Computational complexity

Computational complexity was the main motivation behind Deriche's work. Although it is not usually a significant factor when implementing algorithms on modern workstations, it is still useful:

- to have a feel for the complexity of an algorithm and how it scales,
- to understand the details when considering real-time implementations.

We will now look briefly at the complexity of a number of different types of algorithm. To keep the discussion concrete, we suppose that we want to detect, locate and track a face in an image sequence.

Assume that the face is a frontal view against a bland background as here (=> not really practical).

Although simplistic, schemes such as the above and even simpler ones have been used to drive a number of active vision systems where a pair of stereo cameras are used to track a person's face as they move about.

Algorithms

The algorithms required are:

♦ ***pre-processing***: such as spatial smoothing and, if required, sub-sampling

♦ ***detection***: for which we shall consider two methods:

  * a differentiation and thresholding scheme (any face-like object)
  * template matching (a particular face)

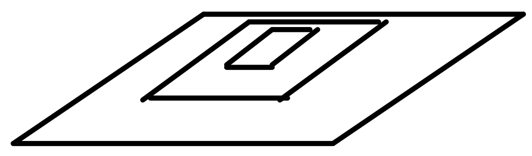
♦ ***location***: compute the centroid of the face

♦ ***tracking***: the *predict, measure update cycle* with:

  * motion estimation from temporal and spatial derivatives for the prediction
  * cross-correlation matching for the measurement

In addition, we shall also consider the use of sub-sampling and an image pyramid in order to reduce the complexity of the most computationally intensive tasks

eg: 3 level pyramid:



Pre-processing

We have already seen several different smoothing algorithms, including:

- uniform, top hat averaging
- Gaussian filters
  - both are separable

(most low-level smoothing algorithms are separable)

- both may be implemented recursively
  - efficient (see eg Deriche for Gaussian)
  - independent of size for uniform filter
  - approximate for Gaussian (see Deriche)

A pyramid, sub-sampled by a factor of 2 in each dimension, can be constructed efficiently:

- eg, if ϑ operations required on the image
$$\sim \vartheta\left(1+\frac{1}{4}+\frac{1}{16}+\ldots\right)=4\vartheta/3$$
required for the whole pyramid (all levels)
  - similar memory efficiency
  - especially suitable if Gaussian filters are used, since $G_\sigma * G_\sigma = G_{\sigma\sqrt{2}}$ etc.
- complexity (ie value of ν) and delay depend on details of the implementation: see Burt (CVGIP 1983) and Crowley (IEEE PAMI, ~1984)
- usually require video-rate throughput and processing delay ≤ 1 frame/output frame

Detection

Consider first detection of a face by differentiation and thresholding. There are several ways of carrying out the **differentiation**, eg:

- forming the gradient image $\nabla I = \left(I_x, I_y\right)$
  - => 2 outputs/input pixel
- calculating $\left|\nabla I\right|$ => 1 output/input pixel
  - may use a Euclidean or Manhattan metric

To detect a face-like object, we use the horizontal derivative, I_x, and threshold its magnitude, $\left|I_x\right|$, to detect the sides of the face, eg *after smoothing*:

- use a 1 x 3 mask to compute $I_x \approx I(n+1,m) - I(n-1,m)$
  - take its absolute value
- complexity:1 arithmetic op./output pixel plus 1 bit-level op./output pixel
- minimum latency: 3+1=4 pixels
  - can reduce to 2+1 =3 pixels delay by using forward or backward difference

Often compare $\left|I_x\right|$ at each pixel to a fixed threshold to **detect** the face edges:

- complexity: 1 comparison op./output pixel
- latency: 1 pixel, but usually combined with the detection operations above

Statistical decision functions

- Performance is *often disappointing* because:
- image conditions may vary, rendering fixed threshold ineffective =>
  - many false positives and/or many false rejects
- multiple positive responses when $|I_x|$ is above threshold because of correlated output on neighbouring pixels
  - in particular caused by the smoothing

=> use an ***adaptive statistical decision function***

Need a model of output, eg:

