EE 589/689 Foundations of computer vision: Lecture notes

Fall quarter 2006, OGI/OHSU

Miguel Á. Carreira-Perpiñán

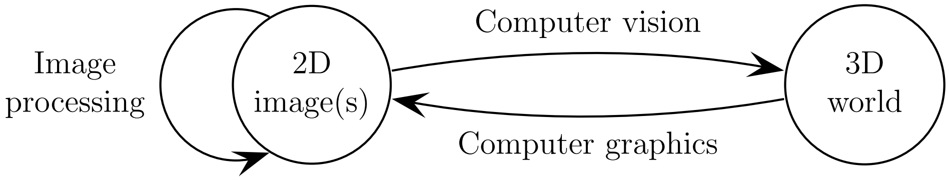
Based mainly on:
David Forsyth and Jean Ponce: *Computer Vision. A Modern Approach.* Prentice-Hall, 2003.

Introduction to computer vision

Computer vision has been around since the 1960s. Recent developments:

- Increasing availability of cheap, powerful cameras (e.g. digital cameras, webcams) and other sensors.
- Increasing availability of massive amounts of image and multimedia content on the web (e.g. face databases, streaming video or image-based communication).
- Increasing availability of cheap, powerful computers (processor speed and memory capacity).
- Introduction of techniques from machine learning and statistics (complex, data-driven models and algorithms).

Three related areas:



- Computer graphics: representation of a 3D scene in 2D image(s).
- Computer vision: recovery of information about the 3D world from 2D image(s); the inverse problem of computer graphics.
- Image processing: operate one one image to produce another image (e.g. denoising, deblurring, enhancement, deconvolution—in particular in medical imaging).

Some problems of computer vision:

- Structure-from-motion (3D reconstruction from multiple views, stereo reconstruction)
- Shape-from-X (single image):
  - shape-from-texture
  - shape-from-shading
  - shape-from-focus
- Segmentation
- Tracking
- Object recognition

A few applications of computer vision:

- Structure-from-motion:
  - Throw away motion, keep structure: image-based rendering (e.g. 3D models of buildings, etc. for architecture or entertainment industry)
  - Throw away structure, keep motion: mobile robot control (we know the structure but not the robot location)
- Image collections:
  - Image retrieval: find me pictures containing cars and trees
  - Image annotation: textual description of objects in image
- Finding faces in a group picture, crowd, etc.
- Recovering articulated pose of a person from a video
- Medical applications:
  - Image enhancement
  - Segmentation of brain
  - Image registration or alignment: compare brains of different people, or brains before/after lesion
  - Blood vessels: track cells
  - Unobstrusive patient monitoring
- HCI: track eye motion; recognize physical gestures (e.g. sign language)

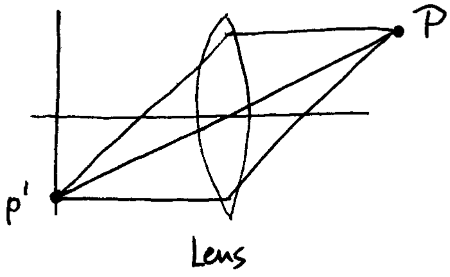
CH. 1: CAMERAS

We don't have direct access to the physical world, but indirect access through the camera(s). How does the camera map the world to the image?

1.1. PINHOLE CAMERA:

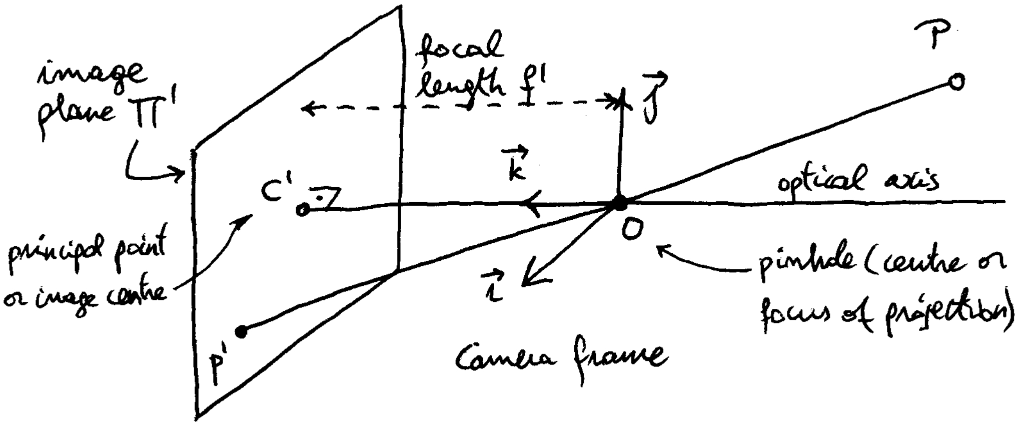
consider camera-centred coordinate system, assume pinhole = point (ignores blurring due to finite size, diffraction due to small size).

FIG. 1.2, 1.3



Lenses are used to gather more light (in pinhole cameras, each image point p' receives a single ray so quite dark) and focus it (= force all rays coming from world point P to converge on the same image point).
Lenses also introduce aberrations (spherical, chromatic...)

Projection = mapping from scene point to image point:



Scene point $P(x, y, z)$, $z < 0$
Its image $p'(x', y')$

* Perspective projection: P, O, p' collinear $\Rightarrow \vec{Op'} = \lambda . \vec{OP} \Rightarrow \begin{cases} x' = \lambda x \\ y' = \lambda y \\ z' = \lambda z \end{cases} \Rightarrow \boxed{\begin{matrix} x' = f' \frac{x}{z} \\ y' = f' \frac{y}{z} \end{matrix}}$

$\underset{f' \text{ constant}}{z'}$ "

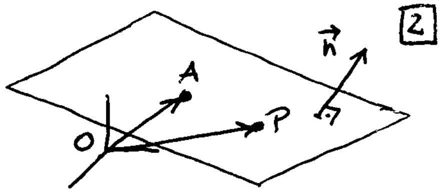
nonlinear; does not preserve distances or angles, but does map lines into lines.

* Affine projection: useful approximation to perspective proj.

- Weak perspective: scene depth << distance to camera $\Rightarrow z \simeq z_r$ constant $\Rightarrow \boxed{\begin{matrix} x' = -m x \\ y' = -m y \end{matrix}}$ where $m = -\frac{f'}{z_r} > 0$ is the magnification; linear.
- Orthographic projection: camera always at a constant distance from scene; normalise $m = -1$ for simplicity $\Rightarrow \boxed{\begin{matrix} x' = x \\ y' = y \end{matrix}}$ (usually unrealistic).

See fig. in next page

CH. 2: GEOMETRIC CAMERA MODELS



2.1. ELEMENTS OF ANALYTICAL EUCLIDEAN GEOMETRY

- Eq. of a plane: $(\vec{OP}-\vec{OA})\cdot\vec{n}=0$ given that $A\in$ plane, normal $\vec{n}$.

  $\Leftrightarrow ax+by+cz-d=0$ where $P=\begin{pmatrix}x\\y\\z\end{pmatrix}$, $\vec{n}=\begin{pmatrix}a\\b\\c\end{pmatrix}$ and $d=\vec{OA}\cdot\vec{n}$ = constant = signed distance from O to plane.

- Homogeneous coordinates: $\begin{pmatrix}x\\y\\z\\1\end{pmatrix}$ so we can write $\Pi\cdot P=0$ with $\Pi=\begin{pmatrix}a\\b\\c\\-d\end{pmatrix}$, $P=\begin{pmatrix}x\\y\\z\\1\end{pmatrix}$.

  Also applicable to 2D: eq. of a line $V\cdot P=0$ with $V=\begin{pmatrix}a\\b\\-c\end{pmatrix}$, $P=\begin{pmatrix}x\\y\\1\end{pmatrix}$.

- Quadric surface (sphere, ellipsoid...): P^TQP with $P_{4\times1}$ in homog. coord. and $Q_{4\times4}$.

- Change of coordinate system by rigid transformation: ${}^BP={}^B_AR\cdot{}^AP+{}^BO_A$ where:
  - ${}^AP,{}^BP$: point in coordinates in systems A, B
  - ${}^BO_A$: translation of the origin of A to the origin of B
  - ${}^B_AR$: rotation matrix $=\begin{pmatrix}\vec{i}_A\cdot\vec{i}_B & \vec{j}_A\cdot\vec{i}_B & \vec{k}_A\cdot\vec{i}_B\\ \vec{i}_A\cdot\vec{j}_B & \vec{j}_A\cdot\vec{j}_B & \vec{k}_A\cdot\vec{j}_B\\ \vec{i}_A\cdot\vec{k}_B & \vec{j}_A\cdot\vec{k}_B & \vec{k}_A\cdot\vec{k}_B\end{pmatrix}=\begin{pmatrix}{}^B\vec{i}_A & {}^B\vec{j}_A & {}^B\vec{k}_A\end{pmatrix}=\begin{pmatrix}{}^A\vec{i}_B^T\\ {}^A\vec{j}_B^T\\ {}^A\vec{k}_B^T\end{pmatrix}$ $\leftarrow$ not homogeneous $\rightarrow$

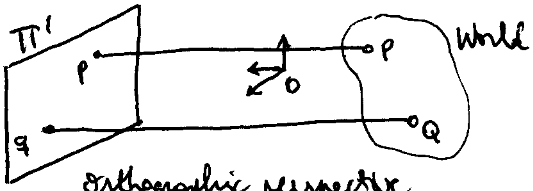
  Properties: preserve distances and angles; ${}^A_BR={}^B_AR^T={}^B_AR^{-1}$; $|{}^B_AR|=1$; the set of rotation matrices forms a noncommutative group wrt matrix product, SO(3).

  In homogeneous coord. the rigid transformation is $\begin{pmatrix}{}^BP\\1\end{pmatrix}={}^B_AT\begin{pmatrix}{}^AP\\1\end{pmatrix}$ (matrix-vector product in $\mathbb{R}^4$) where, writing matrices in blocks with $\vec{0}=(0\ 0\ 0)^T$:

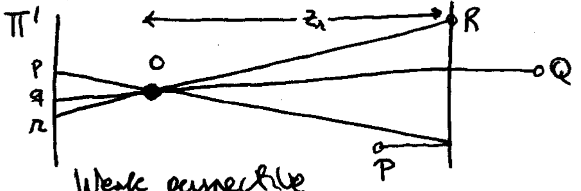
  ${}^B_AT=\begin{pmatrix}{}^B_AR & {}^BO_A\\ \vec{0}^T & 1\end{pmatrix}$. The set of ${}^B_AT$ also forms a noncommutative group wrt matrix product.

- Affine transformation: $T=\begin{pmatrix}A & t\\ 0^T & 1\end{pmatrix}$ where $A_{3\times3}$ is nonsingular but not necessarily a rotation; do not preserve distances or angles; form a group. $T^{-1}=\begin{pmatrix}A^{-1} & -A^{-1}t\\ 0^T & 1\end{pmatrix}$.

- Projective transformation: $T_{4\times4}$ is nonsingular but arbitrary.



orthographic perspective



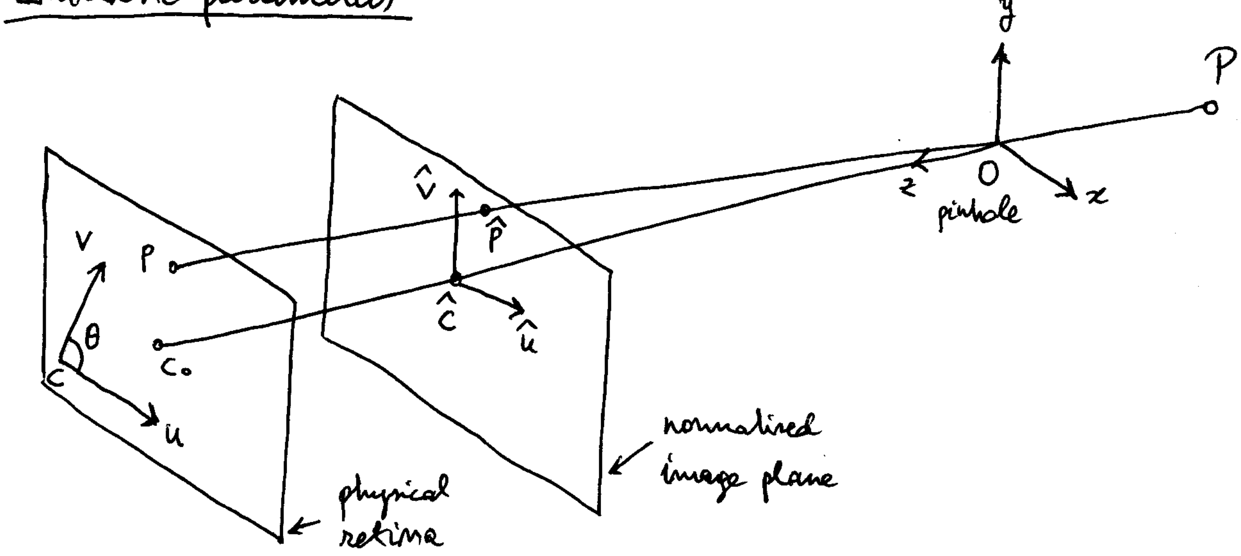
Weak perspective

2.2. Camera Parameters & the Perspective Projection

Consider an external, arbitrary world coord. sys. different from the camera coord. sys. The world and camera coord. sys. are related by a set of physical parameters:

- Intrinsic: relate the camera's physical coordinate system to the idealised coord. sys (focal length of lens, size of pixels, position of principal point, skew).
- Extrinsic: relate the camera's coord sys. to a fixed world coord. syst. and specify its position and orientation in space.

* Intrinsic parameters



- Normalised image plane: parallel to image plane at distance 1 from pinhole.

  Perspective projection: $\begin{cases} \hat{u} = x/z \\ \hat{v} = y/z \end{cases}$

- Physical retina (pixel coord.) at distance $f \neq 1$ from pinhole with image coord. expressed as rectangular pixel units $^1/_l \underset{1/k}{\boxed{\begin{matrix}\text{rect.}\\ \text{pixel}\end{matrix}}}$, magnification $\alpha = kf$, $\beta = lf$ and displacement $\binom{u_0}{v_0}$ of the retina's origin: $\begin{cases} u = \alpha\, x/z + u_0 \\ v = \beta\, y/z + v_0 \end{cases}$

  The retina coordinate system may be slightly skewed by an angle θ: $\begin{cases} u = \alpha \frac{x}{z} - \alpha \cot\theta \frac{y}{z} + u_0 \\ v = \frac{\beta}{\sin\theta} \cdot \frac{y}{z} + v_0 \end{cases}$

- Change of coordinates (in homogeneous coord.): $p = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$, $\hat{p} = \begin{pmatrix} \hat{u} \\ \hat{v} \\ 1 \end{pmatrix}$, $P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$):

  - Normalised → retina: $p = K\hat{p}$ with matrix of intrinsic parameters $\boxed{K = \begin{pmatrix} \alpha & -\alpha\cot\theta & u_0 \\ 0 & \frac{\beta}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{pmatrix}}$
  - Camera → retina: $p = \frac{1}{z} MP$ where $\underset{3\times4}{M} = \left(\underset{3\times3}{K}\ \underset{3\times1}{0}\right)$
  - Camera → normalised: $\hat{p} = \frac{1}{z}\left(\underset{3\times3}{I}\ \underset{3\times1}{0}\right)P$

The intrinsic parameters are generally known from the manufacturer (pixel size, skew) but not always (eg. stock film footage, zoom). Particular cases: zero-skew $(\theta = \frac{\pi}{2})$, unit aspect ratio $(\alpha = \beta)$.

* Extrinsic parameters: now camera frame ≠ world frame, related by rigid transformation: perspective projection eq. world → retina $\boxed{p = \frac{1}{z} M P}$ with perspective projection matrix

$$M = K \cdot (\underset{\text{rotation}}{R} \quad \underset{\text{translation}}{t}) = \begin{pmatrix} \alpha r_1^T - \alpha \cot\theta \, r_2^T + u_0 r_3^T & \alpha t_x - \alpha \cot\theta \, t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} r_2^T + v_0 r_3^T & \frac{\beta}{\sin\theta} \cdot t_y + v_0 t_z \\ r_3^T & t_z \end{pmatrix},$$ explicitly as a function

of 11 parameters $\begin{cases} \text{5 intrinsic: } \alpha, \beta, u_0, v_0, \theta \\ \text{6 extrinsic: 3 rotation angles, 3 translation coord.} \end{cases}$

2.3. Affine Cameras and Affine Projection Equations

Weak perspective (scene's relief ≪ distance to camera): $\begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \begin{pmatrix} x \\ y \\ z_r \end{pmatrix}$

- Camera → normalized: $\begin{pmatrix} \hat{u} \\ \hat{v} \\ 1 \end{pmatrix} = \begin{pmatrix} x/z_r \\ y/z_r \\ 1 \end{pmatrix} = \frac{1}{z_r} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_r \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ in matrix notation.

- World → retina: $p = MP$ with $M = \frac{1}{z_r} K \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_r \end{pmatrix} \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix}$ with $\begin{cases} \text{calibration matrix } K \\ \text{extrinsic par. } R, t \end{cases}$

Since some of the parameters are coupled, we can write $\begin{pmatrix} u \\ v \end{pmatrix} = MP$ with $\underset{2\times4}{M} = \frac{1}{z_r} \begin{pmatrix} k & s \\ 0 & 1 \end{pmatrix} (R_2 \; t_2)$ where $\underset{2\times3}{R_2}$, $\underset{2\times1}{t_2}$ = first two rows of R, t, and k and s are the aspect ratio and skew of the camera. Thus the weak-perspective projection matrix is defined by 2 intrinsic parameters (k, s), 5 extrinsic parameters (3 angles in R_2, 2 coord. in t_2) and 1 scene-dependent parameter z_r.

Orthographic projection: fix z_r = constant (eg. $z_r = 1$). For one camera only.

Ch. 3: Geometric Camera Calibration

- The problem of estimating the intrinsic/extrinsic parameters of a camera.
- Set up a calibration rig (pattern): set of points or lines with known positions wrt world frame $\boxed{\text{FIG. 3.1}}$
- Calibration as an optimisation problem: minimise (eg. by least-squares) the discrepancy between the observed camera features and their positions predicted by the perspective eqs. We know for $i = 1, \dots, N$ the camera image p_i of world point P_i ⇒ $\{ p_i = \frac{1}{z_i} M P_i \}_{i=1}^N$ is an overconstrained ($N >$ #params) system of nonlinear eqs, so minimise $\sum_{i=1}^N \| p_i - \frac{1}{z_i} M P_i \|^2$ by gradient descent, Newton...
- Photogrammetry: engineering field whose aim is to recover quantitative geometric information from ≥ 1 pictures. Applications: cartography, military intelligence, city planning, etc.

CH. 10: THE GEOMETRY OF MULTIPLE VIEWS

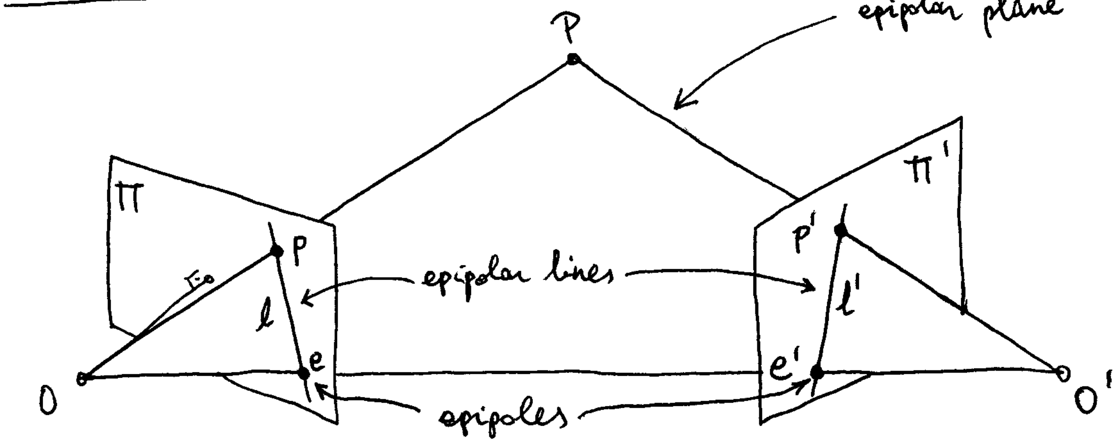
Depth is not directly accessible in a single image, but it can be measured through triangulation from ≥ 2 images:

- Most animals have ≥ 2 eyes or move their head to detect depth
- An autonomous robot is equipped with a stereo or motion analysis system.

