

Image Processing and Computer Vision 2

FS26 – Martin Weisenhorn

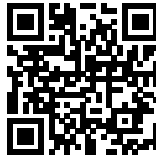
Autoren:

Martina Knobel, Sandro Heidrich, Andrin Rüttsche, Fabian Suter, Dominik Thurnherr

Version:

1.0.20260917

<https://github.com/FabianSuter/IPCv2>



Contents

I Segmentation	3		
1 Intensity based segmentation	3		
1.1 General	3		
1.1.1 Color based clustering	3		
1.1.2 Region based clustering	3		
1.1.3 Noise in thresholding	3		
1.1.4 Illumination, reflectance in thresholding	3		
1.2 Otsu's method (global)	3		
1.2.1 Simplified evaluation	3		
1.2.2 Separability measure	3		
1.2.3 Improvements	3		
1.3 Basic method (iterative)	3		
1.4 k-means clustering	3		
1.5 Comparison	4		
1.5.1 Otsu's method	4		
1.5.2 Basic method	4		
1.5.3 k-Means	4		
2 Region based segmentation	4		
2.1 Splitting and merging	4		
2.1.1 Quadtree	4		
2.2 SLIC - Simple linear iterative clustering	4		
3 Edge detection	4		
3.1 General	4		
3.1.1 Derivatives	4		
3.1.2 Gradient operators	4		
3.1.3 Derivative approximations	4		
3.2 Point detection	4		
3.2.1 Laplacian mask	5		
3.3 Line detection	5		
3.3.1 Wide lines	5		
3.3.2 Discrete Laplacian mask	5		
3.4 Noise mitigation	5		
3.4.1 Sensitivity	5		
3.5 Gradient based edge detection	5		
4 Local algorithms	5		
4.1 Canny Edge detector	5		
4.1.1 Summarization	5		
4.1.2 Nonmaxima suppression	5		
4.1.3 Double thresholding and linking	5		
4.2 Canny inspired algorithms	5		
5 Global Algorithms	5		
5.1 Hough Transform	5		
5.1.1 Parametrization	5		
5.1.2 Matching algorithm	5		
5.1.3 Accumulator cell $A(i,j)$	6		
5.1.4 Hough transform example	6		
5.1.5 Properties	6		
5.1.6 Extensions	6		
5.2 RANSAC (RANDOM SAmple Consensus) for circles	6		
5.2.1 Parameters	6		
5.2.2 Algorithm	6		
5.2.3 Success rate	6		
5.3 RANSAC vs. Hough transform	6		
II Local Features	6		
6 General	6		
6.1 Applications	6		
7 Keypoint detection	6		
7.1 Harris corner detection	6		
7.1.1 Principle components	6		
7.1.2 Requirements to Corner Measure	6		
7.1.3 Point cloud, weighted gradients	7		
7.1.4 Covariance matrix of pointcloud	7		
7.1.5 Corner detection	7		
7.1.6 Improvements	7		
7.2 FAST corner detection	7		
8 Feature description	7		
8.1 General	7		
8.1.1 Scale	7		
8.1.2 Rotation	7		
8.1.3 Intensity normalization	7		
8.1.4 Histogram of Gradients (HoG)	7		
8.2 SIFT features	8		
8.3 Other descriptors	8		
8.3.1 Different approaches and algorithms	8		
8.3.2 Trained descriptors	8		
8.3.3 Selection criteria	8		
9 Feature matching	8		
9.1 Matching metrics	8		
9.2 Brute Force Matching	8		
9.3 Outlier rejection	8		
9.3.1 Ratio test	8		
9.3.2 Cross check	8		
9.3.3 Combination	8		
9.4 Efficient matching	8		
9.4.1 Minimum Distance Classifier	8		
10 Optical Flow	8		
10.1 Brightness constancy & the flow constraint	8		
10.2 Aperture problem	8		
10.3 Lucas–Kanade	9		
III Image formation	9		

11	3D to 2D projection	9	22	Triangulation	14
11.1	General	9	22.0.1	Stereo projection	14
11.1.1	Pinhole-camera model	9	22.0.2	Stereo projection errors	14
11.1.2	Inverted pinhole-camera model	9	22.1	Reprojection error	14
11.1.3	Centered and inverted pinhole-camera model	9	23	Epipolar geometry	14
11.2	Perspective projection	9	23.1	Derivation Step 1	14
11.3	Homogeneous image coordinates	9	23.2	Derivation Step 2	15
11.4	Lines and points	9	23.2.1	Fundamental matrix	15
11.4.1	Representation of a Line	9	23.2.2	Essential matrix	15
11.4.2	Computing a Line from Two Points	9	23.3	Applications	15
11.4.3	Loss of scale information	10	24	Rectification	15
11.5	Camera intrinsics	10	24.1	Stereo rectification	15
12	3D coordinate transforms	10	VI	Object Recognition	15
12.1	Coordinate frame	10	25	Matching by correlation	15
12.1.1	Changing the basis of a point	10	25.1	Cross correlation	15
12.2	Rotation matrix	10	25.2	Normalized cross correlation coefficient	15
12.3	Translation	10	25.3	Template matching	15
12.3.1	Notation	10	26	Patterns (feature vectors)	15
12.3.2	Rotation plus translation	10	27	Object recognition	15
12.4	Rigid motion matrix	10	27.1	Minimum-distance classifier	15
13	3D rotations	10	27.1.1	Finding the decision boundary	16
13.1	Euler angles	10	27.1.2	Flower example	16
13.2	Axis/angle Rodrigues' formula	10	27.2	Maximum A Posteriority (MAP, Bayes) classifier	16
13.2.1	Rotation vector	11	27.2.1	MAP classifier with identical covariance matrices	16
13.3	Interpreting Rodrigues' rotation matrix	11	27.2.2	MAP example	16
13.4	Notation for software code	11	27.2.3	Finding the boundry when $p(x)$ is identical for all classes	16
14	Camera matrix	11	28	Region-Based Features	16
14.1	Projecting points	11	28.1	Contours as polygons	16
IV	Projective geometry	11	28.1.1	Hierarchy of Contours	16
15	Geometric definitions	11	28.1.2	OpenCV: Convex hull	16
15.1	Homogeneous coordinates	11	28.1.3	OpenCV: Bounding box	16
15.2	2D lines	11	28.1.4	OpenCV: minAreaRect	16
15.2.1	Points on a line	11	28.1.5	OpenCV: fitEllipse	16
15.3	Cross product	11	28.2	Simple descriptors	16
15.4	Points/line at infinity	11	28.2.1	Perimeter and area	16
16	Orthography	11	28.2.2	Diameter and area	16
17	Homography	12	28.2.3	Fibres	16
17.1	Properties	12	28.2.4	Example	16
17.2	Mapping from a planar object	12	28.3	Area-based descriptors	16
17.3	Mapping from one camera to another	12	28.3.1	Object centroid	17
17.4	Mapping between rotated camera views	12	28.3.2	2D central moments	17
18	Singular Value Decomposition (SVD)	12	28.3.3	Hu-moments	17
18.1	Proof of SVD	12	28.3.4	Principal axis of inertia	17
19	Homography estimation & Pose estimation	12	28.4	Texture based descriptors	17
19.1	General	12	28.4.1	Co-occurrence matrix	17
19.2	Estimating H	12	28.4.2	Spectral approach	17
19.2.1	Matrix inversion	12	VII	Neural Networks	17
19.2.2	Least squares	12	29	Perceptron	17
19.2.3	Direct Linear Transfrom (DLT)	12	29.1	Two linearly separable classes	17
19.3	Line mapping by homography	12	29.2	Two linearly inseparable classes	17
19.4	Direct Linear Transform (DLT) for Homography Estimation	13	29.2.1	Stochastic Gradient Descent (SGD)	17
19.5	Pose estimation	13	30	Multilayer feedforward NN	18
19.5.1	Dolly zoom	13	30.1	Forward pass	18
20	Camera Calibration	13	30.2	Matrix formulation	18
20.1	Image distortion (intrinsic calibration)	13	30.2.1	Mini Batches	18
20.1.1	Radial distortion	13	30.3	Backpropagation	18
20.2	Tangential distortion	13	30.3.1	Matrix formulation	18
20.2.1	Mapping for radial and tangential undistortion	14	30.4	Softmax output	18
20.3	Other camera issues	14	VIII	Convolutional Neural Networks (CNN)	18
20.3.1	Vignetting	14	31	Neural computations in a CNN	18
20.3.2	Color	14	32	Forward pass	18
20.3.3	Sampling time (Shutter)	14	33	Backpropagation	18
V	3D Vision	14	34	Math-Stuff	18
21	3D vision problems	14	34.1	Trigonometry	19
21.1	Single-view pose-estimation	14	34.2	Crossproduct	19
21.2	Stereo-view depth map	14	34.3	Trace of a matrix	19
21.3	Stereo-view structure-from-motion	14	34.4	Inverse of a 3×3 matrix	19
			34.5	Eigenvalue shortcut for 2×2 matrices	19

I Segmentation

1 Intensity based segmentation

1.1 General

- Single threshold, i.e. $g(x, y) = \begin{cases} 0 & f(x, y) \leq T \\ 255 & f(x, y) > T \end{cases}$
- Multiple thresholds for mult. objects, i.e. $g(x, y) = \begin{cases} 0 & f(x, y) < T \\ 128 & T_1 \leq f(x, y) < T_2 \\ 255 & f(x, y) \geq T \end{cases}$

1.1.1 Color based clustering

Colors are represented with 3D vectors (R, G, B).

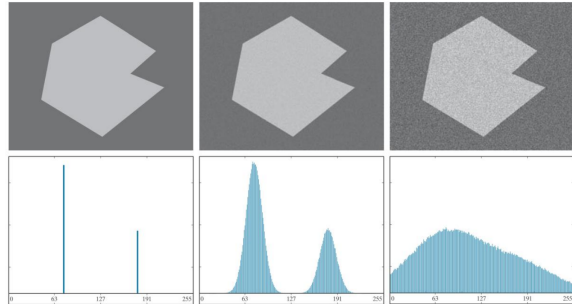
Color segmentation is done by defining k cluster centers $\mathbf{m}_i$ and assigning each color pixel the closest center value. These clusters could be geometrically disconnected on a picture, since pixels with the same color belong to the same cluster.

1.1.2 Region based clustering

Pixels belong to the same cluster if a combination of color distance and spatial distance is below a threshold.

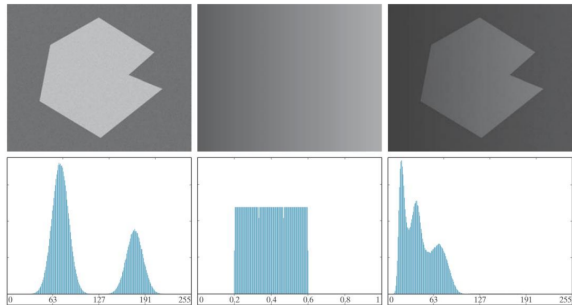
1.1.3 Noise in thresholding

Noise impacts threshold levels and how they need to be placed. Left is noiseless, middle is Gaussian noise and 10 intensity levels, right is Gaussian noise and 50 intensity levels.

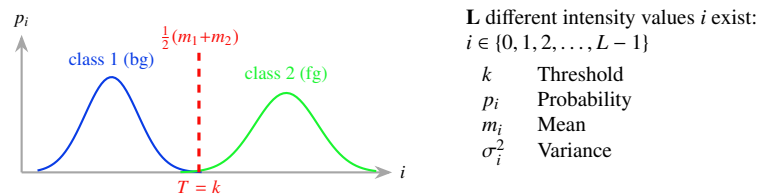


1.1.4 Illumination, reflectance in thresholding

Illumination and reflectance impacts threshold levels and how they need to be placed. Left is a noisy image, middle is intensity ramp from [0.2, 0.6], right is product of both.



1.2 Otsu's method (global)



- In-Class variance of background: $\sigma_1^2(k) := \frac{1}{P_1} \sum_{i=0}^{k-1} p(i)(i - m_1)^2$
- In-Class variance of foreground: $\sigma_2^2(k) := \frac{1}{P_2} \sum_{i=k}^{L-1} p(i)(i - m_2)^2$
- Mean intensity value of classes: $m_1(k) := \frac{1}{P_1} \sum_{i=0}^{k-1} p(i)i$ $m_2(k) := \frac{1}{P_2} \sum_{i=k}^{L-1} p(i)i$
- Probability of classes: $P_1(k) := \sum_{i=0}^{k-1} p(i)$ $P_2(k) := \sum_{i=k}^{L-1} p(i)$
- Within-Class variance (sum): $\sigma_w^2(k) := \sum_{i=0}^{k-1} p_i \cdot (i - m_1(k))^2 + \sum_{i=k}^{L-1} p_i \cdot (i - m_2(k))^2$
- Conditional probability (Bayes' theorem):

$$P(A | B) = \frac{P(A \cap B)}{P(B)} \rightarrow P(i | i \in \text{Class}_1) = \frac{P(i \cap i \in \text{Class}_1)}{\underbrace{P(i \in \text{Class}_1)}_{P_1}}$$

Find the lowest σ_w^2 for the ultimate threshold $\hat{k} = \arg \min_k \{\sigma_w^2(k)\} = \frac{m_1(\hat{k}) + m_2(\hat{k})}{2}$

1.2.1 Simplified evaluation

$$\begin{aligned} \text{Between-Class variance: } \sigma_B^2 &:= \sigma_G^2 - \sigma_w^2(k) \\ &= P_1 \cdot (m_G - m_1)^2 + P_2 \cdot (m_G - m_2)^2 \\ &= \frac{P_1}{1 - P_1} (m_G - m_1)^2 \\ \text{Global variance: } \sigma_G^2 &:= \sum_{i=0}^{L-1} p(i) \cdot (i - m_G)^2 \\ \text{Global mean value: } m_G &:= \sum_{i=0}^{L-1} i \cdot p(i) \end{aligned}$$

Find the highest σ_B^2 for the ultimate threshold $\hat{k} = \arg \max_k \{\sigma_B^2(k)\}$

1.2.2 Separability measure

Classes are easily separable and compact when $\sigma_w^2(\hat{k}) \ll \sigma_G^2$. Separation is not possible, when they are equal.

$$\eta(\hat{k}) := 1 - \frac{\sigma_w^2(\hat{k})}{\sigma_G^2} = \frac{\sigma_B^2(\hat{k})}{\sigma_G^2} = \begin{cases} 1 & \text{perfect separability} \\ 0 < \dots < 1 & \dots \\ 0 & \text{no separability} \end{cases}$$

1.2.3 Improvements

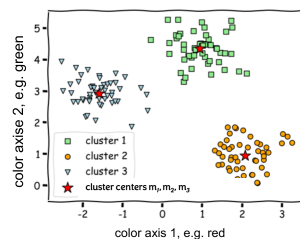
Smooth image first (reduces noise); use *edges* to mask the histogram (only count pixels on/near edges) when foreground objects are small.

1.3 Basic method (iterative)

Alternative to Otsu's method, a simple iterative procedure. T is the threshold variable, like $\hat{k}$ in Otsu's method.

- Initialize the threshold with the global mean intensity. Use the histogram counts n_i of all gray values i .
$$T^{(0)} := m_G = \frac{\sum_{i=0}^{L-1} i n_i}{\sum_{i=0}^{L-1} n_i}$$
- Split the histogram into two classes using the current threshold. All gray values below or equal to the threshold go to class C_1 ; all gray values above the threshold go to class C_2 .
$$C_1 = \{i \mid i \leq T^{(r)}\}, \quad C_2 = \{i \mid i > T^{(r)}\}$$
- Compute the mean gray value of each class. The histogram counts n_i are used as weights. Empty classes are not allowed.
$$m_1^{(r)} = \frac{\sum_{i \in C_1} i n_i}{\sum_{i \in C_1} n_i}, \quad m_2^{(r)} = \frac{\sum_{i \in C_2} i n_i}{\sum_{i \in C_2} n_i}$$
- Update the threshold. The new threshold is placed exactly between the two class means.
$$T^{(r+1)} = \frac{m_1^{(r)} + m_2^{(r)}}{2}$$
- Repeat the split and update steps until the threshold no longer changes significantly. For discrete histograms, it is often enough to check whether the class split stays the same.
$$|T^{(r+1)} - T^{(r)}| < \Delta T$$

1.4 k-means clustering



Each pixel is in a set of color vectors $Q := \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_q\}$
Partition the set Q into $k < Q$ disjoint cluster sets $C := \{C_1, C_2, \dots, C_k\}$
The cluster centers of the sets C_i are
$$M := \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_k\}$$

$$= \arg \min_C \left(\sum_{i=1}^k \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2 \right)$$

- Initialize a set of means: $\mathbf{m}_i(1), i = 1, 2, \dots, k$
- Assign each sample to closest cluster set mean:
 $\mathbf{z}_q \rightarrow C_i$ if $\|\mathbf{z}_q - \mathbf{m}_i\|^2 < \|\mathbf{z}_q - \mathbf{m}_j\|^2$
 $j = 1, 2, \dots, k (j \neq i);$
 $q = 1, 2, \dots, Q$
- Update cluster center means:
$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \quad i = 1, 2, \dots, k$$
- Test for completion with residual error: Stop if E is small enough, else go back to 2
$$E = \sum_{i=1}^k \|\mathbf{m}_i - \mathbf{m}_{i, \text{previous}}\|^2$$

- + Optimizes Within-Class-Variance just like Otsu's and Basic Method
- + Class-borders can be more general than just parallel to color axes
- + The algorithm can be used to cluster any type of vectors not just color
- + Computationally more efficient than Otsu, for vectors that have more than 3D
- + An arbitrary large number Q of classes is allowed
- Not guaranteed to converge to the optimal solution

- Solution depends on starting values
- k needs to be known upfront
- The class borders are limited to convex shapes
- Pixels with color outliers (pixel err. of camera) will have a strong effect on the outcome

1.5 Comparison

1.5.1 Otsu's method

- **Minimizes the Within-Class Variance**
- Good for binary thresholding
- Provides optimal solution in one complex step

1.5.2 Basic method

- **Minimizes the Within-Class Variance**
- For binary thresholding
- Iterative algorithm, not converging to the global optimum in general

1.5.3 k-Means

- **Minimizes the Within-Class Variance**
- Generalization of Basic Method to color- and multidimensional-clustering
- Iterative algorithm, not converging to the global optimum in general
- Good initialization of the cluster centers is key, consider the algorithm known as k-Means++

2 Region based segmentation

2.1 Splitting and merging

Split an image into a set of arbitrary disjoint regions R and merge or split to remove unjustified regions. I.e. $G_{max} = 6, G_{min} = 1$

1	1	1	5
1	2	5	5
1	2	4	5
1	1	6	6

1	1	1	5
1	2	5	5
1	2	4	5
1	1	6	6

1	1	1	5
1	2	5	5
1	2	4	5
1	1	6	6

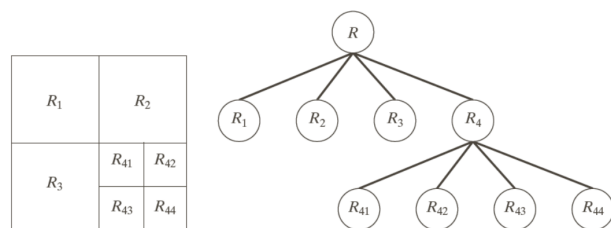
$f(x,y)$

split

merge

$$Q(R) = \begin{cases} TRUE & |G_{max} - G_{min}| \leq 1 \\ FALSE & \text{otherwise} \end{cases}$$

2.1.1 Quadtree



1. Split into for disjoint any region R_i for which $Q(R_i) = FALSE$
2. When no further splitting is possible, merge any adjacent regions where $Q(R_j \cup R_k) = TRUE$
3. Stop when no further merging is possible

2.2 SLIC - Simple linear iterative clustering

k-Means ignores neighbourhoods and creates fringed segments. SLIC takes the neighbourhoods into account by augmenting the color vector $\mathbf{z}$ with the pixel coordinates (x,y) : $\mathbf{z} = [r, g, b, x, y]^T$. A suited distance measure D allows to weight color distance d_c and geometric distance d_s separately:

$$D = \left[\left(\frac{d_c}{d_{cm}} \right)^2 + \left(\frac{d_s}{d_{sm}} \right)^2 \right]^{1/2}$$

$$d_c = [(r_j - r_i)^2 + (g_j - g_i)^2 + (b_j - b_i)^2]^{1/2} \quad d_s = [(x_j - x_i)^2 + (y_j - y_i)^2]^{1/2}$$

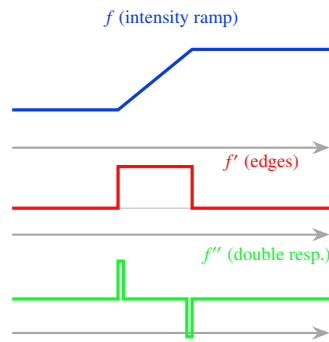
The algorithm is identical to k-Means (siehe 1.4). SLIC is mainly a preprocessing step for other algorithms, for solving the stereo matching problem and for image feature detection.

