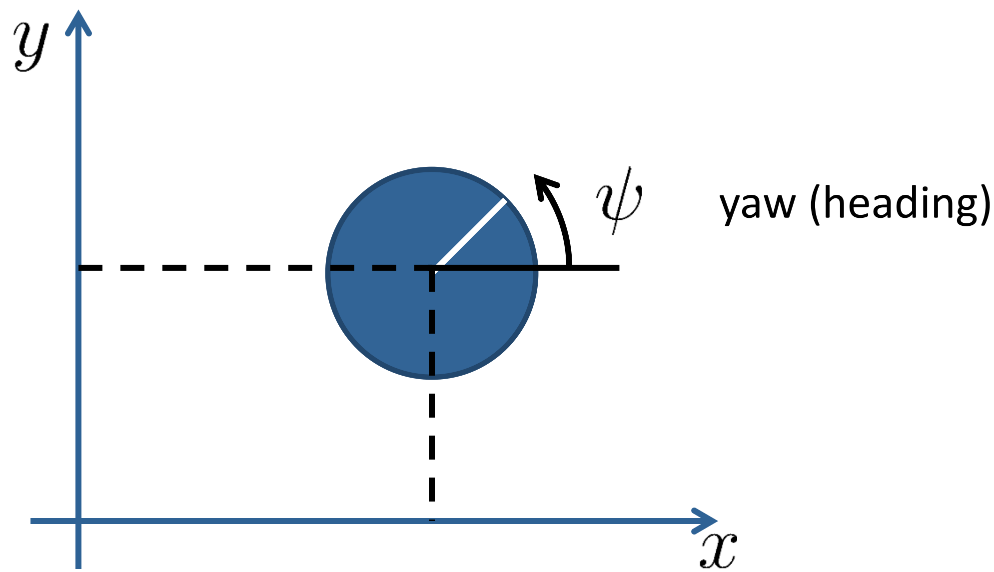
2D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

Examples: Euclidean Transformations

Coordinate Transforms

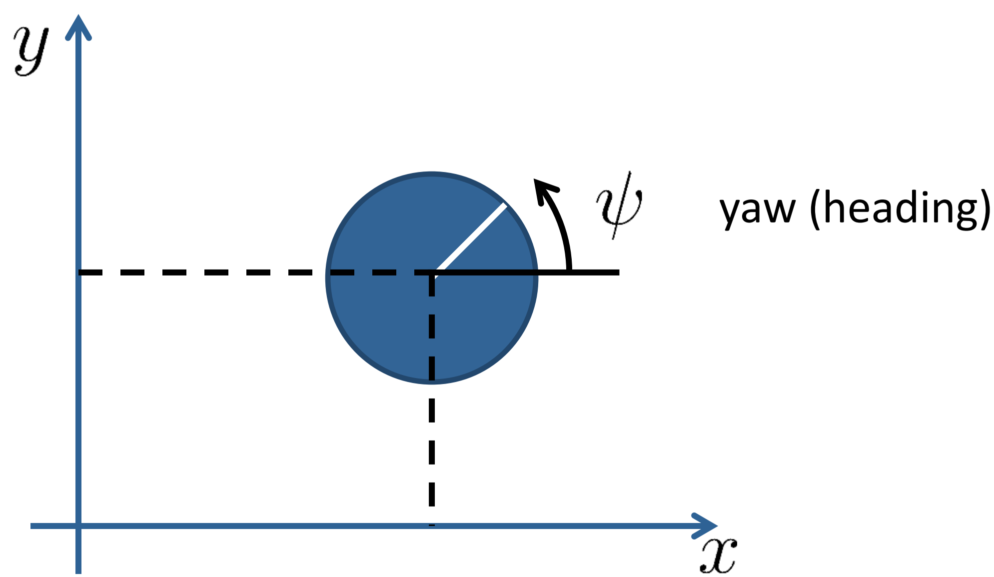
- Robot is located somewhere in space



Coordinate Transforms

- Robot is located somewhere in space

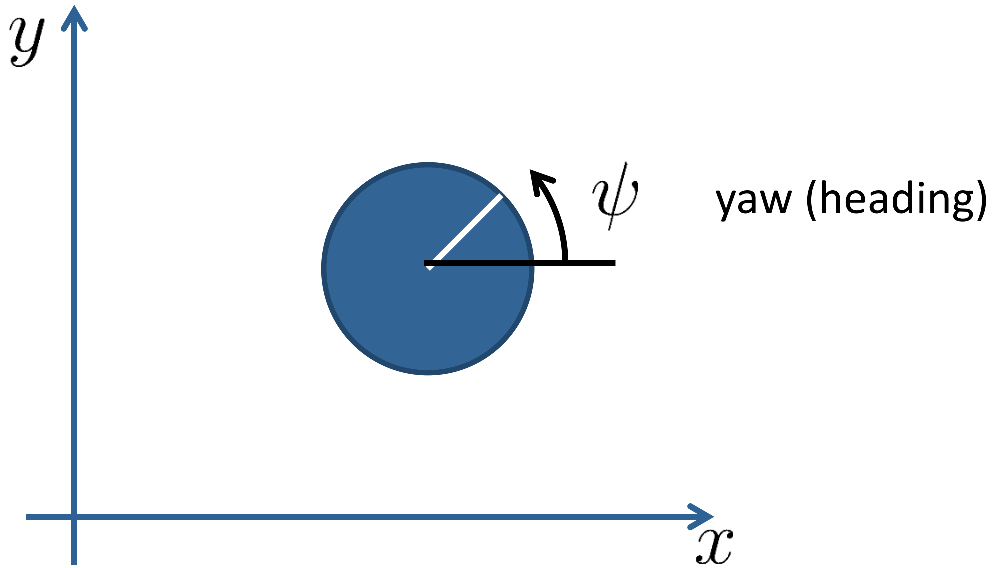
$$X = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} cos\psi & -sin\psi & x \\ sin\psi & cos\psi & y \\ 0 & 0 & 1 \end{pmatrix} \in \mathrm{SE}(2) \subset \mathbb{R}^{3x3}$$



Coordinate Transforms

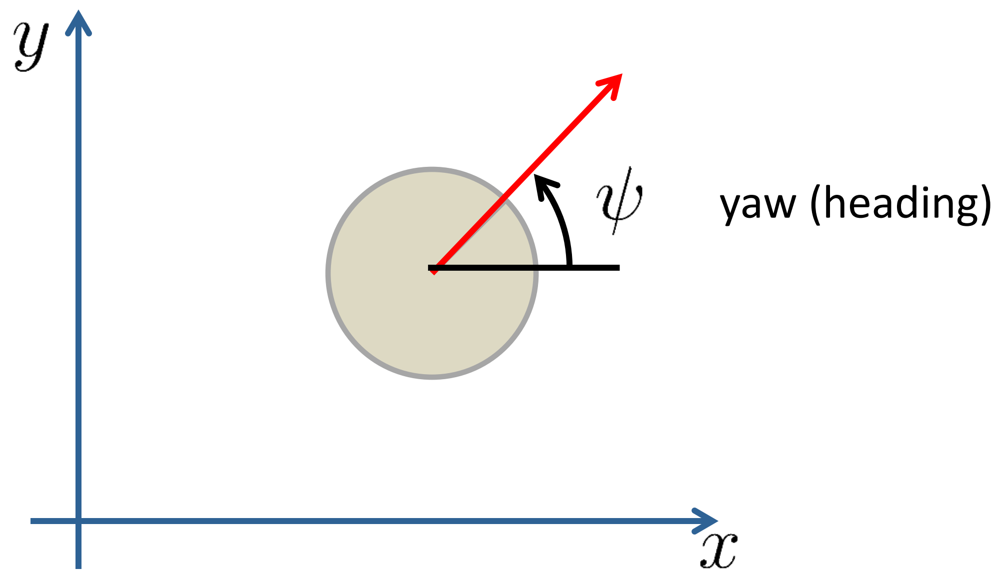
- Robot is located at x=0.7, y=0.5, yaw=45deg

$$X = \begin{pmatrix} \cos 45 & -\sin 45 & 0.7 \\ \sin 45 & \cos 45 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$



Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s (in its local frame)
- What is its speed in the global frame?



Vector Transformation

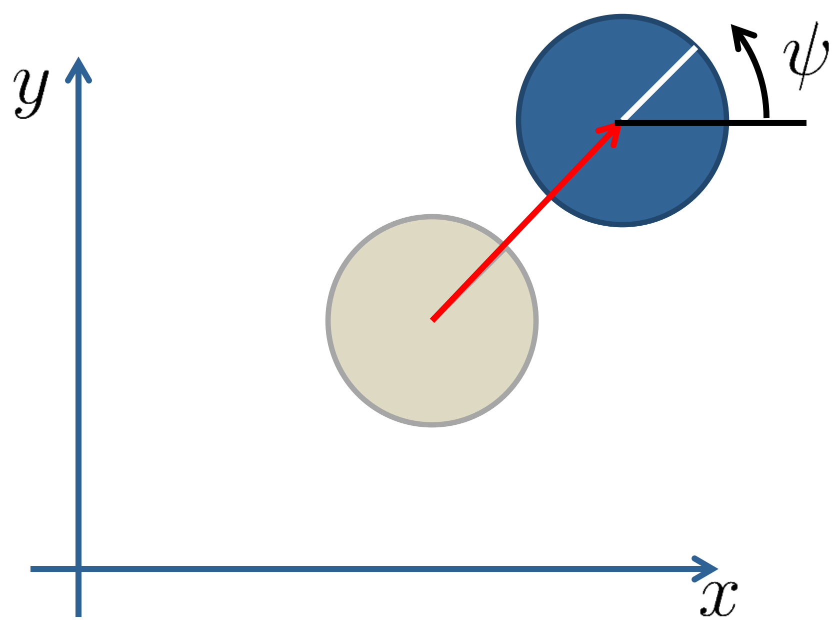
- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s

Inhomogeneous coordinates

$$\mathbf{v}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Homogeneous coordinates

$$\tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$



Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s

$$\begin{aligned}\tilde{\mathbf{v}}_{\text{global}} &= X\tilde{\mathbf{v}}_{\text{local}} \\ &= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\end{aligned}$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s

