Computer Vision Summary 2020

Luca Ebner (ebnerl@student.ethz.ch)

February 6, 2021

1. Contents

2. Introduction and Projective Geometry

Introduction

Goal => Interpreting images

What makes Computer Vision challenging?

- Image data is just an array of numbers (grayscale image)
- Change of view-point totally changes grayscale image
- Illumination changes
- Scale changes (Zoomed In/Out)
- Deformation
- Occlusion
- Background clutter
- Motion
- Object intra-class variation (Different designs of chair)

Projective Geometry

Homogenous representation of 2D points and lines:

$$ax + by + c = (a, b, c)^T (x, y, 1) = l^T x = 0 \tag{1}$$

- A point x lies on the line l if and only if $l^T x = 0$
- The scale of the point and line does not matter
- Inhomogeneous: $x = (x, y)^T$, $X = (x, y, z)^T$
- Homogenous: $x = (x, y, 1)^T = (x_1, x_2, x_3)^T$, $X = (x, y, z, 1)^T = (X_1, X_2, X_3, X_4)^T$

Intersection of lines l and l':

$$x = l \times l' \tag{2}$$

Line through two points x and x':

$$l = x \times x' \tag{3}$$

Intersection of parallel lines $l = (a, b, c)^T$ and $l' = (a, b, c')^T$:

$$l \times l' = (b, -a, 0)^T \tag{4}$$

- Ideal points: $x = (x_1, x_2, 0)^T$
- Line at infinity $l_\infty = (0, 0, 1)^T$

Homogeneous representation of 3D points on a plane $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^T$:

$$\pi_1 X_1 + \pi_2 X_2 + \pi_3 X_3 + \pi_4 X_4 = 0 \tag{5}$$

- A point X lies on a plane π if and only if: $\pi^T X = 0$

Plane $\boldsymbol{\pi}$ from points $\mathbf{X}_1^T\boldsymbol{\pi} = 0, \mathbf{X}_2^T\boldsymbol{\pi} = 0, \mathbf{X}_3^T\boldsymbol{\pi} = 0$:

$$\begin{bmatrix} \boldsymbol{X}_1^T \\ \boldsymbol{X}_2^T \\ \boldsymbol{X}_3^T \end{bmatrix} \boldsymbol{\pi} = 0 \text{ (solve for right nullspace)} \tag{6}$$

Point X from planes $\boldsymbol{\pi}_1^T X = 0, \boldsymbol{\pi}_2^T X = 0, \boldsymbol{\pi}_3^T X = 0$:

$$\begin{bmatrix} \boldsymbol{\pi}_1^T \\ \boldsymbol{\pi}_2^T \\ \boldsymbol{\pi}_3^T \end{bmatrix} \mathbf{X} = 0 \text{ (solve for right nullspace)} \tag{7}$$

Plane representation with its span M:

$$\mathbf{M} = [\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}] \tag{8}$$

- A plane can be represented by 3 points that span the plane.
- Every point X on the plane can be expressed as a linear combination from its span points: $X = \mathbf{M}x$.

Line representation with its span W:

$$W = \begin{bmatrix} A^T \\ B^T \end{bmatrix} \tag{9}$$

- A line can be represented by 2 points A, B that span the line.
- Every point X on the line can be expressed as a linear combination from its span points: $X = \mathbf{W}x$.
- Example for x-axis: $W = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$, where the first line is the point in the origin and the second line is an ideal point on the x-axis.

Line representation by Dual representation $\mathbf{W}^*$:

$$W^* = \begin{bmatrix} P^T \\ Q^T \end{bmatrix} \tag{10}$$

- A line can be represented as the intersection of 2 planes P,Q.
- It holds that $W^*W^T = W{W^*}^T = 0_{2\times 2}$.
- Example for x-axis: $W^* = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, where the first line is the z plane and the second line is the y plane.

Points, lines and planes:

If we have line W and a point A and we want to find a plane from them, we can use the equation

$$\begin{bmatrix} W \\ A^T \end{bmatrix} \boldsymbol{\pi} = 0, \tag{11}$$

which is essentially the same as finding a plane from three points.

If we have a line W^* and a plane $\boldsymbol{\pi}$ and we want to find their intersection point, we can use the equation

$$\begin{bmatrix} W^* \\ \boldsymbol{\pi}^T \end{bmatrix} \boldsymbol{X} = 0 \tag{12}$$

which is essentially the same as finding the intersection point from three planes.

Conics (https://www.youtube.com/watch?v=o8G6pmGSlVw)

- Can be described by a 2nd order equation in the plane:

$$ax^{2"}bxy + cy^2 + dx + ey + f = 0, \text{ inhomogeneous form} \tag{13}$$

$$ax_1^2 + bx_1x_2 + cx_2^2 + dx_1x_3 + ex_2x_3 + fx_3^2 = 0, \text{ homogeneous form} \tag{14}$$

$$x^T\mathbf{C}x = 0, \text{ with } \mathbf{C} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}, \text{ matrix form} \tag{15}$$

- Conics have 5 DOF (6 parameters but scale invariant)
- The tangent line l to a conic $\mathbf{C}$ at point x is given by $l = \mathbf{C}x$.
- A line tangent to the conic $\mathbf{C}$ satisfies $l^T\mathbf{C}^*l = 0$, where $C^* = C^{-1}$ (full rank).
- A conic is degenerate if $\mathbf{C}$ is not full rank.

Conic defined by 5 points:

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{bmatrix} \mathbf{c} = 0, \text{ with } \mathbf{c} = (a, b, c, d, e, f)^T \tag{16}$$

Quadrics

Similar as Conics for 2D, we can can define Quadrics for 3D space.

$$X^T\mathbf{Q}X = 0, \text{ where } \mathbf{Q} : \text{4x4, symmetric} \tag{17}$$

- Quadrics have 9 DOF (10 parameters but scale invariant)
- The tangent plane π to a quadric $\mathbf{Q}$ at point X is given by $\pi = \mathbf{Q}X$.
- A plane tangent to the quadric $\mathbf{Q}$ satisfies $\pi^T\mathbf{Q}^*\pi = 0$, where $Q^* = Q^{-1}$ (full rank).
- A quadric is degenerate if $\mathbf{Q}$ is not full rank.
- A quadric is defined by 9 points

2D projective transformations

DEFINITION: A *projectivity* is an invertible mapping h from P^2 to itself such that three points x_1, x_2, x_3 lie on the same line if and only if $h(x_1), h(x_2), h(x_3)$ also do.

THEOREM: A mapping $h : P^2 \to P^2$ is a projectivity if and only if there exist a non-singular 3x3 matrix $\mathbf{H}$ such that for any point in P^2 reprented by a vector x it is true that $h(x) = Hx$.

DEFINITION: Projective transformation (Homography)

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{18}$$

- Point transformation: $x' = \mathbf{H}x$
- Line transformation: $l' = \mathbf{H}^{-T}l$, where $\mathbf{H}^{-T} = (\mathbf{H}^{-1})^T$
- Conic transformation: $\mathbf{C}' = \mathbf{H}^{-T}\mathbf{C}\mathbf{H}^{-1}$
- Dual Conic transformation: $\mathbf{C}'^{*} = \mathbf{H}\mathbf{C}^{*}\mathbf{H}^{T}$
- The eigenvectors of the homography are points that get mapped to themselves (they don't move)

Hierarchy of 2D Transformations

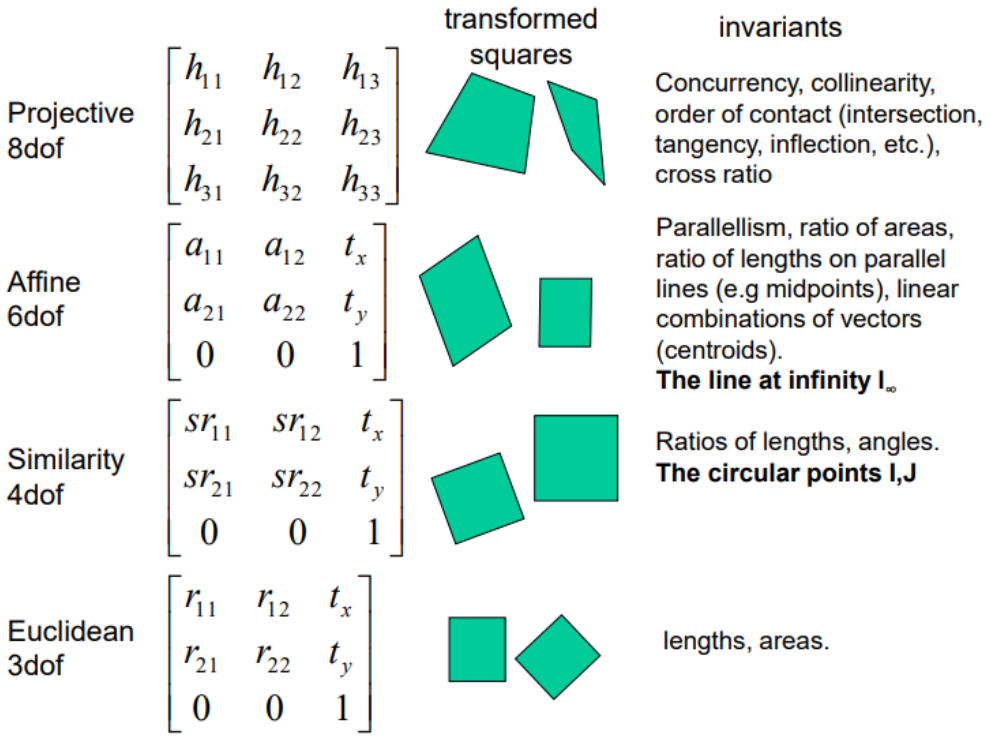


Figure 1: Hierarchy of 2D Transformations

Affine Rectification:

If we for example have a square tile on the floor and we take a photo of it (which is a projection), then the tile in the photo appears not square anymore and the sides are not parallel (see Figure 2). We can correct this by performing a rectification, such that the total transformation between original tile and rectified image is an affine transformation.

$$x_{rect} = \mathbf{H}_{\mathbf{P}}' x_{photo}, \text{ where } \mathbf{H}_{\mathbf{P}}' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} \text{ is the rectification matrix.} \tag{19}$$

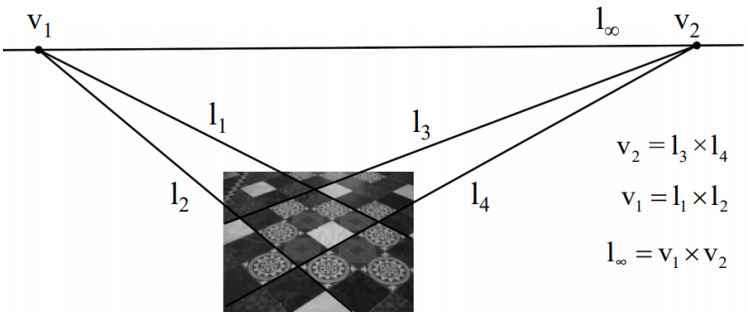


Figure 2: Rectification: Lines which in fact are parallel should intersect in an ideal point and the line that joins the intersection points of l_1, l_2 and l_3, l_4 is a line at infinity. However, in a photo the lines are not parallel anymore and the lines and points are not at infinity. We therefore correct this with a rectification matrix $H_P^{'}$ such that the line is again the line at infinity.

Circular points:

The circular points $I = \begin{pmatrix} 1 \\ i \\ 0 \end{pmatrix}, J = \begin{pmatrix} 1 \\ -i \\ 0 \end{pmatrix}$ are fixed points under the projective transformation $\mathbf{H}$ if and only if $\mathbf{H}$ is a similarity transformation.

$$I^{'} = \mathbf{H}_S I = \begin{bmatrix} scos\theta & ssin\theta & t_x \\ -ssin\theta & scos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 1 \\ i \\ 0 \end{pmatrix} = se^{i\theta} \begin{pmatrix} 1 \\ i \\ 0 \end{pmatrix} = I \tag{20}$$

The circular points can also be written as a conic dual $\mathbf{C}_\infty^*$.

$$\mathbf{C}_\infty^* = \mathbf{H}_S \mathbf{C}_\infty^* \mathbf{H}_S^T, \text{ where } \mathbf{C}_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{21}$$

- The dual conic $\mathbf{C}_\infty^*$ is a fixed conic under the projective transformation $\mathbf{H}$ if and only if $\mathbf{H}$ is a similarity transform.

3D Transformations:

Of course we can not only transform points, lines and conics in 2D, but we can also transform in 3D.

- Point transformation: $X^{'} = \mathbf{H}X$
- Plane transformation: $\pi^{'} = \mathbf{H}^{-T}\pi$
- Quadric transformation: $Q^{'} = \mathbf{H}^{-T}Q\mathbf{H}^{-1}$
- Dual Quadric transformation: $Q^{'*} = \mathbf{H}Q^*\mathbf{H}^T$

Hierarchy of 3D Transformations

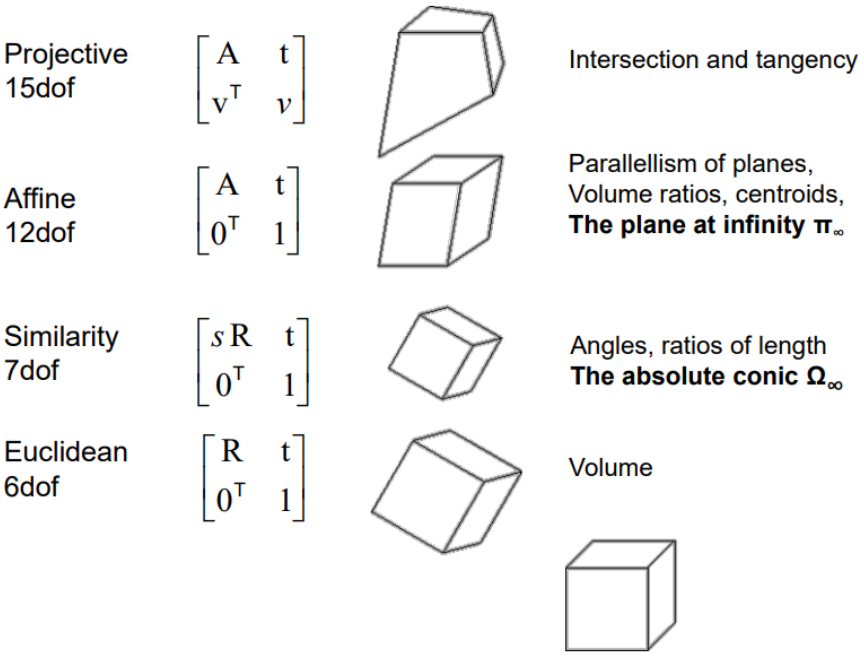


Figure 3: Hierarchy of 3D Transformations

The plane at infinity π_∞:

$$\pi'_\infty = \mathbf{H}^{-T}\pi_\infty = \begin{bmatrix} \mathbf{A}^{-T} & 0 \\ -t^T\mathbf{A}^{-T} & 1 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \pi_\infty \tag{22}$$

- The plane at infinity π_∞ is a fixed plane under a projective transformation $\mathbf{H}$ if and only if $\mathbf{H}$ is an affinity.

