Computer Vision 2 - Multiple View Geometry (IN2228)

Philipp Wulff
Technical University of Munich

Summer Semester 2023

This summary covers the lecture "Computer Vision 2 - Multiple View Geometry" by Prof. Daniel Cremers and Dr. Haoang Li. It builds on the summaries by Marcel Brucker and Franz Kaschner.

1 Mathematical Background: Linear Algebra & Calculus

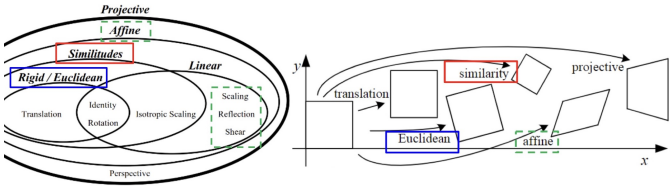
1.1 Set

A Set S of different elements e.g. the natural numbers $\mathbb{N} = \{1, 2, \dots\}$ or all real $n \times n$ matrices $\mathcal{M}(n)$.

Vector Space A set V is called a vector space over the field $\mathbb{R}$ if it is closed under

- vector summation: $+ : V \times V \to V$ and
- scalar multiplication: $\cdot : \mathbb{R} \times V \to V$

1.2 Transformations



Linear: Origin remains unchanged.
Affine: Linear with/without Translation.

Linear Transformations A linear transformation between two linear spaces is a map $L : V \to W$ with:

- $L(x + y) = L(x) + L(y), \quad \forall x, y \in V$
- $L(\alpha x) = \alpha L(x), \quad \forall x \in V, \alpha \in \mathbb{R}$

1.3 Linear Span and Linear Independence

Given a set of vectors $S = \{v_1, \dots, v_k\} \subset V$, the linear span of S is formed by all linear combinations:

$$\text{span}(S) = \left\{ v \in V \,\middle|\, v = \sum_{i=1}^{k} \alpha_i v_i \right\}$$

Linear independence holds if none of the vectors can be expressed as a linear combination of the remaining vectors. In $\mathbb{R}^3$, row reduce the matrix $S = \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{pmatrix}$ using *Gauss-Jordan elimination*. If there is an all-zero row, the set is linearly dependent. Alternatively, **check if the determinant** $\det(S) = 0$ **is zero**. If so, $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ are linearly independent. Linear dependence for a set of vectors holds if:

$$\sum_{i=1}^{n} \alpha_i \mathbf{v}_i = \mathbf{0} \quad \alpha_i \in \mathbb{R} \quad \text{(not all zero)}, \mathbf{v}_i \in \mathbb{R}^n$$

1.4 Basis

A set of vectors $B = \mathbf{v}_1, \dots, \mathbf{v}_n$ is called a basis of V if it is **linearly independent and spans the vector space** V. A basis is a maximal set of linearly independent vectors. Properties for two bases (B and B') of a linear space V:

- B and B' contain the same number of vectors, which is called the dimension of the space V.
- Any vector can be uniquely expressed as a linear combination of the basis vectors.
- All vectors can be expressed as linear combination of vectors of another basis.

The Gram–Schmidt process is a method for orthonormalizing a set of vectors $\{\mathbf{v}_1, ..., \mathbf{v}_n\}$:

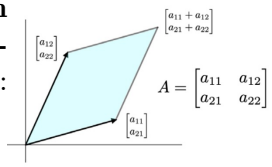
1. Compute the orthogonal basis:
   a) First basis vector as $\mathbf{u}_1 = \mathbf{v}_1$
   b) Then: $\mathbf{u}_k = \mathbf{v}_k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}_j}(\mathbf{v_k})$
   c) Stop for k independent basis vectors.
2. Normalize the orthogonal basis.

1.5 Matrix Properties

1.5.1 Determinant

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + bfg + cdh - ceg - bdi - afh$$

Geometrically, $|\det \mathbf{A}|$: is the **area of a parallelogram in 2D** and the **volume of a parallelepiped in 3D**. Properties:



- $\det(c\mathbf{A}) = c^n \det(\mathbf{A})$
- $\det(\mathbf{A}^T) = \det(\mathbf{A})$
- $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$
- $\det(\mathbf{A}^{-1}) = \det(\mathbf{A})^{-1}$

1.5.2 Range/Span

A matrix's range is the span of its column vectors:

$$\text{range}(\mathbf{A}) = \{\mathbf{y} \in \mathbb{R}^m \mid \exists \mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{y}\}$$

1.5.3 Rank

The rank of a matrix is the dimension of its range: $\text{rank}(\mathbf{A}) = \dim(\text{range}(\mathbf{A}))$. For $\mathbf{A} \in \mathbb{R}^{m \times n}$:

- $\text{rank}(\mathbf{A}) = n - \dim(\ker(\mathbf{A}))$
- $0 \leq \text{rank}(\mathbf{A}) \leq \min\{m, n\}$
- $\text{rank}(\mathbf{A})$ is equal to the maximum number of linearly independent rows or columns of $\mathbf{A}$.
- Sylvester's inequality ($B \in \mathbb{R}^{n \times n}$): $\text{rank}(A) + \text{rank}(\mathbf{B}) - n \leq \text{rank}(\mathbf{AB}) \leq \min\{\text{rank}(\mathbf{A}), \text{rank}(\mathbf{B})\}$
- For any invertible matrices $\mathbf{C} \in \mathbb{R}^{m \times m}$ and $\mathbf{D} \in \mathbb{R}^{n \times n}$, we have: $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{CAD})$.

1.5.4 Kernel/Null Space

The null space or kernel of a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ is the subset of vectors $\mathbf{x} \in \mathbb{R}^m$ which are mapped to zero:

$$\text{null}(\mathbf{A}) \equiv \ker(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^m \mid \mathbf{Ax} = \mathbf{0}\}$$

The LGS $\mathbf{Ax} = \mathbf{b}$ has a non-zero (non-trivial) solution iff $\mathbf{b} \in \text{range}(\mathbf{A})$. Properties:

- $\det(\mathbf{A}) \neq 0 \Leftrightarrow \text{rank}(\mathbf{A}) = n \Leftrightarrow \ker(\mathbf{A}) = \{0\}$
- $m = \text{rank}(\mathbf{A}) + k$, where k is the null space dimension and m is the number of unknowns.

1.5.5 Skew-symmetric Matrices

A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called skew-symmetric if $\mathbf{A}^T = -\mathbf{A}$ (meaning: $a_{ii} = 0$ and $a_{ij} = -a_{ji}$).

- All eigenvalues are all zero or purely imaginary.
- There exist an orthogonal matrix $\mathbf{V}$ and block-diagonal matrix $\mathbf{\Lambda}$ such that $\mathbf{A} = \mathbf{V\Lambda V}^T$.
- The rank of any skew-sym. matrix is even. A 3×3 skew-symmetric matrix has rank 2 or 0.

1.5.6 Eigenvalues and Eigenvectors

Let $\mathbf{A} \in \mathbb{C}^{n \times n}$ be a complex matrix. A non-zero vector $\mathbf{v} \in \mathbb{C}^n$ is called a (right) eigenvector of $\mathbf{A}$ if:

$$\mathbf{Av} = \lambda \mathbf{v} \text{ with } \lambda \in \mathbb{C}, \mathbf{v} \neq 0.$$

- The eigenvectors of a matrix $\mathbf{A}$ associated with different eigenvalues are linearly independent: $\lambda_a \neq \lambda_b \Rightarrow \mathbf{v_a}$ and $\mathbf{v_b}$ are linearly independent
- All eigenvalues $\sigma(\mathbf{A})$ are the roots of the characteristic polynomial $\det(\mathbf{A} - \lambda \mathbf{I}) = \mathbf{0}$.
- If $\mathbf{A} = \mathbf{PBP}^{-1}$ for some non-singular matrix $\mathbf{P}$, then $\sigma(\mathbf{B}) = \sigma(\mathbf{A})$.
- $\sigma(\mathbf{A}) = \sigma(\mathbf{A}^T)$.
- $\mathbf{A}$ is symmetric and PD (or $\mathbf{U} = \mathbf{V}$) $\Rightarrow$ Eigendecomposition and SVD are equivalent.
- *Geometrically*, eigenvectors are the vectors that are stretched, but not rotated by $\mathbf{A}$. E.g. for $\mathbf{A} \in \mathbb{R}^{2 \times 2}$, the eigenvectors are the two spatial directions in which every point slides in them.

Eigendecomposition ($\mathbf{A}, \mathbf{V}, \mathbf{\Lambda} \in \mathbb{R}^{n \times n}$):

$$\mathbf{A} = \mathbf{V\Lambda V}^{-1}, \ \mathbf{\Lambda} = \text{diag}(\lambda_1, ..., \lambda_n), \ \mathbf{V} = \begin{pmatrix} | & & | \\ \mathbf{v_1} & \cdots & \mathbf{v_n} \\ | & & | \end{pmatrix}$$

$\mathbf{A}^{-1} = \mathbf{V}\Lambda^{-1}\mathbf{V}^{-1}$ if the eigen-decomposition exists and $\forall \lambda_i \neq 0$.

1.5.7 Orthogonality

A matrix $\mathbf{A}$ is orthogonal iff its columns *and* rows are orthogonal, i.e. $\mathbf{A}^T\mathbf{A} = \mathbf{AA}^T = \mathbf{I}$. The determinant of an orthogonal matrix is ± 1. It is column-orthogonal iff $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. Property: $\mathbf{A}^T = \mathbf{A}^{-1}$.

1.5.8 Singular Value Decomposition (SVD)

Definitions differ in the shape of $\mathbf{\Sigma}$. Let $\mathbf{A} = \mathbf{U\Sigma V}^T \in \mathbb{R}^{m \times n}$ with $p = \text{rank}(\mathbf{A}) \leq \min(m, n)$.

Compact SVD:
$\mathbf{U} \in \mathbb{R}^{m \times p}$ (orthogonal columns),
$\mathbf{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_p) \in \mathbb{R}^{p \times p}$ and
$V \in \mathbb{R}^{n \times p}$ (orthogonal)

where singular values are ordered: $\sigma_1 \geq ... \geq \sigma_n > 0$. SVD is a **generalization of eigenvalues and eigenvectors to non-square matrices**. The rank of $\mathbf{A}$ equals the number of non-zero singular values. Full SVD also contains the zero-singular-values:

Full SVD:
$\mathbf{U} \in \mathbb{R}^{m \times m}$ (orthogonal),
$\mathbf{\Sigma} = \text{diag}(\sigma_1, ..., \sigma_p) = \begin{pmatrix} \mathbf{D}_{r \times r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{m \times n}$,
$V \in \mathbb{R}^{n \times n}$ (orthogonal)

Connection to the Eigendecomposition

$$\mathbf{A}^T\mathbf{A} = (\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T$$
$$\mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T) = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T$$

Thus, the columns of $\mathbf{V}$ and $\mathbf{U}$ are the Eigenvectors of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ respectively. With $\lambda_1 \geq ... \geq \lambda_r > 0$ as the the non-zero Eigenvalues of $\mathbf{A}\mathbf{A}^T$ or $\mathbf{A}^T\mathbf{A}$, we conclude: $\mathbf{D}_{r\times r} = \begin{pmatrix} \sqrt{\lambda_1} & & \\ & \ddots & \\ & & \sqrt{\lambda_r} \end{pmatrix}$.

Applications Given a matrix $\mathbf{A}$, SVD is useful for finding an orthogonal basis that stays orthogonal after transforming it with $\mathbf{A}$. It can also be used to map points on a circle onto points on a sphere.

1.5.9 Pseudo Inverse (Generalized Inverse)

Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ and $\mathbf{U} \in \mathbb{R}^{m\times m}$. $\mathbf{\Sigma}_1$ is the diagonal matrix of non-zero singular values. Then:

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T \text{ with } \mathbf{\Sigma}^+ = \begin{pmatrix} \mathbf{\Sigma}_1^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

$\mathbf{x} = \mathbf{A}^+\mathbf{b}$ is the least squares solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$.

1.5.10 QR Decomposition

$\mathbf{A} = \mathbf{Q}\mathbf{R}$, where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. We find $\mathbf{Q} = [\mathbf{e}_1 \dots \mathbf{e}_n]$ by applying the Gram-Schmidt process to $\mathbf{A} = [\mathbf{a}_1 \dots \mathbf{a}_n]$. We obtain $\mathbf{R}$ by expressing the columns of $\mathbf{A}$ in the new basis and computing the "similarity" with the basis vectors:

$$\begin{aligned} \mathbf{a}_1 &= \langle \mathbf{e}_1, \mathbf{a}_1 \rangle \mathbf{e}_1 \\ \mathbf{a}_2 &= \langle \mathbf{e}_1, \mathbf{a}_2 \rangle \mathbf{e}_1 + \langle \mathbf{e}_2, \mathbf{a}_2 \rangle \mathbf{e}_2 \\ \mathbf{a}_3 &= \langle \mathbf{e}_1, \mathbf{a}_3 \rangle \mathbf{e}_1 + \langle \mathbf{e}_2, \mathbf{a}_3 \rangle \mathbf{e}_2 + \langle \mathbf{e}_3, \mathbf{a}_3 \rangle \mathbf{e}_3 \\ &\vdots \\ \mathbf{a}_k &= \sum_{j=1}^{k} \langle \mathbf{e}_j, \mathbf{a}_k \rangle \mathbf{e}_j \end{aligned} \Rightarrow R = \begin{bmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \langle \mathbf{e}_1, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_1, \mathbf{a}_n \rangle \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \langle \mathbf{e}_2, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_2, \mathbf{a}_n \rangle \\ 0 & 0 & \langle \mathbf{e}_3, \mathbf{a}_3 \rangle & \cdots & \langle \mathbf{e}_3, \mathbf{a}_n \rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \langle \mathbf{e}_n, \mathbf{a}_n \rangle \end{bmatrix}$$

1.5.11 Symmetric Matrices

A matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ is called symmetric if $\mathbf{S}^T = \mathbf{S}$.

- All eigenvalues of $\mathbf{S}$ are real, i.e. $\sigma(\mathbf{S}) \subset \mathbb{R}$.
- Eigenvectors of $\mathbf{S}$ corresponding to the same distinct eigenvalues are orthogonal.
- There exist n orthonormal eigenvectors of $\mathbf{S}$ which form a basis of $\mathbb{R}^3$. $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ with $\mathbf{V} = (v_1, ..., v_n) \in O(n)$ and $\mathbf{\Lambda} = \text{diag}(\lambda_1, ..., \lambda_n)$.
- $\mathbf{S}$ is PD (PSD), if all eigenvalues are positive (and non-zero).
- Let $\mathbf{S}$ be positive semi-definite and λ_1, λ_n the largest and smallest eigenvalue. Then $\lambda_1 = \max_{|x|=1}\langle x, \mathbf{S}x\rangle$ and $\lambda_n = \min_{|x|=1}\langle x, \mathbf{S}x\rangle$.

1.6 Matrix Operations

1.6.1 Dot Product

The dot product (*inner product*) $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ measures how similar two normalized vectors are.

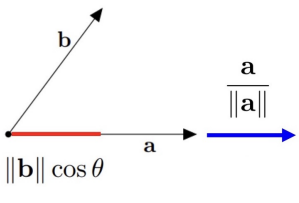
$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T\mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

Two vectors are orthogonal iff $\langle \mathbf{a}, \mathbf{b} \rangle = 0$ and point in the same direction iff $\langle \mathbf{a}, \mathbf{b} \rangle = 1$. Properties:

- linear: $\langle c, \alpha a + \beta b \rangle = \alpha\langle c, a \rangle + \beta\langle c, b \rangle$
- symmetric: $\langle a, b \rangle = \langle b, a \rangle$
- positive definite: $\langle a, a \rangle \geq 0$ and $\langle a, a \rangle = 0 \iff v = 0$

Projection of $\mathbf{b}$ onto $\mathbf{a}$:

$$\text{proj}_{\mathbf{a}}\mathbf{b} = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|} \underbrace{\frac{\mathbf{a}}{\|\mathbf{a}\|}}_{\text{Unit vector along } \mathbf{a}}$$

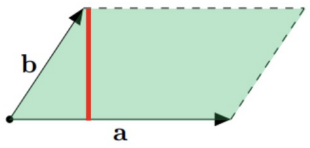


1.6.2 Cross product

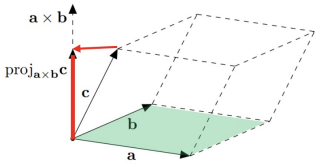
On $\mathbb{R}^3$ we define the cross product as

$$\times : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3 : \underbrace{\mathbf{a}\times\mathbf{b}}_{=-\mathbf{b}\times\mathbf{a}} = \begin{pmatrix} \mathbf{a}_2\mathbf{b}_3 - \mathbf{a}_3\mathbf{b}_2 \\ \mathbf{a}_3\mathbf{b}_1 - \mathbf{a}_1\mathbf{b}_3 \\ \mathbf{a}_1\mathbf{b}_2 - \mathbf{a}_2\mathbf{b}_1 \end{pmatrix}$$

which is a vector orthogonal to $\mathbf{a}$ and $\mathbf{b}$. The right hand rule determines the direction. We define the area of a parallelogram as $\|\mathbf{a}\times\mathbf{b}\| = \|\mathbf{a}\| \, \|\mathbf{b}\| \sin\theta$ and the volume of a parallelepiped as (Triple Product)



$$V = \|\mathbf{a}\times\mathbf{b}\| \underbrace{\frac{(\mathbf{a}\times\mathbf{b})\cdot\mathbf{c}}{\|\mathbf{a}\times\mathbf{b}\|}}_{\|\text{proj}_{\mathbf{a}\times\mathbf{b}}\mathbf{c}\|}$$



The skew-symmetric matrix $[\mathbf{a}]_\times \in \mathbb{R}^{3\times3}$ replaces the cross product with matrix multiplication:

$$\mathbf{a}\times\mathbf{b} = \underbrace{[\mathbf{a}]_\times}_{\text{or } \mathbf{a}^\wedge \text{ or } \hat{\mathbf{a}}}\mathbf{b} = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \cdot \mathbf{b}.$$

The cross product of a vector with itself equals zero ($\mathbf{a}\times\mathbf{a} = 0$). Notation: $\mathbf{a}^\wedge = \mathbf{A}$ and $\mathbf{A}^\vee = \mathbf{a}$.

1.6.3 Kronecker Product and Stack

Given two matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{k\times l}$, we define their Kronecker product:

$$\mathbf{A} \otimes \mathbf{B} \equiv \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{mk\times nl}$$
$$\text{with} \quad (\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$$

Given a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, its stack $\mathbf{A}^S$ is obtained by stacking its n column vectors $a_1, ..., a_n \in \mathbb{R}^m$:

$$\mathbf{A}^S \equiv \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \in \mathbb{R}^{mn} \quad \text{and} \quad u^T\mathbf{A}v = (v \otimes u)^T\mathbf{A}^S$$

1.7 Group

A group is an algebraic structure that combines a set A and an operator $\circ$, e.g. $(\mathbb{N}, \cdot)$, with the properties:

1. $a \circ b \in S$ (Closure)
2. $(a \circ b) \circ c = a \circ (b \circ c)$ (Associative law)
3. $a \circ n = a$ (n is an identity element in S)
4. $a \circ a^{-1} = n$ (every element has an inverse element)

Commutative addition of Abelian groups: $a \circ b = b \circ a$.

Ring A set S combined with two defined operations and properties from above, e.g. $(\mathbb{N}, +, \cdot)$.

Field A ring which is distributive in addition: $a \cdot (b + c) = a \cdot b + a \cdot c$, e.g. $(\mathbb{R}, +, \cdot)$.

Popular Groups

- General linear group (matrix multiplication): $GL(n) = (\{\mathbf{A} \in \mathbb{R}^{n\times n} \mid \det(\mathbf{A}) \neq 0\}, \cdot)$
- Special linear group: $SL(n) = \{\mathbf{A} \in GL(n) \mid \det(\mathbf{A}) = +1\}$
- Orthogonal group (orth. matr.: $\det R = \pm 1$): $O(n) = \{\mathbf{R} \in GL(n) \mid \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}\}$
- Special orthogonal group (rot. matr.): $SO(n) = \{\mathbf{A} \in O(n) \mid \det(\mathbf{A}) = +1\}$
- Affine group (all affine transf. $L(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$): $A(n) = \{\left(\begin{smallmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0}^T & 1 \end{smallmatrix}\right) \mid A \in GL(n), b \in \mathbb{R}^n\}$
- Euclidean group (subgroup of $A(n)$): $E(n) = \{\left(\begin{smallmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{smallmatrix}\right) \mid \mathbf{R} \in O(n), \mathbf{T} \in \mathbb{R}^n\}$
- Special Euclidean group (Subgroup of $E(n)$): $SE(n) = \{\left(\begin{smallmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{smallmatrix}\right) \mid \mathbf{R} \in SO(n), \mathbf{T} \in \mathbb{R}^n\}$ Does rigid-body motions: $L(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{T}$.

These groups are included in the following way:

$$SO(n) \subset O(n) \subset GL(n)$$
$$SE(n) \subset E(n) \subset A(n) \subset GL(n+1)$$

1.7.1 Subgroup test

If G is a group, and H is a subset of G, then H is a subgroup of G iff H is nonempty and closed under $\circ$ and inverses. Combination of these two conditions: For every a and b in H, the element $a \circ b^{-1}$ is in H.

1.8 Norms for $A \in \mathbb{R}^{m\times n}$

Induced 2-norm: $\|A\|_2 = \max_{|x|_2=1} \sqrt{\langle x, A^TAx\rangle} = \sigma_1$

Frobenius norm: $\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2} = \sqrt{\mathrm{Tr}(A^TA)} = \sqrt{\sigma_1^2 + ... + \sigma_n^2}$

With $A^TA = V \cdot \mathrm{diag}(\sigma_1^2, ..., \sigma_n^2) \cdot V^T$ and $\sigma_1^2 \geq \sigma_i^2 \geq 0$.

1.9 Derivative, Gradient and Jacobian

The **derivative** of a *single univariate* function is:

$$f'(x) = \frac{df}{dx}, \text{ where } x \in \mathbb{R} \qquad \text{with } f(x) \in \mathbb{R}.$$

The **gradient** of a *single multivariate* function is:

$$\nabla f(x_1, x_2, ..., x_n) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad \text{with } f(x_1, x_2, ..., x_n) \in \mathbb{R}.$$

The derivative of a *multiple multivariate function* (vector-valued) is the **Jacobian**:

$$\frac{\partial F}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial F_1}{\partial \mathbf{x}_1} & \cdots & \frac{\partial F_1}{\partial \mathbf{x}_n} \\ & \vdots & \\ \frac{\partial F_m}{\partial \mathbf{x}_1} & \cdots & \frac{\partial F_m}{\partial \mathbf{x}_n} \end{pmatrix} \quad \text{with } F(x_1, x_2, ..., x_n) \in \mathbb{R}^m.$$

1.10 Taylor Approximation

First-order Taylor polynomial around some value a:

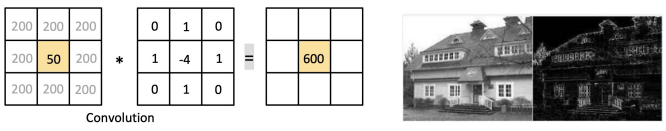
$$f(x)|_a \approx f(a) + \frac{f'(a)}{1!}(x - a)$$

1.11 Laplace Operator

Mathematically, it is defined as:

$$\begin{aligned} \nabla^2 f &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \\ &= [f(x+1, y) + f(x-1, y) + f(x, y+1) \\ &\quad + f(x, y-1)] - 4f(x, y) \end{aligned}$$

where ∇^2 is the Laplacian operator. The Laplace operator is used like a convolution kernel and highlights the pixels with sharp intensity discontinuity (e.g. edge or corner pixels).



2 Motion and Scene Representation

2.1 3D Space

The three-dimensional Euclidean space $\mathbb{E}^3$ consists of all points $\mathbf{p} \in \mathbb{E}^3$ characterized by coordinates

$$\mathbf{X} \equiv (X_1, X_2, X_3)^T \in \mathbb{R}^3.$$

Given two points $\mathbf{X}$ and $\mathbf{Y}$, one can define a bound vector $\mathbf{v}$ as

$$\mathbf{v} = \mathbf{X} - \mathbf{Y} \in \mathbb{R}^3.$$

2.2 Rigid Body Motion

A rigid body motion is a map

$$g_t : \mathbb{R}^3 \to \mathbb{R}^3;\ \mathbf{X} \mapsto g_t(\mathbf{X}),\ t \in [0, T]$$

which preserves the norm and inner product (length), the cross product (orientation) and the triple product (volume) of any two vectors. The rigid body motion can be written as $g_t(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{T}$, where $\mathbf{R} \in SO(3)$ is a rotation matrix from the special orthogonal group and $\mathbf{T} \in \mathbb{R}^3$ is a translation vector.