Objective: understand the geometric & algebraic constraints that hold among multiple views of the same scene:

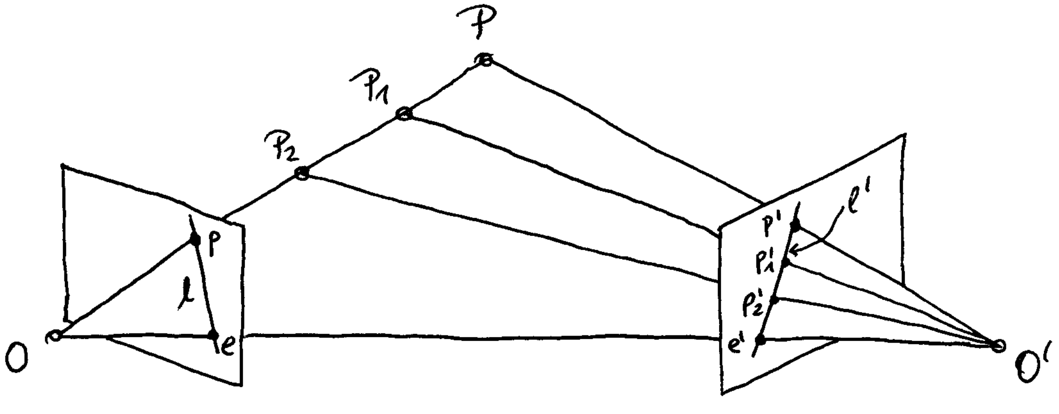
- 2 views: epipolar constraint, represented by 3×3 essential/fundamental matrix
- 3 views: $3\times3\times3$ trifocal sensor
- etc.

10.1. 2 VIEWS



- Consider the normalised camera frames Π, Π' (we could also draw them behind the pinholes but it is simpler in front).
- World point P observed by 2 cameras (with optical centres O, O' and retinas π, π') as images p, p'.
- $P, O, O', p, p' \in$ epipolar plane defined by rays $\overrightarrow{OP}, \overrightarrow{O'P}$.
- Epipole e' = projection of optical centre O on retina π' (ditto for e), ie, where one camera is seen by the other.
- Epipolar line l' associated with p: intersection of epipolar plane and retina π'; contains p' and the epipole e'.

- Epipolar constraint: if p and p' are images of the same world point then p' must lie on the epipolar line associated with p. In other words, the world point and the two optical centers lie on the same plane, thus given a point in one image, its correspondent must lie on a known line in the other image.
- Application: searching for corresponding points. If we have a stereo rig with calibrated cameras (known intrinsic & extrinsic parameters) then if we know the image p in one camera, we can determine the epipolar line l' where the other image p' must lie and limit our search there.



* Calibrated case: intrinsic parameters known, so $p = \hat{p}$. Epipolar constraint means:

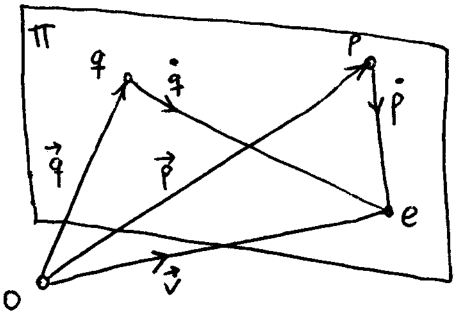
$$\overrightarrow{Op}, \overrightarrow{O'p'}, \overrightarrow{OO'} \text{ coplanar} \Leftrightarrow \overrightarrow{Op}\cdot(\overrightarrow{OO'} \times \overrightarrow{O'p'}) = 0 \overset{\text{homog.}}{\underset{\text{coord.}}{=\!=}} \underset{\overset{\uparrow}{\overrightarrow{Op}}}{p} \cdot (\underset{\overset{\uparrow}{\overrightarrow{OO'}}}{t} \times (\underset{\overset{\uparrow}{\text{Rotation } \pi' \to \pi}}{R} p')) \Leftrightarrow$$

$\boxed{p^T E p' = 0}$ with the essential matrix $E = [t_\times]R$:

- E is defined in terms of normalized coordinates (and independent of p, p').
- $[a_\times]$ = skew-symmetric matrix such that $[a_\times]x = a \times x$; $[a_\times] = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$.
- E has 5 degrees of freedom (3 from R, 2 from t because scale invariant).
- Ep' = epipolar line associated with p' in the first image
  $E^T p$ = " " " " p " " second " (by symmetry).
- $E^T e = -R^T[t_\times]e = 0$ because $t \parallel e \Rightarrow$ E is singular; in fact, rank(E) = 2 and its 2 singular values are equal.
- Only one epipolar line goes through any image point, except for the epipole. All epipolar lines of one camera go through the other's epipole.
- Ex: pure translation ("eyes"): $t = (t_1\ 0\ 0)^T$, $R = I \Rightarrow v = v'$ (horizontal line).

Small motions: moving camera with translational velocity $\vec V$ and rotational velocity $\vec\omega$. Call $\dot p = \begin{pmatrix}\dot u\\ \dot v\\ 0\end{pmatrix}$ the velocity of point p (motion field). Then, rewriting to first order $p^T E p' = 0$ for two frames separated by an infinitesimal time interval δt:

$$\left.\begin{aligned}\vec t &= \delta t\cdot\vec V\\ R &= I + \delta t\cdot[\omega_\times]\\ p' &= p + \delta t\cdot\dot p\end{aligned}\right\} \xrightarrow[\text{terms}]{\text{neglecting higher-order}} \left\{\begin{array}{l} p^T[V_\times][\omega_\times]p - (p\times\dot p)\cdot V = 0\\ \text{instantaneous form of epipolar constraint}\end{array}\right.$$



Pure translation: $\omega = 0 \Rightarrow (p\times\dot p)\cdot V = 0 \Leftrightarrow$ $p, \dot p, V$ coplanar $\Rightarrow$ motion field $\dot p$ points towards the focus of expansion e (infinitesimal epipole).

* Uncalibrated case: intrinsic parameters unknown, so $p = K\hat p$, $p' = K'\hat p'$ (3×3 calibration matrices K, K', normalised image coord. $\hat p, \hat p'$) $\Rightarrow \hat p^T E \hat p' = 0 \Rightarrow$

$\boxed{p^T F p' = 0}$ with the fundamental matrix $F = K^{-T} E K'^{-1}$:

- F is defined in terms of pixel coordinates
- F has 7 d.o.f. (defined up to scale); (10.6): parametrisation in terms of epipoles' coord.
- Fp' = epipolar line associated with point p' in the first image (ditto F^Tp for p in 2nd)
- rank(F) = 2, $Fe' = 0$, $F^Te = 0$.

* Weak calibration

- The problem of estimating F from a set of n correspondences between 2 images taken by cameras with unknown intrinsic parameters.
- In principle, 8 points in general position are enough to determine the 9 elements of F (since the solution is defined up to scale we can set $F_{33} = 1$): solve linear system on F: $p_i^T F p_i' = 0$, $i = 1, \dots, 8$ (eight-point algorithm by Longuet-Higgins), see eq. after (10.7). To avoid numerical instabilities, the coord. of the corresponding points should be normalised so that the entries of the eqs. $p_i^T F p_i' = 0$ are comparable in size.

- It is more robust to use $n > 8$ correspondences: $\min_{\|F\|=1} \sum_{i=1}^{n} (p_i^T F p_i')^2$, quadratic in F.
- Both methods ignore the rank-2 property of F. Possible approach: make zero the smallest singular value of F.
- $p^T F p'$ is proportional to the algebraic distance from p to the epipolar line Fp' (or from p' to $F^T p$), so can also minimize the mean squared geometric distance between the image points and the corresponding epipolar lines: $\sum_{i=1}^{n} \left(d^2(p_i, Fp_i') + d^2(p_i', F^T p_i) \right)$.

FIG. 10.4

- We can also determine F by first calibrating each camera, but this requires estimating 11 parameters per camera.

CH. 11: STEREOPSIS

- Design & implementation of algorithms that mimic our ability to detect depth from the image recorded by our two eyes.
- Applications in robot navigation, cartography, aerial reconnaisance, computer graphics (construction of 3D scene models), etc.
- Stereo vision involves two processes:
  - Fusing features observed by ≥ 2 eyes (hard)
  - Reconstructing their 3D preimage (easy)

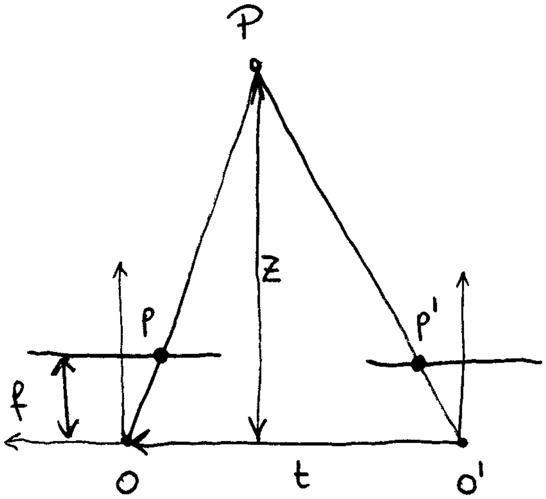
FIG. 11.2

FIG. OF STEREO PAIR & CORRESPONDENCES

- Assume calibrated cameras (intrinsic & extrinsic parameters known wrt fixed world frame).

11.1. RECONSTRUCTION: given a calibrated stereo rig and 2 matching points p, p': reconstruct the scene point P.

* 2D case: disparity = difference in retinal position $d = p' - p$.

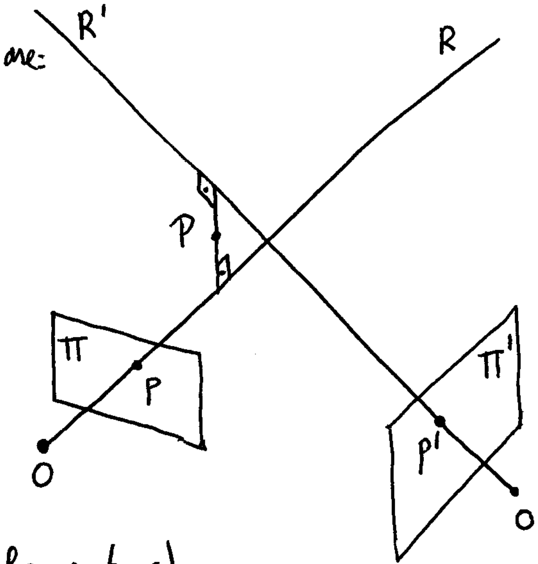


We obtain the depth z by triangulation, ie, by intersecting the rays Op and O'p'. The triangles $\widehat{pPp'}$ and $\widehat{OPO'}$ are similar so:

$$\frac{t + p - p'}{z - f} = \frac{t}{z} \Rightarrow z = f\,\frac{t}{d} \Rightarrow$$

The depth is inversely proportional to the disparity (distant objects seem to move more slowly than close ones).

* 3D case: in theory, P is at the intersection of the two rays $R = Op$ and $R' = Op'$, but in practice they don't intersect due to calibration and feature localisation errors. Approaches:



- Midpoint of segment ⊥ R, R': in coord. of Π (where t, Q are the translation, rotation Π'→Π) the rays are:

  $R: \ ap \ (a \in \mathbb{R}), \quad R': \ t + bQ^T p' \ (b \in \mathbb{R})$

  $\Rightarrow \ w = p \times Q^T p'$ is $\perp \ p, p'$

  Line through point ap (for some fixed a) parallel to w: $ap + cw \ (c \in \mathbb{R})$

  Thus, the endpoints of the segment are ap and $t' + bQ^T p'$ satisfying:

  $ap + c(p \times Q^T p') = t + bQ^T p'$ (3 linear eqs. for a, b, c).

  The triangulated point P is the midpoint of the segment (in coord. of Π).

- Given projection matrices M, M': $\left.\begin{matrix} p = \frac{1}{z} MP \\ p' = \frac{1}{z'} M'P \end{matrix}\right\} \Rightarrow \begin{pmatrix} [p_\times] M \\ [p'_\times] M' \end{pmatrix} P = 0$ overconstrained linear system, solvable by least squares.

- Point Q which minimises $d^2(p, q) + d^2(p', q')$ (nonlinear opt.)

Generalise to >2 cameras

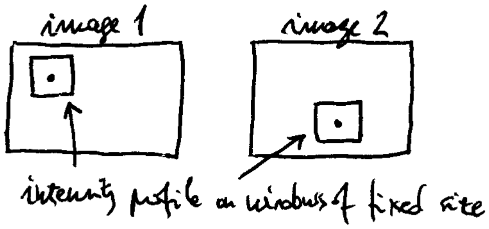
11.3. Binocular Fusion

FIG. 11.2

- Hard to know what point pairs correspond in both images because there are many pixels and many features (eg. edges) in every image.
- We assume that most scene points are visible from both cameras and that corresponding image regions look similar.
- Approach: correlation: compare the intensity profile in the neighbourhood of potential matches by using the correlation:

  $$\text{correl}(u, v) = \left(\frac{u - \bar{u}}{\|u - \bar{u}\|}\right) \cdot \left(\frac{v - \bar{v}}{\|v - \bar{v}\|}\right) \in [-1, 1] = \text{cosine of angle between normalised vectors.}$$

  It can be computed efficiently by recursion (ex. 11.7).



11.4. Using More Cameras further reduces the ambiguity, eg. a 3rd image can be used to check hypothetical matches from the first 2 images. FIG. 11.16

CH. 12: AFFINE STRUCTURE FROM MOTION

- The problem of estimating the 3D shape of a scene from multiple pictures.
- Particular case: stereopsis (intrinsic param. known, extrinsic param. determined wrt fixed world coord. sys.), simpler.
- Cameras' positions and possibly intrinsic param. unknown and may change over time.
- Applications:
  - Image-based rendering: video recorded using hand-held camera (possibly zooming) is used to capture the shape of an object and then render it under new viewing conditions.
  - Active vision systems: capable of dynamically modifying the camera parameters (intrinsic / extrinsic), eg. mobile robot navigation.
- We assume the projections of n points have been matched across m pictures (correspondence problem) and focus on the purely geometric structure-from-motion problem:
  - estimate 3D positions of corresponding scene points (scene structure)
  - estimate projection matrices of the cameras (motion of the points wrt the cameras)
- Assumption: scene's relief $\ll$ depth so the perspective projection is approximated by the affine projection.

  Given n fixed points P_j ($j=1,\dots,n$) observed by m affine cameras we have their images (nonhomogeneous coord.) $p_{ij} = M_i \begin{pmatrix} P_j \\ 1 \end{pmatrix} \underset{2\times 4}{\overset{M_i = (A_i\ b_i)}{=\!=}} A_i P_j + b_i$, $\begin{matrix} i=1,\dots,m \\ j=1,\dots,n. \end{matrix}$

  Structure-from-motion problem: estimate $\{M_i\}_{i=1}^{m}$ and $\{P_j\}_{j=1}^{n}$ from $\{p_{ij}\}_{i,j=1}^{m,n}$.

  We have $2mn$ eqs. and $8m + 3n$ unknowns (sufficiently constrained for large enough m, n).
- Affine ambiguity: M_i, P_j are solutions $\Rightarrow$ $M_i Q$, $Q^{-1}\begin{pmatrix} P_j \\ 1 \end{pmatrix}$ are solutions, where $Q = \begin{pmatrix} C & \mathbf{d} \\ 0^T & 1 \end{pmatrix}$ is an arbitrary affine transformation matrix ($C_{3\times 3}$ nonsingular, $\mathbf{d}_{3\times 1}$).

  Thus, the solutions are only defined up to an affine transformation.

  If the intrinsic param. are known ($K_i = I$ and use normalized image coord. $\hat{p}_{ij}$) then M_i must obey additional constraints which can eliminate the ambiguity. This suggests:

  1. Find a 3D reconstruction up to an affine transformation using ≥ 2 views (essential part).
  2. Use additional views and constraints from known calibration to find a unique 3D reconstruction (Euclidean upgrade).

  The ambiguity still occurs for perspective projection (projective ambiguity).

12.2. Affine Structure and Motion from Two Images

Algebraic motion estimation: $\left.\begin{array}{l} p = AP + b \\ p' = A'P + b' \end{array}\right\} \Leftrightarrow \begin{pmatrix} A & p-b \\ A' & p'-b' \end{pmatrix}\begin{pmatrix} P \\ -1 \end{pmatrix} = 0$. A nontrivial solution exists $\Leftrightarrow \begin{vmatrix} A & p-b \\ A' & p'-b' \end{vmatrix} = 0 = \alpha u + \beta v + \alpha' u' + \beta' v' + \delta = (u\ v\ 1)\, F \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix}$ with $p = \begin{pmatrix} u \\ v \end{pmatrix}, p' = \begin{pmatrix} u' \\ v' \end{pmatrix}$, $F = \begin{pmatrix} 0 & 0 & \alpha \\ 0 & 0 & \beta \\ \alpha' & \beta' & \delta \end{pmatrix}$ and $\alpha, \beta, \alpha', \beta', \delta$ are constants dependent on A, b, A', b'.

- Affine epipolar constraint: $(u\ v\ 1)\, F \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0$ with affine fundamental matrix $F = \begin{pmatrix} 0 & 0 & \alpha \\ 0 & 0 & \beta \\ \alpha' & \beta' & \delta \end{pmatrix}$:
  - Given p, p' must lie on the epipolar line l' defined by $\alpha' u' + \beta' v' + (\alpha u + \beta v + \delta) = 0$, and vice versa.
  - The epipolar lines in each image are parallel to each other. FIG. 12.5

- We estimate the projection matrices from the epipolar constraint. Write $Q = \begin{pmatrix} C & D \\ 0^T & 1 \end{pmatrix}$ invertible and $M = (A\ \ b)Q = (AC\ \ AD + b)$ with $\underset{2\times 3}{A} = \begin{pmatrix} a_1^T \\ a_2^T \end{pmatrix}$, $b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ and likewise $M' = (A'\ b')Q \ldots$

  The epipolar constraint is $0 = \begin{vmatrix} a_1^T C & u - a_1^T D - b_1 \\ a_2^T C & v - a_2^T D - b_2 \\ a_1'^T C & u' - a_1'^T D - b_1' \\ a_2'^T C & v' - a_2'^T D - b_2' \end{vmatrix} = \begin{vmatrix} SC & q - SD - r \\ c^T & v' - d \end{vmatrix}$ with $S = \begin{pmatrix} a_1^T \\ a_2^T \\ a_1'^T \end{pmatrix}$, $q = \begin{pmatrix} u \\ v \\ u' \end{pmatrix}$, $r = \begin{pmatrix} b_1 \\ b_2 \\ b_1' \end{pmatrix}$, $c = C^T a_2'$, $d = a_2'^T D + b_2'$. If S is nonsingular, choose $C = S^{-1}$ and $D = -S^{-1} r$ and write $c = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$; by doing this we are choosing one particular Q and so one particular solution for $\{(M_i, P_j)\}_{i,j}$. [Pf. $I = SS^{-1} = \begin{pmatrix} A \\ a_1'^T \end{pmatrix} S^{-1} = \begin{pmatrix} AS^{-1} \\ a_1'^T S^{-1} \end{pmatrix}$] Thus $M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$, $M' = \begin{pmatrix} 0 & 0 & 1 & 0 \\ a & b & c & d \end{pmatrix}$ and the epipolar constraint takes the form $\begin{vmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & u' \\ a & b & c & v'-d \end{vmatrix} = -au - bv - cu' + v' - d = 0$. The coefficients a, b, c, d can be estimated by linear least squares given enough point correspondences.

