SE3IA11/SEMIP12
Image Analysis

University of Reading

Symbolic Feature Extraction

Lecturer:
Prof. James Ferryman
Computational Vision Group
Email: j.m.ferryman@reading.ac.uk

University of Reading

Introduction

- Many vision applications require symbolic image features (such as lines, circles, etc.) rather than the raw intensity values (iconic image data)

- This lecture asks - how can we convert an iconic description to a symbolic one?

- The symbolic description is required before we can "reason" about the image – visual intelligence – to be covered next term

University of Reading

Describing the Image

- Grouping pixels according to *similarity*, or finding a boundary where there is an abrupt *difference*, are obviously related!

- In both cases, we are collecting together previously independent points in the image to make a “whole” which can be studied in its own right

University of Reading

Describing the Image

- There are many reasons for trying to do this:

  - Practical – It reduces the memory required to describe the image
    - e.g. [line [0 10] [100 110]]
      uses far less computer memory than does (say):
      [[0 10 [1 11] ….. [100 110]]
      Likewise: [circle a b r] (etc, etc.)

    This is possible because the *higher level descriptions* use symbolic concepts which capture some of the redundancy of the *lower level descriptions*

Describing the Image

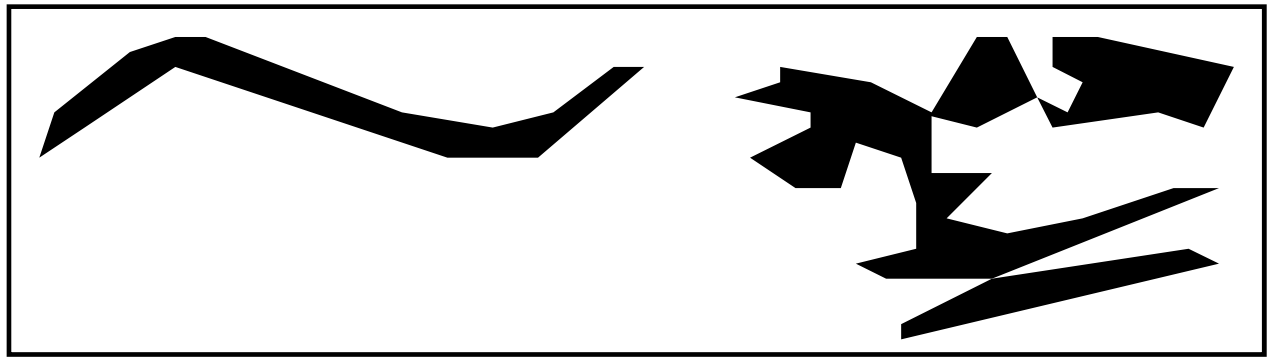
- There are many reasons for trying to do this:
  - Theoretical – High level descriptions allow new knowledge to be mobilised, which cannot be applied at lower levels

    E.g.: What angle has the line with another?
    Would it pass through (-10 0)?
    What is the distance to the line from (99,42)?
    Where does the line cut the circle?

- *To use high-level concepts, we must identify and label the image fragments as symbolic entities, and make explicit their properties*

Describing Regions

- A region is a connected set of pixels – how can it be described?

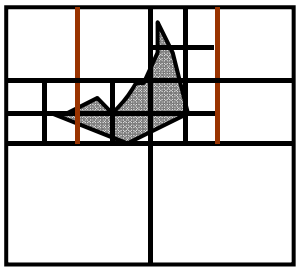


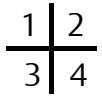
The abstract problem of analysing and describing continuous 2D patterns is very deep!