3 Edge detection

3.1 General

Defined as abrupt intensity change, either at a point or a line, creating an **edge**

3.1.1 Derivatives



- **First-order derivatives** highlight edges of an image
- **Second-order derivatives** highlight fine details, such as thin lines, isolated points and noise
- **Second-order derivatives** produce a double-edge response at ramp and step transitions in intensity
- The sign of a derivative tells, whether the edge transitions from dark to bright or vice versa

3.1.2 Gradient operators

Roberts mask

Single 2x2 mask, used for detecting diagonal lines. Misses a central point and has no smoothing built in, badly approximates the continuous differentiation, see also problem 10.1 in Gonzales a. Woods

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \quad \begin{bmatrix} -1 & 0 & 0 & -1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6) \quad \text{Roberts}$$

Prewitt mask

A 3x3 mask is the smallest mask with a clear center

$$\begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} * \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

vertical			horizontal		
-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

Prewitt

Sobel mask

By far the most popular one

$$\begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} * \begin{bmatrix} +1 & +2 & +1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

Sobel

3.1.3 Derivative approximations

Finite differences approximate derivatives from neighbouring samples of a discrete signal or image. Let the sample spacing be Δx .

Forward difference: The one-sided forward difference uses the current and the next sample:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

The approximation error is

$$O(\Delta x).$$

So it is a first-order approximation.

Backward difference: The one-sided backward difference uses the current and the previous sample:

$$f'(x) \approx \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

The approximation error is also

$$O(\Delta x).$$

Central difference: The two-sided central difference uses the previous and the next sample:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}$$

The approximation error is

$$O(\Delta x^2).$$

Therefore, the central difference is usually more accurate than the one-sided forward or backward difference.

3.2 Point detection

The second derivative is a measure of curvature and thus sensitive to isolated points

$$\text{2nd derivative along vert. axis} \quad \frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$\text{2nd derivative along hor. axis} \quad \frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

$$\text{Sum of them} \quad \nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

-1	-1	-1
-1	8	-1
-1	-1	-1

3.2.1 Laplacian mask

Detects an isolated point when the absolute value of Z exceeds a threshold T

$$Z(x, y) = f(x, y) \star h(x, y)$$

$$g(x, y) = \begin{cases} 1 & \text{if } |Z(x, y)| > T \\ 0 & \text{otherwise} \end{cases}$$

(Set threshold T relative, i.e. in %)

-1	-1	-1
-1	8	-1
-1	-1	-1

3.3 Line detection

Also uses the second derivative and therefore the Laplacian mask. Usually only parts above a certain threshold are kept, otherwise two lines could be detected, at the beginning and at the end (also called double response).

3.3.1 Wide lines

Wide lines, wider than the mask, will get detected twice $\Rightarrow$ The line needs to be thin compared to the filter.

3.3.2 Discrete Laplacian mask

The continuous Laplace mask is perfectly isotropic. Should a line with a known direction be detected, you can use special (discrete) masks. Angles are with respect to the axis system (X: $\downarrow$, Y: $\rightarrow$)

-1	-1	-1	2	-1	-1	-1	2	-1	-1	-1	2
2	2	2	-1	2	-1	-1	2	-1	-1	2	-1
-1	-1	-1	-1	-1	2	-1	2	-1	2	-1	-1
Horizontal			+45°			Vertical			-45°		

3.4 Noise mitigation

3.4.1 Sensitivity

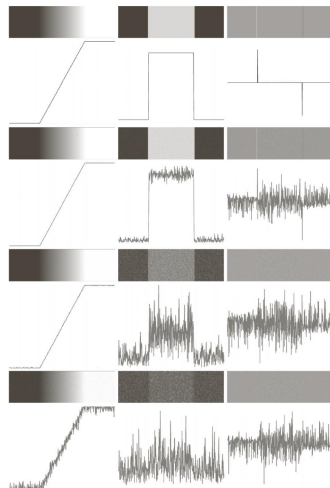
Derivatives are sensitive to noise and results in less obvious features, even when the noise is almost invisible.

Therefore the image usually needs to be **low-pass-filtered** before a derivative based edge detection scheme can be successful.

First column: Ramp edges, corrupted with Gaussian noise of zero mean and deviations of 0.0, 0.1, 1.0 and 10.0

Second column: First derivative images

Third column: Second derivative images



Sobel and Prewitt masks already have some built-in smoothing

3.5 Gradient based edge detection

Since first order derivatives are helpful, it's easy to combine the partial derivatives into one value $\rightarrow$ the magnitude of the gradient

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

The gradient vector points into the direction of greatest change, while the magnitude is the value of the rate of change in this direction.

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2 + g_y^2}$$
$$M(x, y) \approx |g_x| + |g_y|$$

The direction is defined as the angle enclosed between the x -axis and the vector, while the vector is *orthogonal* to the edge.

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y}{g_x} \right]$$

Masks for horizontal and vertical lines

g_x	-1	-1	-1
	0	0	0
	1	1	1

g_y	-1	0	1
	-1	0	1
	-1	0	1

Masks for diagonal lines, i.e. $+/- 45^\circ$

0	1	1	-1	-1	0
-1	0	1	-1	0	1
-1	-1	0	0	1	1

0	1	1	-1	-1	0
-1	0	1	-1	0	1
-1	-1	0	0	1	1

Prewitt

0	1	2	-2	-1	0
-1	0	1	-1	0	1
-2	-1	0	0	1	2

0	1	2	-2	-1	0
-1	0	1	-1	0	1
-2	-1	0	0	1	2

Sobel

3×3 is the smallest mask with a clear center.

All here are non-isotropic

4 Local algorithms

4.1 Canny Edge detector

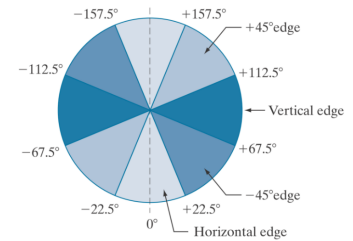
More complex, but also more powerful than the previous ones. Built to identify one pixel wide edges.

4.1.1 Summarization

- Smooth the input image with a Gaussian filter
- Compute the gradient magnitude and angle images
- Apply nonmaxima suppression to the gradient magnitude images
- Use double thresholding and connectivity analysis to detect and link edges

4.1.2 Nonmaxima suppression

Since only 4 directions are natural in a 3×3 neighborhood, round the calculated gradients to the nearest of these four directions.



4.1.3 Double thresholding and linking

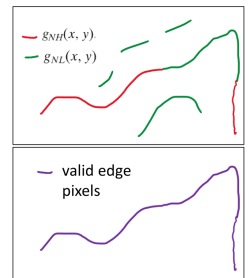
Set two thresholds:

- T_H for values that are almost certainly true edges $g_{NH}(x, y) := \begin{cases} 1 & \text{if } g_N(x, y) \geq T_H \\ 0 & \text{else} \end{cases}$
- T_L for values high enough to filter out noise but much lower than T_H (2 to 3 times), likely still true edges but also possibly erroneous edges $g_{NL}(x, y) := \begin{cases} 1 & \text{if } T_H > g_N(x, y) \geq T_L \\ 0 & \text{else} \end{cases}$

In $g_{NH}(x, y)$ are now the strong edge pixels, while $g_{NL}(x, y)$ contains the weak ones

Run a connectivity check:

- Locate the next unvisited strong edge pixel p in g_{NH}
- Mark all weak pixels in g_{NL} around p as valid
- Check, if all strong edge pixels have been visited
- Set all invalid pixels in g_{NL} to zero



4.2 Canny inspired algorithms

- EDLines and LSD for straight line detection
- Arc Support Lines for ellipse detection

5 Global Algorithms

5.1 Hough Transform

Idea: Identify lines on an image by checking how the pixels match with each possible straight line (template matching)

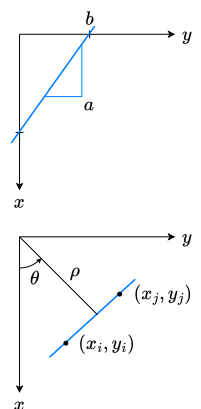
5.1.1 Parametrization

Points (x, y) on a line satisfy the equation $y = ax + b$ with inclination a and offset b . a however could assume arbitrarily large values.

$\Rightarrow$ Define the line normal with (ρ, θ) with the following properties:

- Only two parameters for every line
- Parameters have finite values
- Parameters in defined intervals, if the line intersects the image area

$$\rho = \begin{bmatrix} x \\ y \end{bmatrix}^T \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} = x \cos(\theta) + y \sin(\theta)$$



5.1.2 Matching algorithm

Check with L how many points of the image are on the line $L(x, y, \rho, \theta) = \begin{cases} 1 & \text{if point is on line} \\ 0 & \text{else} \end{cases}$

Count the number d of nonzero points on the image that lie on the line (ρ, θ)

$$d(\rho, \theta) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot L(x, y, \rho, \theta)$$

The combination (ρ, θ) with the highest score d describes the line with the largest number of pixels on the image

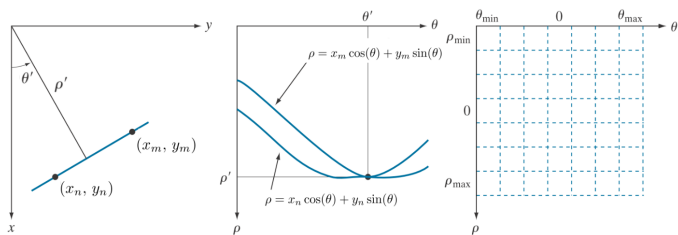
$$(\rho', \theta') = \arg \max_{(\rho, \theta)} \{d(\rho, \theta)\}$$

Compromise for $\Delta\rho, \Delta\theta$:

Smaller steps give a more precise fit to the given line
 Smaller steps require more evaluations → greater computational effort

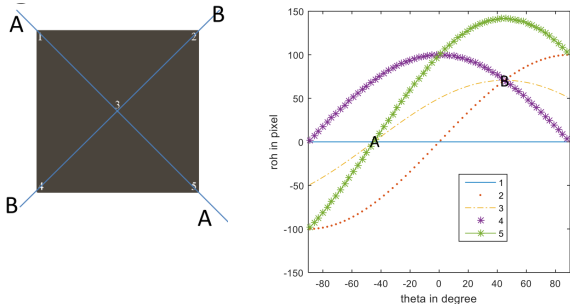
5.1.3 Accumulator cell A(i,j)

Plotting every nonzero product into an accumulator cell gives the *Hough transform* for every line with (ρ_i, θ_i) that matches on x, y .
 When you plot this transform for two separate points, the values where they intersect shows the (ρ, θ) for the line intersecting these two points:



5.1.4 Hough transform example

- Given points:
1. (0, 0)
 2. (0,100)
 3. (50, 50)
 4. (100, 0)
 5. (100,100)



5.1.5 Properties

- + Efficient implementation of template matching
- + Optimal for known templates and white gaussian noise
- Short and interrupted lines capture noise of unnecessarily many image pixels

5.1.6 Extensions

Circles could be found as well with the same algorithm and

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2$$

where c_3 is the radius and (c_1, c_2) define the circle center.
 The parameter space is now 3-dimensional and therefore more expensive.

5.2 RANSAC (RANDOM Sample Consensus) for circles

Find a small disc on an image with a small device. Do Prepressing with gradient magnitude computation and thresholding.
 You'll get a set of points cluttered around the disc and points outside as outliers.
 Find the disc's center and radius

5.2.1 Parameters

- **N° of trials:** Each trial consists of three points chosen at random
- **Sphere width:** Tolerance around circle for points still on circle
- **Min. number of inliers:** Assume a circle if minimum is reached

5.2.2 Algorithm

1. Randomly choose 3 points
2. Compute center and radius of circle with given points
3. Find points within circle's Sphere width (inliers)
4. IF number of inliers > min. number of inliers:
 1. Compute Least Squares fit ⇒ Refined sphere
 2. Find inliers to refined sphere
 3. Compute the resulting sum of squared errors (SSE)
 4. Save *number of inliers, SSE, center radius* to a Candidate List
5. Find highest inlier count on Candidate List
6. If multiple matches, take smallest SSE of matches

The algorithm prefers large circles

For small circles, replace the inlier count as a matching score with $\frac{\text{inlier count}}{\text{circle radius}}$

5.2.3 Success rate

Probability of at least one trial succeeding s denotes the number of inlier points required to uniquely define the model; for line detection, $s = 2$, for circle detection, $s = 3$, for homography detection, $s = 4$.

ε being the probability of picking an outlier

n being the number of trials needed for a given probability

Example:

$$p = 1 - (1 - (1 - \varepsilon)^s)^n$$

$$\varepsilon = \frac{N_{\text{outliers}}}{N_{\text{outliers}} + N_{\text{inliers}}}$$

$$n = \frac{\log(1 - p)}{\log(1 - (1 - \varepsilon)^s)}$$

Desired success rate p	Outlier rate ε	Required number of trials n
1	0	1
0.99	0.01	2
0.99	0.5	35
0.99	0.9	4603
0.999	0.01	2
0.999	0.5	52
0.999	0.9	6905

5.3 RANSAC vs. Hough transform

Common stuff	RANSAC	Hough transform
Assumption of specific model	Computationally inexpensive	Computationally expensive
Ident. of model params	Less reliable	Reliable
Search for params with strongest evidence	Returns a single object over a threshold	Returns multiple objects over a threshold
Robust against outliers	Finds reasonable params and then how much evidence for them	Collects evidence and then which one has the highest

II Local Features

6 General

6.1 Applications

- Camera calibration
- Image stitching
- Object detection / classification
- Pose estimation / tracking
- Motion based segmentation
- 3D object reconstruction
- Simultaneous localization and mapping
- many more, like bone age estimation

7 Keypoint detection

Detectors for blobs and corners

Adaptation to image category. If necessary, both can be used

Alternative: Dense sampling (keypoints lie on a grid)

Scale matters: a rounded contour zoomed out looks like a sharp corner

Corner detection

- Harris corner detection (1998)
cv2.cornerHarris()
cornerSubPix()
- Shi-Thomas corner detection (1994)
cv2.goodFeaturesToTrack()
- FAST (Rosten & Drummond 2005)

Multiscale blob/corner detection

- SIFT (David Lowe 2004)
cv2.SIFT()

7.1 Harris corner detection

7.1.1 Principle components

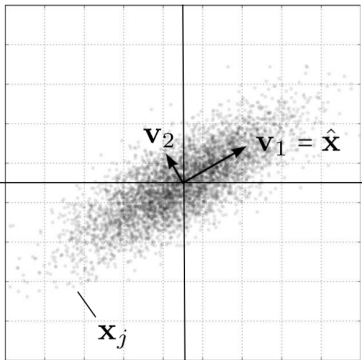
Given a point cloud of N points with co-ordinate vectors $x'_i = [x'_i, y'_i]^T$, we want to know the direction $\hat{\mathbf{x}}$ of the strongest extension, i.e. the direction $\hat{\mathbf{x}}$ of the principal axis.

$$\bar{\mathbf{x}}' := \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{x}'_i \quad \mathbf{x}_i := \mathbf{x}'_i - \bar{\mathbf{x}}' = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\mathbf{C} := \frac{1}{N} \mathbf{B}^T \mathbf{B}, \quad \mathbf{B} := \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

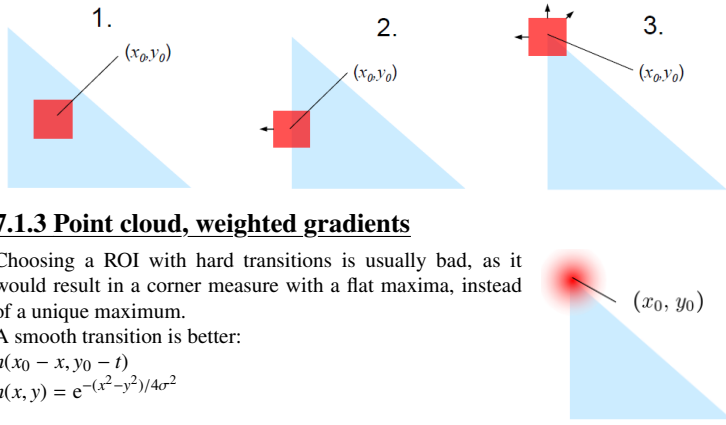
$$\mathbf{C} := \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^T \quad \mathbf{U} := [\mathbf{v}_1, \mathbf{v}_2]$$

$$\mathbf{\Lambda} := \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \quad \lambda_2 \leq \lambda_1$$



7.1.2 Requirements to Corner Measure

- Define a corner measure $R(x, y)$ for a region of interest (ROI)
- Move the ROI over the image and test if the corner measure exceeds a threshold T
- The corner measure must respond in the following way to the three situations:
 1. homogeneous region → measure < T
 2. region with an edge → measure < T
 3. region with a corner → measure > T



7.1.3 Point cloud, weighted gradients

Choosing a ROI with hard transitions is usually bad, as it would result in a corner measure with a flat maxima, instead of a unique maximum.

