Visual Navigation for Flying Robots

3D Geometry and Sensors

Dr. Jürgen Sturm
Organization: Lecture

- Student request to change lecture time to Tuesday afternoon due to time conflicts with other course
- Problem: At least 3 students who are enrolled for this lecture have time Tuesday morning but not on Tuesday afternoon
- Therefore: No change
- Lectures are important, please choose which course to follow
- Note: Still students on the waiting list
Organization: Lab Course

- Robot lab: room 02.09.38 (around the corner)
- Exercises: room 02.09.23 (here)
- You have to sign up for a team before May 1st (team list in student lab)
- After May 1st, remaining places will be given to students on waiting list
- This Thursday: Visual navigation demo at 2pm in the student lab (in conjunction with TUM Girls’ Day)
Today’s Agenda

- Linear algebra
- 2D and 3D geometry
- Sensors
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

\[ \hat{x} = \frac{x}{||x||} \]
Vector Operations

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

\[ x \cdot y = ||x|| ||y|| \cos \theta \]

\( x, y \) are orthogonal if \( x \cdot y = 0 \)

\( y \) is linearly dependent from \( \{x_1, x_2, \ldots \} \) if

\[ y = \sum_{i} k_i x_i \]
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**
  \[ \mathbf{x} \times \mathbf{y} = \begin{pmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{pmatrix} \]

- **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

\[
X = \begin{pmatrix}
    x_{11} & x_{12} & \cdots & x_{1m} \\
    x_{21} & x_{22} & \cdots & x_{2m} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{n1} & x_{n2} & \cdots & x_{nm}
\end{pmatrix}, \quad \in \mathbb{R}^{n \times m}
\]

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

- Column vectors of a matrix

\[
X = \begin{pmatrix}
  x_{11} & x_{12} & \cdots & x_{1m} \\
  x_{21} & x_{22} & \cdots & x_{2m} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{n1} & x_{n2} & \cdots & x_{nm}
\end{pmatrix}
\]

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

- **Row vectors of a matrix**

\[
X = \begin{pmatrix}
     x_{11} & x_{12} & \cdots & x_{1m} \\
x_{21} & x_{22} & \cdots & x_{2m} \\
     \vdots \\
x_{n1} & x_{n2} & \cdots & x_{nm}
\end{pmatrix} = \begin{pmatrix}
x_{1*}^T \\
x_{2*}^T \\
     \vdots \\
x_{n*}^T
\end{pmatrix}
\]
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 \( X = -X^\top \)
- (Semi-)positive definite matrix
- Invertible matrix
- Orthonormal matrix
- Matrix rank

\[
\begin{pmatrix}
0 & -\omega_3 & \omega_2 \\
\omega_3 & 0 & -\omega_1 \\
-\omega_2 & \omega_1 & 0
\end{pmatrix}
\]
Matrix Operations

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

- Scalar multiplication
- Addition/subtraction
- Transposition
- **Matrix-vector multiplication** \( Xb \)
- Matrix-matrix multiplication
- Inversion
Matrix-Vector Multiplication

- **Definition**

\[
X \cdot b = \begin{pmatrix}
x_{11} & x_{12} & \cdots & x_{1m} \\
x_{21} & x_{22} & \cdots & x_{2m} \\
\vdots & \vdots & \ddots & \vdots \\
x_{n1} & x_{n2} & \cdots & x_{nm}
\end{pmatrix}
\begin{pmatrix}
b_1 \\
b_2 \\
\vdots \\
b_m
\end{pmatrix} = \sum_{k=1}^{n} x_{\cdot k} \cdot b_k
\]

- **Geometric interpretation:**
a linear combination of the columns of X scaled by the coefficients of b
Matrix-Vector Multiplication

\[ X \cdot b = \sum_{k=1}^{n} x_\ast k \cdot b_k \]

column vectors

- Geometric interpretation:
  A linear combination of the columns of \( A \) scaled by the coefficients of \( b \)
  \( \rightarrow \) coordinate transformation
Matrix Operations