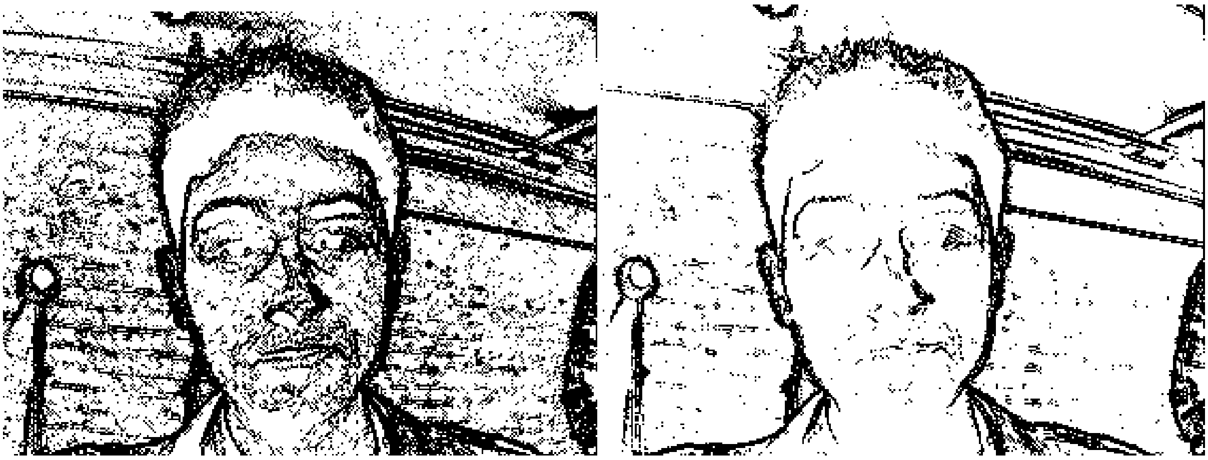
- $|I_x(m,n)| = \mu + \beta(n) + \varepsilon(m,n)$
- LMS estimate of μ and $\beta(n)$ over a P x Q neighbourhood subject to $\sum_{j=1}^{Q} \beta(j) = 0$
- compute the statistic $s = \beta(middle) - \sum_{other} \beta(j)$
- use an F-test for the hypothesis $s = 0$ against an adaptive threshold based on the statistics of the residuals $\varepsilon(m,n)$ in the P x Q neighbourhood
  - see eg. Kurz and Benteftifa (1997, CUP) and references therein

also, work by Deriche and by Crowley on use of adaptive thresholds in Canny edge detector

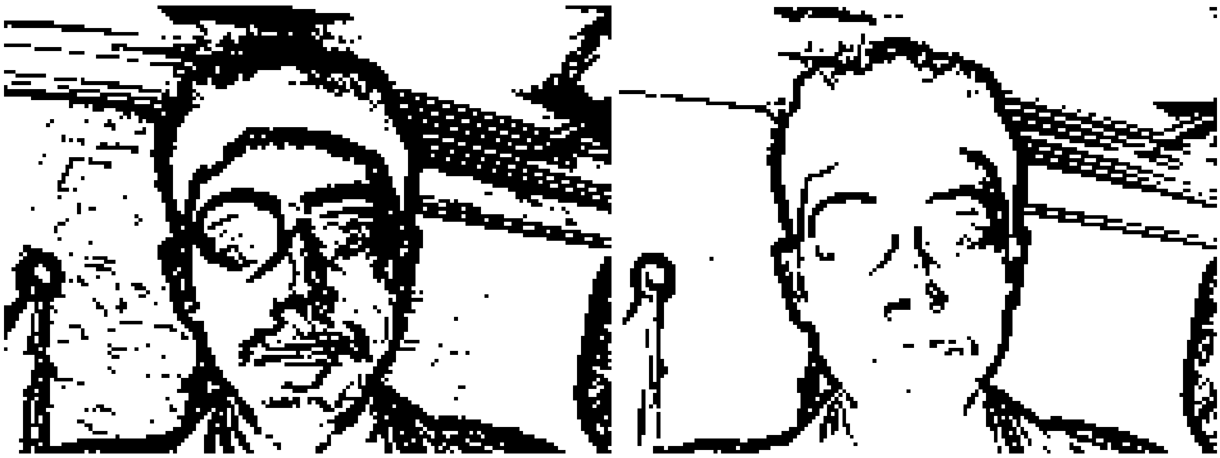
Examples:

For the face image above

♦ gradient image and after fixed thresholded



♦ gradient of smoothed image and after fixed threshold



Many other approaches to edge detection:

♦ useful references in Deriche's paper

♦ eg: Marr-Hildreth edge detector (PRS, 1980):