A smooth transition is better:

$$h(x_0 - x, y_0 - t)$$

$$h(x, y) = e^{-(x^2 - y^2)/4\sigma^2}$$

We define the gradient vector

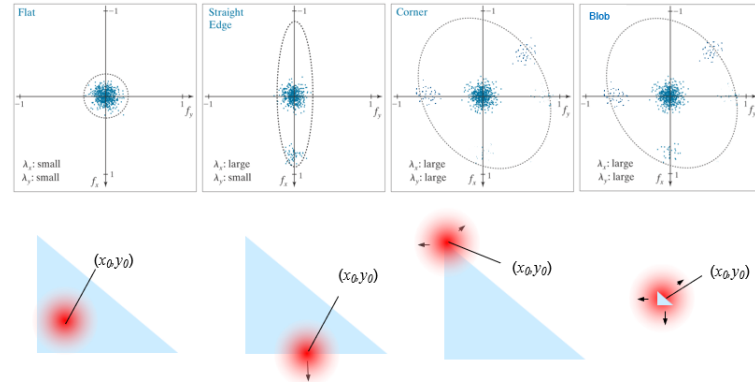
$$\mathbf{g}(x, y) := \begin{bmatrix} f_x(x, y) \\ f_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

To determine if the coordinates (x, y) belong to a corner, we consider the gradients weighted with h , which is Gaussian

$$\mathbf{v}(x, y, s, t) = h(x - s, y - t) \cdot \mathbf{g}(s, t)$$

Depending on the gradients that are considered by the Gaussian weighing kernel, the pointcloud has a different distribution and eigenvalues.

To identify the eigenvalues and eigenvectors, the centroid does not need to be removed first, unlike in principal component analysis



7.1.4 Covariance matrix of pointcloud

$$\begin{aligned} \mathbf{M}(x, y) &= \sum_{s, t} \mathbf{v}(x, y, s, t) \mathbf{v}^T(x, y, s, t) \\ &= \sum_{s, t} h(x - s, y - t) \mathbf{g}(s, t) h(x - s, y - t) \mathbf{g}^T(s, t) \\ &= \sum_{s, t} h(x - s, y - t)^2 \underbrace{\mathbf{g}(s, t) \mathbf{g}^T(s, t)}_{\mathbf{A}(s, t)} \\ &= h^2(x, y) \star \mathbf{A}(x, y) \end{aligned}$$

$$\mathbf{M}(x, y) := w(x, y) \star \mathbf{A}(x, y) = w \star \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

$$\text{with } w(x, y) = h^2(x, y) = e^{-(x^2 - y^2)/2\sigma^2}$$

7.1.5 Corner detection

As shown before, the eigenvalues could be used to determine corners:

$$\lambda_1 \geq \lambda_2 \geq 0$$

Case	Eigenvalue comparison	Interpretation
Flat region	$\lambda_1 \approx 0, \lambda_2 \approx 0$	No significant intensity change in any dir.
Edge	$\lambda_1 \gg 0, \lambda_2 \approx 0$	Strong change in one direction only.
Corner	$\lambda_1 \gg 0, \lambda_2 \gg 0$	Strong change in at least two indep. dir.

But Eigenvalues are expensive to compute.

Instead the following can be done:

$$\begin{aligned} R(x, y) &:= \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \\ &= \det(\mathbf{M}(x, y)) - k \text{Trace}^2(\mathbf{M}(x, y)) \end{aligned}$$

A corner is detected if $R(x, y)$ is larger than some threshold T .

The parameter k can be in the interval $(0, 0.25]$, while a good choice is $k = 0.06$. k helps dampening the influence of an overly big λ_1 to give a false positive, like an edge or an overly blunt corner.

[Harris and Stephens (1988)]

7.1.6 Improvements

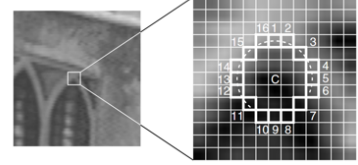
To find more corners for an image, create versions of the same image but with decreasing size.

7.2 FAST corner detection

(Features from Accelerated Segments Test)

A feature is detected at the center C if

- circle of 16 pixels surrounding C
- contains 12 contiguous pixels that are below or above a threshold intensity t of the center pixel



+ very fast

+ low memory requirements

– less discriminative than Harris corner detector

8 Feature description

8.1 General

A feature describes the image patch around a keypoint

Features are made invariant to

- Scaling
- Rotation
- Intensity (illumination, brightness)
- Moderate perspective change (warping)

Many different feature descriptors have been developed: SIFT, SURF, BRIEF, BRISK, ...

8.1.1 Scale

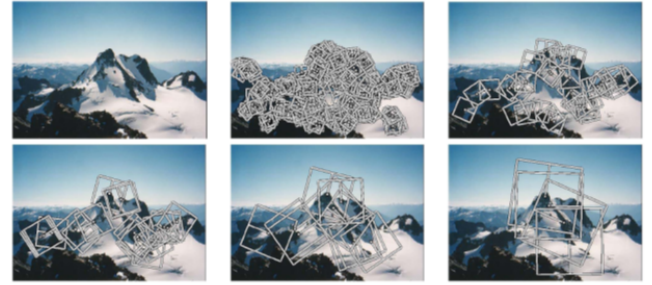
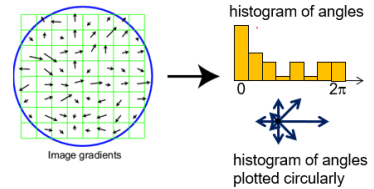


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matter images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

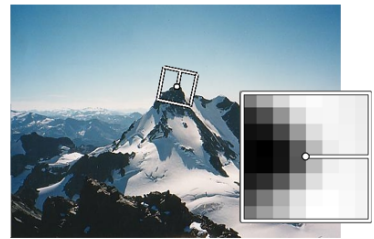
8.1.2 Rotation

- Take square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge



8.1.3 Intensity normalization

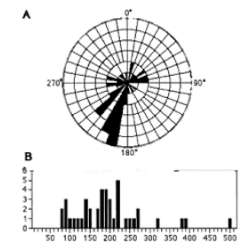
Take square window around detected feature and perform intensity normalization (explained further down). Specifically, the so called histogram of gradients will be normalized.



8.1.4 Histogram of Gradients (HoG)

N orientation bins. The gradients angle determines the bin to which the gradients accumulates

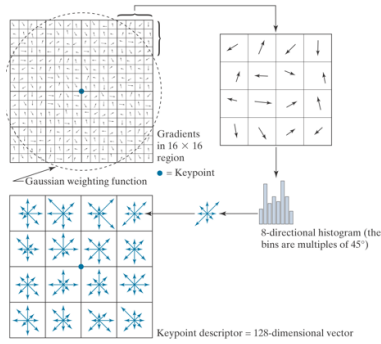
Histogram of Gradients



8.2 SIFT features

(Scale Invariant Feature Transform)

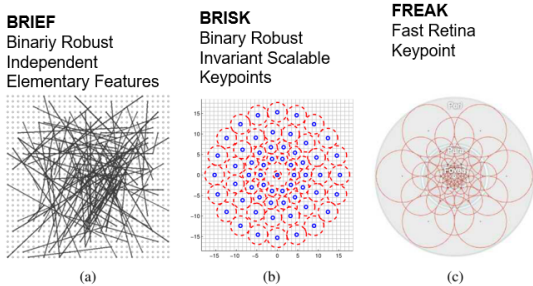
- Associate an orientation to each detected keypoint
- A square patch with the associated orientation is considered around the detected keypoint
- The square patch has 16x16 pixels
- A HoG is computed for subsquares of 4x4 pixels
- Gradient values are assigned to 8 orientation bins. Bins have an 8-bit counter
- 16 histograms per keypoint with 8 bins
→ 128 entries = 128 bytes per feature vector



8.3 Other descriptors

8.3.1 Different approaches and algorithms

- SURF (Speeded Up Robust Features) [Bay, Van Gool, 2015]
- BRIEF (Binary Robust Independent Elementary Features) [Calonder, Lepetit, 2012]
- BRISK (Binary Robust Invariant Scalable Keypoints) [Leutenegger, Siegwart, 2011]
- FAST (Features from Accelerated Segment Test) [Rosten, 2005]
- SIFT (Scale Invariant Feature Transform) [Lowe, 2004]



8.3.2 Trained descriptors

- LIFT (Learnt Invariant Feature Transform) [Moo, 2016]
- Implicit in CNNs (Convolutional Neuronal Network)
- Super Glue, light glue, disc,

8.3.3 Selection criteria

- Performance for the given image material
- Computational time
- Memory requirement
- Availability (Library, Ease of Implementation)

9 Feature matching

An object is described by a set of local features. Feature matching means finding a reference image within a query image.

9.1 Matching metrics

Denote the feature vectors from the **reference** image with $\mathbf{v}_j$ $j \in [0, M-1]$

Denote the feature vectors from the **query** image with $\mathbf{w}_i$ $i \in [0, N-1]$

Euclidean distance between features $d_{i,j} = \|\mathbf{v}_j - \mathbf{w}_i\|_2$

The smaller the Euclidean distance, the better the match

9.2 Brute Force Matching

The brute force matcher yields the match. I.e., per reference feature there will be one match to a query feature. A query feature can be matched by zero, one or more reference features. Only if the distance $d_{i,j}$ is below a threshold Θ , the match is accepted

True positives = # of correct matches that exceed the threshold Θ

False positives = # of false matches that exceed the threshold Θ
False positive matches are called "outliers"

N_e is the number of Euclidean distance evaluations, N_q is the number of features in the query image, and N_r is the number of features in the reference image.

$$N_e = N_q \cdot N_r$$

If the full distance matrix is computed, then N_e is zero, since crossmatching can be read from the distance matrix without additional computations.

$$N_e = 0$$

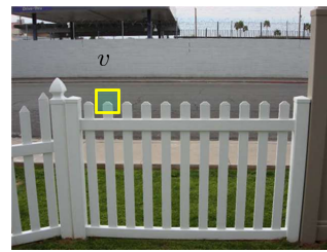
9.3 Outlier rejection

The cause of false positives or outliers is that feature descriptors can be similar even though they describe unrelated regions

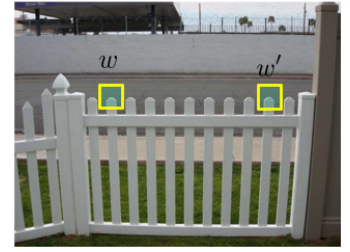
Human made objects often show multiple features that are almost identical, see the images below

Hence, making a feature descriptor more precise cannot eliminate feature ambiguity. Thus, ambiguous features should be detected and sorted out.

Reference image



Query image



9.3.1 Ratio test

Assuming the feature v matches roughly the same with w and w' .

The ratio test searches the best match w and the second-best match w' and evaluates the

$$\text{ratio } r := \frac{\|\mathbf{v} - \mathbf{w}\|_2}{\|\mathbf{v} - \mathbf{w}'\|_2} = \frac{d_{\text{best}}}{d_{\text{second-best}}} \leq \theta \leq 1$$

Ratios close to 1 or bigger than threshold θ will be identified as unreliable

9.3.2 Cross check

If the feature v matches best with w , check the opposite as well.

If yes, the match is reliable, otherwise it is ambiguous and gets rejected

9.3.3 Combination

The cross check and the ratio test are redundant to a certain extent. It has been shown empirically that a consecutive application of both minimizes the false positive rate.

9.4 Efficient matching

Brute Force Matching is expensive (M queries and N reference features, with matrices of both, optimal result but expensive)

- Replace the optimal or brute force KNN matcher by an approximate nearest neighbor algorithm (FLANN)
- Different approaches exist for approximate nearest neighborhood algorithms
- They do not provide the optimal match in all cases but with high probability

9.4.1 Minimum Distance Classifier

Each class ω_i is represented by a mean vector m_i . A feature vector x is assigned to the class with the smallest Euclidean distance:

$$D_i(x) = \|x - m_i\|$$

$$\hat{i} = \arg \min_i D_i(x)$$

Equivalently, one can use the discriminant function

$$d_i(x) = x^T m_i - \frac{1}{2} m_i^T m_i$$

and classify by

$$\hat{i} = \arg \max_i d_i(x).$$

The decision boundary between two classes ω_i and ω_j is given by

$$D_i(x) = D_j(x)$$

or equivalently

$$d_i(x) = d_j(x).$$

Geometrically, this boundary is the perpendicular bisector of the line segment between the two class means m_i and m_j .

10 Optical Flow

Optical flow = apparent motion of brightness patterns (ideally the projection of 3D motion into the 2D image plane); *dense* = estimated per pixel.

10.1 Brightness constancy & the flow constraint

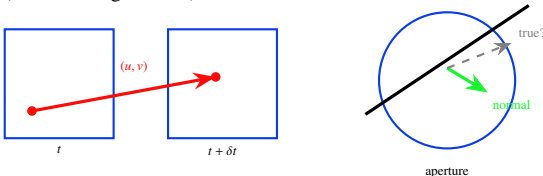
Assumptions: (1) brightness of a point is constant, (2) small motion. $I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$. Taylor expand $\Rightarrow$

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \iff I_x u + I_y v + I_t = 0.$$

(u, v) = optical flow; I_x, I_y = spatial gradients (Sobel); I_t = temporal difference of frames. One equation, two unknowns.

10.2 Aperture problem

Local motion is ambiguous: only the flow component *normal* to an edge is observable (1 DOF along the line).



10.3 Lucas–Kanade

Spatial coherency: assume the same flow for all pixels in a patch (e.g. $5 \times 5 = 25$ pixels)
 $\Rightarrow$ over-determined linear system

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_B \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}.$$

Least squares: $B^T B(u, v)^T = -B^T(I_t)$; solvable iff $A = B^T B$ has two nonzero eigenvalues (gradient point cloud spans 2D — a corner). **Coarse-to-fine pyramids** handle large displacements (the small-motion assumption holds at coarse levels).

III Image formation

11 3D to 2D projection

11.1 General

• Camera with lens

A camera with adjusted optical lens captures the light falling onto its surface and guides it to a specific location on the image sensor. All captured light rays originating from the same object point fall onto the same specific location on the sensor.

• Pinhole camera

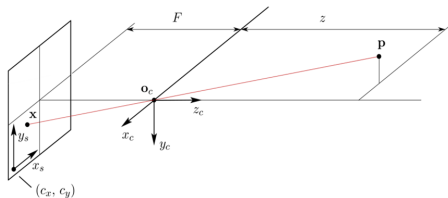
A pinhole camera just captures the light falling into the small pinhole

• Ideal lens-camera

An ideal lens camera does the same mapping of the 3D world to the 2D sensor as a pinhole camera

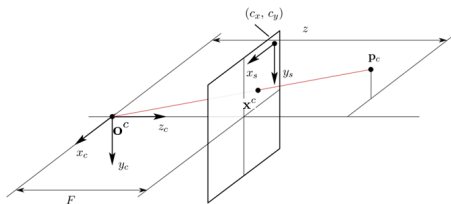
11.1.1 Pinhole-camera model

- Camera origin is the position of the pinhole o^c
- Pinhole to sensor distance F
- Position of the object point $\mathbf{p}^c$
- Position of the image point $\mathbf{x}^c$
- Camera coordinate-frame



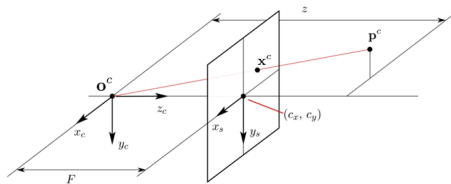
11.1.2 Inverted pinhole-camera model

We can think of getting the same image with the sensor in front of the pinhole by assuming that only those rays hit the origin that pass through the pinhole



11.1.3 Centered and inverted pinhole-camera model

Sometimes it is convenient to use a translated sensor coordinate frame, so that the center is on the z -axis



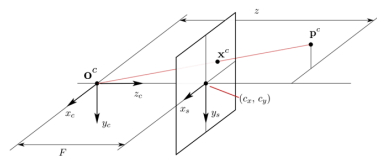
11.2 Perspective projection

We use the centered and inverted pinhole camera model and set $F = 1$

The ray from the object point $\mathbf{p}^c$ to the camera origin passes through point $\mathbf{x}^c$ on the sensor

The intercept theorem yields the following:

- The object point $\mathbf{p}^c$ gets projected onto $\mathbf{x}^c$
- Notice that $\mathbf{x}^c$ only specifies the direction of the ray, but not the position of the object point itself
- The image coordinates $\mathbf{x}_s$ is the projection of the point $\mathbf{x}^c$ or equivalently of the point $\mathbf{p}^c$ to the camera



$$\mathbf{x}^c = \frac{F}{z} \mathbf{p}^c = \frac{1}{z} \mathbf{p}^c = \frac{1}{z} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x/z \\ y/z \\ 1 \end{bmatrix}$$

$$\mathbf{x}_s := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{x}^c = \frac{1}{z} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{p}^c$$

11.3 Homogeneous image coordinates

Corresponding to the image coordinates $\mathbf{x}_s$, let us define the so-called homogeneous image coordinates

$$\tilde{\mathbf{x}}_s = \begin{bmatrix} \tilde{x}_{s,x} \\ \tilde{x}_{s,y} \\ \tilde{x}_{s,z} \end{bmatrix} = \begin{bmatrix} x_s \cdot z \\ y_s \cdot z \\ z \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} \cdot z$$

From these we can compute the image coordinates or more specifically, the inhomogeneous image coordinates

$$\tilde{\mathbf{x}}_s \mapsto \mathbf{x}_s = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \frac{1}{\tilde{x}_{s,z}} \begin{bmatrix} \tilde{x}_{s,x} \\ \tilde{x}_{s,y} \end{bmatrix}$$

Notice that homogeneous coordinates indicate the same image point no matter what their scaling is. We indicate this equivalence by writing

$$\tilde{\mathbf{x}}_s = \lambda \tilde{\mathbf{x}}_s, \quad \lambda \in \mathbb{R}/0$$

In general, homogeneous coordinates are obtained by adding a dimension with unit length and by applying an arbitrary scaling

$$\mathbf{x}_s \mapsto \tilde{\mathbf{x}}_s = \lambda \begin{bmatrix} \mathbf{x}_s \\ 1 \end{bmatrix}$$

Inserting the definitions from before lets us interpret homogeneous coordinates as a 3D vector representing a ray direction

$$\tilde{\mathbf{x}}_s = \mathbf{p}^c = \lambda \mathbf{p}^c$$

11.4 Lines and points

A point in an image is usually given in inhomogeneous coordinates as

$$\mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix}.$$

For computations in projective geometry, this point is written in homogeneous coordinates as

$$\tilde{\mathbf{p}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

The advantage of homogeneous coordinates is that geometric operations such as computing lines and intersections can be expressed conveniently using cross products.