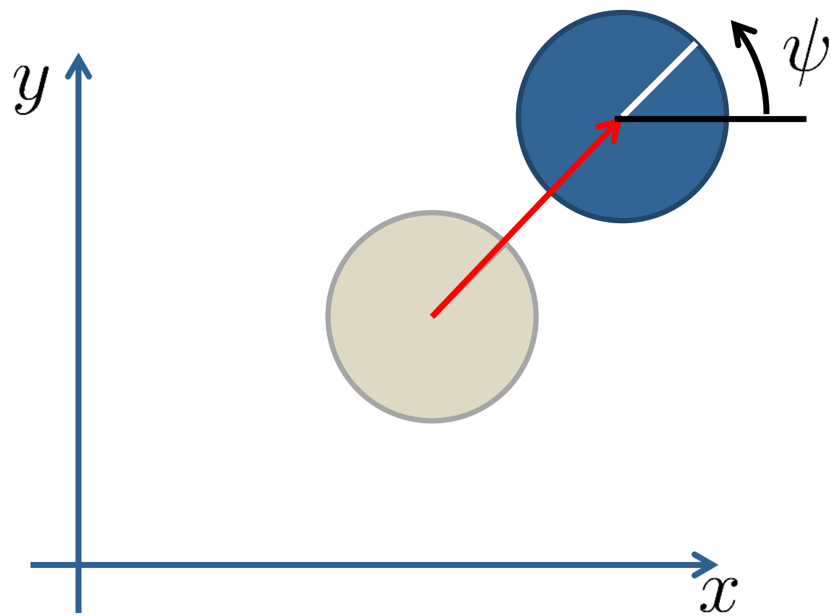
$$
\begin{aligned}
\tilde{\mathbf{v}}_{\text{global}} &= X \tilde{\mathbf{v}}_{\text{local}} \\
&= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix}
\end{aligned}
$$

Vector Transformation

- Robot is located at x=0.7, y=0.5, yaw=45deg
- Robot moves forward with 1m/s

$$\tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$\tilde{\mathbf{v}}_{\text{global}} = \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix}$$



Vector Transformation

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

Vector Transformation

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}}$$

$$\tilde{\mathbf{v}}_{\text{local}} = X^{-1} \tilde{\mathbf{v}}_{\text{global}}$$

Inverse Transformations

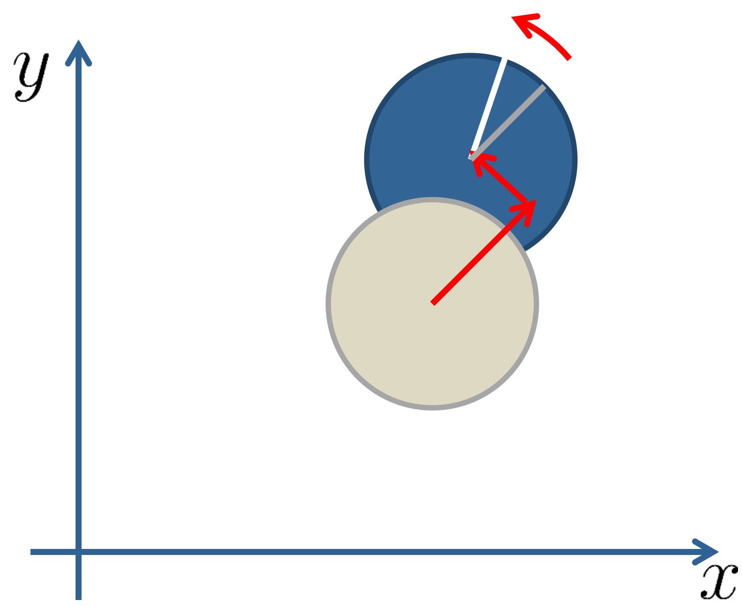
- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\tilde{\mathbf{v}}_{\text{global}} = X \tilde{\mathbf{v}}_{\text{local}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tilde{\mathbf{v}}_{\text{local}}$$

$$\tilde{\mathbf{v}}_{\text{local}} = X^{-1} \tilde{\mathbf{v}}_{\text{global}} = \begin{pmatrix} R^\top & -R^\top \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tilde{\mathbf{v}}_{\text{global}}$$

Coordinate System Transformations

- Now consider a different motion
- Robot moves 0.2m forward, 0.1m sideways, and turns by 10deg



Coordinate System Transformations

- Robot moves 0.2m forward, 0.1m sideways, and turns by 10deg

$$U_1 = \begin{pmatrix} \cos 10 & -\sin 10 & 0.2 \\ \sin 10 & \cos 10 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

Coordinate System Transformations

- After this motion, the robot pose (in the global frame) becomes

$$X_2 = XU$$

$$= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \cdots$$

Coordinate System Transformations

Note: The order matters

- Move 1m forward, then turn 90deg left
- Turn 90deg left, then move 1m forward

$$AB \neq BA$$

3D Transformations

- Translation

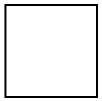
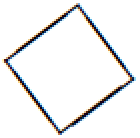
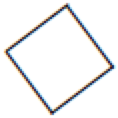
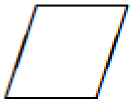
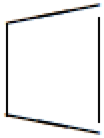
$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{4\times 4} \bar{\mathbf{x}}$$

- Euclidean transform (translation + rotation), (also called the Special Euclidean group SE(3))

$$\bar{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Scaled rotation, affine transform, projective transform…

3D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times 4}$	15	straight lines	

3D Euclidean Transformtions

- Translation $\mathbf{t}$ has 3 degrees of freedom
- Rotation R has 3 degrees of freedom

$$X = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D Rotations

- Rotation matrix
  (also called the special orientation group SO(3))

- Euler angles
- Axis/angle
- Unit quaternion

Rotation Matrix

- Orthonormal 3x3 matrix

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

- Column vectors correspond to coordinate axes
- Special orientation group $R \in \mathrm{SO}(3)$
- What operations do we typically do with rotation matrices?

Rotation Matrix

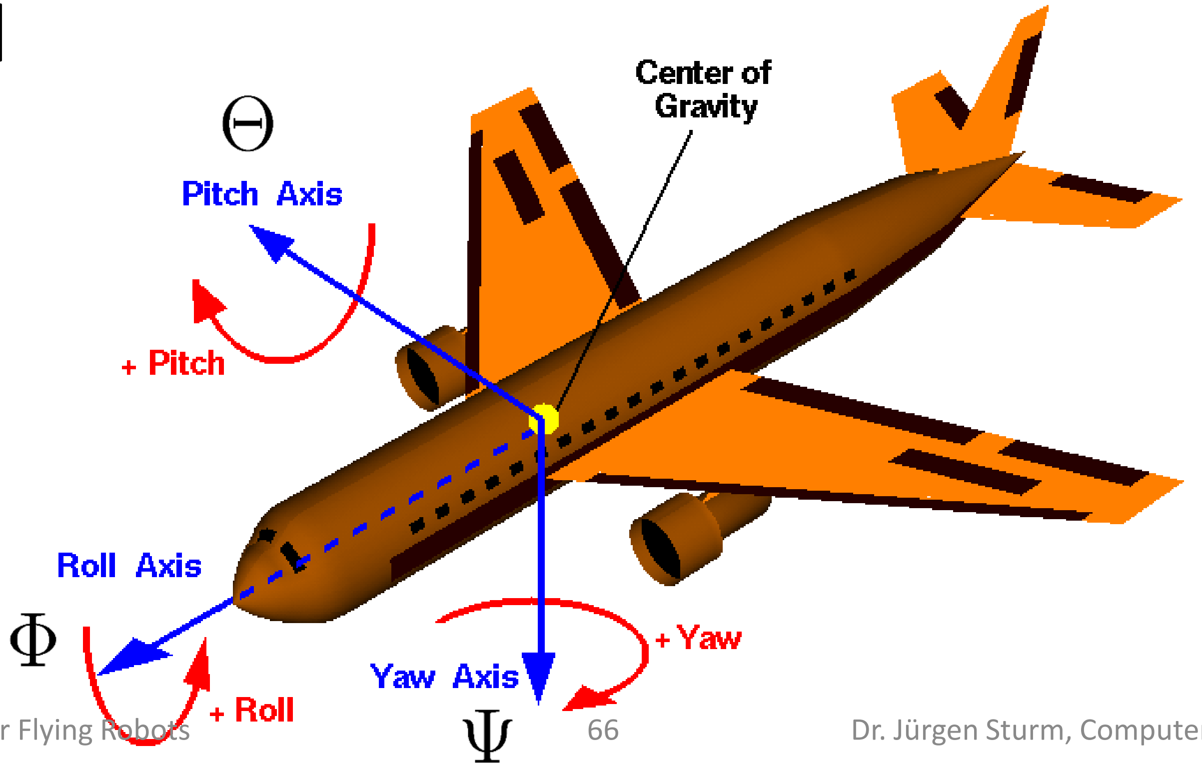
- Orthonormal 3x3 matrix

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

- Advantage: Can be easily concatenated and inverted (how?)
- Disadvantage: Over-parameterized (9 parameters instead of 3)

Euler Angles

- Product of 3 consecutive rotations (e.g., around X-Y-Z axes)
- Roll-pitch-yaw convention is very common in aerial



Roll-Pitch-Yaw Convention

- Yaw Ψ, Pitch Θ, Roll Φ to rotation matrix

$$R = R_Z(\Psi)R_Y(\Theta)R_X(\Phi)$$
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix} \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix}$$

- Rotation matrix to Yaw-Pitch-Roll

$$\phi = \operatorname{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)$$
$$\psi = -\operatorname{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right)$$
$$\theta = \operatorname{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)$$

Euler Angles

- Advantage:
  - Minimal representation (3 parameters)
  - Easy interpretation
- Disadvantages:
  - Many “alternative” Euler representations exist (XYZ, ZXZ, ZYX, …)
  - Difficult to concatenate
  - Singularities (gimbal lock)