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

- **Operator** \( \mathbb{R}^{n \times m} \times \mathbb{R}^{m \times p} \rightarrow \mathbb{R}^{n \times p} \)

- **Definition**
  \[
  C = AB
  = A \begin{pmatrix} b_{1}^* & b_{2}^* & \cdots & b_{p}^* \end{pmatrix}
  \]

- **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
  \[ \tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3 \]

- 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

\[
\tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{w}\bar{x}
\]

- Points with \( \tilde{w} = 0 \) are called points at infinity or ideal points
Geometric Primitives in 2D

- 2D line

\[ \vec{l} = (a, b, c)^\top \]

- 2D line equation

\[ \vec{x} \cdot \vec{l} = ax + by + c = 0 \]
Geometric Primitives in 2D

- Normalized line equation vector

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

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)

\[
\hat{n} = (\cos \theta, \sin \theta)^\top
\]
Geometric Primitives in 2D

- Line joining two points: $\tilde{l} = \tilde{x}_1 \times \tilde{x}_2$
- Intersection point of two lines: $\tilde{x} = \tilde{l}_1 \times \tilde{l}_2$
Geometric Primitives in 3D

- **3D point**
  
  (same as before)

\[ \mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3 \]

- **Augmented vector**

\[ \tilde{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4 \]

- **Homogeneous coordinates**

\[ \hat{\mathbf{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \\ \hat{w} \end{pmatrix} \in \mathbb{P}^3 \]
Geometric Primitives in 3D

- 3D plane
  \[ \tilde{m} = (a, b, c, d)^\top \]

- 3D plane equation
  \[ \bar{x} \cdot \tilde{m} = ax + by + cz + d = 0 \]

- Normalized plane with unit normal vector
  \[ m = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{n}, d) \]
  \[ (\|\hat{n}\| = 1) \]
  and distance \( d \)
Geometric Primitives in 3D

- 3D line \( r = (1 - \lambda)p + \lambda q \) through points \( p, q \)

- Infinite line:
  \[ \lambda \in \mathbb{R} \]

- Line segment joining \( p, q \):
  \[ 0 \leq \lambda \leq 1 \]
2D Planar Transformations
2D Transformations

- **Translation**

\[ x' = x + t \]

\[ x' = \begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix} \bar{x} \]

\[ \bar{x}' = \begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{x} \]

where \( \mathbf{I} \) is the identity matrix (2x2) and \( \mathbf{0} \) is the zero vector.
2D Transformations

- Rotation + translation (2D rigid body motion, or 2D Euclidean transformation)

\[ x' = R x + t \quad \text{or} \quad \bar{x}' = \begin{pmatrix} R & t \\ 0^\top & 1 \end{pmatrix} \bar{x} \]

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

is an orthonormal rotation matrix, i.e., \( RR^\top = I \)

- Distances (and angles) are preserved
2D Transformations

- Scaled rotation/similarity transform

\[ x' = sRx + t \quad \text{or} \quad \bar{x}' = \begin{pmatrix} sR & t \\ 0^\top & 1 \end{pmatrix} \bar{x} \]

- Preserves angles between lines
2D Transformations

- Affine transform

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

- Parallel lines remain parallel
2D Transformations

- Projective/perspective transform

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

- Note that \( \tilde{H} \) is homogeneous (only defined up to scale)

- Resulting coordinates are homogeneous

- Parallel lines remain parallel
## 2D Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
<th># DoF</th>
<th>Preserves</th>
<th>Icon</th>
</tr>
</thead>
</table>
| translation          | \[
\begin{bmatrix} I & t \end{bmatrix}_{2\times3}\] | 2     | orientation  |      |
| rigid (Euclidean)    | \[
\begin{bmatrix} R & t \end{bmatrix}_{2\times3}\] | 3     | lengths      |      |
| similarity           | \[
\begin{bmatrix} sR & t \end{bmatrix}_{2\times3}\] | 4     | angles       |      |
| affine               | \[
\begin{bmatrix} A \end{bmatrix}_{2\times3}\] | 6     | parallelism  |      |
| projective           | \[
\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times3}\] | 8     | straight lines |    |
3D Transformations