11.4.1 Representation of a Line

A line in the image plane can be written as

$$ax + by + c = 0.$$

In homogeneous form, the line is represented by the vector

$$\tilde{\mathbf{l}} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

A homogeneous point

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

lies on the line if

$$\tilde{\mathbf{l}}^T \tilde{\mathbf{p}} = 0.$$

This gives

$$\begin{pmatrix} a & b & c \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = ax + by + c = 0.$$

11.4.2 Computing a Line from Two Points

If two points lie on the same line, the line can be computed using the cross product of the two homogeneous point vectors:

$$\tilde{\mathbf{l}} = \tilde{\mathbf{p}}_1 \times \tilde{\mathbf{p}}_2.$$

Similarly, the intersection point of two lines can be computed by taking the cross product of the two line vectors:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2.$$

The resulting vector

$$\tilde{\mathbf{x}} = \begin{pmatrix} x_h \\ y_h \\ w_h \end{pmatrix}$$

is again a homogeneous point. To obtain the corresponding inhomogeneous coordinates, one divides by the third component:

$$x = \frac{x_h}{w_h}, \quad y = \frac{y_h}{w_h}.$$

Parallel Lines

If two lines are parallel, their cross product will yield:

$$\vec{l}_1 \times \vec{l}_2 = \mathbf{0}.$$

11.4.3 Loss of scale information

A homogeneous vector is invariant to scaling. Hence, $\vec{p}^c$ can be reconstructed from $\vec{x}_s$ only up to scale.

A more intuitive explanation is, that the image only depends on the direction of the light rays falling into the camera and not on the distance of their origin.

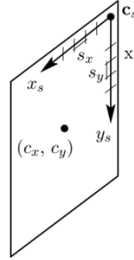
Loss of scale information is an intrinsic property of the pinhole camera model.

11.5 Camera intrinsics

- Focal length F in m
- Pixel spacings s_x, s_y in m
- Sensor origin coordinates c_x, c_y in pixels

Assumptions:

Rows and columns of the sensor are perfectly equidistant and orthogonal



Metric point on sensor plane

$$\vec{x}_s'' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \frac{F}{z} \vec{p}^c$$

Pixel coordinates on sensor plane

$$\vec{x}_s' = \begin{bmatrix} \frac{1}{s_x} & 0 \\ 0 & \frac{1}{s_y} \end{bmatrix} \cdot \vec{x}_s''$$

Pixel coordinates on sensor plane, non-cent.

$$\vec{x}_s = \vec{x}_s' + \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

Camera mapping from 3D to pixel on image

($f_x = F/s_x$ and $f_y = F/s_y$)

$$\vec{x}_s = \frac{1}{z} \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \end{bmatrix} \cdot \vec{p}^c + \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

Expanded to homogeneous coordinates

$$\tilde{\vec{x}}_s = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \cdot \vec{p}^c = \mathbf{K} \cdot \vec{p}^c$$

12 3D coordinate transforms

12.1 Coordinate frame

Coordinate frame:

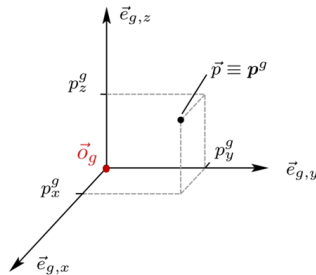
- Basis: $\vec{E}_g := [\vec{e}_{g,x}, \vec{e}_{g,y}, \vec{e}_{g,z}]$
- Origin: $\vec{o}_g$
- Coordinate frame: $\vec{E}_g; \vec{o}_g$

Point:

$$\vec{p} = \vec{e}_{g,x} \cdot p_x^g + \vec{e}_{g,y} \cdot p_y^g + \vec{e}_{g,z} \cdot p_z^g + \vec{o}_g$$

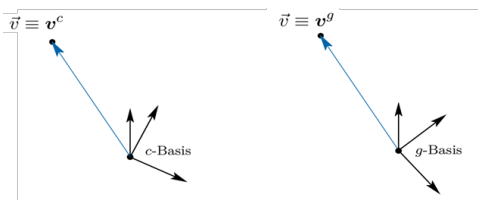
$$\vec{p} = \vec{E}_g \cdot \vec{p}^g + \vec{o}_g$$

$$\vec{p}^g = \begin{bmatrix} p_x^g \\ p_y^g \\ p_z^g \end{bmatrix} \quad (\text{referred frame denoted as superscript } g)$$



12.1.1 Changing the basis of a point

Rotation is equivalent to changing the basis that is used to represent a point



Recalling from before we have

$$\vec{v}^c = \vec{E}_c^T \cdot \vec{v} \quad \vec{v}^g = \vec{E}_g \cdot \vec{v}^g$$

The matrix denoted as $\mathbf{R}_g^c$ describes the intended rotation and is thus a rotation matrix

12.2 Rotation matrix

$$\mathbf{R}_g^c = \begin{bmatrix} \vec{e}_{c,x} \\ \vec{e}_{c,y} \\ \vec{e}_{c,z} \end{bmatrix} \cdot \underbrace{\begin{bmatrix} \vec{e}_{c,x} & \vec{e}_{c,y} & \vec{e}_{c,z} \end{bmatrix}}_{\vec{E}_g} = \begin{bmatrix} \vec{e}_{c,x} \vec{e}_{g,x} & \vec{e}_{c,y} \vec{e}_{g,x} & \vec{e}_{c,z} \vec{e}_{g,x} \\ \vec{e}_{c,x} \vec{e}_{g,y} & \vec{e}_{c,y} \vec{e}_{g,y} & \vec{e}_{c,z} \vec{e}_{g,y} \\ \vec{e}_{c,x} \vec{e}_{g,z} & \vec{e}_{c,y} \vec{e}_{g,z} & \vec{e}_{c,z} \vec{e}_{g,z} \end{bmatrix} = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix}$$

Other rules:

$$\mathbf{R}_g^c = [\mathbf{e}_{g,x}^c, \mathbf{e}_{g,y}^c, \mathbf{e}_{g,z}^c] = \begin{bmatrix} (\mathbf{e}_{c,x}^g)^T \\ (\mathbf{e}_{c,y}^g)^T \\ (\mathbf{e}_{c,z}^g)^T \end{bmatrix}, \quad (\mathbf{R}_g^c)^T \mathbf{R}_g^c = \mathbf{I}, \quad \mathbf{R}_g^c = (\mathbf{R}_c^g)^T$$

$\mathbf{e}_{g,x}^c$ = Where does the x -axis of g point, written in c ?

$\mathbf{e}_{g,y}^c$ = Where does the y -axis of g point, written in c ?

$\mathbf{e}_{g,z}^c$ = Where does the z -axis of g point, written in c ?

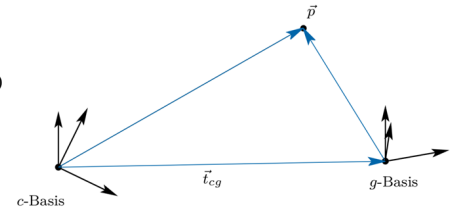
12.3 Translation

Translation between two origins:

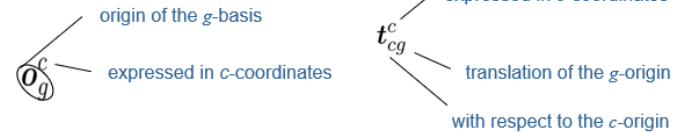
$$\vec{t}_{cg}^c = \vec{o}_g - \vec{o}_c \quad (\text{General})$$

$$\vec{t}_{cg}^c = \mathbf{o}_g^c - \mathbf{o}_c^c \quad (\text{Expr. in } c\text{-coord.})$$

Order of subscripts matters!



12.3.1 Notation



12.3.2 Rotation plus translation

With $\vec{p}^g$ given, $\vec{p}^c = \mathbf{R}_g^c \cdot \vec{p}^g + \vec{t}_{cg}^c$

$$\vec{p}^c = \mathbf{R}_g^c \cdot (\vec{p}^g + \mathbf{R}_c^g \cdot \vec{t}_{cg}^c)$$

$$= \mathbf{R}_g^c \cdot (\vec{p}^g + \vec{t}_{cg}^g)$$

$$= \mathbf{R}_g^c \cdot (\vec{p}^g - \vec{t}_{gc}^g)$$

12.4 Rigid motion matrix

Usage:

- for transformation of position vectors from one coordinate frame to another
- for representation of the relative pose between two coordinate frames
- allows very simple inversion of coordinate change

$$\vec{p}^c = \mathbf{R}_g^c \cdot \vec{p}^g + \vec{t}_{cg}^c$$

$$\begin{bmatrix} \vec{p}^c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_g^c & \vec{t}_{cg}^c \\ [0, 0, 0] & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{p}^g \\ 1 \end{bmatrix}$$

$$\vec{p}^c = \mathbf{T}_g^c \cdot \vec{p}^g \quad \vec{p}^g = [\mathbf{T}_c^g]^{-1} \cdot \vec{p}^c$$

$$\mathbf{T}_c^g = (\mathbf{T}_g^c)^{-1} = \begin{bmatrix} \mathbf{R}_c^g & \vec{t}_{cg}^c \\ [0, 0, 0] & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{R}_c^g & -\mathbf{R}_c^g \vec{t}_{cg}^c \\ [0, 0, 0] & 1 \end{bmatrix}$$

13 3D rotations

13.1 Euler angles

A rotation matrix can be factorized into rotations around three cardinal axes e.g. x, y, z by angles φ, θ and ψ

When the angle of the inner matrix approaches 90° , the rotation axis of the outer two rotation matrices become aligned. Thus, the product of the three rotation matrices suffers a reduction in degrees of freedom: Instead of three it has only one degree of freedom. This effect is called “gimbal lock”

To prevent angles near 90° of the inner factor, it is often possible to sort the three factors appropriately.

To completely avoid gimbal lock, other parameterizations for rotation matrices must be used.

$$\mathbf{R}(\varphi, \theta, \psi) = \mathbf{R}(\varphi) \cdot \mathbf{R}(\theta) \cdot \mathbf{R}(\psi)$$

$$\mathbf{R}_\varphi(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix}$$

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

$$\mathbf{R}_\psi(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

13.2 Axis/angle Rodrigues' formula

The image shows a vector $\mathbf{v}$ getting rotated around the axis $\hat{\mathbf{n}}$ by the angle θ and yielding the rotated vector $\mathbf{u}$

$$\mathbf{v}_\parallel = \hat{\mathbf{n}}(\hat{\mathbf{n}} \cdot \mathbf{v}) = (\hat{\mathbf{n}}\hat{\mathbf{n}}^T) \mathbf{v}$$

$$\mathbf{v}_\perp = \mathbf{v} - \mathbf{v}_\parallel = (\mathbf{I} - \hat{\mathbf{n}}\hat{\mathbf{n}}^T) \mathbf{v}$$

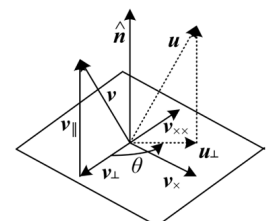
$$\mathbf{v}_\times = \hat{\mathbf{n}} \times \mathbf{v}_\perp = \hat{\mathbf{n}} \times \mathbf{v} = [\hat{\mathbf{n}}]_\times \mathbf{v}$$

$$\mathbf{v}_{\times \times} = \hat{\mathbf{n}} \times \mathbf{v}_\times = [\hat{\mathbf{n}}]_\times^2 \mathbf{v} = -\mathbf{v}_\perp$$

$$\mathbf{v}_\parallel = \mathbf{v} - \mathbf{v}_\perp = \mathbf{v} + \mathbf{v}_{\times \times} = (\mathbf{I} + [\hat{\mathbf{n}}]_\times^2) \mathbf{v}$$

$$\mathbf{u}_\perp = \cos \theta \mathbf{v}_\perp + \sin \theta \mathbf{v}_\times = (\sin \theta [\hat{\mathbf{n}}]_\times - \cos \theta [\hat{\mathbf{n}}]_\times^2) \mathbf{v}$$

$$\mathbf{u} = \mathbf{u}_\perp + \mathbf{v}_\parallel = (\mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_\times + (1 - \cos \theta) [\hat{\mathbf{n}}]_\times^2) \mathbf{v}$$



Rotation matrix with Rodrigues' formula:

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

13.2.1 Rotation vector

The Euler theorem states that any rotation in 3D can be expressed as a rotation around a constant axis in space around a specific angle.

We describe the rotation axis by a vector $\hat{\mathbf{n}}$ and the rotation angle by the scalar θ . Taken both together we obtain the so-called axis angle representation for a 3D rotation

Multiplying the two we get the so-called rotation vector $\boxed{\boldsymbol{\omega} := \theta \cdot \hat{\mathbf{n}}}$

The rotation vector is a minimal parametrization of the three degrees of freedom of any 3D rotation The term minimal representation means that there exists no representation with viewer parameters.

Unfortunately, this parametrization for 3D rotations is not unique. E.g., adding a multiple of 360° to any θ is equivalent to θ .

For small angles θ , Rodrigues' formula yields

$$\mathbf{R}(\boldsymbol{\omega}) \approx \mathbf{I} + \sin \theta [\hat{\mathbf{n}}]_{\times} \approx \mathbf{I} + \theta [\hat{\mathbf{n}}]_{\times} = \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix}$$

13.3 Interpreting Rodrigues' rotation matrix

Consider the two bases $\mathbf{R}_g^c = [\mathbf{e}_{g,x}^c, \mathbf{e}_{g,y}^c, \mathbf{e}_{g,z}^c]$ and $\mathbf{R}_c^c$.

Assume, rotating the c-basis so that it becomes the g-basis, is achieved by rotating it with the rotation vector $\omega_{c,g}^c = \hat{\mathbf{n}}_{c,g}^c \theta_{c,g}$

This should allow us to write $\mathbf{R}_g^c = \mathbf{R}(\omega_{c,g}^c) \cdot \mathbf{R}_c^c \Rightarrow \boxed{\mathbf{R}_g^c = \mathbf{R}(\omega_{c,g}^c)}$

This equation allows us to interpret the meaning of a rotation specified by a rotation vector

13.4 Notation for software code

Formula	Identifier	Description
$\vec{o}_j$		Origin of coordinate system j
$\vec{R}_j$		Basis of coordinate system j
$(\vec{R}_j, \vec{o}_j)$		Coordinate system j
$\vec{v}_j$		Direction vector or space vector j
$\vec{x}_j$		Location vector or point in space j
$\mathbf{R}_j^i$	R_i_j	Basis of coordinate system j expressed in coordinates of coordinate system i , $\mathbb{R}^{3 \times 3}$
$\mathbf{o}_j^i$	o_i_j	Origin of coordinate system j expressed in i -coordinates
$\mathbf{t}_{ij}^i = \mathbf{o}_j^i - \mathbf{o}_i^i$	t_i_j_j	Direction vector of the translation of coordinate system j relative to the coordinate system i expressed in i coordinates, given that $\mathbf{o}_i^i = 0$
$\mathbf{p}^i$	p_i	Point coordinates of point in space $\vec{p}$ expressed in inhomogeneous i -coordinates, $\mathbb{R}^{3 \times 1}$
$\tilde{\mathbf{p}}^i$	xh_i	Point coordinates of point in space $\vec{p}$ expressed in homogeneous i -coordinates, $\mathbb{R}^{4 \times 1}$
$\mathbf{v}^i = \mathbf{R}_j^i \mathbf{v}^j$		Coordinate transformation for direction vectors $\mathbf{v}^i, \mathbf{v}^j \in \mathbb{R}^{3 \times 1}$
$\mathbf{T}_j^i = \begin{bmatrix} \mathbf{R}_j^i & \mathbf{t}_{ij}^i \\ 0 & 1 \end{bmatrix}$	T_i_j_j	Pose matrix for the representation of the relation from coordinate system j to coordinate system i , $\mathbb{R}^{4 \times 4}$
$\underbrace{\begin{bmatrix} \mathbf{p}^i \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}^i} = \mathbf{T}_j^i \underbrace{\begin{bmatrix} \mathbf{p}^j \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}^j}$		Coordinate transformation for location vectors in homogeneous coordinates $\tilde{\mathbf{p}}^i, \tilde{\mathbf{p}}^j \in \mathbb{R}^{4 \times 1}$
$\mathbf{b}^i$	b_i	Image coordinates in inhomogeneous i -coordinates, $\mathbb{R}^{2 \times 1}$
$\tilde{\mathbf{b}}^i$	bh_i	Image coordinates in homogeneous i -coordinates, $\mathbb{R}^{3 \times 1}$
$\mathbf{K}_i$	K__i	Intrinsic matrix of the camera i , $\mathbb{R}^{3 \times 3}$
$\tilde{\mathbf{b}}^i = \mathbf{K}_i \mathbf{p}^i$		Mapping of location vector $\mathbf{p}^i$ to homogeneous image coordinates of the camera i
$\mathbf{b}^i = \begin{bmatrix} \tilde{b}_x^i / \tilde{b}_z^i \\ \tilde{b}_y^i / \tilde{b}_z^i \end{bmatrix}$		Mapping of homogeneous image coordinates of the camera i to inhomogeneous image coordinates

14 Camera matrix

14.1 Projecting points

We want to project points specified in any coordinate frame to a camera sensor

$$\tilde{\mathbf{x}}_s = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \cdot \mathbf{p}^c$$

For this we transform a point specified in some local (l)-coordinates to camera (c)-coordinates

$$\underbrace{\begin{bmatrix} \mathbf{p}^c \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}^c} = \underbrace{\begin{bmatrix} \mathbf{R}_l^c \\ [0, 0, 0] \end{bmatrix}}_{\mathbf{T}_l^c} \cdot \underbrace{\begin{bmatrix} \mathbf{t}_{cl}^c \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}^l}$$

The simple form is

$$\underbrace{\begin{bmatrix} x_s z \\ y_s z \\ z \end{bmatrix}}_{\tilde{\mathbf{x}}_s} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{P}_l^c} \cdot \underbrace{\begin{bmatrix} \mathbf{R}_l^c & \mathbf{t}_{cl}^c \end{bmatrix}}_{\mathbf{P}_l^c} \cdot \underbrace{\begin{bmatrix} \mathbf{p}^l \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}^l}$$

$$\tilde{\mathbf{x}}_s = \mathbf{P}_l^c \cdot \tilde{\mathbf{p}}^l \quad \text{with} \quad \mathbf{P}_l^c \in \mathbb{R}^{3 \times 4}$$

Finally homogenous coordinates must be mapped into inhomogeneous coordinates so that we obtain pixel coordinates

$$\tilde{\mathbf{x}}_s \mapsto \mathbf{x}_s = \begin{bmatrix} x_s \\ y_s \end{bmatrix} = \begin{bmatrix} \tilde{x}_{s,x} / \tilde{x}_{s,z} \\ \tilde{y}_{s,y} / \tilde{y}_{s,z} \end{bmatrix}$$

IV Projective geometry

15 Geometric definitions

15.1 Homogeneous coordinates

$\tilde{\mathbf{x}}$ is the homogeneous image coordinate of the projected point. For a point $\mathbf{p}^c$ lying on the corresponding camera ray, the vector $\mathbf{p}^c$ is proportional to the ray direction, since the ray passes through the camera centre.