2.3 The Special Orthogonal Group: SO(3)

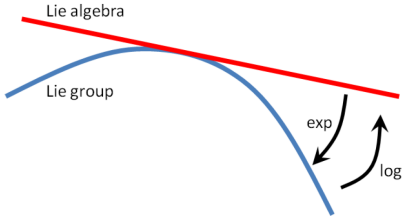
The effect of any infinitesimal rotation $\mathbf{R} \in SO(3)$ can be approximated by an element from the space of skew-symmetric matrices $\mathfrak{so}(3) = \{\hat{w} \,|\, w \in \mathbb{R}^3\}$. The rotation group $SO(3)$ is called a Lie group and the linear space $\mathfrak{so}(3)$ is called its Lie algebra.

Definition "Lie group": Smooth manifold that is also a group, s.t. the group operations multiplication and inversions are continuous (smooth) maps. The SO(n) and SE(n) groups are continuous in real space and are therefore Lie Groups.

$$\mathrm{SO}(n) = \{\mathbf{R} \in \mathbb{R}^{n \times n} \,|\, \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\}$$

Definition "Lie algebra": The tangent space of a Lie group at the identity element is called the associated Lie algebra. While the multiplication operator is well-defined on the SO(3) group, the addition operator is not, which affects the derivative computation. We use the Lie algebra to obtain an expression for the derivative of rotation matrices from the SO(3) group.

$$\mathfrak{so}(3) = \left\{ w \in \mathbb{R}^3 \text{ or } \hat{w} = \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix} \in \mathbb{R}^{3\times 3} \right\}$$



The matrix exponential defines a map from the Lie algebra to the Lie group and, conversely, the logarithm maps elements in SO(3) to $\mathfrak{so}(3)$:

$$\exp : \mathfrak{so}(3) \to SO(3); \quad \hat{w} \mapsto \exp(\hat{w})\ (= \mathbf{R})$$
$$\log : SO(3) \to \mathfrak{so}(3); \quad \mathbf{R} \mapsto \log(\mathbf{R})\ (= \hat{w}).$$

Where Rodrigues' formula gives the matrix exponential:

$$\exp(\hat{w}) = \mathbf{I} + \frac{\hat{w}}{\theta}\sin(\theta) + \frac{\hat{w}^2}{\theta^2}(1 - \cos(\theta)) = \mathbf{R}$$

and the inverse logarithmic mapping is given by:

$$\theta = \|w\| = \cos^{-1}\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right) \quad \text{(angle)}$$

$$\frac{w}{\|w\|} = \frac{1}{2\sin(\|w\|)}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} \quad \text{(axis)}$$

for all $\mathbf{R} \neq \mathbf{I}$. If $w = 0$, we have $\exp(\hat{w}) = \mathbf{I}$. Any orthogonal transformation $\mathbf{R} \in SO(3)$ can be achieved by rotating about the angle $|w|$ around an axis $\frac{w}{|w|}$.

2.4 The Special Euclidean Group: SE(3)

The space of rigid body motions given by the group of special euclidean transformations

$$SE(3) \equiv \left\{ g = (\mathbf{R}, \mathbf{T}) \,\middle|\, \mathbf{R} \in SO(3), \mathbf{T} \in \mathbb{R}^3 \right\}.$$

In homogeneous coordinates, we have:

$$SE(3) \equiv \left\{ g = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix} \,\middle|\, \mathbf{R} \in SO(3), \mathbf{T} \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4\times 4}$$

And the set of all twists forms its Lie algebra $\mathfrak{se}(3)$:

$$\mathfrak{se}(3) \equiv \left\{ \hat{\xi} = \begin{pmatrix} \hat{\mathbf{w}} & \mathbf{v} \\ \mathbf{0}^T & 0 \end{pmatrix} \,\middle|\, \hat{\mathbf{w}} \in \mathfrak{so}(3), \mathbf{v} \in \mathbb{R}^3 \right\} \subset \mathbb{R}^{4\times 4}$$
$$\text{with} \quad \mathbf{v}(t) = \dot{\mathbf{T}}(t) - \hat{\mathbf{w}}(t)\mathbf{T}(t)$$

where v is the linear velocity and w is the angular velocity. Operators $\wedge$ and $\vee$ convert between a twist $\hat{\xi} \in \mathfrak{se}(3)$ and its twist coordinates $\xi \in \mathbb{R}^6$:

$$\hat{\xi} \equiv \begin{pmatrix} v \\ w \end{pmatrix}^{\wedge} \equiv \begin{pmatrix} \hat{w} & v \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{4\times 4}$$
$$\begin{pmatrix} \hat{w} & v \\ 0 & 0 \end{pmatrix}^{\vee} = \begin{pmatrix} v \\ w \end{pmatrix} \in \mathbb{R}^6$$

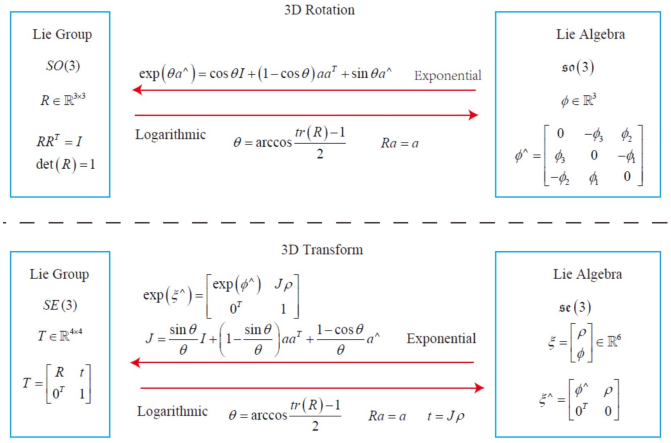
We define an exponential map from the Lie Algebra to the Lie Group $\exp : \mathfrak{se}(3) \to SE(3);\ \hat{\xi} \mapsto \exp(\hat{\xi})$:

For $w = 0$: $\quad \exp(\hat{\xi}) = \begin{pmatrix} \mathbf{I} & \mathbf{v} \\ \mathbf{0}^T & 1 \end{pmatrix}$

For $w \neq 0$: $\quad \exp(\hat{\xi}) = \begin{pmatrix} \exp(\hat{w}) & \mathbf{V}\mathbf{v} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix} = g$

with $\mathbf{V} = \mathbf{I}_3 + \frac{1 - \cos\theta}{\theta^2}\hat{\mathbf{w}} + \frac{\theta - \sin\theta}{\theta^3}\hat{\mathbf{w}}^2$ and $\theta = \|\mathbf{w}\|_2$

More info: http://ethaneade.com/lie.pdf and this tutorial.
In summary:



2.5 BCH Formula

We compute derivatives on $\mathfrak{so}(3)$ using the BCH formula. The problem is that the sum $\mathbf{R}_1 + \mathbf{R}_2$ is not defined on SO(3) and it also does not directly correspond to the product of two vectors on $\mathfrak{so}(3)$.

2.6 Representing the Camera Motion

The rigid-body transformation

$$g(t) = \begin{pmatrix} \mathbf{R}(t) & \mathbf{T}(t) \\ \mathbf{0}^T & 1 \end{pmatrix} \in SE(3)$$

represents the motion from a fixed world frame to the camera frame at time t. In particular we assume that at time $t = 0$ the camera frame coincides with the world frame, i.e. $g(0) = \mathbf{I}$. For any point $\mathbf{X}_0$ in world coordinates, its coordinates in the camera frame at time t are:

$$\mathbf{X}(t) = \mathbf{R}(t)\mathbf{X}_0 + \mathbf{T}(t)$$

or in homogeneous representation

$$\mathbf{X}(t) = g(t)\mathbf{X}_0.$$

We denote the transformation from the points in frame t_1 to the points in frame t_2 by $g(t_2, t_1)$:

$$\mathbf{X}(t_2) = g(t_2, t_1)\mathbf{X}(t_1).$$

$$g(t_1, t_2)g(t_2, t_1) = I \Leftrightarrow g^{-1}(t_2, t_1) = g(t_1, t_2)$$

For velocity transformation it applies

$$\dot{\mathbf{X}}(t) = \hat{w}(t)\mathbf{X}(t) + v(t) \quad \text{or}$$

$$\dot{\mathbf{X}}(t) = \hat{\mathbf{V}}(t)\mathbf{X}(t) \quad \text{with}$$

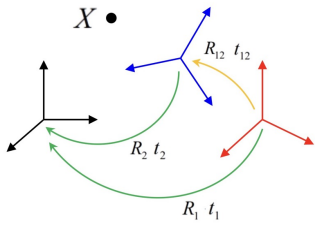
$$\hat{\mathbf{V}}(t) = \begin{pmatrix} \hat{w}(t) & v(t) \\ 0 & 0 \end{pmatrix} \in se(3)$$

We calculate **relative poses** from absolute poses:

$$X_W = R_1 X_1 + t_1 = R_2 X_2 + t_2$$

$$R_1 X_1 + (t_1 - t_2) = R_2 X_2$$

$$\underbrace{R_2^T R_1}_{R_{12}} X_1 + \underbrace{R_2^T (t_1 - t_2)}_{t_{12}} = X_2$$



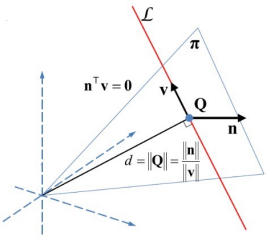
We invert a tranformation via:

$$\mathbf{Y} = \mathbf{R}\mathbf{X} + \mathbf{t} \quad \Rightarrow \quad \mathbf{X} = \mathbf{R}^T(\mathbf{Y} - \mathbf{t})$$

$$\mathbf{T} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

2.7 Motion of A 3D Line

We represent a 3D line using its *Plücker Coordinates* $(\mathbf{v}, \mathbf{n})$, where $\mathbf{v}$ is the direction vector of the line and $\mathbf{n} = \mathbf{Q} \times \mathbf{v}$ is the normal of the projection plane that we calculate from the cross product with some point on the line $\mathbf{Q}$. Plücker coordinates have 4 DOF.



We transform a Plücker line from the ith to the jth coordinate frame via:

$$\begin{pmatrix} \mathbf{n}_j \\ \mathbf{v}_j \end{pmatrix} = \begin{pmatrix} \mathbf{R}_{ji} & [\mathbf{t}_{ji}]_\times \mathbf{R}_{ji} \\ \mathbf{0} & \mathbf{R}_{ji} \end{pmatrix} \begin{pmatrix} \mathbf{n}_i \\ \mathbf{v}_i \end{pmatrix}$$

which changes the norm of $\mathbf{n}$ and not of $\mathbf{v}$.

2.8 Similarity Transformation in 3D: Sim(3)

We define

$$\mathbf{T}_S = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix}$$

which is a 7 DOF transformation. The scale for the rotation enables matching of point clouds at different scales.

2.9 Adjoint Map

A viewer in frame A is displaced relative to the current frame by a transformation $g_{xy} : Y = g_{xy}X(t)$. The velocity in frame A is then

$$\dot{Y}(t) = g_{xy}\dot{X}(t) = g_{xy}\hat{V}(t)X(t) = g_{xy}\hat{V}g_{xy}^{-1}Y(t).$$

This shows that the relative velocity of points observed from camera frame A is represented by the twist

$$\hat{V}_y = g_{xy}\hat{V}g_{xy}^{-1} \equiv ad_{g_{xy}}(\hat{V})$$

with the adjoint map

$$ad_g : se(3) \to se(3);\ \hat{\xi} \mapsto g\hat{\xi}g^{-1}.$$

2.10 More Ways of Expressing Rotation

We differentiate extrinsic rotations about axes of the fixed, global coordinate system from intrinsic rotations about the dynamic coordinate system attached to a moving body.

2.10.1 Axis-Angle Representation

Rotation about θ around a unit vector $\mathbf{e}$, represented by the rotation vector $\theta\mathbf{e} = (\mathbf{e}, \theta)$. We convert an axis-angle representation into a rotation matrix using Rodrigues' formula:

$$\begin{aligned}\mathbf{R}(\mathbf{n}, \theta) &= \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T + \sin\theta\mathbf{n}^\wedge \\ &= \begin{pmatrix} n_x^2(1-c\theta)+c\theta & n_x n_y(1-c\theta)+n_z s\theta & n_x n_z(1-c\theta)-n_y s\theta \\ n_x n_y(1-c\theta)-n_z s\theta & n_y^2(1-c\theta)+c\theta & n_y n_z(1-c\theta)+n_x s\theta \\ n_x n_z(1-c\theta)+n_y s\theta & n_y n_z(1-c\theta)-n_x s\theta & n_z^2(1-c\theta)+c\theta \end{pmatrix}\end{aligned}$$

And vice versa:

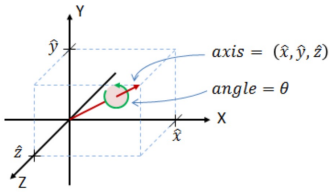
$$\mathbf{n} = \frac{1}{2\sin\theta}\begin{pmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{pmatrix}, \quad \theta = \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right)$$

$$\begin{aligned}\mathrm{tr}(\mathbf{R}) &= \cos\theta\,\mathrm{tr}(\mathbf{I}) + (1 - \cos\theta)\,\mathrm{tr}\left(\mathbf{n}\mathbf{n}^T\right) + \sin\theta\,\mathrm{tr}\left(\mathbf{n}^\wedge\right) \\ &= 3\cos\theta + (1 - \cos\theta) \\ &= 1 + 2\cos\theta\end{aligned}$$

The axis-angle representation has 4 parameters with one DOF removed due to the unit vector constraint.

2.10.2 Quaternion

We define a rotation using a set of four numbers $\mathbf{q} = (q_0, q_1, q_2, q_3)$ that parameterize a complex number with three imaginary axes: $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$



$$q_0 = \cos\left(\frac{\theta}{2}\right) \quad q_1 = \hat{x}\sin\left(\frac{\theta}{2}\right) \quad q_2 = \hat{y}\sin\left(\frac{\theta}{2}\right)$$
$$q_3 = \hat{z}\sin\left(\frac{\theta}{2}\right) \quad \mathbf{q} = [\underbrace{\cos\frac{\theta}{2}}_{\text{real}}, \underbrace{\mathbf{n}\sin\frac{\theta}{2}}_{\text{imaginary}}]$$

3D points can be represented as quaternions with a zero-real-coordinate, $\mathbf{p} = (0, x, y, z)$. We convert quaternions to a rotation matrix using the *Hamilton product* (derivation skipped):

$$R = \begin{pmatrix} 1 - 2\left(q_y^2 + q_z^2\right) & 2\left(q_x q_y - q_w q_z\right) & 2\left(q_w q_y + q_x q_z\right) \\ 2\left(q_x q_y + q_w q_z\right) & 1 - 2\left(q_x^2 + q_z^2\right) & 2\left(q_y q_z - q_w q_x\right) \\ 2\left(q_x q_z - q_w q_y\right) & 2\left(q_w q_x + q_y q_z\right) & 1 - 2\left(q_x^2 + q_y^2\right) \end{pmatrix}$$

where $s = 1$ for a unit quaternion. Quaternions have 4 parameters, but only 3 DOF because of the unit quaternion constraint.

2.10.3 Euler Angles

Alternative representation to parametrize rotation matrices $\mathbf{R} \in SO(3)$. Given a basis $(\hat{w}_1, \hat{w}_2, \hat{w}_3)$ of the Lie algebra $so(3)$, the Lie-Cartan coordinates of the second kind are defined as:

$$\beta : (\beta_1, \beta_2, \beta_3) \to \exp\left(\beta_1\hat{w}_1\right)\exp\left(\beta_2\hat{w}_2\right)\exp\left(\beta_3\hat{w}_3\right).$$

For the basis representing rotation around z-, y-, x-axis $(w_1 = (0,0,1)^T, w_2 = (0,1,0)^T, w_3 = (1,0,0)^T)$, the coordinates $\beta_1, \beta_2, \beta_3$ are called Euler angles.

We can convert Euler angles into a rotation matrix through intrinsic rotations about the dynamic axes:

$$\begin{aligned}\mathbf{R}_{0,3} = \mathbf{R}_{0,1}\mathbf{R}_{1,2}\mathbf{R}_{2,3} &= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \\ &= \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \\ &= \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix}\end{aligned}$$

and $\alpha = \mathrm{atan2}(r_{32}, r_{33})$

$$\beta = \mathrm{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2})$$

$$\gamma = \mathrm{atan2}(r_{21}, r_{11})$$

In practice, we avoid using Euler angles to avoid "gimbal lock", where we loose a DOF due to the alignment of two rotation axes.

Summary

	Rotation $SO(3)$	Rigid-body $SE(3)$
Matrix representation	$\mathbf{R} \in GL(3)$: $\mathbf{R}^T\mathbf{R} = I$, $\det(\mathbf{R}) = +1$	$g = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix}$
3-D coordinates	$\mathbf{X} = \mathbf{R}\mathbf{X}_0$	$\mathbf{X} = \mathbf{R}\mathbf{X}_0 + \mathbf{T}$
Inverse	$\mathbf{R}^{-1} = \mathbf{R}^T$	$g^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix}$
Exp. representation	$\mathbf{R} = \exp(\hat{w})$	$g = \exp(\hat{\xi})$
Velocity	$\dot{\mathbf{X}} = \hat{w}\mathbf{X}$	$\dot{\mathbf{X}} = \hat{w}\mathbf{X} + v$
Adjoint map	$\hat{w} \mapsto \mathbf{R}\hat{w}\mathbf{R}^T$	$\hat{\xi} \mapsto g\hat{\xi}g^{-1}$

2.11 3D Scene Representations

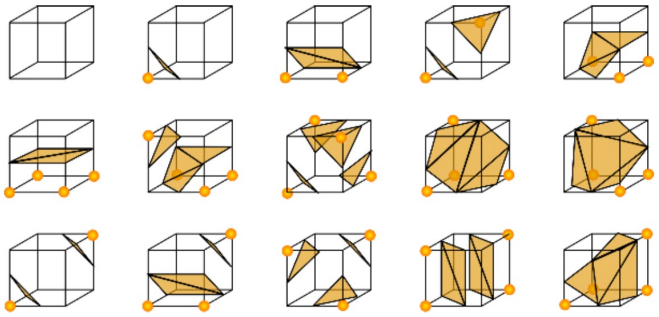
Point Cloud A point cloud is a discrete set of data points in space with corresponding Cartesian coordinates. Point clouds are typically *sparse*.

Voxel Grid A voxel grid is a 3D grid where each cell is a "voxel". Voxel grids are useful in deep learning when 3D convolutions are used, but are expensive due to the *density* of the representation. Thus, we use low-resolution voxel grids in practice.

Mesh A polygon mesh is a collection of *vertices, edges and faces* that define the shape of a polyhedral object. Faces are combined to *polygons*, which form *surfaces*. Generally, triangle meshes work well for simple geometries and are easier to use, but quad meshes provide more accurate results on complex geometries.

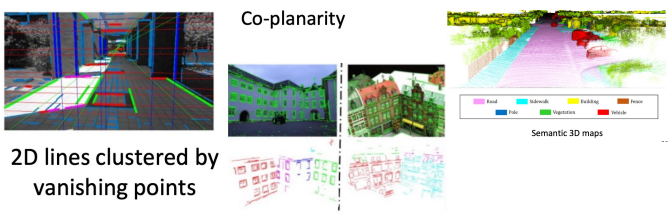
	Voxel	Point cloud	Polygon mesh
Memory efficiency	Poor	Not good	Good
Textures	Not good	No	Yes
For neural networks	Easy	Not easy	Not easy

Signed Distance Function (SDF; Implicit Surface) An SDF is a function that maps a point in space to the signed-distance between the point and the nearest point on the object's surface. The set of all points with zero-SDF-values is called the *isosurface* and represents the object surface. The sign represents whether points are in front of or behind the surface. We can extract a mesh from the isosurface using the *Marching Cubes (MC) algorithm*. MC requires a voxel grid with SDF values as input and then places a mesh vertex whenever it registers a change in the sign between two voxels. There exist 15 possible isosurface configurations that can be extracted from a cube with 8 neighboring voxels.



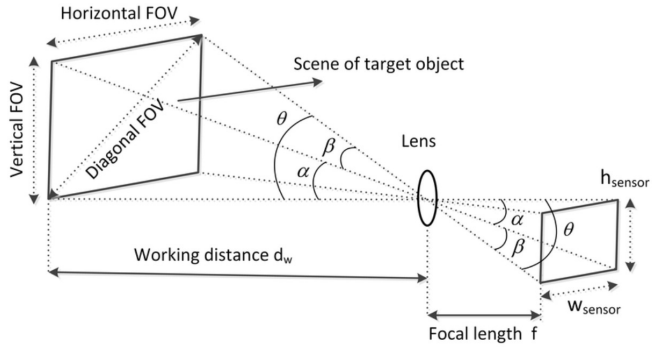
"Line" Cloud The line features are more useful than points in a point cloud, but are also more difficult to extract under bad environment conditions.

Integrated Information In certain environments additional information is available that we can *integrate* into the scene representation. E.g. in urban environments we know that vertical lines are supposed to be parallel to each other and orthogonal to horizontal lines. In case of the facade of a house it may make sense to enforce co-planarity of all line for instance. We can also assign points class labels that we know from semantic segmentation.

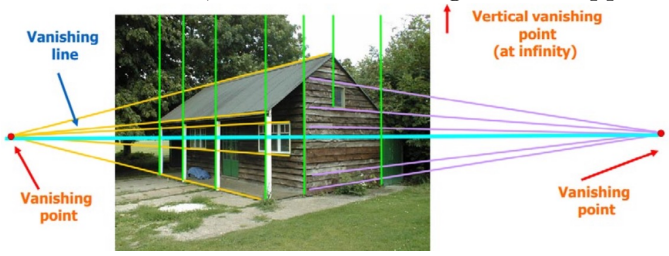


3 Image Formation

3.1 Ideal Perspective / Pinhole Camera



Perspective effects in perspective projections with the pinhole camera: 1) object size is inversely proportional to the distance; 2) parallel lines intersect at a *vanishing point* where the connection between two horizontal vanishing points is the *horizon*; 3) for convenience reasons, the *virtual image plane* is placed in front of the lens, such that the image is not flipped.



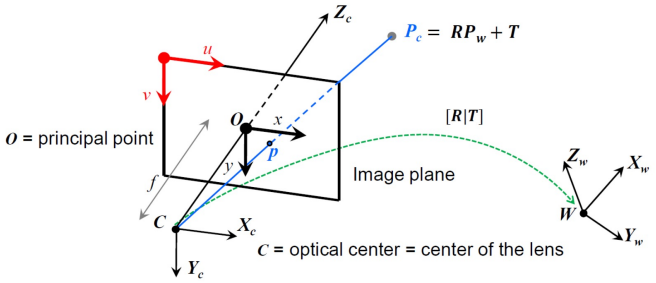
The angles between the edges that span the viewing frustrum are called *angle of view (AOV)* or *FOV angle* and the extents of the scene at some target distance are called *field of view (FOV)*:

$$\text{AOV} = 2\arctan\left(\frac{\text{sensor size}}{2f}\right) \qquad \text{[angle]}$$

$$\text{FOV} = 2\tan\left(\frac{\text{AOV}}{2}\right)\cdot d_w \qquad \text{[length]}$$

3.1.1 Intrinsic Camera Parameters

Pinhole camera model with the *image plane* (informally referred to as the *projection plane*) in front:



The origin for image coordinates is in the center of the 2D image plane with expansion $(-1,-1)$ and $(1,1)$. The origin of pixel coordinates is moved to the bottom- or top-left corner to enable a purely positive expansion e.g. $(0,0)$ and $(1024,768)$. The **intrinsic/calibration camera matrix K** is:

$$\mathbf{K} = \mathbf{K}_s\mathbf{K}_f = \begin{pmatrix} s_x & s_\theta & o_x \\ 0 & s_y & o_y \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{pmatrix}$$

Its entries can be interpreted as follows:

- f: focal length in [mm]
- s_x: scaling to pixels in x-direction [pixels/mm]
- $f \cdot s_x = \alpha_x$: unit length in x-direction [pixels]
- s_y: scaling to pixels in y-direction [pixels/mm]
- $f \cdot s_y = \alpha_y$: unit length in y-direction [pixels]
- α_x/α_x: aspect ratio σ
- s_θ: skew factor for non-square pixels
- $f \cdot s_\theta$: skew of the pixel
- o_x: x-coordinate of principal point [pixels]
- o_y: y-coordinate of principal point [pixels]

For modern cameras, it it safe to assume $s_\theta = 0$ and $\alpha_x = \alpha_y$. The **extrinsic camera matrix** describes the camera motion $g = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix}$.