- Once these coeffs. have been estimated, the 2 projection matrices are known and the 3D position of any point $\tilde{P}$ can be estimated from its image coord. by solving the system of 4 linear eqs $\begin{pmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & u' \\ a & b & c & v'-d \end{pmatrix}\begin{pmatrix} \tilde{P} \\ -1 \end{pmatrix} = 0$. (The first 3 rows yield $\tilde{P} = \begin{pmatrix} u \\ v \\ u' \end{pmatrix}$ without the need for the coeffs., but using all 4 eqs. is more robust.)

- If S is (close to) singular, try $S = \begin{pmatrix} a_1^T \\ a_2^T \\ a_2'^T \end{pmatrix}$. If this is also singular, the two image planes are parallel and the scene structure cannot be recovered. [Proof: both a_1' and a_2' are l.c. of a_1 and a_2 $\Rightarrow \underset{(B \text{ invertible})}{A' = BA} \Rightarrow \left\{\begin{array}{l} p = AP + b \\ p' = BAP + b' \end{array}\right\} \Rightarrow \left\{\begin{array}{l} AP = p - b \\ AP = B^{-1}(p' - b') \end{array}\right.$. See FIG. 12.5.]

12.3. Affine Structure and Motion from Multiple Images

* Tomasi–Kanade factorisation approach: given m images of n points $P_1, \dots, P_n$, define the $2m \times n$ data matrix $D = (q_1 \cdots q_n) = AP$ with $q_i = \begin{pmatrix} p_{1i} \\ \vdots \\ p_{mi} \end{pmatrix}$ for point P_i, $P = (P_1 \cdots P_n)$ is $3 \times n$, and $A = \begin{pmatrix} A_1 \\ \vdots \\ A_m \end{pmatrix}$ is $2m \times 3$. Thus rank$(D) = 3$, though in practice D has full rank due to image noise, errors in localisation of image points, and the fact that cameras are not really affine.

If $D = UWV^T$ is the SVD of D, with $U_{2m \times n}$ orthogonal, $W_{n \times n}$ diagonal with singular values $w_{ii} \geq 0$, and $V_{n \times n}$ orthogonal, then $D_3 = U_3 W_3 V_3^T$ (corresp. to the largest 3 singular values) is the best rank-3 approximation to D in the Frobenius norm sense, ie; it minimises:

$$\sum_{i,j} \| p_{ij} - A_i P_j \|^2 = \sum_j \| q_j - A P_j \|^2 = \| D - AP \|_F^2 \quad \text{over } A_{2m \times 3},\ P_{3 \times n}.$$

Thus, we can take $A = U_3$ and $P = W_3 V_3^T$ as representative of the (affine) camera motion and scene shape, respectively. Note again the affine ambiguity: $D = (AQ)(Q^{-1}P)$ is also a solution.

~~12.4. FROM AFFINE TO EUCLIDEAN IMAGES~~ Advantages: reliable, simple.

Disadvantages: uses affine camera model (an extension exists for perspective projection, but it doesn't always converge to the correct solution); all points must be visible in all images.

* It is also pssible to use a 2-view algorithm image by image, carefully selecting correspondences

12.4. From Affine to Euclidean Images

* Euclidean constraints & calibrated affine cameras:

- Weak perspective: $\underset{2\times4}{M} = (A\ b) = \frac{1}{z_r} \begin{pmatrix} k & s \\ 0 & 1 \end{pmatrix} (R_2\ t_2)$ where R_2, t_2 contain rows 1-2 of rotation, translation

  calibrated ⇒ use normalised image coord $(k=1, s=0) \Rightarrow M = \frac{1}{z_r}(R_2\ t_2) \Rightarrow A_{2\times3}$ has orthogonal rows.

- Ortographic perspective: $z_r = 1 \Rightarrow A_{2\times3}$ has orthonormal rows. (cannot recover depth z).

* Other constraints: all cameras may have the same intrinsic parameters.

* Computing Euclidean upgrades from multiple views: Euclidean upgrade = affine transformation $Q_{4\times4} = \begin{pmatrix} C & D \\ 0^T & 1 \end{pmatrix}$ that maps the affine shape onto the Euclidean shape: $\hat{M} = (AC\ \ AD + b)$, $\hat{P} = C^{-1}(P - D)$.

Assume $m \geq 3$ images and consider the ortographic case for simplicity. Then the Euclidean constraints imply the following overconstrained system of $3m$ eqs:

$$\left.\begin{array}{l}\hat{a}_{i1}^T \hat{a}_{i2} = 0 \\ \|\hat{a}_{i1}\|^2 = \|\hat{a}_{i2}\|^2 = 1\end{array}\right\} \Leftrightarrow \left.\begin{array}{l} a_{i1}^T CC^T a_{i2} = 0 \\ a_{i1}^T CC^T a_{i1} = 1 \\ a_{i2}^T CC^T a_{i2} = 1 \end{array}\right\} \quad i = 1, \dots, m$$

- quadratic in C, solvable by nonlinear LSQ
- linear in CC^T, solvable by linear LSQ, then obtain C by Cholesky decomposition (if CC^T pos. def.).

FIG. 12.7

We can only recover Q up to rotations of C (since CC^T is invariant to them) and translations D, ie, up to rigid motions of the world $\Leftrightarrow$ given the images we cannot know the longitude/latitude, or what is N-S-E-W; for that we need absolute coord. of some world points.

12.5. Affine Motion Segmentation

Now we allow the scene points to move independently from each other. Assume we have k rigid objects moving around and we take an image sequence of m frames.

- By reducing the data matrix to its row-echelon form and applying connected-components we obtain one group of points corresponding to each object; but not robust with noise.
- Shape interaction matrix (Costeira-Kanade): assume we know the assignments of points to objects and write for each object $i = 1, \dots, k$:

$$\text{Data matrix } \underset{2m \times n_i}{D^{(i)}} = \begin{pmatrix} p_{11}^{(i)} & \cdots & p_{1n_i}^{(i)} \\ \vdots & & \\ p_{m1}^{(i)} & \cdots & p_{mn_i}^{(i)} \end{pmatrix}, \quad \underset{2m \times 4}{M^{(i)}} = \begin{pmatrix} M_1^{(i)} \\ \vdots \\ M_m^{(i)} \end{pmatrix}, \quad \underset{4 \times n_i}{P^{(i)}} = \begin{pmatrix} P_1^{(i)} & \cdots & P_{n_i}^{(i)} \\ 1 & \cdots & 1 \end{pmatrix} \Rightarrow$$

$$\underset{2m \times n}{D} = (D^{(1)} \cdots D^{(k)}), \quad \underset{2m \times 4k}{M} = (M^{(1)} \cdots M^{(k)}), \quad \underset{4k \times n}{P} = \operatorname{diag}(P^{(1)}, \dots, P^{(k)}), \quad n = \sum_{i=1}^{k} n_i,$$

$D = MP$, thus $\operatorname{rank}(D) \leq 4k$. Each object traces a different 4D subspace when undergoing rigid motion (4 dof per object, or less for planes (3) or lines (2)).

Compute the SVD of D for its $4k$ largest s.v. $D = U_{4k} W_{4k} V_{4k}^T$ (= in the noiseless case, $\simeq$ in the noisy case). The $4k$-dim subspace spanned by the rows of D is the same as that spanned by the rows of either P (unknown in practice) or V_{4k}^T (known).

Thus the $n \times n$ matrix $Z = V_{4k} V_{4k}^T$ (<u>shape interaction matrix</u>):

- is uniquely defined (pf- $Z = \sum v_i v_i^T$)
- maps $\mathbb{R}^n$ onto the $4k$-dim subspace spanned by the rows of V_{4k}^T
- is <u>block diagonal</u> because P is block diagonal, thus the cluster structure is more obvious in Z than in D (pf- define $M = UW_{4k}^{\frac{1}{2}}Q$, $P = Q^{-1} W_{4k}^{\frac{1}{2}} V_{4k}^T$ up to affine transf. $\Rightarrow V_{4k} V_{4k}^T = P^T (PP^T)^{-1} P$.)

In practice the points from each object are not ordered within D or Z, so Z is not block diagonal. One approach to recover the ordering is to minimise the sum of the squares of the off-diagonal block entries over all rows & columns permutations; blocks (clusters) are then detected in the permuted matrix Z. Once we know which point belongs to which object, we can recover each object independently. FIG. 12.8

Does not work with perspective projection, only affine.

Other approach: first segment objects, then apply Tomasi-Kanade factorisation to each object.

CH. 14: SEGMENTATION BY CLUSTERING

- Segmentation (= grouping, perceptual organisation) is the mid-level vision problem of:
  - Obtaining a compact representation of helpful information in the image (generally speaking)
  - More specifically, breaking the image into regions of coherent colour & texture; often a first step for object recognition.

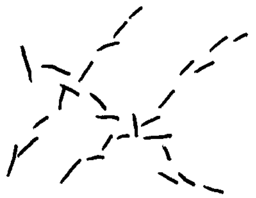
  It is not well defined, though in some applications (eg. cytology) it is possible to define and evaluate it more specifically.

- Clustering: general problem of splitting a data set into meaningful groups (clusters). Not well defined either and not clear how to evaluate the quality of a clustering result, but lots of algorithms in statistics and machine learning which are useful in practice.

- Segmentation as clustering: cluster pixels (or other image features: edges, lines, etc.) into meaningful segments, using a clustering algorithm.

  Ch. $\substack{15\\16}$: segmentation as fitting (fit a model such as a line or Gaussian mixture).

- Examples:
  - cluster greyscale or colour image into segments.
  - Cluster a set of edges into lines (edge = (location, orientation)):



- What criteria should a segmentation method use to decide which pixels/features belong together?

14.2. HUMAN VISION: GROUPING & GESTALT

- Context affects how things are perceived
- Gestalt school of psychology: grouping = tendency of the visual system to perceive certain things in a picture together.
  - Ex: segments of different length are perceived differently in different contexts. FIG. 14.2
  - Humans often separate the image into figure (the significant object) and ground (the background on which the figure lies).
  - Factors which predispose a set of elements to be grouped: PAG. 306-307
  - Also happens in the perception of speech & sound.
  - Ecological argument: things are grouped together because doing so produces representations that are helpful for the visual world:

    * common fate: the components of objects tend to move together

    * symmetry: many real objects are roughly symmetric

    * continuity is useful to explain occlusion { - occluding object FIG. 14.6
    - illusory contour FIG. 14.8

- Unfortunately it is hard to use these rules to derive sequential algorithms.

14.3. APPLICATIONS: SHOT BOUNDARY DETECTION & BACKGROUND SUBTRACTION

Background subtraction: anything that doesn't look like a known background is interesting.

- In many applications, objects appear on a largely stable background. Ex: detecting parts on a conveyor belt; counting cars in an overhead view of a road (bkg = road); HCI (bkg = room).
- Idea: subtract background estimate from image and look for large absolute values.
- Alg. 14.1: background subtraction by moving average
  - Form a background estimate $B^{(0)}$ (eg. by taking a picture)
  - At each frame F:
    * update background estimate: $B^{(n+1)} = w_a F + \sum_i w_i B^{(n-i)}$ for some weights $w_a, \{w_i\}$ with weights decreasing towards the past.
    * subtract background estimate from F and report the value of each pixel j where $|F_j - B_j^{(n+1)}| > \Theta$ for some threshold Θ.

It has some ability to adapt to changes in the background (eg. road getting rainy).

Moving average = temporal filter (eliminates high frequencies of change).

problem: an object that spends a lot of time in one place can bias the average seriously.

FIGS. 14.9, 14.10

Shot boundary detection:

- Long sequences of video consist of several shots (editing) = much shorter subsequences that show largely the same object (eg. TV news: anchor vs event footage).
- Want to detect the shot boundaries & represent each shot by a key frame (useful to summarise the content of a video for quick browsing).
- Alg. 14.2: shot boundary detection using interframe differences

  Boundary occurs whenever $\mathrm{dist}(F_{n+1}, F_n) > \Theta$.
- Some definitions of distance:
  - Frame differencing: sum of the squares of the differences at each pixel. Sensitive to camera shakes.
  - Histogram distance: between colour histograms of each frame. Insensitive to spatial arrangement of colours in the frame (thus to small shaking).
  - Block comparison: split frame in blocks, compare histograms in each block and take the largest block difference.
  - Edge differencing: instead of comparing pixel values, compare edges.

14.4 Image Segmentation by Clustering Pixels

* Image features:

- Pixel intensity or colour; don't use RGB, use a uniform colour space such as LUV or LAB where Euclidean distances between feature vectors match perceptual distances between colours (see 6.3.2).
- Pixel position (i,j) in the image (nearby pixels are likely in the same object).
- Histogram of colours or grey levels (for distance between clusters)
- Texture: compute the histogram over a reasonably sized window of the outputs of a collection of filters spanning a range of scale and orientation.

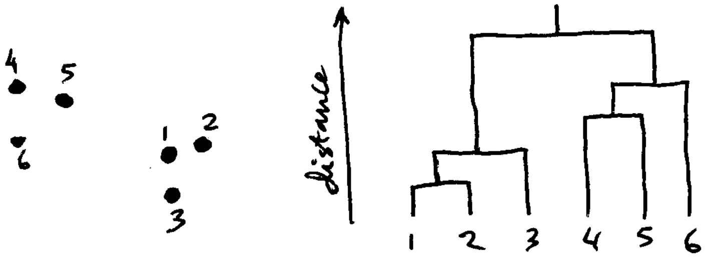
The distance in the feature space may be represented with a weighted Euclidean distance.

* Hierarchical clustering methods: they naturally yield a hierarchy of clusters:

- Divisive (top-down): partition the data set into groups that are dissimilar, recursively partition groups.
- Agglomerative (bottom-up): merge data points that are similar into groups, recursively merge groups.

Both are greedy algorithms: at each step they find the best split or merge.

- What is a good intercluster distance?
  - Single-link clustering: distance between closest elements; tends to yield extended clusters.
  - Complete-link " : distance between farthest elements; tends to yield rounded clusters.
  - Average-link " : average distance between all elements; also rounded clusters.
- How many clusters are there? Difficult to tell; may examine the dendrogram generated by the algorithm (hard: too many pixels) or use some threshold on the distance or number of clusters.



agglomerative single-link clustering

* k-means

- Idea: write an objective function ϕ that expresses goodness of clustering and optimise it over clusterings.
- $$\phi(\text{clusters}) = \sum_{i \in \text{clusters}} \sum_{\substack{j \in i\text{th} \\ \text{cluster}}} \|x_j - c_i\|^2$$

data point $x_j \in \mathbb{R}^D$

cluster centre c_i = average of all points in cluster i

$\Rightarrow$ minimise ϕ over $c_1, \dots, c_k \in \mathbb{R}^D$ (so need to assume there are k clusters).

Can't test all clusters combinations (huge number) so iterative algorithm.

- Algorithm: choose k data points as cluster centres; iterate till there is no change in cluster centres:
  - allocate each data point to its closest cluster centre (needs care to avoid empty clusters)
  - update cluster centres as the means of each cluster [prove it]

Each step decreases ϕ or leaves it unchanged; ϕ is bounded below. Thus the algorithm converges – but to one of the many local optima of ϕ (not necessarily the global one).

FIGS. 14.13 – 14.15

14.5. SEGMENTATION BY GRAPH-THEORETIC CLUSTERING

- Idea:
  - Build a graph where vertices = data points, edges' weights = given by a similarity (affinity) function which is large if the vertices are similar and small otherwise. ⇒ needs $\left\{\begin{array}{l}\text{graph}\\ \text{affinity}\end{array}\right.$
  - Cut the graph into good subgraphs (= clusters) ≡ many strong edges within subgraphs, few (which we cut) between subgraphs. ⇒ needs objective function for graph cut.

FIG. 14.16

  - Affinity measures: usually Gaussian $e^{-\frac{1}{2}\left(\frac{d(x,y)}{\sigma}\right)^2}$ where the distance $d(x,y)$ is modulated by a scale parameter. FIG. 14.18

* Spectral clustering

- Idea: the clustering structure in the data is enhanced in the eigenvectors of the affinity matrix A. In an ideal case, $a_{ij} = 0$ for points i,j in different clusters, thus A is block-diagonal (if we reorder the data by clusters). Then, the eigenvectors of A are eigenvectors of the blocks padded out with zeroes and act as cluster indicators. Ex: for $k=2$ clusters:

$$A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix};\ \text{if } u_1 \text{ is eigenvector of } A_1 \Rightarrow A_1 u_1 = \lambda_1 u_1 \Rightarrow A \underbrace{\begin{pmatrix} u_1 \\ 0 \end{pmatrix}}_{} = \lambda_1 \begin{pmatrix} u_1 \\ 0 \end{pmatrix}$$

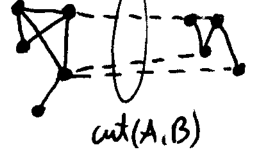
$$\text{elements} \left\langle \begin{array}{l} \text{zero, not cluster 1} \\ \text{nonzero, cluster 1} \end{array} \right.$$

In practice, A is not exactly block-diagonal; all the elements of each eigenvector are nonzero. Threshlding them can still recover the clusters in some cases.

FIG. 14.19 – 14.20

* Normalised cuts

- Idea: cut the graph into 2 connected components such that the cost of the cut is small wrt the uncut edges in each component: $\min_{A,B} \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$ where:
  - cut(A,B) = sum of weights of edges crossing between A and B.
  - assoc(A,V) = " " " " " " " " V (V = all vertices).



One can show that minimising the normalised cut is equivalent to $\min\limits_{y_i \in \{1,-b\}} \frac{y^T(D-A)y}{y^T D y}$ where:

- $y = (y_1, \dots, y_N)^T$ with $y_i \in \{1, -b\}$ for a certain $b \in \mathbb{R}$; y_i indicates the cluster of point i.
- D = degree matrix = $\mathrm{diag}(d_1, \dots, d_N)$ with $d_i = \sum_{j=1}^{N} A_{ij}$.

This is an integer programming problem, for which no efficient exact algorithms are known (it is NP-complete). We can find an approximate solution by relaxing y to be real, then thresholding it:

$$\min_{y \in \mathbb{R}^N} \frac{y^T(D-A)y}{y^T D y} \overset{\text{[Pf. equate grad to 0]}}{\Leftrightarrow} (D-A)y = \lambda D y, \overset{z = D^{\frac{1}{2}} y}{\Leftrightarrow} Nz = \mu z \text{ s.t.} \begin{cases} z^T z = 1 \\ z^T D^{\frac{1}{2}} 1 = 0 \end{cases}, \text{ where:}$$

- $y = 0$ and $y = \vec{1}$ (vector of ones) give trivial minima so we need to remove them. (thus the constraints on z).
- $D - A$ = <u>graph Laplacian</u>, with eigenvalue 0 for eigenvector $\vec{1}$.
- $N = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ = <u>normalised affinity matrix</u>, with eigenvalue $\mu_1 = 1$ for eigenvector $D^{\frac{1}{2}}\vec{1}$
  $\mathcal{L} = I - N$ = normalised graph Laplacian, with eigenvalues $\lambda = 1 - \mu$.

Thus, the solution is the second leading eigenvector of N. Finding the optimal threshold can be done easily since there are only N different cuts.

To obtain $k > 2$ clusters, we could either recursively partition A and B, or compute the leading k eigenvectors of N. In the latter case we need to untangle the eigenvectors because the multiplicity of $\mu = 1$ equals the number of clusters in the ideal case, so its eigenspace is degenerate, eg. for $k = 3$:

$$D^{\frac{1}{2}}\begin{pmatrix} 1_A \\ 0_B \\ 0_C \end{pmatrix}, \ D^{\frac{1}{2}}\begin{pmatrix} 0_A \\ 1_B \\ 0_C \end{pmatrix}, \ D^{\frac{1}{2}}\begin{pmatrix} 0_A \\ 0_B \\ 1_C \end{pmatrix} \text{ are eigenvectors of } \mu = 1 \text{ for } N \text{ with } A = \begin{pmatrix} A_1 & & O \\ & A_2 & \\ O & & A_3 \end{pmatrix},$$

and a numerical eigensolver will return a different eigenbasis.