Conversely, the ray direction corresponding to an image point is given by:

$$\tilde{\mathbf{x}} = \tilde{w} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \cdot \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\mathbf{p}^c}$$

$$\mathbf{r}^c \sim \mathbf{K}^{-1} \tilde{\mathbf{x}}$$

$$\mathbf{K}^{-1} = \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix}$$

15.2 2D lines

Implicit line $ax + by + c = \langle \tilde{\mathbf{x}}, \tilde{\mathbf{l}} \rangle = 0$, with $\tilde{\mathbf{l}} = [a, b, c]^\top = \lambda [\hat{\mathbf{n}}^\top, -d]^\top$, $\hat{\mathbf{n}}$ = normal vector of line, d = distance from origin.

15.2.1 Points on a line

Define the orthogonal vector

$$\hat{\mathbf{n}}_{\perp} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \hat{\mathbf{n}}, \quad \hat{\mathbf{n}} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

Compute points

$$\mathbf{x}_s = d \hat{\mathbf{n}} + a \hat{\mathbf{n}}_{\perp}$$

Reach any point on the line with

$$\alpha \in \mathbb{R}$$

15.3 Cross product

Intersection of two lines: $\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$; Line through two points: $\tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2$.

(With inhomogeneous coordinates you would have to solve a linear system.)

15.4 Points/line at infinity

Parallel lines $\tilde{\mathbf{l}}_1, \tilde{\mathbf{l}}_2$ have the same normal vector $\hat{\mathbf{n}}$ but different distances d_1, d_2

$$\tilde{\mathbf{l}}_1 = \begin{bmatrix} \hat{\mathbf{n}} \\ d_1 \end{bmatrix}, \quad \tilde{\mathbf{l}}_2 = \begin{bmatrix} \hat{\mathbf{n}} \\ d_2 \end{bmatrix}$$

Their point of intersection is

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2 = \begin{bmatrix} \hat{\mathbf{n}} \\ d_1 \end{bmatrix} \times \begin{bmatrix} \hat{\mathbf{n}} \\ d_2 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{n}}_{\perp} \\ 0 \end{bmatrix}$$

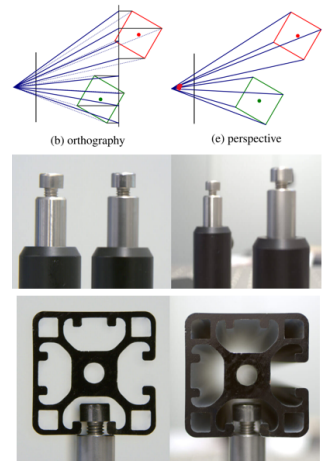
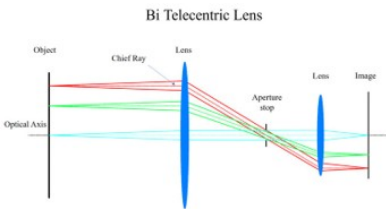
The intersection therefore lies perpendicular to the normal vector at an infinite distance (division by zero)

16 Orthography

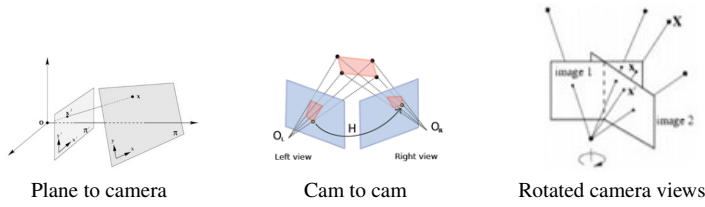
An orthography simply drops the z component of a three-dimensional coordinate p to obtain a 2D point x.

Orthographic projection, often leads to simpler computer vision algorithms than perspective projection.

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \mathbf{p}^c$$



17 Homography



17.1 Properties

- Up to 8 degrees of freedom
- Uniquely defined by 4 point correspondences
- Maps lines to lines
- Linear transformation in homogeneous image coordinates
- Invertible unless > 2 of the 4 correspondences lie on the same line

17.2 Mapping from a planar object

$$\underbrace{\begin{bmatrix} x_{s,i} \\ y_{s,i} \\ 1 \end{bmatrix}}_{\tilde{\mathbf{x}}_i^c} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}_c} \cdot \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\Pi_0} \cdot \underbrace{\begin{bmatrix} \mathbf{R}_i^c & \mathbf{t}_{cl}^c \\ 0 & 1 \end{bmatrix}}_{\mathbf{H}_i^c} \cdot \underbrace{\begin{bmatrix} \mathbf{p}_i^l \\ 1 \end{bmatrix}}_{\tilde{\mathbf{x}}_i^l}$$

Since it is planar, the third column of the rotation matrix becomes irrelevant.

$\tilde{\mathbf{x}}_i^c$ are the points i on the camera image

The factor λ can be chosen such that the lower right component equals 1

For known K : only **6 degrees of freedom** – 3 rotation + 3 translation; the parameters are constrained $\Rightarrow$ can recover calibration parameters (translation, rotation, focal length)

17.3 Mapping from one camera to another

As shown before, we can map a point on a plane to two separate cameras

Invert the first homography

Inserting it into the second we get

With the total homography matrix

$$\begin{aligned} \tilde{\mathbf{x}}^1 &= \lambda_1 \cdot \mathbf{H}_g^1 \cdot \tilde{\mathbf{p}}^g, & \tilde{\mathbf{x}}^2 &= \lambda_2 \cdot \mathbf{H}_g^2 \cdot \tilde{\mathbf{p}}^g \\ \tilde{\mathbf{p}}^g &= \frac{1}{\lambda_1} \cdot (\mathbf{H}_g^1)^{-1} \cdot \tilde{\mathbf{x}}^1 \\ \tilde{\mathbf{x}}^2 &= \lambda_2 \cdot \mathbf{H}_g^2 \cdot \frac{1}{\lambda_1} \cdot (\mathbf{H}_g^1)^{-1} \cdot \tilde{\mathbf{x}}^1 = \mathbf{H}_l^1 \cdot \tilde{\mathbf{x}}^1 \\ \mathbf{H}_l^1 &= \lambda_2 \cdot \mathbf{H}_g^2 \cdot \frac{1}{\lambda_1} \cdot (\mathbf{H}_g^1)^{-1} \end{aligned}$$

17.4 Mapping between rotated camera views

The camera rotates around the origin. We express the scene points in the frame of camera 1

With the image point the scene point can be recovered up to distance

Inserting this to the first image, we get

With the total homography matrix

$$\begin{aligned} \tilde{\mathbf{x}}_i^1 &= \mathbf{K}_1 \cdot \mathbf{p}_i^1, & \tilde{\mathbf{x}}_i^2 &= \mathbf{K}_2 \cdot \mathbf{R}_1^2 \cdot \mathbf{p}_i^1 \\ \mathbf{p}_i^1 &= \lambda \cdot \mathbf{R}_2^1 \cdot \mathbf{K}_2^{-1} \tilde{\mathbf{x}}_i^2 \\ \tilde{\mathbf{x}}_i^1 &= \lambda \cdot \mathbf{K}_1 \cdot \mathbf{R}_2^1 \cdot \mathbf{K}_2^{-1} \cdot \tilde{\mathbf{x}}_i^2 = \mathbf{H}_2^1 \cdot \tilde{\mathbf{x}}_i^2 \\ \mathbf{H}_2^1 &= \lambda \cdot \mathbf{K}_1 \cdot \mathbf{R}_2^1 \cdot \mathbf{K}_2^{-1} \end{aligned}$$

18 Singular Value Decomposition (SVD)

With **Eigenvalue Decomposition**, we can decompose any asymmetric matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$, $\mathbf{M} = \mathbf{M}^T$ according to

$$\mathbf{M} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T \text{ with } \mathbf{V} = [\mathbf{v}_1 \mathbf{v}_2 \dots \mathbf{v}_m], \mathbf{v}_i \in \mathbb{R}^{m \times 1}, i = 1, 2, \dots, m$$

and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_m)$, with real $\lambda_i \geq \lambda_{i+1} \geq 0$.

$$\mathbf{M} \mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad \mathbf{v}_1 = \arg \max \{ \mathbf{v}^T \mathbf{M} \mathbf{v} \}, \quad \text{and} \quad \lambda_1 = \max \{ \mathbf{v}^T \mathbf{M} \mathbf{v} \}$$

SVD states that any real-valued $m \times n$ matrix $\mathbf{A}$ can be written as

$$\mathbf{A}_{m \times n} = \mathbf{U}_{m \times p} \mathbf{\Sigma}_{p \times p} \mathbf{V}_{p \times n}^T = [\mathbf{u}_0 \dots \mathbf{u}_{p-1}] \begin{bmatrix} \sigma_0 & & \\ & \ddots & \\ & & \sigma_{p-1} \end{bmatrix} \begin{bmatrix} \mathbf{v}_0^T \\ \vdots \\ \mathbf{v}_{p-1}^T \end{bmatrix}, \quad p = \min(m, n)$$

$\mathbf{U}$ and $\mathbf{V}$ are orthonormal ($\mathbf{U}^T \mathbf{U} = \mathbf{I}$) and so are their column vectors ($\mathbf{u}_i \mathbf{u}_j^T = \mathbf{v}_i \mathbf{v}_j^T = \delta_{ij}$). The singular values are all non-negative and are ordered ($\sigma_0 \geq \sigma_1 \geq \dots \geq \sigma_{p-1} \geq 0$).

18.1 Proof of SVD

Let $\mathbf{M} := \mathbf{A}^T \mathbf{A}$ with $\mathbf{M} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$ and $\mathbf{A}$ being a $m \times n$ matrix with rank r .

Given $\mathbf{M} \mathbf{v}_i = \lambda_i \mathbf{v}_i$ we observe

where δ_{ij} is the Kronecker delta function. This returns $\mathbf{A} \mathbf{v}_i$ as orthogonal. Therefore the orthogonal vectors $\mathbf{u}_i$ can be defined

The scalar products are therefore

It follows that an orthonormal matrix

can be composed so that

Right multiplication with $\mathbf{V}^T$ yields

$$\begin{aligned} \mathbf{v}_j^T \mathbf{M} \mathbf{v}_i &= \mathbf{v}_j^T \mathbf{A}^T \mathbf{A} \mathbf{v}_i = \lambda_i \mathbf{v}_j^T \mathbf{v}_i = \lambda_i \delta_{i-j} \\ \mathbf{v}_j^T \mathbf{A}^T &= \sigma_j \mathbf{u}_j^T, \quad \mathbf{A} \mathbf{v}_i = \sigma_i \mathbf{u}_i \text{ with } \sigma_i^2 := \lambda_i \\ \sigma_i^2 \mathbf{u}_j^T \mathbf{u}_i &= \sigma_i^2 \delta_{i-j} \quad \text{nonzero if } i = j \\ \mathbf{U} &:= [\mathbf{u}_0 \dots \mathbf{u}_{p-1}], \quad p = \min(m, n) \\ \mathbf{A} \mathbf{V} &= \mathbf{U} \mathbf{\Sigma} \text{ with } \mathbf{\Sigma} := \text{diag}(\sigma_0, \dots, \sigma_{p-1}) \\ \mathbf{A} &= \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \end{aligned}$$

19 Homography estimation & Pose estimation

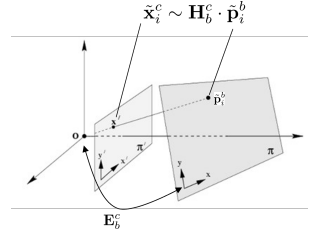
19.1 General

Inputs:

- point correspondences ($\tilde{\mathbf{x}}_i^c, \tilde{\mathbf{p}}_i^b$)
- If available, calibration matrix $\mathbf{K}_c$

To compute:

- If not available, calibration matrix $\mathbf{K}_c$
- Camera extrinsic matrix (rel. pose between object and camera) $\mathbf{E}_b^c$



19.2 Estimating H

- Matrix inversion** from exactly 4 correspondences
- Least squares** from > 4 correspondences (pseudo-inverse $\mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1}$)
- Direct Linear Transform (DLT)** from > 4 correspondences

19.2.1 Matrix inversion

Build equations according to:

$$\begin{bmatrix} \lambda x_{d,i} \\ \lambda y_{d,i} \\ \lambda \end{bmatrix} = \underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}}_{\mathbf{H}_u^d} \cdot \begin{bmatrix} x_{u,i} \\ y_{u,i} \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{d,1} \\ y_{d,1} \\ \vdots \\ x_{d,i} \\ y_{d,i} \\ \vdots \\ x_{d,A} \\ y_{d,A} \end{bmatrix} = \underbrace{\begin{bmatrix} x_{u,1} & y_{u,1} & 1 & 0 & 0 & 0 & -x_{u,1} \cdot x_{d,1} & -y_{u,1} \cdot x_{d,1} \\ 0 & 0 & 0 & x_{u,1} & y_{u,1} & 1 & -x_{u,1} \cdot y_{d,1} & -y_{u,1} \cdot y_{d,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{u,i} & y_{u,i} & 1 & 0 & 0 & 0 & -x_{u,i} \cdot x_{d,i} & -y_{u,i} \cdot x_{d,i} \\ 0 & 0 & 0 & x_{u,i} & y_{u,i} & 1 & -x_{u,i} \cdot y_{d,i} & -y_{u,i} \cdot y_{d,i} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{u,A} & y_{u,A} & 1 & 0 & 0 & 0 & -x_{u,A} \cdot x_{d,A} & -y_{u,A} \cdot x_{d,A} \\ 0 & 0 & 0 & x_{u,A} & y_{u,A} & 1 & -x_{u,A} \cdot y_{d,A} & -y_{u,A} \cdot y_{d,A} \end{bmatrix}}_{\mathbf{A}} \cdot \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}}_{\mathbf{g}}$$

$$x_d \cdot (g x_u + h y_u + 1) = a x_u + b y_u + c$$

$$y_d \cdot (g x_u + h y_u + 1) = a x_u + b y_u + c$$

Solving for θ provides the coefficients of the Homography

- $\mathbf{A}$ can only be inverted, if # unknowns = # equations
- Equations must be linearly independent

19.2.2 Least squares

Instead of solving the equation system exactly, the least squares solution can be computed e.g. by computing the pseudoinverse of $\mathbf{A}$.

- + Arbitrarily large number of point correspondences
- + Noise tolerant
- Numerical accuracy sometimes insufficient

19.2.3 Direct Linear Transform (DLT)

We use the following definition

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}}_{\tilde{\mathbf{x}}} = \underbrace{\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}}_{\mathbf{H}} \cdot \underbrace{\begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}}$$

(1) Applying the cross product with the left side vector yields

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}}_{\tilde{\mathbf{x}}} \times \underbrace{\begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix}}_{\mathbf{H}} \cdot \underbrace{\begin{bmatrix} p_1 \\ p_2 \\ 1 \end{bmatrix}}_{\tilde{\mathbf{p}}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{0}}$$

Matrix $\mathbf{H}$ expressed by row vectors

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \mathbf{0}^T & -\tilde{\mathbf{p}}^T & x_2 \tilde{\mathbf{p}}^T \\ \tilde{\mathbf{p}}^T & \mathbf{0}^T & -x_1 \tilde{\mathbf{p}}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix}$$

This gives the following

$$\mathbf{H}_u^d = \lambda \cdot \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

With SVD we can use the raw vectors to provide the coefficients of the homography matrix

- + Applicable to any homogeneous equation system
- + Arbitrary number of equations
- + Good numerical accuracy

19.3 Line mapping by homography

A point is mapped by a homography as

$$\tilde{\mathbf{x}}' \sim \mathbf{H} \tilde{\mathbf{x}}.$$

A line is represented by

$$\ell = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \ell^T \tilde{\mathbf{x}} = 0.$$

The corresponding mapped line is

$$\ell' \sim \mathbf{H}^{-T} \ell$$

because

$$\tilde{\mathbf{x}} \sim \mathbf{H}^{-1} \tilde{\mathbf{x}}'$$

and therefore

$$\ell^T \mathbf{H}^{-1} \tilde{\mathbf{x}}' = 0 \Rightarrow (\mathbf{H}^{-T} \ell)^T \tilde{\mathbf{x}}' = 0.$$

19.4 Direct Linear Transform (DLT) for Homography Estimation

A homography maps homogeneous image points according to

$$\tilde{\mathbf{x}}' \sim \mathbf{H}\tilde{\mathbf{x}}$$

with

$$\tilde{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \tilde{\mathbf{x}}' = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}.$$

The homography is only defined up to scale, therefore

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11}x + h_{12}y + h_{13} \\ h_{21}x + h_{22}y + h_{23} \\ h_{31}x + h_{32}y + h_{33} \end{bmatrix}.$$

Define

$$W = h_{31}x + h_{32}y + h_{33}.$$

Then the inhomogeneous image coordinates are

$$u = \frac{h_{11}x + h_{12}y + h_{13}}{W}, \quad v = \frac{h_{21}x + h_{22}y + h_{23}}{W}.$$

Multiplying by W gives two linear equations in the unknown homography coefficients:

$$uW - (h_{11}x + h_{12}y + h_{13}) = 0,$$

$$vW - (h_{21}x + h_{22}y + h_{23}) = 0.$$

With the unknown vector

$$\mathbf{h} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}^T,$$

one point correspondence $(x, y) \mapsto (u, v)$ gives the two DLT rows

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & ux & uy & u \end{bmatrix} \mathbf{h} = 0,$$

$$\begin{bmatrix} 0 & 0 & 0 & -x & -y & -1 & vx & vy & v \end{bmatrix} \mathbf{h} = 0.$$

For n point correspondences, all rows are stacked into a system

$$\mathbf{A}\mathbf{h} = 0.$$

Because $\mathbf{H}$ has 9 entries but is only defined up to scale, at least 4 point correspondences are required. With exactly 4 correspondences, the solution is exact if the equations are independent. With more than 4 correspondences, the system is solved in a least-squares sense, usually by SVD:

$$\mathbf{A}\mathbf{h} = 0, \quad \mathbf{h} = \text{right singular vector of } \mathbf{A} \text{ belonging to the smallest singular value.}$$

The vector $\mathbf{h}$ is then reshaped into the homography matrix $\mathbf{H}$.