The absolute conic and dual quadric:

The absolute conic Ω_∞ is a point conic on the plane at infinity π_∞ that satisfies $X_1^2 + X_2^2 + X_3^2 = 0$ and $X_4 = 0$.

- The absolute conic Ω_∞ is a fixed conic under the projective transformation $\mathbf{H}$ if and only if $\mathbf{H}$ is a similarity transformation.

The absolute conic can also be represented in the dual space as the absolute dual quadric $\Omega^*_\infty = \begin{bmatrix} I & 0 \\ 0^T & 0 \end{bmatrix}$

- The absolute dual quadric Ω^*_∞ is a fixed quadric under the projective transformation $\mathbf{H}$ if and only if $\mathbf{H}$ is a similarity transformation.

3. Camera Models and Calibration

Vanishing points:

Parallel lines, such as the sides from a road or the walls of a building do not neccesarily appear parallel on a photo but instead these lines meet at a vanishing point as seen in Figure 4.

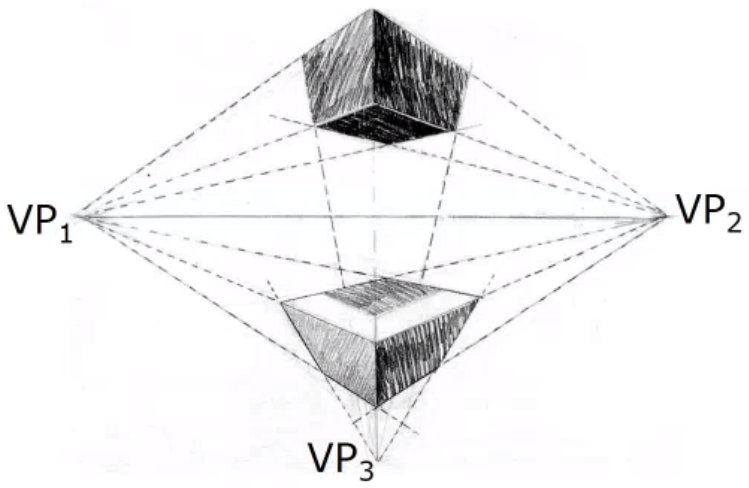


Figure 4: Vanishing points

Geometric properties of projection from 3D into 2D:

- Points to to points
- Lines go to lines
- Planes go to whole image or half-plane
- Polygons go to polygons
- Degenerate cases: The line through focal point yields a point, Plane through focal point yields a line

Pinhole camera model:

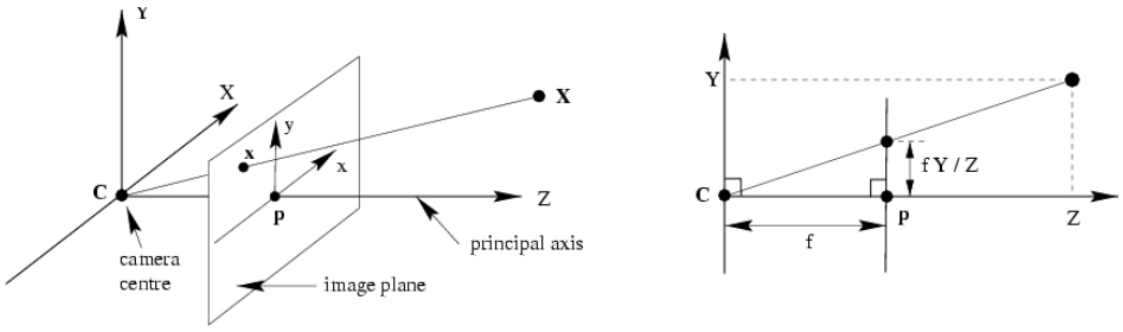


Figure 5: Pinhole camera model. f: focal length

The projection from the 3D world into the 2D image plane is $(X, Y, Z) \rightarrow (fX/Z, fY/Z)$, which can be written as a linear projection in homogeneous coordinates:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = PX = x \tag{23}$$

Projection matrix:

The matrix that projects 3D points into the 2D image plane is the so called projection matrix **P**. We can split **P** into the camera intrinsic matrix **K** and the extrinsics **R** and **t**:

$$x = PX = K \begin{bmatrix} R & t \end{bmatrix} X = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & -RC \end{bmatrix} X, \text{ with} \tag{24}$$

α_i: focal length scaled with width/height of image
s: skew factor (if the camera image would not be rectangular)
p_i: offset from camera origin to image origin
R: rotation matrix from world to camera
$t = -RC$: translation from camera origin to world origin expressed in the camera frame. (C: Camera position expressed in world frame)

- **P** has 11 DOF. (5 intrinsic parameters, 3 rotation parameters, 3 translation parameters)
- In reality, cameras are never perfect and have several distortion effects, where the radial distortion has the biggest effect. The equation including radial distortion is $x = KR_{dist} \begin{bmatrix} R & t \end{bmatrix} X$, where $R_{dist} = (1 + K_1(x^2 + y^2) + K_2(x^2 + y^2)^2 + ...)$

Direct Linear Transform (DLT):

If we have i correspondences between 2D points on an image and 3D points in the world, we can compute the projection matrix **P** with the DLT algorithm:

$$x_i = PX_i \Rightarrow [x_i]_\times PX_i = 0 = \begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0^T & -x_i X_i^T \\ -y_i X_i^T & x_i X_i^T & 0^T \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = A_i' \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = 0, \tag{25}$$

where $P_1 = (P_{11}, P_{12}, P_{13}, P_{14})^T$, $P_2 = (P_{21}, P_{22}, P_{23}, P_{24})^T$, $P_3 = (P_{31}, P_{32}, P_{33}, P_{34})^T$. The matrix A' above has rank two, which means we can just drop one of the rows:

$$\begin{bmatrix} 0^T & -w_i X_i^T & y_i X_i^T \\ w_i X_i^T & 0^T & -x_i X_i^T \end{bmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = A_i p = 0 \tag{26}$$

- Since P has 11 DOF and we have two independent equations per correspondence equation (eq. 26), we need $5.5 \sim 6$ correspondence point pairs. We can stack their equations in the form of $Ap = 0$, with $A = (A_1, A_2, ..., A_6)^T$
- The p vector can then be computed by performing a singular value decomposition (SVD) and taking the right nullspace.

– Degenerate cases: All points lie on a plane, all points lie on a twisted cubic

Gold Standard Algorithm:

In reality, when we try to calibrate a camera there are a few things we have to consider. Firstly, we have to normalize our correspondence points for **numerical stability**. Also, the collected 2D-3D correspondences of course do not match perfectly. It therefore makes sense to use the DLT result as an **initial solution** and then **optimize** it. In the end, we again have to **denormalize** the found result to optain the real P. The Gold Standard Algorithm goes as follows:

1. **Normalization**

$$\widetilde{x} = Tx = \begin{bmatrix} \sigma_{2D} & 0 & \bar{x} \\ 0 & \sigma_{2D} & \bar{y} \\ 0 & 0 & 1 \end{bmatrix}^{-1} x, \tag{27}$$

where σ_{2D}: mean distance to the origin, $c_{xy} = (\bar{x}, \bar{y})$: centroid.

$$\widetilde{X} = UX = \begin{bmatrix} \sigma_{3D} & 0 & 0 & \bar{X} \\ 0 & \sigma_{3D} & 0 & \bar{Y} \\ 0 & 0 & \sigma_{3D} & \bar{Z} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} X, \tag{28}$$

where σ_{3D}: mean distance to the origin, $c_{XYZ} = (\bar{X}, \bar{Y}, \bar{Z})$: centroid.

2. **Direct Linear Transform**

$$\begin{bmatrix} 0^T & -w_1 X_1^T & y_1 X_1^T \\ w_1 X_1^T & 0^T & -x_1 X_1^T \\ ... & ... & ... \\ 0^T & -w_6 X_6^T & y_6 X_6^T \\ w_6 X_6^T & 0^T & -x_6 X_6^T \end{bmatrix} \widetilde{p} = 0 \Rightarrow \text{ SVD, right nullspace } \Rightarrow \widetilde{P} \tag{29}$$

3. **Minimization of geometric error**

Take initial solution from DLT and iteratively optimize $\widetilde{P}$ by minimizing the geometric error between the 2D points and the reprojected points.

$$\min_P \sum_i d(\widetilde{x}_i, \widetilde{P}\widetilde{X}_i)^2 \tag{30}$$

4. **Denormalization**

$$P = T^{-1}\widetilde{P}U \tag{31}$$

Conclusions from Lab 1:

– If unnormalized points are used, P matrix is scaled by a large factor (numerically unstable). However, no big effects can be seen in the results.

– More points correspondences are not necessarily better.

– Accuracy highly depends on which point correspondences were used. Spreading the points works best.

– Euclidian distance as an error is not perfect for minimizing (Maximizing accuracy). The error increased with more correspondences.

4. Feature Extraction

Local image descriptors are a tool for matching things between two images. We can use it for: Content based web image search, Detect objects in crowded scenes, Creating panorama images, Tracking, Reconstruction and many more.

The general approach of matching two images by their descriptors can be seen in Figure 6.

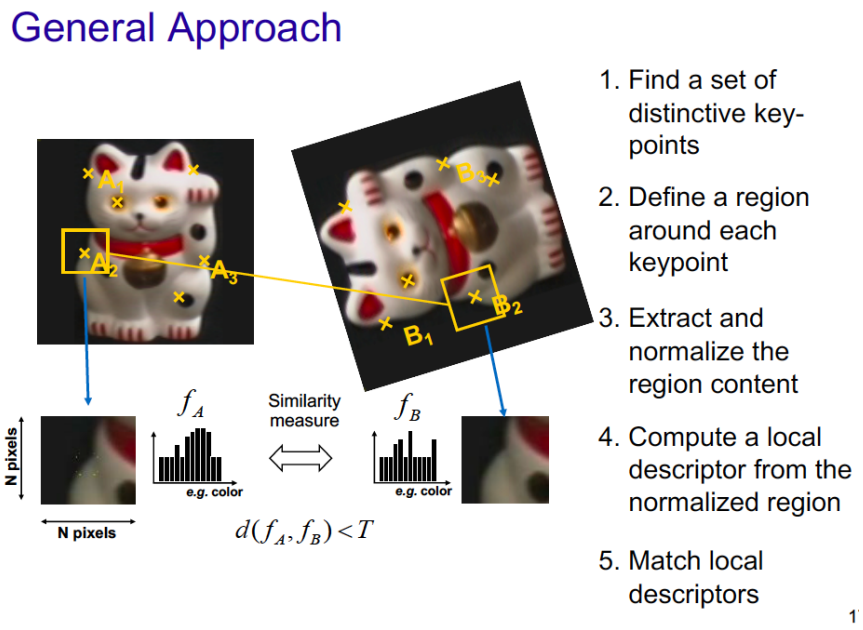


Figure 6: General approach of feature matching

Requirements for features:

- Region extraction needs to be **repeatable and accurate** (Ideally invariant to translation, rotation, scale and robust to lighting, noise, blur).
- **Locality:** Features are local, therefore robust to occlusion/clutter.
- **Quantity:** We need enough regions to cover one object.
- **Distinctiveness:** Regions should be "interesting", such as corners or edges.
- **Efficiency:** Close to real time performance

Examples for existing Detectors are: Hessian and Harris, Laplacian, DoG, Harris-/Hessian-Laplace, Harris-/Hessian-Affine, EBR IBR, MSER, Salient Regions, **SIFT** (very important), ...

In the last year, machine learning became more important. However, for many usecases (accurate geometric recovery, efficiency, no training), SIFT remains unmatched. Future algorithms will combine Deep learning and keypoint matching.

Harris Detector:

Let's say we have identified a meaningful local feature in an image, such as a corner of an object. We can then define a window $(x, y) \in W$ around it. If we shift this window by some pixels (u, v), the sum of squared distances (SSD) between the pixel intensities of the original and the shifted window should differ quite much. We can write this as

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2, \tag{32}$$

Where $E(u, v)$ is the SSD value between the pixel intensities $I(x, y)$ of the two windows and $w(x, y)$ is some window function that for example can be used for weighting of the pixels.
If now (u, v) is small, we can use Taylor expansion and rewrite (32) as

$$E(u, v) = \sum_{x,y} w(x, y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}. \tag{33}$$

If we now denote

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}, \tag{34}$$

we realize that the eigenvalues of M indicate how big the gradients in our image window are. This means that we can determine if the window we are looking at contains a corner or edge by just looking at the magnitude of the eigenvalues (both eigenvalues big $\rightarrow$ corner, one eigenvalue big, other small $\rightarrow$ edge).

We can now think of a function to score 'cornerness' that detects how strong the corner is that we have in our window pixel patch. In case of the Harris detector, the function looks as follows:

$$R = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(M) - \kappa \text{trace}^2(M) \tag{35}$$

To detect corners in an image, one can simply compute the Harris response value of the pixel patches and if R is bigger than a set threshold, then the pixel corresponding to this window is in fact a corner.

Remarks:

- The Harris detector uses grayscale images.
- The window function $w(x, y)$ is usually chosen to be a gaussian weighting, so that the result is **rotation invariant**.
- Harris detector is **not scale invariant**, since the window size/gaussian filter has a fixed size.
- κ is usually between $[0.04, 0.06]$
- Results are well suited for stereo image matching

An example **Harris Detector Algorithm** could look as follows:

1. Compute image gradients I_x, I_y
2. Compute square of derivatives I_x^2, I_y^2 (element-wise)
3. Apply gaussian filter $I_{i,filtered} = \text{imgaussfilt}(I_i)$
4. Compute Harris response R for each pixel. If $R(x, y) > thresh \Rightarrow$ corner
5. Perform non-maximum suppression (to avoid multiple corner responses for the same corner)

Scale invariant Region Selection:

The Harris detector defines point of interest. In order to compare these points to the points of another image, we need to compute the descriptor over a region around the found points. The problem here is that it is not clear how big this region should be. Depending on the region size, the descriptors could potentially look completely different even though their points do in fact match.

This can be solved by designing a scale invariant **signature function** on the region. For a point in an image, we can consider the signature function as a function of region size. Followingly, the region size for which the signature function has its maximum should be independent of the image scale, as seen in Figure 7.