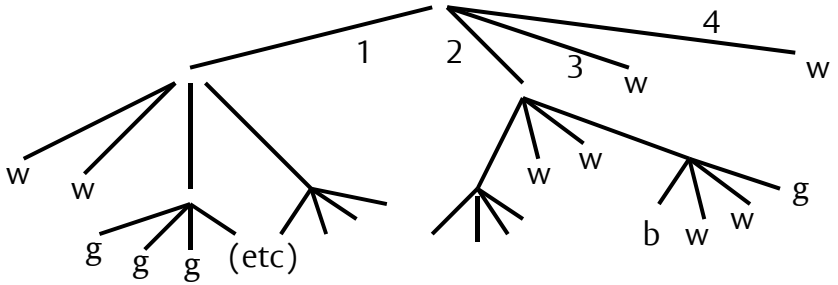
- Our task is somewhat simpler, because in discrete distributions there can be no infinite whorls, or other singularities (the subject of much discussion in mathematical analysis)

Describing Regions

- Unstructured representations:
  - Simple literal maps – e.g. 2D boolean matrix
    Fastest, dumbest, highly inefficient on memory
  - Recursive subdivisions, e.g. quadtrees







Rule: recursively explode any grey (g) square, until all squares are white (w) or black (b) (up to some minimum element size)

- Efficient on memory (especially for compact regions)
- Very fast for many operations (esp. union & intersection)

Extensively used in *constructive* applications (e.g. graphics)

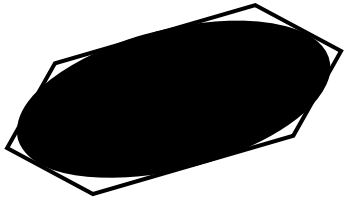
However: non-topological (i.e. adjacency is not preserved), so some "simple" operations are difficult (e.g. nearest points in two regions)

Describing Regions

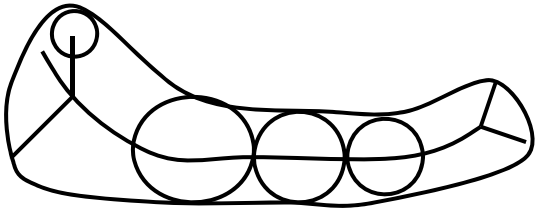
University of Reading

- Structured representations:

  - Polyline approxmations of the boundary: [x0 y0] … [xn yn]]



  - Ribbons: An axis, and a generator function. E.g. The median axis: the loci of centres of circles which touch the boundary at two places



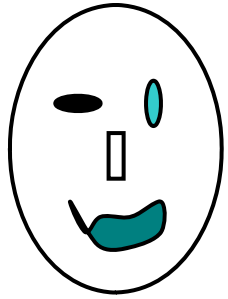
  - Splines: A set of control points with simple analytic curves (e.g. cubic) interpolated locally

  - Fourier descriptions (of a 2D image or a 1D curve)

Describing Regions

- Note how the different representations *make explicit* different properties of the regions (which may or, equally, may not be useful)

- However, some patterns have special significance – even if bizarrely distorted!



We also need to capture symmetries, compactness, and relationships between the parts

Classifying Regions

- However, none of these representations *classify* the segmented region, i.e. associate it with a higher order class, which can be reasoned about
- Some classes have a convenient *analytic expression*, straight lines, circles, conics, polynomial curves, etc. etc. From these can be made triangles, squares, simple curved shapes
- *Classification* depends on making a comparison between properties of the data and stored definitions of classes
- Once the class membership is accepted, then the battery of concepts and methods associated with the high-level class become available to subsequent analysis

Classifying Regions

University of Reading

- Measuring properties of regions:

- A wide range of methods are available:

  - *Statistical*
    - Moments – 1st order (= mean), 2nd order (= inertia), …
  - Topological
    - Number of connected regions
    - Number of holes (c.f. the Euler number)
  - Geometrical
    - Length of boundaries, types of concavity
    - Aspect ratio, compactness …

2D regions turn out to be very difficult to describe adequately - more attention has been directed towards describing their boundary lines, in order to discover consistencies, symmetries, etc.

University of Reading

The Canny Operator

- Various operators (Roberts, Sobel, Prewitt) were described in previous lectures
  - for locating pixels where there is significant discontinuity in intensity (edge detection)
- Such pixels are often called *edgels* (for edge elements)
- A far more popular operator for edge detection is due to John Canny (hence the name) who reported his result in 1984 as part of his PhD work

University of Reading

The Canny Operator

- The Canny Operator has become the “standard” method in advanced vision systems
- It overcomes some of the disadvantages of simpler operators but takes far more computing effort!
- The Canny Operator attempts to find the points in the image where the derivative of the greylevel has a local maximum
  - i.e. the points of inflection of the greylevel surface