This suggests estimating the number of clusters by the multiplicity of $\mu = 1$, or in practice with noisy data, by looking for a big eigengap in the spectrum (which doesn't work well).

Advantages of spectral clustering:

- can work well with clusters having fancy shapes (wiggly) and with overlapping clusters
- can use many affinity functions

Disadvantages:

- it is heuristic and can often produce bad results
- high computational cost: $\Theta(N^3)$; can be reduced to $\Theta(N^2)$ if sparse affinity matrix

FIGS. 14.23 - 14.24

Mean-shift clustering

Represent each pixel $\mathbf{x}_n$, $n = 1, \dots, N$ by a feature vector as in spectral clustering, typically position & intensity (i, j, I) or colour (i, j, L^*, u^*, v^*).

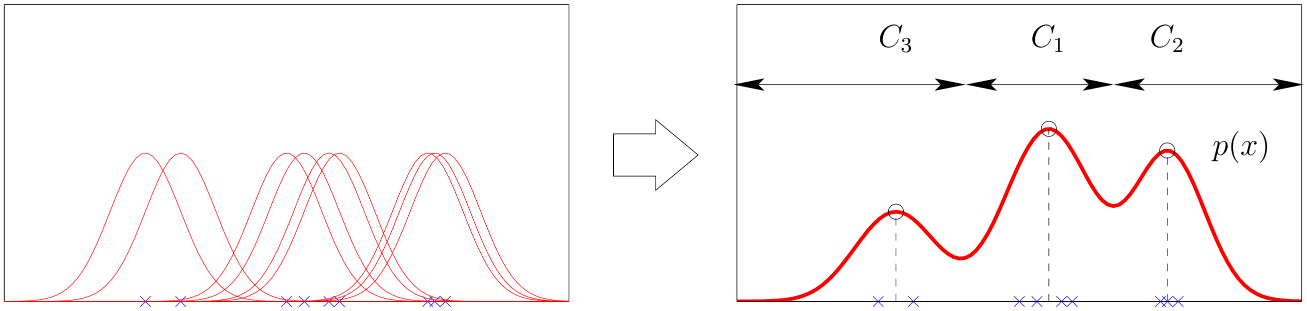
Idea: define a function that represents the density of the data set $\{\mathbf{x}_n\}_{n=1}^N \subset \mathbb{R}^D$, then declare each maximum as a cluster representative and assign each pixel to a maximum via the mean-shift algorithm.

Kernel density estimate (smooth multivariate histogram) with *bandwidth* σ:

$$p(\mathbf{x}) = \sum_{n=1}^{N} p(n) p(\mathbf{x}|n) = \frac{1}{N} \sum_{n=1}^{N} K\left(\frac{\mathbf{x} - \mathbf{x}_n}{\sigma}\right) \qquad \mathbf{x} \in \mathbb{R}^D$$

The kernel K satisfies $\int K(\mathbf{x})\, d\mathbf{x} = 1$ and $K(\mathbf{x}) \geq 0$ (so the kernel is a pdf). Typical kernels:

- Gaussian (infinite support): $K\left(\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right) \propto \exp\left(-\frac{1}{2}\left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2\right)$
- Epanechnikov (finite support): $K\left(\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right) \propto \begin{cases} 0, & \left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\| > 0 \\ 1 - \left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2, & \text{otherwise} \end{cases}.$



Mean-shift algorithm for Gaussian kde: maxima (also minima, saddle points) of $p(\mathbf{x})$ satisfy

$$\mathbf{0} = \nabla p(\mathbf{x}) \propto -\frac{1}{N} \sum_{n=1}^{N} e^{-\frac{1}{2}\left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2} \left(\frac{\mathbf{x} - \mathbf{x}_n}{\sigma^2}\right) \propto p(\mathbf{x}) \sum_{n=1}^{N} p(n|\mathbf{x})(\mathbf{x} - \mathbf{x}_n) \Longrightarrow \mathbf{x} = \sum_{n=1}^{N} p(n|\mathbf{x}) \mathbf{x}_n = \mathbf{f}(\mathbf{x})$$

with "shifts" $\mathbf{x} - \mathbf{x}_n$ (thus $\nabla p(\mathbf{x}) \propto$ mean shift) and posterior probabilities (by Bayes' th.)

$$p(n|\mathbf{x}) = \frac{p(\mathbf{x}|n)p(n)}{p(\mathbf{x})} = \frac{\exp\left(-\frac{1}{2}\left\|(\mathbf{x} - \mathbf{x}_n)/\sigma\right\|^2\right)}{\sum_{n'=1}^{N} \exp\left(-\frac{1}{2}\left\|(\mathbf{x} - \mathbf{x}_{n'})/\sigma\right\|^2\right)}.$$

This shows that the fixed points of $\mathbf{f}$ are stationary points of p, and suggests defining a fixed-point iterative scheme by starting from a data point $\mathbf{x}_n$ and iteratively applying $\mathbf{f}$ till convergence (and repeating this for all pixels). It is possible to prove that this algorithm converges from nearly any initial $\mathbf{x}$ to a maximum with linear convergence rate (in fact it is an EM algorithm).

Advantages:

- Nonparametric clustering: only need to set σ.
- No step size needed.

- Works well with clusters having complex shapes.
- The number of clusters is determined automatically by σ.

Disadvantages:

- The mean-shift iteration is slow.
- Large total computational cost: $\mathcal{O}(kN^2)$ where k = average number of mean-shift iterations per pixel ($k \approx$ 20–100). Accelerations are possible that produce almost the same segmentation.

```
Gaussian mean-shift (GMS) algorithm

for $n \in \{1, \dots, N\}$                                              For each data point
    $\mathbf{x} \leftarrow \mathbf{x}_n$                                  Starting point
    repeat                                                               Iteration loop
        $\forall n$: $p(n|\mathbf{x}) \leftarrow \frac{\exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_n)\sigma\|^2\right)}{\sum_{n'=1}^{N}\exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_{n'})/\sigma\|^2\right)}$
        $\mathbf{x} \leftarrow \sum_{n=1}^{N} p(n|\mathbf{x})\mathbf{x}_n$  Update x
    until $\mathbf{x}$'s update < tol
    $\mathbf{z}_n \leftarrow \mathbf{x}$                                  Maximum
end
connected-components($\{\mathbf{z}_n\}_{n=1}^{N}$)                        Clusters
```

```
Gaussian blurring mean-shift (GBMS) algorithm

repeat                                                                   Iteration loop
    for $m \in \{1, \dots, N\}$                                          For each data point
        $\forall n$: $p(n|\mathbf{x}_m) \leftarrow \frac{\exp\left(-\frac{1}{2}\|(\mathbf{x}_m-\mathbf{x}_n)/\sigma\|^2\right)}{\sum_{n'=1}^{N}\exp\left(-\frac{1}{2}\|(\mathbf{x}_m-\mathbf{x}_{n'})/\sigma\|^2\right)}$
        $\mathbf{y}_m \leftarrow \sum_{n=1}^{N} p(n|\mathbf{x}_m)\mathbf{x}_n$  One GMS step
    end
    $\forall m$: $\mathbf{x}_m \leftarrow \mathbf{y}_m$                    Update whole data set
until stop
connected-components($\{\mathbf{x}_n\}_{n=1}^{N}$)                        Clusters
```

Figure 1: Pseudocode. The "connected-components" step collects all equivalent but numerically slightly different points.

Mean-shift blurring clustering

Like mean-shift clustering, but actually move data points at each step. It obtains very similar segmentations to those of mean-shift clustering but quite faster (cubic convergence rate for Gaussian clusters): the total computational cost is still $\mathcal{O}(kN^2)$ but k is quite smaller ($k \approx$ 5). It is related to spectral clustering, since effectively the algorithm is iterating ($\mathbf{X} \leftarrow \mathbf{PX}$, update $\mathbf{P}$) where $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ and $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ (stochastic, random-walk matrix), and $A_{mn} = \exp\left(-\frac{1}{2}\|(\mathbf{x}_m - \mathbf{x}_n)/\sigma\|^2\right)$ are the Gaussian affinities and $\mathbf{D} = \operatorname{diag}\left(\sum_{n=1}^{N} A_{mn}\right)$ the degree matrix.

Exercises

1. Derive the mean-shift algorithm for a general kernel K.
2. Prove that the mean-shift algorithm is gradient ascent on $p(\mathbf{x})$ with an adaptive step size.
3. Derive the mean-shift algorithm for the Gaussian kde with a bandwidth that is a full covariance matrix $\boldsymbol{\Sigma}_n$ for each point.

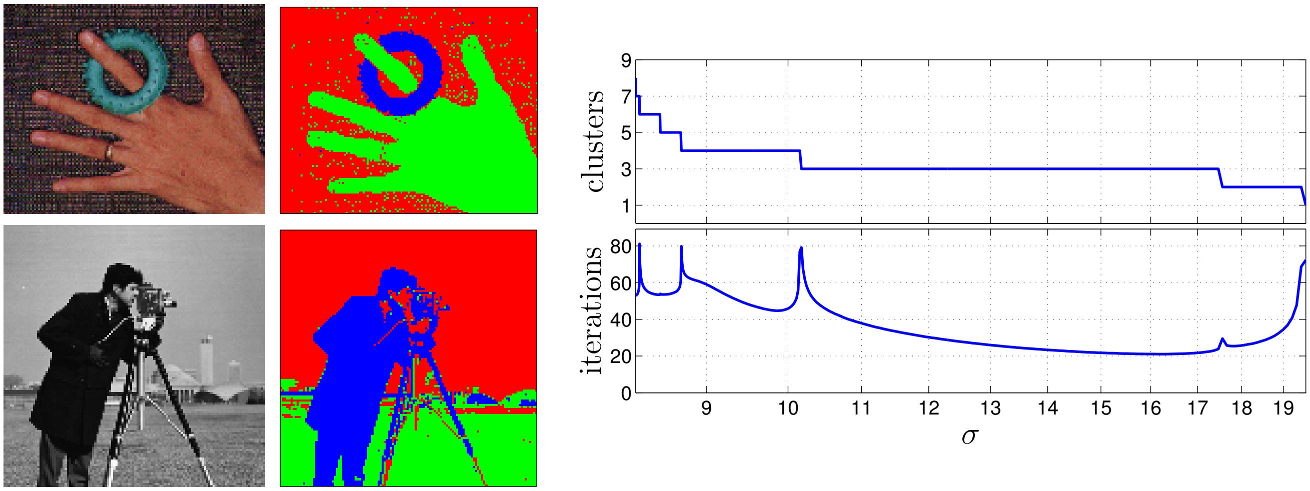


Figure 2: Segmentation results with GMS for `hand` 50 × 40.

References

[1] Keinosuke Fukunaga and Larry D. Hostetler. The estimation of the gradient of a density function, with application in pattern recognition. *IEEE Trans. Information Theory*, IT–21(1):32–40, January 1975.

[2] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(8):790–799, August 1995.

[3] Miguel Á. Carreira-Perpiñán. Mode-finding for mixtures of Gaussian distributions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1318–1323, November 2000.

[4] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[5] Miguel Á. Carreira-Perpiñán. Acceleration strategies for Gaussian mean-shift image segmentation. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *Proc. of the 2006 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'06)*, pages 1160–1167, New York, NY, June 17–22 2006.

[6] Miguel Á. Carreira-Perpiñán. Fast nonparametric clustering with Gaussian blurring mean-shift. In William W. Cohen and Andrew Moore, editors, *Proc. of the 23rd Int. Conf. Machine Learning (ICML–06)*, pages 153–160, Pittsburgh, PA, June 25–29 2006.

[7] Miguel Á. Carreira-Perpiñán. Gaussian mean shift is an EM algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*. To appear.

CH. 15: SEGMENTATION BY FITTING A MODEL

15.1 THE HOUGH TRANSFORM

Edge map for an image: FIG. 8.14

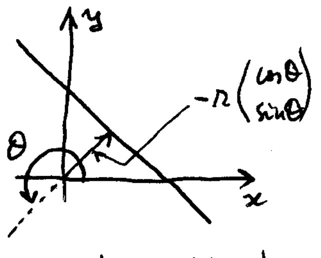
Problems in line fitting:

- Given the points that belong to the line, what is the line? Easy.
- Which points belong to which line? } Hard
- How many lines are there? }

Idea: have each point vote for each line that could pass through it, then look for lines with many votes (also applicable to fitting circles or other structures).

Parameterise a line as (θ, r): $x\cos\theta + y\sin\theta + r = 0$

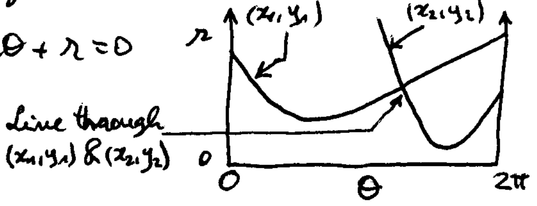
[Q: how to turn $ax + by + c = 0$ into this form? ↗]



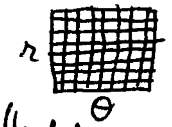
$\theta \in [0, 2\pi)$ angle, $r \geq 0$ distance to origin

Thus the line space is a half-infinite cylinder.

Lines that pass through (x,y) ≡ curve $x\cos\theta + y\sin\theta + r = 0$ in line space.



Practically, we need to discretise the line space $[0,2\pi] \times [0,R]$ (with large enough R), which introduces problems:



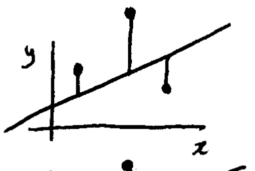
- Quantisation errors: difficult to pick a good grid size { too coarse: to many lines/bucket; too fine: no bucket has a large vote
  FIG. 15.1
- Noise: phantom lines created by chance collinearities, particularly in textured regions, which mask the real line. FIG. 15.2 – 4

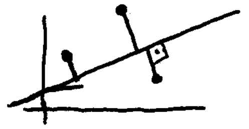
For best results, minimise the number of irrelevant features by tuning the edge detector to smooth out texture, setting the illumination to produce high-contrast edges, etc.

{ Which points belong to which line? Tag the votes.
{ How many lines are there? Count the peaks in the array.

15.2. FITTING LINES

- Important because images often contain straight lines: buildings, machines...
- If all the points that belong to a particular line $y = ax + b$ are known:
  - Least-squares: minimise errors in y (vertical distance):
    $$\arg\min_{a,b} \sum_i \|y_i - (ax_i + b)\|^2 \Rightarrow \begin{pmatrix}\overline{x^2} & \overline{x}\\ \overline{x} & 1\end{pmatrix}\begin{pmatrix}a\\ b\end{pmatrix} = \begin{pmatrix}\overline{y^2}\\ \overline{y}\end{pmatrix}$$
  - Total least-squares: minimise point-line (orthogonal) distance: symmetric wrt x, y; better fit than LSQ.





Writing the line as $ax+by+c=0$ with $a^2+b^2=1$, the distance $\binom{x}{y}$-to-line is $|ax+by+c|$ [prove it] $\Rightarrow \arg\min_{a,b} \sum_i (ax_i+by_i+c)^2$ s.t. $a^2+b^2=1 \Rightarrow \begin{cases}\text{Lagrange mult.}\\ \lambda\in\mathbb{R}\end{cases}$
$\begin{pmatrix}\overline{x^2} & \overline{xy} & \bar{x}\\ \overline{xy} & \overline{y^2} & \bar{y}\\ \bar{x} & \bar{y} & 1\end{pmatrix}\begin{pmatrix}a\\b\\c\end{pmatrix} = \lambda\begin{pmatrix}a\\b\\0\end{pmatrix}$. From the last row we get $c=-(a\bar{x}+b\bar{y})$ [$\Rightarrow$ centre of mass $\in$ line] and substituting it back we get $\overset{\to\text{ covariance matrix}}{\Sigma}\cdot\binom{a}{b}=\lambda\binom{a}{b}$, an eigenvalue problem whose solution is the largest eigenvector (principal component). In general, the k principal components give the best total LSQ fit using a k-dim subspace.

- Which point is on which line? Combinatorial search space.

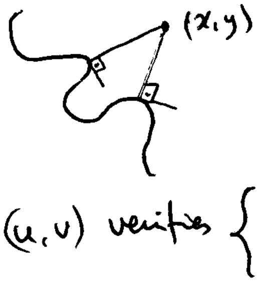
  <u>k-means for lines</u>: assume k lines and minimise $\sum_{\text{lines } l_i}\sum_{x_j\in l_i} \text{dist}(l_i,x_j)^2$ over correspondences and lines by iterating:
  - allocate each point to its closest line
  - update lines by fitting to their points

  Initialise with random lines, or random assignments; stop when the lines change little, or when no labels have changed, or when the objective function changes little.

15.3 FITTING CURVES

Same objective (minimise sum of squared point-to-curve distances) but more difficult, so use approximations. Assume differentiable curves.

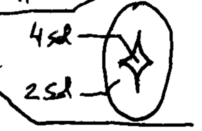
* <u>Implicit curves</u>: coordinates satisfy an eq. $\phi(x,y)=0$ (= zero level of surface ϕ), typically a polynomial eq. = algebraic curves: degree 2 are the conic sections (line, circle, ellipse, hyperbola, parabola; $\boxed{\text{TAB. 15.1}}$), higher-order polynomials rarely used because it is difficult to get a stable fit.



Distance from point (x,y) to curve $\phi(x,y)=0$ $\equiv$ distance from (x,y) to closest point (u,v) in the curve. The vector $\overrightarrow{(x,y)(u,v)}$ is orthogonal to the curve and the curve normal at (u,v) is $\left(\frac{\partial\phi}{\partial x},\frac{\partial\phi}{\partial y}\right)\Big|_{(u,v)} \Rightarrow$

(u,v) verifies $\left\{\begin{array}{ll} (u,v)\in\text{curve}: & \phi(u,v)=0 \\ \text{tangent at } (u,v) \perp \overrightarrow{(x,y)(u,v)}: & ((x,y)-(u,v))\cdot\left(-\frac{\partial\phi}{\partial y},\frac{\partial\phi}{\partial x}\right)\Big|_{(u,v)}=0\end{array}\right\}$ 2 eqs. for 2 unknowns u,v

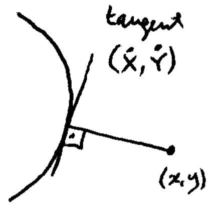
Solvable in principle, but many solutions (including maxima, saddles). Ex. ellipse



In general, for a polynomial of order d, we have two eqs. (each a poly. of order d) with up to d^2 solutions. Thus it is practically better to use approximations such as the <u>algebraic distance</u> $\phi(x,y)$, which is 0 iff $(x,y)\in$ curve, generalises to higher dimension (eg. implicit surfaces) and yields an easier numerical problem.

Since $\mu\phi(x,y)=0$ represents the same curve ($\mu\neq 0$), the algebraic distance should be normalised; $\frac{\phi(x,y)}{\|\nabla\phi(x,y)\|}$ is useful, and exact for straight lines. Both approximations behave as expected for points near the curve (since, by Taylor's th., $\|\phi\|$ increases as we go away from the curve) but farther away they may not.

* Parametric curves: coordinate are functions of a parameter whose variation trace the curve: $(X(t),Y(t))$, $t\in[t_{min},t_{max}]$ TAB. 15.2. Often $X(t)$, $Y(t)$ are polynomials (easier root-finding). The closest point in the curve to a given point (x,y) is either at the curve ends ($t=t_{min}$ or $t=t_{max}$), or along a normal to the curve: $\left((x,y)-(X(t),Y(t))\right)\cdot\left(\frac{\partial X}{\partial t},\frac{\partial Y}{\partial t}\right)=0$ (1 eq. for 1 unknown t). Easier than implicit curves but still many solutions. Second difficulty: reparameterisation.