3.2 Perspective Projection

From Camera to Image Coordinates The perspective transformation π from camera coordinates $\boldsymbol{X}$ to image coordinates x is given by

$$\pi : \mathbb{R}^3 \to \mathbb{R}^2; \quad \boldsymbol{X} \mapsto \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} = \pi(\boldsymbol{X}) = \begin{pmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \end{pmatrix}$$

or in homogeneous coordinates;

$$Z\boldsymbol{x} = Z\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \boldsymbol{\Pi}_0\boldsymbol{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

where $\boldsymbol{\Pi}_0$ is the standard projection matrix. We usually denote Z as $\lambda = \text{const} > 0$ and multiply with the focal length to obtain image-space coordinates in [mm]:

$$\lambda\boldsymbol{x} = \mathbf{K}_f\Pi_0\boldsymbol{X}.$$

From World to Image Coordinates Transformation of point $\boldsymbol{X}_0$ in world coordinates to a point $\boldsymbol{X}$ in camera coordinates:

$$\boldsymbol{X} = g\boldsymbol{X}_0 = \begin{pmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{pmatrix}\boldsymbol{X}_0.$$

Transformation from world coordinates to image coordinates:

$$\lambda\boldsymbol{x} = \mathbf{K}_f\boldsymbol{\Pi}_0 g\boldsymbol{X}_0.$$

To Pixel Coordinates The pixel coordinates $\mathbf{x}' = (u\ v\ 1)^T$ as a function of homogeneous camera coordinates $\boldsymbol{X}$ are given by

$$\lambda \boldsymbol{x}' = \lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K}_s \mathbf{K}_f \mathbf{\Pi}_0 \boldsymbol{X} = \mathbf{K} \mathbf{\Pi}_0 \boldsymbol{X}$$

$$\text{which is equivalent to} \quad \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{f s_x X}{Z} + o_x \\ \frac{f s_y Y}{Z} + o_y \\ 1 \end{pmatrix}$$

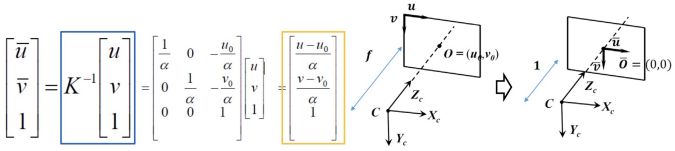
and as function of world coordinates $\boldsymbol{X}_0$ by

$$\lambda \boldsymbol{x}' = \mathbf{K} \mathbf{\Pi}_0 \boldsymbol{X} = \mathbf{K} \mathbf{\Pi}_0 g \boldsymbol{X}_0 \equiv \mathbf{\Pi} \boldsymbol{X}_0.$$

The matrix $\mathbf{\Pi} \equiv \mathbf{K}\mathbf{\Pi}_0 g = (KR, KT)$ is the general projection matrix.

Summary for Homogeneous Coordinates: 4D world coordinates $\boldsymbol{X}_0 \xrightarrow{g \in SE(3)}$ 4D camera coordinates $\boldsymbol{X} \xrightarrow{\mathbf{K}_f \mathbf{\Pi}_0}$ 3D image coordinates $\boldsymbol{x}$ in [mm] $\xrightarrow{\mathbf{K}_s}$ 3D pixel coordinates $\boldsymbol{x}'$.

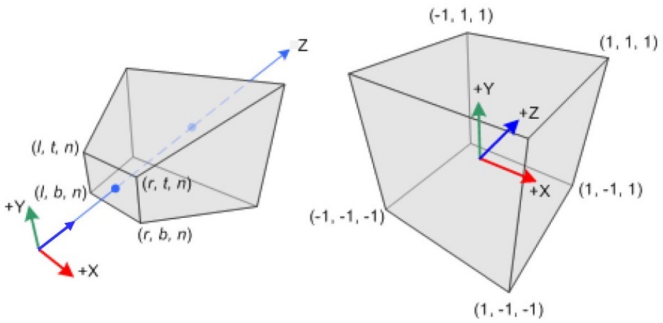
Normalized Image Plane In order to interpret image-space features as camera-space features (e.g. a 2D point as a 3D point), we construct a virtual image plane with $f = 1$ and the origin of the pixel coordinates at the principal point.



Comparison to Computer Graphics The main difference that in CG, we transform the viewed volume, i.e. the pyramidal frustum in front of the image, into a unit cube that is described by normalized device coordinates (NDC; $\in [-1, 1]$). Mathematically, we project a camera-space point $\mathbf{x}_{\text{eye}}$ into the *NDC-space* via:

$$\begin{pmatrix} x_{\text{clip}} \\ y_{\text{clip}} \\ z_{\text{clip}} \\ w_{\text{clip}} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{Perspective matrix } \mathbf{P}} \begin{pmatrix} x_{\text{eye}} \\ y_{\text{eye}} \\ z_{\text{eye}} \\ w_{\text{eye}} \end{pmatrix}$$

$$\text{such that} \quad \begin{pmatrix} x_{\text{ndc}} \\ y_{\text{ndc}} \\ z_{\text{ndc}} \end{pmatrix} = \begin{pmatrix} x_{\text{clip}}/w_{\text{clip}} \\ y_{\text{clip}}/w_{\text{clip}} \\ z_{\text{clip}}/w_{\text{clip}} \end{pmatrix}$$



In contrast to CV, every projected point $\mathbf{x}_{\text{ndc}}$ retains normalized depth information. We get *image space* coordinates by projecting the NDC space onto the XOY-plane and stretching it to the screen width and height.

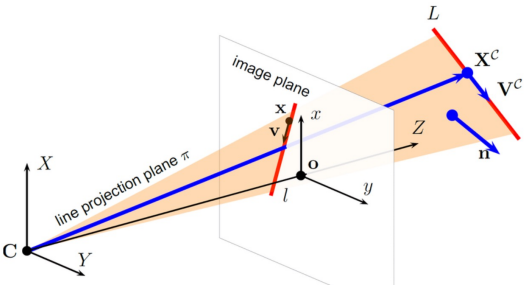
Line Projection Naively, we project a line into image space by projecting each of its endpoints. The *intrinsic matrix for line projection* $\mathcal{K}$ allows a one-step computation of the line coordinates :

$$\mathbf{l} = \mathcal{K}\mathbf{n} \quad \text{with} \quad \mathcal{K} = \begin{pmatrix} f_y & 0 & 0 \\ 0 & f_x & 0 \\ -f_y x_0 & -f_x y_0 & f_x f_y \end{pmatrix}$$

where $\mathbf{n}$ is the normal of the **projection plane** of the line according to its Plücker coordinates and the 2D line $\mathbf{l}$ is defined as:

$$ax + by + c = 0 \quad \text{with } \mathbf{l} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \underbrace{\begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}}_{\text{two points}}$$

We can calculate it from the cross product of the vectors from the principal points to the line's image-space end-points.

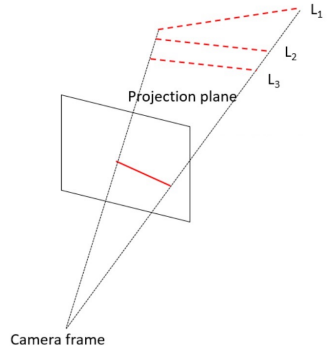


An image-space line $\mathbf{l}$ corresponds to a world-space line $\mathbf{L}$ if their projection-plane-normals are parallel (*l*ine constraint):

$$\mathbf{n}^{\mathbf{l}} \propto \mathbf{n}^{\mathbf{L}} \quad \Rightarrow \quad \mathcal{K}^{-1}\mathbf{l} \propto [\mathbf{R}[\mathbf{t}]_\times \mathbf{R}]\mathbf{L}$$

Two parallel vectors have a cross product of zero. Because a 3×3 skew-symmetric matrix has rank 3, this parallelism of each 3D-2D line correspondence provides two constraints.

All 3D vectors in a projection plane are projected onto the same 2D line which is known as the **projection ambiguity problem**. L_1, L_2, L_3 share the same normal $\mathbf{n}$ up to a scalar factor, which is irrelevant when computing $\mathbf{l} = \mathcal{K}\mathbf{n}$. We say, the projections are *identical up-to-scale*.

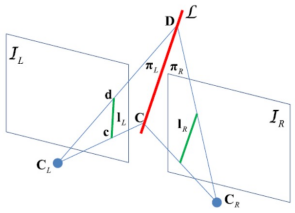


Relationship between Points, Lines and Planes We describe a **3D plane** using a four-dim. vector $(A\ B\ C\ D)^T$, where the first three coordinates are the normal and D is the projected distance from the origin. These coordinates must satisfy the constraint $Ax + By + Cz = D$. A **3D point** $\mathbf{P} = (X\ Y\ Z\ 1)^T$ lies on a plane if $\mathbf{P}^T\mathbf{n} = 0$ holds.

The projection plane of an image-space line $\mathbf{l}_L$ is:

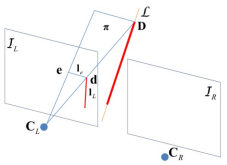
$$\underbrace{\pi_{\mathbf{L}}}_{4\times1} = \underbrace{\mathbf{P}^T}_{(3\times4)^T} \underbrace{\mathbf{l}_L}_{3\times1} \in \mathbb{R}^4$$

Proj. matrix



The intersection between a 3D line $\mathcal{L} = (\mathbf{n}, \mathbf{v})$ and 3D plane π is:

$$\mathbf{D} = \mathbf{L}\pi \quad \text{with} \quad \mathbf{L} = \begin{pmatrix} [\mathbf{n}]_\times & \mathbf{v} \\ -\mathbf{v}^T & 0 \end{pmatrix}$$



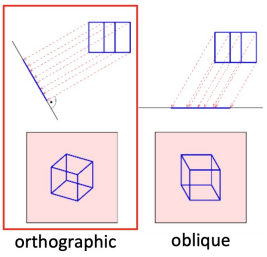
An image-space point $\mathbf{x}$ corresponds to a world space point $\mathbf{X}$, if their ray direction vectors $\mathbf{d}$ (starting from the principal point) are parallel (*point constraint*):

$$\mathbf{d}^{\mathbf{x}} \propto \mathbf{d}^{\mathbf{X}} \quad \Rightarrow \quad \mathcal{K}^{-1}\mathbf{x} \propto [\mathbf{R}\ \mathbf{t}]\mathbf{X}$$

3.3 Parallel Projection

Whereas the perpective projection looks realistic, since object size changes inversely with the distance, the parallel projection is better to obtain exact measurements of objects. That is because parallel 3D lines remain parallel in the 3D projection, since the viewing frustrum has the shape of a cuboid.

Orthographic Projection This is a special case where the projection rays are perpendicular to the projection plane. We use this projection to projection 3D points into a birds-eye-view of the scene, which is useful in robotics applications.



Oblique Projection The projection rays intersect with the image plane at a skew angle.

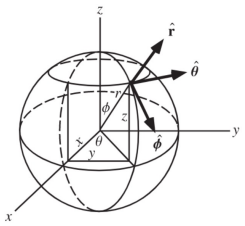
3.4 Spherical Projection

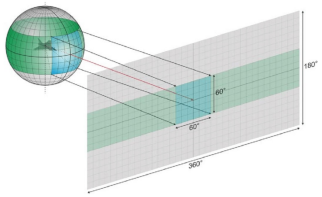
Instead of a planar projection surface, one can consider a spherical projection surface given by the unit sphere $\mathbb{S}^2 = \{\mathbf{x} \in \mathbb{R}^3 \mid \|\mathbf{x}\| = 1\}$. The spherical projection π_s of a 3D point $\boldsymbol{X}$ is given by:

$$\pi_s : \mathbb{R}^3 \to \mathbb{S}^2; \quad \boldsymbol{X} \mapsto \boldsymbol{x} = \frac{\boldsymbol{X}}{|\boldsymbol{X}|}$$

Next, we convert our Cartesian coordinates $\mathbf{x}$ to spherical coordinates azimuth $\theta = \arctan(\frac{y}{x}) \in [0, 2\pi)$ and the polar angle $\phi = \arccos(\frac{z}{r}) \in [0, \pi]$ with $r = \sqrt{x^2 + y^2 + z^2} = 1$ for a unit sphere. We get the image space coordinates by linearly mapping the spherical coordinates onto a 2D plane.

$$x = r\cos\theta\sin\phi \qquad x = r\sin\theta\sin\phi \qquad z = r\cos\phi$$



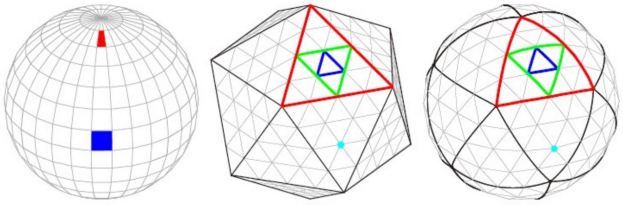


The whole projection equation becomes:

$$\lambda\boldsymbol{x}' = K\Pi_0 g\boldsymbol{X}_0$$

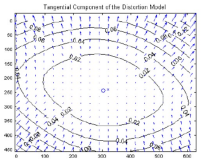
is similar to the previous one except for $\lambda = \|\boldsymbol{X}\| = \sqrt{X^2 + Y^2 + Z^2}$.

Other Expressions of a Sphere We obtain an *icosahedral spherical representation* by extruding all the vertices of icosahedron sub-faces to the unit sphere.

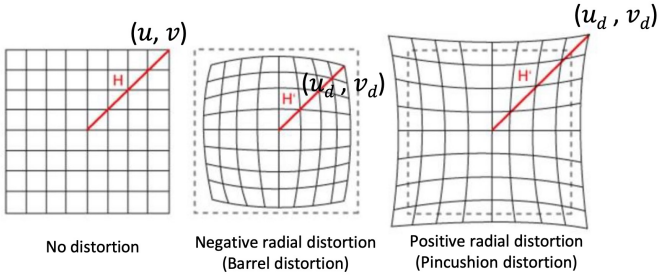


3.5 Distortion

Tangential distortion happens if lens and image sensor are not perfectly parallel. The projected image will appear stretched/staunched in certain directions.



Radial Distortion This occurs, when light rays bend more at the lens' edges than at its optical center.



In simple quadratic model of radial distortion, we calculate the distorted coordinates $\boldsymbol{x}'_d$ from (ideal) non-distorted coordinates $\boldsymbol{x}'$:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = (1 + k_1 r^2) \begin{pmatrix} u - u_0 \\ v - v_0 \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \quad \text{with} \quad r^2 = (u - u_0)^2 + (v - v_0)^2$$

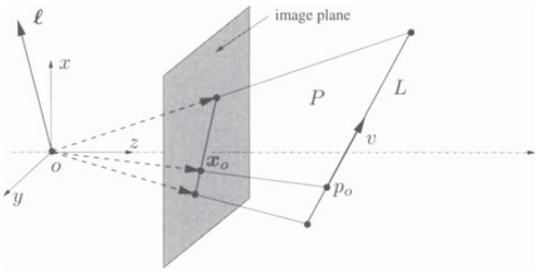
where r is the radius and k_1 is a parameter estimated during calibration.
A more general model with an arbitrary center of distortion $\mathbf{c}$ is

$$\boldsymbol{x} = \mathbf{c} + f(r)(\boldsymbol{x}_d - \mathbf{c}) \quad \text{with} \quad r = |\boldsymbol{x}_d - \mathbf{c}|$$

$$\text{and} \quad f(r) = 1 + a_1 r + a_2 r^2 + a_3 r^3 + a_4 r^4.$$

3.6 Preimage and Coimage [old]

Due to the unknown scale factor, each point is mapped not to a single point $\boldsymbol{x}$, but to an equivalence class of points $\boldsymbol{y} \sim \boldsymbol{x}$.



A line L in 3D is characterized by a base point $\boldsymbol{X}_0 = (X_0, Y_0, Z_0, 1)^T \in \mathbb{R}^4$ and a vector $\boldsymbol{V} = (V_1, V_2, V_3, 0)^T \in \mathbb{R}^4$:

$$\boldsymbol{X} = \boldsymbol{X}_0 + \mu \boldsymbol{V} \subset \mathbb{R}^4, \quad \mu \in \mathbb{R}$$

The image of the line L is given by

$$\boldsymbol{x} \sim \Pi_0 \boldsymbol{X} = \Pi_0(\boldsymbol{X}_0 + \mu \boldsymbol{V}) = \Pi_0 \boldsymbol{X}_0 + \mu \Pi_0 \boldsymbol{V}$$

All points $\boldsymbol{x}$ treated as vectors from the origin o span a 2D subspace P. The intersection of this plane P with the image plane gives the image of the line. P is called the preimage of the line.
A preimage of a point or a line in the image plane is the largest set of 3D points that give rise to an image equal to the given point or line.

The coimage of a point or a line is the subspace in $\mathbb{R}^3$ that is the unique orthogonal complement, i.e. the normal vector, of its preimage. Image, preimage and coimage are equivalent.

In the case of a line L with its 2D preimage, we have

$$\boldsymbol{\ell}^T \boldsymbol{x} = 0 \text{ and } P = \text{span}(\hat{\boldsymbol{\ell}}).$$

$\boldsymbol{\ell} \in \mathbb{R}^3$ is the normal vector to the preimage (2D subspace) of line L and $\text{span}(\hat{\boldsymbol{\ell}})$ is the span of row vectors of $\hat{\boldsymbol{\ell}}$.

	Image	Preimage	Coimage
Point	$\text{span}(\boldsymbol{x}) \cap$ im. plane	$\text{span}(\boldsymbol{x}) \subset \mathbb{R}^3$	$\text{span}(\hat{\boldsymbol{x}}) \subset \mathbb{R}^3$
Line	$\text{span}(\hat{\boldsymbol{\ell}}) \cap$ im. plane	$\text{span}(\hat{\boldsymbol{\ell}}) \subset \mathbb{R}^3$	$\text{span}(\boldsymbol{\ell}) \subset \mathbb{R}^3$

3.7 Projective Geometry

One can represent 3D points by a general 4D vector $X = (XW, YW, ZW, W) \in \mathbb{R}^4$ remembering that only the direction of this vector is of importance. We therefore identify the point in homogeneous coordinates with the line connecting it with the origin.

3.8 Supplementary Knowledge

Depth of Field This is the distance between the nearest an farthest objects in a scene that appear acceptably sharp in an image. A narrower aperture increases the depth of field, but reduces the exposure of the image (darker).

Depth Camera A camera that estimates the depth of each pixel by measuring the return time of each light ray (*time-of-flight* measurement principle).

Rolling/Global Shutter Camera With a rolling shutter, every row/pixel in the image is read on after the other at different times. This may cause distortions for moving objects. A global shutter exposes and reads all pixels simultaneously. When two images are recorded with rolling shutters that use different read-out directions (such that the motion-induced distortion is different), we can recover the motion and undistorted image from point correspondences.

Event Camera Each pixel inside an event camera operates independently and asynchronously, reporting changes in brightness as they occur, and staying silent otherwise. We use this to improve robustness, efficiency and accuracy in visual odometry.

4 Camera Calibration

Camera calibration is the process of *simultaneously* determining the extrinsic parameters $(\mathbf{R}, \mathbf{t})$ and intrinsic parameters $\mathbf{K}$, plus the lens distortion. In general, we estimate the intrinsic parameters first and subsequently obtain the extrinsic parameters from VO/SLAM.

4.1 Tsai's Method

We measure the 3D position of at least 6 3D control points on a calibration target, as well as their 2D projected coordinates. This methods uses a single image. We need to solve for the coefficients in the linear mapping from 3D world coordinates $\mathbf{p}$ to homogeneous pixel coordinates using the *direct linear transform (DLT)*:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}}_{=\mathbf{M} \in \mathbb{R}^{3\times 4}} \underbrace{\begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}}_{=\mathbf{p}} = \begin{pmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \\ \mathbf{m}_3^T \end{pmatrix} \mathbf{p}$$

Conversion to non-homogeneous pixel coordinates leads to a system of linear equations with the values of $\mathbf{M}$ as coefficients:

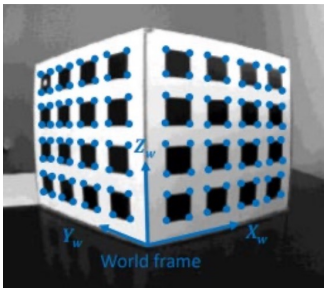
$$u = \frac{\lambda u}{\lambda} = \frac{\mathbf{m}_1^{\mathrm{T}} \cdot \mathbf{p}}{\mathbf{m}_3^{\mathrm{T}} \cdot \mathbf{p}} \qquad v = \frac{\lambda v}{\lambda} = \frac{\mathbf{m}_2^{\mathrm{T}} \cdot \mathbf{p}}{\mathbf{m}_3^{\mathrm{T}} \cdot \mathbf{p}} \quad \Rightarrow \quad \underbrace{\begin{matrix} (\mathbf{m}_1^{\mathrm{T}} - u\mathbf{m}_3^{\mathrm{T}}) \cdot \mathbf{p} = 0 \\ (\mathbf{m}_2^{\mathrm{T}} - v\mathbf{m}_3^{\mathrm{T}}) \cdot \mathbf{p} = 0 \end{matrix}}_{= \underbrace{\begin{pmatrix} \mathbf{p}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & -u\mathbf{p}^{\mathrm{T}} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{p}^{\mathrm{T}} & -v\mathbf{p}^{\mathrm{T}} \end{pmatrix}}_{\mathbf{Q}} \begin{pmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{pmatrix} = \mathbf{0}}$$

If we have n corresponding points, we can stack these equations into a big matrix $\mathbf{Q} \in \mathbb{R}^{2n \times 12}$ and use it to solve $\mathbf{Qm} = \mathbf{0}$, such that we need 6 point correspondences. Mathematically $rank(\mathbf{Q}) = 11$ suffices for a unique (up-to-scale), non-zero solution. If we have more than 6 points (over-determined solution), we find the last-squares solution (e.g. with SVD) by minimzing $\|\mathbf{QM}\|^2$, subject to the constraint $\|\mathbf{M}\|^2 = 1$ to avoid the trivial solution.

$\arg\min_b \ \|Ab\|_2^2$ subject to $\|b\|_2 = 1$	$\mathbf{A} = \mathbf{U\Sigma V}$	Optimal solution b* is the column of V corresponding to the smallest singular value.

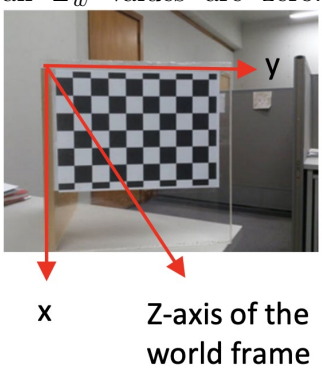
We recover an orthogonal matrix $\mathbf{R}$, vector $\mathbf{t}$ and an upper-triangular matrix $\mathbf{K}$ from $\mathbf{M}$ using the RQ-decomposition $\mathbf{M} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$.

In practice, we use a *single image* with *more than 6 non-coplanar points* (on at least two 3D surfaces). Checkerboard corners can be detected with below 0.1 pixels accuracy.



4.2 Zhang's Method

This method works with *3D coplanar points* (allowing to use a planar "calibration grid"), but relies on multiple images from different views. As opposed to Tsai's method with world coordinate frame is placed in the calibration surface, such that all Z_w values are zero.



$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \begin{pmatrix} X_w \\ Y_w \\ 0 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{pmatrix}}_{\text{Homography } \mathbf{H} \in \mathbb{R}^{3\times 3}} \begin{pmatrix} X_w \\ Y_w \\ 1 \end{pmatrix}$$

Starting from this equation, we create a system of linear equations with the values of $\mathbf{H}$ as the coefficients in the same fashion as for Tsai's method.

$$u = \frac{\lambda u}{\lambda} = \frac{\mathbf{h}_1^{\mathrm{T}} \cdot \mathbf{p}}{\mathbf{h}_3^{\mathrm{T}} \cdot \mathbf{p}} \qquad v = \frac{\lambda v}{\lambda} = \frac{\mathbf{h}_2^{\mathrm{T}} \cdot \mathbf{p}}{\mathbf{h}_3^{\mathrm{T}} \cdot \mathbf{p}} \quad \Rightarrow \quad \underbrace{\begin{matrix} (\mathbf{h}_1^{\mathrm{T}} - u\mathbf{h}_3^{\mathrm{T}}) \cdot \mathbf{p} = 0 \\ (\mathbf{h}_2^{\mathrm{T}} - v\mathbf{h}_3^{\mathrm{T}}) \cdot \mathbf{p} = 0 \end{matrix}}_{= \begin{pmatrix} \mathbf{p}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & -u\mathbf{p}^{\mathrm{T}} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{p}^{\mathrm{T}} & -v\mathbf{p}^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = \mathbf{0}}$$

Given n 2D-3D point correspondences, the result is $\mathbf{Qh} = \mathbf{0}$, with $\mathbf{Q} \in \mathbb{R}^{2n \times 9}$ and the solution vector $\mathbf{h} \in \mathbb{R}^9$. For a unique, up-to-scale, non-trivial solution, we require $rank(\mathbf{Q}) = 8$, such that we need at least 4 non-collinear points.