- Translation

\[ \begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x} \end{pmatrix} \]

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

\[ \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x} \end{pmatrix} \]

- Scaled rotation, affine transform, projective transform...
# 3D Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Matrix</th>
<th># DoF</th>
<th>Preserves</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>$\begin{bmatrix}1 &amp;</td>
<td>t \end{bmatrix}_{3\times4}$</td>
<td>3</td>
<td>orientation</td>
</tr>
<tr>
<td>rigid (Euclidean)</td>
<td>$\begin{bmatrix}R &amp;</td>
<td>t \end{bmatrix}_{3\times4}$</td>
<td>6</td>
<td>lengths</td>
</tr>
<tr>
<td>similarity</td>
<td>$\begin{bmatrix}sR &amp;</td>
<td>t \end{bmatrix}_{3\times4}$</td>
<td>7</td>
<td>angles</td>
</tr>
<tr>
<td>affine</td>
<td>$\begin{bmatrix}A \end{bmatrix}_{3\times4}$</td>
<td>12</td>
<td>parallelism</td>
<td></td>
</tr>
<tr>
<td>projective</td>
<td>$\begin{bmatrix}\tilde{H} \end{bmatrix}_{4\times4}$</td>
<td>15</td>
<td>straight lines</td>
<td></td>
</tr>
</tbody>
</table>
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 \( R \in SO(3) \)
- Main disadvantage: Over-parameterized (9 parameters instead of 3)
Euler Angles

- Product of 3 consecutive rotations
- Roll-pitch-yaw convention is very common in aerial navigation (DIN 9300)
Euler Angles

- **Yaw \(\Psi\), Pitch \(\Theta\), Roll \(\Phi\) to rotation matrix**

\[
R = R_Z(\Psi)R_Y(\Theta)R_X(\Phi)
\]

\[
= \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos \Phi & \sin \Phi \\
0 & -\sin \Phi & \cos \Phi
\end{bmatrix}
\begin{bmatrix}
\cos \Theta & 0 & -\sin \Theta \\
0 & 1 & 0 \\
\sin \Theta & 0 & \cos \Theta
\end{bmatrix}
\begin{bmatrix}
\cos \Psi & \sin \Psi & 0 \\
-\sin \Psi & \cos \Psi & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos \Theta \cos \Psi & \cos \Theta \sin \Psi & -\sin \Theta \\
\sin \Phi \sin \Theta \cos \Psi - \cos \Phi \sin \Psi & \sin \Phi \sin \Theta \sin \Psi + \cos \Phi \cos \Psi & \sin \Phi \cos \Theta \\
\cos \Phi \sin \Theta \cos \Psi + \sin \Phi \sin \Psi & \cos \Phi \sin \Theta \sin \Psi - \sin \Phi \cos \Psi & \cos \Phi \cos \Theta
\end{bmatrix}
\]

- **Rotation matrix to Yaw-Pitch-Roll**

\[
\phi = \text{Atan2} \left( -r_{31}, \sqrt{r_{11}^2 + r_{21}^2} \right)
\]

\[
\psi = -\text{Atan2} \left( \frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)} \right)
\]

\[
\theta = \text{Atan2} \left( \frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)} \right)
\]
Euler Angles

Advantage:
- Minimal representation (3 parameters)
- Easy interpretation

Disadvantages:
- Many “alternative” Euler representations exist (XYZ, ZXZ, ZYX, ...)
- Singularities (gimbal lock)
Gimbal Lock

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

1. Rotations in Euler angles can be defined like gimbal system with three circles
2. When all three circles are lined up, the whole system can only move in two dimensions from this configuration, this is a gimbal lock
3. Usage of quaternions can help to avoid such situations
Axis/Angle

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

- Assume we have a rotational motion in SO(3)
  \[ R(t) \in SO(3) \quad t \in \mathbb{R} \]

- As this rotations are orthonormal matrices, we have
  \[ R(t)R^\top(t) = I \]