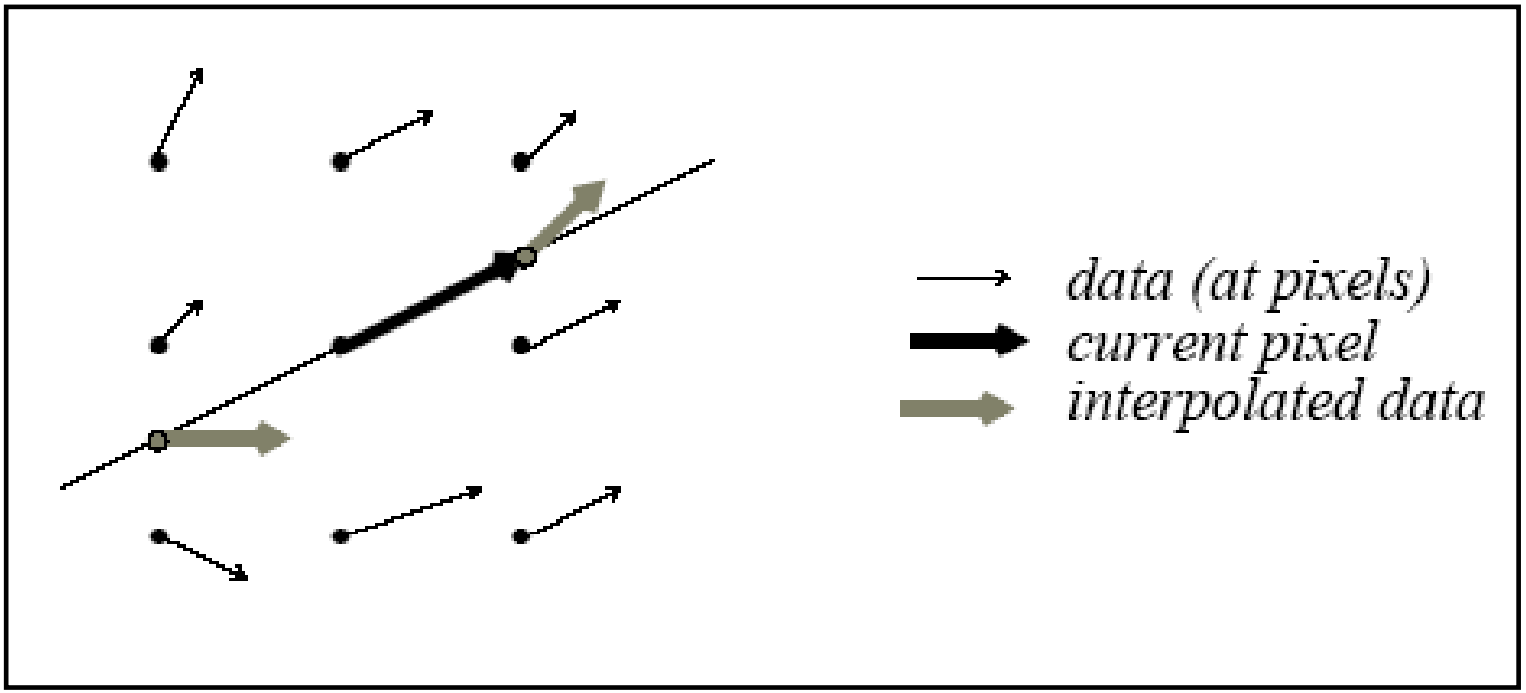
University of Reading

Major Steps in Canny Operator

(1) Estimate the gradient vector at each pixel by taking the simple partial derivatives defined by the weights (-1 0 1). This provides the gradient direction and magnitude

(2) At each pixel, examine the magnitudes of the gradient at adjacent points along the direction of the gradient at the current pixel. Since the gradient direction might not point directly towards a pixel, this requires the values to be interpolated between two nearest pixels