Example: For the correspondence

$$(x, y) = (2, 1), \quad (u, v) = (5, 4),$$

the two DLT equations are

$$\begin{bmatrix} -2 & -1 & -1 & 0 & 0 & 0 & 10 & 5 & 5 \end{bmatrix} \mathbf{h} = 0,$$

$$\begin{bmatrix} 0 & 0 & 0 & -2 & -1 & -1 & 8 & 4 & 4 \end{bmatrix} \mathbf{h} = 0.$$

19.5 Pose estimation

Homography matrix proportional to camera matrix, orthonormal basic vectors and translation vector

$$\mathbf{H}_b \sim \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix}$$

Equation for centered image coordinates for $\tilde{\mathbf{x}}_{s,i} = \mathbf{H}_b^c \cdot \tilde{\mathbf{p}}_i^b$

$$\begin{bmatrix} x_{s,i} - c_x \\ y_{s,i} - c_y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \lambda \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_i^b \\ 1 \end{bmatrix}$$

Assuming the focal lengths are equal ($f_x = f_y = f$)

$$\begin{bmatrix} x_{s,i} - c_x \\ y_{s,i} - c_y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f \end{bmatrix}}_{\mathbf{H}_b^c} \cdot \lambda \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_i^b \\ 1 \end{bmatrix}$$

And then writing it with explicit components

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f \end{bmatrix} \cdot \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix}$$

Multiplying with the inverted diagonal matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f \end{bmatrix} \cdot \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \sim \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix}$$

Multiplying the left side we get

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ fh_{31} & fh_{32} & fh_{33} \end{bmatrix} \sim \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix}$$

Since the first two columns represent orthogonal vectors, we can stipulate

Giving us the equation for f

$$h_{11}h_{12} + h_{21}h_{22} + f^2h_{31}h_{32} = 0$$

$$f = \sqrt{-\frac{h_{11}h_{12} + h_{21}h_{22}}{h_{31}h_{32}}}$$

With the scaling vector λ we get

$$\frac{1}{\lambda} \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ fh_{31} & fh_{32} & fh_{33} \end{bmatrix} \sim \begin{bmatrix} \mathbf{r}_{b,x}^c, \mathbf{r}_{b,y}^c, \mathbf{t}_{cb}^c \end{bmatrix}$$

Unit length of first column is set by λ , for the second it is

$$\hat{\mathbf{r}}_{b,y}^c = \frac{\mathbf{r}_{b,y}^c}{\|\mathbf{r}_{b,y}^c\|}, \quad \mathbf{r}_{b,z}^c = \mathbf{r}_{b,x}^c \times \hat{\mathbf{r}}_{b,y}^c$$

This gives us the corresponding rigid motion or pose matrix

$$\mathbf{E}_b^c = \begin{bmatrix} \mathbf{R}_b^c & \mathbf{t}_{cb}^c \\ [0, 0, 0] & 1 \end{bmatrix} \text{ with } \mathbf{R}_b^c = \begin{bmatrix} \mathbf{r}_{b,x}^c, \hat{\mathbf{r}}_{b,y}^c, \mathbf{r}_{b,z}^c \end{bmatrix}$$

This method fails (sign of square root for f negative) when sensor is coplanar with object ($h_{31}h_{32} = 0$) or with outliers

19.5.1 Dolly zoom

Large distance + large $f \equiv$ small distance + small f .

Distance therefore can't be inferred from projected size alone (even with known physical size) unless f is known.

20 Camera Calibration

Intrinsic parameters describe the inside of the camera (K + distortion); **extrinsic** describe relative poses outside.

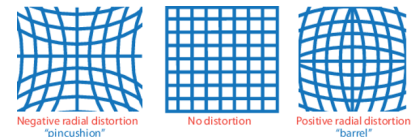
20.1 Image distortion (intrinsic calibration)

20.1.1 Radial distortion

Lenses are rotation-symmetric: points shift with radius r .

x_d, y_d Image coordinates in distorted image: center at (0,0)

x, y Image coordinates in undistorted image



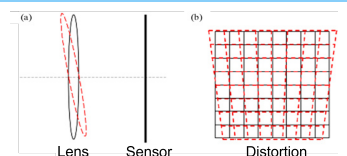
$$x = x_d \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad \text{Mapping for } x \quad r^2 = x_d^2 + y_d^2 \quad \text{Square of radial distance}$$

$$y = y_d \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad \text{Mapping for } y$$

20.2 Tangential distortion

x_d, y_d Image coordinates in distorted image: center at (0,0)

x, y Image coordinates in undistorted image



$$x = x_d + [2p_1 x_d y_d + p_2 (r^2 + 2x_d^2)] \quad \text{Mapping for } x \quad r^2 = x_d^2 + y_d^2 \quad \text{Square of radial distance}$$

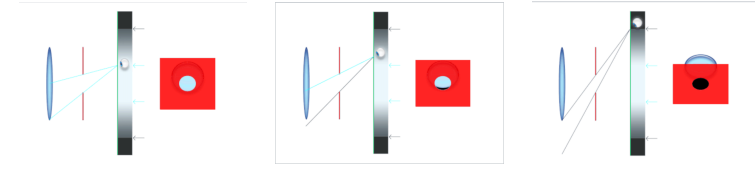
$$y = y_d + [2p_2 x_d y_d + p_1 (r^2 + 2y_d^2)] \quad \text{Mapping for } y$$

20.2.1 Mapping for radial and tangential undistortion

$$\begin{bmatrix} x \\ y \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \cdot \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2 (r^2 + 2x_d^2) \\ 2p_2 x_d y_d + p_1 (r^2 + 2y_d^2) \end{bmatrix}$$

20.3 Other camera issues

20.3.1 Vignetting



Caused by construction (lense borders, limited aperture)

Correct by scaling the intensity $f'(x, y)$ with a local correction factor $h(x, y)$

20.3.2 Color

Reason:

Color specific sensitivity variations form chip to chip due to production variability
Individual nonlinear response of RGB cells

Countermeasure:

E.g. Calibrate with color pattern as a reference to linearize the response of the RGB cells

20.3.3 Sampling time (Shutter)

Global shutter chip

All pixels are sampled simultaneously

Rolling shutter chip

Image rows are sampled sequentially



V 3D Vision

21 3D vision problems

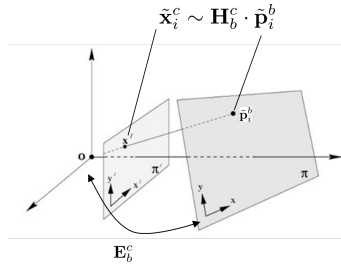
21.1 Single-view pose-estimation

Input:

- Vectors of markerpoints in object coordinates
- (Camera intrinsic parameters)

Desired output:

- Camera intrinsic matrix (i.e. calibration matrix)
- Camera extrinsic matrix (i.e. relativ pose object/camera)



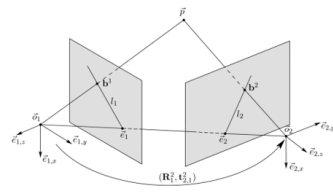
21.2 Stereo-view depth map

Input:

- Camera intrinsics
- Camera extrinsics
- Left right point correspondences

Desired output:

- Depth map (z-distances of object surface shown on each pixel)
- Point cloud (3D-coordinates of object points)



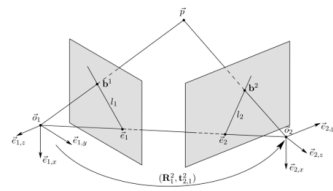
21.3 Stereo-view structure-from-motion

Input:

- Camera intrinsics
- Left right point correspondences

Desired output:

- Camera extrinsics
- 3D point coordinates



22 Triangulation

Input:

- Camera intrinsics $(k_{11}, k_{12}, k_{13}, p_{11}, p_{12}), (k_{21}, k_{22}, k_{23}, p_{21}, p_{22}), \mathbf{K}_1, \mathbf{K}_2$
- Camera extrinsics $\mathbf{R}_1^2, \mathbf{t}_{21}^2$
- Left to right correspondences $(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2)$

Desired output:

3D position of point in 3D in coordinates of camera 1 $\mathbf{p}^1$

22.0.1 Stereo projection

Two cameras form a stereo camera. A stereo camera is converging if their cameras view overlapping regions of a 3D scene

A single point within the overlapping region is projected to points that can be expressed in homogeneous image coordinates

$$\tilde{\mathbf{x}}_i = \mathbf{P}_1^i \cdot \tilde{\mathbf{p}}^1 \quad i = 1, 2$$

The homogeneous 3D point has four components

$$\tilde{\mathbf{p}}^1 \in \mathbb{R}^4 \setminus \{0\}$$

The projection matrices are

$$\mathbf{P}_1^i = \mathbf{K}_i \cdot [\mathbf{R}_i^1 | \mathbf{t}_{i1}^1] \in \mathbb{R}^{3 \times 4}$$

The above model is valid for undistorted camera images only. For their undistortion, the camera intrinsics are required

$$(k_{11}, k_{12}, k_{13}, p_{11}, p_{12}), (k_{21}, k_{22}, k_{23}, p_{21}, p_{22})$$

22.0.2 Stereo projection errors

The true projections deviate from the ideal projections due to various effects summarized as noise.

An often used model assumes random distortions of the inhomogeneous coordinates, so that the distorted coordinates are

$$\mathbf{x}'_i = \mathbf{x} + \mathbf{n} \quad \mathbf{n} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}_2)$$

Here $\mathbf{I}_2$ is the 2×2 unit matrix and the covariance matrix of the noise vector $\mathbf{n}$ is

$$\sigma^2 \mathbf{I}_2$$

Consequently, the noisy homogeneous image coordinates are

$$\tilde{\mathbf{x}}_i = \lambda \begin{bmatrix} x_i + n_x \\ y_i + n_y \\ 1 \end{bmatrix}$$

22.1 Reprojection error

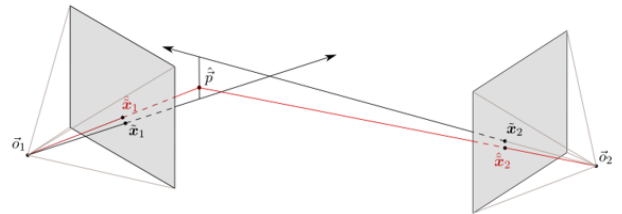
A good estimate of the 3D point minimizes a suited error measure

The reprojection error $\mathbf{e}_i$ summarizes the squared distance between the measured image point and the image point that directs towards an assumed 3D point

$$\mathbf{e}_i := \|\tilde{\mathbf{x}}^i - \mathbf{P}_1^i \cdot \tilde{\mathbf{p}}^1\|_2^2$$

With the before stated $\tilde{\mathbf{p}}^1 \in \mathbb{R}^4 \setminus \{0\}$, the sum of the reprojection errors in both images is

$$\mathbf{e} := \|\tilde{\mathbf{x}}^1 - \mathbf{P}_1^1 \cdot \tilde{\mathbf{p}}^1\|_2^2 + \|\tilde{\mathbf{x}}^2 - \mathbf{P}_2^2 \cdot \tilde{\mathbf{p}}^1\|_2^2$$



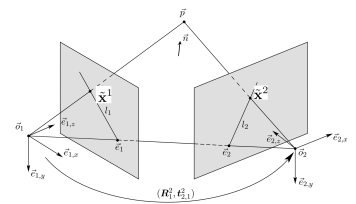
23 Epipolar geometry

A 3D point is mapped to the image 1 at the coordinates $\tilde{\mathbf{x}}^1 = \mathbf{K}_1 \cdot \mathbf{p}^1$.

The corresponding point on image 2 lies on the epipolar line l_2 .

This means, test if the two image points are on the same epipolar plane This can be broken down to two steps:

1. Express the epipolar plane's normal vector $\tilde{\mathbf{n}}$ by point $\tilde{\mathbf{x}}^1$
2. Check whether $\tilde{\mathbf{x}}^2$ is orthogonal to $\tilde{\mathbf{n}}$



23.1 Derivation Step 1

Express the normal vectors $\mathbf{p}^1$ and $\mathbf{p}^2$ by the homogeneous points $\tilde{\mathbf{x}}^1$ and $\tilde{\mathbf{x}}^2$

Recall the identity

$$\tilde{\mathbf{x}}^i = \lambda_i \mathbf{K}_i \mathbf{p}^i, \quad \text{for } i = 1, 2$$

Notice the unknown nonzero scaling term

$$\lambda_i$$

We can solve the equation for the desired point coordinates

$$\mathbf{p}^i = \frac{1}{\lambda_i} \mathbf{K}_i^{-1} \tilde{\mathbf{x}}^i, \quad \text{for } i = 1, 2$$

We assume two vectors

$$\mathbf{p}, \mathbf{u} \in \mathbb{R}^3$$

The cross product of a vector with itself is the zero vector

$$\mathbf{p} \times \mathbf{p} = \mathbf{0}$$

The cross product can be written as a matrix-vector product.

$$\mathbf{u} \times \mathbf{p} = \hat{\mathbf{U}} \cdot \mathbf{p}$$

$$\hat{\mathbf{U}} := \mathbf{u}_\times := \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} \mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

To represent $\vec{n}$ in coordinates of camera 2:

Derivation:

$$\begin{aligned}\mathbf{n}^2 &= \lambda \mathbf{t}_{2,1}^2 \times \mathbf{p}^2 \\ &= \lambda \mathbf{t}_{2,1}^2 \times (\mathbf{R}_1^2 \mathbf{p}^1 + \mathbf{t}_{2,1}^2) \\ &= \lambda \mathbf{t}_{2,1}^2 \times \mathbf{R}_1^2 \mathbf{p}^1 \\ &= \lambda \hat{\mathbf{T}} \mathbf{R}_1^2 \mathbf{p}^1 \\ &= \frac{\lambda}{\lambda_1} \hat{\mathbf{T}} \mathbf{R}_1^2 \mathbf{K}_1^{-1} \tilde{\mathbf{x}}^1\end{aligned}$$

Identities used:

$$\begin{aligned}\mathbf{p}^2 &= \mathbf{R}_1^2 \mathbf{p}^1 + \mathbf{t}_{2,1}^2 \\ 0 &= \mathbf{t}_{2,1}^2 \times \mathbf{t}_{2,1}^2 \\ \hat{\mathbf{T}} &= \mathbf{t}_{2,1}^2 \times \\ \mathbf{p}^i &= \frac{1}{\lambda_i} \mathbf{K}_i^{-1} \tilde{\mathbf{x}}^i\end{aligned}$$

23.2 Derivation Step 2

Test if $\tilde{\mathbf{x}}^2$ lies on the epipolar plane, i.e. if the corresponding point $\mathbf{p}^2$ is orthogonal to $\mathbf{n}^2$

23.2.1 Fundamental matrix

The test is satisfied if

$$(\mathbf{p}^2)^T \cdot \mathbf{n}^2 = 0$$

Into this equation, we substitute the identities

$$\mathbf{n}^2 = \frac{\lambda}{\lambda_1} \hat{\mathbf{T}} \mathbf{R}_1^2 \mathbf{K}_1^{-1} \tilde{\mathbf{x}}^1$$

$$\mathbf{p}^2 = \frac{1}{\lambda_2} \mathbf{K}_2^{-1} \tilde{\mathbf{x}}^2$$

Finally we get the test formulation

$$(\tilde{\mathbf{x}}^2)^T \underbrace{(\mathbf{K}_1^{-1})^T \hat{\mathbf{T}} \mathbf{R}_1^2 \mathbf{K}_1^{-1}}_{\mathbf{F}_1^2} \tilde{\mathbf{x}}^1 = 0$$

The matrix $\mathbf{F}_1^2$ is known as the **Fundamental matrix**

23.2.2 Essential matrix

With the same equation from before

$$(\tilde{\mathbf{x}}^2)^T \underbrace{(\mathbf{K}_1^{-1})^T \hat{\mathbf{T}} \mathbf{R}_1^2 \mathbf{K}_1^{-1}}_{\mathbf{F}_1^2} \tilde{\mathbf{x}}^1 = 0$$

$$\mathbf{p}^i = \frac{1}{\lambda_i} \mathbf{K}_i^{-1} \tilde{\mathbf{x}}^i$$

$$(\mathbf{p}^2)^T \underbrace{\hat{\mathbf{T}} \mathbf{R}_1^2}_{\mathbf{E}_1^2} \mathbf{p}^1 = 0$$

and the identity

we can express a similar constraint for the object point $\vec{p}$ expressed in two camera coordinates:

The matrix $\mathbf{E}_1^2$ is known as the **Essential matrix**

23.3 Applications

Stereo matches violating the epipolar constraint can be considered outliers. However, all corresponding image points, do not exactly satisfy the epipolar constraint in practice.