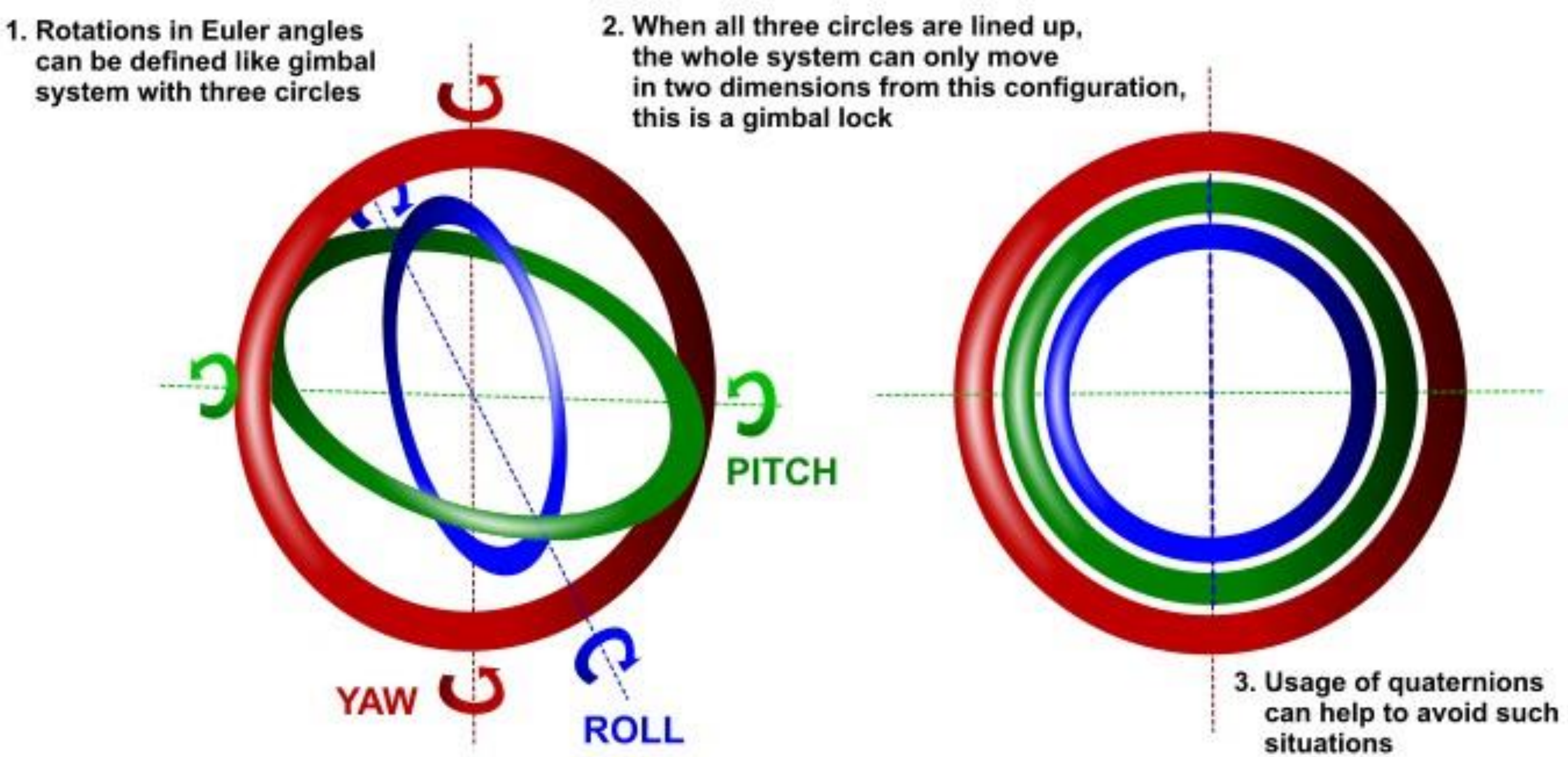
Euler Angles

- Euler angles (3 parameters)
  - Concatenation: convert to rotation matrix, multiply, convert back
  - Inverse: convert to rotation matrix, invert, convert back

$$R_Z(\psi_1)R_Y(\theta_1)R_X(\phi_1) \cdot R_Z(\psi_2)R_Y(\theta_2)R_X(\phi_2)$$
$$\neq R_Z(\psi_1+\psi_2)R_Y(\theta_1+\theta_2)R_X(\phi_1+\phi_2)$$

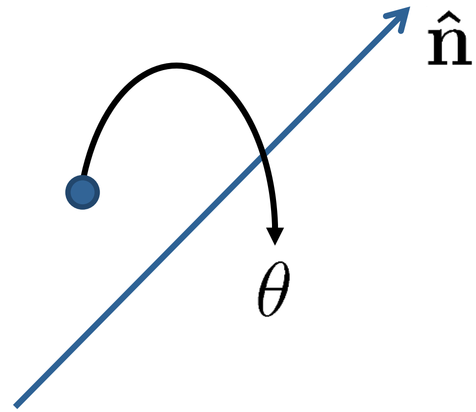
Gimbal Lock

- When the axes align, one degree-of-freedom (DOF) is lost…



Axis/Angle

- Represent rotation by
  - rotation axis $\hat{\mathbf{n}}$ and
  - rotation angle θ
- 4 parameters $(\hat{\mathbf{n}}, \theta)$
- 3 parameters $\boldsymbol{\omega} = \theta \hat{\mathbf{n}}$
  - length is rotation angle
  - also called the angular velocity
  - minimal but not unique (why?)



Conversion

- Rodriguez' formula

$$R(\hat{\mathbf{n}}, \theta) = I + \sin\theta[\hat{\mathbf{n}}]_\times + (1 - \cos\theta)[\hat{\mathbf{n}}]_\times^2$$

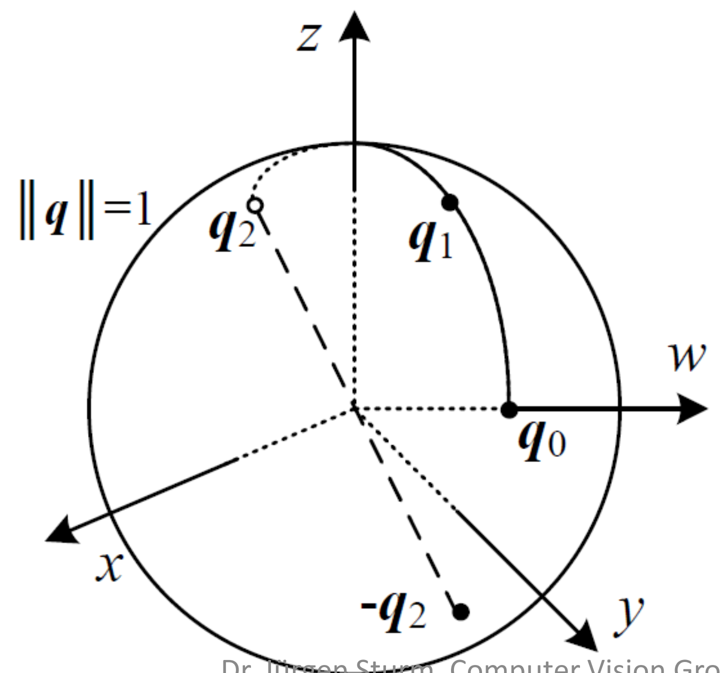
- Inverse

$$\theta = \cos^{-1}\left(\frac{\operatorname{trace}(R) - 1}{2}\right), \hat{\mathbf{n}} = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

see: An Invitation to 3D Vision, Y. Ma, S. Soatto, J. Kosecka, S. Sastry, Chapter 2 (available online)

Unit Quaternions

- Quaternion $\mathbf{q} = (q_x, q_y, q_z, q_w)^\top \in \mathbb{R}^4$
- Unit quaternions have $\|\mathbf{q}\| = 1$
- Opposite sign quaternions represent the same rotation $\mathbf{q} = -\mathbf{q}$
- Otherwise unique



Unit Quaternions

- Advantage: multiplication and inversion operations are efficient
- Quaternion-Quaternion Multiplication

$$\begin{aligned} \mathbf{q}_0\mathbf{q}_1 &= (\mathbf{v}_0, w_0)(\mathbf{v}_1, w_1) \\ &= (\mathbf{v}_0 \times \mathbf{v}_1 + w_0\mathbf{v}_1 + w_1\mathbf{v}_0, w_0 w_1 - \mathbf{v}_0\mathbf{v}_1) \end{aligned}$$

- Inverse (flip sign of v or w)

$$\begin{aligned} \mathbf{q}^{-1} &= (\mathbf{v}, w)^{-1} \\ &= (\mathbf{v}, -w) \end{aligned}$$

Unit Quaternions

- Quaternion-Vector multiplication (rotate point p with rotation q)

$$\mathbf{p}' = \mathbf{v}\bar{\mathbf{p}}\mathbf{q}^{-1}$$

with $\bar{\mathbf{p}} = (x, y, z, 0)^\top$

- Relation to Axis/Angle representation

$$\mathbf{q} = (\mathbf{v}, w) = (\sin\frac{\theta}{2}\hat{\mathbf{n}}, \cos\frac{\theta}{2})$$