Recovering $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ from Multiple Views Compared to Tsai's method, we cannot obtain $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ through QR-decomposition, because our modified transformation matrix is not orthogonal. Instead, we use the fact that the intrinsic matrix is constant for all views captured with the same camera. For the jth view, we have:

$$\mathbf{H}^j = \underbrace{\begin{pmatrix} h_{11}^j & h_{12}^j & h_{13}^j \\ h_{21}^j & h_{22}^j & h_{23}^j \\ h_{31}^j & h_{33}^j & h_{33}^j \end{pmatrix}}_{[\mathbf{h}_{j1}\ \mathbf{h}_{j2}\ \mathbf{h}_{j3}]} = s \cdot \underbrace{\begin{pmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}} \cdot \underbrace{\begin{pmatrix} r_{11}^j & r_{12}^j & t_1^j \\ r_{21}^j & r_{22}^j & t_2^j \\ r_{31}^j & r_{32}^j & t_3^j \end{pmatrix}}_{[\mathbf{r}_{j1}\ \mathbf{r}_{j2}\ \mathbf{t}_j]}$$

First, we express the columns of the unkown transformation matrix using the known homographies:

$$\mathbf{r}_{j1} = \lambda \mathbf{K}^{-1}\mathbf{h}_{j1} \qquad \mathbf{r}_{j2} = \lambda \mathbf{K}^{-1}\mathbf{h}_{j2}$$
$$\mathbf{t}_j = \lambda \mathbf{K}^{-1}\mathbf{h}_{j3} \qquad \lambda = s^{-1}$$

Next, we enforce constraints on our expressions for the rotation (orthogonality and normality):

$$\mathbf{r}_{j1}^T\mathbf{r}_{j2} = 0 \quad \Rightarrow \quad \mathbf{h}_{j1}^T(\mathbf{K}^{-1})^T\mathbf{K}^{-1}\mathbf{h}_{j2} = 0$$
$$\mathbf{r}_{j1}^T\mathbf{r}_{j1} = \mathbf{r}_{j2}^T\mathbf{r}_{j2} = 1 \quad \Rightarrow \quad \begin{aligned} 1 &= \mathbf{h}_{j1}^T(\mathbf{K}^{-1})^T\mathbf{K}^{-1}\mathbf{h}_{j1} \\ &= \mathbf{h}_{j2}^T\underbrace{(\mathbf{K}^{-1})^T\mathbf{K}^{-1}}_{=\mathbf{B}}\mathbf{h}_{j2} \end{aligned}$$

where $\mathbf{B}$ is a 3×3 symmetric matrix from which we can **extract the 6 intrinsic parameters**. We find the solution by solving $\mathbf{Ab} = \mathbf{0}$ where $\mathbf{b} \in \mathbb{R}^6$ is a vector with the relevant values in $\mathbf{B}$ and $\mathbf{A} \in \mathbb{R}^{2n \times 6}$ is a matrix with 2 rows (one per constraint) for n views. Mathematically, 3 views are sufficient to find a solution, but we need $n > 3$ views in practice. Use the following fact to translate each constraint into a row of $\mathbf{A}$:

$$\mathbf{h}_a^T\mathbf{B}\mathbf{h}_b = \begin{pmatrix} h_a h_{b1} \\ h_{a1}h_{b2} + h_{a2}h_{b1} \\ h_{a2}h_{b2} \\ h_{a3}h_{b1} + h_{a1}h_{b3} \\ h_{a3}h_{b2} + h_{a2}h_{b3} \\ h_{a3}h_{b3} \end{pmatrix}^T \begin{pmatrix} B_{11} \\ B_{12} \\ B_{22} \\ B_{13} \\ B_{23} \\ B_{33} \end{pmatrix}$$

Next, we **extract the extrinsic parameters** for each view. E.g. for the jth view, the transformation matrix becomes:

$$[\mathbf{R}|\mathbf{t}]^j = \begin{pmatrix} | & | & | & | \\ \mathbf{r}_{j1} & \mathbf{r}_{j2} & (\mathbf{r}_{j1} \times \mathbf{r}_{j2}) & \mathbf{t}_j \\ | & | & | & | \end{pmatrix}$$

4.3 Image Undistortion

We account for distortion, by modifying the perspective projection model from camera to pixel coordinates by including **tangential and radial distortion coefficients**, denoted k_n and p_n respectively. Instead of $u = f_x y + c_x$ and $v = f_y y + c_y$ with image coordinates $(x\ y\ 1)^T$, we have:

$$u = f_x x_{\text{dist}} + c_x \quad \text{and} \quad v = f_y y_{\text{dist}} + c_y$$

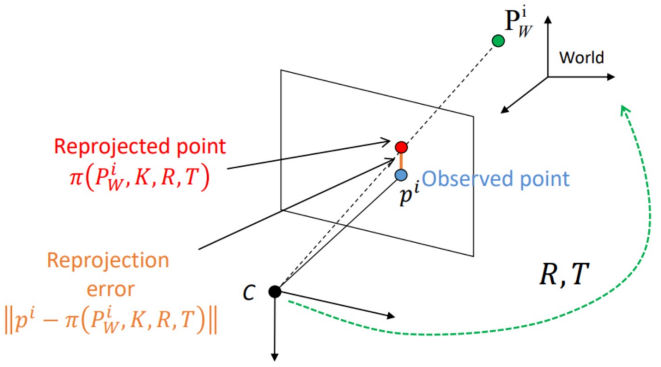
using some polynomal distortion model, e.g.:

$$x_{\text{dist}} = x\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 xy + p_2$$
$$y_{\text{dist}} = y\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1\left(r^2 + 2y^2\right) + 2p_2 xy$$
with $r^2 = x^2 + y^2$

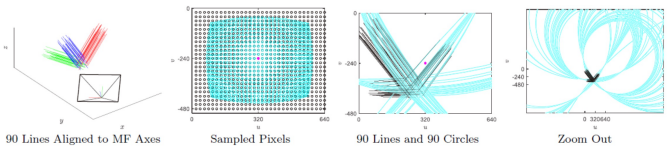
Then, we find the distortion coefficients by:

1. Initially setting $p_n = k_n = 0$.
2. Performing Zhang's method to obtain a good "initial guess" of the intrinsic and extrinsic parameters.
3. Projecting the world-space points using the computed projection matrix and calculating the re-projection error between the computed and detected image-space points.
4. Minimizing the error with gradient descent jointly w.r.t. the intrinsic, extrinsic and distortion parameters:

$$\underset{\mathbf{K},\mathbf{R},\mathbf{t},k_1,p_1,\ldots}{\operatorname{argmin}} \sum_{i=1}^{n} \left\|\mathbf{p}^i - \pi(\mathbf{P}_W^i, \mathbf{K}, \mathbf{R}, \mathbf{t}, k_1, p_1, \ldots)\right\|^2$$



Line-based Undistortion Another approach to image undistortion is by finding the intrinsic, extrinsic and distortion parameters that lead to vanishing points which maximize the number of inlier lines.

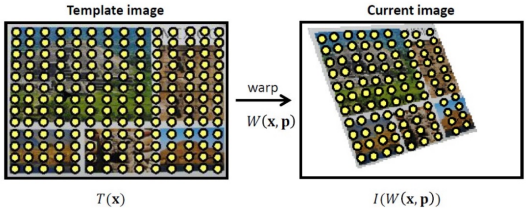


5 Estimating Point Correspondences

In practice, we observe characteristic image features like brightness or color values at individual pixels instead of points or lines. There exist two types of point matching strategies: direct and indirect methods. **Direct methods** exploit all information in an image, leading to higher accuracy and robustness, have reduced computational cost with higher frame rates (since no RANSAC is needed), but are sensitive to initialisation and only work with small frame-to-frame motions. **Indirect methods** work by detecting and matching features (points/lines), can cope with large frame-to-frame motions and strong illumination changes, but are slow due to costly feature extraction, matching and outlier removal.

5.1 Direct Methods (Optical Flow; Small Motion)

Direct methods make use of the apparent 2D motion field observable between consecutive images of a video, known as **optical flow**, which differs from the motion of objects in the scene. They estimate the parameters $\mathbf{p}$ of a transformation $W(\mathbf{x}, \mathbf{p})$ from a template image T to a new image I, by processing the change in pixel intensities. Every yellow dot in the image below represents a tracked pixel.



Note that this is a *chicken-and-egg problem*, because we have two unknown parameters (the corresponding pixels and the transformation parameters) that are mutually determined. In practice, we solve the problem by adding constraints, such as constant brightness for corresponding pixels.

We make the following assumptions: Let $\mathbf{x}(t)$ denote a moving point at time t, and $I(\mathbf{x}, t)$ a pixel intensity in a video sequence.

1. **Brightness constancy ("optical flow constraint")**: the brightness of the pixels to track does not change over consecutive frames, i.e. $I(\mathbf{x}(t), t) = \text{const.}, \forall t$. It follows that

$$\frac{d}{dt} I(\mathbf{x}(t), t) = \nabla I^T \frac{d\mathbf{x}}{dt} + \frac{\partial I}{\partial t} = 0$$

.

2. **Temporal consistency**: the motion $\mathbf{v}$ is constant over a neighborhood Ω around $\mathbf{x}$:

$$\nabla I(\mathbf{x}', t)^T \mathbf{v} + \frac{\partial I}{\partial t}(\mathbf{x}', t) = 0, \quad \forall x' \in \Omega$$

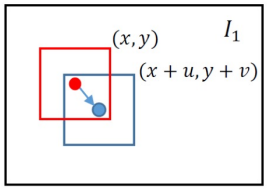
The frame-to-frame motion should measure only 1-2 pixels in practice.

3. **Spatial coherence**: All pixels in the template undergo the same transformation, which requires all points to lie on the same 3D surface and unusual in practice.

5.1.1 Kanade-Lucas-Tomasi (KLT) Tracker

This method solves two sub-problems: the **Tomasi-Kanade method** selects the most reliable image features/patches for tracking and the **Lucas-Kanade method** tracks/aligns these image features/patches.

Simplified Case (Pure Translation) We assume that all pixels in the template are moved by the same motion vector $(u \;\; v)^T$ between the template image I_0 and a new image I_1 (which is usually never the case). Now, we want to minimize the sum of squared differences (SSD) between intensities of corresponding pixels in the template and the new image:



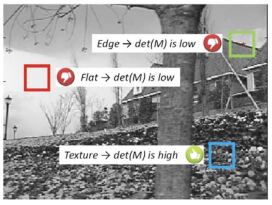
$$\begin{aligned}\text{SSD}(u,v) &= \sum_{x,y\in\Omega} (I_0(x,y) - \underbrace{I_1(x+u, y+v)}_{\approx I_1(x,y) + \frac{\partial I_1}{\partial x}u + \frac{\partial I_1}{\partial y}v})^2 \\ &\approx \sum_{x,y\in\Omega} (\Delta I - I_x u - I_y v)^2\end{aligned}$$

where Ω is the size of the image patch that we select for tracking, ΔI is the intensity change at (x, y) and $\nabla I = (I_x \;\; I_y)^T$ are the partial derivatives of I_1 at (x, y). We obtain a solution from the first-order optimality condition:

$$\begin{aligned}\nabla\text{SSD} &\approx \begin{pmatrix} -2\sum I_x(\Delta I - I_x u - I_y v) \\ -2\sum I_y(\Delta I - I_x u - I_y v)\end{pmatrix} = \mathbf{0} \\ \Rightarrow \underbrace{\begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix}}_{\mathbf{M}} \begin{pmatrix} u \\ v \end{pmatrix} &= \underbrace{\begin{pmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{pmatrix}}_{=\mathbf{b}}\end{aligned}$$

If $\mathbf{M}$ is invertible, we obtain the motion vector $(u \;\; v)^T = \mathbf{M}^{-1}\mathbf{b}$. Therefore, we need to ensure that $\det(\mathbf{M}) = \lambda_1\lambda_2\cdots\lambda_n \neq 0$, by **selecting an image patch, whose coefficient matrix $\mathbf{M}$ has large eigenvalues**.

Edges and flat patches for instance results in low and textured patches result in high determinants. Then, we **track the image patch along the computed motion vector**. In practice, we select good patches in the first image and track the same points in all subsequent images.



General Case (Warping) We assume that all pixels in the template image T undergo a warping transformation with the parameters $\mathbf{p}$:

$$\text{SSD} = \sum_{\mathbf{x}\in T}[\underbrace{I(W(\mathbf{x},\mathbf{p}+\Delta\mathbf{p}))}_{\approx I(W(\mathbf{x},\mathbf{p}))+\nabla I\frac{\partial W}{\partial \mathbf{p}}\Delta\mathbf{p}} - T(\mathbf{x})]^2$$

where we simplify the warping transformation using the first-order Taylor approximation. Finding the parameters $\mathbf{p}$ that minimize this constraint with the first-order optimality condition is difficult. Instead, we leverage the Gauss-Newton method to minimize the SSD iteratively. We obtain the parameter update vector by equating $\frac{\partial \text{SSD}}{\partial \Delta\mathbf{p}}$ to zero and solving for $\Delta\mathbf{p}$:

$$2\sum_{\mathbf{x}\in T}\left[\nabla I\frac{\partial W}{\partial \mathbf{p}}\right]^T\left[I(W(\mathbf{x},\mathbf{p}))+\nabla I\frac{\partial W}{\partial \mathbf{p}}\Delta\mathbf{p}-T(\mathbf{x})\right]=0$$

$$\Rightarrow \Delta\mathbf{p} = H^{-1}\sum_{\mathbf{x}\in T}\left[\nabla I\frac{\partial W}{\partial \mathbf{p}}\right]^T[T(\mathbf{x})-I(W(\mathbf{x},\mathbf{p}))]$$

$$\text{with } H=\sum_{\mathbf{x}\in T}\left[\nabla I\frac{\partial W}{\partial \mathbf{p}}\right]^T\left[\nabla I\frac{\partial W}{\partial \mathbf{p}}\right]$$

Alternative Derivation of the General Case The Lucas-Kanade method generates sparse flow vectors under the assumption of constant motion in a local neighborhood (temporal consistency assumption). It follows that the least square error of point $\mathbf{x}$ is:

$$\begin{aligned}E(\mathbf{v}) &= \int_{W(\mathbf{x})}|\nabla I(\mathbf{x}',t)^T\mathbf{v}+I_t(\mathbf{x}',t)|^2 d\mathbf{x}'\\ &= \mathbf{v}^T\mathbf{M}\mathbf{v}+2\mathbf{q}^T\mathbf{v}\end{aligned}$$

$$\text{with } \mathbf{M}=\int_{W(x)}\nabla I\nabla I^T\,\mathrm{d}\mathbf{x}' \qquad \mathbf{q}=\int_{W(x)}I_t\nabla I\,\mathrm{d}\mathbf{x}'$$

$$\nabla I=\begin{pmatrix}I_x\\I_y\end{pmatrix}=\begin{pmatrix}\frac{\partial I}{\partial x}\\\frac{\partial I}{\partial y}\end{pmatrix} \qquad I_t=\frac{\mathrm{d}I}{\mathrm{d}t}$$

We set $\frac{dE}{d\mathbf{v}}=0$ to get the local flow vector (velocity):

$$\mathbf{v}=\frac{d\mathbf{x}}{dt}=-\mathbf{M}^{-1}\mathbf{q}$$

given that $\mathbf{M}$ is invertible ($\det(\mathbf{M})\neq 0$). $\mathbf{M}$ is PSD.

$$\text{Solution:}\quad \mathbf{v}=\begin{pmatrix}\frac{m_{22}q_1-m_{12}q_2}{m_{12}^2-m_{11}m_{22}}\\\frac{m_{11}q_2-m_{12}q_1}{m_{12}^2-m_{11}m_{22}}\end{pmatrix}\in\mathbb{R}^2$$

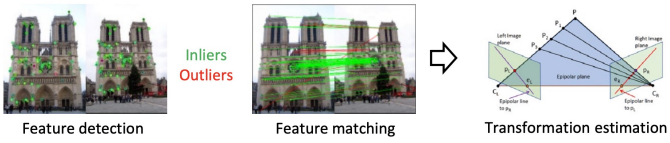
Aperture problem It is impossible to determine the motion $\mathbf{v}$ orthogonal to the gradient direction $\nabla\mathbf{I}$ in regions with constant gradient direction. In these regions $\det\mathbf{M}=0$.

KLT Tracker Algorithm

1. For time t, at each point $\mathbf{x}\in\Omega$ compute:
$$M(x)=\int_\Omega\begin{pmatrix}I_x^2 & I_xI_y\\I_xI_y & I_y^2\end{pmatrix}\,\mathrm{d}\mathbf{x}'.$$
2. Keep all points $\mathbf{x}\in\Omega$ for which $\det(\mathbf{M}(\mathbf{x}))$ is greater or equal to a threshold $\theta>0$.
3. The local velocity of these points is given by:
$$\mathbf{b}(\mathbf{x},t)=-\mathbf{M}(\mathbf{x})^{-1}\begin{pmatrix}\int I_xI_t\,\mathrm{d}\mathbf{x}'\\\int I_yI_t\,\mathrm{d}\mathbf{x}'\end{pmatrix}.$$
4. Repeat for the points $\mathbf{x}+\mathbf{b}$ at time $t+1$.

5.2 Indirect Methods (Large Motion)

We compute the transformation between two images by: **1) detecting and matching features**; **2) computing the transformation** and 3) optionally refining the initial estimate by minimizing a squared sum of reprojection errors.



5.2.1 Feature Detectors

The feature detector is responsible for detecting and matching features that are invariant to scale, rotation and view point changes. A **blob** is a group of connected pixels in an image that share some common property (e.g. the same grayscale value). Blobs have less localization accuracy than corners, but are more distinctive. A **corner** is defined a the intersection of two or more edges. They have high localization accuracy, but are less distinctive than blobs.

Moravec's Corner Detector In the region around a corner, the image gradient is large in all directions, which is not true for edges or flat regions. Thus, shifting a window horizontally, vertically or diagonally should cause large intensity changes around a corner. For a given patch:

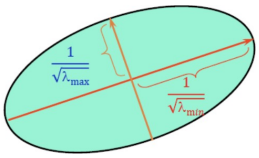
1. compute the SSD between all pairs of corresponding pixels when shifting the window along all directions, e.g. for a shift $\Delta\mathbf{x}$:
$$\text{SSD}(\Delta\mathbf{x})=\sum_{\mathbf{x}\in\Omega}(I(\mathbf{x})-I(\mathbf{x}+\Delta\mathbf{x})^2$$

2. keep the lowest SSD (*interest measurement*),
3. if the interest measurement is higher than a threshold, the patch center is a corner.

Harris' Corner Detector This is an **efficient** version of Moravec's method, since it does not explicitly shift the window. We approximate the SSD:

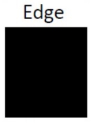
$$\begin{aligned}\text{SSD}(\Delta\mathbf{x}) &\approx \sum_{\mathbf{x}\in\Omega}(I_x(\mathbf{x})\Delta x + I_y(\mathbf{x})\Delta y)^2 \\ &\approx \underbrace{(\,\Delta x\ \Delta y\,)}_{\substack{\text{manually}\\ \text{assigned}}} \underbrace{\begin{pmatrix}\sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2\end{pmatrix}}_{\substack{\mathbf{M}\text{, encodes the}\\ \text{SSD change}}} \begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}\end{aligned}$$

We can efficiently compute $\mathbf{M}$ using $\mathbf{M} = \mathbf{R}^{-1}\mathbf{\Lambda}\mathbf{R}$, using the eigenvector matrix $\mathbf{R}$ and diagonal eigenvalue matrix $\mathbf{\Lambda}$, where the largest eigenvalue corresponds to the direction of the quickest intensity change. Accordingly, both eigenvalues should be large for a corner-patch.




Edge

$$M = \begin{bmatrix}\sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2\end{bmatrix} = \begin{bmatrix}0 & 0\\ 0 & \lambda_2\end{bmatrix}$$


Flat region

$$M = \begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$


Corner

$$M = \begin{bmatrix}\cos\frac{\pi}{4} & -\sin\frac{\pi}{4}\\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4}\end{bmatrix}\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}\begin{bmatrix}\cos\frac{\pi}{4} & \sin\frac{\pi}{4}\\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4}\end{bmatrix}$$

Harris' corner detector is **not scale invariant**, because the window size is fixed.

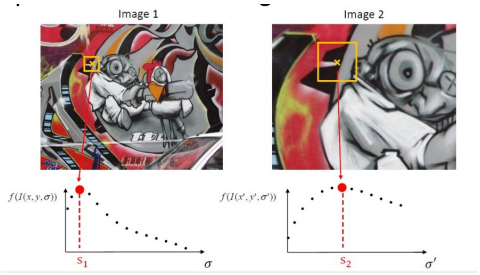
5.2.2 Feature Matching

Feature Descriptor Each detected feature (pixel/patch/blob) is described by one or more *descriptors*, which describes the pixel information around a feature as a scalar or vector. It should be *distinctive* (unique w.r.t. other features), *robust to geometric* (scale-, rotation- & view-point-invariance) and *illumination changes*.

Brute Force Matching Given two images with N features each, nearest neighbors search requires N^2 comparisons. Validating each right-image-patch at S tentative scales for all N left-image-patches creates the complexity $\mathcal{O}(N^2S)$. Another drawback of this method is that we assume the scale of the left images' patches as optimal. We avoid unreliable points at repetitive image patterns, by comparing the ratio of distances to the 1st and 2nd closest descriptors to a threshold.

General Pipeline Given two feature descriptors $H(\mathbf{x})$ and $F(\mathbf{x})$ from different images:

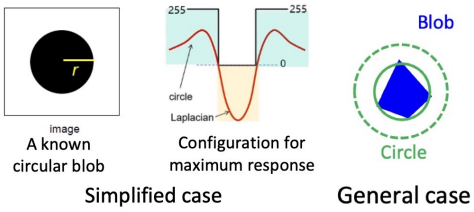
1. **Determine the scale, rotation and viewpoint change of each patch.** We determine the *scale* by assigning a function to each patch, that maps the patch size σ to some value and has extrema that are invariant to the path size. Then, the sizes σ_1 and σ_2 at which the functions associated to two patches achieve (the same) extremum are the sizes at which the patches have the same scale (correspond to the same 3D area).



Our function is a convolution of the image with the Laplacian of the Gaussian kernel (LoG):

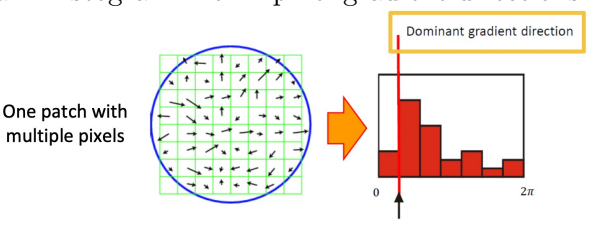
$$\text{LoG}(x, y, \sigma) = \nabla^2 G_\sigma(x, y) = \frac{\partial^2 G_\sigma(x, y)}{\partial x^2} + \frac{\partial^2 G_\sigma(x, y)}{\partial y^2}$$

where $\sigma = \frac{r}{\sqrt{2}}$ is the kernel "radius" (scale) and needs adjustment. Mathematically, we find the scale that leads to the maximum convolution response with the image patch (e.g. a blob).



In practice, we find σ by trying some candidates. Once σ_1 and σ_2 are known, we can rescale both patches to a canonical patch size. This way, we determine the **scale of each patch individually & automatically**. We filter overlapping blobs using non-maximum-suppression (NMS).

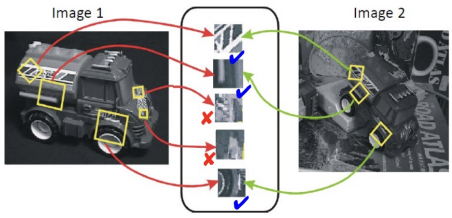
Next, we find the patch *rotation* via the eigenvectors of its Harris' matrix $\mathbf{M}$ or by extracting the dominant directions from a histogram of pixel-gradient-directions.