15.4 FITTING AS A PROBABILISTIC INFERENCE PROBLEM

15.5 ROBUSTNESS

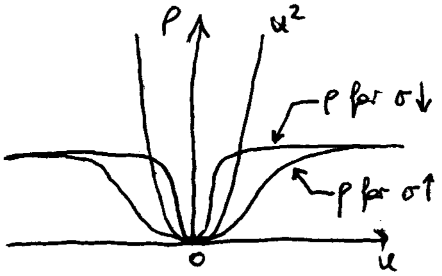
(Total) least squares places a large weight on large errors and is thus sensitive to outliers (often caused by measurement errors or correspondence errors), which can severely bias the estimated line. FIG. 15.7

Approaches:

- modify the model to use heavy tails for the noise (M-estimators).
- introduce an explicit outlier model (ch. 16.2.4).
- search for points that seem to be good (RANSAC).

* M-estimators

Optimisation problem: $E(\theta)=\sum_{i=1}^{N}\rho(\underbrace{r_i(\overset{\text{data point}}{x_i};\overset{\text{model parameters}}{\theta})}_{\text{residual}};\sigma)$



$\rho(u;\sigma)$ = heavy-tailed function, eg. $\frac{u^2}{u^2+\sigma^2}$ (many exist); saturates for large residuals, ie, does not penalise outliers arbitrarily strongly.

Scale σ: $\begin{cases}\sigma\uparrow: & \rho\simeq u^2/\sigma^2 \quad \text{similar to LSQ}\\ \sigma\downarrow: & \rho\simeq 1 \quad \text{ignores data}\end{cases}$

Gradient of objective function: $\frac{\partial E}{\partial\theta}=\sum_{i=1}^{N}\rho'(r_i(x_i;\theta))\frac{\partial}{\partial\theta}r_i(x_i;\theta)$; $\rho'(u;\sigma)$ = influence function

One problem is that the objective function is more nonlinear, so it is harder to optimise (iteratively) and has more local optima, thus being more heavily dependent on the initial parameter values. To get a good initial value, use the LSQ solution obtained for a subsample and try different subsamples (to see if we hit on one with few outliers). Also, update scale during iterations (see rule in book). $\boxed{\text{FIG. 15.9 - 15.10}}$

* RANSAC (RANdom SAmple CONSENSUS)

Idea: search the data set for good points (= not outliers), i.e. points producing models that match many of the remaining points. For example, to fit lines:

- choose sample ($\underline{n}$ points drawn uniformly at random)
- fit line to it
- count good points (at distance $\leq \underline{t}$ from line)
- if $\geq \underline{d}$ good points, refit line using all good points

} repeat $\underline{k}$ times and choose the best fit of them

User parameters:

- Number of samples n = minimum number of points needed to fit (2 for a line, 3 for a circle...).
- Number of iterations k: if $w \in (0,1)$ is one estimate of the fraction of good points in the dataset, then the probability that an n-sample is good is (approximately, assuming sampling with replacement) $p = w^n$. Thus the number of draws K needed to get one good sample is distributed as a geometric distribution of parameter p:

$$P[K=k] = (1-p)^{k-1}p \text{ which has } \begin{cases} \text{mean } E[K] = \frac{1}{p} & \left[\text{Pf. use } E[k] = \sum_{k=1}^{\infty} kP(k) = -p\frac{\partial}{\partial p}\sum_{k=1}^{\infty}(1-p)^k \text{ and } \sum_{k=0}^{\infty} r^k = \frac{1}{1-r}\right] \\ \text{standard deviation } \sqrt{E[(K-E[K])^2]} = \frac{1}{p}\sqrt{1-p} & \left[\text{Pf. use } \partial^2/\partial p^2\right] \end{cases}$$

Then take k = mean + a few standard deviations to be on the safe side.
Or, take k such that $(1-p)^k$ is small enough.
(May also estimate w from the sequence of fitting attempts).

- Telling whether a point is close (t): trial and error.
- Number of points that must agree d: choose d so that $(1-w)^d$ is small.

RANSAC is very popular in computer vision and applied to many different problems, eg. to fit fundamental matrices.

CH. 16: SEGMENTATION AND FITTING USING PROBABILISTIC METHODS

- Ch. 15: methods using local models of similarity (= comparing pairs of pixels), non-probabilistic.
- Ch. 16: global ~~models~~ models based on probability that explain a large dataset $\{x_i\}_{i=1}^N$ in terms of a small number of parameters.

16.1 MISSING DATA PROBLEMS, FITTING AND SEGMENTATION

Assume there is an underlying complete data set consisting of $\{(x_i, m_i)\}_{i=1}^N$ where x_i is the feature vector and m_i the source or component it comes from:

- Segmenting colour images into regions: sources of colour (perhaps objects) that generate the pixels.
- Segmenting tokens into collinear groups: sources = lines
- Segmenting a motion sequence into moving regions: sources of motion (objects) that gen. the pixels.

If we observed the source of each x_i (eg. if we knew which point belonged to which line), this would be a fitting problem (easy).

* Ex. 16.1: image segmentation: probabilistic mixture model for a pixel feature $x \in \mathbb{R}^D$ assuming M components, with parameters Θ:

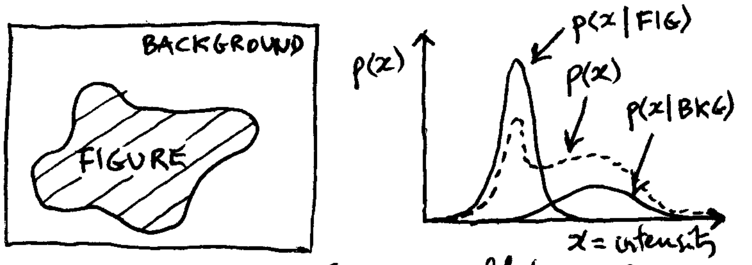
$$p(x;\Theta) = \sum_{m=1}^{M} p_m(x|m)p(m) = \sum_{m=1}^{M} p_m(x;\theta_m)\pi_m \quad \text{where}$$

- $p(m) = \pi_m \in (0,1)$, with $\sum_{m=1}^{M} \pi_m = 1$: probability a priori of component m (mixing weights)
- $p(x|m) = p_m(x;\theta_m)$: probability model for x from component m, with parameters θ_m; eg. $p_m(x;\theta_m) = |2\pi\Sigma_m|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-\mu_m)^T\Sigma_m^{-1}(x-\mu_m)}$ (Gaussian with mean μ_m, covariance Σ_m).

Both $p(x)$ and $p_m(x;\theta_m)$ for each m are probability density functions (pdf).

$\{\pi_1, \dots, \pi_M\}$ is a probability mass function (pmf).

Total parameters for $p(x;\Theta)$: $\Theta = \{(\pi_1,\theta_1), \dots, (\pi_M,\theta_M)\} = \{(\pi_m, \mu_m, \Sigma_m)\}_{m=1}^M$.



Gaussian blobs in feature space

To generate a pixel feature vector:

1. Choose (draw) a component $\{1, \dots, M\}$ from $\{\pi_m\}$.
2. Choose an x from $p_m(x;\theta_m)$.

For segmentation, we have $\{x_i\}_{i=1}^N$ but not $\{m_i\}_{i=1}^N$ and want to:

1. Determine $\{\theta_m\}_{m=1}^M$ and $\{\pi_m\}_{m=1}^M$ (to fit the model).
2. Determine m_i from $p_m(m|x_i)$ using Bayes' th. (to segment the image).

Objective function: maximum likelihood $\prod_{i=1}^{N} p(x_i;\Theta)$ or equivalently maximum log-likelihood $L(\Theta) = \sum_{i=1}^{N} \log p(x_i;\Theta)$ (usually simpler):

- Assumes points $\{x_i\}_{i=1}^{N}$ drawn iid (independently identically distributed); unrealistic because neighbouring pixels are not independent, but simplifies the estimation.
- Many nonlinear optimisation methods may be used but a particularly suitable one for mixture models is the EM algorithm.
- If we knew $\{m_i\}_{i=1}^{N}$ we could fit $p_m(x;\theta_m)$ to its features $\{x_i\}$ very easily (fitting a Gaussian).

[Q: assuming the true parameters Θ that correspond to a given image, does sampling N pixels from $p(x;\Theta)$ produce a realistic image?]

*Ex. 16.2: fitting lines to point sets: assume M lines in the plane, each with parameters θ_m (slope, intercept) and take as pdf for a token w a mixture $p(w) = \sum_{m=1}^{M} \pi_m p_m(w|\theta_m)$ with total parameters $\Theta = \{\pi_m, \theta_m\}_{m=1}^{M}$. Given observed tokens $\{w_i\}_{i=1}^{N}$, maximise the likelihood $\prod_{i=1}^{N} p(w_i|\Theta)$ over Θ. Again, if we knew which token was from what line, we could fit each line separately.

Mixture modelling and the EM algorithm

Generally, in mixture modelling the component distributions $p(x|m)$ are simple distributions with parameters θ_m (eg Gaussian with mean μ_m, covariance Σ_m), while the mixture $p(x) = \sum_{m=1}^{M} \pi_m p(x|m)$ (with parameters $\Theta = \{\pi_m, \theta_m\}$) is a more complex but more powerful model. Fitting a single component separately is generally easy, eg for a Gaussian the maximum likelihood estimate is the sample mean and covariance [prove it] — it doesn't require an iterative optimisation algorithm. Maximising the log-likelihood can be done by nonlinear optimisation (eg. gradient descent, Newton's method...) but the optimisation algorithm is often complicated (eg it needs to respect constraints: $\pi_m \in (0,1)$, $\sum \pi_m = 1$, covariance matrices are positive definite...). However, for mixture models (and more generally for missing data problems) there exists a general framework to derive an iterative optimisation algorithm, the EM (expectation-maximisation) algorithm, which usually is very simple to implement (though perhaps complicated to derive for a given problem), converges globally (from any initial parameter values) and respects the constraints (usually).

Idea: given an initial estimate of the parameters Θ:

E step: estimate the (distribution of the) missing data

M step: fit by maximum likelihood the parameters Θ assuming the missing data estimate

} Iterate till little change of Θ.

EM is a "soft" version of k-means where the correspondences between components $m=1,\dots,M$ and data points $x_1,\dots,x_N$ are not binary but continuous-valued: $p(m|x_i) \in (0,1)$. Thus, in the M step each component m fits its parameters to all the data but each data point contribute proportionally to its weight $p(m|x_n)$.

EM for mixture models: assume a complete-dataset $\{(x_i, m_i)\}_{i=1}^N$ where $m_i \in \{1,\dots,M\}$ is the index of the component where x_i came from; we only observe $\{x_i\}_{i=1}^N$ and want to estimate the mixture parameters $\Theta = \{\pi_m, \theta_m\}_{m=1}^M$ by maximising the log-likelihood

$$L(\Theta) = \sum_{i=1}^N \log p(x_i|\Theta). = \sum_{i=1}^N \log \sum_{m=1}^M \pi_m p(x_i|m;\theta_m).$$

E step: assuming the current value for The parameters Θ^{old}, compute the distribution of the missing data $p(m_i|x_i;\Theta^{old})$:

$$p(m_i|x_i;\Theta^{old}) \overset{\text{Bayes' th.}}{=\!=} \frac{p(x_i|m_i;\Theta^{old})\,p(m_i;\Theta^{old})}{p(x_i;\Theta^{old})} = \frac{p(x_i|m_i;\Theta^{old}_{m_i})\,\pi^{old}_{m_i}}{\left\{\sum_{m=1}^M p(x_i|m;\Theta^{old}_m)\,\pi^{old}_m\right.} = W_{im_i} \in (0,1)$$

↳ denominator normalises so that $\sum_{m=1}^M W_{im} = 1$

~~M step: maximise the expected complete-data log-likelihood over the parameters~~

and write the expected complete-data log-likelihood:

$$Q(\Theta;\Theta^{old}) = \sum_{i=1}^N E_{p(m_i|x_i;\Theta^{old})}\{\log p(x_i,m_i;\Theta)\} = \sum_{i=1}^N \sum_{m_i=1}^M \overset{\text{constant}}{W_{im_i}} \cdot \overset{\text{function of }\Theta}{\log p(x_i,m_i;\Theta)}$$

M step: maximise the expected complete-data log-likelihood over the parameters:

$$\Theta^{new} = \arg\max_{\Theta} Q(\Theta;\Theta^{old})$$

It can be shown that the log-likelihood $L(\Theta)$ increases or doesn't change after each EM iteration and that (under certain conditions) the algorithm converges to a local maximum likelihood estimate Θ^*. [Pf: use Jensen's inequality: $\log(\sum_i a_i b_i) > \sum_i a_i \log b_i$ if $a_i, b_i > 0$. to show that $Q(\Theta;\Theta^{old}) - Q(\Theta^{old};\Theta^{old}) < L(\Theta) - L(\Theta^{old})$. $\Rightarrow$ the increase in log-likelihood is at least as much as the increase in Q, but Q is easier to optimise: no $\log \sum_m (\cdots)$.]

16.2 The EM Algorithm in Practice

16.2.1 Example: image segmentation with Gaussian mixtures: assume N data points $x_1,\dots,x_N$ (one feature vector $\in\mathbb{R}^D$ per pixel) and model x as a Gaussian mixture with M components:

- component m is a Gaussian with parameters μ_m, σ_m^2: $p(x|m) = (2\pi\sigma_m^2)^{-\frac{D}{2}} e^{-\frac{1}{2}\left\|\frac{x-\mu_m}{\sigma_m}\right\|^2}$.
- Gaussian mixture is $p(x;\Theta) = \sum_{m=1}^{M} \pi_m\, p(x|m;\mu_m,\sigma_m^2)$.

E step: π_m, μ_m, σ_m^2 are the old parameter values:

$$w_{im_i} = \frac{p(x_i|m_i;\mu_{m_i},\sigma_{m_i}^2)\pi_{m_i}}{\sum_{m=1}^{M} p(x_i|m;\mu_m,\sigma_m^2)\,\pi_m} = \frac{\pi_{m_i}\,(2\pi\sigma_{m_i}^2)^{-\frac{D}{2}}\, e^{-\frac{1}{2}\left\|\frac{x_i-\mu_{m_i}}{\sigma_{m_i}}\right\|^2}}{\sum_{m=1}^{M}\pi_m\,(2\pi\sigma_m^2)^{-\frac{D}{2}}\, e^{-\frac{1}{2}\left\|\frac{x_i-\mu_m}{\sigma_m}\right\|^2}}.$$

M step: since $p(x_i,m_i;\Theta) = p(m_i|\Theta)\,p(x_i|m_i;\Theta) = \pi_{m_i}(2\pi\sigma_{m_i}^2)^{-\frac{D}{2}} e^{-\frac{1}{2}\left\|\frac{x_i-\mu_{m_i}}{\sigma_{m_i}}\right\|^2}$:

$$Q(\Theta;\Theta^{old}) = \sum_{i=1}^{N}\sum_{m=1}^{M} w_{im_i}\log p(x_i,m_i;\Theta) = \sum_{i=1}^{N}\sum_{m=1}^{M} w_{im_i}\left(\log\pi_{m_i} - \frac{D}{2}\log 2\pi\sigma_{m_i}^2 - \frac{1}{2}\left\|\frac{x_i-\mu_{m_i}}{\sigma_{m_i}}\right\|^2\right).$$

The maxima satisfy $\nabla_\Theta Q = 0$; we obtain a unique, closed-form solution:

$$\pi_m^{new} = \frac{1}{N}\sum_{i=1}^{N} w_{im}$$ (← needs a Lagrange multiplier for $\sum_{m=1}^{M}\pi_m = 1$: $Q - \lambda\left(\sum_{m=1}^{M}\pi_m - 1\right)$)

$$\mu_m^{new} = \sum_{i=1}^{N} w_{im}\,x_i \Big/ \sum_{i=1}^{N} w_{im}$$

$\boxed{\text{FIG. 16.1-2}}$

$$\sigma_m^{2,new} = \sum_{i=1}^{N} w_{im}\left\|x_i-\mu_m^{new}\right\|^2 \Big/ D\sum_{i=1}^{N} w_{im}$$

Once we have fitted the model (ie, we have parameter estimates $\pi_m^*, \mu_m^*, \sigma_m^{2*}$), we can segment by binarising the assignment probabilities: pixel x_i goes to the segment m that maximises $p(m|x_i;\Theta^*)$.

16.2.2 Example: line fitting: each component models probabilistically a line as follows: the observed points (from that line) are generated by (1) choosing a point along the line and (2) perturbing it orthogonal to the line with Gaussian noise: $\begin{pmatrix}x\\y\end{pmatrix} = \begin{pmatrix}u\\v\end{pmatrix} + n\cdot\begin{pmatrix}a\\b\end{pmatrix}$ where $\begin{pmatrix}u\\v\end{pmatrix}$ is a point on the line $au+bv+c=0$, $a^2+b^2=1$ and $n\sim\mathcal{N}(0,\sigma^2)$. Thus $p(x,y;a,b,c)\propto e^{-\frac{1}{2}\left(\frac{ax+by+c}{\sigma}\right)^2}$, and we need a constraint $a^2+b^2=1$. If we have only one component then maximising the log-likelihood $-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(ax_i+by_i+c)^2$ s.t. $a^2+b^2=1$ becomes the total least squares estimate.
[Problem: $p(x,y;a,b,c)$ is an improper distribution because $\int p(x,y;a,b,c)\,dx\,dy$ diverges; but it still defines a useful likelihood] $\boxed{\text{FIG. 16.3-4}}$

16.2.3. Example: motion segmentation: motion sequences often consist of large regions that have similar motion internally. Assume the motion field comes from a mixture model (layered motion model):

- At each pixel in the image there is a motion vector connecting it to a pixel in the next image (motion field).
- Each mixture component is a parametric pdf for a motion field.
- The overall motion at each pixel is the mixture of M components.

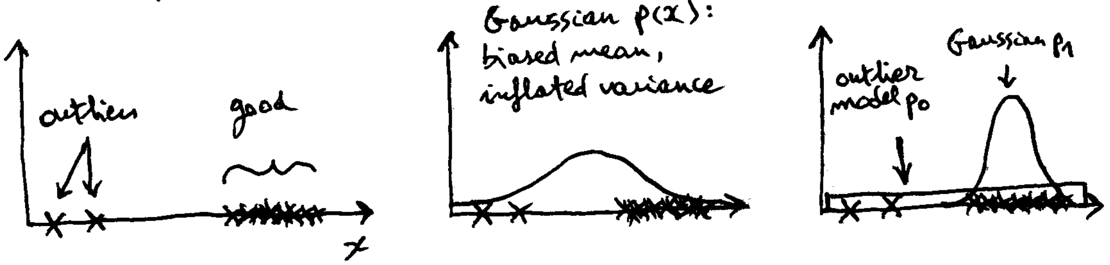
This model averages a set of M distinct, internally consistent motion fields called layers which could come from a set of rigid objects at different depths and a moving camera. FIG. 16.5

given a sequence with 2 images, we want to determine (a) which motion field a pixel belongs to, and (b) the parameter values for each field. It is a missing data problem where the missing data is the motion field to which a pixel belongs.

- Motion model: pixel $\vec{p}_1 = (u, v)$ in image 1 moves to $\vec{p}_2 = \vec{p}_1 + \vec{m}(\vec{p}_1; \theta)$ in image 2. Common choice for motion model: affine motion $\vec{m}(\vec{p}; \theta) = A\vec{p} + b$ with $\theta = \{A, b\}$.
- So the intensity at these two pixels, $I_1(\vec{p}_1)$ and $I_2(\vec{p}_2)$, is the same up to measurement noise (assumed Gaussian with standard deviation σ).
- Expected Complete-data log-likelihood: $L(V, \Theta) = -\sum_{\vec{p}, m} V_{\vec{p}, m} \cdot \frac{1}{2} \left(\frac{I_1(\vec{p}) - I_2(\vec{p} + \vec{m}(\vec{p}; \theta_m))}{\sigma} \right)^2$.