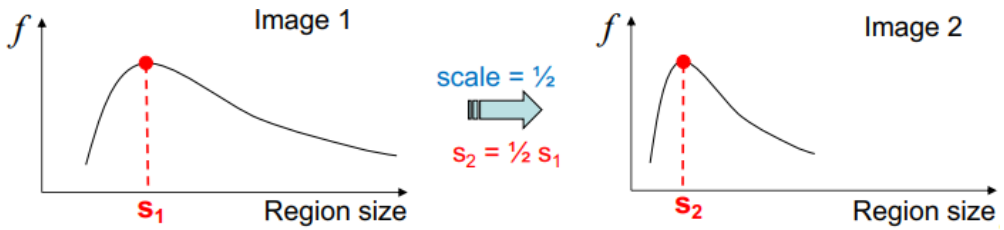


Figure 7: The region size S_1, S_2 for which the signature functions have their maximum are invariant to the image scale. Note that the scale invariant region size S is found for each image independently.

- A possible choice for such a signature function is the Laplacian-of-Gaussian (LoG, "blob" detector). The region scale where the laplacian response maximizes is what we call the *characteristic scale*. Instead of the real Laplacian of Gaussian, the Difference of Gaussians can be used as an approximation, which for example is what is done in SIFT. (advantages: no need to compute 2nd derivatives, gaussians are computed anyway)
- Short summary:
  Given: Two images of same scene with large scale difference. Goal: Find the same interest points independently of the image scale in both images. Solution; Search for maxima of suitable signature functions in scale and in space (over image)

Local Descriptors

Once we have detected points of interest as well as the optimal region size, we can finally think about how to describe these regions in order to match them between images. There are countless possibilities to construct a descriptor:

- Pixel intensities (+: simple, good for templates, -: sensitive to illumination/geometry change)
- Color histograms (+: rotation invariant, -: shape does not matter)
- Spatial histograms (+: shape matters (a bit), -: not rotation invariant)

A more elaborate method is called **Orientation normalization**. Here, one computes the gradient of the pixel patch and put the gradients inside bins of a histogram. Then, the dominant orientation is used to rotate the whole patch to normalize it.

SIFT (Scale Invariant Feature Transform):

SIFT is an extraordinarily robust matching technique. It describes both a detector as well as a descriptor.

- can handle changes in viewpoint up to 60 degrees out-of-plane rotation
- can handle significant changes in illumination
- fast and efficient (real time)
- Lots of code available

Computation of SIFT descriptor:

1. divide path into 4x4 sub-patches (16 cells).
2. Compute histogram of gradient orientations (8 bins) for all pixels inside each cell $\Rightarrow$ descriptor size $4 \times 4 \times 8 = 128$.
3. Find dominant direction of whole patch.
4. Rotate patch according to the dominate angle.

SURF:

- efficient computation by 2D box filters and integral images (6 times faster than sift!)
- equivalent quality for object identification
- GPU implementation available

5. Optical Flow & Particle Filters

As we move a camera through the world, the objects in the image of course move as well. If we now project their velocities into the image plain, we get what we call *Optical Flow*.

Figure 8: Optical flow: projection of object velocities onto image plane.

- Optical Flow = 2D velocity field describing the apparent motion between image frames
- Motion Field = 2D motion field representing the projection of the 3D motion of points in the scene onto the image plane
- Difference: Motion field is the projection of the *real* velocities, optical flow is the velocity field of the *apparent* motion of the pixels.
  Imagine a video of a completely dark object moving in a completely dark scene. On the video you would see nothing, which means we don't have any optical flow but the motion field would still represent the real motion of the object.

Given two consecutive image frames, the goal is now to estimate the motion of each pixel. Hereby we assume **brightness and color constancy**. This allows for pixel to pixel comparison (not image features). Additionally we assume **small motions**, which allows us to linearize the brightness/color constancy constraint.

On a high level, the approach now is to look for nearby pixels with the same color between two consecutive image frames. For a really small space-time step it then must hold that

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t), \tag{36}$$

which means that the brightness of a certain pixel between two consecutive image frames is the same. This further implies the **Brightness Constancy Equation**:

$$\frac{\delta I}{\delta x}\frac{dx}{dt} + \frac{\delta I}{\delta y}\frac{dy}{dt} + \frac{\delta I}{\delta t} = 0 = I_x u + I_y v + I_t, \tag{37}$$

where I_x, I_y represent the image gradients, u, v are the flow velocities, I_t is the temporal gradient. I_x, I_y can be computed from spacial difference and I_t for example with forward euler difference between the images. But how do we get u, v?

To get u, v, which are two unknowns, we need two equations. One equation is the Brightness Constancy Equation and the other one comes comes from the method we choose to use: **Smooth Flow** (Horn-Schunck) or **Constant Flow** (Lucas-Kanade).

Constant Flow:

- Assumption: Flow is locally smooth
- Assumption: Pixels in a local patch have the same displacement u, v

From the assumptions we can deduce that the pixels of local patch in the image, e.g. a 5x5 pixel patch, all have the same (constant) flow. From a 5x5 pixel patch we have 25 Brightness Constancy Equations:

$$\begin{bmatrix} I_x(p_1)u + I_y(p_1)v \\ I_x(p_2)u + I_y(p_2)v \\ ... \\ I_x(p_{25})u + I_y(p_{25})v \end{bmatrix} = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ ... \\ -I_t(p_{25}) \end{bmatrix} \tag{38}$$

Solving this for u, v is equivalent to solving

$$A^T A x = \sum_{p \in P} \begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \sum_{p \in P} \begin{bmatrix} -I_x I_t \\ -I_y I_t \end{bmatrix} = A^T b \tag{39}$$

by using the pseudo-inverse:

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b \tag{40}$$

For this to be solvable, it must hold that $A^T A$ is invertible, $A^T A$ should not be too small, and λ_1/λ_2 should not be too small (λ_1 is the bigger eigenvalue).

We have already seen the above matrix in the Harris Corner detector, what is the correlation?
Remember that corners are where λ_1, λ_2 are big. This is where Lucas-Kanada optical flow works best! In other words, Corners are a good place to compute optical flow. If we have no corners (for example in the barber's pole illusion), the optical flow and motion field might differ a lot.

Summary:

- Assumption: All pixels of a local image patch are assumed to have the same optical flow u, v.
- Lucas-Kanada optical flow is a local method and works with sparse image patches
- The equation to solve for u, v is equation (40).
- Lucas-Kanada optical flow works best on corners.

Smooth Flow:

A more global approach is the Horn-Schunck optical flow. Since most objects in the world are rigid or deform elastically and are moving together coherently we expect the optical flow fields to be smooth, rather than patch-wise constant. The idea is to enforce brightness constancy and a smooth flow field

- Assumption: Small motion
- Assumption: Brightness Constancy*

* Whereas in the constant flow method we had brightness constancy as a hard constraint, in the Smooth Flow approach we allow for the brightness value to not be exactly zero but very close. In fact, instead of setting $I_x u + I_y v + I_t = 0$, we just try to minimize

$$\min_{u,v} [I_x u_{i,j} + I_y v_{i,j} + I_t]^2, \tag{41}$$

where i, j denote the pixel coordinates.

In addition, we don't want the flow to vary too much between neighbor pixels:

$$\min_u (u_{i,j} - u_{i+1,j})^2, \ \min_v (v_{i,j} - v_{i,j+1})^2 \tag{42}$$

The whole optimization problem can then be formulated as

$$\min_{u,v} \sum_{i,j} E_s(i,j) + \lambda E_d(i,j), \tag{43}$$

where $E_d(i,j) = [I_x u_{i,j} + I_y v_{i,j} + I_t]^2$,
$E_s(i,j) = \frac{1}{4}[(u_{i,j} - u_{i+1,j})^2 + (u_{i,j} - u_{i,j+1})^2 + (v_{i,j} - v_{i+1,j})^2 + (u_{i,j} - u_{i,j+1})^2]$ and λ is a weighting factor.

After some mathematical hocuspocus, we can solve this minimization problem by iterating with the update formulas for $u_{i,j}, v_{i,j}$:

$$u = u - \frac{I_x u + I_y v + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x \tag{44}$$

$$v = v - \frac{I_x u + I_y v + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y \tag{45}$$

The Horn-Schunck Optical Flow Algorithm then looks as follows:

1. Precompute image gradients I_x, I_y
2. Precompute temporal gradients I_t
3. Initialize flow field: $u_{i,j} = 0, v_{i,j} = 0$
4. While not converged: $u = u - \frac{I_x u + I_y v + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x$, $v = v - \frac{I_x u + I_y v + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$

Bayesian Filtering:

The moving object of interest is characterized by an underlying state $\mathbf{X}$, which consists of the true parameters we care about. The measurement $\mathbf{y}$ is the noisy observation that results from the underlying state $\mathbf{X}$. In each timestep, we are interested in knowing the new state $\mathbf{X_t}$, and we do this by taking into account the newly arrived measurement $\mathbf{y_t}$ and finding the most likely state $\mathbf{X_t}$.

What we have to reach this goal are all the past measurements as well as some knowledge about the dynamics of state transitions. The steps of tracking are:

1. **Prediction:**
   What is the next state distribution $P(X_t|y_0, ..., y_{t-1})$ given the current belief and our knowledge about the system dynamics?

$$P(X_t|y_0, ..., y_{t-1}) = \int \underbrace{P(X_t|X_{t-1})}_{\text{dynamic model}} \underbrace{P(X_{t-1}|y_0, ...y_{t-1})}_{\text{prior belief}} dX_{t-1} \tag{46}$$

2. **Correction:**
   Compute an updated estimate of the state $P(X_t|y_0, ..., y_t)$ from the made prediction and the newly arrived measurement.

$$P(X_t|y_0, ..., y_t) = \frac{\overbrace{P(y_t|X_t)}^{\text{observation model}} \overbrace{P(X_t|y_0, ...y_{t-1})}^{\text{prior belief}}}{\int P(y_t|X_t) P(X_t|y_0, ...y_{t-1}) dX_t} \tag{47}$$

3. Repeat at next timestep

In this sense, tracking can be seen as the process of propagating the posterior probability distribution of the state given measurements across time. The assumptions for the above stated steps are:

- Only the immediate past matters: $P(X_t|X_0, ..., X_{t-1} = P(X_t|X_{t-1})$, Dynamic Model
- Measurements depend only on the current state: $P(y_t|X_0, y_0, ...X_{t-1}, y_{t-1}, X_t) = P(y_t, X_t)$, Observation Model

We have one problem now: The state distribution $P(X_t|y_0, ..., y_t)$ might be insanely complex which makes it very difficult to come up with a meaningful expression for the state distribution. An idea to work around this is particle filtering.

Particle Filtering:

The key idea behind particle filtering is to represent the state distribution in a non-parametric way, by working with samples (particles) from the distribution rather than with the distribution itself. If the number of these particles is chosen large enough, the density distribution of the particles is equivalent to the probability distribution we are looking for.

For the Prediction step, we progagate the sampled points (particles) from the prior density for the state with the dynamic model ($\rightarrow$*a priori* particles).
In the Correction step, we weight the a priori particles according to the newest observation. From the weighted density we then draw a new set of particles by resampling ($\rightarrow$ *posteriori* particles).

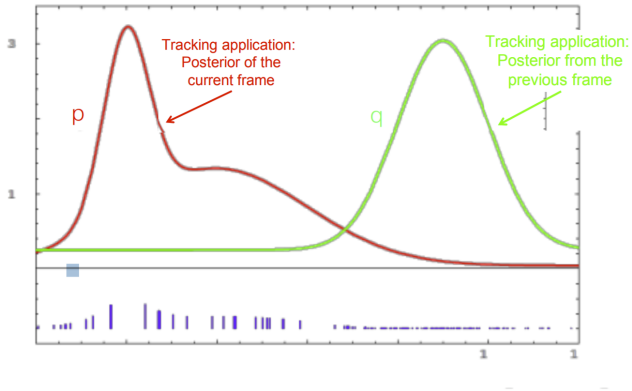


Figure 9: Importance sampling: p is the state probability distribution, which we don't know. Instead, we start from a proposal distribution q (either freshly initialized or the most recent belief from the prior timestep). From q we then sample some particles (purple data markers) and assign each of them a weight (height of the purple marker) according to how likely they are, given the current observation. Note that we have many particles on the right side, since this is where q peaks, but they have very little weight. However, on the left side we have only a few particles but with very large weights. Once we have our weighted particles, we can simply draw a new set of particles from the weighted samples. The newly achieved particle set will already look much more similar to p.

The slides after this point got quite confusing, the lecturer ran out of time and stopped the lecture without

finishing on the topic. However, I recommend to have a closer look at the code of *Lab03 - Particle Filter*, it gets pretty clear there.

Tracking with Particle Filtering

1. Initial sampling of initial distribution (prior)
2. Calculate weights based on observation
3. Resample using the new weights (Importance sampling)
4. Repeat until convergence

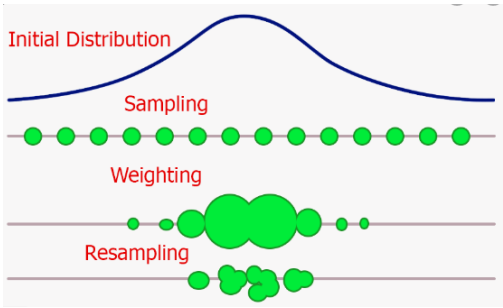


Figure 10: Steps of particle filtering.

6. Multi-View Geometry & Structure from Motion

Two-view geometry

- **Correspondence geometry**: Given an image point x in the first image, how does this constrain the position of the corresponding point x' in the second image?
- **Camera geometry (motion)**: Given a set of corresponding image points ($x_i \to x_i'$) what are the camera matrices P and P' for the two views?
- **Scene geometry (structure)**: Given corresponding image points ($x_i \to x_i'$) and camera matrices P, P', what is the position of X in space (real-world coordinates)?