Interpolation of Image Data



Major Steps in Canny Operator

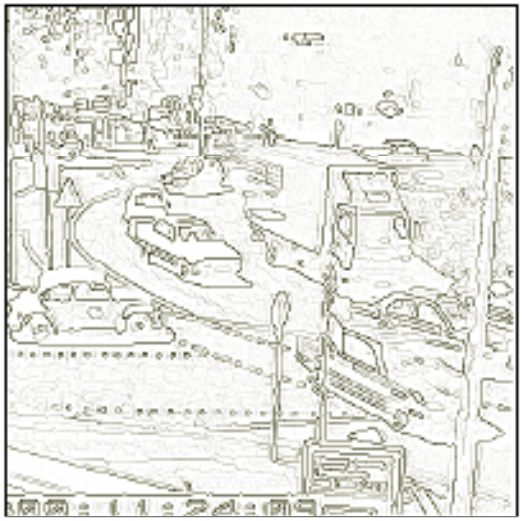
- (3) *Non-maximum suppression:* only record those pixels whose magnitudes are *greater* than their interpolated neighbours
  - also require them to exceed some minimal threshold to avoid pure "noise"

This picks out the peaks along *lines of steepest ascent* without suppressing points because they are smaller than their neighbours *along the contours*

University of Reading

Example of Canny Edge Detection

Intensity image



Grad mag. of Canny edgels

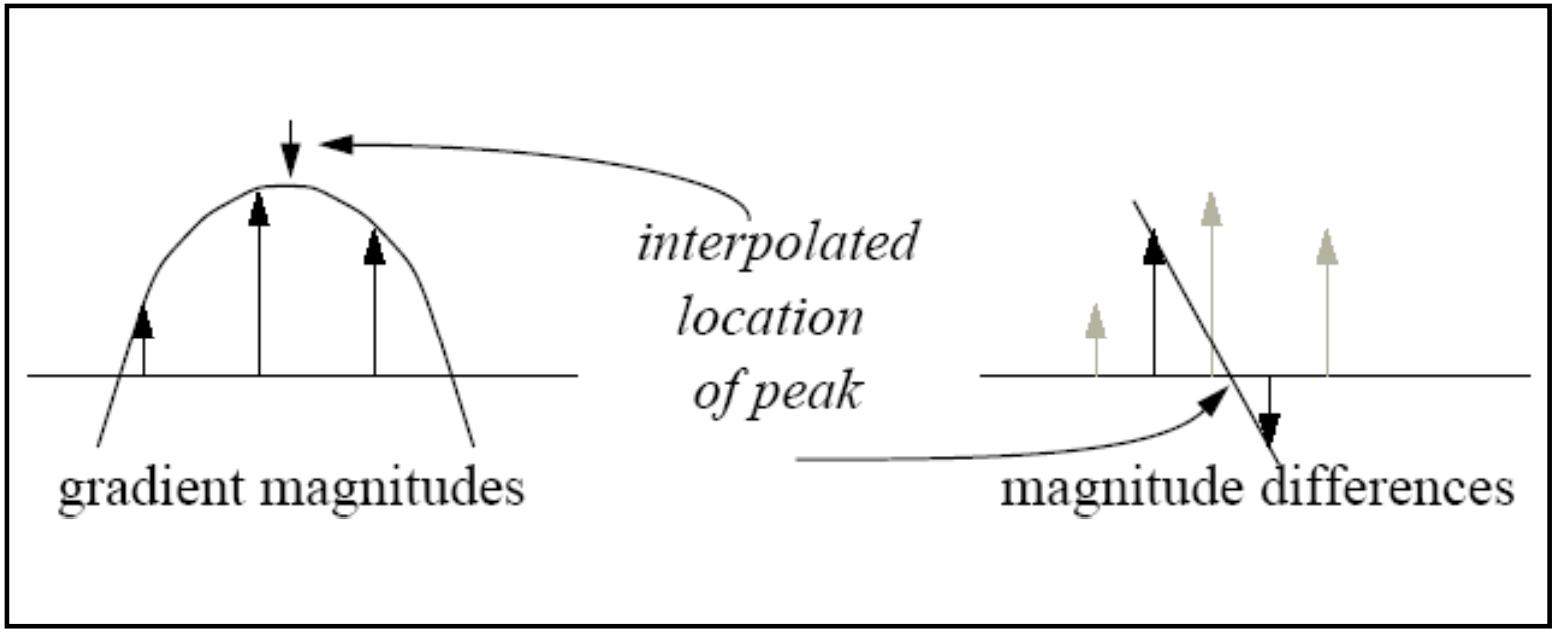
University of Reading

Major Steps in Canny Operator

- (4) The position of the edge can now be obtained to sub-pixel accuracy, by looking at the local derivative of the gradient
  - i.e. the differences between the gradient of the current pixel and its two interpolated neighbours, and finding the zero point between them
    - necessarily, one neighbour is +ve and the other is –ve