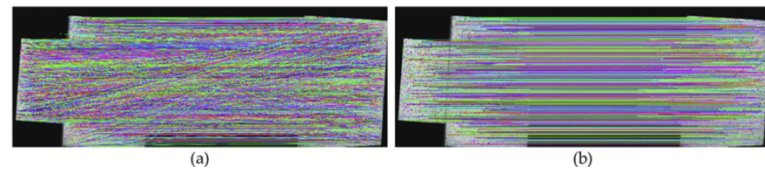
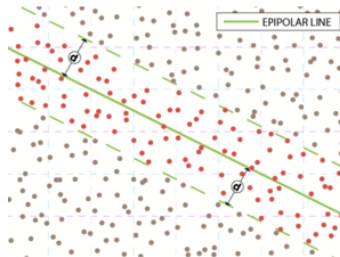
Reasons

- Imperfect camera calibration
- Viewpoint change due to baseline
- Image noise

Solution

- Search on a slice with a radius d around the epipolar line
- This however provides imperfect matches, hence an error tolerant i.e. robust, triangulation method is needed
- Use a robust rig that robustly aligns the cameras, pay attention to temperature change, if needed repeat calibration

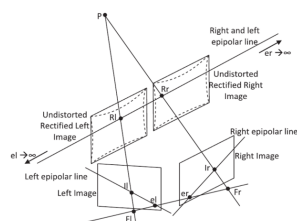
Feature matching by SIFT: (a) coarse matches; (b) removal of wrong matches who violate the epipolar constraint



24 Rectification

Rectification means to introduce new sensor planes:

- the optical centers of the cameras remain unchanged
- the epipolar lines become horizontal and identical in the left and right rectified images



24.1 Stereo rectification

Object distance

z

Baseline

b

Focal length

f

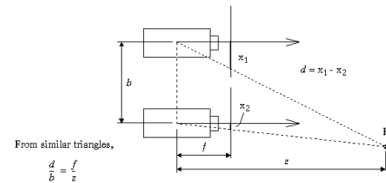
Image coordinate resolution (Disparity)

Δd

Depth resolution

Δz

$$\Delta z = \frac{z^2}{bf} \Delta d$$



VI Object Recognition

25 Matching by correlation

25.1 Cross correlation

Given are two signal vectors

$$\mathbf{a} = [a_0, a_1, \dots, a_{n-1}] \quad \mathbf{b} = [b_0, b_1, \dots, b_{n-1}]$$

Their scalar or inner product is defined as

$$\mathbf{a}^T \mathbf{b} = |\mathbf{a}| \cdot |\mathbf{b}| \cdot \cos(\theta) \quad \mathbf{a}^T \mathbf{b} = \sum_{i=0}^{n-1} a_i b_i$$

25.2 Normalized cross correlation coefficient

$$\gamma(\mathbf{a}, \mathbf{b}) := \frac{(\mathbf{a} - \bar{\mathbf{a}})^T (\mathbf{b} - \bar{\mathbf{b}})}{|\mathbf{a} - \bar{\mathbf{a}}| \cdot |\mathbf{b} - \bar{\mathbf{b}}|} \quad \bar{\mathbf{a}} := \frac{1}{n} \sum_{i=0}^{n-1} a_i \quad \bar{\mathbf{b}} := \frac{1}{n} \sum_{i=0}^{n-1} b_i$$

25.3 Template matching

Given a mask $w(x, y)$

of size $m \times n$ and an image $f(x, y)$

$$c(x, y) = \sum_s \sum_t w(s, t) f(x + s, y + t)$$

$$\gamma(x, y) := \frac{\sum_s \sum_t [w(s, t) - \bar{w}][f(x + s, y + t) - \bar{f}_{xy}]}{\left\{ \sum_s \sum_t [w(s, t) - \bar{w}]^2 \sum_s \sum_t [f(x + s, y + t) - \bar{f}_{xy}]^2 \right\}^{1/2}}$$

Using the coefficient from before we can check for **perfect correspondence with +1** and **perfect anti-correspondence with -1**

Very useful, but impaired by scaling and rotation

26 Patterns (feature vectors)

The goal is to assign a given pattern to its class. Patterns are usually arranged as feature vectors.

The example data shown on the right, set the stage for the subsequent discussion:

There are three classes, but only two descriptors are available:

- Petal length
- Petal width

The goal is to classify a flower based on these two measurements into one of the three classes

Hence the feature space is two dimensional and the goal of automatic classification is to separate this space into three regions corresponding to the three classes

This can clearly be done for the Iris setosa, but separating virginica from versicolor is not possible

Hence we might need different and/or additional features

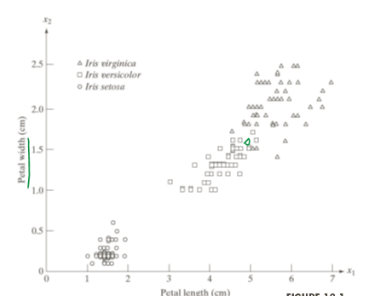
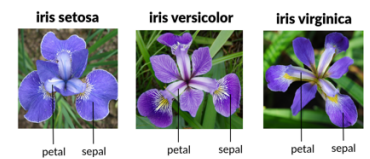


FIGURE 12.1 Three types of iris flowers described by two measurements.



27 Object recognition

27.1 Minimum-distance classifier

The mean vector of a class is a powerful prototype vector

$$\mathbf{m}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in w_j} \mathbf{x}_j$$

One simple matching function is the Euclidean distance D_j

$$D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \quad (\|\mathbf{a}\| = (\mathbf{a}^T \mathbf{a})^{1/2})$$

The rule is as stated before, feature vector $\mathbf{x}$ is assigned to class w_i , if $D_i(\mathbf{x})$ is the smallest distance

$$D_i(\mathbf{x}) = \min_{j \in 1, 2, \dots, W} \{D_j(\mathbf{x})\}$$

This is quite intuitive, but we will use discriminants $d_i(\mathbf{x})$ instead, to minimize the number of computations:

$$d_i(\mathbf{x}) \quad i = 1, 2, \dots, W$$

Class w_i is selected for a feature vector $\mathbf{x}$, if

$$d_i(\mathbf{x}) > d_j(\mathbf{x}) \quad j = 1, 2, \dots, W; j \neq i$$

The same class j that minimizes the Euclidean distance, maximizes the discriminant function

$$d_j(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T \mathbf{m}_j$$

27.1.1 Finding the decision boundary

Now the decision boundary between class w_i and w_j for a minimum distance classifier can be described as the solutions $\mathbf{x}$ of

$$\begin{aligned}d_{ij}(\mathbf{x}) &= d_i(\mathbf{x}) - d_j(\mathbf{x}) \\&= \mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j) - \frac{1}{2}(\mathbf{m}_i - \mathbf{m}_j)^T(\mathbf{m}_i + \mathbf{m}_j) \\&= 0\end{aligned}$$

Clearly, this is an equation for a straight line, if the feature space is 2 dimensional, or a hyperplane for higher dimensions

This line goes through the midpoint of the line connecting the centroids of the classes i and j , because $\mathbf{x} = \frac{1}{2}(\mathbf{m}_i + \mathbf{m}_j)$ satisfies the equation.

Furthermore, this line is perpendicular to the line connecting the centroids of the classes i and j , since starting at $\mathbf{x} = \frac{1}{2}(\mathbf{m}_i + \mathbf{m}_j)$, the equation stays satisfied as long as the inner product $\mathbf{x}^T(\mathbf{m}_i - \mathbf{m}_j)$ does not change its value

27.1.2 Flower example

With Iris versicolor (w_1) and Iris setosa (w_2), the mean vectors are $\mathbf{m}_1 = (4.3, 1.3)^T$ and $\mathbf{m}_2 = (1.5, 0.3)^T$.

Now we get:

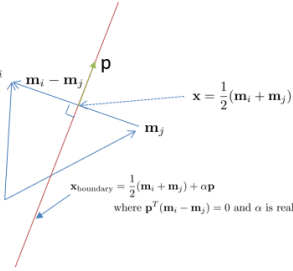
$$d_1(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_1 - \frac{1}{2} \mathbf{m}_1^T \mathbf{m}_1 = 4.3x_1 + 1.3x_2 - 10.1$$

$$d_2(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_2 - \frac{1}{2} \mathbf{m}_2^T \mathbf{m}_2 = 1.5x_1 + 0.3x_2 - 1.17$$

$$d_{12}(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 2.8x_1 + 1.0x_2 - 8.9 = 0$$

$$d_{12}(\mathbf{x}) > 0 \quad \mathbf{x} \text{ belongs to } w_1$$

$$d_{12}(\mathbf{x}) < 0 \quad \mathbf{x} \text{ belongs to } w_2$$



27.2 Maximum A Posteriority (MAP, Bayes) classifier

The simplest case is a 1D problem with two classes. Both conditional probability density functions are assumed to be Gaussian with known means and variances, hence the discrimination functions are

$$d_j(\mathbf{x}) = p(\mathbf{x} | w_j)P(w_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(\mathbf{x}-\mathbf{m}_j)^2}{2\sigma_j^2}} P(w_j)$$

In a more realistic case, the pattern $\mathbf{x}$, (i.e., the feature vector) is not one dimensional but n-dimensional. Still the assumption is, that the conditional probabilities are n-dimensional Gaussian ($|C_j|$ is the determinant of C_j)

The parameters of an n-dimensional Gaussian probability density function are the mean vector and the covariance matrix

These are in practice estimated using the sample mean and the sample covariance matrix

Combined we get the decision function for n-dimensional Gaussian pattern classes

$$d_j(\mathbf{x}) = \ln P(w_j) - \frac{1}{2} \ln |C_j| - \frac{1}{2} [(\mathbf{x} - \mathbf{m}_j)^T C_j^{-1} (\mathbf{x} - \mathbf{m}_j)]$$

27.2.1 MAP classifier with identical covariance matrices

When covariance matrices are equal, the function becomes simpler

$$d_j(\mathbf{x}) = \ln P(w_j) + \mathbf{x}^T C^{-1} \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T C^{-1} \mathbf{m}_j$$

27.2.2 MAP example

$$C = \frac{1}{16} \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}, \mathbf{m}_1 = \frac{1}{4} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}, \mathbf{m}_2 = \frac{1}{4} \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$$

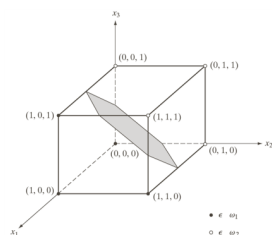
$$d_j(\mathbf{x}) = \mathbf{x}^T C^{-1} \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T C^{-1} \mathbf{m}_j$$

$$d_1(\mathbf{x}) = 4x_1 - 1.5$$

$$d_2(\mathbf{x}) = -4x_1 + 8x_2 + 8x_3 - 5.5$$

$$d_1(\mathbf{x}) - d_2(\mathbf{x}) = 8x_1 - 8x_2 - 8x_3 + 4 = 0$$

(Class border(planar))



27.2.3 Finding the boundary when $p(x)$ is identical for all classes

$$\text{MAP boundary between } \omega_i, \omega_j : p(x | \omega_i)P(\omega_i) = p(x | \omega_j)P(\omega_j)$$

This is allowed when $p(x)$ is identical for all classes and cancels in Bayes' rule. Use this for equal misclassification costs.

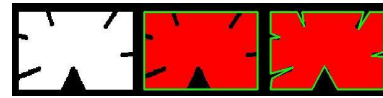
28 Region-Based Features

28.1 Contours as polygons

Contours can be represented by polygons.

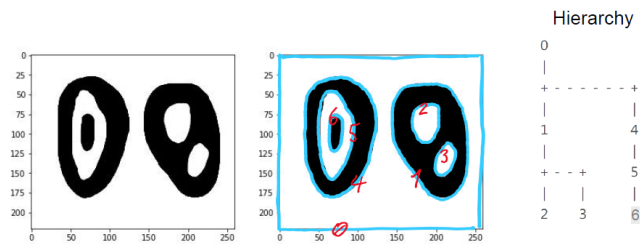
List of corner points in clockwise order:

$$[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}] \quad \mathbf{p}_i \in \mathbb{R}^2$$



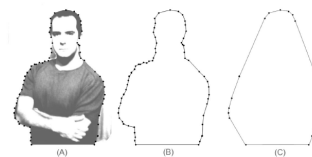
28.1.1 Hierarchy of Contours

Useful to describe relations of multiple regions. E.g. `cv2.findContours()` provides the following hierarchy of contours for a binary image



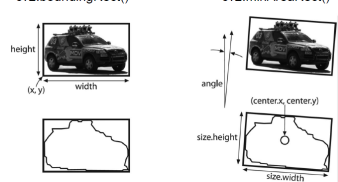
28.1.2 OpenCV: Convex hull

`cv2.convexHull()`



28.1.3 OpenCV: Bounding box

`cv2.boundingRect()`



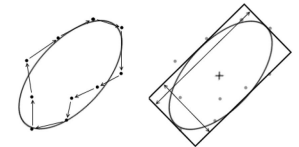
28.1.4 OpenCV: minAreaRect

The bounding box has one of its edges in parallel with the object's convex hull.

1. Compute the convex hull of the cloud.
2. For each edge of the convex hull:
3. compute the edge orientation (with arctan),
4. rotate the convex hull using this orientation in order to compute easily the bounding rectangle area with min/max of x/y of the rotated convex hull,
5. Store the orientation corresponding to the minimum area found,
6. Return the rectangle corresponding to the minimum area found.

28.1.5 OpenCV: fitEllipse

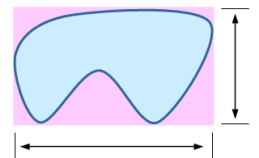
Ellipse can be fit to corner points of a polygon. The fitting is the solution of a linear Least Squares problem.



28.2 Simple descriptors

Simple shape descriptors are based on

- the area of an object
- the length of the perimeter
- minimum and maximum diameter
- area of bounding rectangle



The descriptors are independent of translation, scale and rotation and are easy to compute.

28.2.1 Perimeter and area

- Formfactor $F = 4\pi \frac{\text{area}}{\text{perimeter}^2}$
- Solidity $S = \frac{\text{area}}{\text{convex area}}$
- Convexity $C = \frac{\text{convex perimeter}}{\text{perimeter}}$

28.2.2 Diameter and area

- Aspect Ratio $A = \frac{\text{max diameter}}{\text{min diameter}}$
- Roundness $R = \frac{4}{\pi} \frac{\text{area}}{(\text{max diameter})^2}$
- Compactness $P = \sqrt{R}$
- Extent $E = \frac{\text{area}}{\text{area bounding rectangle}}$

28.2.3 Fibres

- Curl $C_r = \frac{\text{length}}{\text{fibre length}}$
- Elongation $E_l = \frac{\text{fibre length}}{\text{fibre width}}$

28.2.4 Example

$$\begin{aligned}A_c &= \pi \cdot r^2 & F_c &= 4 \cdot \pi \cdot \frac{\pi \cdot r^2}{(2 \cdot \pi \cdot r)^2} = 1.0 \\P_c &= 2 \cdot \pi \cdot r \\A_s &= a^2 & F_s &= 4 \cdot \pi \cdot \frac{a^2}{16 \cdot a^2} = 0.785 \\P_s &= 4 \cdot a \\A_t &= \frac{\sqrt{3}}{4} \cdot a^2 & F_t &= 4 \cdot \pi \cdot \frac{\sqrt{3} \cdot a^2}{36 \cdot a^2} = 0.603 \\P_t &= 3 \cdot a\end{aligned}$$

28.3 Area-based descriptors

Are based on area, moments and principal components. The are defined for binary and grayvalue images. The following example is with image size $M \times N$

28.3.1 Object centroid

General:

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y)$$

Centroid coordinates:

$$M_x = \frac{m_{10}}{m_{00}} \quad M_y = \frac{m_{01}}{m_{00}}$$

28.3.2 2D central moments

Are independent of translation

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \quad \bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}}$$

Normalized central moments:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^\gamma} \quad \gamma = \frac{p+q}{2} + 1$$

28.3.3 Hu-moments

Independent of translation, scale or rotation

$$\begin{aligned} \Phi_1 &= \eta_{20} + \eta_{02} \\ \Phi_2 &= (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2 \\ \Phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\ \Phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\ \Phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\ &\quad (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ \Phi_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + \\ &\quad 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\ \Phi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\ &\quad (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})[(3\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \end{aligned}$$

28.3.4 Principal axis of inertia

$$\begin{aligned} m_{10} &= \sum_{\text{Object}} x & m_{20} &= \sum_{\text{Object}} x^2 & m_{00} &= \sum_{\text{Object}} 1 \\ m_{01} &= \sum_{\text{Object}} y & m_{02} &= \sum_{\text{Object}} y^2 & m_{11} &= \sum_{\text{Object}} x \cdot y \end{aligned}$$

$$M_{xx} = m_{20} - \frac{m_{10}^2}{m_{00}} \quad M_{yy} = m_{02} - \frac{m_{01}^2}{m_{00}} \quad M_{xy} = m_{11} - \frac{m_{10} \cdot m_{01}}{m_{00}}$$

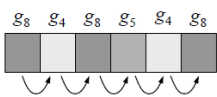
$$\text{Angle of principal axis: } \Theta = \frac{1}{2} \cdot \arctan\left(\frac{2 \cdot M_{xy}}{M_{xx} - M_{yy}}\right) \quad (\text{use } \text{atan2}(\cdot, \cdot) \text{ in code!})$$

28.4 Texture based descriptors

No precise mathematical definition. Three approaches: **statistical** (histogram moments = global; co-occurrence = local), **spectral** (Fourier), **structural**.

28.4.1 Co-occurrence matrix

Determine the frequency of gray value changes according to the relative position of the pixels and scale with number of histogram. Often the number of gray values is reduced.



change	frequency
$g_8 \rightarrow g_4$	1
$g_4 \rightarrow g_8$	2
$g_8 \rightarrow g_5$	1
$g_5 \rightarrow g_4$	1

Example:

Choose position operators according to texture, results vary.