Epipolar geometry

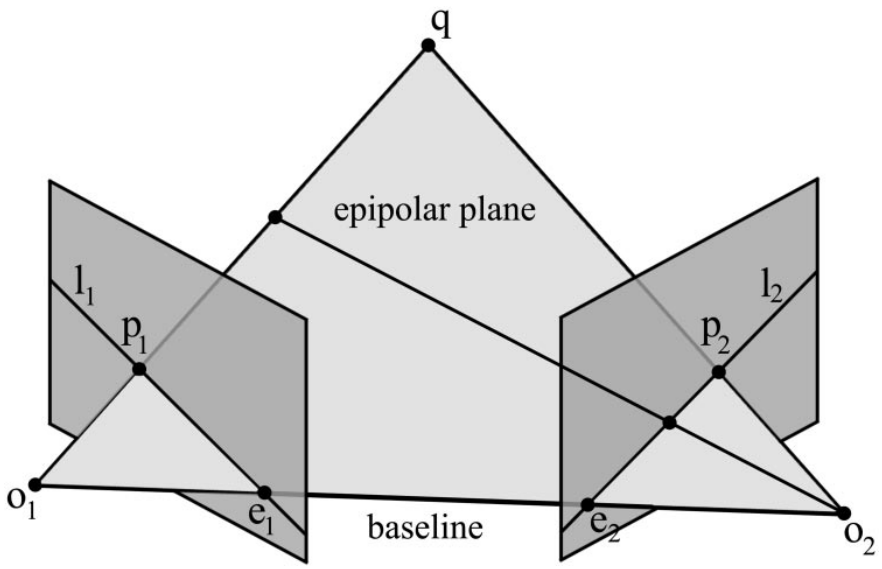


Figure 11: Epipolar geometry with e = **epipoles,** l = **epipolar lines, epipolar plane, baseline,** O = **position of camera/observer.**

Epipoles = projection of projection center in other image

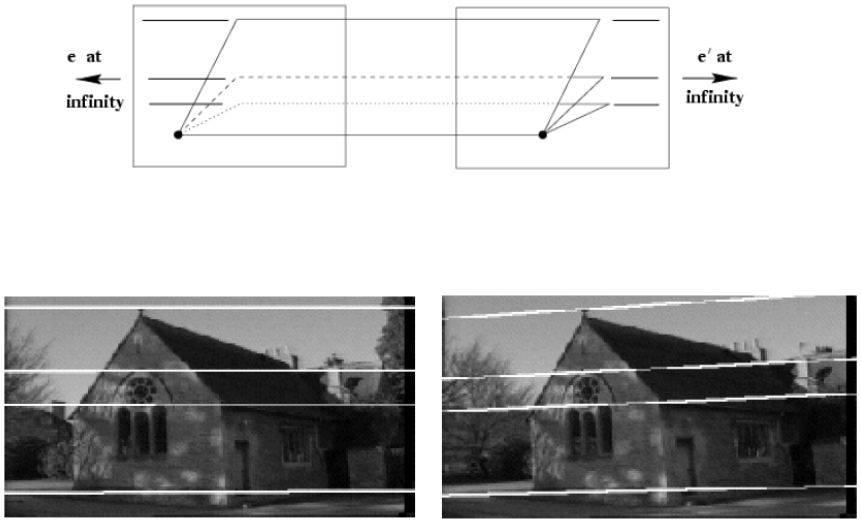


Figure 12: Epipolar geometry with camera in **parallel motion**.

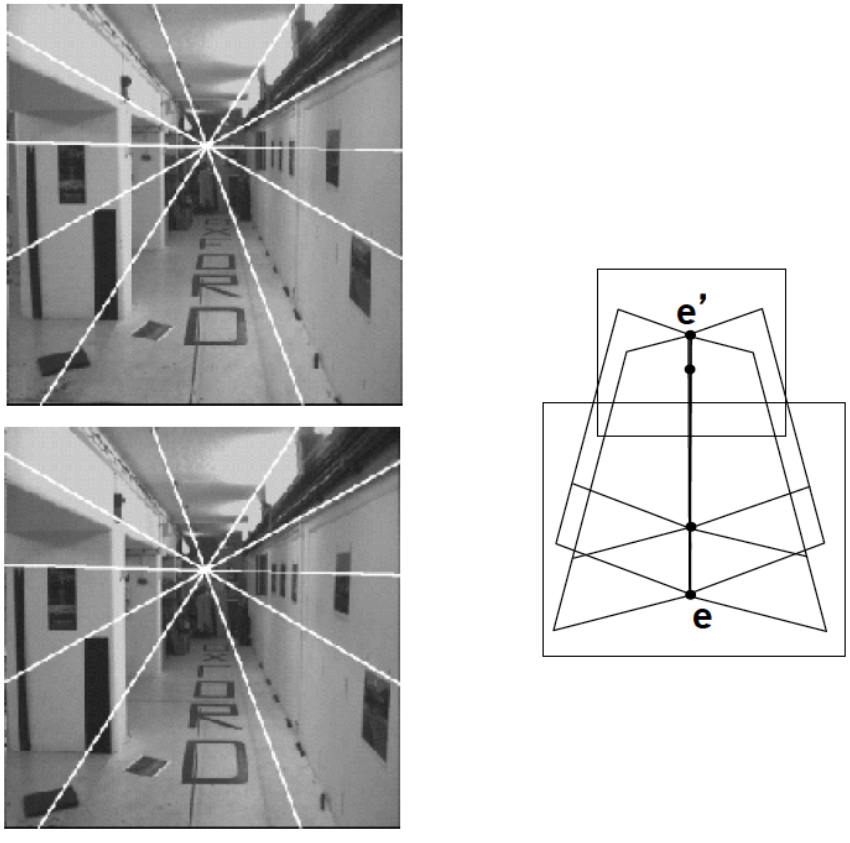


Figure 13: Epipolar geometry with camera in **forward motion**.

Fundamental matrix F (`https://www.youtube.com/watch?v=auhpPoAqprk`)

7 DOF (translation, rotation, calibration parameters)
Shape: 3×3 matrix
Allows projective mapping of points to lines

Satisfies for all correspondences: $x'^T F x = 0$ Epipoles e: $Fe = 0$ Epipolar lines: $l' = Fx$ and $l = F^T x'$

Normalized Eight-point algorithm

1. Select 8 point correspondences.
2. Normalize and shift points so that the mean coordinates are at the center of the image and the scale of the image is [-1, 1]
3. Form matrix M by stacking 8 rows.

$$M * vec(F) = \begin{bmatrix} xx' & xy' & x & yx' & yy' & y & x' & y' & 1 \end{bmatrix} * vec(F) = 0 \tag{48}$$

4. SVD on M => U, S, V
5. Select column of V that corresponds to smallest eigenvalue in S (last column) and reorder accordingly. `F_normalized = [last_col_V(1:3,1) last_col_V(4:6,1) last_col_V(7:9,1)];`
6. After denormalizing and rescaling the resulting matrix we can enforce singularity contraint $det(F) = 0$. This can be achieved by decomposing F to U, S, V and setting $S(3,3) = 0$. By calculating $U * S * V$ we obtain the final fundamental matrix $\hat{F}$.

If only 7 correspondence points would be used the algorithm would result in a set of solutions $F = F_1 + \lambda F_2$.

Essential matrix
5 DOF => rotation(3) and translation(2)
Satisfies $x'^T E x = 0$
Can be used with the 5-point algorithm **but only if the calibration matrix (intrinsic property K) is known**.

OPEN CHALLENGES in 'Structure from motion':

Large scale structure from motion
– Complete building
– Complete city
Life long mapping
– Merging maps
– Detecting change
– Avoid degradation over time

7. Model Fitting (RANSAC, EM, ...)

Hough transform

Straight lines can be represented in the (θ, ρ) space with the following notation. Each point in xy-space will have a curve in (θ, ρ)-space which corresponds to all the possible lines through the point in xy-space.

$$x \cos\theta + y \sin\theta = \rho \tag{49}$$

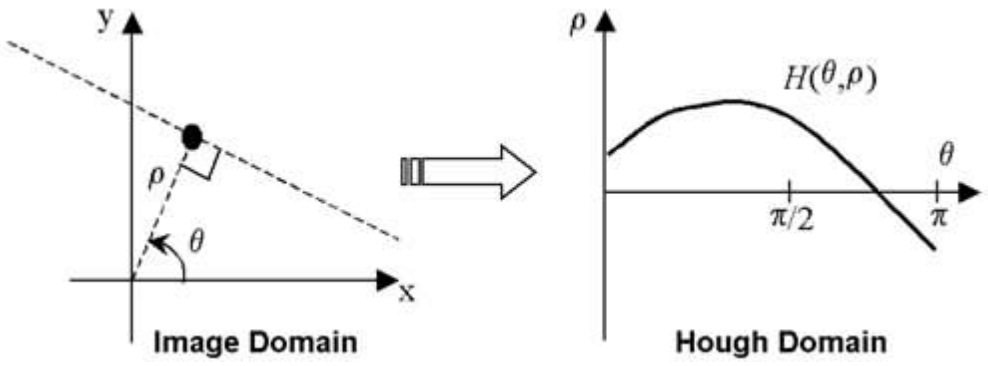


Figure 14: Straight line representation in (θ, ρ) space.

Interpretation of Hough Images

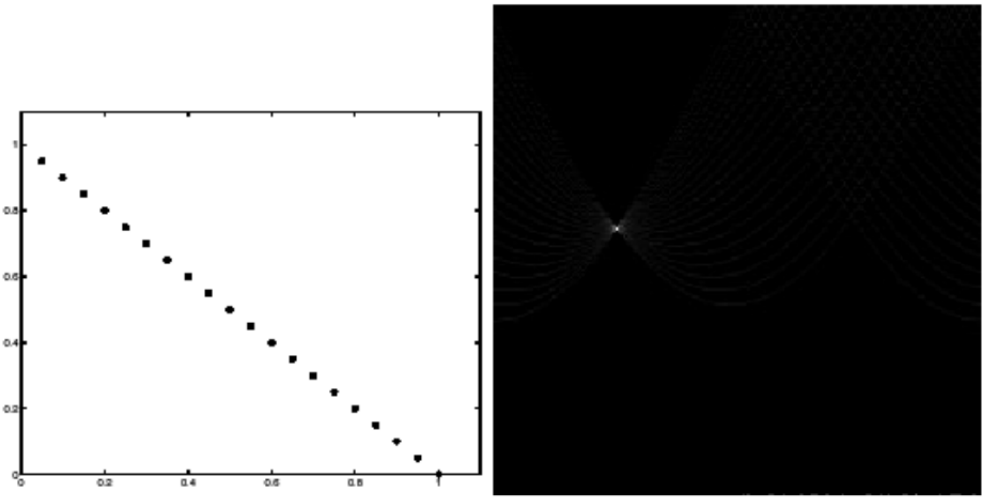


Figure 15: Each node (maximum intensity) in the Hough domain represents a straight line in the image.

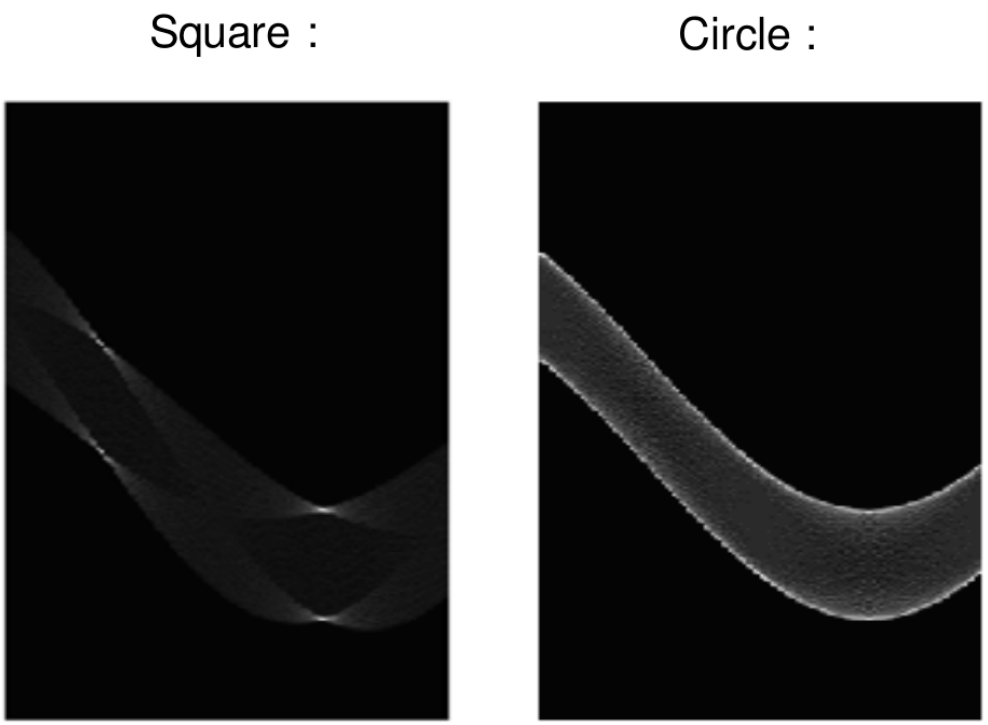


Figure 16: Square and circle representation in Hough Domain.

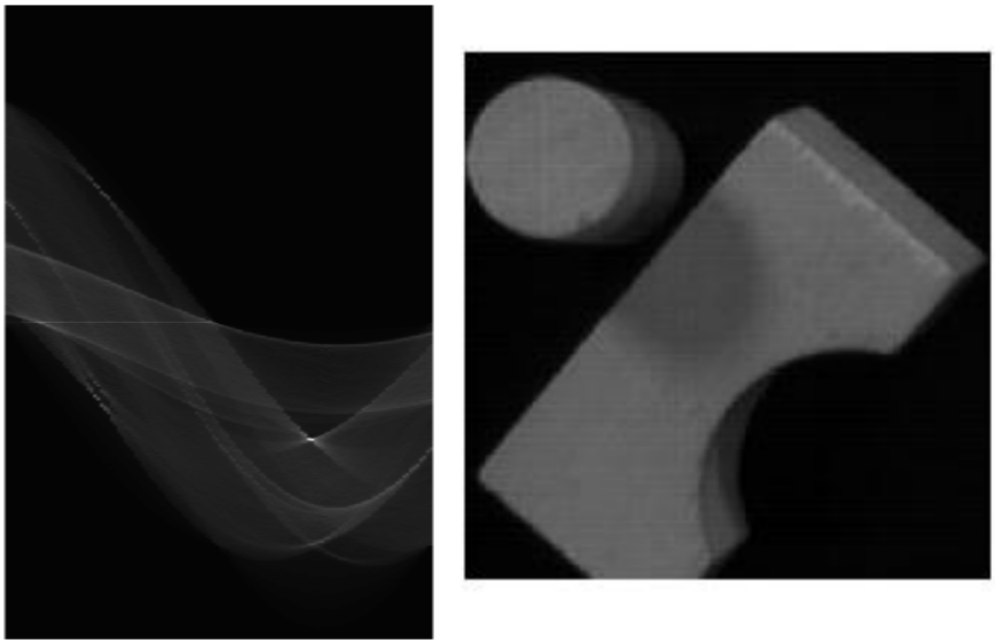


Figure 17: Objects representation in Hough Domain.