This is equivalent to fitting a parabola (a quadratic) to the gradient magnitude data and finding its peak

Edge Localisation



Major Steps in Canny Operator

- (5) We are then left with a sparse map of "edgels", which (with luck) form long streams of profiles, running along the hillsides in the grey-surface, at the points where the hill is steepest

- Each edgel is tagged by its exact (sub-pixel) position, and the magnitude and direction of the gradient vector

Major Steps in Canny Operator

- (6) Except in very unusual image conditions, each edgel has only two neighbouring edgels (or 1 if it is at an end)

- A *connect* process is then run, to tie together the profiles into sequences (or "strings")

- This process uses *hysteresis* to help pay more attention to edgels which form large structures, without being distracted by small weak strings
  - i.e. find an edgel which exceeds a fairly high theshold, then propagate along its neighbours provided they exceed a lower threshold

Multiple Scales

- Like all differentiation processes, the Canny operator is intrinsically sensitive to noise
- To maintain this under control, the image is first of all smoothed by means of a Gaussian blur function
  - whose width is controlled by parameter σ
- As we have seen in previous lectures, smoothing is essentially a process of integration, which is the "opposite" of differentiation
- By trading off the two we can select particular scales of edges
  - e.g. if we look at a bush, we might select the overall outline (low frequency, large scale), or pick out individual leaves (high frequency, fine scale)
- By varying the width of the Gaussian, it is possible to bias the Canny operator to pick out a particular *scale of detail*