- Now take the derivative on both sides (w.r.t. \( t \))
  \[ \dot{R}(t)R^\top(t) + R(t)\dot{R}^\top(t) = 0 \]
  \[ \dot{R}(t)R^\top(t) = -(\dot{R}(t)R^\top(t))^\top \]

- Thus, \( \dot{R}(t)R^\top(t) \) must be skew-symmetric, i.e.,
  \[ [\omega(t)]_\times = \dot{R}(t)R^\top(t) \]
Derivation of Angular Velocities

Linear ordinary differential equation (ODE)

\[ \dot{R}(t) = [\omega]_\times R(t) = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} R(t) \]

- Solution of this ODE

\[ R(t) = \exp([\omega]_\times)R(0) \]

- Conversions

\[ R = \exp([\omega]_\times) \quad [\omega]_\times = \log R \]
Derivation of Angular Velocities

→ Linear ordinary differential equation (ODE)

\[ \dot{R}(t) = [\omega]_\times R(t) = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix} R(t) \]

- The space of all skew-symmetric matrices is called the tangent space

\[ \text{so}(3) = \{ [\omega]_\times \in \mathbb{R}^{3\times3} \mid \omega \in \mathbb{R}^3 \} \]

- Space of all rotations in 3D (Special orientation group)

\[ \text{SO}(3) = \{ R \in \mathbb{R}^{3\times3} \mid R^\top R = I, \det R = 1 \} \]
Conversion

- Rodriguez’ formula

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

- Inverse

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

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

- The exponential map can be generalized to Euclidean transformations (incl. translations)

- Tangent space \( \text{se}(3) = \text{so}(3) \times \mathbb{R}^3 \)

- (Special) Euclidean group \( \text{SE}(3) = \text{SO}(3) \times \mathbb{R}^3 \)
  (group of all Euclidean transforms)

- Rigid body velocity

\[
\xi = \left( \omega_x, \omega_y, \omega_z , v_x, v_y, v_z \right) \in \mathbb{R}^6
\]

  angular vel.  linear vel.
Exponential Twist

- Convert to homogeneous coordinates

\[ \hat{\xi} = \begin{pmatrix}
0 & -\omega_z & \omega_y & \nu_x \\
\omega_z & 0 & -\omega_x & \nu_y \\
-\omega_y & \omega_x & 0 & \nu_z \\
0 & 0 & 0 & 0
\end{pmatrix} \in \mathfrak{se}(3) \]

- Exponential map between \( \mathfrak{se}(3) \) and \( \text{SE}(3) \)

\[ M = \exp \hat{\xi} \quad \hat{\xi} = \log M \]

- There are also direct formulas (similar to Rodriguez)
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 really fast
- Quaternion-Quaternion Multiplication

\[ q_0q_1 = (v_0, w_0)(v_1, w_1) \]
\[ = (v_0 \times v_1 + w_0v_1 + w_1v_0, w_0w_1 - v_0v_1) \]

- Inverse (flip sign of v or w)

\[ q_0/q_1 = (v_0, w_0)/(v_1, w_1) \]
\[ = (v_0, w_0)(v_1, -w_1) \]
\[ = (v_0 \times v_1 + w_0v_1 - w_1v_0, -w_0w_1 - v_0v_1) \]
Unit Quaternions

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

$$p' = v\bar{p}/q$$

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

- Relation to Axis/Angle representation

$$q = (v, w) = (\sin \frac{\theta}{2} \hat{n}, \cos \frac{\theta}{2})$$
Spherical Linear Interpolation (SLERP)

- Useful for interpolating between two rotations

**procedure** \( \text{slerp}(q_0, q_1, \alpha) : \)

1. \( q_r = q_1 / q_0 = (v_r, w_r) \)
2. if \( w_r < 0 \) then \( q_r \leftarrow -q_r \)
3. \( \theta_r = 2 \tan^{-1}(\|v_r\|/w_r) \)
4. \( \hat{n}_r = \mathcal{N}(v_r) = v_r / \|v_r\| \)
5. \( \theta_\alpha = \alpha \theta_r \)
6. \( q_\alpha = (\sin \frac{\theta_\alpha}{2} \hat{n}_r, \cos \frac{\theta_\alpha}{2}) \)
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