3D Orientations

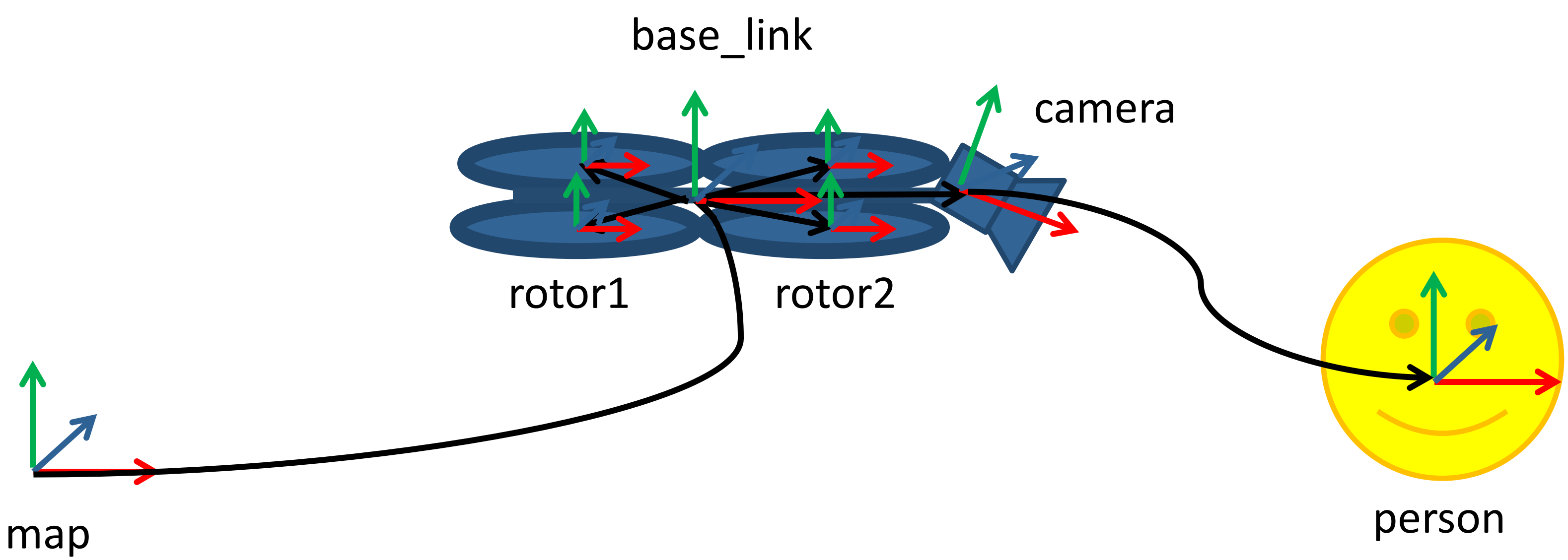
- **Note:** In general, it is very hard to “read” 3D orientations/rotations, no matter in what representation
- **Observation:** They are usually easy to visualize and can then be intuitively interpreted
- **Advice:** Use 3D visualization tools for debugging (RVIZ, libqglviewer, …)

C++ Libraries for Lin. Alg./Geometry

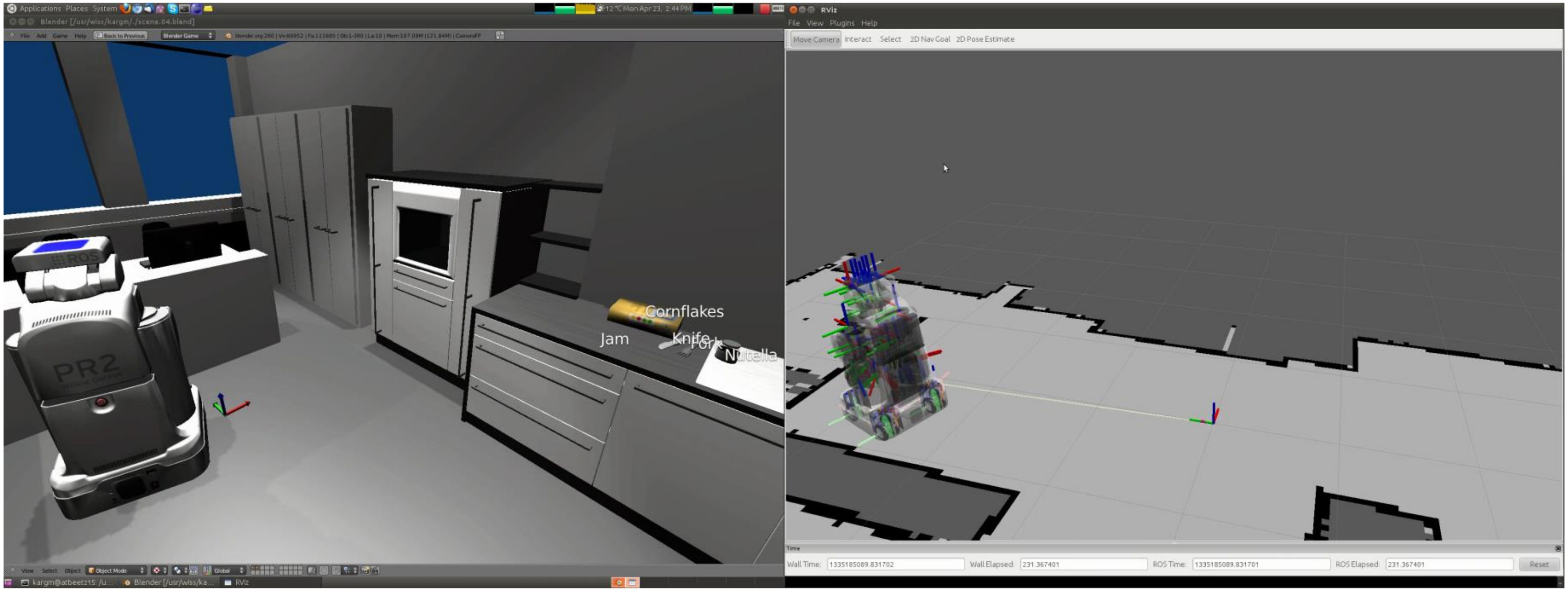
- Many C++ libraries exist for linear algebra and 3D geometry
- Typically conversion necessary
- Examples:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, many functions, plain C)
  - boost::array (used by ROS messages)
  - Bullet library (3D geometry, used by ROS tf)
  - Eigen (both linear algebra and geometry, my recommendation)

Example: Transform Trees in ROS

- TF package represents 3D transforms between rigid bodies in the scene as a tree
- Collects transformations
- Simple query interface