Finally, we account for the *viewpoint*, by computing the computing the $\mathbf{M}$ matrix and nor-

malizing the ellipse that results from its eigenvectors and eigenvalues into a circular patch.

2. **Warp each patch into a canonical patch.**



3. **Establish patch correspondences based on similarity of the warped patches** using brute force search and the sum of squared differences (SSD), sum of absolute differences (SAD) or normalized cross correlation (NCC):

$$\text{SSD} = \sum_{\mathbf{x}\in\Omega} (H(\mathbf{x}) - F(\mathbf{x}))^2 \in [0, \infty)$$

$$\text{SAD} = \sum_{\mathbf{x}\in\Omega} |H(\mathbf{x}) - F(\mathbf{x})| \in [0, \infty)$$

$$\text{NCC} = \frac{\sum_{\mathbf{x}\in\Omega} (H(\mathbf{x}) - \mu_H)(F(\mathbf{x}) - \mu_F)}{\sqrt{\sum_{\mathbf{x}\in\Omega} (H(\mathbf{x}) - \mu_H)^2} \sqrt{\sum_{\mathbf{x}\in\Omega} (F(\mathbf{x}) - \mu_F)^2}} \in [-1, 1]$$

$$\text{with } \mu_H = \frac{1}{|\Omega|} \sum_{\mathbf{x}\in\Omega} H(\mathbf{x}) \qquad \mu_F = \frac{1}{|\Omega|} \sum_{\mathbf{x}\in\Omega} F(\mathbf{x})$$

where μ_H and μ_F are used to account for the difference in average intensity of the two images.

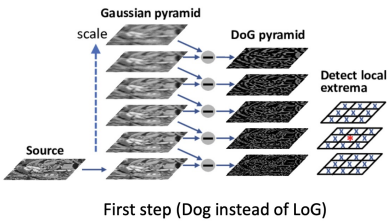
The "automatic" matching strategy allows to determine the scale (out of S candidates) for N patches per image before doing the matching (complexity $\mathcal{O}(2NS + N^2)$). The drawback of this pipeline, is that small errors/noise in rotation, scale and variance will affect the warping and thus the matching score.

5.2.3 Scale-invariant Feature Transform (SIFT)

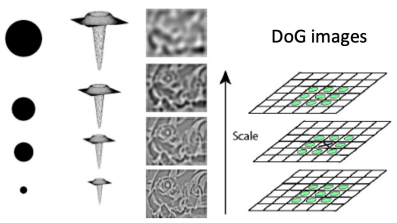
Instead of comparing the pixels of warped patches with SSD (patch descriptors), we compare the **census descriptors** associated with those warped patches, which is less sensitive to noise. We extract key points via the more efficient *difference of Gaussian* (DoG) kernel:

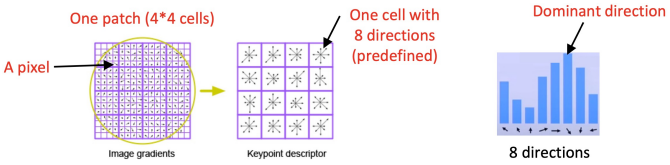
$$\text{LoG}(x, y) \approx \text{DoG}(x, y) = G_{k\sigma}(x, y) - G_{\sigma}(x, y),$$

where we **first apply two Gaussian kernels at different scales** $k\sigma$ to an image (producing two blurred images of the same size) and then take their difference.



We compare each pixel a DoG image to its 26 neighbors (8 at the same scale and 9 at the adjacent lower and upper scales respectively). If the pixel is a local extrema w.r.t. its neighbors it is a SIFT feature/keypoint $(x, y, k\sigma)$. Optionally, we obtain additional keypoints by finding local maxima in the DoG images at multiple downscaled resolution (a.k.a. at different *octaves*). We compute the *census descriptor* of a keypoint by considering the circular region around it, where the radius is determined by σ, and de-rotating the patch according to its dominant direction. **Secondly**, we divide the patch into e.g. 4×4 cells, generate histograms with the 8 dominant gradient directions for of each cell and concatenate all histograms into a 1D vector (the SIFT feature).



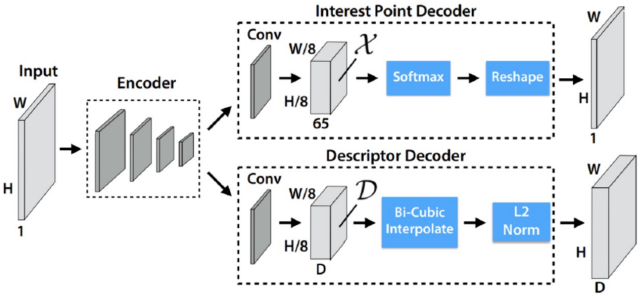


The complete output of the algorithm is:

- the patch location (x, y),
- the patch scale $k\sigma$,
- the patch orientation and
- the $4 \times 4 \times 8 = 128$ element-long 1D vector.

5.3 Deep Learning-based Method: SuperPoint

This architecture ("Self Supervised Interest Point Detection and Description", 2018) regresses the keypoint locations and the descriptor feature vectors simultaneously. Its keypoint detections are less accurate than SIFT and it is less efficient, but its descriptors are better.



6 2D-2D Geometry (Reconstruction from Two Views: Linear Algorithms)

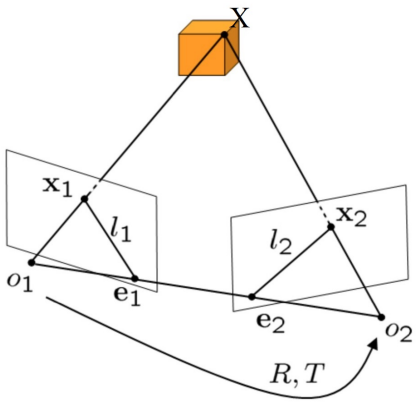
6.1 The Reconstruction Problem

We assume, that (1) a **set of corresponding points in two frames** from the same camera is given; (2) the **intrinsic camera parameters are known**; and (3) the **scene is static**.
We estimate the relative transformation between camera frames and then perform 3D reconstruction by triangulation.

Scale Ambiguity This is the phenomenon that down-scaling an object and moving it away from the image image plane have the same effect in image space. Thus, in the *monocular* case, we are only able to recover 5 DoF (3 rotation parameters and 2 parameters that describe the direction of translation). Using a stereo or depth camera, we can recover the scale/length of the translation as well.

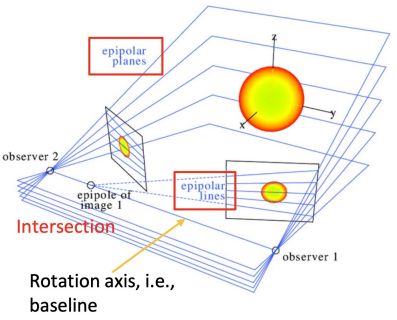
Problem Formulation Given n image-space point correspondences with $4n$ knowns, we want to approximate $5 + 3n$ unknowns. A solution exists iff $n \geq 5$, although we need more point correspondences in practice.

Notation:



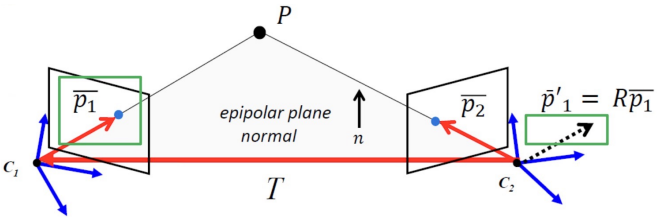
- $\mathbf{X}$: point in camera 1's coordinate system
- $\mathbf{x}_1$ and $\mathbf{x}_2$: projections of $\mathbf{X}$
- $\mathbf{o}_1$ and $\mathbf{o}_2$: optical centers of each camera
- epipolar plane: plane span by the optical centers $(\mathbf{o}_1, \mathbf{o}_2)$ and $\mathbf{X}$. For the volume (triple product) of the corresponding parallelepiped follows $\mathbf{x}_2^T(\mathbf{T} \times \mathbf{R}\mathbf{x}_1) = \mathbf{x}_2^T \hat{\mathbf{T}}\mathbf{R}\mathbf{x}_1 = 0$.
- epipoles $\mathbf{e}_1$ and $\mathbf{e}_2$: intersections of the line $(\mathbf{o}_1, \mathbf{o}_2)$ with each image plane in or outside of the image boundaries. They are also the intersection of different epipolar lines.
- epipolar lines $\mathbf{l}_1$ and $\mathbf{l}_2$: intersections between the epipolar plane and the image planes

Different point-projections effectively rotate the epipolar plane around the connection between the principle points (a.k.a. **baseline**). Each plane "casts" the epipolar lines onto the image plane. These lines intersect in the epipoles.



6.2 The Epipolar Constraint

Two corresponding points lie on their corresponding epipolar lines.



We define normalized image coordinates $\bar{\mathbf{x}}_1 = (\bar{u}_1\ \bar{v}_1\ 1)^T$ and $\bar{\mathbf{x}}_2$ (which requires knowledge of the intrinsics). Using the orthogonality of the epipolar plane normal with the image points $\bar{\mathbf{x}}_2^T \cdot \mathbf{n} = 0$, we have:

$$\bar{\mathbf{x}}_2^T(\underbrace{\mathbf{T} \times \bar{\mathbf{x}}_1'}_{=\mathbf{n}}) = \bar{\mathbf{x}}_2^T(\mathbf{T} \times (\mathbf{R}\bar{\mathbf{x}}_1)) = \bar{\mathbf{x}}_2^T \underbrace{[\mathbf{T}]_\times \mathbf{R}}_{=\mathbf{E}} \bar{\mathbf{x}}_1 = 0$$

where $\bar{\mathbf{x}}_1'$ is the point $\bar{\mathbf{x}}_1$ expressed in coordinate frame 2 and $\mathbf{E}$ is the *essential matrix*. The space of all essential matrices is called the essential space $\mathcal{E} = \{\hat{\mathbf{T}}\mathbf{R} \mid \mathbf{R} \in SO(3), \mathbf{T} \in \mathbb{R}^3\}$.

Essential Matrix A nonzero matrix $\mathbf{E} = [\mathbf{T}]_\times \mathbf{R} \in \mathbb{R}^{3\times 3}$ is an essential matrix iff it has a SVD $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ with $\mathbf{\Sigma} = \text{diag}\{\sigma, \sigma, 0\}$ for some $\sigma > 0$ and $\mathbf{U}, \mathbf{V} \in SO(3)$. For the essential matrix, $\det \mathbf{E} = 0$ and $\mathbf{E}\mathbf{E}^T\mathbf{E} - \frac{1}{2}\text{tr}(\mathbf{E}\mathbf{E}^T)\mathbf{E} = \mathbf{0}$ hold.

Fundamental Matrix Formulating the epipolar constraint with un-normalized image coordinates $(\mathbf{x}_1, \mathbf{x}_2)$ via $\bar{\mathbf{x}}_i = \mathbf{K}_i^{-1}\mathbf{x}_i$ leaves:

$$\mathbf{x}_2^T \underbrace{\mathbf{K}_2^{-T}\mathbf{E}\mathbf{K}_1^{-1}}_{=\mathbf{F}} \mathbf{x}_1 = 0$$

where $\mathbf{F}$ is the fundamental matrix. Its advantage over the essential matrix $\mathbf{E}$ is that we do not need

to normalize image coordinates via the intrincs matrices, such that this approach works **without calibration**.

6.3 Eight-Point Algorithm

The eight-point method uses ≥ 8 correspondences to estimate the 9 parameters of $\mathbf{E}$ under the assumption that they are independent. Procedure:

1. Recover the essential matrix $\mathbf{E}$ from the epipolar constraints of a set of point correspondences.
2. Extract the relative rotation & transl. from $\mathbf{E}$.

Let

$$\mathbf{e} = (e_{11}\ e_{21}\ e_{31}\ e_{12}\ e_{22}\ e_{32}\ e_{13}\ e_{23}\ e_{33})^T \in \mathbb{R}^9$$

be the stack of $\mathbf{E}$ and

$$\mathbf{a} \equiv \boldsymbol{x}_1 \otimes \boldsymbol{x}_2 = (x_1x_2, x_1y_2, x_1z_2, y_1x_2, y_1y_2, y_1z_2, z_1x_2, z_1y_2, z_1z_2)^T \in \mathbb{R}^9$$

be the Kronecker product of the vectors $\boldsymbol{x}_i \in \mathbb{R}^3$. The epipolar constraint can be written as:

$$\mathbf{x}_2^T\mathbf{E}\mathbf{x}_1 = \mathbf{a}^T\mathbf{e} = 0$$

For n point pairs, we can combine this into the linear system ($\mathbf{Q} \in \mathbb{R}^{n\times 9}$):

$$\mathbf{Q}\mathbf{e} = \mathbf{0}, \qquad \text{with } \mathbf{Q} = \begin{pmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_n^T \end{pmatrix}$$

At least $\text{rank}(\mathbf{Q}) = 8$ point pairs are needed for a unique solution (up to scale). In degenerated cases, the solution is not unique even for 8 or more point pairs (e.g. when all points lie on a plane). In an overdetermined system, we find a solution by minimizing $\|\mathbf{Q}\mathbf{e}\|^2$ subject to the constraint $\|\mathbf{e}\|^2 = 1$.

Numerical Instability of $\mathbf{Q}\mathbf{e} = \mathbf{0}$ In practice, $\mathbf{Q}$'s columns have different orders of magnitude, leading to a poor least-squares solution. The **normalized 8-point algorithm** solves this, by rescaling the two point sets to zero mean and standard deviation $\sqrt{2}$.

1. Normalize the points: $\hat{\mathbf{p}}_i = \frac{\sqrt{2}}{\sigma}(\mathbf{p}_i - \mu)$

   with $\mu = \frac{1}{N}\sum_{i=1}^{n}\mathbf{p}_i$ and $\sigma = \frac{1}{N}\sum_{i=1}^{n}\|\mathbf{p}_i - \mu\|^2$

   or $\hat{\mathbf{p}}_i = \mathbf{B}_{1/2}\mathbf{p}_i$ with $\mathbf{B}_{1/2} = \begin{pmatrix} \frac{\sqrt{2}}{\sigma} & 0 & -\frac{\sqrt{2}}{\sigma}\mu_x \\ 0 & \frac{\sqrt{2}}{\sigma} & -\frac{\sqrt{2}}{\sigma}\mu_y \\ 0 & 0 & 1 \end{pmatrix}$

2. Estimate the normalized $\hat{\mathbf{F}}$ from $n > 8$ normalized point pairs, by solving

$$\hat{\mathbf{p}}_2^T\hat{\mathbf{F}}\hat{\mathbf{p}}_1 = \hat{\mathbf{p}}_2^T\underbrace{\mathbf{B}_2^T\hat{\mathbf{F}}\mathbf{B}_1}_{=\mathbf{F}}\hat{\mathbf{p}}_1 = 0$$

3. Retrieve the unnormalized $\mathbf{F}$.

Projecting $\mathbf{e}$ onto essential space [old] Numerically estimated coefficients $\mathbf{e}$ of the essential matrix $\mathbf{E}$ may not match its properties. Thus, we reproject them onto to the essential space $\mathcal{E}$. Let $\mathbf{M} \in \mathbb{R}^{3\times 3}$ be an arbitrary matrix with SVD $\mathbf{M} = \mathbf{U}\text{diag}\{\lambda_1, \lambda_2, \lambda_3\}\mathbf{V}^T$, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Then, the essential matrix $\mathbf{E}$ minimizing the Frobenius norm $\|\mathbf{M} - \mathbf{E}\|_f^2$ is given by

$$\mathbf{E} = \mathbf{U}\cdot\text{diag}\{\sigma, \sigma, 0\}\cdot\mathbf{V}^T \quad \text{with } \sigma = \frac{\lambda_1 + \lambda_2}{2}$$

6.4 Five-Point Method

When considering dependencies between elements of $\mathbf{E}$, **5 correspondences suffice** (*minimal solution*). Since $\mathbf{R}$ and $\mathbf{T}$ have 3 and 2 DoF respectively, we have $\text{rank}(\mathbf{Q}) = 5$ and the null space has dimension 4. We can compute a null space basis $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W} \in \mathbb{R}^9$ from $\mathbf{Q} \in \mathbb{R}^{5\times 9}$, which spans the solution space of $\mathbf{Q}\mathbf{e} = \mathbf{0}$. We define our solution as

$$\mathbf{e} = x\mathbf{X} + y\mathbf{Y} + z\mathbf{Z} + w\mathbf{W}$$

with unknown coefficients (x, y, z) and $w = 1$, since the scale DoF is irrelevant. We find these unknowns by substituting $\mathbf{e}$ into two essential matrix properties that provide a non-linear system with 3 constraints:

$$\det\mathbf{E} = 0 \qquad \mathbf{E}\mathbf{E}^T\mathbf{E} - \frac{1}{2}\text{tr}(\mathbf{E}\mathbf{E}^T)\mathbf{E} = \mathbf{0}$$

6.5 Pose Recovery from E

Derivation We recover $\mathbf{R}$ and $\mathbf{t}$ from the essential matrix $\mathbf{E}$ by computing its SVD $\mathbf{E} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ (enforcing $\mathbf{U}, \mathbf{V} \in SO(3)$) and decomposing $\mathbf{\Sigma}$ further, which conveniently produces a skew-symmetric and an orthogonal matrix:

$$\begin{aligned}\mathbf{E} &= \mathbf{U}\begin{pmatrix}\sigma&0&0\\0&\sigma&0\\0&0&0\end{pmatrix}\mathbf{V}^T = \mathbf{U}\begin{pmatrix}0&\pm\sigma&0\\\mp\sigma&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&\mp1&0\\\pm1&0&0\\0&0&1\end{pmatrix}\mathbf{V}^T \\ &= \mathbf{U}\begin{pmatrix}0&\pm\sigma&0\\\mp\sigma&0&0\\0&0&0\end{pmatrix}\underbrace{\mathbf{U}^T\mathbf{U}}_{=\mathbf{I}}\begin{pmatrix}0&\mp1&0\\\pm1&0&0\\0&0&1\end{pmatrix}\mathbf{V}^T = [\mathbf{t}]_\times\mathbf{R}\end{aligned}$$

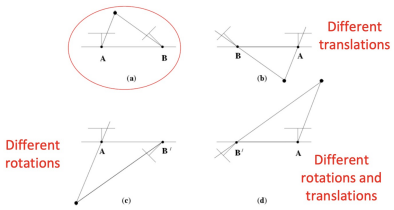
For $\mathbf{t}$, we conclude:

$$[\mathbf{t}]_\times = [\mathbf{U}\begin{pmatrix}0\\0\\\pm\sigma\end{pmatrix}]_\times \Rightarrow \mathbf{t} = \begin{pmatrix}\mathbf{U}_0&\mathbf{U}_1&\mathbf{U}_2\end{pmatrix}\begin{pmatrix}0\\0\\\pm\sigma\end{pmatrix} = \pm\mathbf{U}_2\sigma$$

Result Formally, we write the decomposition of $\mathbf{\Sigma}$ via a Z-axis-rotation by about $\pm\frac{\pi}{2}$, i.e. $\mathbf{R}_Z(\pm\frac{\pi}{2}) = \begin{pmatrix}0&\mp1&0\\\pm1&0&0\\0&0&1\end{pmatrix}$. We obtain four possible solutions:

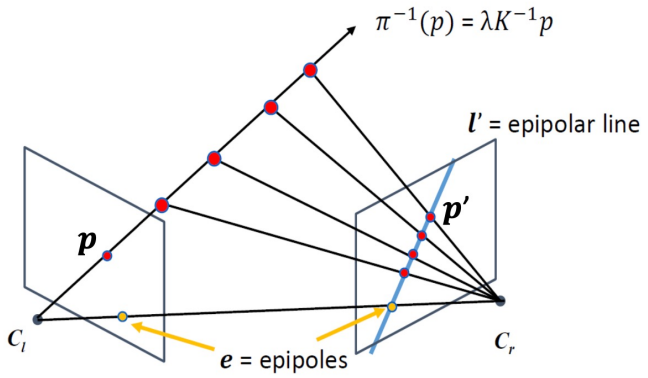
$$\begin{aligned}\mathbf{R}_1 &= \mathbf{U}\mathbf{R}_Z(+\frac{\pi}{2})\mathbf{V}^T & [\mathbf{t}_1]_\times &= \mathbf{U}\mathbf{R}_Z(+\frac{\pi}{2})\mathbf{\Sigma}\mathbf{U}^T \\ \mathbf{R}_2 &= \mathbf{U}\mathbf{R}_Z(-\frac{\pi}{2})\mathbf{V}^T & [\mathbf{t}_2]_\times &= \mathbf{U}\mathbf{R}_Z(-\frac{\pi}{2})\mathbf{\Sigma}\mathbf{U}^T\end{aligned}$$

In general, only **one of these gives meaningful (positive) depth values**, where the points lie in front of both cameras.



6.6 1D Correspondence Search

By searching for point correspondence only along epipolar lines we **simplyfiy the brute-force correspondence search from 2D to 1D**. We find the correspondence to a point in a the left image by testing points in the right image that lie on the respective epipolar line.



6.7 3D Reconstruction

We recover the 3D structure and poses by computing the intersection of corresponding rays, using 2D-2D-correspondences as input.

6.7.1 Triangulation (General Case)

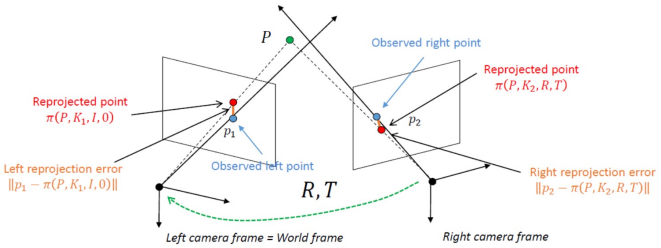
Triangulation is the problems of determining the 3D position of a point given a set of 2D points for a "left" and "right" camera and known camera intrinsics and extrinsics (e.g. from Tsai's method and the epipolar constraint respectively). In this case, we reconstruct the 3D structure **sparsely**. By convention we express all points in the frame of the left camera. Given a correspondence-pair, produce a linear system of 6 equations with 4 constraints and 3 unknowns:

$$\lambda_1 \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \underbrace{\mathbf{K}_1[\mathbf{I}|\mathbf{0}]}_{=\mathbf{M}_1} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad \overset{\mathbf{p}_1,\mathbf{P} \text{ are collinear}}{\Rightarrow} \quad \mathbf{0} = [\mathbf{p}_1]_\times \mathbf{M}_1 \cdot \mathbf{P}$$

$$\lambda_2 \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = \mathbf{K}_2[\mathbf{R}|\mathbf{T}] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \quad \Rightarrow \quad \mathbf{0} = [\mathbf{p}_2]_\times \mathbf{M}_2 \cdot \mathbf{P}$$

from which we compute the intersecting 3D point. Geometrically, due to noise we find the midpoint of the shortest 3D line segment connecting the two rays via least-squares-approximation (e.g. with SVD). We can improve this approximation further by minimizing the reprojection-error:

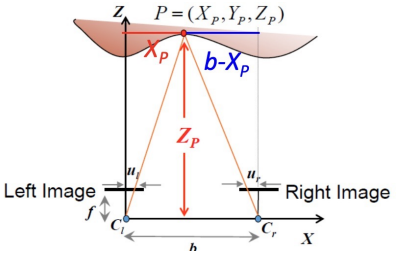
$$\mathbf{P} = \text{argmin}(\|\mathbf{p}_1 - \pi(\mathbf{P}, \mathbf{K}_1, \mathbf{I}, \mathbf{0})\|^2 + \|\mathbf{p}_2 - \pi(\mathbf{P}, \mathbf{K}_2, \mathbf{R}, \mathbf{T})\|^2)$$



6.7.2 Stereo Vision (Simplified Case)

The simplified case requires preprocessing of the camera planes based on known extrinsic camera parameters, such that the epipolar lines become horizontal and co-linear. This allows reconstructing the 3D structure **densely** (for every pixel).

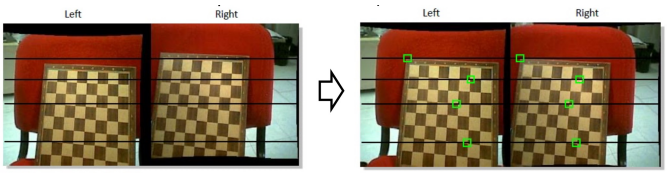
Disparity & Depth We use the disparity $u_r - u_l$ between correspondences between the left and right *X-axis-aligned* images to determine depth. Specifically, we compute the depth from similar triangles using the baseline measurement b:



$$\frac{f}{Z_P} = \frac{u_l}{X_P} \qquad \frac{f}{Z_P} = \frac{-u_l}{b - X_P} \qquad \Rightarrow \quad Z_P = \frac{bf}{u_r - u_l}$$

Using a large baseline b reduces the depth error, but complicates the search problem for close objects (since projections may be outside of one plane). Inversely for small baselines.