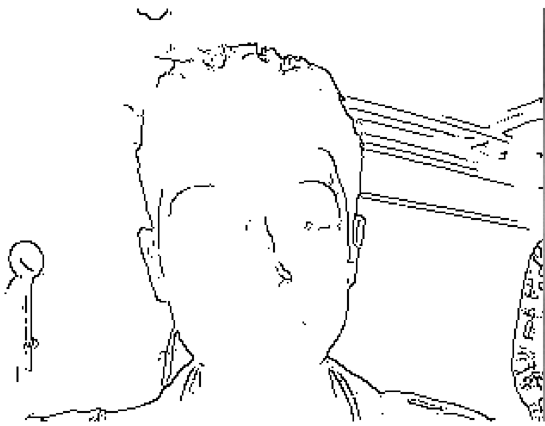
  * Laplacian of smoothed image
  * detect edges as *zero-crossings*

cf Full Canny edge detector:

♦ more complicated (see eg: Trucco and Verri, 1998)
  - non-maximal suppression
    * to avoid multiple responses at neighbouring pixels
  - high and low thresholds
    * retain weak edge pixels for further processing
  - edge linking and hysteresis step
    * to include weak edge pixels linked to strong ones

Example:

♦ Canny output and face image for comparison

Edge detection is still an area of research, eg:

♦ optimal filter design
  * Spacek (1983), Petrou (1984), Canny (1986), etc.

♦ alternative approaches
  * eg: Smith ~ 1993, Harris (1995, et seq)
  * regularisation, eg: Poggio et al (1995)
  * global methods, eg: GNC, Blake and Zisserman (1987); Markov random fields, Geman (1984 et seq, 1993), Besag (1974, et seq, 1993)

♦ performance characterisation: Haralick 1995, Forstner 1996

Template matching

Having detected that there may be a face in the image, we need some sort of masking or grouping operation to find the face region.

- ♦ There are many approaches to this. We shall assume that a fixed template T of P x Q pixels is to be matched to the image pixels =>
  - * cross-correlation of T with the image I
  - * or computation of SSD or SAD

In either case, computational complexity is very high. For example: consider cross-correlation

$$C(m,n) = \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} T(p,q) I(m+p, n+q)$$

with fixed coefficients $T(p,q)$.

- ♦ computational complexity is very high:
  - PQ multiplication + (P-1)+(Q-1)P additions $\approx 2PQ$ arith. ops/output pixel
  - processing time $\leq$1 pixel
  - video rate => ~ 20PQ MOPS
  - if P~Q~128 pixels, => ~320 GOPS !! =>
- ♦ latency is also high, at least:
  - (P-1) rows +Q pixels + 1 pixel ~ 8.2 ms
- ♦ memory required ~ (P-1) rows + Q pixels =>
  ~98 Kbytes if P~ Q ~128

Efficient Template matching

Use of powerful DSPs, VLSI devices, and multiprocessor systems can help meet the above processing requirement

- but it is *so high* that better algorithms and efficient implementations are needed
  - * cannot expect template to be separable

Often, a multi-resolution approach is used

- sub-sample *both* the image and the template by a a factor 2^{ν} (ν levels of the pyramid)
- reduces processing for an M x N image to:
- $\sim \dfrac{2MNPQ}{2^{2\nu}2^{2\nu}}$ arith. ops. (mult. & add) at νth level
- plus, for subsequent levels of pyramid:

$$2\left(2^2\frac{PQ}{2^{2\nu-2}}+2^2\frac{PQ}{2^{2\nu-4}}+\ldots 2^2\frac{PQ}{2^0}\right)$$

$$=8PQ\left(\frac{1-\left(\frac{1}{4}\right)^{\nu}}{1-\frac{1}{4}}\right)=\frac{32PQ}{3}\left(1-\left(\frac{1}{4}\right)^{\nu}\right) \Rightarrow$$

- ie $\sim \dfrac{2PQ}{2^{2\nu}2^{2\nu}}+\dfrac{32PQ}{3MN}\left(1-\left(\dfrac{1}{4}\right)^{\nu}\right)$

arith. ops/output pixel (i)

Multi-resolution matching

The computational complexity is dominated by the first term in (i) especially for large images.

For example:

- ♦ if P=Q=128, M=N=512 ν=3 =>
  - * ~4+1=5 ops/output pixel =>
  - * for video rate ~ 50 MOPS
  - * ~ same result for *larger* images, ie larger MN (NB: ops/output pixel)

Note: if we use the edge pixels, the above can be reduced single bit (logical) operations

- ♦ usually pre-compute and buffer the image pyramid =>
  - • requires ~1.33 frames to be buffered
  - • latency of 1 frame + processing delay
    - * latter is negligible according to (i) above, though, in practice, can expect it to be larger than (i) unless special care is taken with addressing etc.