Difficulties with the Hough transform:

- Sometimes hard to distinguish peaks
- Noise causes lines to be missed

Line fitting

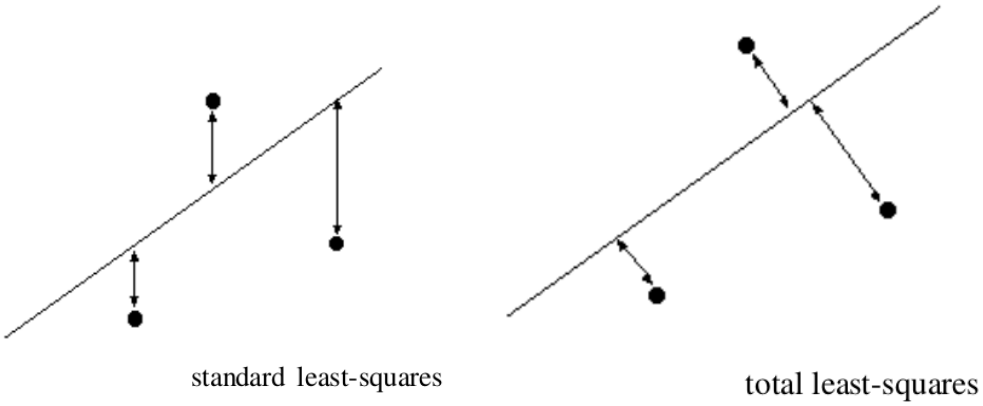


Figure 18: Choice of model is important.

Strategy 1: Incremental line fitting

Algorithm: Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large.

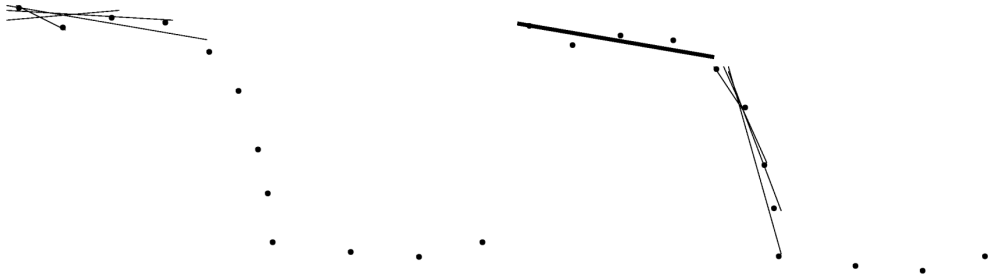


Figure 19: Fitting line and breaking the curve if residual is too large (point 5 or 6).

Strategy 2: K-meams fitting

Number of lines have to be known.

Algorithm: K-means line fitting by allocating points to the closest line (which was placed at random) and then refitting.

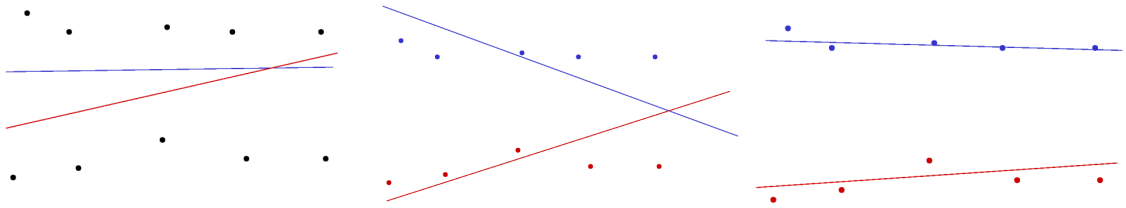


Figure 20: K-means line fitting.

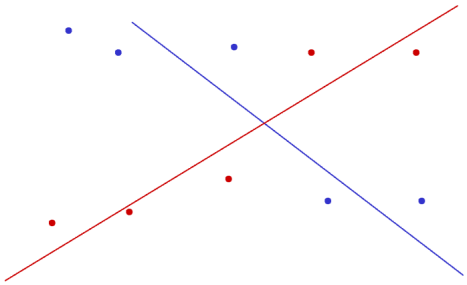


Figure 21: Problem with K-means line fitting: If initial placement of lines is bad, cases like this could happen.

To increase robustness the following approaches can be used:
– Least square can not handle outliers, because it assumes Gaussian error distribution.

M-estimators The euclidian distance (quadric ρ function), which is used in least squares, gives too much weight to outliers. To counteract this effect the following robust norm should be used:

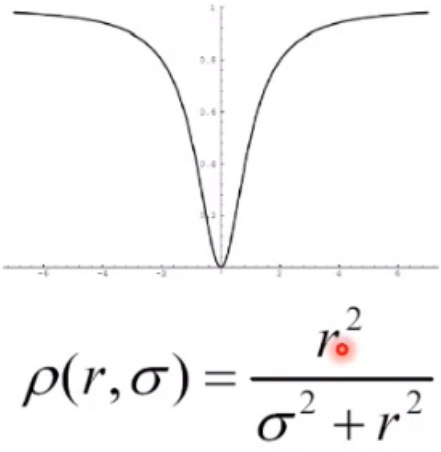


Figure 22: Robust norm weighs outliers only with a fixed weight of 1. However, the choice of σ is critical.

RANSAC (RANdom SAmpling Consensus)

– Pick sample at random
– Fit model to that
– If point is close to model it is part of the signal otherwise it is noise
– Resample and refit iterative

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
  n — the smallest number of points required
  k — the number of iterations required
  t — the threshold used to identify a point that fits well
  d — the number of nearby points required
    to assert a model fits well
Until k iterations have occurred
  Draw a sample of n points from the data
    uniformly and at random
  Fit to that set of n points
  For each data point outside the sample
    Test the distance from the point to the line
      against t; if the distance from the point to the line
      is less than t, the point is close
  end
  If there are d or more points close to the line
    then there is a good fit. Refit the line using all
    these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

Figure 23: RANSAC Algorithm.

Choose the right distance threshold t: Choose t so probability for inlier is e.g. 0.95, (often empirically)

Choose number of samples N: Choose N that, with probability p, at least one random sample is free from outliers.

$$((1-(1-e)^s)^N) = 1-p \tag{50}$$

$$N = \frac{log(1-p)}{log(1-(1-e)^s)} \tag{51}$$

Adaptively determining number of samples
Probability of being an outlier e is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found.

`while` N `> sample_count repeat`

- `Choose a sample and count the number of inliers`
- `Set` e `= 1 - (number of inliers) / (number of total points)`
- `Recompute` N `with new` e `using equation 51`

Fitting curves other than lines? In practice it is hard to compute distance between point and curve

EM: Expectation Maximization

EM algorithm is an iterative method to find (local) maximum likelihood estimates of parameters in models. (Parameter of lines, image segmentation) can solve "Missing variable problems".

Issues with EM:
– Local maxima: No guarantee to be at right maximum
Starting: k-means to cluster the points is often a good idea (initial guess)

Segmentation with EM
Segmentation in a video: Look at motion fields of different frames

Algorithm:
The EM Segmentation uses an approach to represents every segment as a Gaussian distribution with mean μ_k and covariance Σ_k. A Gaussian mixture model is created by weighting the Gaussians with parameter α_k. However, this approach requires a known number of segments K. A Gaussian mixture is defined by the following formula:

$$p(x_l \mid (\alpha, \mu, \Sigma)) = \sum_{k=1}^{K} \alpha_k p(x_l \mid (\mu_k, \Sigma_k)) \tag{52}$$

$$p(x_l \mid (\mu_k, \Sigma_k)) = \frac{1}{(2\pi)^{n/2}|\Sigma_k|^{1/2}} \exp(-\frac{1}{2}(x_l - \mu_k)^T \Sigma_k^{-1}(x_l \mu_k)) \tag{53}$$

First, the parameters for the K Gaussian mixtures get initialized with random values within the range of the L^*a^*b and uniform weights for α. With a series of consecutive Expectation and Maximization steps the parameters for the Gaussian mixtures can then be determined. For each pixel, the **probability to be included in a segment** is calculated in the **Expectation step**. These probabilities get then used to **maximize the expectation of the complete log likelihood** in the Maximization step. The iterations are stopped when the difference between two consecutively generated μ parameters are smaller than a certain threshold.

Lab conclusions:
RANSAC: The more movement between two images, the more wrong matches and thus more iterations are needed. Also the higher the amount of random samples N the higher the number of iterations M.
EM with image Segmentation: EM was considerably faster in computation time than Mean-Shift. However, the need to specify the number of segments beforehand is a big downside of the EM algorithm.

8. Stereo Matching & Multi-View Stereo

Applications stereo vision: Mars Rover

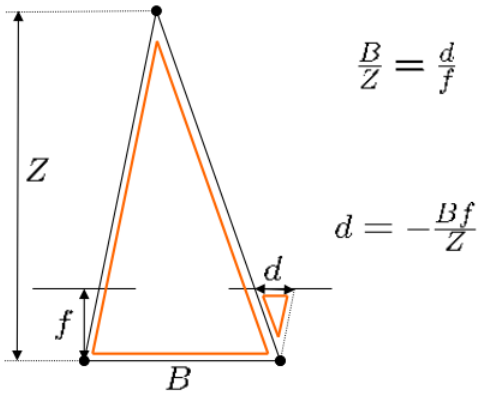


Figure 24: Stereo geometry.

- Z = distance between camera and object
- f = focal length
- B = distance between cameras
- d = **disparity**, difference of pixels of object in image 1 and image 2.

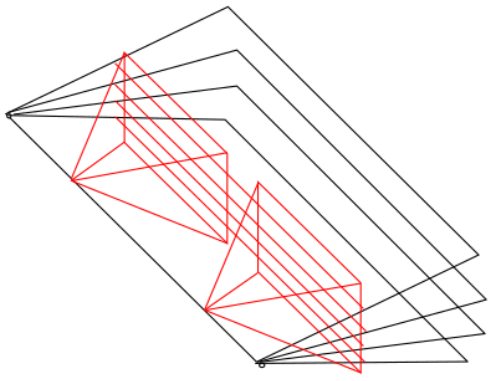


Figure 25: Epipolar lines are parallel (red) in a stereo setup.

Feature matching in stereo images

1. epipolar constraints = parallel
2. ordering constraints = if no occlusions the feature should have same order from left to right
3. Uniqueness constraints = One feature corresponds to maximum one feature

4. Disparity constraint = Surface of object does usually not have very deep holes. Thus we can assume that surface stays in a certain disparity band. Smoothness.

Disparity map encodes depth correspondences.
Results from experiments: Problems with finding disparity map of sky because no correspondence matching possible, dark areas like doors windows also problematic, matching of trees also difficult.

Disparity/Depth map computation:

Winner takes all method

1. First, rectify images to achieve parallel epipolar lines and epipoles at infinity.
2. One image was shifted by d pixels and the other image was subtracted pixelwise and squared. to obtain matrix of squared differences of intensities for each pixel.
3. Move window over matrix. For every pixel location the average of the differences within the window was computed.
4. The winner-takes-all method varies the shift distance d and keeps only the smallest average difference for each pixel.

Graph cut method = Graph labeling problem

1. The pixels represent the graph nodes and the disparities the labels
2. The total cost is formed by the cost of assigning a label to each pixel and the cost of assigning labels to the neighboring pixels.
3. The first cost is calculated by computing the sum of squared differences of image 1 and the by d pixels shifted image 2.
4. The second cost is higher if the neighboring pixels have different labels
5. total cost gets minimized using library GCMex.

Conclusion: Small window size already yield better result than Winner takes all. Few depth discontinuities.

9. Specific Object Recognition

Number of visual object categories: 10'000 to 30'000
the Buckingham palace? Identification *What's in the scene?* Semantic segmentation (Trees, Mountain, Building, People, Ground) *What type of scene is it?* Scene categorization (Outdoor Marketplace) *What are these people doing?* Activity Recognition

Category recognition:

- Find all the people
- Often within a single image
- Often 'sliding window'

Instance recognition:

- Is this face James?
- Find this specific famous building
- Often within database of images

Recognition challenges

- Variable viewpoint
- Variable illumination
- Variable scale
- Deformation (Different body poses of human)
- Occlusions
- Background clutter
- Intra-class variation (Different types of chairs)

History of ideas in recognition:
– General shape primitives (Cylinder, cone, etc.)
– By component (e.g. 3 cylinders + 1 cone = Lamp)
– Eigenfaces (Person recognition)
– Color histograms
– Sliding window approaches
– Local features
– Parts-and-shape models - Object is a set of parts with its relative locations (e.g Head = Hair + eyes + nose)
– Bags of features
– *Present:* Combined local and global methods, deep learning

Visual words
Indexing local features in high-dimensional descriptor feature space

Map high-dimensional descriptors to words by quantizing the feature space via clustering. Every cluster center represents a visual word.

Challenges for visual vocabulary formation:
– Vocabulary size, number of words
– Sampling strategy: where to extract features?
– Clustering algorithm
– Unsupervised (unlabeled training data) vs. supervised (labeled training data)

Use inverted index to allow fast lookups:

Word ID	Image ID
1	3
2	2,5
3	3
4	1

One big issue:
Key requirement: Sparsity. If most images contain most words, then we're not better off than exhaustive search (= Comparing visual word distribution of a query versus every page.)

Right vocabulary size:
– Too small: Visual words not representative of all patches
– Too big: Quantization artifacts

Computational efficiency = vocabulary trees

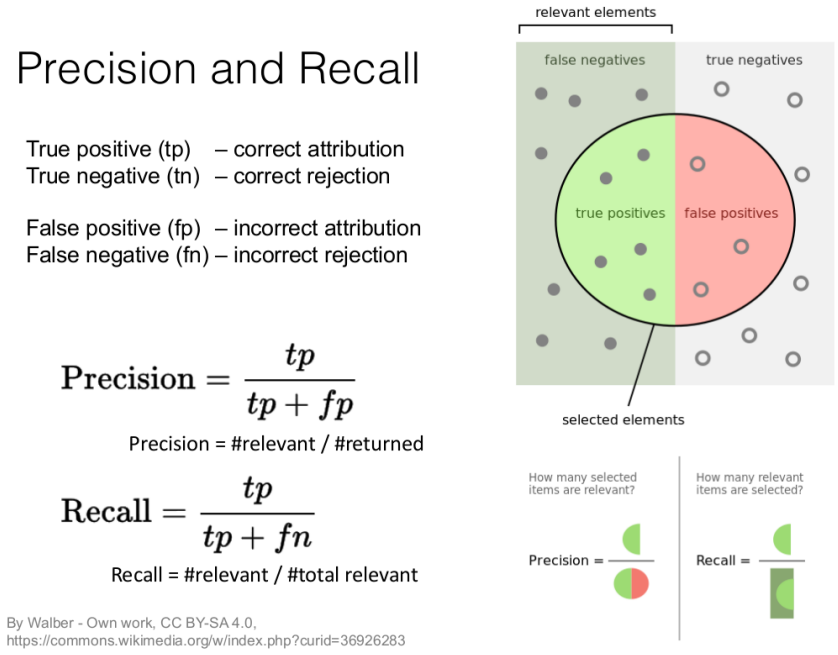


Figure 26: Precision and recall definitions. Used for scoring retrieval quality.