Stereo Rectification Stereo rectification warps the left and right images into new "rectified" images such that the *epipolar lines are collinear and coincide with the horizontal scanlines.*



Given a perspective projection $\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{K} \left(\mathbf{R}_{W \to C} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + \mathbf{T}_{W \to C} \right)$, first we reformulate the transformation such that is goes from camera

to world frame, where the translation vector $\mathbf{T}_{C\to W}$ is the position of the cameras' optical centers and $\mathbf{R}_{W\to C} = \mathbf{R}_{C\to W}^{-1}$:

$$\lambda_L \begin{pmatrix} u_L \\ v_L \\ 1 \end{pmatrix} = \mathbf{K_L} \mathbf{R}_L^{-1} \left(\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} - \mathbf{C}_L \right)$$
$$\lambda_R \begin{pmatrix} u_R \\ v_R \\ 1 \end{pmatrix} = \mathbf{K_R} \mathbf{R}_R^{-1} \left(\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} - \mathbf{C}_R \right)$$

In theory, if the matrices $\{\mathbf{K}_L, \mathbf{K}_R\}$ and $\{\mathbf{R}_L, \mathbf{R}_R\}$ are the same, the image planes are aligned. In the second step, we define "new" cameras and remove the unknown $\mathbf{X}_w$ by expressing it via its projection:

$$\hat{\lambda}_L \begin{pmatrix} \hat{u}_L \\ \hat{v}_L \\ 1 \end{pmatrix} = \hat{\mathbf{K}}\hat{\mathbf{R}}^{-1} \left(\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} - \mathbf{C}_L \right) = \lambda_L \hat{\mathbf{K}}\hat{\mathbf{R}}^{-1}\mathbf{R}_L\mathbf{K}_L^{-1} \begin{pmatrix} u_L \\ v_L \\ 1 \end{pmatrix}$$
$$\hat{\lambda}_R \begin{pmatrix} \hat{u}_R \\ \hat{v}_R \\ 1 \end{pmatrix} = \lambda_R \hat{\mathbf{K}}\hat{\mathbf{R}}^{-1}\mathbf{R}_R\mathbf{K}_R^{-1} \begin{pmatrix} u_R \\ v_R \\ 1 \end{pmatrix}$$

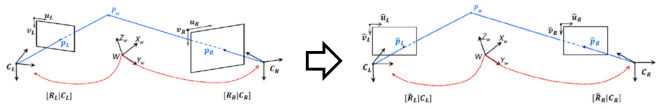
We choose the new shared $\hat{\mathbf{K}}$ and $\hat{\mathbf{R}}$ from:

$$\hat{\mathbf{K}} = \frac{\mathbf{K}_L + \mathbf{K}_R}{2} \qquad \hat{\mathbf{R}} = \begin{pmatrix} | & | & | \\ \hat{\mathbf{r}}_1 & \hat{\mathbf{r}}_2 & \hat{\mathbf{r}}_3 \\ | & | & | \end{pmatrix}$$

with $\hat{\mathbf{r}}_1 = \dfrac{\mathbf{C}_R - \mathbf{C}_L}{\|\mathbf{C}_R - \mathbf{C}_L\|}$ (makes image planes parallel to the baseline)

and $\hat{\mathbf{r}}_2 = \underbrace{\mathbf{r}_{3L}}_{\text{3rd column}} \times \hat{\mathbf{r}}_1$ $\qquad \hat{\mathbf{r}}_3 = \hat{\mathbf{r}}_1 \times \hat{\mathbf{r}}_2$

Notably, although we transform the 3D image planes here, this transformation remaps the pixel coordinates in 2D.

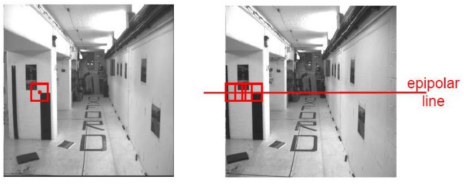


Disparity Map To Point Cloud After stereo rectification and correspondence search, we calculate the disparity for every correspondence. Closer objects have bigger disparity and appear brighter in the disparity map.

We compute the depth for every pixel and get 3D points with:

$$z = \text{depth}(i,j) \qquad x = \frac{(j - c_x)\cdot z}{f_x} \qquad y = \frac{(i - c_y)\cdot z}{f_y}$$

Dense Correspondence Search We conduct dense correspondence search along the same scanlines of rectified left and right images. Comparing via **block-wise similarity** (using a window around each point) is more reliable than pixel-wise similarity. Increasing the window size, smoothes out noise, but captures less detail, resulting in smoothed disparity maps.

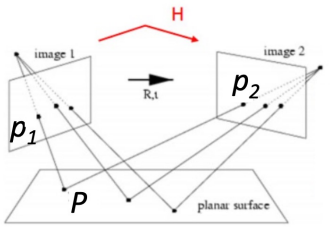


We improve block-based matching by use of "soft" constraints besides the epipolar constraint. The *disparity gradient* uses the fact that the disparity changes smoothly between points that lie on the same surface.

As opposed to the **general case (sparse correspondences)**, in the **simplified case (dense correspondences)** descriptors of keypoints are not subject to significant scale and viewpoint changes and keypoints do not lie in arbitrary positions in the image, but on the aligned and horizontal epipolar lines.

6.8 Homography

A homography is a transformation of point correspondences (typically 2D-2D) and encodes **co-planarity** information. We include the expression for a point $\mathbf{P}$ on a 3D plane $\mathbf{n}^T\mathbf{P} + d = 0$ in the perspective projection equation from the left to the right camera frame:



$$\mathbf{p}_2 = \mathbf{K}(\mathbf{RP} + \mathbf{t}) = \mathbf{K}\left(\mathbf{RP} + \mathbf{t}\frac{\mathbf{n}^T\mathbf{P}}{-d}\right)$$
$$= \mathbf{K}\left(\mathbf{R} + \mathbf{t}\frac{\mathbf{n}^T}{-d}\right)\underbrace{\mathbf{K}^{-1}\mathbf{p}_1}_{=\mathbf{P}} = \mathbf{H}\mathbf{p}_1$$

where the homography matrix encodes the relative camera pose information up to scale with 8 DoF. Without loss of generality, we fix the last element to 1:

$$\begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}$$
$$\Rightarrow \quad u_2 = \frac{H_{11}u_1 + H_{12}v_1 + H_{13}}{H_{31}u_1 + H_{32}v_1 + 1}$$
$$v_2 = \frac{H_{21}u_1 + H_{22}v_1 + H_{23}}{H_{31}u_1 + H_{32}v_1 + 1}$$

This equation provides two constraints, such that we need **at least 4 co-planar point correspondences** to determine a homography matrix.

A 2D-2D homography is *similar* to the essential matrix in that they both encode relative pose information and certain point correspondences can be fitted by either matrix. It is *different* in that the essential matrix is derived from at least 5 arbitrary point correspondences, whereas the homography is derived from at least 4 co-planar point correspondences.

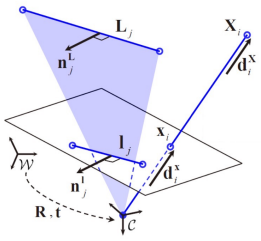
7 3D-2D Geometry

In 2D-2D Geometry we recover **relative** poses between the respective left and right camera frames and treat the first camera frame as the world frame in VO/SLAM/SFM, since the estimated translation is up to scale. In comparison, 3D-2D correspondences are used to estimate **absolute** camera poses.

7.1 Perspective-n-Points (PnP)

We assume that the camera intrinsics are known and use a set of $n \geq 3$ 3D-2D point correspondences to determnine the 6 DoF absolute pose of the camera w.r.t. a world frame.

Algebraic Illustration We use the facts that the ray vectors of the image-space and camera-space points are parallel and that a 3×3 skew-symmetric matrix has rank 2, such that each 3D-2D correspondence provides 2 constraints:



$$\mathbf{d}_i^{\mathbf{x}} \equiv \mathbf{d}_i^{\mathbf{X}} \quad \Rightarrow \quad \mathbf{K}^{-1}\mathbf{x}_i \equiv [\mathbf{R}\ \mathbf{t}]\mathbf{X}_i$$

Thus, we need **at least 3 correspondences** to determine the absolute pose.

3D-2D Correspondence Establishment There exist multiple methods:

- We can reconstruct 3D points via triangulation from known 2D-2D correspondences in the first two camera images and then assign the respective 2D feature descriptors to the 3D points. For subsequent camera images, we extract pixel-space 2D descriptors and compare them to those of the 3D points. This procedure is expensive.
- If the new camera pose is known, we can project known 3D locations onto a camera's image plane and search for 2D features in small regions around those projections.

7.2 Classical Algorithms

Direct Linear Transform (DLT) This general-purpose algorithm is used to estimate the elements of the transformation matrix as the solution-vector to a system of linear equations. For a normalized point, we have:

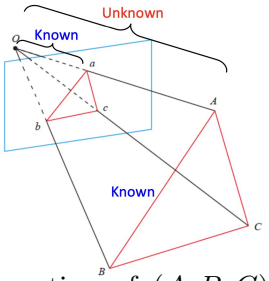
$$\lambda \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} m_1 & m_2 & m_3 & m_4 \\ m_5 & m_6 & m_7 & m_8 \\ m_9 & m_{10} & m_{11} & m_{12} \end{pmatrix}}_{=\begin{pmatrix} - & \mathbf{m}_1 & - \\ - & \mathbf{m}_2 & - \\ - & \mathbf{m}_3 & - \end{pmatrix} = [\mathbf{R}|\mathbf{t}]} \underbrace{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}_{=\mathbf{P}}$$

$$\Rightarrow \quad u_1 = \frac{m_1 X + m_2 Y + m_3 Z + m_4}{m_9 X + m_{10} Y + m_{11} Y + m_{12}} \quad \Rightarrow \quad \mathbf{m}_1^T\mathbf{P} - \mathbf{m}_3^T\mathbf{P}u_1 = 0$$
$$v_1 = \frac{m_5 X + m_6 Y + m_7 Z + m_8}{m_9 X + m_{10} Y + m_{11} Y + m_{12}} \quad \Rightarrow \quad \mathbf{m}_2^T\mathbf{P} - \mathbf{m}_3^T\mathbf{P}v_1 = 0$$

which gives us a linear system with 12 unknowns and 2 constraints per point, meaning that we need **at least 6 points**:

$$\begin{pmatrix} \mathbf{P}_1^T & 0 & -u_1\mathbf{P}_1^T \\ 0 & \mathbf{P}_1^T & -v_1\mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_N^T & 0 & -u_1\mathbf{P}_N^T \\ 0 & \mathbf{P}_N^T & -v_1\mathbf{P}_N^T \end{pmatrix} \begin{pmatrix} \mathbf{m}_1 \\ \mathbf{m}_2 \\ \mathbf{m}_3 \end{pmatrix} = \mathbf{0}$$

Perspective-3-Points (P3P) With **3 known correspondences**, we can construct three pairs of triangles: $\Delta Oab \rightarrow \Delta OAB$, $\Delta Obc \rightarrow \Delta OBC$ and $\Delta Oac \rightarrow \Delta OAC$. We use the known distances between points in image-space and world-space respectively to estimate the unknown transformation of (A, B, C) into the camera frame. The law of cosines gives us:



$$\overline{OA}^2 + \overline{OB}^2 - 2\overline{OA}\cdot\overline{OB}\cos\langle a,b\rangle = \overline{AB}^2$$
$$\overline{OB}^2 + \overline{OC}^2 - 2\overline{OB}\cdot\overline{OC}\cos\langle a,c\rangle = \overline{BC}^2$$
$$\overline{OA}^2 + \overline{OC}^2 - 2\overline{OA}\cdot\overline{OC}\cos\langle a,c\rangle = \overline{AC}^2$$

$$x^2 + y^2 - 2xy\cos\langle a,b\rangle = \frac{\overline{AB}^2}{\overline{OC}^2} \qquad (*)$$

$x = \overline{OA}/\overline{OC}$, $y = \overline{OB}/\overline{OC}$, $v = \overline{AB}^2/\overline{OC}^2$, $u = \overline{BC}^2/\overline{AB}^2$, $w = \overline{AC}^2/\overline{AB}^2$, $\Rightarrow$

$$y^2 + 1^2 - 2y\cos\langle a,b\rangle = \frac{\overline{BC}^2}{\overline{OC}^2} = uv$$
$$x^2 + 1^2 - 2x\cos\langle a,b\rangle = \frac{\overline{AC}^2}{\overline{OC}^2} = wv$$

which is a system with three unknowns and three constraints. Substituting the first equation into the second and third to eliminate v gives a multivariate polynomial system that can be solved for x and y:

$$(1-u)y^2 - ux^2 - 2\cos\langle b,c\rangle y + 2uxy\cos\langle a,b\rangle + 1 = 0$$
$$(1-w)x^2 - wy^2 - 2\cos\langle a,c\rangle x + 2wxy\cos\langle a,b\rangle + 1 = 0$$

Then, compute $\overline{OC}$ from equation $(*)$ and obtain $(\overline{OA}, \overline{OB})$ from the definitions of (x, y). Finally, we compute world-to-camera transformation from these 3D-3D point pairs.

7.3 Advanced Algorithms

Efficient Perspective-n-Points (EPnP) This algorithm requires more point correspondences than P3P, but is more accurate and efficient than DLT given the same set of correspondences. We follow three steps:

1. **Find 4 control points $\mathbf{c}_j$ in the camera and world frames** ($\mathbf{c}_j^c = [\mathbf{R}|\mathbf{t}]\begin{pmatrix}\mathbf{c}_j^w\\1\end{pmatrix}$). Ordinary points in the camera frame can be expressed via a linear combination of the control points:

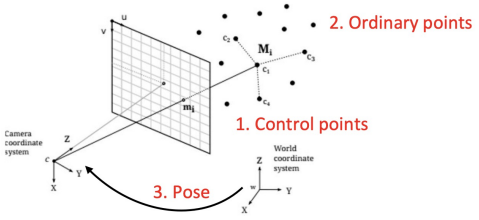
$$\mathbf{p}_i^c = [\mathbf{R}|\mathbf{t}]\begin{pmatrix}\mathbf{p}_j^w\\1\end{pmatrix} = [\mathbf{R}|\mathbf{t}]\begin{pmatrix}\sum_{j=1}^4 \alpha_{ij}\mathbf{c}_j^w\\\sum_{j=1}^4 \alpha_{ij}\end{pmatrix} = \sum_{j=1}^4 \alpha_{ij}\mathbf{c}_j^c$$

The perspective projection relates the known 3D-2D correspondences to the unknown control points:

$$\omega_i\begin{pmatrix}\mathbf{u}_i\\1\end{pmatrix} = \mathbf{K}\mathbf{p}_i^c = \mathbf{K}\sum_{j=1}^4 \alpha_{ij}\mathbf{c}_j^c = \begin{pmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\0 & 0 & 1\end{pmatrix}\sum_{j=1}^4 \alpha_{ij}\begin{pmatrix}x_j^c\end{pmatrix}$$

$$\Rightarrow \quad \sum_{j=1}^4 \left(\alpha_{ij} f_x x_i^c + \alpha_{ij}(c_x - u_i) z_j^c\right) = 0$$
$$\sum_{j=1}^4 \left(\alpha_{ij} f_y y_i^c + \alpha_{ij}(c_y - v_i) z_j^c\right) = 0$$

from which we construct a system of linear equations $\mathbf{M}\mathbf{x} = \mathbf{0}$ with two constraints per 3D-2D correspondence, i.e. $\mathbf{M} \in \mathbb{R}^{2n\times 12}$ and $\mathbf{x} \in \mathbb{R}^{12}$. We need **at least 6 correspondences** to solve for the 12 unknowns of the control points.



2. **Obtain the coordinates of each ordinary point in the camera and world frames** from the control points:

$$\mathbf{p}_i^w = \sum_{j=1}^4 \alpha_{ij}\mathbf{c}_j^w \qquad \text{with} \sum_{j=1}^4 \alpha_{ij} = 1$$

where the coefficients α_{ij} are called *homogeneous barycentric coordinates*.

3. Finally, **use 3D-3D correspondences to compute the closed-form solution of rotation and translation**.

Iterative Method for PnP Alternatively, we formulate the PnP problem as a non-linear least-squares problem by minimizing the re-projection errors:

$$\mathbf{T}* = \text{argmin}_{\mathbf{T}} \frac{1}{2}\sum_{i=1}^n \left\| \mathbf{u}_i - \frac{1}{s_i}\mathbf{K}\mathbf{T}\mathbf{P}_i \right\|^2$$

where the extrinsics matrix $\mathbf{T}$ needs to be initialized using a solution from DLT/EPnP.

7.4 Perspective-n-Lines (PnL)

["Details will not be asked in the exam"] We use 3D-2D line correspondences to estimate all 6 DoF of rotation and translation between the world and camera frame. Lines are described in Plücker coordinates $(\mathbf{n}, \mathbf{v})$ and relate between world and camera frame via:

$$\begin{pmatrix}\mathbf{n}^c\\\mathbf{v}^c\end{pmatrix} = \begin{pmatrix}\mathbf{R} & [\mathbf{t}]_\times\mathbf{R}\\\mathbf{0} & \mathbf{R}\end{pmatrix}\begin{pmatrix}\mathbf{n}^w\\\mathbf{v}^w\end{pmatrix}$$

where our goal is to estimate the 18 unknown parameters of the transformation matrix. We know the 2D-3D corresponding points and use the line projection equation for constraints:

$$\mathbf{l} = \underbrace{\begin{pmatrix}f_y & 0 & 0\\0 & f_x & 0\\-f_y x_0 & -f_x y_0 & f_x f_y\end{pmatrix}}_{=\mathcal{K}}\mathbf{n}^c$$

where $\mathbf{l}$ is the 2D image line. Each line correspondence can provide 2 linear constraints, such that we need at least 9 correspondences to solve the DLT problem.

8 3D-3D Geometry

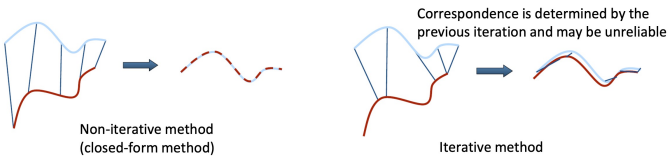
We want to estimate transformations between 3D point clouds with known or unknown 3D-3D correspondences, which is a.k.a. *point cloud registration*. If the correct correspondences are known, the rotation and translation can be calculated in closed form using a *non-iterative method*. Otherwise, we performs *iterations*. Given **at least 3 3D-3D correspondences** we find the transformation from SE(3) that solves this linear system of equations between two coordinate frames:

$$\begin{pmatrix} X_{k-1}^i \\ Y_{k-1}^i \\ Z_{k-1}^i \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} X_k^i \\ Y_k^i \\ Z_k^i \\ 1 \end{pmatrix} \quad \text{for points } i = 1, 2, 3, \dots$$

Alternatively, we find the $\mathbf{R}$ and $\mathbf{t}$ that minimize the sum of squared errors between two point sets $X = \{\mathbf{x}_1, \dots, \mathbf{x}_{N_x}\}$ and $P = \{\mathbf{p}_1, \dots, \mathbf{p}_{N_p}\}$:

$$E(\mathbf{R}, \mathbf{t}) = \frac{1}{N_p} \sum_{i=1}^{N_p} \|\mathbf{x}_i - \mathbf{R}\mathbf{p}_i - \mathbf{t}\|^2$$

where we optionally simultaneously optimize for $\mathbf{p_i}$, if the (correct) correspondences are not known. For point clouds at different scales, we estimate a transformation from Sim(3).



The following methods optimize transformations from SE(3). Horn's method and Umeyama's method work with Sim(3).

8.1 Non-iterative Method

Given two point sets and assuming $N_x = N_p$, we retrieve normalized point sets first:

$$X' = \{\mathbf{x}_i - \mu_x\} = \{\mathbf{x}'_i\} \qquad \text{with } \mu_x = \frac{1}{N_x} \sum_{i=1}^{N_x} \mathbf{x}_i$$
$$P' = \{\mathbf{p}_i - \mu_p\} = \{\mathbf{p}'_i\}$$

Then, we find the optimal solution for the transformation: ["Derivation irrelevant for the exam"]

$$\begin{aligned} E(\mathbf{R}, \mathbf{t}) &= \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{R}\mathbf{p}_i - \mathbf{t}\|^2 \\ &= \sum_{i=1}^{n} \|\mathbf{x}'_i - \mathbf{R}\mathbf{p}'_i - \mathbf{t} - \mu_x + \mu_x - \mathbf{R}\mu_p + \mathbf{R}\mu_p\|^2 \\ &= \cdots = \sum_{i=1}^{n} \underbrace{\|\mathbf{x}_i - \mu_x - \mathbf{R}(\mathbf{p}_i - \mu_p)\|^2}_{\text{only w.r.t. } \mathbf{R}} + \underbrace{n \|\mathbf{x}_i - \mathbf{R}\mu_p - \mathbf{t}\|}_{\substack{\text{force this to 0} \\ \rightarrow \text{ Constraint for } \mathbf{t}}} \end{aligned}$$

$$\begin{aligned} \Rightarrow \mathbf{R}^* &= \operatorname*{argmin}_{\mathbf{R}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_x - \mathbf{R}(\mathbf{p}_i - \mu_p)\|^2 \\ &= \operatorname*{argmin}_{\mathbf{R}} \sum_{i=1}^{n} \|\mathbf{x}'_i - \mathbf{R}\mathbf{p}'_i\|^2 \\ &= \operatorname*{argmin}_{\mathbf{R}} \sum_{i=1}^{n} (\mathbf{x}_i'^T \mathbf{x}'_i + \mathbf{p}_i'^T \underbrace{\mathbf{R}^T\mathbf{R}}_{=\mathbf{I}} \mathbf{p}'_i - 2\mathbf{x}_i'^T \mathbf{R}\mathbf{p}'_i) \\ &= \operatorname*{argmin}_{\mathbf{R}} \sum_{i=1}^{n} (-2\mathbf{x}_i'^T \mathbf{R}\mathbf{p}'_i) = \operatorname*{argmax}_{\mathbf{R}} \sum_{i=1}^{n} (\mathbf{x}_i'^T \mathbf{R}\mathbf{p}'_i) \\ &= \operatorname*{argmax}_{\mathbf{R}} \operatorname{tr}\left(\mathbf{R} \sum_{i=1}^{n} (\mathbf{p}'_i \mathbf{x}_i'^T)\right) \end{aligned}$$

We conclude that performing SVD on $\mathbf{W} = \sum_{i=1}^{N_p} \mathbf{p}'_i \mathbf{x}_i'^T = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$ leaves us with

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \quad \text{and} \quad \mathbf{t} = \mu_x - \mathbf{R}\mu_p$$

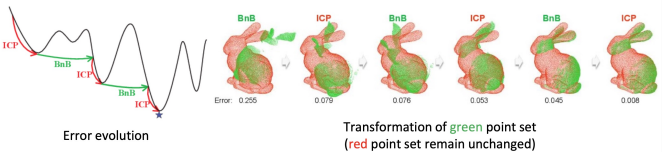
8.2 Iterative Closest Point (ICP)

This method converges if the corresponding points are "close enough" prior to the optimization. It partially solves the problem of determining the correct point correspondences.

Algorithm Given two point clouds with unknown 3D-3D correspondences:

1. Determine corresponding points based on the smallest distance.
2. Compute $\mathbf{R}$ and $\mathbf{t}$ via the non-iterative method.
3. Apply $[\mathbf{R}|\mathbf{t}]$ to the point cloud to be registered.
4. Compute the error $E(\mathbf{R}, \mathbf{t})$.
5. Repeat from 1. if the error exceeds a threshold.

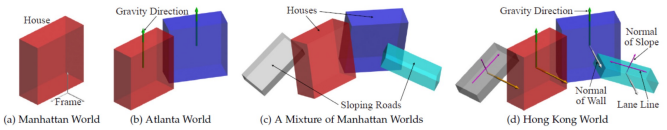
Variants ["Irrelevant for the exam"] In order to improve accuracy, correspondences may be weighted by their noise. We can also reject outliers point pairs to improved robustness.
Performing linear optimization steps of ICP can cause getting stuck in local minima, but we can use a global search method (e.g. branch-and-bound; BnB) to escape them. In practice, we combine ICP and BnB to improve the efficiency of pure BnB.