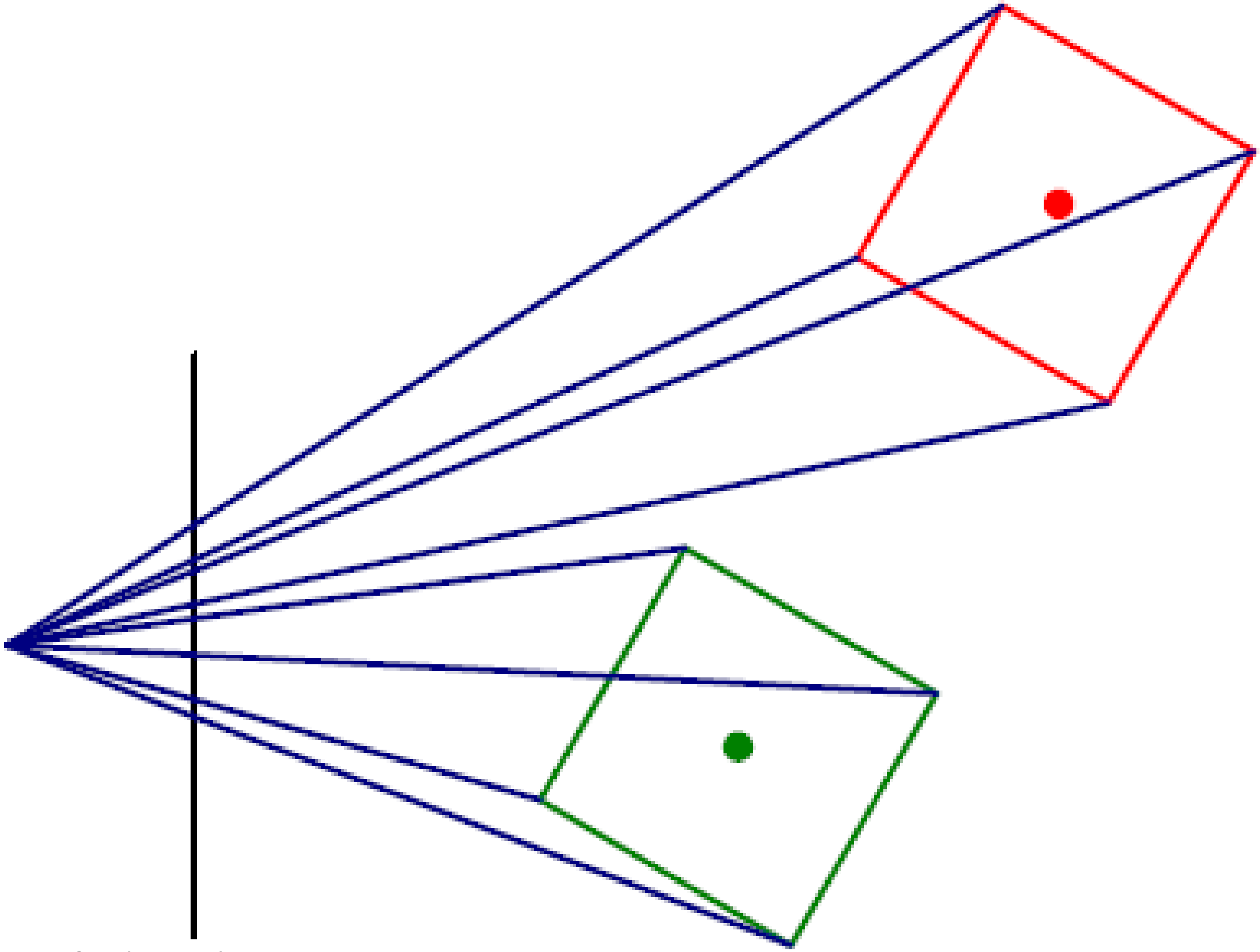
Example: Video from PR2



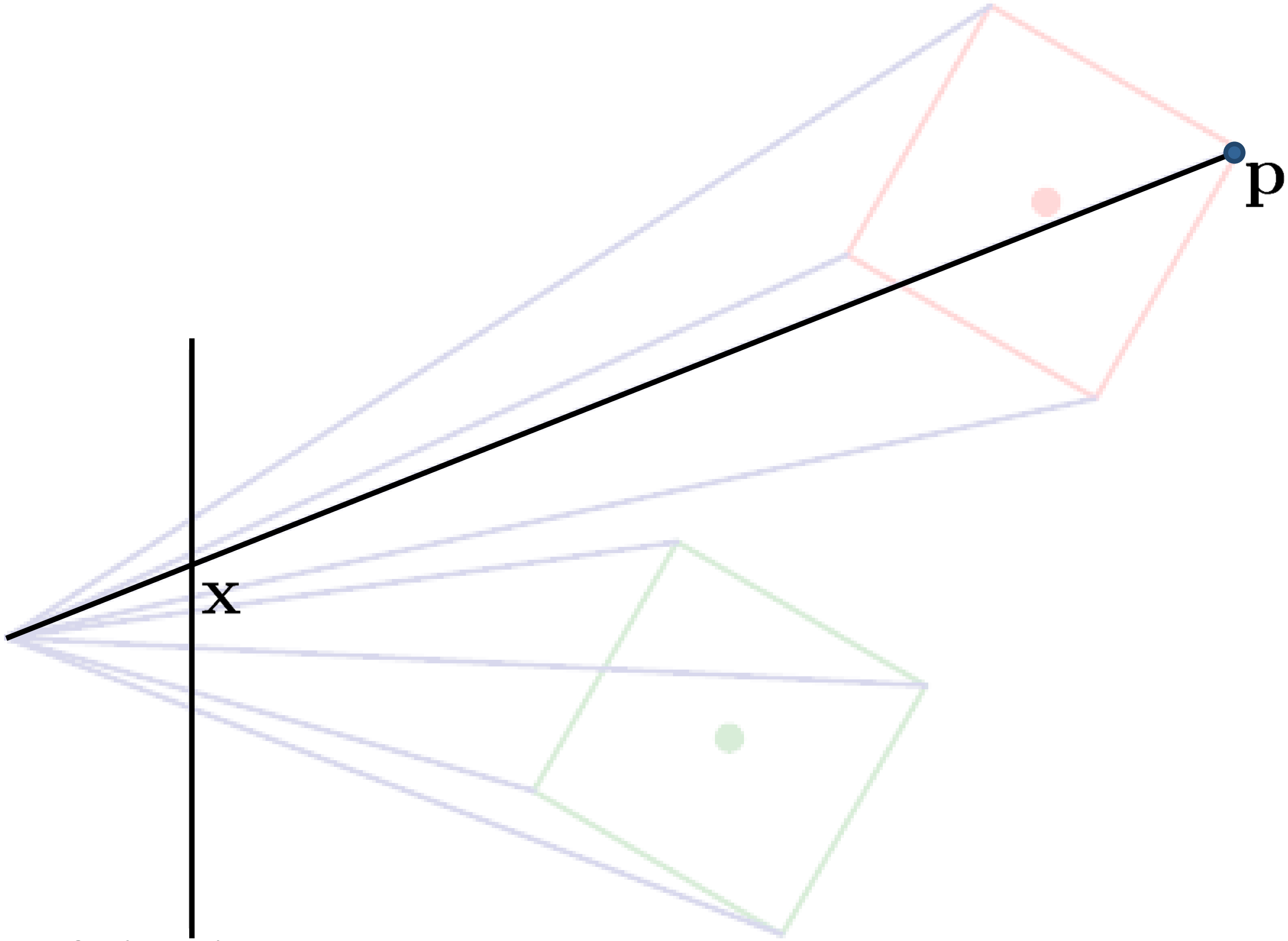
3D to 2D Projections

- Orthographic projections
- Perspective projections

3D to 2D Perspective Projection



3D to 2D Perspective Projection



3D to 2D Perspective Projection

- 3D point $\mathbf{p}$ (in the camera frame)
- 2D point $\mathbf{x}$ (on the image plane)
- Pin-hole camera model

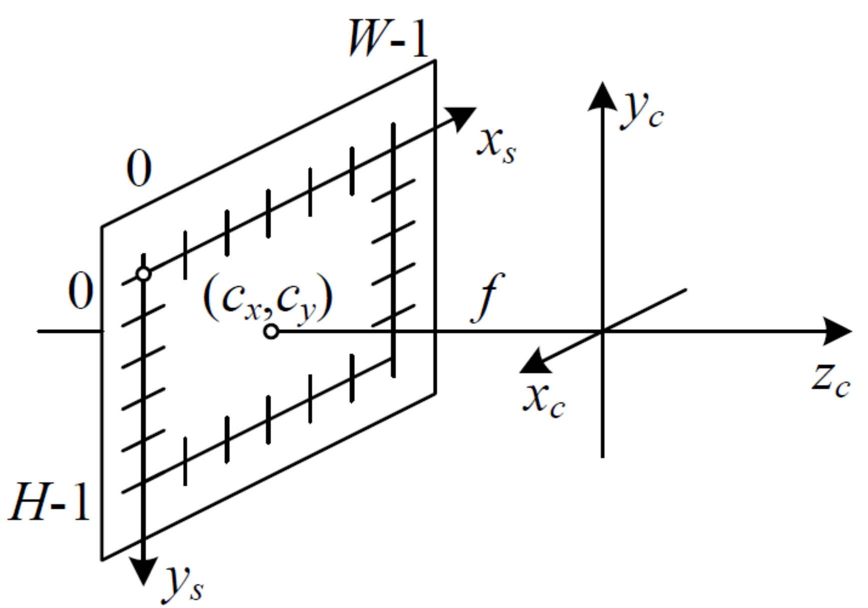
$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tilde{\mathbf{p}}$$

- Remember, $\tilde{\mathbf{x}}$ is homogeneous, need to normalize

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \end{pmatrix}$$

Camera Intrinsics

- So far, 2D point is given in meters on image plane
- But: we want 2D point be measured in pixels (as the sensor does)



Camera Intrinsics

- Need to apply some scaling/offset

$$\tilde{\mathbf{x}} = \underbrace{\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{intrinsics } K} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \tilde{\mathbf{p}}$$

- Focal length f_x, f_y
- Camera center c_x, c_y
- Skew s

Camera Extrinsics

- Assume $\tilde{\mathbf{p}}_w$ is given in world coordinates
- Transform from world to camera (also called the camera extrinsics)

$$\tilde{\mathbf{p}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \tilde{\mathbf{p}}_w$$

- Full camera matrix

$$\tilde{\mathbf{x}} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & \mathbf{t} \end{pmatrix} \tilde{\mathbf{p}}_w$$

Recap: 2D/3D Geometry

- Points, lines, planes
- 2D and 3D transformations
- Different representations for 3D orientations
  - Choice depends on application
  - Which representations do you remember?
- 3D to 2D perspective projections

- You **really** have to know 2D/3D transformations by heart (read Szeliski, Chapter 2)

Sensors

Sensors

- Tactile sensors
  Contact switches, bumpers, proximity sensors, pressure
- Wheel/motor sensors
  Potentiometers, brush/optical/magnetic/inductive/capacitive encoders, current sensors
- Heading sensors
  Compass, infrared, inclinometers, gyroscopes, accelerometers
- Ground-based beacons
  GPS, optical or RF beacons, reflective beacons
- Active ranging
  Ultrasonic sensor, laser rangefinder, optical triangulation, structured light
- Motion/speed sensors
  Doppler radar, Doppler sound
- Vision-based sensors
  CCD/CMOS cameras, visual servoing packages, object tracking packages

Example: Ardrone Sensors

- Tactile sensors
  Contact switches, bumpers, proximity sensors, pressure
- Wheel/motor sensors
  Potentiometers, brush/optical/magnetic/inductive/capacitive encoders, **current sensors**
- Heading sensors
  Compass, infrared, inclinometers, **gyroscopes**, **accelerometers**
- Ground-based beacons
  GPS, **optical** or RF **beacons**, reflective beacons
- Active ranging
  Ultrasonic sensor, laser rangefinder, optical triangulation, structured light
- Motion/speed sensors
  Doppler radar, Doppler sound
- Vision-based sensors
  CCD/**CMOS cameras**, **visual servoing packages**, object tracking packages

Characterization of Sensor Performance

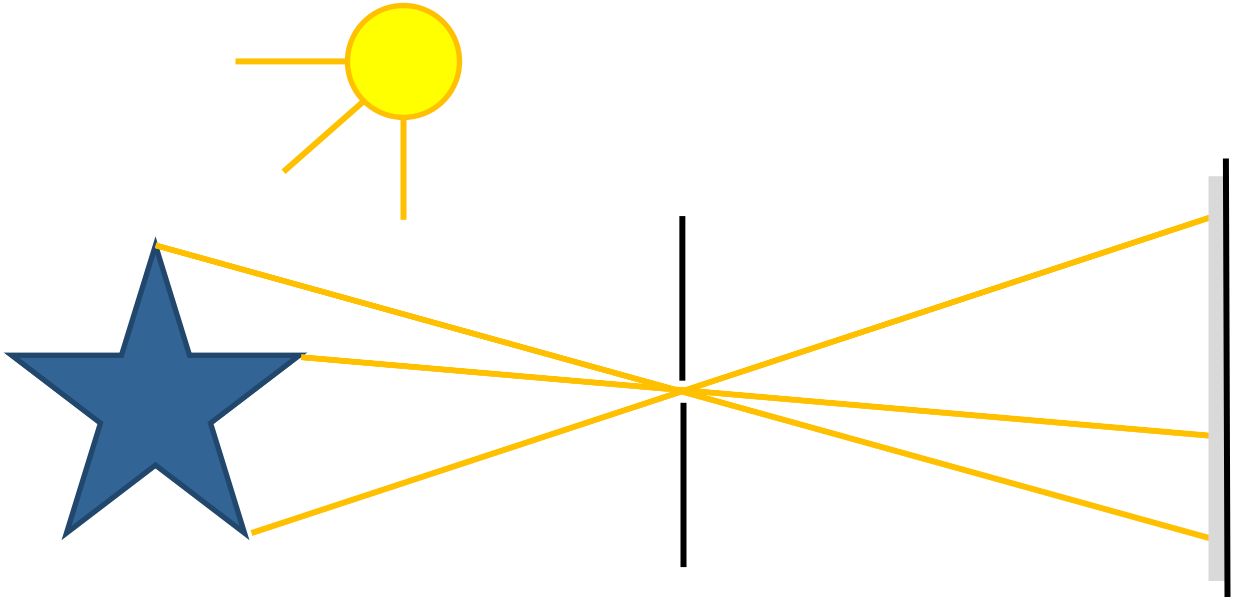
- Bandwidth or Frequency
- Delay
- Sensitivity
- Cross-sensitivity (cross-talk)
- Error (accuracy)
  - Deterministic errors (modeling/calibration possible)
  - Random errors
- Weight, power consumption, …