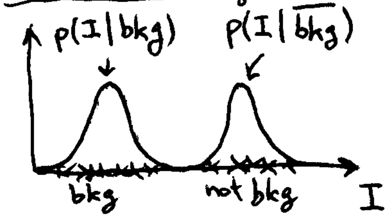
Other advantages of layered motion models: they expose motion boundaries; new sequences can be reconstructed. FIG. 16.6-7

16.2.4. Outlier model: define a mixture model where there is one component having heavy tails which is dedicated to modelling outliers. Ideally, this component catches the outliers and leaves the other components free to model the good points more accurately. (local optima are still a problem). Missing data: whether a point is an outlier or not.



Mixture $p(x) = \pi p_0(x) + (1-\pi) p_1(x)$
- π = frequency of outliers
- p_0 = outlier model, typically uniform over the range of the data. FIG. 16.8

16.2.5. Background subtraction: missing data = whether a point is background or not. Could take both models (bkg, $\overline{\text{bkg}}$) as Gaussian (with different parameters). Fit parameters with EM using pixel intensities from an image sequence. Then classify each pixel x in a given image as background if $p(\text{bkg} \mid I(x)) > p(\overline{\text{bkg}} \mid I(x))$. FIG. 14.10-11



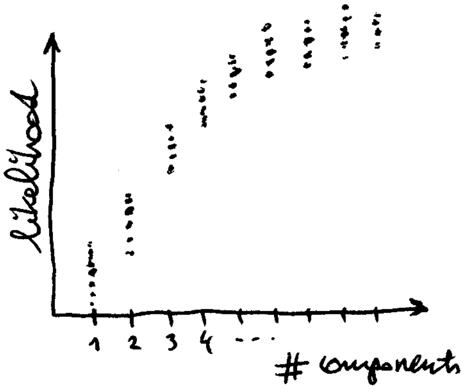
* Advantages of the EM algorithm: arises naturally in missing data problems (in particular mixtures), is usually simple to implement (eg no step sizes) and increases the likelihood monotonically, usually converging from most initial points.

* Difficulties with the EM algorithm:

- Like any other algorithm, EM converges to one local optimum of the likelihood function, of (possibly) many existing, which are usually associated with the combinatorial nature of the search space. (eg the line fitting problem). The following helps:
    - Initialize EM from several different parameter values (possibly randomly chosen), then choose the best estimate (with highest likelihood value).
    - Preprocess the data carefully, eg try to minimise noise and outliers; maybe use the Hough transform to guess good initial line fits.
    - Use a better model, eg do not model the pixels independently.
- Unlike other algorithms (eg. Newton's method, quasi-Newton), EM has typically a linear convergence rate and is thus slow (slower the "more" data are missing), though accelerated versions exist. In some applications this may not matter if we are happy with a rough parameter estimate.

16.3 MODEL SELECTION: WHICH MODEL IS THE BEST FIT?

The missing data problems we have discussed (and many other problems from previous chapters) assume the number of components is known in advance. Finding the number of components is a model selection problem: each model has a different number of components (thus a different number of parameters) and we want to find the one that fits the data best. However, we cannot use the value of the likelihood alone, since (ignoring the issue of local optima) this grows monotonically with the model size (more components ⇒ likelihood ↑, error ↓). For example, we can fit to zero error any set of points by passing a line through every pair of points. However, we prefer smaller models because:



- Such a large model is unlikely to correspond to reality and thus would generalise poorly to a different data set measured from the same problem. By fitting extremely well the particular training set that we happen to have observed, we may not fit well the data that we did not observe (selection bias).
  (overfitting)

- Smaller problems are computationally cheaper and more elegant.

One approach to model selection is to minimise a weighted average of the negative log-likelihood (the data term) and a term that increases with the number of components (model selection term). There are several such terms which are useful in practice (P = # free parameters, N = # data points):

- An information criterion (AIC): $\min \; -L(\Theta) + P$

  It ignores the number of data points and tends to overfit in practice, ie to choose a model with too many parameters.
- Bayes information criterion (BIC): $\min \; -L(\Theta) + \frac{P}{2}\log N$.
- Description length (MDL): choose the model that encodes the data set most crisply in terms of transmission: encoding cost = (cost of encoding the model parameters) + (cost of encoding the data set given the model parameters). It yields BIC.

To minimise these criteria, one needs to try different numbers of components $M = 1, 2, 3 \ldots$, fitting the parameters Θ for each M (possibly obtaining several estimates and choosing the best, to avoid bad local optima), then evaluating the criterion.

* Cross-validation: idea: split the training set into two parts, then use one to fit the model (try different numbers of components) and the other to test the fit. A model with too many parameters will fit the first data set but not the second.

- Using a single choice of split still has selection bias; we could average over all such splits but this would be unwieldy.
- Practically useful approximation: leave-one-out cross-validation: fit a model to each $(N-1)$-point subset of the training set (there are N), compute the error on the remaining points, and sum the errors to obtain an estimate of the model error. Choose the model that minimises that.

CH. 17: TRACKING WITH LINEAR DYNAMIC MODELS

Tracking in computer vision = inference about the motion of an object given a sequence of images.

Applications:

- Motion capture: track a moving person, then render a cartoon character or thousands of virtual extras in a crowd scene with realistic motion; or create slightly different motions (from the measured ones).
- Recognition from motion: identify an object or tell what it is doing from its motion.
- Surveillance, targeting...

17.1 TRACKING AS AN ABSTRACT PROBABILISTIC INFERENCE PROBLEM

Consider a sequence of frames $i = 0, 1, 2 \ldots$ At each frame, model the observation of the object as a random continuous vector y_i, and model the object as having some internal state x_i (another random continuous vector) which we do not observe and want to infer.

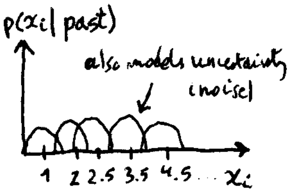
unobserved	$x_0, x_1, x_2, x_3, \ldots$
observed	$y_0, y_1, y_2, y_3, \ldots$

Example:

dynamics $x_i = x_{i-1} + 1$

observations $y_i = -2x_i$

i:	1,	2,	3,	4,	5...
x_i:	1,	2,	2.5,	3.5,	4.5
y_i:	-2,	-3.9,	-5.1,	-7,	-8



Main problems:

- Prediction: given observations $y_0, \ldots, y_{i-1}$, predict the next state x_i: needs $p(x_i | y_0, \ldots, y_{i-1})$.
- Data association: which of the measurements at frame i tell us about the state x_i? Also needs $p(x_i | y_0, \ldots, y_{i-1})$.
- Correction: after we do observe the real, relevant measurement y_i, compute $p(x_i | y_0, \ldots y_i)$.

* Independence assumptions (to simplify the model):

① Only the immediate past matters (Markov process): $p(x_i | x_0, \ldots, x_{i-1}) = p(x_i | x_{i-1})$.

② Measurements depend only on the current state (ie, y_i is conditionally independent of all other measurements given x_i): $p(y_0, \ldots, y_i | x_0, \ldots, x_i) = p(y_0 | x_0) \cdots p(y_i | x_i)$.

Inferring the state at each frame is done inductively with Bayes' th. by assuming that we know:

- The initial state distribution $p(x_0)$
- The dynamic model $p(x_i | x_{i-1})$
- The observation model $p(y_i | x_i)$.

$\Big\}$ Part of the model definition (up to the user) but with parameters that may be tuned for best performance.

Note: we propagate not the value of the state but its distribution (= entire set of possible states with their probabilities).

$$\text{Recall:} \begin{cases} M \equiv \text{marginalisation}: & p(A) = \int p(A, B)\, dB \\ C \equiv \text{conditioning}: & p(A, B) = p(A | B)\, p(B). \end{cases}$$

- Initial step: $p(x_0|y_0) \overset{\text{Bayes}}{=} \frac{p(y_0|x_0)p(x_0)}{p(y_0)} \overset{M}{=} \frac{\overbrace{p(y_0|x_0)}^{\text{obs model}}\, p(x_0)}{\int p(y_0|x_0)p(x_0)\,dx_0 \} \text{ normalises}}$.

- step i: assume we know $p(x_{i-1}|y_0,\dots,y_{i-1})$:
  - Prediction: $p(x_i|y_0,\dots,y_{i-1}) \overset{M}{=} \int p(x_i,x_{i-1}|y_0,\dots,y_{i-1})\,dx_{i-1} =$

$$\overset{C}{=} \int p(x_i|x_{i-1},y_0,\dots,y_{i-1})\, p(x_{i-1}|y_0,\dots,y_{i-1})\, dx_{i-1} \overset{\textcircled{1}}{=} \int \overbrace{p(x_i|x_{i-1})}^{\text{dyn. model}}\ \overbrace{p(x_{i-1}|y_0,\dots,y_{i-1})}^{\text{step } i-1}\, dx_{i-1}.$$

need to compute this integral

  - Correction: $p(x_i|y_0,\dots y_i) \overset{C}{=} \frac{p(x_i,y_0,\dots,y_i)}{p(y_0,\dots,y_i)} \overset{C,C}{=} \frac{p(y_i|x_i,y_0,\dots,y_{i-1})\cdot p(x_i|y_0,\dots,y_{i-1})\cdot p(y_0,\dots,y_{i-1})}{p(y_0,\dots,y_i)}$

equivalently, use Bayes th.

$$\overset{\textcircled{2}}{=} p(y_i|x_i)\,p(x_i|y_0,\dots,y_{i-1}) \underbrace{\frac{p(y_0,\dots,y_{i-1})}{p(y_0,\dots,y_i)}}_{\text{Indep. of } x_i\text{, so normalises}} \overset{\circledast}{=} \frac{\overbrace{p(y_i|x_i)}^{\text{obs. model}}\ \overbrace{p(x_i|y_0,\dots,y_{i-1})}^{\text{prediction}}}{\int p(y_i|x_i)\,p(x_i|y_0,\dots,y_{i-1})\,dx_i}.$$

need to compute this integral

$\circledast$ Can also be seen from

$$p(y_0,\dots,y_i) \overset{M}{=} \int p(x_i,y_0,\dots,y_i)\,dx_i \overset{C,C,\textcircled{2}}{=\cdots=} p(y_0,\dots,y_{i-1})\int p(y_i|x_i)\,p(x_i|y_0,\dots,y_{i-1})\,dx_i.$$

The choice of model for $p(y_i(x_i)$ and $p(x_i|x_{i-1})$ should:

- Be accurate enough for our purposes.
- Allow to compute the two integrals in the prediction and correction steps.

Simplest case: linear dynamics, linear observations, Gaussian noise ⇒ Kalman filter. Generalisations exist to nonlinear, nongaussian cases (EKF, particle filters...).

17.2 LINEAR DYNAMICAL MODELS

If using linear transformations and gaussian densities, the integrals can be solved analytically and yield gaussian densities again (with certain means and covariances). Consider $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}^m$ for $i=0,1,2\dots$:

- Dynamics: the state advances as a linear function of the previous state plus additive Gaussian noise: $x_i|x_{i-1} \sim \mathcal{N}(D_i x_{i-1}, \Sigma_{d_i})$ or equivalently $x_i = D_i x_{i-1} + n_{d_i}$ where D_i is a $d\times d$ matrix and $n_{d_i} \sim \mathcal{N}(0,\Sigma_{d_i})$ is zero-mean noise with $d\times d$ covariance matrix Σ_{d_i}. (Also possible to have a control input u: $x_i = D_i x_{i-1} + B_i u_{i-1} + n_{d_i}$.)
- Observations: the measurement is a linear function of the current state plus additive gaussian noise: $y_i|x_i \sim \mathcal{N}(M_i x_i, \Sigma_{m_i})$ or equivalently $y_i = M_i x_i + n_{m_i}$ where M_i is an $m\times d$ matrix and $n_{m_i} \sim \mathcal{N}(0,\Sigma_{m_i})$ is zero-mean noise with $m\times m$ covariance matrix Σ_{m_i}.

From a statistical learning point of view:

- Given known model parameters (D_i, etc.), the <u>inference</u> problem is to infer the state sequence $x_0, x_1, x_2 \cdots$ (or the sequence of posterior distributions for them) from the observations sequence $y_0, y_1, y_2 \cdots$ (ie, <u>tracking</u>).
- The <u>learning</u> problem (which we will not see) is to infer the model parameters from observed data. We will assume the model parameters ($D_i, \Sigma_{d_i}, M_i, \Sigma_{m_i}$ for $i = 0, 1, 2 \ldots$) are known; they may be constant, or depend on the frame i.

This seemingly simple model is very powerful. Examples for a moving object, calling p_i = position (eg in 2D), v_i = velocity $\frac{dp_i}{dt}$, a_i = acceleration $\frac{dv_i}{dt}$:

- <u>Drifting points (random walk)</u>: $x_i = p_i$, $D_i = I$ (commonly used for objects for which no better dynamic model is known). It implies we expect the state not to move too much — according to the noise covariance Σ_{d_i}.
- <u>Constant velocity</u>: $p_i = p_{i-1} + \Delta t \cdot v_{i-1}$ and $v_i = v_{i-1}$; state $x = \begin{pmatrix} p \\ v \end{pmatrix}$ with $D_i = \begin{pmatrix} I & \Delta t \cdot I \\ 0 & I \end{pmatrix}$; commonly we observe only the position: $M_i = (I \ 0)$. FIG. 17.1

  From a physics point of view, knowledge of the dynamical equations would allow us to integrate the movement and deterministically predict p at any time in the future without the need for observations — in this case, a straight line. In reality, the dynamics are an approximate model, do not include all possible factors that influence the system, and the external conditions may change, so we need indirect information about the state (the observations) to keep on track, and a way to deal with noise. Probabilistic models for tracking provide a framework to deal with uncertain data.

- ~~Note that~~ — <u>Constant acceleration</u>: $p_i = p_{i-1} + \Delta t \cdot v_{i-1}$, $v_i = v_{i-1} + \Delta t \cdot a_{i-1}$, and $a_i = a_{i-1}$; state $x = \begin{pmatrix} p \\ v \\ a \end{pmatrix}$ with $D_i = \begin{pmatrix} I & \Delta t \cdot I & 0 \\ 0 & I & \Delta t \cdot I \\ 0 & 0 & I \end{pmatrix}$; $M_i = (I \ 0 \ 0)$ (only observe position). FIG. 17.2

  Effectively, by introducing v and a in the state we are introducing information about p_{i-1}, p_{i-2}.

- <u>Periodic motion in 1D</u>: the position satisfies (say) $\frac{d^2p}{dt^2} = -p$ (harmonic oscillator) or equivalently $x = \begin{pmatrix} p \\ v \end{pmatrix}$, $\frac{dx}{dt} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} x = Sx$ [prove it]. If we integrate this 1st-order ODE with a forward Euler method with step length Δt we obtain $x_i = x_{i-1} + \Delta t \cdot \frac{dx_{i-1}}{dt} = x_{i-1} + \Delta t \cdot S \cdot x_{i-1}$ $= \begin{pmatrix} 1 & \Delta t \\ -\Delta t & 1 \end{pmatrix} x_{i-1}$.

17.3 KALMAN FILTERING

Under the linear-Gaussian assumption, the posterior distributions for prediction $p(x_i|y_0,\dots,y_{i-1})$ and correction $p(x_i|y_0,\dots,y_i)$ turn out to be Gaussian, thus all we need is to compute their means and covariances [proof: book]. The result is the Kalman filter, which updates the mean and covariance of the prediction & correction distributions at each frame:

Algorithm 17.2: Kalman filter $\boxed{\text{FIG. 17.3-4}}$

- Dynamical/Observation model: $x_i \sim \mathcal{N}(D_i x_{i-1}, \Sigma_{d_i})$, $y_i \sim \mathcal{N}(M_i x_i, \Sigma_{m_i})$
- Start assumption: user gives $\bar{x}_0^+$, Σ_0^+ for $p(x_0)$
- At each frame i, update:
  - Prediction $p(x_i|y_0,\dots,y_{i-1}) \sim \mathcal{N}(\bar{x}_i^-, \Sigma_i^-)$ with $\bar{x}_i^- = D_i \bar{x}_{i-1}^+$, $\Sigma_i^- = \Sigma_{d_i} + D_i \Sigma_{i-1}^+ D_i^T$.
  - Correction $p(x_i|y_0,\dots,y_i) \sim \mathcal{N}(\bar{x}_i^+, \Sigma_i^+)$ with $\bar{x}_i^+ = \bar{x}_i^- + K_i(y_i - M_i\bar{x}_i^-)$,
    $\Sigma_i^+ = (I - K_iM_i)\Sigma_i^-$, gain $K_i = \Sigma_i^- M_i^T (\underbrace{M_i\Sigma_i^- M_i^T + \Sigma_{m_i}}_{\text{always pd [prove it]}})^{-1}$.

Notes: (easier to understand in the 1D case where $x_i, y_i \in \mathbb{R}$)

- Innovation or residual: $y_i - M_i\bar{x}_i^-$ = measurement − prediction
- $\bar{x}_i^+ = \bar{x}_i^- + K_i(y_i - M_i\bar{x}_i^-)$ = prediction + gain × residual = weighted average of prediction and measurement.
- $\begin{cases} \overset{\to \text{ noisy data}}{\text{large}}\ \Sigma_{m_i} \Rightarrow \text{trust dynamics} \Leftarrow \text{small } \Sigma_i^- \\ \text{small } \Sigma_{m_i} \Rightarrow \text{trust measurement} \Leftarrow \text{large } \underset{\hookrightarrow \text{ uncertain prediction}}{\Sigma_i^-} \end{cases}$ − large $\underset{\hookrightarrow \text{ noisy dynamics}}{\Sigma_{d_i}}$ ⇒ trust measurement
- $\Sigma_i^- > \Sigma_{d_i}$ (prediction more uncertain than dynamics)

  $\Sigma_i^+ < \Sigma_i^-$ (correction less uncertain than prediction)
- If $\Sigma_{d_i}, \Sigma_{m_i}$ are approx. constant then $K_i, \Sigma_i^-, \Sigma_i^+$ quickly stabilise.
- K_i and Σ_i^+ can be computed before the observation y_i arrives.

* Forward-backward smoothing: while $p(x_i|y_0,\dots,y_i)$ is the best estimate of x_i up to frame i, an improved estimate of x_i for $i<N$ is obtained from $p(x_i|y_0,\dots,y_N)$, i.e., taking into account the future behaviour of x_i. For real-time tracking (eg. tracking a car in the opposite lane) we cannot wait to get future measurements (we need to predict now), but for other applications we may afford collecting a few future measurements before updating, or we may be tracking offline (eg. forensic tracking of a videotape)

Algorithm 17.3: Kalman smoothing $\boxed{\text{FIG. 17.5-6}}$

To get the distribution $p(x_i \mid y_0, \dots, y_N)$ at each frame i:

1. Forward Kalman filter (runs forward in time): $p(x_i \mid y_0, \dots, y_i) \sim \mathcal{N}(\bar{x}_i^{f+}, \Sigma_i^{f+})$
2. Backward " " (" backward " "): $p(x_i \mid y_{i+1}, \dots, y_N) \sim \mathcal{N}(\bar{x}_i^{b-}, \Sigma_i^{b-})$
3. Combine both to obtain $p(x_i \mid y_0, \dots, y_N) \sim \mathcal{N}(\bar{x}_i^*, \Sigma_i^*)$ with $\begin{cases} \Sigma_i^* = ((\Sigma_i^{f+})^{-1} + (\Sigma_i^{b-})^{-1})^{-1} \\ \bar{x}_i^* = \Sigma_i^* \left((\Sigma_i^{f+})^{-1} \bar{x}_i^{f+} + (\Sigma_i^{b-})^{-1} \bar{x}_i^{b-} \right). \end{cases}$

Σ_i^* = "harmonic mean" of fwd/bkwd covariances

$\bar{x}_i^*$ = weighted mean of fwd/bkwd means

- For backward smoothing, one problem is choosing the prior $p(x_N)$: typical vision applications are asymmetric in that we may have a good idea of where the object started ($p(x_0)$) but not of where it ends ($p(x_N)$). Using $p(x_N \mid y_0, \dots, y_N)$ (the forward filter prediction) is a dubious idea because it will lead to overconfident predictions (too narrow distributions).