University of Reading

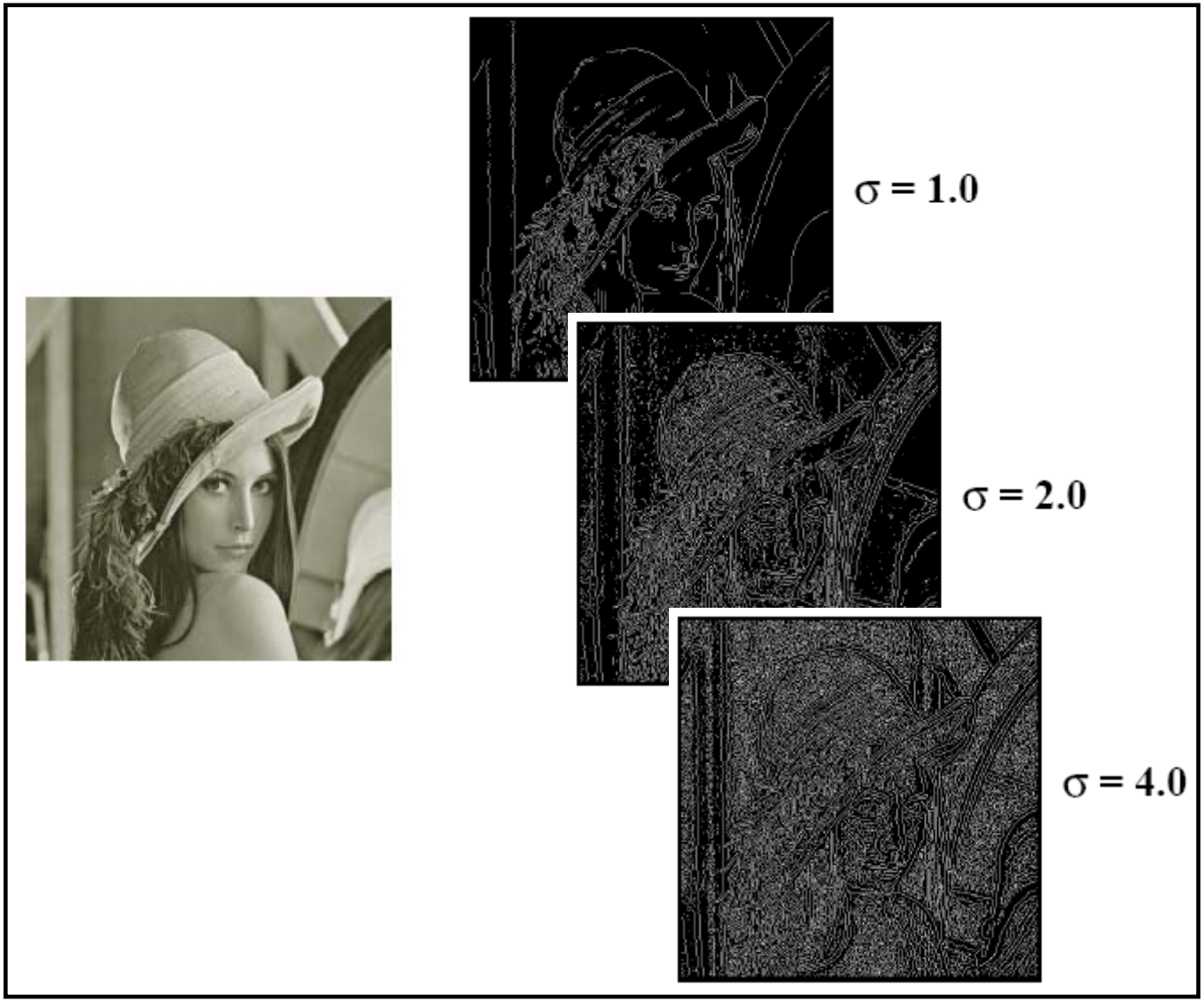
Multiple Scales

- Thus it is possible to run the whole process several times at different scales to identify fine edges and coarse edges separately (see example next slide), but (of course) it makes the process far more computationally expensive

- Q: in practice how do you determine which scale is most relevant to any particular task?

- A: use of multiple scales

Canny Edgels at Multiple Scales



University of Reading

Describing Lines

- As we have seen, early image processing (i.e. the initial processing of the image data) often yields a sequence of edgels, which form a connected line
- All that is made explicit is the geometrical positions of the edgels or strings and their ordinal positions
- There is a wide variety of methods for deriving descriptions of a line in terms of higher order entities (straight lines, arcs of circles, …)

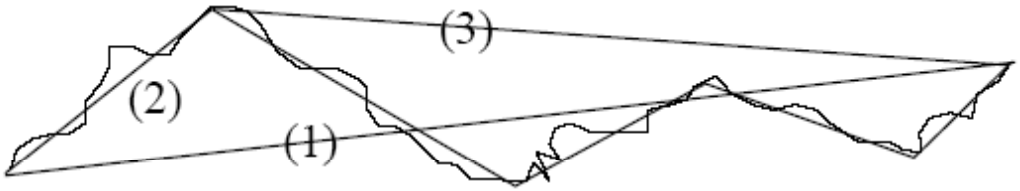
University of Reading

Line Extraction

- The next phase after the Canny operator is typically to try to build descriptions of the "strings" of Canny edgels in terms of high-level symbolic entities
  - for example, straight lines, and circular arcs
- A common approach is use a polyline approximation
  - i.e. each edgel string is described by a set of straight line segments
- The derivation of a good polyline description is surprisingly difficult, mainly since the approximation can be made at many different scales of detail
  - it is very hard to specify an appropriate scale for any particular purpose

Line Extraction

- One common approach is to specify a "noise" criterion (c) and use a *split-and-merge* technique:



Polyline Approximation

- *Fit a straight line between the two ends (1)*
- *If the mean squared error of the points from the line >c then*
- *Split the input data at the point of maximum deviation, making sub-lines (2) & (3)*
- *Iterate for the two sub-lines (2) and (3) etc.*