9 Single-view Geometry

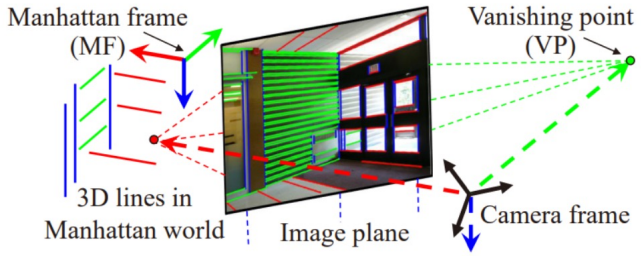
We use structural regularities in man-made environments (e.g. orthogonality and parallelism) to recover camera poses and 3D structure.

Man-made environments can be abstracted by various structural models. The **Manhattan world** has one horizontal and two vertical dominant directions (DDs). The **Atlanta world** has one vertical and multiple horizontal DDs. The **Hong Kong world** has one vertical, multiple horizontal and multiple sloping DDs.

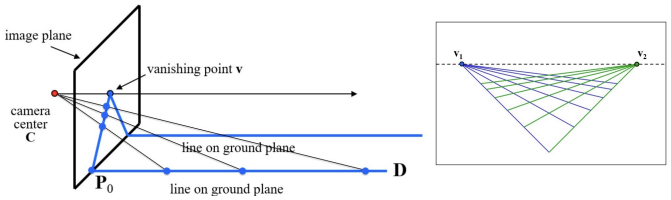


9.1 Vanishing Point Expression

Vanishing Direction 2D image-space lines projected from parallel 3D lines intersect at a "vanishing point" in the image plane (in or outside the image). A vanishing direction is parallel to a 3D **dominant direction** ("equivalent") and is the connection between a vanishing point and the optical center.



Vanishing Point Intuition When projecting 3D points on the same 3D line into image-space, the 3D point at infinite depth corresponds to the vanishing point of the resulting 2D image-space line. In other words, the vanishing point can be treated as the projection of a 3D point a infinity.



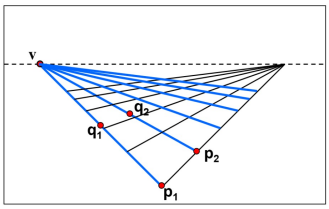
Mathematical Representation Any point on the 3D line can be expressed as:

$$\mathbf{P}_t = \begin{pmatrix} P_x+tD_x \\ P_y+tD_y \\ P_z+tD_z \\ 1 \end{pmatrix} \approx \begin{pmatrix} P_x/t+D_x \\ P_y/t+D_y \\ P_z/t+D_z \\ 1/t \end{pmatrix} \overset{\text{if } t\to\infty}{\Rightarrow} \mathbf{P}_\infty = \begin{pmatrix} D_x \\ D_y \\ D_z \\ 0 \end{pmatrix}$$

where the division by t works, because homogeneous coordinates are equivalent up to a scale. The point $\mathbf{P}_\infty$ only depends on the line direction (dominant direction) $\mathbf{D}$. Accordingly, the following relation holds up to a scale:

$$\mathbf{v} = \mathbf{K}\mathbf{P}_\infty = \mathbf{K}\mathbf{D} \quad \text{and} \quad \underbrace{\mathbf{K}^{-1}\mathbf{v}}_{\substack{\text{vanishing} \\ \text{dir.}}} = \mathbf{P}_\infty = \underbrace{\mathbf{D}}_{\substack{\text{dominant} \\ \text{dir.}}}$$

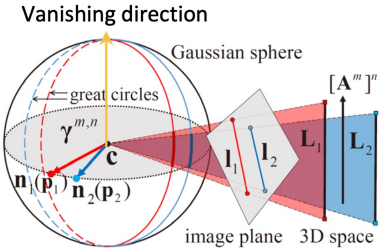
Alternative Mathematical Representation With knowledge of two point pairs $\mathbf{p}_1, \mathbf{q}_1$ and $\mathbf{p}_2, \mathbf{q}_2$ on two lines that intersect in the vanishing point, we compute the intersection as:



$$\mathbf{v} = \underbrace{(\mathbf{p}_1 \times \mathbf{q}_1)}_{\substack{\text{Homogen. coord.} \\ \text{of a 2D line}}} \times (\mathbf{p}_2 \times \mathbf{q}_2)$$

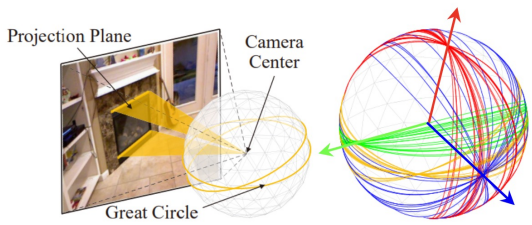
We reduce the effect of noise by using more than two lines and computing the least-squares solution for the intersection point. If the **2D lines corresponding to the same vanishing point are almost parallel, the vanishing point may be far outside of the image**. Thus, noise in the image-space points can cause large errors in the calculation of vanishing points.

Representation on a Sphere Instead of computing vanishing points in the *unbounded image-plane*, we map image lines into great circles on a sphere. There are three steps:



1. we construct a sphere around the camera center and have two 3D lines from the same dominant direction,
2. the intersection of the projection plane for each line with the sphere is a great circle,
3. the vector between the camera center and the intersection point of the great circles is the vanishing direction.

Thus, instead of estimating the vanishing point, we estimate the vanishing direction (which is analogous) on a unit sphere. The calculation in this *bounded space of the unit sphere* is more robust to noise. Moreover, the spherical representation can *encode the orthogonality of 3 vanishing directions.*



9.2 Vanishing Point Estimation

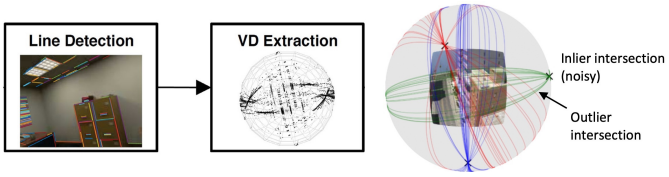
We define a **Manhattan frame** $\mathcal{M}$ that has its axes aligned to the dominant directions. We define the rotation $\mathbf{M}$ from the Manhattan to the camera frame as:

$$\mathbf{M} \cdot [\mathbf{d}_1^{\mathcal{M}}\ \mathbf{d}_2^{\mathcal{M}}\ \mathbf{d}_3^{\mathcal{M}}] = [\mathbf{d}_1\ \mathbf{d}_2\ \mathbf{d}_3] = \mathbf{M}$$

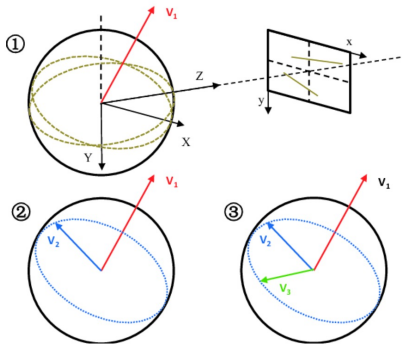
$$\text{with } \mathbf{d}_1^{\mathcal{M}} = \begin{pmatrix}1\\0\\0\end{pmatrix} \quad \mathbf{d}_2^{\mathcal{M}} = \begin{pmatrix}0\\1\\0\end{pmatrix} \quad \mathbf{d}_3^{\mathcal{M}} = \begin{pmatrix}0\\0\\1\end{pmatrix}$$

9.2.1 Census-based Methods

These old-fashioned methods first detect image-space lines and then construct their corresponding great circles. The vanishing points are areas on the unit sphere with a high density of noisy intersections of great circles. This method is *sensitive to outliers*.



9.2.2 Sampling-based Method



First, we randomly sample two image-space lines associated to the same vanishing point, compute their great circles and extract the first dominant direction. We sample a third image-space space line associated to another vanishing point. The second dominant direction 1) lies within this third circle and 2) is orthogonal to the first dominant direction. The third dominant direction is orthogonal to the first and second directions. Because we cannot guarantee that this composition of the three random samples, we resample multiple times and filter out the outlier dominant directions. This method is *efficient*.

9.2.3 Search-based Method

This is a more *accurate* method using brand-and-bound. ["Not covered in the class."]

9.3 Applications

Single-view 3D Reconstruction We combine vanishing points and 3D box priors to reconstruct a 3D structure.

Camera Pose Optimization 3D dominant directions can be used to improve visual SLAM, since the vanishing points encode rotation information.

Camera Calibration Zhang's and Tsai's method require knowledge of the exact 3D geometry and 3D-2D correspondences, which is a limitation in practice. We assume that the principal point is centered in the image and that the pixels are square, such that we only estimate the focal length. The new projection matrix is:

$$\begin{pmatrix}f&0&0\\0&f&0\\0&0&0\end{pmatrix}[\mathbf{R}|\mathbf{t}] = \begin{pmatrix}r_{11}&r_{12}&r_{13}&t_1\\r_{21}&r_{22}&r_{23}&t_2\\r_{31}/f&r_{32}/f&r_{33}/f&t_3/f\end{pmatrix}$$

1. Estimate the vanishing points in 2D.
2. Define 3D points in the Manhattan frame at the origin $(0\ 0\ 0\ 1)^T$ and at infinity along the X, Y and Z axes, i.e. $(1\ 0\ 0\ 0)^T$, etc (0 homogeneous coordinate means infinity).
3. Estimate the projection matrix from the **Manhattan** frame to the camera image (up-to-scale):

$$w\begin{pmatrix}x\\y\\1\end{pmatrix} = \begin{pmatrix}|&|&|&|\\\pi_1&\pi_2&\pi_3&\pi_4\\|&|&|&|\end{pmatrix}\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix} = \mathbf{\Pi X}$$

   where $\mathbf{X}$ is a 3D point in the Manhattan frame. We use the 3D-2D correspondences established in steps 1 and 2 to compute the columns of $\mathbf{\Pi}$ individually:

   $\pi_1 = \mathbf{\Pi}(1\ 0\ 0\ 0)^T = \mathbf{v}_x =$ Known vanishing point

   Similarly, we have $\pi_2 = \mathbf{v}_y$ and $\pi_3 = \mathbf{v}_y$.

   $\pi_4 = \mathbf{\Pi}(0\ 0\ 0\ 1)^T =$ Measured origin projection

   We conclude, $\mathbf{\Pi} = [\mathbf{v}_x\ \mathbf{v}_y\ \mathbf{v}_z\ \mathbf{o}]$.

4. We account for each point-projection's scale with:

$$\mathbf{\Pi} = [a\mathbf{v}_x\ b\mathbf{v}_y\ c\mathbf{v}_z\ d\mathbf{o}] = \begin{pmatrix}r_{11}&r_{12}&r_{13}&t_1\\r_{21}&r_{22}&r_{23}&t_2\\r_{31}/f&r_{32}/f&r_{33}/f&t_3/f\end{pmatrix}$$

   which is equivalent to:

$$\begin{pmatrix}v_{x1}&v_{y1}&v_{z1}&o_1\\v_{x2}&v_{y2}&v_{z2}&o_2\\fv_{x3}&fv_{y3}&fv_{z3}&fo_3\end{pmatrix} = \begin{pmatrix}r_{11}/a&r_{12}/b&r_{13}/c&t_1/d\\r_{21}/a&r_{22}/b&r_{23}/c&t_2/d\\r_{31}/a&r_{32}/b&r_{33}/c&t_3/d\end{pmatrix}$$

   Because of the orthogonality constraint of the rotation matrox, the first two columns must be orthogonal, such that:

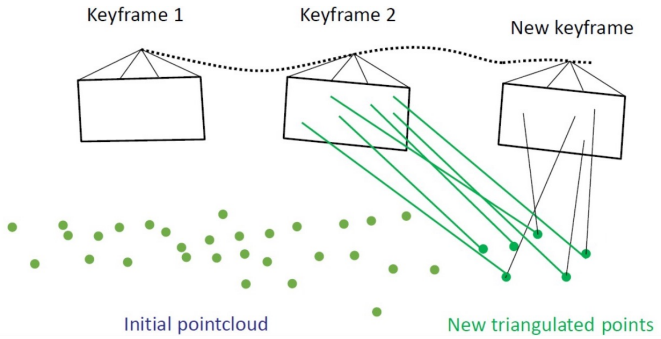
$$v_{x1}v_{y1} + v_{x2}v_{y2} + f^2 v_{x3}v_{y3} = 0 \quad \Rightarrow \quad f = \sqrt{\frac{v_{x1}v_{y1} + v_{x2}v_{y2}}{-v_{x3}v_{y3}}}$$

10 Combination of Different Configurations

10.1 2D-2D and 3D-2D (Monocular Camera)

Herein, we use 2D-2D and 3D-2D correspondences for localizing camera and triangulation for reconstructing the 3D point cloud (mapping). This is the typical pipeline for **visual odometry (VO)/ sequential structure from motion (SfM)**. In detail:

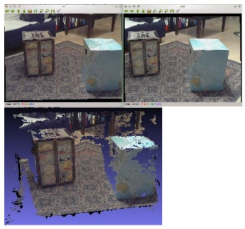
1. Given the 1st and 2nd frame, we **initialize the relative pose using 2D-2D epipolar geometry** (5-point or 8-point method). Because the result is up-to-scale, we normalize the translation vector. We produce an "initial point cloud" through triangulation, where its scale is determined by the norm of the translation vector. We treat the **first camera frame as the world frame**.
2. Given a new view, we establish 3D-2D correspondences between the reconstructed 3D point cloud and the new frame and estimate the *absolute pose* of the new view (e.g. with DLT/P3P/EPnP). Herein, the **scale of the extrinsic parameters is aligned to the pre-defined scale** of the previously reconstructed point cloud.
3. We refine our estimate via bundle adjustment and use the new estimated absolute pose to triangulate *new* 3D points in the world frame.
4. Repeat from (2.) for subsequent frames.



10.2 2D-2D and 3D-3D (Stereo Camera)

We use the stereo setup to retrieve dense 3D reconstructions at every time step and align them via 3D-3D correspondences. This pipeline is often referred to as **RGBD-SLAM/ Stereo-SLAM/ Laser-SLAM**.

1. For a given stereo image pair, we reconstruct 3D points in the left camera (*local map*), using depth from disparity. Again, we treat the first camera as the world frame and its 3D points as the *global map*.
2. At the next time step, we generate another local map in the current left camera frame and estimate the relative rotation and translation w.r.t. the world frame by aligning two point sets with ICP. We transform the current local map to the world frame to increment the global map.



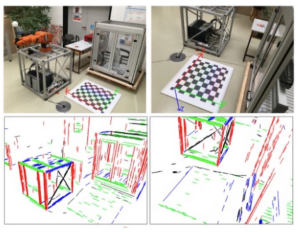
A local 3D map reconstructed by an image pair

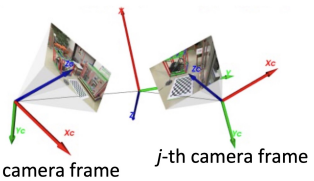
10.3 2D-2D and Single-view (Monocular Camera)

This pipeline is an **addition to the 2D-2D and 3D-2D pipeline** in man-made environments for improved accuracy. We use the fact that the same dominant directions can be observed in two images without overlap ("global constraint"). We use this constraint to align local dominant directions (in the current camera frame) to global dominant directions (i.e. in the first camera frame) using a rotation:

$$\mathbf{g} \propto \mathbf{R}_i \mathbf{d}_i$$

where $\mathbf{g}$ is a global dominant direction and $\mathbf{d}_i$ is a dominant direction of the ith camera. With known dominant directions in all camera frames and the world frame, we find the rotations $\mathbf{R}_i$ that align them best.





10.4 Full Visual SLAM Pipeline

1. **Initial guess**: we can initialize each new pose with the previous camera's pose transformed by relative transformation between the preceeding two cameras (assuming constant velocity and/or acceleration).
2. Optimize the pose by **minimizing the reprojection error** of known keypoints.
3. Optimize the pose further by **aligning the dominant directions** (in structured environments only).

11 Photometric Error and Direct SLAM

Indirect (feature-based) SLAM abstracts the images as a set of keypoints, matches them and uses the epipolar constraints plus bundle adjustment to compute poses. It has the advantage that features are relatively robust to viewpoint change and illumination variation, but at the cost of only discarding image information without corners or high brightness. Plus, it only reconstructs a sparse 3D map, needs time-consuming outlier removal and noise propagation from the feature-detection and matching to the pose optimization step is problematic for overall optimality.

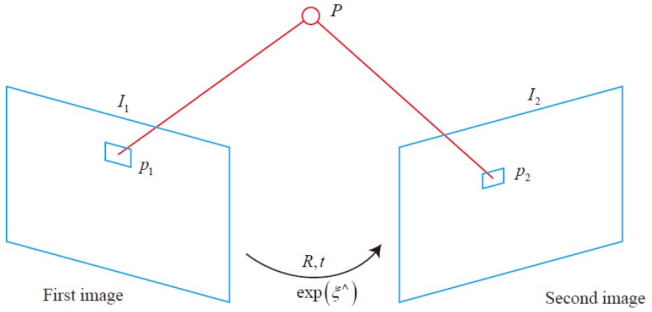
Direct SLAM These methods determine feature correspondences and camera motion simultaneously. Compared to indirect methods, they produce a dense 3D map and use more image information.

Photometric Error The photometric error is the intensity difference between a pixel pair in two images:

$$e = I_1(\mathbf{p}_1) - I_2(\mathbf{p}_2)$$

where $\mathbf{p}_1, \mathbf{p}_2$ are the pixel coordinates. It relies on the *brightness consistency assumption*, that two corresponding pixels have the same brightness. We can take N pixels in the left image into account by minimizing the least-squares error:

$$\min \sum_{i=1}^{N} \mathbf{e}_i^T \mathbf{e}_i \qquad \text{where } \mathbf{e}_i = \mathbf{I}(\mathbf{p}_1) - \mathbf{I}(\mathbf{p}_2)$$



11.1 Direct Method

Problem Formulation We aim to optimize the translation between two cameras by minimizing the photometric error. Therein, $\mathbf{p}_1$ is known and $\mathbf{p}_2$ is treated as a function of the unknown parameters $\mathbf{R}$, $\mathbf{t}$ and Z_1 via the perspective projection equation:

$$\mathbf{K}^{-1}\mathbf{p}_1 = \mathbf{K}^{-1}\begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \frac{1}{Z_1}\mathbf{P} \qquad \mathbf{p_2} = \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = \mathbf{K}(\mathbf{R}\mathbf{P} + \mathbf{t})$$

Thus, we find the transformation from the $(k-1)^{\text{th}}$ to the k^{th} camera as well as the backprojected 3D points by minimizing the error over N known pixels:

$$\mathbf{P}^i, \mathbf{R}, \mathbf{t} = \underset{\mathbf{P}^i, \mathbf{R}, \mathbf{t}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(I_{k-1}(\mathbf{p}_{k-1}^i) - I_k(\pi(\mathbf{P}^i, \mathbf{K}, \mathbf{R}, \mathbf{t})) \right)$$

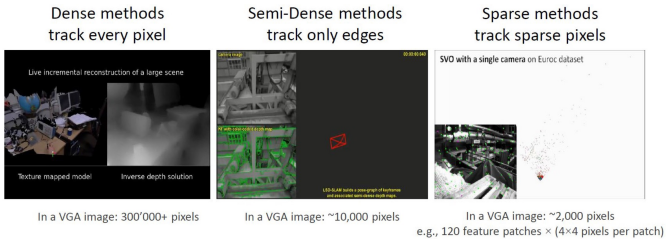
leaving a total of $N + 2$ unkowns (N depth values, 1 rotation and 1 translation).

Unkown Depth In practice, we obtain an initial guess for the depth values via RGB-D or stereo camera. Without this initial guess, the optimization likely gets stuck in local optima. Besides, we improve the numerical stability by using the inverse of depth:

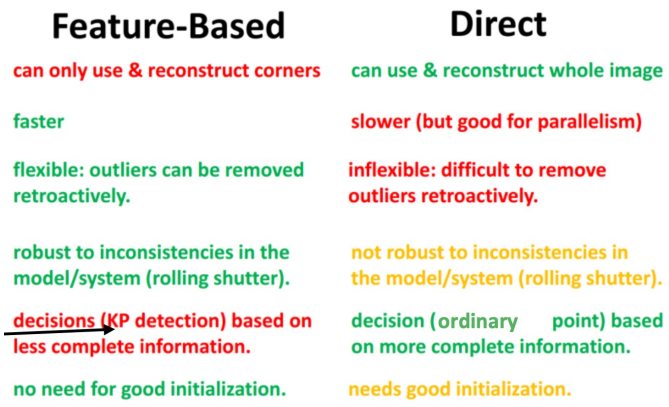
$$\rho = \frac{1}{d} = \frac{1}{\|\mathbf{p} - \mathbf{c}_0\|} \in (0, 1]$$

where $\mathbf{p}$ is the 3D point and $\mathbf{c}_0$ is the camera center.

Pixel Selection Although **dense tracking** of all pixels is possible in theory, points with non-obvious pixel gradients will not contribute much to motion estimation and real-time calculation is infeasible. The **semi-dense direct method** computes the Jacobian of each pixels and only tracks pixels with significant gradients (i.e. large values in the Jacobian). This method tracks only edges and reconstructs a semi-dense structure. We can also track parse key-points (**sparse direct method**), which uses e.g. Harris' corner detector and tracks a few hundred of those key-points, without needing to compute feature descriptors. This method is the fastest, but creates a sparse reconstruction.



Pros and Cons Using all pixels to estimate the transformation makes the solution more accurate and robust to motion blur and weak textures. Increasing the frame rate reduces the computational cost per frame and no RANSAC is needed. On the other hand, the method is sensitive to initialization and works with limited motion between frames. Large baselines often violate the brightness consistency assumption, such that direct SLAM is not suitable.



11.1.1 Direct SLAM Methods

These methods generally follow the same workflow as PTAM (popular indirect method): they are keyframe based, and alternate between localization and mapping as independent threads. Most work real-time at 30Hz, with global optimization happening asynchronously every once in a while. The following methods support at least monocular or stereo cameras.

11.1.2 LSD-SLAM

Large scale semi-dense SLAM minimizes the photometric error *directly* with a semi-dense 3D map. Poses and 3D map are optimized separately.

11.1.3 DSO

Direct sparse odometry minimizes the photometric error and represents the 3D structure as a sparse large gradients' depth map. Poses and structure are optimized jointly and additionally corrects the photometric error by the difference in exposure time of the two frames.

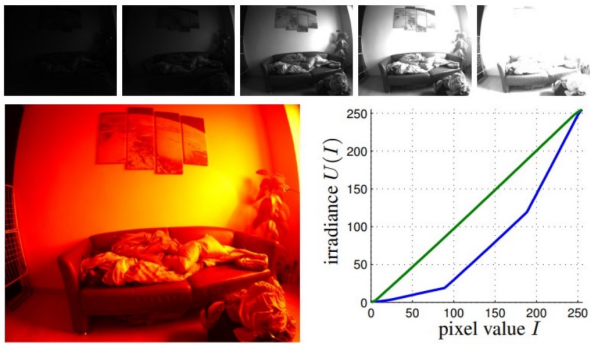
11.1.4 SVO

Semi-direct visual odometry combines both domains, as it uses direct methods for frame-to-frame motion estimation and indirect methods for frame-to-keyframe pose refinement. It uses a probabilistic model for depth estimation and is faster than real-time.

11.2 Photometric Calibration

In practice, the brightness consistency assumption is affected by different factors. We improve the accuracy of the photometric error, by modifying brightness in three ways:

- **Response function**: we leverage irradiance ("light energy per time") consistency, by mapping energy to the pixel intensity. When calibrating an image, we map its pixel values to their irradiances according to the reponse function.



- **Exposure time**: assuming constant energy per time, we modify brightness according to exposure time.

- **Vignetting** is a reduction of image brightness towards the image periphery caused by manufacturing defects.

12 Bundle Adjustment & Nonlinear Optimization

Previous linear approaches are elegant because the compute optimal solutions in closed form. For noisy point locations, they often provide sub-optimal performance or even fail. A Bayesian formulation determines the most likely camera transformation and 2D coordinates $\boldsymbol{x}$ by performing a maximum á posteriori estimate. This approach however involves modeling probability densities on a fairly complicated space. Under the assumption that the observed 2D point coordinates $\tilde{\boldsymbol{x}}$ are corrupted by zero-mean Gaussian noise, maximum likelihood estimation leads to bundle adjustment

The name "bundle adjustment" is derived from the attempt of adjusting the bundles of light rays emitted from the 3D points.