Let’s Have a Closer Look

- Cameras
- Gyroscope
- Accelerometers
- GPS
- Range sensors

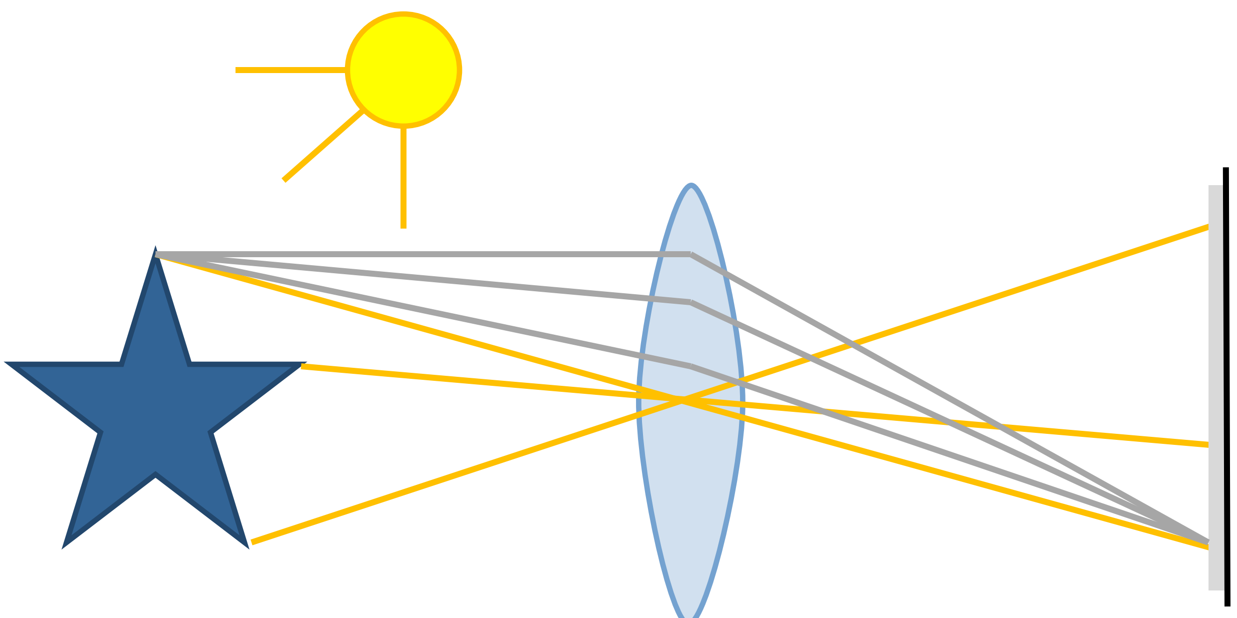
Pinhole Camera

- Lit scene emits light
- Film/sensor is light sensitive



Lens Camera

- Lit scene emits light
- Film/sensor is light sensitive
- A lens focuses rays onto the film/sensor



Real Cameras

- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens

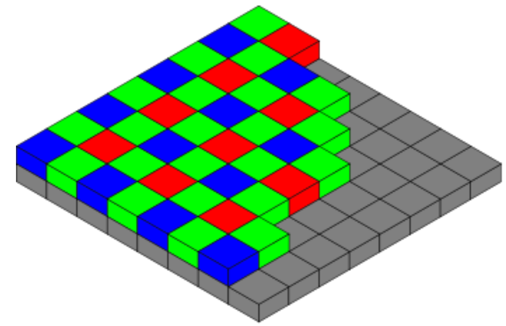
Radial Distortion

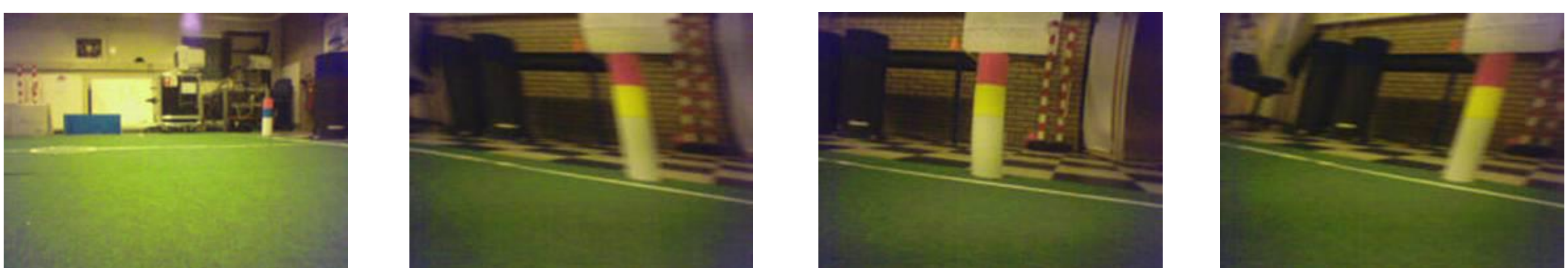
- Radial distortion of the image
  - Caused by imperfect lenses
  - Deviations are most noticeable for rays that pass through the edge of the lens
- Typically compensated with a low-order polynomial

$$\hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$
$$\hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)$$

Digital Cameras

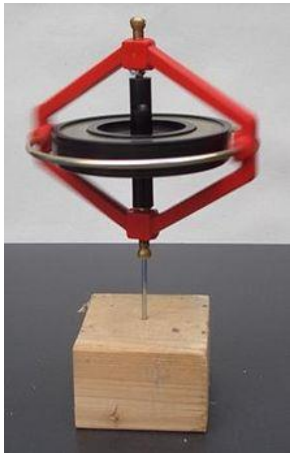
- Vignetting
- De-bayering
- Rolling shutter and motion blur
- Compression (JPG)
- Noise

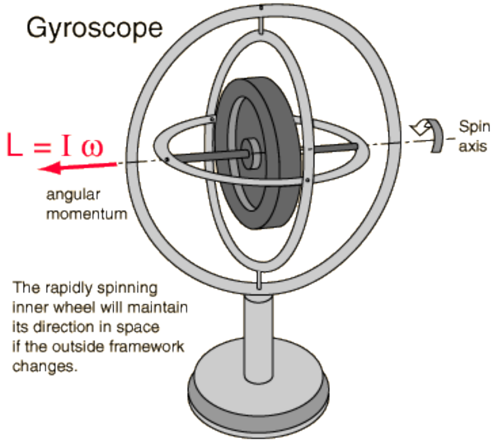




Mechanical Gyroscope

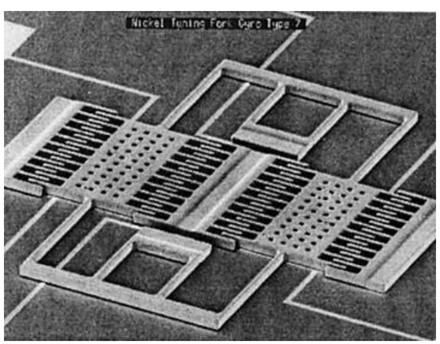
- Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)
- Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)
- Wheel keeps orientation due to angular momentum (standard gyro)

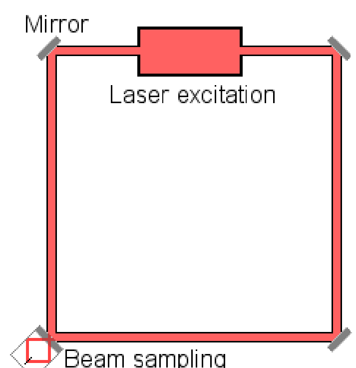


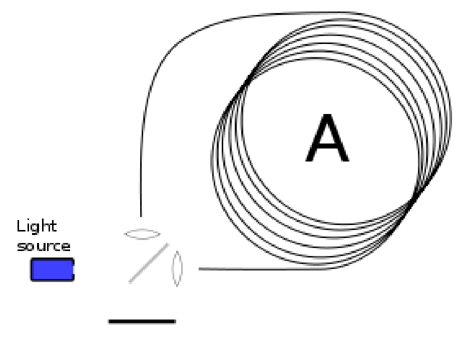


Modern Gyroscopes

- Vibrating structure gyroscope (MEMS)
  - Based on Coriolis effect
  - “Vibration keeps its direction under rotation”
  - Implementations: Tuning fork, vibrating wheels, …
- Ring laser / fibre optic gyro
  - Interference between counter-propagating beams in response to rotation







Accelerometer

- Measures all external forces acting upon them (including gravity)
- Acts like a spring-damper system
- To obtain inertial acceleration (due to motion alone), gravity must be subtracted

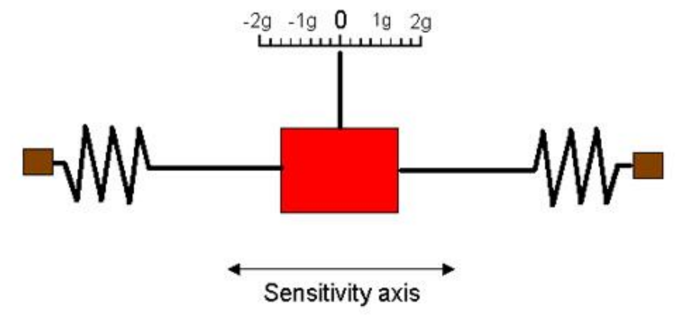
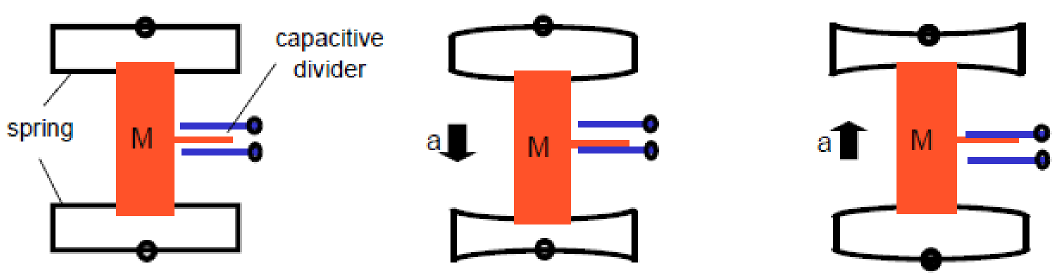
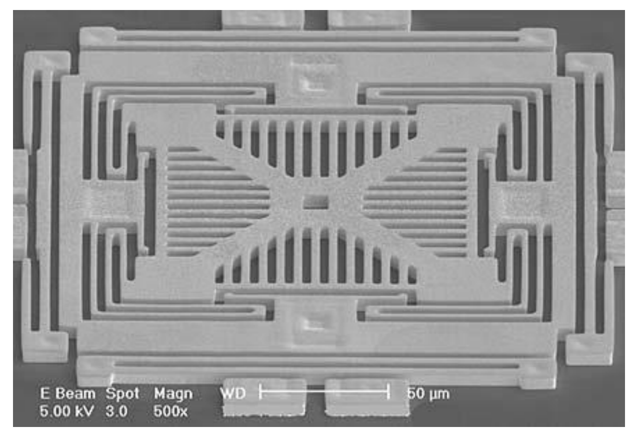