Spacial information of features on objects can be used to filter our bad matches. => Spacial verification

Recognition via alignment:
Pros:

- Effective for reliable features within clutter
- Great for matching specific instances

Cons:

- Not suited for category recognition

Estimating homography using RANSAC

- Get **four** point correspondences (randomly)
- Compute H (Homography) using **DLT**
- Count **inliers**
- Keep H if **largest number of inliers**

Summary:
Object instance recognition

1. Find keypoints, compute descriptors
2. Match descriptors
3. Fit transformation parameters
4. Return object if number of inliers $>$ T

Keys to efficiency in object recognition

- Use visual words
- Use inverted index

10. Recognition and Reconstruction of Humans

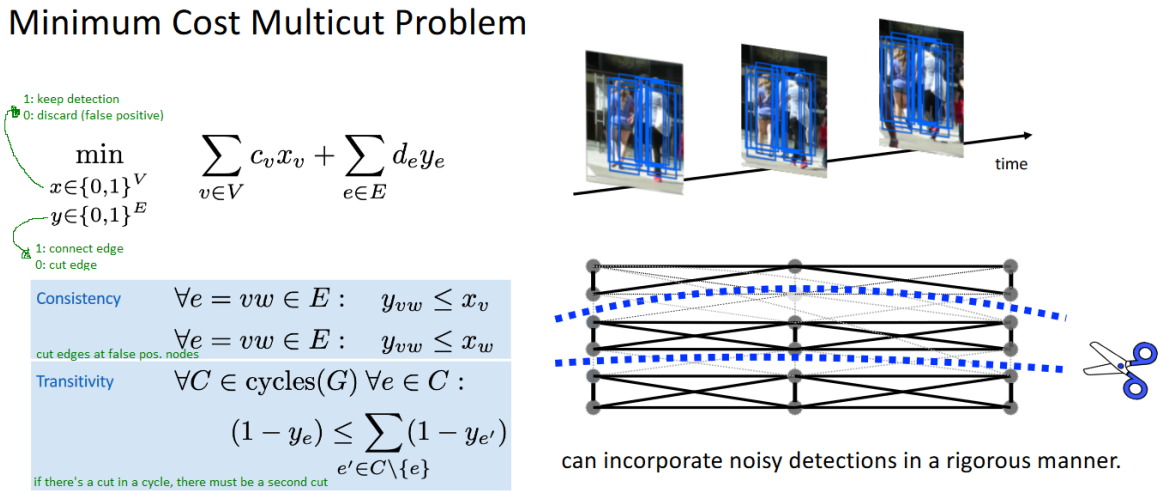
The human body is very complex and diverse. These problems are currently far from being solved.

Applications: human tracking, human pose estimation, face (or full body) replacement in movies, fit 3D body model in image

People recognition from visual input

Problem is cast as a discrete graph optimization problem (graph decomposition, e.g. by solving minimum cost multicut problem).

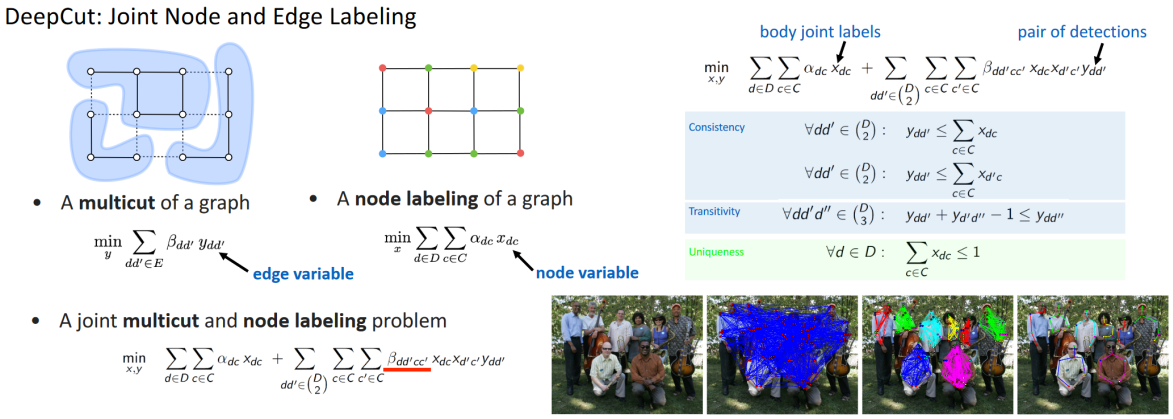
Human Tracking: Find bounding box around people and track each of them over the whole video. Detect all humans in the video (many bounding boxes, noisy, occlusions), then connect the correct boxes over multiple frames (graph decomposition: find the cuts in the graph of all possible connections to separate the trajectories of different people). Difficulty: re-identifying people with large temporal distance (approach: "lifted multicut" that incorporates connections over longer time with weights based on a "re-identify" neural net → correct local errors).



Human pose estimation:

Standard 2-stage approach: 1. Person detection, 2. Single person pose estimation.

DeepCut: jointly estimates number of people, their locations, their poses, partial occlusions and truncation. The problem is formulated as a graph cut problem (joint multicut and node labelling problem).



Generative human body models

View 3D human body models as a function of pose (Θ) and shape (β) $\rightarrow M(\Theta, \beta)$.

SMPL (skinned multi-person linear model): Template mesh $\rightarrow$ shape blend (modify person's body shape) $\rightarrow$ pose blend shapes (modify person's pose) $\rightarrow$ final mesh

11. Tracking

Follow the movements of something (point, region, template) or somebody (object with known identity)
Applications: Autonomous driving, safety monitoring, sports, AR/VR, image editing,...

Related exercise: Lab 9 - Condensation Tracker (not covered in this lecture though, see particle filters)

Tracking a point (simple approach based on intensity)

Find the displacement h of point x going from frame 0 to 1.
Energy function: $E(h) = [I_0(x+h) - I_1(x)]^2 \xrightarrow[\text{(small h)}]{\text{Taylor}} E(h) = [I_0(x) + hI_0'(x) - I_1(x)]^2$
Minimize energy ($\frac{\partial E}{\partial h} = 0$): $h = \frac{I_1(x) - I_0(x)}{I_0'(x)}$

Problems:

- Aperture problem: Motion in directions with zero gradient is unclear. To solve this we can assume a single motion for a region of pixels (neighbors move with pixel x).
  Find movement (u, v) of pixel region by solving (least squares):

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{54}$$

  $\rightarrow$ Good image features (with large structural eigenvalues, i.e. strong corners) are also good for tracking.
  $\leftrightarrow$ see optical flow for more details (same approach, there we look at the whole image instead of a small region)

- Local minima: In the linearization step we implicitly assume the solution to be the closest local minimum. If there are multiple local minima (or a big displacement h), this assumption doesn't hold and can lead to bad results. To avoid these problems, the frame rate should be faster than the motion of the tracked point.

Template Tracking (Lucas-Kanade tracker)

This algorithm allows for tracking a bigger image region based on a template image patch. The template can be transformed (translation, rotation, scaling, projective). This is captured by the motion model (or warp) $W(x, p)$. Like in the point tracking approach we define an energy E (summing over the image patch) and minimize it to figure out the motion parameters p.

$$E = \sum_x [I(W(x, p + \Delta p)) - T(x)]^2 \tag{55}$$

I: image (frame 0), T: target (frame 1)

We compute the Taylor expansion of E, and minimize the energy (derivation on slide 53) to get

$$\Delta p = H^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T [T(x) - I(W(x, p))]^2 \tag{56}$$

with

$$H = \sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T \left[\nabla I \frac{\partial W}{\partial p} \right]$$

The pseudocode and a diagram of the algorithm is shown in Figure 27.

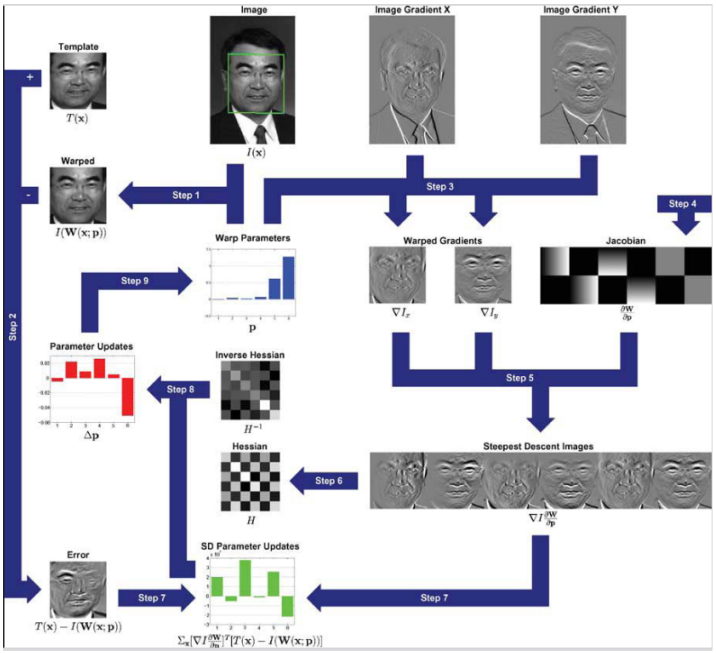


Figure 27: Lucas-Kanade template tracker.

Pros:

- beautiful framework
- It can handle different parameter spaces (various possible parametrizations for the motion model)
- Fast convergence in high frame-rate video (only few iterations needed per frame)

Cons:

- Not robust to image noise or large displacements (due to linearization, same local minima problem as in the point tracker)
- Some transformations are impossible to parametrize (e.g. 3D rotations where new image information would appear in reality)

Mean-Shift Tracking (non-parametrized template tracking)

Maximize a similarity function (that is e.g. based on a color histogram) between image and target using the mean-shift algorithm. A current mean (center) of pixels is iteratively updated by moving it to the centroid of pixels within a chosen radius (see Figure 28).

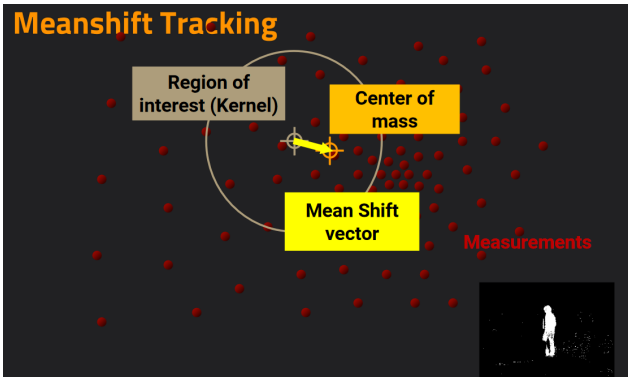


Figure 28: Mean-shift tracking.

Things to know:

- It works without an explicit motion model (aka non-parametrized)
- Efficient, can be used for real-time tracking applications (typically takes only few steps per frame)
- If the displacement is large, we can use a brute-force approach by sampling multiple starting positions and see where most of them converge
- There are different possible choices of feature space such as a color space, grayscale or gradients

Tracking by Detection (for more complex objects)

Most commonly used nowadays. Instead of iteratively finding the displacement/warp of the object, we first detect it and then connect it to the previous frames.

Tracking by features: Given an object template, for each frame we use this workflow:

- Detect key points (ideally invariant to scale, rotation or perspective change)
- Compute feature descriptors, e.g. HOG (rotation invariant), SIFT, SURF, FAST
- Match keypoint descriptors. This can be accelerated using a tree-structure (good for low-dimensional feature spaces), approximate nearest neighbors (for high dimensional feature spaces), hashing, parallel implementation.
- Eliminate outliers (e.g. using RANSAC)

This approach is more robust compared to the template tracking and is also efficient enough to run in real-time.

Tracking by Model

Detect occurrences of a model (e.g. pedestrian, car) in every frame and connect them based on their similarity.

- Detection: train an object detector using supervised learning (earlier approaches based on features, modern detectors use CNNs that are both accurate and fast).
- Space-time analysis: Connect the detections over multiple frames to estimate trajectories.

- Online Learning: Update the detection model in the loop based on the tracking results (to account for changes over time, e.g. illumination). Drift (detector loses track of the object because the model has changed too much) can be avoided by anchoring the model in the initial model (always make sure that the updated model is able to detect the initial model).
- For a more targeted search region (while still avoiding a purely local search) there are strategies like **Sliding Window** (image classification inside a window that slides over the frame to find the occurrence of our model) or **Divide and Conquer** (split the image in two halves, find the one which is more likely to contain our object, repeat process on that part of the split).
- Recent approaches use CNNs that combine detection and tracking.

Use location information (can be used to compensate detection errors)

We can assume a simple motion model (no motion, constant velocity, constant acceleration) to limit the search space for the object detection. More powerful motion modelling techniques such as Kalman filters can be used for more complicated motions.

Multiple Objects

- We can model the interaction between objects (e.g. social behaviour)
- Matching candidates from two frames can be formulated as a bipartite graph matching problem (can be solved by e.g. Hungarian algorithm). Similarity between candidates can be quantified in many ways (box overlap, feature distance, motion priors, social constraints,...)
- Similarity learning: learn how similar two candidates are. The loss can be binary (Do the candidates match? Yes/No) or triplet (more/less similar).
- Contrastive learning (expansion of similarity learning): Use many boxes around the ground truth to increase number of similar samples. This makes the model better at discriminating very similar candidates that should or should not be matched.

12. Image Segmentation

Task of partitioning an image into region that have some uniform property.

Relevant exercise: Lab 5 - Image Segmentation (mean-shift and EM)

Gestalt School (psychology): Grouping is key to visual perception. The whole is greater than the sum of its parts.
Grouping Factors: proximity, similarity, common fate, common region, parallelism, symmetry, continuity, closure. → These factors make intuitive sense but are very difficult to translate into algorithms

Superpixels: Group similar looking pixels into superpixels to speed up further processing.

Segmentation as clustering: Image segmentation can be viewed as a pixel clustering problem (k-means, EM, mean-shift).

k-Means clustering (quick and easy)

Randomly initialize k cluster centers and iterate between the following two steps (until the centers don't change anymore):

- Given the cluster centers, determine the points in each cluster. (find the closest center for each point)
- Given the points in each cluster, find the cluster center. (set center to the mean of points in the cluster)

Different feature spaces can be used for this algorithm (grayscale, RGB, filter bank responses,...), as long as we can define a `mean` and a `distance` (SSD) operator that stick to the same concept (e.g. euclidean space).

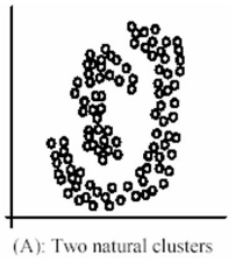
Spatial coherence: We can group pixels based on similarity and position to get spatially smooth results (imagine a feature space like R, G, B, X, Y). Other approaches are better at this (see random fields).