Solving the non-linear optimization problems requires computing the derivatives w.r.t. SO(3) and SE(3) matrices., which may nclude the summation/substraction. If two matrices are from SO(3) or SE(3), their sum is not guaranteed to be in the same space. For this reason we perform the optimization in the corresponding Lie algebra and map the result back to the Lie group ("no further details discussed").

12.1 Error Metrics

The quality of the estimated camera pose can be measured with different metrics.

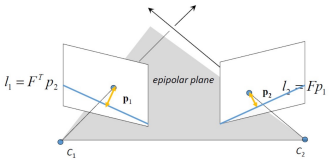
Algebraic Error This error measures how "ill-defined" the epipolar geometry is given the estimated pose. E.g. the 8-point algorithm seeks to minimize the algebraic error given normalized image points:

$$err = \|\mathbf{QE}\|^2 = \sum_{i=1}^{N} (\bar{\mathbf{p}}_2^{i\,T} \mathbf{E} \bar{\mathbf{p}}_1^{\mathbf{i}})^2$$

$$\text{with } \left\|\bar{\mathbf{p}}_2^T \mathbf{E} \bar{\mathbf{p}}_1\right\| = \left\|\bar{\mathbf{p}}_2^T \cdot (\mathbf{E}\bar{\mathbf{p}}_1)\right\| = \left\|\bar{\mathbf{p}}_2^T\right\| \|\mathbf{E}\bar{\mathbf{p}}_1\| \cos(\theta)$$
$$= \|\bar{\mathbf{p}}_2\| \underbrace{\|[\mathbf{t}]_\times \mathbf{R}\bar{\mathbf{p}}_1\|}_{\substack{\text{epipolar plane normal} \\ \text{in the right frame}}} \cos(\theta)$$

This error is only zero if the normalized image points $\bar{\mathbf{p}}_1, \bar{\mathbf{p}}_2$ and the translation vector $\mathbf{t}$ are coplanar.

Epipolar Line Distance Given the essential matrix and a point correspondence, this metric measures the distance between a point and its epipolar line. The fundamental matrix formulation for the epipolar constraint describes the distance between a point and a line:

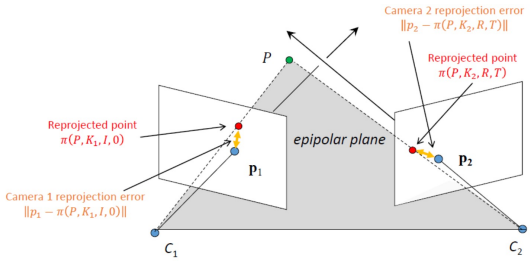


$$d = \underbrace{\begin{pmatrix} u_2^i \\ v_2^i \\ 1 \end{pmatrix}^T}_{\text{right point}} \underbrace{\mathbf{F} \begin{pmatrix} u_1^i \\ v_1^i \\ 1 \end{pmatrix}}_{\substack{\text{right} \\ \text{epipolar line}}} \quad \text{with } \mathbf{F} = \mathbf{K}_2^{-1}\mathbf{E}\mathbf{K}_1^{-1}$$

This distance $\text{dot}(\mathbf{p}, \mathbf{l})$ is zero, if the point lies on the line. This metric is efficient compared to the reprojection error, because it avoids triangulation but requires known 2D-2D geometry.

Reprojection Error We triangulate the 3D points and compute the sum of squared errors between the corresponding 2D points and the reprojections. It is more accurate, because is measures the error directly w.r.t. the input points.

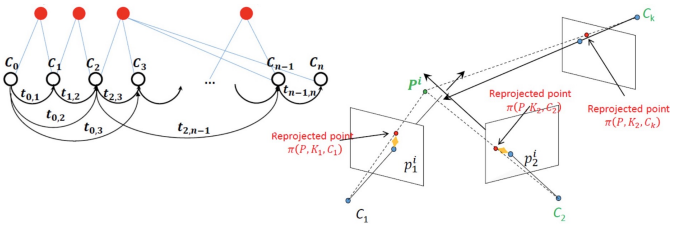
$$err = \sum_{i=1}^{N} \left\|\mathbf{p}_1^i - \pi(\mathbf{P}^i, \mathbf{K}_1, \mathbf{I}, \mathbf{0})\right\|^2 + \left\|\mathbf{p}_2^i - \pi(\mathbf{P}^i, \mathbf{K}_2, \mathbf{R}, \mathbf{t})\right\|^2$$



We use the reprojection error for pose and 3D optimization and evaluation of calibration accuracy.

12.2 Bundle Adjustment

Bundle Adjustment is the extension of two-view reprojection minimization to the multi-view case. We reformulate the problem as a *graph optimization problem* where each node is a unknown camera parameter and the edges are the constraints. The first camera is treated as the world frame.



We jointly optimize all n camera poses and N 3D points:

$$\mathbf{P}^i, \mathbf{C}_1, \ldots, \mathbf{C}_n = \underset{\mathbf{X}^i, \mathbf{C}_1, \ldots, \mathbf{C}_n}{\text{argmin}} \sum_{k=1}^{n} \sum_{i=1}^{N} \rho\left(\mathbf{p}_k^i - \pi(\mathbf{P}^i, \mathbf{K}_k, \mathbf{C}_k)\right)$$

where ρ is a M-estimator (e.g. $\text{L}_2, \text{L}_1, \text{Huber}$). The same optimization problem can be parametrized dif-

ferently. We introduce $\boldsymbol{x}_i^j$ to denote the true 2D coordinate associated with the measured coordinate $\tilde{\boldsymbol{x}}_i^j$:

$$E(\{\boldsymbol{x}_1^j, \lambda_1^j\}_{j=1\dots N}, R, T) = \\ = \sum_{j=1}^{N} ||\boldsymbol{x}_1^j - \tilde{\boldsymbol{x}}_1^j||^2 + ||\tilde{\boldsymbol{x}}_2^j - \pi(R\lambda_1^j \boldsymbol{x}_1^j + T)||^2$$

Alternatively, a constrained optimization by minimizing the cost function:

$$E(\{\boldsymbol{x}_i^j\}_{j=1\dots N}, R, T) = \sum_{j=1}^{N}\sum_{i=1}^{2} ||\boldsymbol{x}_i^j - \tilde{\boldsymbol{x}}_i^j||^2$$

with constraints for $j = 1, \dots, N$:

$$\boldsymbol{x}_2^{jT} \hat{T} R \boldsymbol{x}_1^j = 0 \text{ (epipolar constraint)}$$
$$\boldsymbol{x}_1^{jT} e_3 = 1,\ \boldsymbol{x}_2^{jT} e_3 = 1 \text{ (lying on image plane)}$$

The highly non-convex cost function requires good initialization. Hence, it is used in the **last step in a reconstruction pipeline**. We perform bundle adjustment in real-time by with two **acceleration strategies**:

- We can use a **sliding window** that only optimizes a sub-map/sub-graph at a time. Finally, we fine-tune the "initial guess" with global optimization over all maps.
- **Motion only**: if the 3D map is known, we only need to optimize the camera parameters.

Photometric Bundle Adjustment We can extend the photometric error formulation to multiple views and optimize the poses of all cameras jointly. This strategy is very accurate, but depends on good initialization.

12.3 Nonlinear Optimization

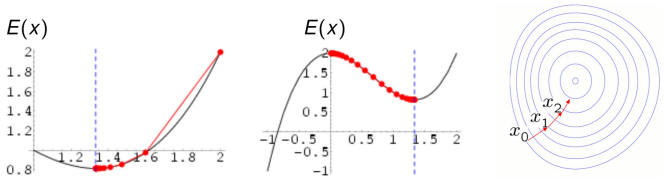
Nonlinear optimization refers to the problem of fitting a non-linear curve to a set of observed points. It consists of selecting a suitable function (the perspective projection equation in our case) and estimating the parameters by the least-squares method.

12.3.1 Gradient Descent ("Steepest Method")

Gradient descent is a first-order optimization method. It aims at computing a local minimum of a (generally) non-convex cost function by iteratively stepping in the direction in which the energy decreases most. This is given by the negative energy gradient:

$$x_{k+1} = x_k - \epsilon \frac{dE}{dx}(x_k) \quad k = 0, 1, 2, \dots$$

For the case of convex E, the found local minimum will also be the global minimum. The step size ϵ can be chosen differently in each iteration. Gradient descent is very popular, however not the fastest method in most cases.



12.4 Least Squares Estimation [old]

Linear (ordinary) least squares is a method for estimating a set of parameters in a linear regression model. Assume there is a linear relationship of the form

$$a_i = b_i^T x + \eta_i$$

with input $b_i \in \mathbb{R}^d$, output $a_i \in \mathbb{R}$, $i \in \{1, \dots, n\}$, an unknown vector $x \in \mathbb{R}^d$ and zero-mean Gaussian noise $\eta \sim \mathcal{N}(0, \boldsymbol{\Sigma})$, $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I_n}$. The maximum likelihood estimation of x leads to the ordinary least squares problem (convex):

$$\min_x \sum_i (a_i - x^T b_i)^2 = (a - \boldsymbol{B}x)^T (a - \boldsymbol{B}x).$$

For general $\boldsymbol{\Sigma}$, we get the generalized least squares problem:

$$\min_x (a - \boldsymbol{B}x)^T \boldsymbol{\Sigma}^{-1} (a - \boldsymbol{B}x).$$

This is a quadratic cost function with positive definite $\boldsymbol{\Sigma}^{-1}$. It has the closed-form solution:

$$\hat{x} = (\boldsymbol{B}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{B})^{-1} \boldsymbol{B}^T \boldsymbol{\Sigma}^{-1} a$$

With $\boldsymbol{\Sigma}$ being diagonal (no correlation between variables), we get weighted least squares:

$$\min_x \sum_i w_i (a_i - x^T b_i)^2, \quad \text{with } w_i = \sigma_i^{-2}.$$

For the case of unknown matrix $\boldsymbol{\Sigma}$, there exist iterative estimation algorithms such as feasible generalized least squares or iteratively reweighted least squares.

Iterative reweighted least squares aims at minimizing generally non-convex optimization problems of the form

$$\min_x \sum_i w_i(x) |a_i - f_i(x)|^2,$$

with some known weighting function $w_i(x)$. A solution is obtained by iterating the following problem:

$$x_{t+1} = \arg\min_x \sum_i w_i(x_t) |a_i - f_i(x)|^2.$$

For the case that fi is linear, i.e. $f_i(x) = x^T b_i$, each subproblem is simply a weighted least squares problem that can be solved in closed form. Nevertheless, this iterative approach will generally not converge to a global minimum of the original (nonconvex) problem.

Nonlinear least squares estimation aims at fitting observations with a nonlinear model of the form $a_i \approx f(b_i, x)$ for some function f parametrized with an unknown vector x. Minimizing the sum of squares error

$$\min_x \sum_i r_i(x)^2, \quad \text{with} \quad r_i(x) = a_i - f(b_i, x)$$

is generally a non-convex optimization problem. The optimality condition is given by

$$\sum_i r_i \frac{\partial r_i}{\partial x_j} = 0, \quad \forall j \in \{1, ..., d\}.$$

Typically one uses iterative algorithms to solve these equation.

12.5 Newton's Method

Newton's method is a second-order method that makes use of the second derivative. Let x_t be the estimated solution after t iterations. Then the Taylor approximation of the cost function $E(x)$ in the vicinity of this estimate is (1. fitting a parabola):

$$E(x) \approx E(x_t) + \mathbf{J}^T(x - x_t) + \frac{1}{2}(x - x_t)^T \boldsymbol{H}(x - x_t)$$

with the Jacobian $\mathbf{J} = \frac{dE}{dx}(x_t)$ and the Hessian matrix $\boldsymbol{H} = \frac{d^2E}{dx^2}(x_t)$. The optimality condition is (2. going to minimum of the parabola):

$$\frac{dE}{dx} = \mathbf{J} + \boldsymbol{H}(x - x_t) = 0$$

The next iterate is the minimizer x:

$$x_{t+1} = x_t - \boldsymbol{H}^{-1}\mathbf{J}.$$

which requires $\boldsymbol{H}$ to be positive definite. When applicable, second-order methods are often faster (measured in number of iterations) than first-order methods, but computing and inverting the Hessian may be challenging/costly.

12.6 The Gauss-Newton Algorithm

The Gauss-Newton algorithm is a method to solve nonlinear least-squares problems of the form:

$$\min_x \sum_i r_i(x)^2$$

It is an approximation of the Newton method and expresses the Hessian via the Jacobian:

$$H \approx 2\boldsymbol{J}^T\boldsymbol{J} \quad \text{with the Jacobian } \boldsymbol{J} = \frac{dr}{dx}.$$

Computing the derivative of the first order Taylor approximation of the sum of squared errors w.r.t. x, equating it to zero and solving for x leads to:

$$\begin{aligned} x_{t+1} &= x_t + \Delta \\ \Delta &= -(\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T r. \end{aligned}$$

In contrast to the Newton's algorithm, the Gauss-Newton algorithm does not require the computation of the Hessian. This approximation of the Hessian is valid if the residuum r_i is small or if it is close to linear.

12.7 The Levenberg-Marquardt Algorithm [old]

The Newton algorithm can be modified (damped)

$$x_{t+1} = x_t - (\boldsymbol{H} + \lambda \boldsymbol{I}_n)^{-1} g$$

to create a hybrid between the Newton method ($\lambda = 0$) and a gradient descent with step size $\frac{1}{\lambda}$ (for $\lambda \to \infty$).

In the same manner Levenberg suggested to damp the Gauss-Newton algorithm for nonlinear least squares:

$$\begin{aligned} x_{t+1} &= x_t + \Delta \\ \Delta &= -(\boldsymbol{J}^T\boldsymbol{J} + \lambda \boldsymbol{I}_n)^{-1}\boldsymbol{J}^T r \end{aligned}$$

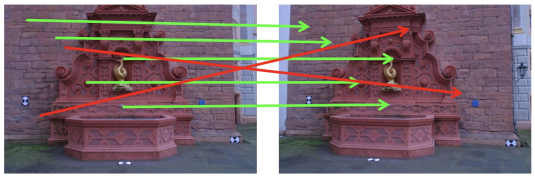
Marquardt suggested a more adaptive componentwise damping of the form

$$\Delta = -(\boldsymbol{J}^T\boldsymbol{J} + \lambda \cdot \mathrm{diag}(\boldsymbol{J}^T\boldsymbol{J}))^{-1}\boldsymbol{J}^T r$$

which avoids slow convergence in directions of small gradient.

13 Robust Estimation

Generally, correspondences are determined based on descriptor similarity. Ideally, all correspondences fit the same geometric transformation, but in practice, they are contaminated by outliers (i.e. mis-matches). Outliers are caused by repetitive patterns, geometric and photometric changes, noise, occlusions, moving objects, etc. Robust estimation by outlier removal is essential for reliable and accurate localization.

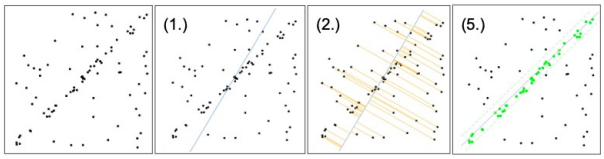


The following algorithms are general methods for model fitting in the presence of outliers, such as essential/fundamental matrix or homography estimation, PnP, etc. A data point that can be roughly fitted by a model is called an inlier.

13.1 Expectation Maximization (EM) Algorithm

1. Compute an initial solution by minimizing the least-squares error $\min \sum r_i^2$ using all points.
2. **E-Step:** Based on the initial model, compute the per-point residual error r_i. Each point receives a weight $w_i = e^{-r_i} \in [0, 1]$ that describes the probability that a point is an inlier.
3. **M-Step:** Re-estimate the model parameters by minimizing a weighted least-squares error $\min \sum w_i r_i^2$.
4. Repeat steps 2 and 3 until convergence.
5. Weights above a threshold mark inlier points.

Visualization for a line-fitting example:

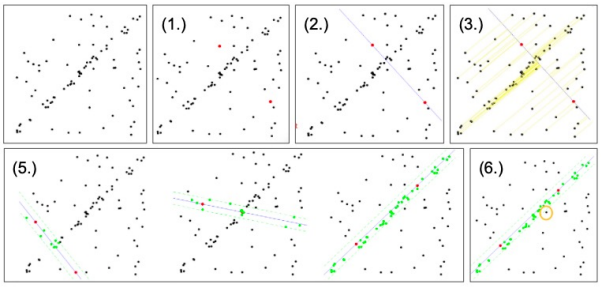


If the initial solution is far from the ground-truth, the EM algorithm converges to the wrong overall solution.

13.2 Random Sample Consensus (RANSAC)

This is the most popular method for model fitting in the presence of outliers. Compared to EM, it is not sensitive to initialization and is non-deterministic, such that results vary from run to run. Pipeline:

1. Randomly sample n points for the *minimal case* (e.g. 5-point algorithm $\rightarrow$ 5 correspondences).
2. "Hypothesize" the model using the n samples, regardless of whether the result makes sense.
3. Compute the residual error for each point given the hypothesized model (i.e. camera pose estimation $\rightarrow$ point-to-epipolar-line distance).
4. Generate the inlier set by selecting all points whose residual error is below some threshold. All other points are outliers w.r.t. this model. If the model was hypothesized via its *true inliers*, this inlier set should have a high cardinality.
5. Perform k iterations over steps 1.-4.
6. Select the inlier set with the highest cardinality.
7. Optional: Compute the final model parameters using all inlier points. This step can improve accuracy by noise compensation.

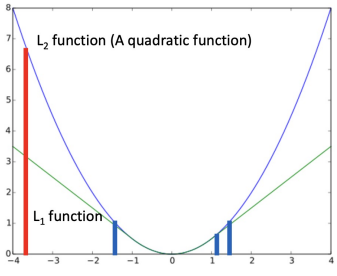


Minimal Case Instead of sampling the minimal case in step (1.), we can also sample a *quasi-minimal case*, e.g. for the 8-point algorithm. However, the minimal case is preferred, because of the greater likelihood of sampling a true inlier set in step (1.). However, *redundant samples* over-determine the linear system and compensate noise like in step (7.).

Number of Iterations If we have a rough guess of the inlier ratio $w = \frac{\text{No. of inliers}}{\text{No. of points}}$, we can guarantee at least one valid sampling among $k = \frac{\log(1-p)}{\log(1-w)}$ samplings with confidence p.

13.3 Robust Kernel (M-estimator)

In the photometric or re-projection error, outliers (e.g. points outside of the image boundaries) contribute to a high sum-of-squared-errors loss. The Huber loss combines the L_1 and L_2 losses to *implicitly* reduce the effect of outliers.



14 SLAM and SfM

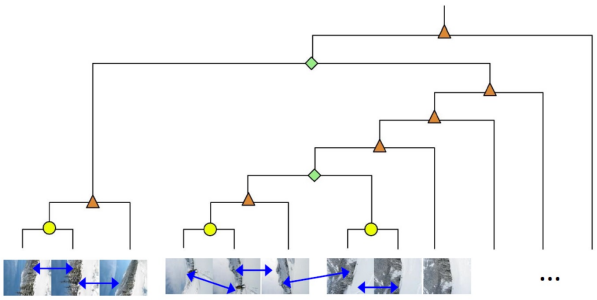
14.1 Structure from motion (SfM)

SfM is the process of estimating the 3D structure of a scene from a set of 2D images. Given a set of images with feature correspondences, we compute the 3D positions of features ("structure"), the extrinsic and optionally the intrinsic camera parameters ("motion"). SfM consists of:

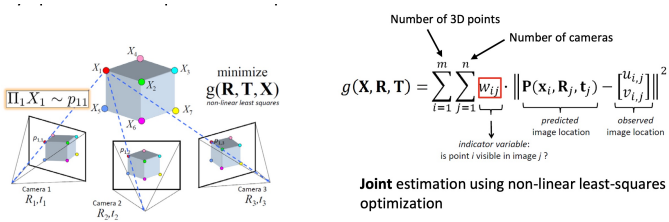
1. Feature detection and feature matching.
2. Camera pose estimation (via epipolar geometry, PnP, etc.).
3. 3D reconstruction by triangulation.

These steps are optimized jointly through bundle adjustment.

Hierarchical SfM This method takes a set of unordered images as input and estimates the 3D structure and camera poses in a bottom-up way. We detect feature correspondences between all possible pairs of unordered images and create a topological tree based on the number of matches: any two images with many matches are close to each other. First, we generate local models based on 2D-2D geometry (○) and 3D-2D geometry (Δ; via PnP) These initial steps are called *two-view or three-view SfM*. Then, we merge these models based on 3D-3D geometry (♢).



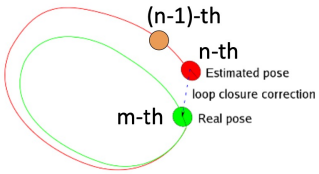
At every step we can optimize camera poses and 3D points jointly:



Sequential SfM Input images are sequential and this method is equivalent to visual odometry (VO; chapter 10). We incrementally estimate 3D structure and camera poses.

14.2 Simultaneous Localization and Mapping (SLAM)

SLAM can be treated as a combination of VO and loop closure. Noise in VO leads to a drift error over time, but SLAM guarantees global consistency through loop closure.

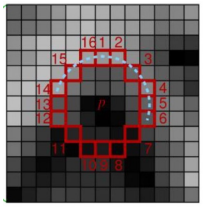


1. **Loop detection:** if we detect similarity in the n^{th} input image or point-cloud with respect to a m^{th} instance before, we have detected a closed loop.
2. **Loop correction:** we use the estimated m^{th} pose and computed Sim(3) transformation to update the n^{th} pose. Then we use the estimated relative pose to update the $n-1^{\text{th}}$ pose.

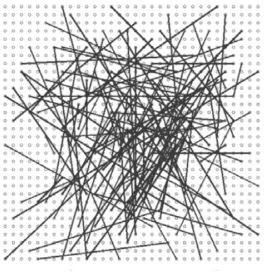
Parallel Tracking and Mapping (PTAM) This monocular-only, feature based algorithm relies on FAST corner detection and jointly optimizes poses and structure by minimizing the re-projection error with sliding-window bundle-adjustment. In order to avoid storing local maps for every frame, it only stores the point-clouds of **key-frames** to build the global map. It also improves efficiency to real-time by performing localization and mapping on **parallel threads**. Its disadvantages are that relocalization only works in a small neighborhood and there is no global optimization.

ORB-SLAM This method improves upon PTAM by use of ORB-features that are faster to compute and match. The ORB-feature description uses the *oriented FAST corner detector* for feature detection and the *rotated BRIEF descriptor* for high-speed descriptor computation and matching.

Features from Accelerated Segment Test (FAST) A point is a FAST corner if it is brighter or darker than a set of pixels on a ring around it plus/minus some threshold. Is is the fastest corner detection to-date, but is very sensitive to image noise.



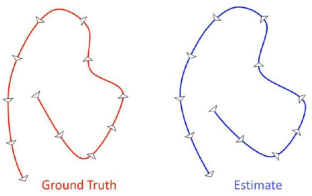
Binary Robust Independent Elementary Features (BRIEF) This algorithm samples 128 intensity pairs (p_1^i, p_2^i) within a square patch around the detected keypoint. The i-th element of the descriptor vector is 1 if $I(p_1^i) < I(p_2^i)$ else 0. This descriptor allows fast Hamming distance matching, but is not scale/rotation invariant.



14.3 Evaluation Metrics for SLAM

We evaluate a SLAM algorithm by comparing the estimated trajectory with the ground truth trajectory obtained from GPS or motion tracking systems.
Naively, aligning the first poses and measuring the difference between the final poses is *not repeatable* because SLAM methods are non-deterministic and *not meaningful* because the error in the final pose is sensitive to the trajectory shape and the trajectories can have different scales.

Absolute Trajectory Error (ATE) We obtain 3D-3D point correspondences by aligning timestamps. Then, we align the trajectory estimate to the ground truth using a similarity transform $(\mathbf{R}, \mathbf{t}, s)$ by minimizing the sum of squared position errors:



$$\mathbf{R}, \mathbf{t}, s = \underset{\mathbf{R},\mathbf{t},s}{\operatorname{argin}} \sum_{k=0}^{n} \left\| \hat{\mathbf{C}}_k - s\mathbf{R}\mathbf{C}_k - \mathbf{t} \right\|^2$$

Then we compute the RMSE after alignment:

$$\text{RMSE} = \sqrt{\frac{\sum_{k=0}^{n} \left\| \hat{\mathbf{C}}_k - s\mathbf{R}\mathbf{C}_k - \mathbf{t} \right\|^2}{n}}$$

The ATE captures the global error using a single number, but does not encode relative error of sub-trajectories.