Fig. 1. Schematic structure of an accelerometer

MEMS Accelerometers

- Micro Electro-Mechanical Systems (MEMS)
- Spring-like structure with a proof mass
- Damping results from residual gas
- Implementations: capacitive, piezoelectric, …

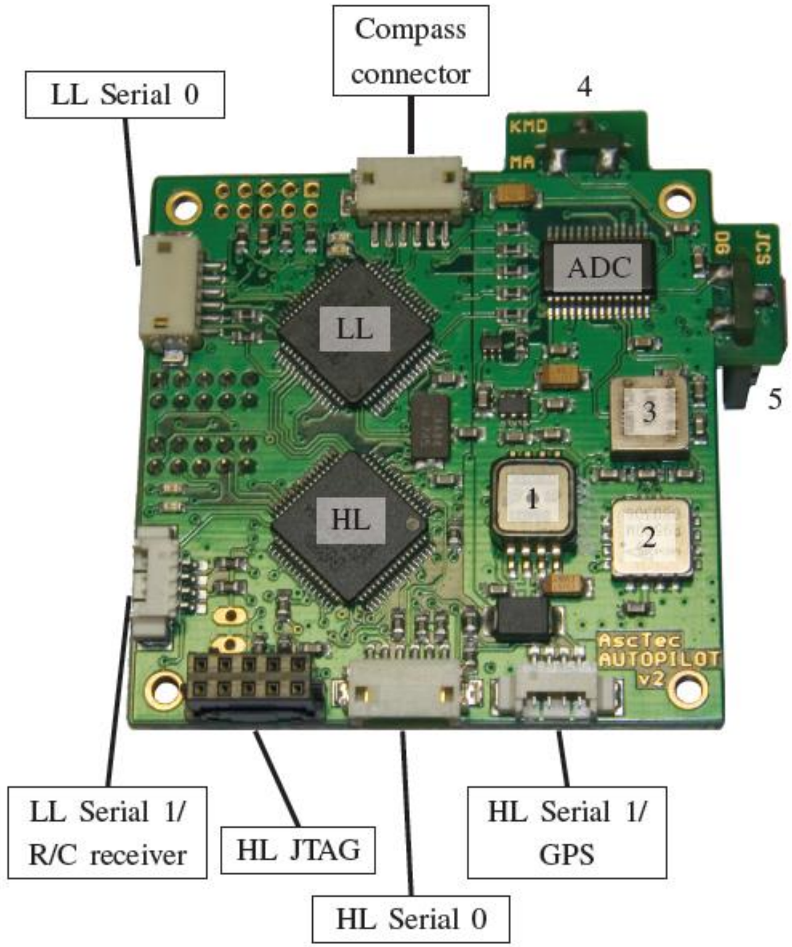




Inertial Measurement Unit

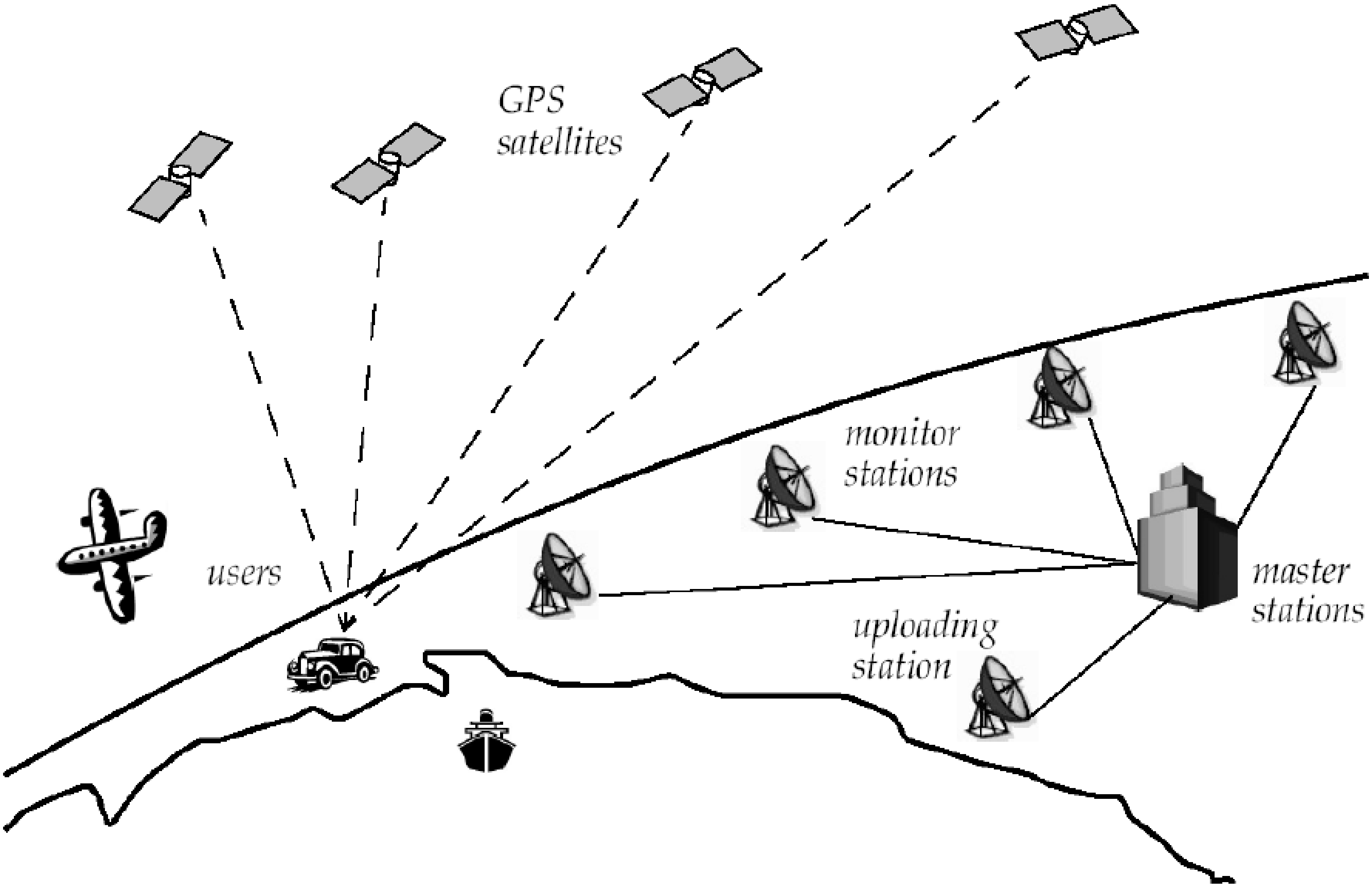
- 3-axes MEMS gyroscope
  - Provides angular velocity
  - Integrate for angular position
  - Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias
- 3-axes MEMS accelerometer
  - Provides accelerations (including gravity)
- Can we use these sensors to estimate our position?

Example: AscTec Autopilot Board



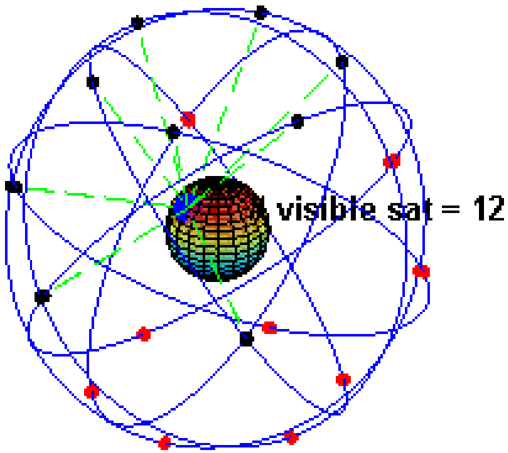
1:	pressure sensor	3:	yaw gyro
2:	acceleration sensor	4:	nick gyro
		5:	roll gyro

GPS


GPS satellites
monitor stations
users
master stations
uploading station

GPS

- 24+ satellites, 12 hour orbit, 20.190 km height
- 6 orbital planes, 4+ satellites per orbit, 60deg distance



- Satellite transmits orbital location + time
- 50bits/s, msg has 1500 bits → 12.5 minutes

GPS

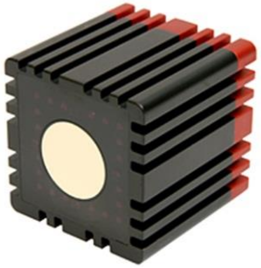
- Position from pseudorange
  - Requires measurements of 4 different satellites
  - Low accuracy (3-15m) but absolute
- Position from pseudorange + phase shift
  - Very precise (1mm) but highly ambiguous
  - Requires reference receiver (RTK/dGPS) to remove ambiguities

Range Sensors

- **Sonar**

- **Laser range finder**

- Time of flight camera



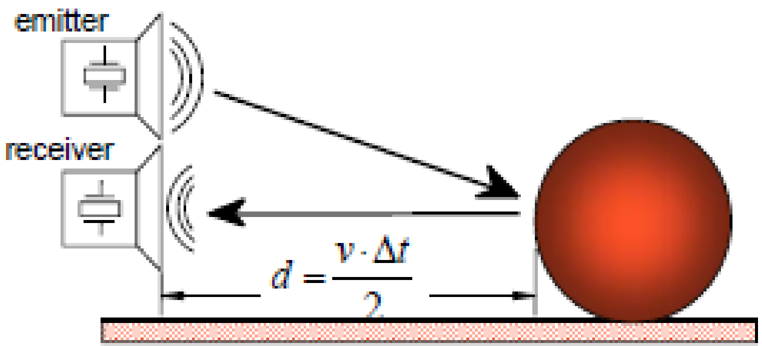
- Structured light
  (will be covered later)