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

\[
\tilde{x} = \begin{pmatrix}
    f_x & s & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0
\end{pmatrix}
\tilde{p}
\]

- Focal length \( f_x, f_y \)
- Camera center \( c_x, c_y \)
- Skew \( s \)
Camera Extrinsics

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

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

- Full camera matrix

$$
\tilde{x} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R & t \end{pmatrix} \tilde{p}_w
$$
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)
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.
Example: Video from PR2
Sensors
Classification of Sensors

What:
- Proprioceptive sensors
  - Measure values internally to the system (robot)
  - Examples: battery status, motor speed, accelerations, ...
- Exteroceptive sensors
  - Provide information about the environment
  - Examples: compass, distance to objects, ...

How:
- Passive sensors
  - Measure energy coming from the environment
- Active sensors
  - Emit their proper energy and measure the reaction
  - Better performance, but influence on environment
Classification of 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, ...
Sensors

- Motor/wheel encoders
- Compass
- Gyroscope
- Accelerometers
- GPS
- Range sensors
- Cameras
Motor/wheel encoders

- Device for measuring angular motion
- Often used in (wheeled) robots
- Output: position, speed (possibly integrate speed to get odometry)
Motor/wheel encoders

- **Working principle:**
  - Regular: counts the number of transitions but cannot tell direction
  - Quadrature: uses two sensors in quadrature phase-shift, ordering of rising edge tells direction
  - Sometimes: Reference pulse (or zero switch)
Magnetic Compass

- Measures earth’s magnetic field
- Inclination angle approx. 60deg (Germany)
- Does not work indoor/affected by metal
- Alternative: gyro compass (spinning wheel, aligns with earth’s rotational poles, for ships)
Magnetic Declination

- Angle between magnetic north and true north
- Varies over time
- Good news ;-) : by 2050, magnetic declination in central Europe will be zero
Magnetic Compass

- Sensing principle: Hall sensor
- Construction: 3 orthogonal sensors
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?
Inertial Measurement Unit

- IMU: Device that uses gyroscopes and accelerometers to estimate (relative) position, orientation, velocity and accelerations
- Integrate angular velocities to obtain absolute orientation
- Subtract gravity from acceleration
- Integrate acceleration to linear velocities
- Integrate linear velocities to position
- Note: All IMUs are subject to drift (position is integrated twice!), needs external reference
Example: AscTec Autopilot Board

- LL Serial 0
- Compass connector
- LL Serial 1/ R/C receiver
- HL JTAG
- HL Serial 1/ GPS
- HL Serial 0

1: pressure sensor
2: acceleration sensor
3: yaw gyro
4: nick gyro
5: roll gyro
GPS
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 $\rightarrow$ 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**
Camera

- Let’s design a camera
  - Idea 1: put a piece of film in front of an object
  - Do we get a reasonable image?
Camera

- Add a barrier to block off most of the rays
  - This reduces blurring
  - The opening known as the \textit{aperture}
  - How does this transform the image?
Camera Lens

- A lens focuses light onto the film
  - Rays passing through the optical center are not deviated
Camera Lens

- A lens focuses light onto the film
  - Rays passing through the center are not deviated
  - All rays parallel to the **Optical Axis** converge at the **Focal Point**
Camera Lens

- There is a specific distance at which objects are “in focus”
- Other points project to a “blur circle” in the image
Lens Distortions

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

- 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
Dead Reckoning and Odometry

- Estimating the position $x_t$ based on the issued controls (or IMU) readings $u_t$
- Integrated over time $x_t = f(x_{t-1}, u_t)$
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 $u_t$
  - Horizontal speed ($v_x/v_y$)
  - Height ($z$)
  - Roll, Pitch, Yaw
- Integrate these values to get robot pose $x_t = f(x_{t-1}, u_t)$
  - Position ($x/y/z$)
  - Orientation (e.g., r/p/y)
Summary

- Linear Algebra
- 2D/3D Geometry
- Sensors