17.4 DATA ASSOCIATION

What exactly is in y_i? Of all the image measurements we can take at frame i, eg edges, some are informative and some are not (called <u>clutter</u>). <u>Data association</u> = determining which measurements are informative — a difficult problem in computer vision applications. Consider a single object moving in clutter, eg. a ball moving on a fixed or slowly varying background.

* <u>choosing the nearest neighbours</u>: consider the predicted distribution of the measurements at frame i:

$$p(y_i = y \mid y_0, \dots, y_{i-1}) \overset{M.C}{=\!=} \int p(y_i = y \mid x_i, y_0, \dots, y_{i-1})\, p(x_i \mid y_0, \dots, y_{i-1})\, dx_i \overset{②}{=}$$

$$= \int p(y_i \neq y \mid x_i)\, p(x_i \mid y_0, \dots, y_{i-1})\, dx_i = \sum_{\text{states } i} \begin{pmatrix} \text{prob. given we are} \\ \text{at state } i \end{pmatrix} \times \begin{pmatrix} \text{prob. we reached} \\ \text{state } i \end{pmatrix}.$$

For Kalman filters, this is again Gaussian: $p(y_i \mid y_0, \dots, y_{i-1}) \sim \mathcal{N}(\overset{M}{D}_i \bar{x}_i^-, \overset{M}{D}_i \Sigma_i^- \overset{M}{D}_i^T + \Sigma_{m_i})$
[intuitive proof: choose $x_i \sim x_i \mid y_0, \dots, y_{i-1}$; compute $M_i x_i$, which will have mean $\overset{M}{D}_i \bar{x}_i^-$ and covariance $\overset{M}{D}_i \Sigma_i^- \overset{M}{D}_i^T$; finally, add zero-mean independent noise with covariance Σ_{m_i}.]

Now consider the following procedure to measure observations: segment the image into regions and have each region r produce a measurement y^r (eg. its mean in feature space, such as location & colour). We then choose the region with largest $p(y_i = y^r | y_0, \dots, y_{i-1})$. In other words, we choose the measurement closest to where we expect them. FIG. 17.8-9

This tends to work because:

- the background is usually significantly different from the object
- the dynamic model helps.

It fails if we happen to have a big region which looks like the object and is near locations predicted by the dynamics; this also has the danger that the Kalman filter believes it is doing well so its covariances are narrow, yet it tracks the wrong thing.

Effectively, this method uses only measurements that are consistent with the model predictions.

* Gating and probabilistic data association: instead of choosing the region most like the measurement:

1. Gating: exclude all regions which are too different from the measurement, [eg. beyond 3 standard deviations from the predicted mean (adjust this for each practical problem).
2. Probabilistic data association: use all other measurements, weighing them according to their similarity to the prediction: $\sum_r p(y_i = y^r | y_0, \dots, y_{i-1}) \cdot y^r$.

PART V: HIGH-LEVEL VISION: GEOMETRIC METHODS

CH. 18: MODEL-BASED VISION

Object recognition as a correspondence problem: what is the object pose (= position + orientation) that best matches the image?

- Objects don't scatter features all over the image: the features of an object obey some constraints because of belonging to the object (which has a certain shape and dimensions) and because cameras produce fairly orderly projections.
- Then, knowing correspondences for a few features, we can obtain correspondences for many more features.

Assume we have a collection of geometric models of the objects of interest (the modelbase).

Idea of the algorithm (hypothesize and test):

- Hypothesize a correspondence between a collection of image features and a collection of object features and determine the camera matrix that projects the objects onto the image.
  - We should generate few, good hypotheses and relatively quickly.
  - Features: image points, eg. from intersections (simplest feature: many correspondences so more search); local representation of image, eg. vector of filter outputs (complex feature: fewer correspondences so less search).
- Generate a rendering of the object (backprojection).
- Compare the rendering with the image (verification) and, if both are sufficiently similar, accept the hypothesis.
  - Needs reliable comparison methods (difficult issue).

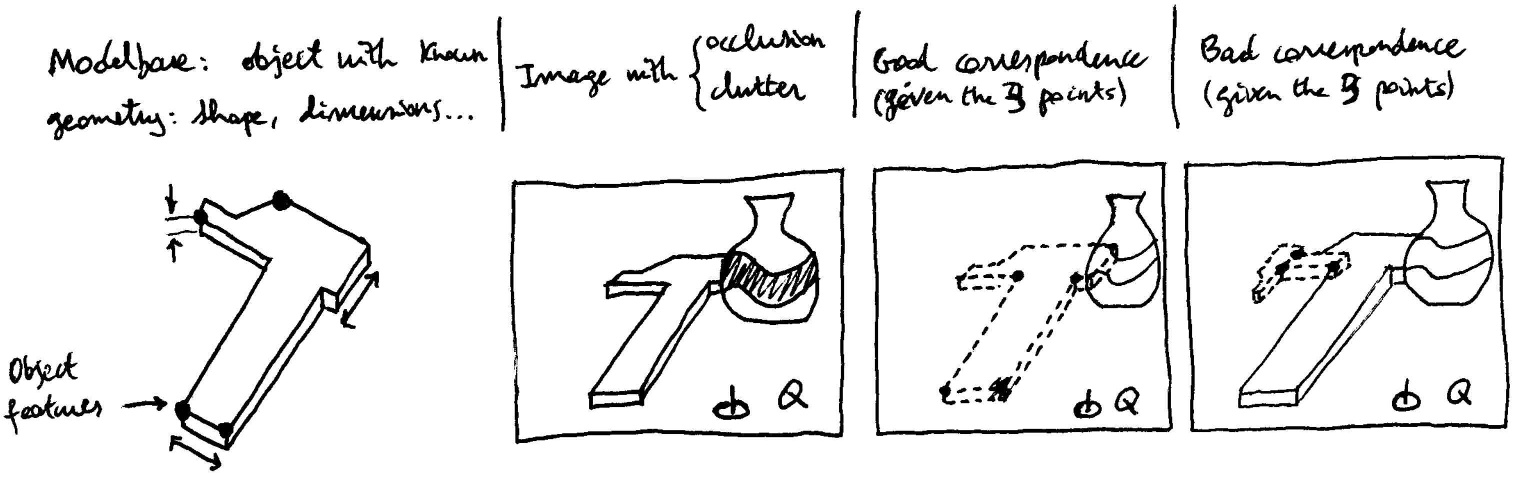

Modelbase: object with known geometry: shape, dimensions...

Image with {occlusion, clutter}

Good correspondence (given the 3 points)

Bad correspondence (given the 3 points)

- Naively, many hypotheses: if M image features, N object features in each object and L objects then LN^M possible correspondences (for each object, each image feature might correspond with any object feature).

- Geometric constraints limit the size of the search space, eg. the distance between projected points are constrained by the distance in the object and the 2D projection (camera matrix). Thus we can determine the projection parameters from a few correspondences and then using the projection model we can determine the other correspondences (pose consistency or alignment methods).

18.2 OBTAINING HYPOTHESES BY POSE CONSISTENCY

- Frame group: group of features (image or object features) that can be used to yield a camera hypothesis, eg:
  - 3 points
  - 3 directions and (to establish scale) a point
  - etc.

Algorithm 18.1: alignment FIG. 18.1 (to test for one object)

```
For all {object frame groups O, image frame groups F}
    For all correspondences between elements of F and O
        Infer camera parameters
        Render the object with the camera matrix
        If the rendering conforms to the image, the object is present
```

For affine cameras, write the projection of world point P onto image point p with camera matrix $M_{2\times4}$: $p = M\binom{P}{1}$. Thus, given a hypothetical correspondence (p, P), we get 2 linear eqs. for the 8 unknowns in M:

$$\left.\begin{aligned} p_1 &= m_{11}P_1 + m_{12}P_2 + m_{13}P_3 + m_{14} \\ p_2 &= m_{21}P_1 + m_{22}P_2 + m_{23}P_3 + m_{24} \end{aligned}\right\} \Rightarrow$$

so we need 4 points in general position to determine M (ie, a frame group of 4 points).

Once alg. 18.1 has tested for the presence of individual objects, we compare the camera matrices for each pair of recognised objects: if they are sufficiently different then the two hypotheses are incompatible.

Note:

- Many true correspondences will verify satisfactorily and yield approximately the same pose and camera matrix (eg. for different parts of the same object).
- Some false correspondences (eg from clutter) will verify satisfactorily, but yield widely different estimates of pose and camera.

So we can cluster the votes in pose space to detect the true pose — this is the Hough transform idea (so it inherits its problems re. selection of bucket size, sensitivity to noise).

18.5 VERIFICATION

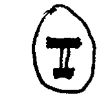
- Illumination is a big problem: we don't know how the object should look for a given pose; this makes comparing pixel values (between the image and the rendered object) unreliable.
  In some cases (eg. objects on a conveyor belt, with known illumination) this may not be a problem.
- Test: render the silhouette of the object (with the camera model) and compare it to edge points in the image (edge proximity).
  - Verification score: fraction of the length of predicted silhouette edges that lie near, and have similar orientation, to actual image edges (should remove hidden lines from the silhouette).
  - ~~should remove hidden lines~~ Unreliable because there are many sources of edges in an image (eg. textured regions), and we have no guarantee that we are using the good edges in the verification:
    + no edges where the model predicts edges ⇒ model is probably wrong.
    + edges " " " " " ⇒ model may or may not be right

    FIG. 18.5
  - In highly textured regions, most models in most poses can get high verification scores; a widely used approach is to tune the edge detector to smooth texture heavily but without blurring object edges.
- Possible to use additional sources of information, eg. certain objects are easily recognised by their distinctive texture.

PART VI: HIGH-LEVEL VISION: PROBABILISTIC AND INFERENTIAL METHODS

CH. 22: FINDING TEMPLATES USING CLASSIFIERS

Many object recognition problems involve looking for image windows that have a simple shape and stylised content (= not many details or variations from object to object):

- Frontal faces at a coarse scale ≃ oval window + ▬ for eyes, mouth + ▯ nose: 
- Camera mounted on front of car sees stop signs as having about the same shape & appearance.

We search all windows of a particular shape and size and test them for the object; if we don't know the size or orientation the object may have, we also search over scale and orientation. (template matching). Some objects can be found effectively by template matching (eg. faces, road signs) but others cannot (eg. people: too much variation in shapes).

To build the test (that tells eg. whether an oval ~~is~~ represents a face), we use classifiers trained on a large set of examples.

22.1 CLASSIFIERS

- Assume we have a training set $\{(x_n, y_n)\}_{n=1}^{N}$ of labelled examples (supervised problem): feature vector x_n and corresponding label y_n (class of object that generated x_n).
- We want a rule (a function) that maps a given x to y (ie. that classifies a feature vector). It may be a probabilistic rule: $p_s(i|x)$.
- Notation: "$i \to j$" means (a point of) class i is classified as class j;
  "$p_s(i|x)$" means our classifier strategy classifies x as class i;
  "$p(i|x)$" means the true class-conditional probabilities (unknown).
- We need to define the cost of mislabelling for each class, as this influences the choice of rule: call $L(i \to j)$ the loss function, which gives the relative cost when an object of type i is classified as type j (with $L(i \to i) = 0$).

  Risk function of a particular classifier strategy s = expected loss under s:

  $$R(s) = \sum_{i,j} p_s(i \to j)\, L(i \to j).$$

  Ex: x = patient
  label $y \in \{ill, \overline{ill}\}$
  classifier = doctor

  $L =$

$i \backslash j$	ill	$\overline{ill}$
ill	0	10
$\overline{ill}$	2	0

  $$R(s) = 10 \cdot p_s(ill \to \overline{ill}) + 2 \cdot p_s(\overline{ill} \to ill).$$

- Expected loss for a point x when choosing class i : $\sum_j \underbrace{p(j|x)\,L(j\to i)}_{\text{expected loss if } x \text{ is of class } j}$
- <u>Decision boundary</u>: separates regions of feature space x corresponding to each label.

* <u>2-class classifier that minimises risk</u> (as a function of the unknown class (prior) and class-conditional probabilities): the <u>optimal classifier</u> works as follows:

- For any point x along the decision boundary, either choice of class has the same expected loss:

$$\underbrace{p(1|x)\,L(1\to 1) + p(2|x)\,L(2\to 1)}_{\text{choosing class } j=1} = \underbrace{p(1|x)\,L(1\to 2) + p(2|x)\,L(2\to 2)}_{\text{choosing class } j=2} \Rightarrow p(2|x)\,L(2\to 1) = p(1|x)\,L(1\to 2) \Rightarrow$$

$$\left(\begin{array}{l}\text{Bayes' th.:}\\ p(j|x) = \frac{p(x|j)\,p(j)}{p(x)}\end{array}\right) \Rightarrow \quad p(x|2)\,p(2)\,L(2\to 1) = p(x|1)\,p(1)\,L(1\to 2) \Big\} \begin{array}{l}\text{characterises points } x\\ \text{on the decision boundary}\end{array}$$

- For points x off the boundary, it chooses the class with the lowest expected loss:

$$p(1|x)\,L(1\to 2) \underset{\hookrightarrow \text{ choose class 2}}{\overset{\to \text{ choose class 1}}{\gtrless}} p(2|x)\,L(2\to 1)$$

In general for multiple classes and assuming $L(i\to j) = \begin{cases} 1, & i \neq j \text{ (same loss for each outcome)} \\ 0, & i = j \\ d < 1, & \text{no decision} \end{cases}$,
the best strategy (<u>Bayes classifier</u>) is:

- If $p(k|x) > p(i|x)\ \forall k \neq i$ and $p(k|x) > 1-d \Rightarrow$ choose class k
- If ties between several such $k \Rightarrow$ choose uniformly at random among such k
- If $p(k|x) \leq 1-d \Rightarrow$ refuse to decide.

$\boxed{\text{FIG. 22.1-2}}$

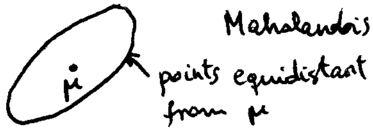
<u>Bayes risk</u>: total risk for the Bayes classifier (generally nonzero) = smallest possible risk achievable by any classifier. Unknown since we don't know the true probabilities $p(x,i)$. The risk of a good classifier should converge to the Bayes risk as the number of examples increases.

* In practice we don't know $p(x,i) = \overbrace{p(x|i)}^{\text{class-conditional probabilities}} \cdot \overbrace{p(i)}^{\text{prior prob.}}$. There are two general approaches to determine a classifier from an example data set:

1) <u>Generative approach: explicit probabilistic model</u>: learn a probabilistic model of $p(i|x)$, or $p(x|i)$, from the training set and plug them into Bayes' rule (plug-in classifier). But:
   - the model that best approximates $p(i|x)$ need not be the best classifier
   - in fact, a good classifier need not yield a good representation of the data.
   - estimating $p(i|x)$ in high dimensions is a hard problem.

2) Discriminative approach (determining decision boundaries directly): learn a model for the decision boundary directly, ignoring the probability model. This works well because the decision boundaries are what determines the performance of the classifier, not the details of the probability model away from the boundaries. A particular important case is linearly separable data: a hyperplane is enough to model the boundary. The discriminative approach solves an easier problem; however, with only a boundary there is no reliable indication of the extent to which an example belongs to one or another class.

* Ex: plug-in classifier for normal class-conditional densities (parametric): N classes, kth class contains N_k examples $\{x_{ki}\}_{i=1}^{N}$ ⇒ maximum likelihood estimate for each $p(x|k)$: $\mu_k = \frac{1}{N_k}\sum_{i=1}^{N_k} x_{ki}$, $\Sigma_k = \frac{1}{N_k}\sum_{i=1}^{N_k}(x_{ki}-\mu_k)(x_{ki}-\mu_k)^T$; estimate $p(k) = \frac{N_k}{N}$. To classify x, choose the class k that minimises $\underbrace{\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)}_{\text{Mahalanobis square distance}} - p(k) + \frac{1}{2}\log|\Sigma_k|$ [prove it]. This yields elliptical boundaries (linear if $\Sigma_k = \Sigma\ \forall k$).



* Ex: nearest-neighbours classifier (nonparametric): (k,l)-nearest-neighbours classifier: to classify x, find the k nearest neighbours of x in the training set and classify x with the class that has the highest number of votes n if $n > l$, else refuse to classify. $(k,0)$ = k-nearest-neighbours classifier, $(1,0)$ = nearest-neighbour classifier.

They work well despite their simplicity: if the number of examples is large, the risk is not far from the Bayes risk. Computationally, for a new x we need to find the k nearest neighbours, which is costly in high dimensions: $\mathcal{O}(DN)$ (N examples, D dim.) naively (checking the distance to every example) and not much faster with more sophisticated algorithms. We also require a definition of distance.

* Estimating and improving performance:

- Overfitting, crossvalidation
- Bootstrapping (don't confuse with the bootstrap): often a small number of training cases are the really important ones that determine the classifier behaviour, with the rest having only a small influence. To ensure we have those cases we need a large enough training set which means a higher computational cost. A classifier may be obtained at less cost if, instead of training with the whole data, we first train with a subset of it, then add the false positives and false negatives to the subset; we retrain again, etc.

22.2 BUILDING CLASSIFIERS FROM HISTOGRAMS

A histogram (normalised by the number of pixels to obtain relative frequencies) can be used to estimate nonparametrically the class-conditional density if the space dimension is low (so the number of bins required isn't too large).

* Ex: finding skin pixels using a classifier: this is important for eg. gesture-based interfaces. Skin has a characteristic colour distribution, which allows to build a skin finder by classifying pixels by their colour. Take x as RGB colour and model the class-conditional densities $p(x|\text{skin})$ and $p(x|\overline{\text{skin}})$ with histograms; estimate $p(\text{skin}) = 1 - p(\overline{\text{skin}})$ as the proportion of skin pixels on a (large) dataset. The Bayes classifier then yields: $\boxed{\text{FIG. 22.3}}$

$$p(\text{skin}|x)\cdot L(\text{skin}\to\overline{\text{skin}}) \gtrless p(\overline{\text{skin}}|x)\cdot L(\overline{\text{skin}}\to\text{skin}) \Leftrightarrow p(\text{skin}|x) \underset{\overline{\text{skin}}}{\overset{\text{skin}}{\gtrless}} \theta = \frac{L(\overline{\text{skin}}\to\text{skin})}{L(\text{skin}\to\overline{\text{skin}})+L(\overline{\text{skin}}\to\text{skin})} = \begin{array}{c}\text{relative}\\ \text{loss}\end{array}$$

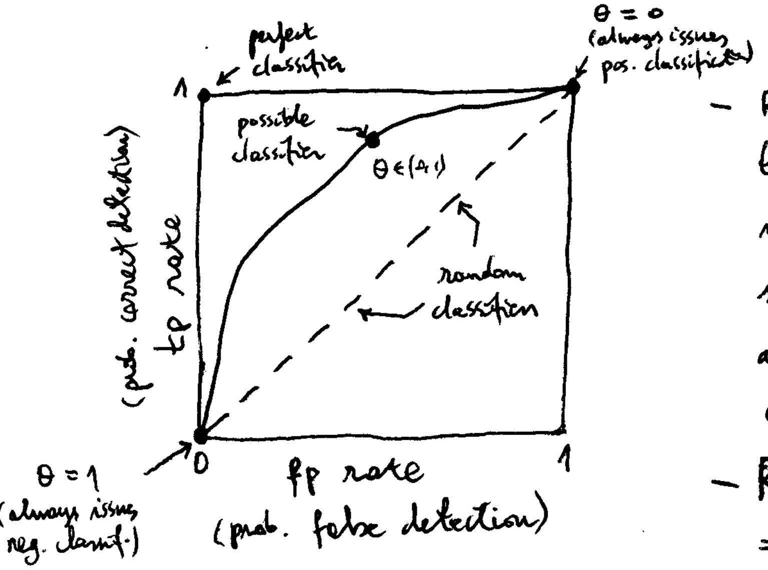
Varying the loss function L is equivalent to varying $\theta \in [0,1]$; this yields a family of classifiers, one for each θ. Each of these classifiers has a different false-positive and false-negative rate and can be conveniently represented by the receiver operating characteristic (ROC) curve: $\boxed{\text{FIG. 22.4}}$ (see Fawcett: "An intro to ROC analysis", 2006).