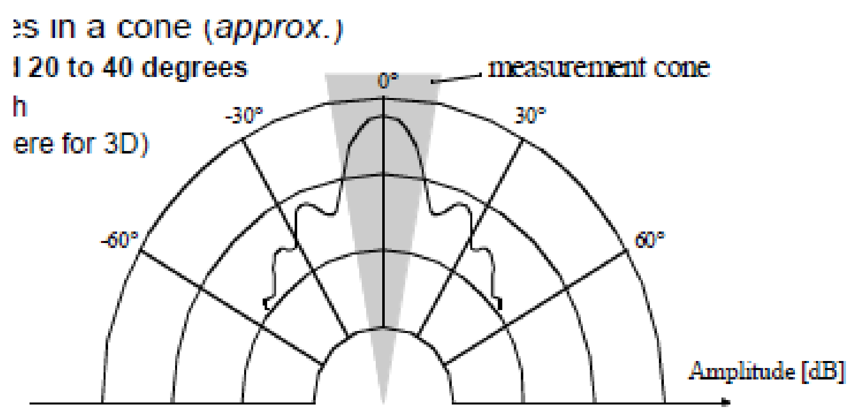
Range Sensors

- Emit signal to determine distance along a ray
- Make use of propagation speed of ultrasound/light
- Traveled distance is given by $d = c \cdot t$
- Sound speed: 340m/s
- Light speed: 300.000km/s

Ultrasonic Range Sensors

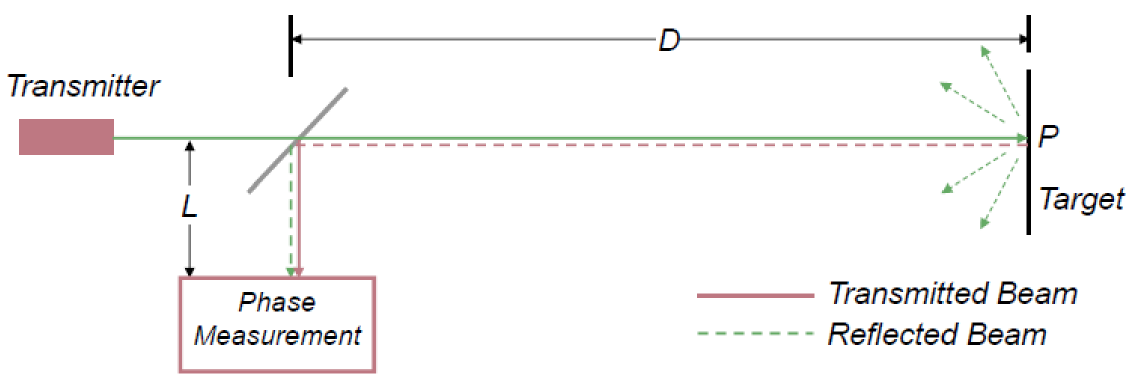
- Range between 12cm and 5m
- Opening angle around 20 to 40 degrees
- Soft surfaces absorb sound
- Reflections → ghosts
- Lightweight and cheap

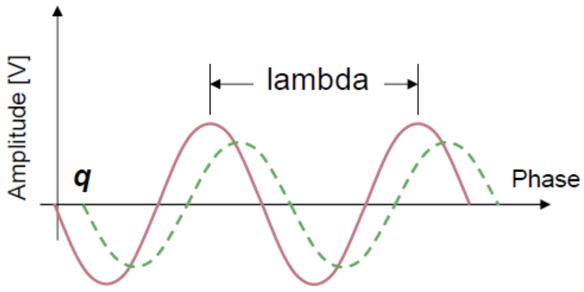




Laser Scanner

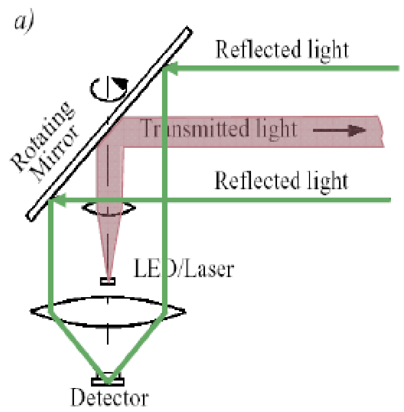
- Measures phase shift
- Pro: High precision, wide field of view, safety approved for collision detection
- Con: Relatively expensive + heavy



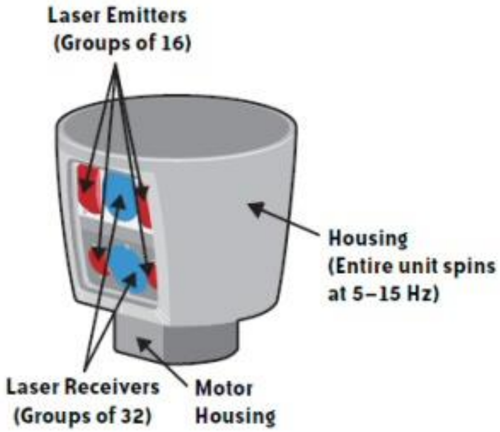


Laser Scanner

- 2D scanners



- 3D scanners



Exercise Sheet 1

Coordinate Systems

- The pose of a robot can be described by 6 parameters:
  - Three-dimensional Cartesian coordinates
  - Three Euler angles roll, pitch, yaw.
- The state space of such a system is six-dimensional

$$\mathbf{x}_t = (x, y, z, \phi, \theta, \psi)^\top$$

- Robot makes sensor observations usually in its ego-centric frame (as seen by the robot)

Odometry Motion Model

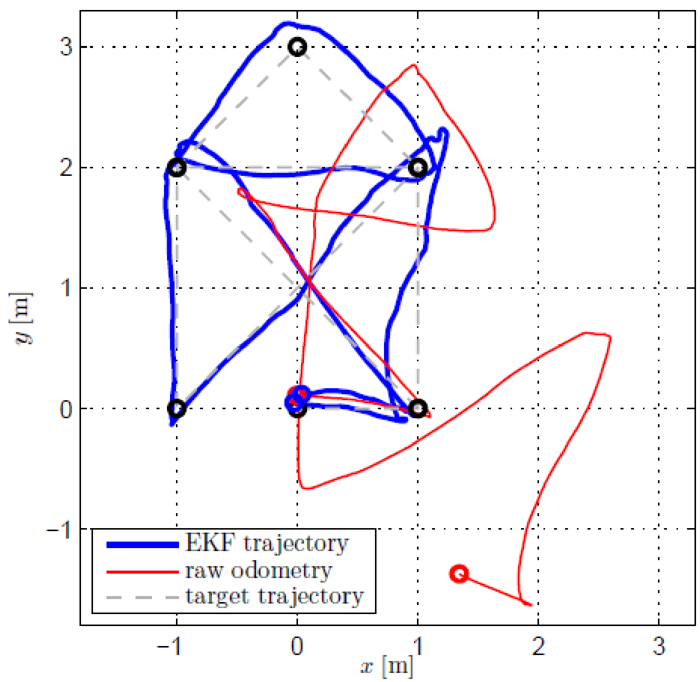
- In practice, one often finds two types of motion models:
  - **Odometry-based**
  - **Velocity-based** (**dead reckoning**)
- Odometry-based models are used when systems are equipped with distance sensors (e.g., wheel encoders).
- Velocity-based models have to be applied when no wheel encoders are given.

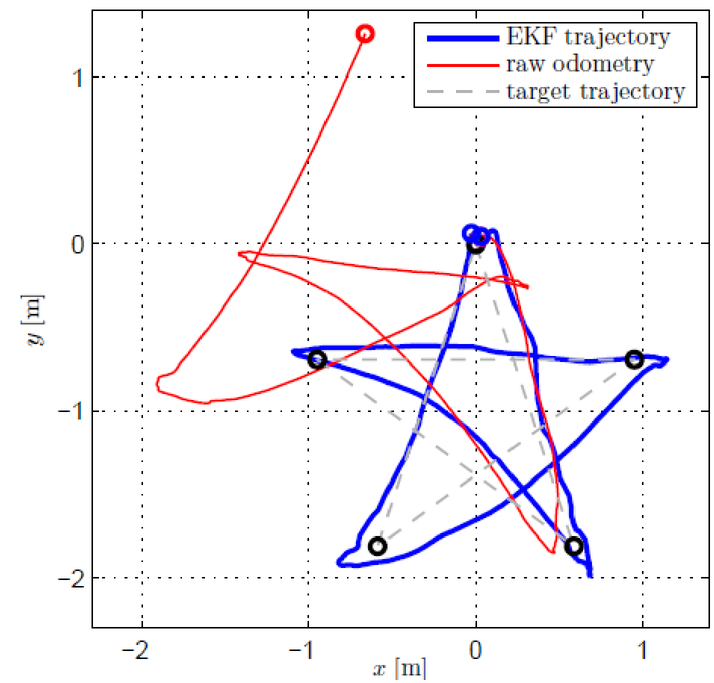
Dead Reckoning

- Mathematical procedure to determine the present location of a vehicle
- Achieved by calculating the current pose of the vehicle based on its velocities and the elapsed time

Dead Reckoning

- Estimating the position $\mathbf{x}_t$ based on the issued controls (or IMU readings) $\mathbf{u}_t$
- Integrate over time $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t)$





Exercise Sheet 1

- Odometry sensor on Ardrone is an integrated package
- Sensors
  - Down-looking camera to estimate motion
  - Ultrasonic sensor to get height
  - 3-axes gyroscopes
  - 3-axes accelerometer
- IMU readings $\mathbf{u}_t$ (in provided bag file)
  - Horizontal speed (vx/vy) in its local frame (!)
  - Height (z) in the global frame
  - Roll, Pitch, Yaw in the global frame
- Integrate these values to get robot pose $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t)$
  - Position (x/y/z) in the global frame
  - Orientation (e.g., r/p/y) in the global frame

Lessons Learned Today

- Linear algebra
- 2D/3D geometry
- Sensors
- Exercise sheet 1: Robot odometry
  - Due next Tuesday, 10am
  - Hand in via email to visnav2013@vision.in.tum.de