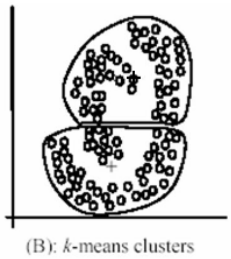
Pros:

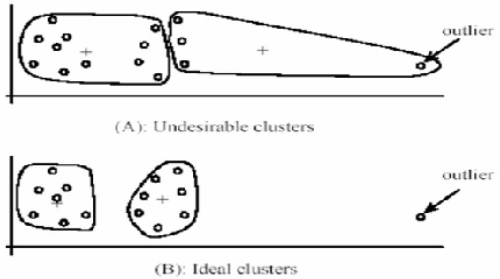
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons:

- Setting k (you need to know how many clusters there are)
- Sensitive to initial cluster centers
- Sensitive to outliers (every point ends up in a cluster)
- Detects spherical clusters only (implicitly assumes spherical clusters)
- Assuming means can be computed


(A): Two natural clusters


(B): k-means clusters



Mixture of Gaussians / Expectation Maximization (probabilistic clustering) "k-means on steroids"

Assume that the observed data is generated by sampling a continuous function ($\rightarrow$ generative model, described by a mixture of Gaussians). The Expectation Maximization algorithm iteratively finds the maximum likelihood estimates for parameters Θ over all the data points.

Applications: any clustering problem, many model estimation problems, missing data problems, finding outliers, segmentation problems, ...

Mixture of Gaussians (MoG):

The generative model is a linear combination of k Gaussian blobs with means μ_b, covariance matrices V_b and dimension d. Θ summarizes μ and V of all the k Gaussians.

$$P(x|\Theta) = \sum_{b=1}^{k} \alpha_b P(x|\Theta_b), \tag{57}$$

where

$$P(x|\Theta_b) = \frac{1}{\sqrt{(2\pi)^d |V_b|}} \exp\left[-\frac{1}{2}(x-\mu_b)^T V_b^{-1}(x-\mu_b)\right]$$

Expectation Maximization (EM) (algorithm to train the MoG model):

Find parameters Θ that maximize the likelihood function $P(X) = \prod_{x_i \in X} P(x_i|\Theta)$.

1. E-step: given the current guess of blobs (Θ), compute the probabilistic ownership of each point. (All the points always belong to all the clusters. Typically, a point has a high probability only for one cluster after some iterations)
2. M-step: given the ownership probabilities, update blobs (Θ) to maximize the likelihood function.
3. Repeat until convergence.

EM Details

- **E-step**
  - **Compute probability that point x is in blob b, given current guess of θ**
$$P(b|x, \mu_b, V_b) = \frac{\alpha_b P(x|\mu_b, V_b)}{\sum_{i=1}^{K} \alpha_i P(x|\mu_i, V_i)}$$
- **M-step**
  - **Compute overall probability that blob b is selected**
$$\alpha_b^{new} = \frac{1}{N} \sum_{i=1}^{N} P(b|x_i, \mu_b, V_b) \qquad (N \text{ data points})$$
  - **Mean of blob b**
$$\mu_b^{new} = \frac{\sum_{i=1}^{N} x_i P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$
  - **Covariance of blob b**
$$V_b^{new} = \frac{\sum_{i=1}^{N} (x_i - \mu_b^{new})(x_i - \mu_b^{new})^T P(b|x_i, \mu_b, V_b)}{\sum_{i=1}^{N} P(b|x_i, \mu_b, V_b)}$$

Pros:

- Probabilistic interpretation
- Soft assignments between points and clusters (makes training more stable)
- Generative model, can predict novel datapoints
- Relatively compact storage ($\mathcal{O}(kd^2)$)

Cons:

- Requires initialization (typically using the output of k-means)
- Converges to local minimum (no global guarantees, dependence on initialization)
- Need to specify k
  (possible solution: model selection based on AIC/BIC (information criterion) $\rightarrow$ trade-off between energy (fit quality) and simplicity (size of k) of the model)
- Need to choose the mathematical form of the generative model (here: Gaussian, we could also use other probabilistic models)
  $\rightarrow$ requires some knowledge about the data (not necessarily a disadvantage)
- Numerical problems are often a nuisance

Mean-Shift Segmentation (model-free clustering)

This algorithm iteratively finds modes (local maxima) in a given distribution $H(x)$. A cluster contains all the points that are in the same attraction basin (the region for which all trajectories lead to the same mode). The resolution of the modes is determined by setting the window size h.

Iterative Mode Search:

1. Initialize seed center and window W.
2. Calculate the center of gravity (mean) of the points in W: $\sum_{x \in W} xH(x)$
3. Shift the search window's center to the mean.
4. Repeat steps 2 and 3 until convergence.

Overall Algorithm:

1. Choose features (color, gradients, texture,...)
2. Initialize windows at individual pixel locations (e.g. at all the pixels, at randomly sampled pixels,...)
3. Start mean-shift (iterative mode search) from each window until convergence
4. Merge windows that end up near the same "peak" (mode)

Pros:

- General, application independent tool (only requires a way to find a "mean")
- Model-free (doesn't assume any prior shape of the clusters)

- Just a single parameter (window size h) $\rightarrow$ h has physical meaning (scale of clustering), better than choosing the number of clusters k
- Finds variable numbers of modes (k) given the same h
- Robust to outliers (they end up in their own cluster)

Cons:

- Output depends on window size h, window size selection is not trivial
- Computationally expensive
- Does not scale well with increasing dimension of feature space (hard to find a meaningful notion of distance)

Hough Transforms (edge-based) (see 6. Model Fitting for details)

Uses the structure of shapes to extract them in a parameter space of limited dimension. The hough transform can detect simple shapes in edge images (lines, circles, ...) that separate image regions that you wish to segment.

Line detection example:

- Detect edges (e.g. using canny edge detector)
- Apply Hough transform on edge image
- Find peaks in hough image (e.g. using mean-shift)
  $\rightarrow$ each peak corresponds to a line, stronger peak means longer line

Remarks:

- Time consuming computation
- Robust to outliers and lots of points that are not on a line
- Peak detection gets difficult with noise in edge coordinates (peaks become smeared)
- Works also for circles and even higher order parametric shapes, but it gets harder as the number of shape parameters increases (curves in parameter space become less likely to intersect which makes it harder to find peaks; effect of noise increases)

Graph Cuts (interactive segmentation)

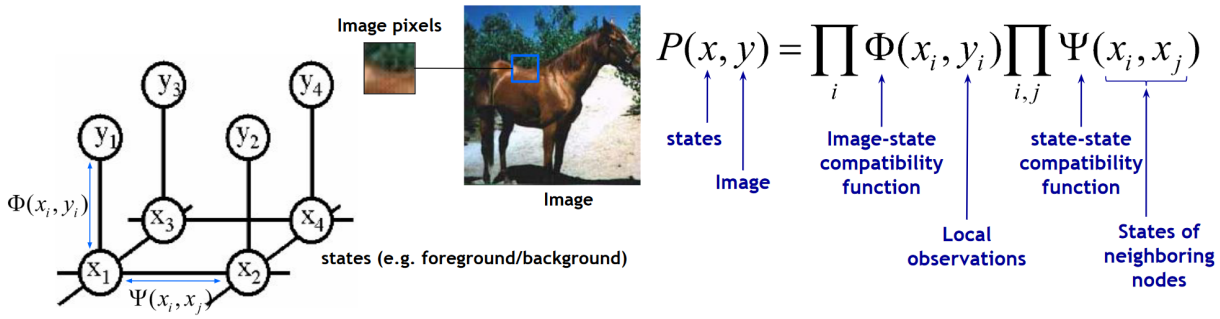
Graph cut optimization is an approach to solve energy minimization problems (that could e.g. be modelled using MRF). For certain classes of energy functions it is fast enough to solve dense problems where every pixel is a node.

Applications:

GrabCut, an interactive image segmentation tool. The user marks some rough foreground and background regions with a brush (to get an initial segmentation) and corrects the segmentation with additional brush strokes.

Markov Random Fields (MRF): MRF is an approach to create rich probabilistic models in a local, modular way. The goal is to find the optimal labelling (find the hidden states x) of the MRF. Observed

evidence y (e.g. pixel values), hidden states x (e.g. cluster membership), neighborhood relations Ψ (modelling assumptions, e.g. neighbors are more likely to have the same hidden state).



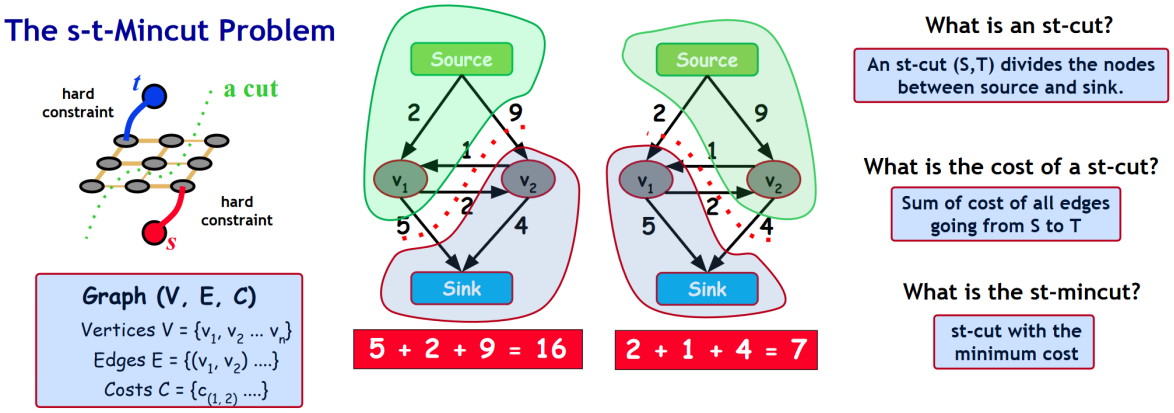
Maximizing the joint probability P (global field probability) is equivalent to minimizing an energy function E (sums are easier to handle numerically than products):

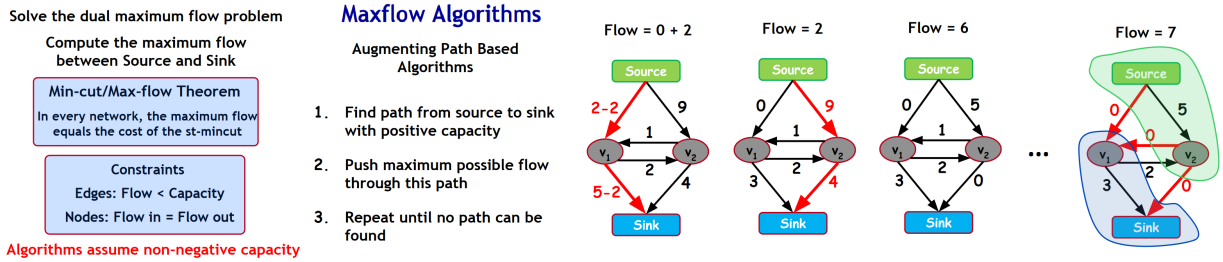
$$E(x,y) = \sum_i \phi(x_i, y_i) + \sum_{i,j} \psi(x_i, x_j), \tag{58}$$

where the potentials (ϕ, ψ) could e.g. be $\phi = -\log \Phi$ (unary potentials) and $\psi = -\log \Psi$ (pairwise potentials, used to achieve spatial smoothness). The potentials are not necessarily tied to the probabilistic interpretation, they are defined according to the problem at hand (defined such that lower is better).

Graph Cuts for Optimal Boundary Detection:

To solve a binary assignment problem (e.g. foreground/background segmentation), we can convert the MRF into a source-sink graph and find the minimum cost cut (solving the Min-Cut problem) using a Max-Flow algorithm (there's a lot of them). S-t graph cuts can only globally minimize *binary energies* (two possible node states) that are *submodular* (some mathematical statement that ensures that the graph can be modified to have only positive edge capacities/costs). Non-submodular (and even multi-label) problems can still be addressed approximately (e.g. using alpha-expansion algorithm).





Pros:

- Powerful technique, based on probabilistic model (MRF)
- Applicable to a wide range of problems
- Very efficient algorithms available for vision problems (grid graphs, low number of states, specific pairwise potentials)

Cons:

- Graph cuts only solve a limited (but useful) class of models
  ($\rightarrow$ requires submodular energy functions, can only capture part of the expressiveness of MRFs, can only approximate multi-label cases (with a few specific exceptions))

Learning-Based Approaches

More based on data and less reliant on human knowledge. General scheme (pre-deep-learning):

$$\text{Object} \xrightarrow{Sensors} \text{Measurements} \xrightarrow[Extraction]{Feature} \text{Features } \{v_j\}_{j=1}^{M} \xrightarrow{Classification} \text{Object Class } \{c_i\}_{i=1}^{K}$$

Discriminative learning means finding a mapping $f(v_j) = c_j$ based on M examples of features and their corresponding class (e.g. segmentation) out of a vocabulary of K classes. Often the mapping can be interpreted probabilistically: $f(v_j) \sim p(c_j|v_j)$ or $f(v_j) \sim \arg_{c_j} \max p(c_j|v_j)$.

K-nearest neighbors (KNN):

Find the k nearest neighbors (in feature space) within the training dataset for a sample. The mapping $f(v) = c$ is defined through the labels of these neighbors, i.e. a function $f(v) = f(c_1, ..., c_k)$ (e.g. majority voting).

Pros:

- Simple to implement & understand
- Efficient implementations possible for approx. NNs
- Distance definition is flexible

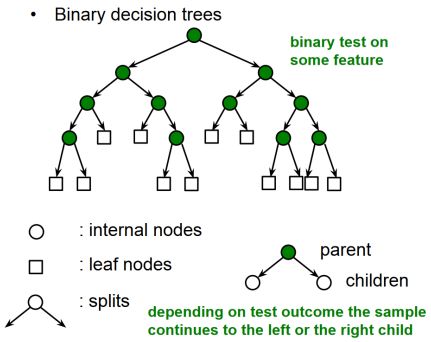
Cons:

- Highly dependent on distance definition and k
- Need to keep training data in memory for distance computations (high memory usage)

- High-dimensional problems might need many training samples for accuracy (labelling training data is expensive)
- Other methods have a better generalization ability (better accuracy on new test sets)

Random Forests:

A random forest is a collection of (e.g. binary) decision trees that are trained in a manner that incorporates randomness. Random forests are slow to train but very fast to apply during testing, which makes them popular in industry.



Training: Setting up all the trees in the forest (deciding on the node tests, deciding when a leaf is reached). Deciding on the node tests can be achieved by randomly generating tests (e.g. thresholds on single features) and finding the "best" test that minimizes entropy (class distribution is more peaked / less uniform after the test). This splitting continues until a node is declared a leaf node based on some stopping criterion (e.g. all the samples in it belong to the same class).

Testing/Inference: Applying the trees to incoming samples. A sample (feature vector) is dropped in at the top node. At each internal node there is a binary question (in the case of binary trees). The sample will end up in one of the node leafs; each leaf corresponds to a prediction of the class/label for samples getting there. Usually this is repeated on multiple trees (trained with the same approach but different due to randomization) and the results are combined (e.g. voting, averaging) to increase robustness.