Exercise:

- ♦ Use (i) to explore how many levels of the image pyramid it is best to use.
- ♦ Comment on whether your conclusions are affected by the overhead of pre-computing the pyramid.

Location

Having detected a face in the image, we now imagine calculating its **location** by computing the centroid of the edge pixels in the face region

$$\underline{x} = \frac{1}{\mathcal{N}} \sum_{\mathcal{R}} \left(mS(M,n), nS(m,n) \right)$$

- where $\mathcal{N}$ =# pixels in region $\mathcal{R}$ and we have used S to indicate that we could, for example, be calculating the centroid of the edge pixels in the face region, or of some other processing output
- above is a special case of calculating the moments of the region
- can simplify by using Greens theorem, but needs care as to stability and robustness
- computational complexity and the latency is determined by the size of the region $\mathcal{R}$

Example: suppose we want to calculate the centroid of the face edge => above simplifies to a sum over the edge contours, $\mathcal{C}$: $\underline{x} = \frac{1}{|\mathcal{C}|} \sum_{\mathcal{C}} (m,n)$

- complexity ~ $2|\mathcal{C}|$ where $\mathcal{C}$ = # contour pixels
- => low complexity, eg ~2(P+Q) if the face region is of size P x Q
- similarly: latency ~ P rows

Motion estimation

One of the simplest ways to estimate image motion is via the motion constraint equation:

$$\nabla \mathrm{I} + \frac{\partial \mathrm{I}}{\partial \mathrm{t}} = \frac{\mathrm{dI}}{\mathrm{dt}} = 0 \qquad \text{(ii)}$$

- ♦ if the motion is only in the x direction =>

$$\mathrm{u} = -\frac{\partial \mathrm{I}}{\partial \mathrm{t}} \Big/ \frac{\partial \mathrm{I}}{\partial \mathrm{x}} \qquad \text{(iii)}$$

- ♦ need to calculate the spatial and temporal derivatives =>
  - * smooth image before applying (i) or (iii)
  - * spatial derivatives have already been discussed so concentrate on temporal
- ♦ smooth with a causal filter, eg. an exponential
- ♦ => easy recursive implementation
- ♦ in principle, use backward difference

$$\frac{\partial \mathrm{I}}{\partial \mathrm{t}} \approx I(m,n,t) - I(m,n,t-1)$$

- ♦ => low complexity, latency ~ 1 frame
- ♦ in practice, differentiate the filter analytically
- ♦ then, recursive calculation => low complexity, and, if filtered images stored, latency ~ 1 pixel

Exercise:

- ♦ Carry out the above for a filter $\alpha e^{-\alpha t}$ for $t \geq 0$.

Predict, measure, update cycle

Above may be used to *detect, segment* (group), and *locate*, a face, and, from the *motion* to *predict* where it will be in subsequent frames =>

- ♦ could focus subsequent processing on where the face is *expected* to be
- ♦ *must* include a region of *uncertainty*
    - ∗ need to estimate it (cf. Kalman filter)
    - ∗ say U x V pixels
- ♦ reduces processing by UV/MN
- ♦ can apply to all subsequent processing, including, low-level filtering operations
    - • sometimes filtering is always carried out over the whole image, eg because
        - ∗ carried out in dedicated system
        - ∗ simpler implementation (eg for a recursive temporal filter)
- ♦ requires an initialisation step or "boot" mode which *must* itself be *fast enough*
    - ∗ not necessarily video rate
    - ∗ could use other inputs
- ♦ needs a recovery mechanism when fails
    - ∗ to ensure "continuous" real-time operation

Examples: Inoue (software tracker), Murray (active stereo system), Craw (driver eye tracker, in UNIX!).

Cross-correlation matching

Often used as the measurement step in the *predict, measure, update* cycle accurately to locate an object => applied over a restricted *region of interest*

- ♦ similar to template matching, but
  - * template may change every frame
  - for tracking, may reduce search space by using a small search window centred on location found in previous image and/or by use of a multi-resolution pyramid
  - eg. Kalman filter for estimating location and its accuracy and for predicting the expected position in next frame and its uncertainty
    - * => complicated algorithm
  - necessary to analyse [cf: Knuth]
    - * expected or average case
    - * worst case
  - algorithm's behaviour is data dependent
    - * not completely predictable
    - * need to build mechanisms to recover when tracking fails
  - similarly requires a "boot" mode or mechanism to initialise the algorithm