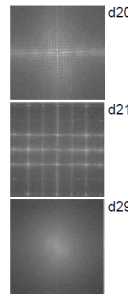


Descriptor	Explanation	Formula
Response of G		$p_{ij} = \frac{G_{ij}}{\sum_i \sum_j G_{ij}}$
Maximum probability	Strongest response of G	$\max(p_{ij})$
Correlation	How correlated a pixel is to its neighbour. Range 1 to -1. Not defined if $\sigma_r = 0$ or $\sigma_c = 0$	$\sum_{i=1}^K \sum_{j=1}^K \frac{(i - m_r)(j - m_c)p_{ij}}{\sigma_r \sigma_c}$
Contrast	Intensity contrast from a pixel is to its neighbour over full image. Range 0 to $(K-1)^2$	$\sum_{i=1}^K \sum_{j=1}^K (i - j)^2 p_{ij}$
Uniformity (Energy)	Range [0, 1]. 1 for a constant image	$\sum_{i=1}^K \sum_{j=1}^K p_{ij}^2$
Homogeneity	Spatial closeness of distribution. Range [0, 1]. Max when G is diagonal.	$\sum_{i=1}^K \sum_{j=1}^K \frac{p_{ij}}{1 + i - j }$
Entropy	Randomness. 0 when all $p_{ij} = 0$, max when all p_{ij} equal. Max value is $2 \log_2(K)$	$-\sum_{i=1}^K \sum_{j=1}^K p_{ij} \log_2(p_{ij})$

28.4.2 Spectral approach

Compute the spectrum of a textured object, a scaled spectrum eliminates dependence on illumination

$$S(u, v) = FFT(f(x, y)) \quad S_n(u, v) = \frac{|S(u, v)|}{\sqrt{\sum_{u=2}^m \sum_{v=2}^n |S(u, v)|^2}}$$



	Energy	Contrast	Entropy	Homogeneity
uniform gray (128)	1.0	0.0	0.0	1.00
d20	1.0	3.6 e6	1168.0	3.41
d21	1.0	5.80 e6	1219.0	2.80
d29	1.0	5.75 e6	1781.0	5.87
noise $m=0, \sigma=0.1$	1.0	7.58 e6	1295.0	2.90

14.05.2025 40

VII Neural Networks

NNs learn decision functions directly from data — no probability-density (Gaussian) assumption needed.

29 Perceptron

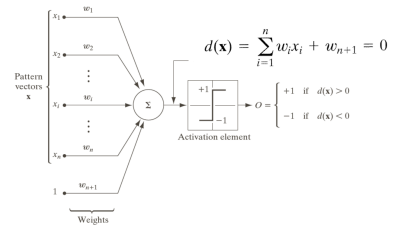
This network uses a single neuron, called a perceptron. It learns a linear decision function that separates two linearly separable training sets.

For an n -dimensional input vector, the hyperplane is described as

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n + w_{n+1} = 0$$

With the sign function $d(\mathbf{x})$ we get the following:

- $d(\mathbf{x}) > 0 \Rightarrow$ Class c_1
- $d(\mathbf{x}) < 0 \Rightarrow$ Class c_2
- $d(\mathbf{x}) = 0 \Rightarrow$ decision boundary

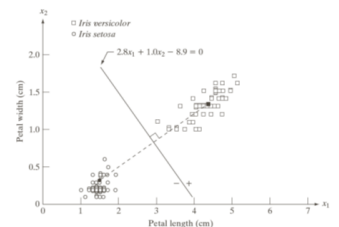


29.1 Two linearly separable classes

The basic idea of the algorithm is that the pattern vectors of the training set are not presented all at once but one by one and repeated. In each training step, one of the pattern vectors is used to check the classifier for correct classification.

The weight vector is only changed, if a misclassification happens.

The training stops, when there are no more misclassifications for all training samples (pattern vectors).



29.2 Two linearly inseparable classes

We can't find a weight vector to separate classes, this is by definition not possible.

We try to minimize some squared error (r represents the desired response, $r = +1$ if $y(k)$ belongs to c_1 and $r = -1$ for c_2)

Minimizing the weight works in principle but it is complex and memory hungry.

29.2.1 Stochastic Gradient Descent (SGD)

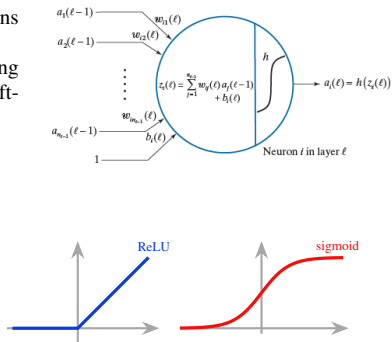
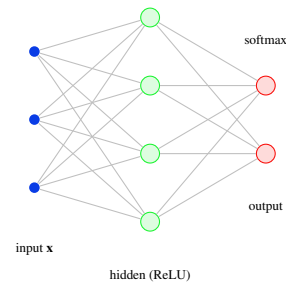
The basic idea is to simplify the algorithm by still considering only one pattern vector at a time and to correcting the weight vector

The weight vector $\mathbf{w}$ is corrected such that the error $J(\mathbf{w})$ gets reduced

$$J(\mathbf{w}) = \frac{1}{2} (r - \mathbf{w}^T \mathbf{y}(k))^2$$

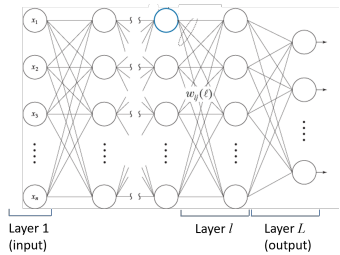
30 Multilayer feedforward NN

Stacked perceptrons with smooth activations
 $\Rightarrow$ multiclass, nonlinear boundaries.
 Hidden layers use ReLU (non-vanishing gradient $\Rightarrow$ faster); output uses softmax/sigmoid.



30.1 Forward pass

Computing the network outputs for an input vector $\mathbf{x}$, is denoted forward pass (weights and the biases must be known)
 The components of $\mathbf{x}$ are for consistency denoted as the outputs of layer $l = 1$



The pre-activation of neuron i in layer l is

Then the output of neuron i in layer l is calculated by running the pre-activation through the activation function

Evaluations are done for each layer l so that the final outputs are obtained

30.2 Matrix formulation

The outputs of layer 1 are the components of input vector $\mathbf{x}$, i.e. $\mathbf{a}(1) = \mathbf{x}$

Given the proper definition of the weight matrix

$$\mathbf{W}(l) = \begin{bmatrix} w_{11}(l) & w_{12}(l) & \dots & w_{1n_{l-1}}(l) \\ w_{21}(l) & w_{22}(l) & \dots & w_{2n_{l-1}}(l) \\ \vdots & \vdots & \dots & \vdots \\ w_{n_l1}(l) & w_{n_l2}(l) & \dots & w_{n_ln_{l-1}}(l) \end{bmatrix}$$

The pre-activations can be written as a matrix vector multiplication and a vector addition

$$\mathbf{z}(l) = \mathbf{W}(l)\mathbf{a}(l-1) + \mathbf{b}(l) \quad l = 2, 3, \dots, L$$

Finally, the activation of the element-wise activation functions, such as ReLU and sigmoid are applied component-wise

$$\mathbf{a}(l) = h[\mathbf{z}(l)] = \begin{bmatrix} h(z_1(l)) \\ h(z_2(l)) \\ \dots \\ h(z_{n_l}(l)) \end{bmatrix}$$

30.2.1 Mini Batches

During training, the forward pass is usually calculated for a mini-batch of n_p input vectors. This is different to the single input vectors handled in each iteration of the stochastic gradient descent algorithm (delta rule).
 The vectors of a mini-batch are concatenated column wise to create an $n \times n_p$ matrix $\mathbf{X}$, where n is the dimension of an input vector and n_p is the number of input vectors in the mini-batch
 This concatenation then results in concatenated pre-activations and activations but everywhere else stays the same

TABLE 12.2
Steps in the matrix computation of a forward pass through a fully connected, feedforward multilayer neural net.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Feedforward	For $\ell = 2, \dots, L$, compute $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell)$ and $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$
Step 3	Output	$\mathbf{A}(L) = h(\mathbf{Z}(L))$

30.3 Backpropagation

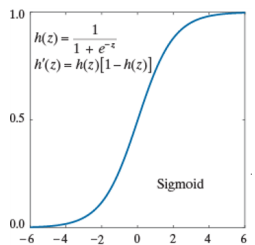
$$\text{Loss } E = \frac{1}{2} \sum_{j=1}^{n_L} (r_j - a_j(L))^2$$

$$\text{Define } \delta_j(L) = \frac{\partial E}{\partial z_j(L)} = \dots = \frac{\partial E}{\partial a_j(L)} h'(z_j(L))$$

With the sigmoid function we get

$$\delta_j(L) = h(z_j(L)) [1 - h(z_j(L))] [a_j(L) - r_j]$$

The idea is to then work from L to $L - 1$, then from $L - 1$ to $L - 2$, and so on



30.3.1 Matrix formulation

TABLE 12.3
Matrix formulation for training a feedforward, fully connected multilayer neural network using backpropagation. Steps 1–4 are for one epoch of training. $\mathbf{X}$, $\mathbf{R}$, and the learning rate parameter α , are provided to the network for training. The network is initialized by specifying weights, $\mathbf{W}(1)$, and biases, $\mathbf{B}(1)$, as small random numbers.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Forward pass	For $\ell = 2, \dots, L$, compute: $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell)$; $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$; $h'(\mathbf{Z}(\ell))$; and $\mathbf{D}(L) = (\mathbf{A}(L) - \mathbf{R}) \odot h'(\mathbf{Z}(L))$
Step 3	Backpropagation	For $\ell = L-1, L-2, \dots, 2$, compute $\mathbf{D}(\ell) = (\mathbf{W}^T(\ell+1)\mathbf{D}(\ell+1)) \odot h'(\mathbf{Z}(\ell))$
Step 4	Update weights and biases	For $\ell = 2, \dots, L$, let $\mathbf{W}(\ell) = \mathbf{W}(\ell) - \alpha \mathbf{D}(\ell)\mathbf{A}^T(\ell-1)$, $\mathbf{b}(\ell) = \mathbf{b}(\ell) - \alpha \sum_{i=1}^{n_{\ell}} \delta_k(\ell)$, and $\mathbf{B}(\ell) = \text{concatenate} \{ \mathbf{b}(\ell) \}$, where the $\delta_k(\ell)$ are the columns of $\mathbf{D}(\ell)$

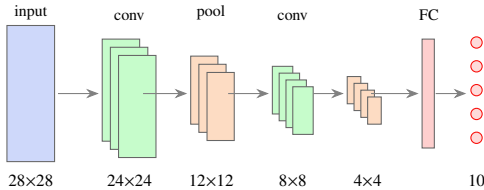
30.4 Softmax output

For class-posterior outputs use **softmax** (not element-wise): $h_{\text{SM},i}(\mathbf{z}) = e^{z_i} / \sum_k e^{z_k}$ (sums to 1). With **one-hot** labels and **cross-entropy** loss $L = \sum_i r_i \log \frac{1}{a_i^{(l)}}$, the output delta is

simply the negative error: $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{r}$. **ReLU hidden + softmax/CE output learns much faster** than sigmoid+MSE everywhere (XOR: solved in ~ 10 epochs vs. thousands).
 Initialization: **He** (ReLU), **Xavier** (softmax/sigmoid). A 3-layer NN trained with SSE approximates the Bayes decision as #samples $\rightarrow \infty$ (multispectral example: NN 95.6–96.5% vs. Bayes 96.2%).

VIII Convolutional Neural Networks (CNN)

For multidimensional (image) data; the network learns its own filters at multiple abstraction levels — no hand-crafted features.



31 Neural computations in a CNN

The convolution value at any point (x, y) , where l and k span the dimension of the kernel w

$$w \star a_{x,y} = \sum_l \sum_k w_{l,k} a_{x-l,y-k}$$

Assume w is 3×3 , then this equation expands into the expression

$$w \star a_{x,y} = w_{1,1}a_{x-1,y-1} + \dots + w_{3,3}a_{x-3,y-3}$$

Going from a 2D indexing to a 1D indexing $i = 3(l-1) + k$. Hence the convolution can also be written in this fashion

$$w \star a_{x,y} = w_1a_1 + w_2a_2 + \dots + w_9a_9 = \sum_{i=1}^9 w_i a_i$$

To get the usual NN summation for the pre-activation z , a bias is added

$$z = \sum_{i=1}^9 w_i a_i + b = w \star a_{x,y} + b$$

The activation is calculated by running the pre-activation through the activation function

$$\mathbf{a} = h(\mathbf{z})$$

Convolving the receptive field of the input image with the learned 2D filter weights and then adding a learned bias before running the result through an activation function, is identical to the steps performed in a fully connected neural network.

What is different though is the fact, that the same learned weights/bias are used for all the pixels in a given feature map

32 Forward pass

Using a layer notation again, the pre-activation in a given layer l at a given pixel (x, y) can be calculated

$$z_{x,y}(l) = \sum_l \sum_k w_{l,k}(l) a_{x-l,y-k}(l-1) + b(l)$$

33 Backpropagation

The forward pass is very similar to fully connected NN's.

For backpropagation, the kernel does NOT depend on where it is by (x, y) , this is one of the advantages of a CNN.

The kernel weights are shared for the entire feature map

$$\delta_{x,y}(l) = h' \left(z_{x,y}(l) \right) \left[\delta_{x,y}(l+1) \star \text{rot180}(w(l+1)) \right]$$

34 Math-Stuff

- Derivative definitions from second lecture

34.1 Trigonometry

α	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	$\frac{2\pi}{3}$	$\frac{3\pi}{4}$	$\frac{5\pi}{6}$	π	$\frac{7\pi}{6}$	$\frac{5\pi}{4}$	$\frac{4\pi}{3}$	$\frac{3\pi}{2}$	$\frac{5\pi}{3}$	$\frac{7\pi}{4}$	$\frac{11\pi}{6}$	2π
α°	0°	30°	45°	60°	90°	120°	135°	150°	180°	210°	225°	240°	270°	300°	315°	330°	360°
$\sin(\alpha)$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{2}$	-1	$-\frac{\sqrt{3}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{1}{2}$	0
$\cos(\alpha)$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{2}$	-1	$-\frac{\sqrt{3}}{2}$	$-\frac{\sqrt{2}}{2}$	$-\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1
$\tan(\alpha)$	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	$\pm\infty$	$-\sqrt{3}$	-1	$-\frac{\sqrt{3}}{3}$	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	$\pm\infty$	$-\sqrt{3}$	-1	$-\frac{\sqrt{3}}{3}$	0
$\cot(\alpha)$	$\pm\infty$	$\sqrt{3}$	1	$\frac{\sqrt{3}}{3}$	0	$-\frac{\sqrt{3}}{3}$	-1	$-\sqrt{3}$	$\pm\infty$	$\sqrt{3}$	1	$\frac{\sqrt{3}}{3}$	0	$-\frac{\sqrt{3}}{3}$	-1	$-\sqrt{3}$	$\pm\infty$

34.2 Crossproduct

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

wenn $a_3 = b_3 = 1$, dann gilt: $\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 - b_2 \\ b_1 - a_1 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$

34.3 Trace of a matrix

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$\text{tr}(A) = a + e + i = \text{sum of Eigenvalues}$$

34.4 Inverse of a 3 × 3 matrix

For a general matrix

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},$$

the inverse is

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ dh - eg & bg - ah & ae - bd \end{bmatrix},$$

with

$$\det(A) = a(ei - fh) - b(di - fg) + c(dh - eg).$$

The inverse exists only if

$$\det(A) \neq 0.$$

Shortcut with column vectors: If the matrix is written with its column vectors

$$A = \begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{bmatrix},$$

then

$$\det(A) = \mathbf{a}^T (\mathbf{b} \times \mathbf{c}).$$

The inverse can be written as

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} (\mathbf{b} \times \mathbf{c})^T \\ (\mathbf{c} \times \mathbf{a})^T \\ (\mathbf{a} \times \mathbf{b})^T \end{bmatrix}.$$

This is often easier to remember: the rows of A^{-1} are cross products of the other two columns.

Useful special case: camera intrinsic matrix: For

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

the inverse is

$$K^{-1} = \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix}.$$

Useful special case: pure translation homography: For

$$H = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix},$$

the inverse is obtained by changing the sign of the translation:

$$H^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}.$$

For line mapping by a homography, use

$$\ell' \sim H^{-T} \ell$$

with

$$H^{-T} = (H^{-1})^T.$$

34.5 Eigenvalue shortcut for 2 × 2 matrices

For a symmetric 2×2 matrix

$$M = \begin{bmatrix} a & b \\ b & d \end{bmatrix},$$

the eigenvalues λ_1, λ_2 satisfy

$$\text{tr}(M) = \lambda_1 + \lambda_2$$

and

$$\det(M) = \lambda_1 \lambda_2.$$

Therefore, for many classification tasks it is not necessary to compute the eigenvalues explicitly.

Important shortcut: If

$$\det(M) = 0$$

and

$$\text{tr}(M) > 0,$$

then one eigenvalue must be zero and the other one must equal the trace:

$$\lambda_1 = \text{tr}(M), \quad \lambda_2 = 0$$

because

$$\lambda_1 \lambda_2 = 0, \quad \lambda_1 + \lambda_2 = \text{tr}(M).$$

Application to Harris corner detection: For the Harris matrix

$$M = \sum_i \begin{bmatrix} I_{x,i}^2 & I_{x,i} I_{y,i} \\ I_{x,i} I_{y,i} & I_{y,i}^2 \end{bmatrix},$$

the eigenvalues describe how much gradient energy exists in independent directions.

$$\lambda_1 \geq \lambda_2 \geq 0$$

Useful exam check: If

$$\det(M) = 0 \quad \text{and} \quad \text{tr}(M) > 0,$$

then the matrix has only one nonzero eigenvalue. Therefore the neighbourhood is classified as an edge, not a corner.

$$\det(M) = 0, \text{tr}(M) > 0 \Rightarrow \text{edge}$$

- **Homogeneous pt** $\tilde{\mathbf{x}} = \lambda[x, y, 1]^T$; inhom. $x = \tilde{x}_1 / \tilde{x}_3$.
 - **Line** $\tilde{\mathbf{l}} = [a, b, c]^T$, $\langle \tilde{\mathbf{x}}, \tilde{\mathbf{l}} \rangle = 0$.
 - **Cross product** as matrix: $a \times b = [a]_{\times} b$.
 - **SVD** $A = U \Sigma V^T$; $A \mathbf{x} = 0 \Rightarrow \mathbf{x} = \mathbf{v}_{\min}$ (smallest σ).
 - **Rotation** $R^{-1} = R^T$, $\det R = 1$.
- **Rigid motion** inverse: $T^{-1} = \begin{bmatrix} R^T & -R^T t \\ 0 & 1 \end{bmatrix}$.
 - **Calibration** $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$.
 - **Epipolar** $\tilde{\mathbf{x}}_2^T E \tilde{\mathbf{x}}_1 = 0$, $E = [t]_{\times} R$.
 - **Depth** $z = bf/d$; $\sigma_z = \frac{z^2}{bf} \sigma_n$.
 - **Softmax** $\sigma_i = e^{z_i} / \sum_k e^{z_k}$; CE $\text{grad} = \mathbf{a} - \mathbf{r}$.
 - **Sigmoid** $h' = h(1 - h)$; **ReLU** $h' = \mathbb{I}[z > 0]$.