Pros:

- Easy to implement
- Very efficient during testing/inference
- Can easily use diverse features
- Can handle high dimensional spaces

Cons:

- Lots of parametric choices (what are the tests, how do you randomize, stopping criterion, ...)
- Needs a lot of training data
- Training can take time

13. Object Class Recognition

As opposed to specific object recognition, object class recognition generalizes the problem to recognize any instance of a certain object class (e.g. recognize *any* car). Focus of this lecture: classification (is there a car in this image?) and detection (where is the car?).

Challenges (Robustness): illumination, object pose, clutter, occlusions, intra-class variation, viewpoint

Bag of Words (BoW):

Summarize an image based on its distribution (histogram) of visual word occurrences. This histogram is a vector of fixed dimensionality (vocabulary size) that can be used with various machine learning algorithms. Works quite well for whole-image classification (Google image search worked with BoW until a few years ago).

Visual words: map high-dimensional descriptors to words by quantizing (clustering, cluster centers are the prototype "words") the feature space. The *visual vocabulary* is generated by sampling (extracting features, e.g. dense sampling) and clustering features of the training set.

Spatial information: A BoW is an orderless representation. This is a pro (generalizability) and a con (discriminative power) at the same time. The are some approaches to incorporate spatial information in BoWs, such as visual phrases (frequently co-occurring words), semi-local features (describe neighborhood), adding position tho each descriptor, counting BoWs in sub-grids of the image only.

BoW for image classification: Given BoW descriptions of images from different classes, we can learn a model for distinguishing them by using a cl assifier (e.g. nearest neighbor, SVM, Boosting, Bayesian classification, CNN...).

Pros:

- Flexible to geometry / deformations / viewpoint
- Compact summary for image content
- Provides vector representation for sets (bags) across images
- Empirically good recognition results in practice

Cons:

- Basic model ignores geometry
- Background and foreground mixed when bag covers whole image (could e.g. lead to cheetahs being classified due to background features)
- Interest points or sampling: no guarantee to capture object-level parts (most common strategy: dense sampling)
- Optimal vocabulary formation remains unclear

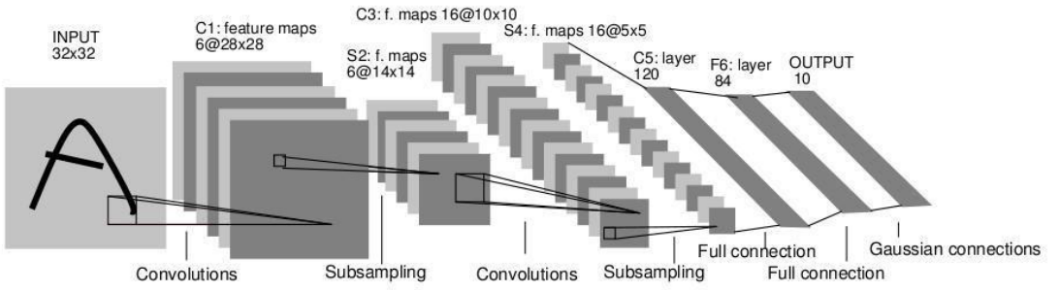
Support Vector Machine (SVM):

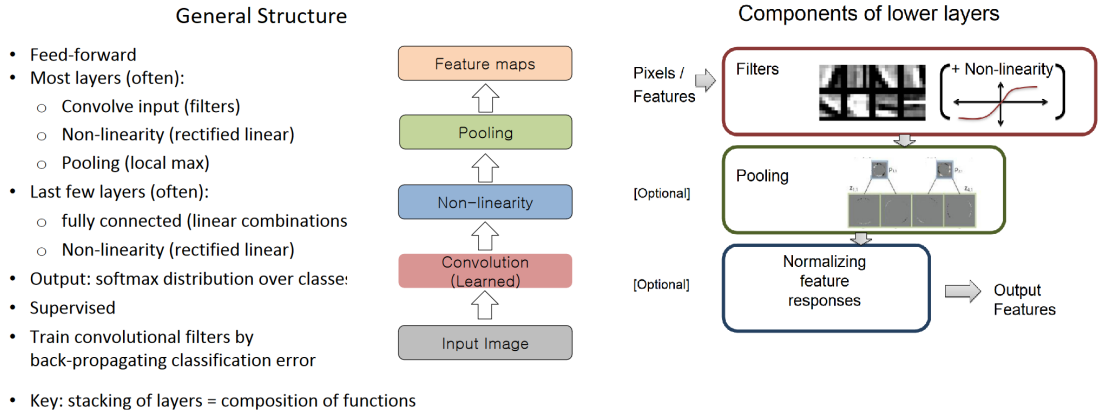
Discriminative classifier based on an optimal separating hyperplane (decision boundary in feature space). It maximizes the margin between positive and negative training examples.

Convolutional Neural Network (CNN)

CNNs not only learns the classifier but also the features (main distinguishing element compared to classical approaches). Much of the performance improvement comes from that (learning features for a specific task).

In the early layers of the network (convolutional layers → "learning features"), the input is convolved with multiple independent filters. The thickness of a layer is its number of channels (a.k.a. banks, kernels), it defines how many feature maps are considered by the next layer. The last few layers are typically fully connected and "take care" of the classification. In reality, there's no sharp distinction between layers responsible for features or classification (although it's easier to think about it like that), it's more of a continuum.





Non-linearity: Essential to model complex input-output-mappings. Today, *Rectified linear (ReLU)* is more popular (thanks to faster training and simpler backprop) than *tanh* or *sigmoid*.

Pooling & Striding: Summarize the statistics of neighboring pixels (using *max* or *sum*) instead of preserving one value for each pixel. This leads to more tolerance of small shifts. Typically, *pooling* is combined with *striding* (skipping some pixels in the filtering step). This combination reduces the image resolution in each NN layer and allows to aggregate information over larger and larger image regions.

Normalization: Locally normalize (0 mean, 1 stddev) to equalize the feature maps. This reduces the variance in magnitude across feature maps

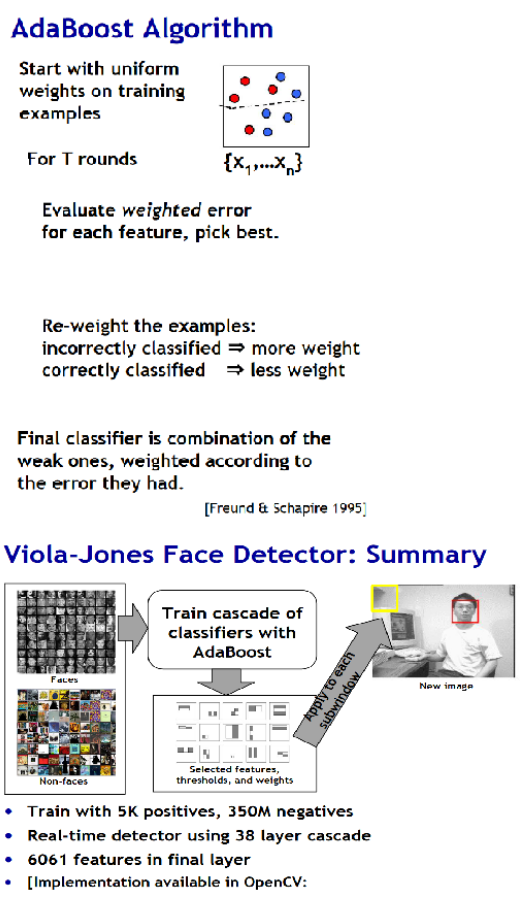
End-to-end training: In the training phase, all the parameters of the fully connected layers, as well as the parameters of all the filters are optimized to achieve the best performance on the training dataset.

Boosting

Build a strong classifier by combining many "weak" classifiers which only need to be better than chance. It's a sequential process (iteratively add weak classifiers) that is flexible regarding the choice of classifiers.

AdaBoost (simple boosting algorithm): Add a classifier, re-weight the datapoints (give more weight to wrongly classified points), add a classifier that improves performance given the new weights, repeat.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$



Sliding-Windows (Detection via Classification)

Using sliding windows we can do detection via classification. Slide a window over the image (possibly also on multiple scales) and classify the content of that window (using any classifier). This is a brute-force approach relying on many local decisions.

- **Pros**
  - Simple detection protocol to implement
  - Good feature choices critical
  - Past successes for certain classes
  - Good detectors available (Viola&Jones, HOG, etc.)

- **Cons/Limitations**
  - High computational complexity
    - For example: 250,000 locations x 30 orientations x 4 scales = 30,000,000 evaluations!
    - This puts tight constraints on the classifiers we can use.
    - If training binary detectors independently, this means cost increases linearly with number of classes.
  - With so many windows, false positive rate better be low
  - Typically need fully supervised training data (= bounding-boxes)
  - Some object do not fit a box well (diagonal bottle)
  - Sensitive to partial occlusion (unless in training data)

Proposal-based Detection

Sliding window approaches require to search a 4-dimensional space over each image (window position, height, width), which leads to a lot of detector (classifier) evaluations. A proposal-based approach reduces this number of required evaluations (from $\sim$ 1M to $\sim$ 1k) by relying on low-level *perceptual organization* cues, which allows the detector to use more powerful features and classifiers ($\rightarrow$ enabled the use of CNNs). The proposal mechanism can be trained and it's possible to make it category-independant.

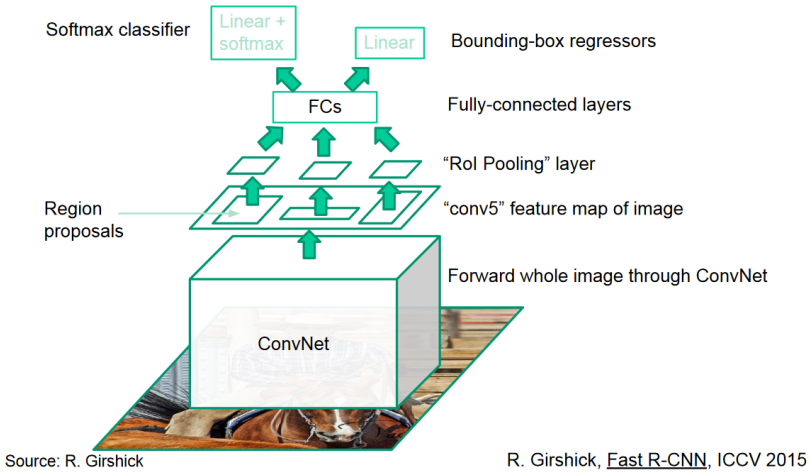
Objectness: general properties of objects.

An object has a closed boundary in space, usually a different appearance than its surroundings and might be unique (salient) in the image.
$\rightarrow$ Create a function that describes the probability that a window w covers an object. (many approaches exist to do this, active research)

R-CNN: Approach combining object proposals with CNN features.

Run object proposal on the input image, warp the image regions to a square shape (required input shape), run each object proposal through a CNN, classify the resulting features using SVM.

Fast R-CNN: Improved version of the (rushed) R-CNN paper (basic idea: switch order of object proposal and CNN) that is a lot faster to train and test.



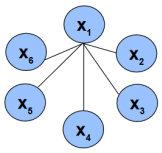
Implicit Shape Models

This approach is similar to Bag of Words but it additionally encodes spatial structure. Assuming a 'star' model, each object contains visual words that are in a consistent configuration (but independent from each other apart from their relation to the object 'center').

Approach:

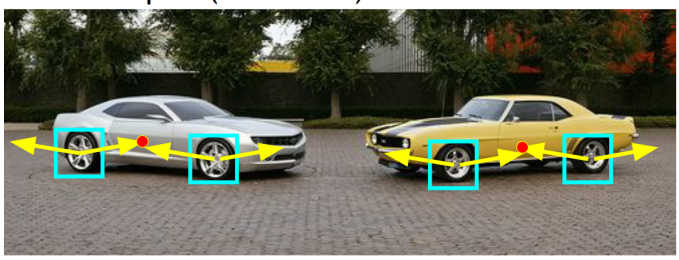
- Training:
  - Learn appearance codebook (extract local features, cluster to get codebook)
  - Learn spatial distributions (match codebook to train images, record matching positions on object)
- Testing/Inference:
  - Match codebook to test images
  - Find object centers using probabilistic voting
  - For segmentation: back projection of hypotheses, refinement, segmentation

'Star' model



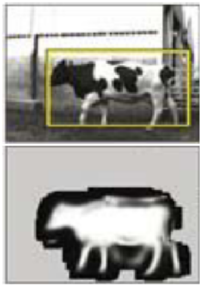
- e.g. Implicit Shape Model (ISM)
- Parts mutually independent
- Recognition complexity: O(NP)
- Method: Gen. Hough Transform

- Objects are detected as consistent configurations of the observed parts (visual words).

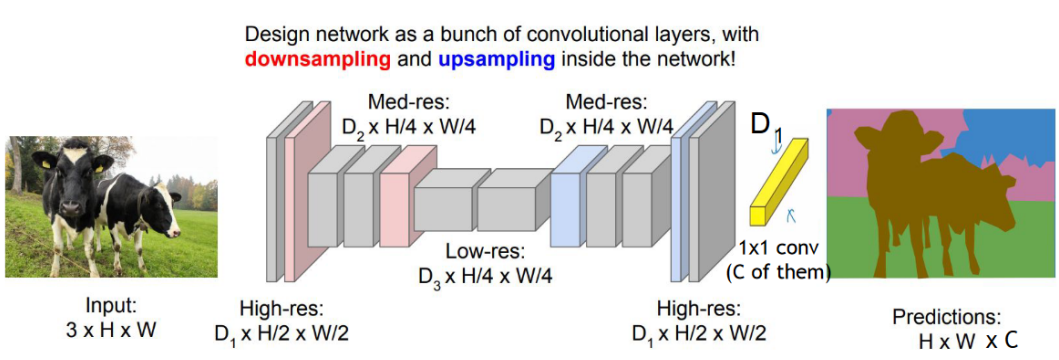


Test image

- **Pros:**
  - Works well for many different object categories
    - Both rigid and articulated objects
  - Flexible geometric model
    - Can recombine parts seen on different training examples
  - Learning from relatively few (50-100) training examples
  - Optimized for detection, good localization properties
- **Cons:**
  - Needs supervised training data
    - Object bounding boxes for detection
    - Segmentations for top-down segmentation
  - Only weak geometric constraints
    - Result segmentations may contain superfluous body parts. Model cannot enforce counts.
  - Purely representative model
    - No discriminative learning



Segmentation with Neural Networks (semantic segmentation)



- append 1x1 convolutions to go from feature channels to number of classes C
- predictions at the same resolution as input image H x W
- no fully connected layers anywhere
  (different from AlexNet for whole-image classification we saw before)