Confusion matrix (contingency table):

		true class	
		skin	$\overline{\text{skin}}$
hypothesized class	skin	tp	fp
	$\overline{\text{skin}}$	fn	tn
column totals		P	N

Common metrics $\in [0,1]$:

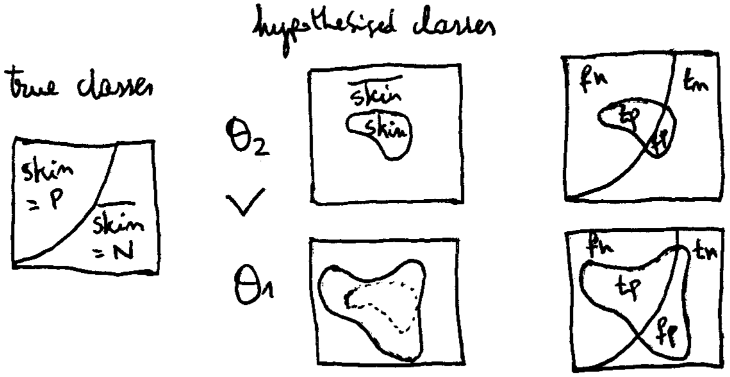
- false-positive rate $= p(\text{skin}|\overline{\text{skin}}) = \frac{fp}{N} = 1 -$ specificity
- true-positive (hit) rate $= p(\text{skin}|\text{skin}) = \frac{tp}{P} =$ recall, sensitivity
- false-negative (miss) rate $= p(\overline{\text{skin}}|\text{skin})$.
- precision $= \frac{tp}{tp+fp}$, accuracy $= \frac{tp+tn}{P+N}$.



- ROC curves have long been used in signal detection theory and medical decision making; its use in machine learning has increased in recent years since they offer a more flexible way to evaluate a classifier's performance than just using the classification accuracy.
- Random classifier: $p(\text{skin}|x)$ = random value in $[0,1]$ $\Rightarrow p(\text{skin}|\overline{\text{skin}}) = p(\text{skin}|\text{skin}) = \theta$ (diagonal line).

Properties of ROC curves:

- Nondecreasing:



If $\theta_2 > \theta_1$, we issue more positives (all issued with θ_1, plus some more):

$(\text{tp rate})_1 \geq (\text{tp rate})_2$

$(\text{fp rate})_1 \geq (\text{fp rate})_2$

- Always above the diagonal (= random classifiers) because, for a classifier at (fp, tp) below the diagonal, negating its decision gives $(1-fp, 1-tp)$ above the diagonal.
  [Pf. Its true positives become false negatives and its false positives become true negatives]

- A discrete classifier (one which outputs a class label rather than a posterior probability) appears as a single point at its rates (fp, tp).

- Conservative classifiers issue positives only with strong evidence: near (0,0)
  Liberal " " " with weak evidence: near (1,1).

- Invariant to choice of prior: changing p(skin) is equivalent to changing θ. [Pf: the rates of fp, tp do not depend on the ratio of P to N.]
  Accuracy, precision do depend on the prior. Precision-recall curves are often used in information retrieval (retrieval = classification) instead of ROC curves.

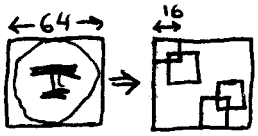
- A different type of classifier (eg. a neural net) will yield a different ROC curve.
  One criterion to compare ROC curves is by the area under the ROC curve (the larger the better).

- Difficult to extend to >2 classes.

* Ex: face finding assuming independent template responses (naive Bayes): in high dimensions, a histogram has too many bins; however, if (depending on the problem) we can afford to assume independence of the dimensions for the class-conditional probabilities, then $p(x_1, \dots, x_D \mid \text{class } i) = p(x_1 \mid \text{class } i) \cdots p(x_D \mid \text{class } i)$ which can be built with D univariate histograms.

- Assume the face occurs at a fixed, known scale (we could search smoothed and resampled versions of the image to find larger faces) and occupies a region of known shape (oval or square for frontal faces, polygon for lateral faces). Say we take a 64×64 square.

- Split this into n overlapping regions $r_1, \dots, r_n$ (eg 4 16×16); quantise the space of each region into k levels (eg with k-means).



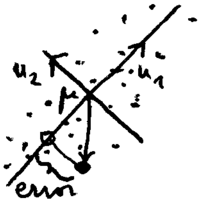
- Modelling the 64×64 image probabilistically for faces yields $p(\text{img} \mid \text{face}) = p(r_1, \dots, r_n \mid \text{face})$, which requires an n-dim histogram with k^n bins (computationally infeasible). Naive Bayes: model instead as $p(\text{img} \mid \text{face}) = p_1(r_1 \mid \text{face}) \cdots p_n(r_n \mid \text{face})$, which requires n 1-D histograms, or one 2D histogram for (level $\in \{1, \dots, k\}$, location $\in \{1, \dots, n\}$), with nk bins. Same for $p(\text{img} \mid \overline{\text{face}})$.
- The approximations (quantisation, naive Bayes) yield a poor generative model 4 faces, but work well for recognising faces. The quantisation also helps to remove minor variations caused by noise, skin irregularities, etc. FIG. 22.5

22.3 FEATURE SELECTION

- We want to find a set of features that best classifies a dataset.
- Directly using all the pixel values with a classifier is not a good idea: the feature space is very complicated and we'd not have large enough datasets to learn a good classifier.
- Also, it is better to introduce a priori knowledge such as illumination or scaling, rather than providing (a huge number of) examples that represent differences in illumination or scaling.
- Assume we have an initial feature set (eg the pixel intensities) and that we want to obtain a subset of features that are linear combinations of the initial ones: calling $x_1, \dots, x_n \in \mathbb{R}^d$ (d features) the training set, we want a new training set with $l \ll d$ features $y_i = A x_i + b$ ($A_{l \times d}$, $b_{l \times 1}$).

* Principal component analysis (PCA) selects features that maximise the variance of the data. Call $\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$ and $\Sigma = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T$ the mean and covariance of the training set (note: Σ is a biased estimator while the sample covariance $\frac{n}{n-1} \Sigma$ is unbiased). Define a new scalar zero-mean feature $u(x) = u_1^T (x - \mu)$ with $u_1 \in \mathbb{R}^d$. Then $\text{Var}(u) = \frac{1}{n} \sum_{i=1}^{n} u(x_i)^2 = u^T \Sigma u$ which is maximised for fixed $\|u_1\|$ by the first eigenvector (associated with the largest eigenvalue) of Σ. If we require a 2^{nd} zero-mean scalar feature that maximises the variance but being orthogonal to v_1 then we obtain the 2^{nd} eigen-

vector, etc. We can obtain the result in general from $\max_{U_{d\times l}} \text{tr}(U^T\Sigma U)$ s.t. $U^TU = I$, which is given by the leading l eigenvectors of Σ, $U = (u_1, \dots, u_l)$, for which $\text{tr}(U^T\Sigma U) = \sum_{i=1}^{l} \lambda_i$ (leading eigenvalues of Σ).



- Equivalently, the PCA features minimise the squared reconstruction error of the training set when projected to an l-dim. linear subspace: assume zero-mean $X_{d\times n} = (x_1, \dots, x_n) \Rightarrow \min_{\substack{U^TU=I \\ U_{d\times l}}} \|X - UU^TX\|_F^2 \overset{\|A\|_F^2 = \text{tr}\, AA^T}{=\!=\!=\!=} \max_{\substack{U^TU \\ U_{d\times l}}} \text{tr}\, U^T\Sigma U.$

  Pf. assume $\text{rank}(X) = d < n$ so $\text{rank}(UU^TX) = l$, and write the SVD of X as $X = ASB^T$ with orthogonal $A_{d\times d}$, $B_{n\times d}$, $S = \text{diag}(s_1, \dots, s_d)$ containing the singular values $s_1 \geq \dots \geq s_d > 0$.

  $\Rightarrow \min_{\text{rank}(\tilde{X}) = l} \|X - \tilde{X}\|$ is achieved by $\tilde{X} = UU^TX = A_l S_l B_l^T$ (the leading l singular values)

  $\Rightarrow U = A_l$ (note $\lambda_i = \frac{1}{n} s_i^2$ since $\Sigma = \frac{1}{n} XX^T$).
- PCA eliminates linearly redundant features (those which are l.c. of other features); these appear associated with null eigenvalues of Σ.
- The PCA features are uncorrelated. [Pf. Their covariance, $U^T\Sigma U$, is diagonal.]
- Projecting on the eigenvector basis $u_1, \dots, u_l$ gives the set of l linear features that (1) preserves the most variance and (2) minimises the reconstruction error.
- Depending on the data set, PCA may or may not give a good representation of the data. FIG. 22.6, 7, 9

* Identifying individuals with PCA:

- Face identification: given a face image, whose face is it? Two types of methods:
  - Extract facial features (nose, lips, eyes...) and match their geometry (height, width...) and relative position with descriptions stored in the database.
  - Template matching: directly compare brightness patterns. Ex: eigenfaces.
- Eigenfaces are the principal components of a data set of face images:
  - Offline, given a database of face images: (1) subtract the mean and (2) project on the top l eigenvectors (eigenfaces) of the covariance matrix. The resulting features for a given face are its coordinates in the eigenface basis: $= a_1 \cdot$ $+ \dots + a_l \cdot$.
  - Online, given a new face image, we compute its features $\vec{a}$ (by subtracting the mean and projecting on the eigenfaces) and apply a nearest-neighbour classifier:

1. If the reconstruction error $> \varepsilon_1$, classify as non-face (too far from the face space).
2. Otherwise, let $\vec{f}$ be the face in the dataset that is closest to the test image in face space (eigenfaces). Then, if the distance to $\vec{f}$ is $< \varepsilon_2$ classify the test ~~image~~ face as $\vec{f}$, else classify it as unknown face (and optionally add it to the database and recompute the eigenfaces).

- Ex: Turk & Pentland use 2500 128×128 images of 16 subjects, corresponding to all combinations of 3 face orientations, 3 head scales and 3 lighting conditions. $\boxed{\text{FIG. 22.8}}$ $\boxed{\text{TAB. 22.1}}$

* Canonical variates (Fisher discriminant analysis):

- PCA does not use class information (is unsupervised) and thus need not yield features that are good for classification. $\boxed{\text{FIG. 22.9}}$
- Call μ_j, Σ_j the mean and covariance of class j; $S_W = \sum_{j=1}^{c} \Sigma_j$ the sum of within-class covariances (assumed full-rank); μ, S_B the mean and covariance of all class means (between-class covariance). We want a set of axes that yield compact clusters for each class but separate clusters for different classes.
- specifically for a single scalar feature $u(x) = u_1^T x$, maximise the ratio of between-class variance to within-class variance:

$$\max_{u_1 \neq 0} \frac{u_1^T S_B u_1}{u_1^T S_W u_1} \overset{\text{Rayleigh quotient}}{\Longleftrightarrow} \max u_1^T S_B u_1 \text{ s.t. } u_1^T S_W u_1 = 1 \xrightarrow{\text{Lagrange multiplier}} S_B u_1 = \lambda S_W u_1,$$

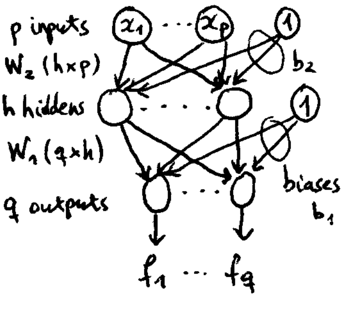
a generalised eigenvalue problem whose solution u_1 is the generalised eigenvector associated with the largest generalised eigenvalue λ. Canonical variate = u_1.

Particular case: for 2 classes, $u_1 = S_W^{-1}(\mu_1 - \mu_2)$, which makes an angle in $[-\frac{\pi}{4}, \frac{\pi}{4}]$ with the line joining the class means.

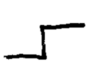
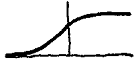
- Subsequent maximisation of the objective subject to orthogonality for ~~m~~ l features yields the generalised eigenvectors associated with the leading l generalised eigenvalues.
- Using $l < d$ canonical variates reduces the dimension to l while best preserving the separation between classes.

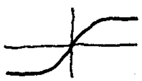
22.4 (FEEDFORWARD) NEURAL NETS

- Feedforward neural net = parametric function $f: x \in \mathbb{R}^p \to y \in \mathbb{R}^q$, eg with one fully-connected hidden layer: $f(x) = \phi(\underset{\text{weights}}{W_1} \cdot \phi(\underset{\text{weights}}{W_2} x + \overset{\text{biases}}{b_2}) + \overset{\text{biases}}{b_1})$



The nonlinearity $\phi: \mathbb{R} \to \mathbb{R}$ is applied elementwise; typical ϕ:

 threshold (hard to optimise),  logistic $\phi(x) = 1/1+e^{-x}$

 $\phi(x) = \tanh x$

- Flexible: with many hidden units, we can approximate a lot of functions.
- Trained to minimise the LSQ error on a training set $\{(x_n, y_n)\}$ (supervised problem, regression): $\min_{W,b} E(W,b) = \frac{1}{2} \sum_{n=1}^{N} \| y_n - f(x_n; W, b) \|^2$. Any LSQ optimisation algorithm is valid (eg. Gauss-Newton, Levenberg-Marquardt) but often one just uses gradient descent because W has many weights.
- The gradient $\frac{\partial E}{\partial W}$ can be computed layer-by-layer using the chain rule (backpropagation; see book). However, computing the exact gradient needs to sum over all examples x_n ("batch learning), which is computationally costly. Instead, we can compute the gradient for one example alone and update the parameters immediately ("online" learning, stochastic gradient descent), cycling over all examples (possibly in random order). For sufficiently small gradient steps both the batch and (on the average) the online versions decrease the error.
- Model selection for the number of units and layers: in practice, solved by trial and error.
- To produce a classifier with a neural net, we need to approximate the function $p(i|x)$, but we don't know the value of $y_n = p(\cdot|x_n)$. However, we can use $y_n = (0 \dots 1 \dots 0)$ (a 1 for the class of x_n) and then classify x_n into the largest component of $f(x)$.

* Finding faces using neural nets: the neural net acts as a classifier which is applied to windows of the image, possibly at different scales (template matching).

- Correction wrt illumination:
  - A simple illumination model is a linear ramp: one side is dark, the other bright, with a smooth transition (eg. face illuminated from the right) ⇒ fit a linear ramp to the intensity values and subtract it from the image window.
  - Can also do this to the log-intensity (the illumination is approximately additive in the log domain), but not much difference in practice.
  - Histogram-equalise the window to ensure its histogram is the same as that of a set of reference images. FIG. 22.14
- Correction wrt face orientation (RBK, 1998): (1) estimate the orientation of the window with one neural net, (2) rotate window to vertical, (3) classify this with a second neural net. FIG. 22.15-16

* Convolutional neural nets: feedfwd neural net but not fully connected; each layer acts as a set of spatially localised filters, so that the sequence of layers acts as a sophisticated feature extractor, with the last layer being a classifier. Ex: for handwritten characters, oriented bar filters are useful: given a map of orientation bars in the image (layer 1), layer 2 extracts spatial relations between bars, etc. We specify by hand the architecture (# layers, connectivity between them, # units) and train all parameters together. The net rectifies the handwritten character, achieving invariance to scaling, rotation, translation. FIG. 22.17-18 + description.

3-JAN-2006

EM ALGORITHM FOR FINITE MIXTURES

- $p(x,m) = \pi_m p(x|m)$, $p(x) = \sum_{m=1}^{M} \pi_m p(x|m)$, parameters $\Theta = \{\pi_m, \underbrace{\mu_m, \Sigma_m}_{}\}_{m=1}^{M}$
  $p(x|m) \sim \mathcal{N}(x; \mu_m, \Sigma_m)$
- Log-likelihood $L(\Theta) = \sum_n \log p(x_n) - \lambda \sum_m \pi_m = \sum_n \log \sum_m \pi_m p(x_n|m) - \underbrace{\lambda \sum_m \pi_m}_{}$
  given sample $\{x_n\}$ — Lagrange multip for $\sum_m \pi_m = 1$
- E step: complete log-likelihood $\sum_n \log p(x_n, m_n)$ as if m_n was known.

  Posterior probabilities (given old parameters) of missing variables given observed variable:

  $$h_{m_n,n} = p^{old}(m_n|x_n) = \frac{p^{old}(x_n|m_n)\,\pi_{m_n}}{\sum_{m_n'=1}^{M} p^{old}(x_n|m_n)\,\pi_{m_n}} = \text{fixed numbers (indep of } \Theta)$$

  Expected complete log-likelihood $Q(\Theta|\Theta^{old}) = \sum_n E_{p^{old}(m_n|x_n)}\{\log p(x_n, m_n)\} =$

  $$= \sum_n \sum_{m_n=1}^{M} h_{m_n,n} \log \pi_{m_n} p(x_n|m_n) - \lambda \sum_m \pi_m = \sum_m Q_m$$

- M step: Q has a unique maximum with a closed form:

  $$\frac{\partial Q}{\partial \pi_m} = \sum_n \frac{h_{mn}}{\pi_m} - \lambda = 0 \overset{\sum_m \pi_m = 1}{\Longrightarrow} \pi_m^{new} = \frac{\sum_n h_{mn}}{\sum_{n,m} h_{mn}} = \frac{1}{N} \sum_n h_{mn}$$

  $$\frac{\partial Q}{\partial \mu_m} = \sum_n h_{mn} \Sigma_m^{-1} (x_n - \mu_m) = 0 \Rightarrow \mu_m^{new} = \frac{\sum_n h_{mn} x_n}{\sum_n h_{mn}}$$

  $$\frac{\partial Q}{\partial \Sigma_m^{-1}} = \sum_n h_{mn} \frac{1}{2}\left(\Sigma_m - (x_n - \mu_m)(x_n - \mu_m)^T\right) = 0 \Rightarrow \Sigma_m^{new} = \frac{\sum_n h_{mn}(x_n - \mu_m^{new})(x_n - \mu_m^{new})^T}{\sum_n h_{mn}}$$

  $$\frac{\partial \log|A|}{\partial A} = (A^T)^{-1} \qquad \frac{\partial (x^T A x)}{\partial A} = x x^T$$

- Proof that EM increases the likelihood monotonically:

  $$Q(\Theta|\Theta^{old}) - Q(\Theta^{old}|\Theta^{old}) = \sum_n \sum_{m_n} p^{old}(m_n|x_n) \log \frac{p(m_n)\,p(x_n|m_n)}{p^{old}(m_n)\,p^{old}(x_n|m_n)} \leq \left(\begin{array}{l}\text{Jensen's inequality:} \\ \log \sum_i a_i x_i \geqslant \sum_i a_i \log x_i\end{array}\right)$$

  $$\leq \sum_n \log \sum_{m_n} p^{old}(m_n|x_n) \frac{p(m_n)\,p(x_n|m_n)}{p^{old}(m_n)\,p^{old}(x_n|m_n)} = \sum_n \log \sum_{m_n} \frac{p(m_n)\,p(x_n|m_n)}{p^{old}(x_n)} = \sum_n \log \frac{p(x_n)}{p^{old}(x_n)} =$$

  $$= L(\Theta) - L(\Theta^{old}).$$

  Thus the increase in log-likelihood is at least as much as the increase in Q, but Q is easier to optimise (no $\log \sum_m$).

  Lower bound: $L(\Theta) \geqslant Q(\Theta|\Theta^{old}) + k$ with $K = L(\Theta^{old}) - Q(\Theta^{old}|\Theta^{old})$.