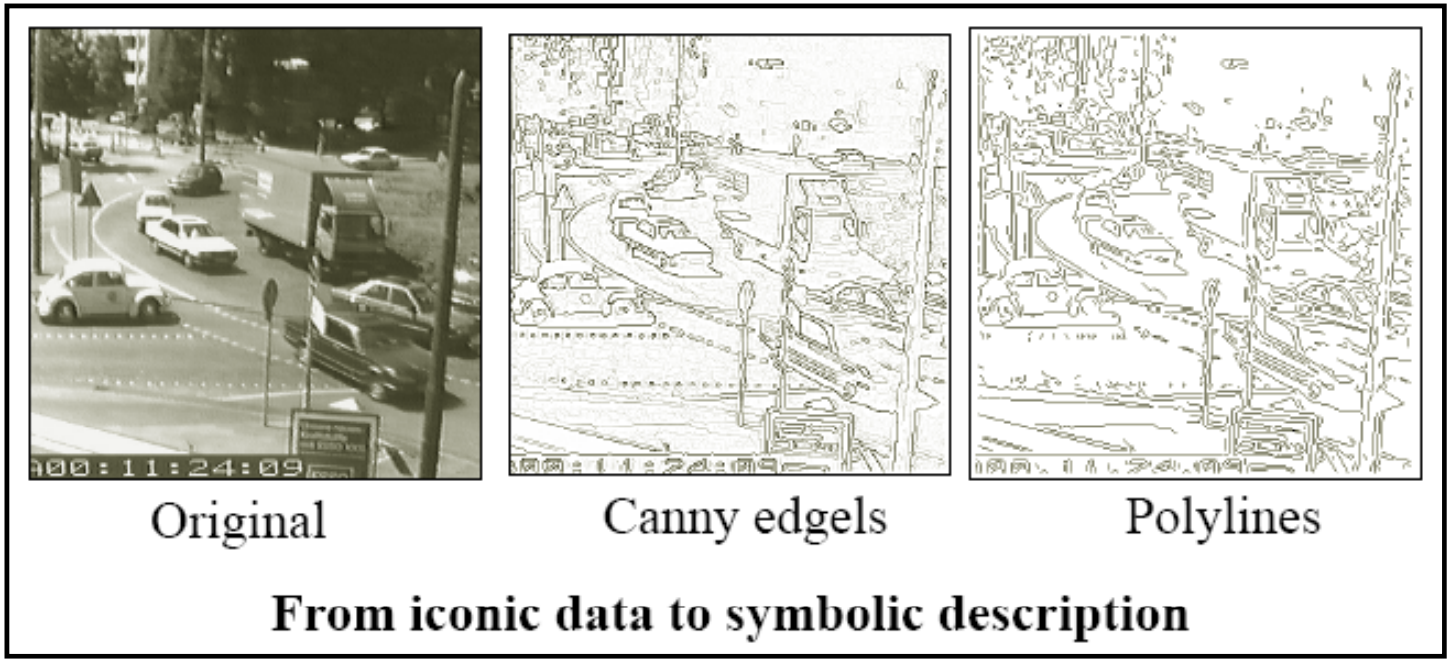
University of Reading

Line Extraction

- Other error terms are possible which may be computationally cheaper
  - E.g.:
    - using the maximum deviation as the criterion of acceptability of a line
    - using the curvature maximum as the splitting point

Line Extraction

- The following is an example of polyline approximation:



University of Reading

Line Extraction

- Notice how poor the results are (it is much harder for a human to interpret the polyline image, than the original image – think WHY it's so)!
- However, the big achievement is that we have reduced the *mass of low-quality data* in the image (any individual pixel of which could have come from any possible object), to a far *smaller set of higher-quality data* in a symbolic description
  - which we hope will help discriminate between objects (the subject of exploration next term)
- In this case, we only have a few hundred entities, stored in a totally symbolic database consisting of theoretical lines and arcs, etc., which will be used to reason about the identity of the objects
  - we will also be exploring this topic next term too

University of Reading

Intrinsic Representation of a Curve

- A cartesian representation of a curve (e.g. $\{(x_i, y_i): (i=1 \ldots n)\}$) is usually inconvenient: it fails to make explicit important concepts like distance along the curve (e.g. the length of the curve)
- The use of an *intrinsic parameter* is common: i.e. the curve is parameterised by *s*, the distance along the curve from an origin
- The curve is now represented as $\{ (x(s),y(s)) : a \leq s \leq b \}$
- Note: we still need the curve to be sampled at discrete points, which we usually want to be equally spaced
- The new samples *will not correspond to the original (x,y) points*

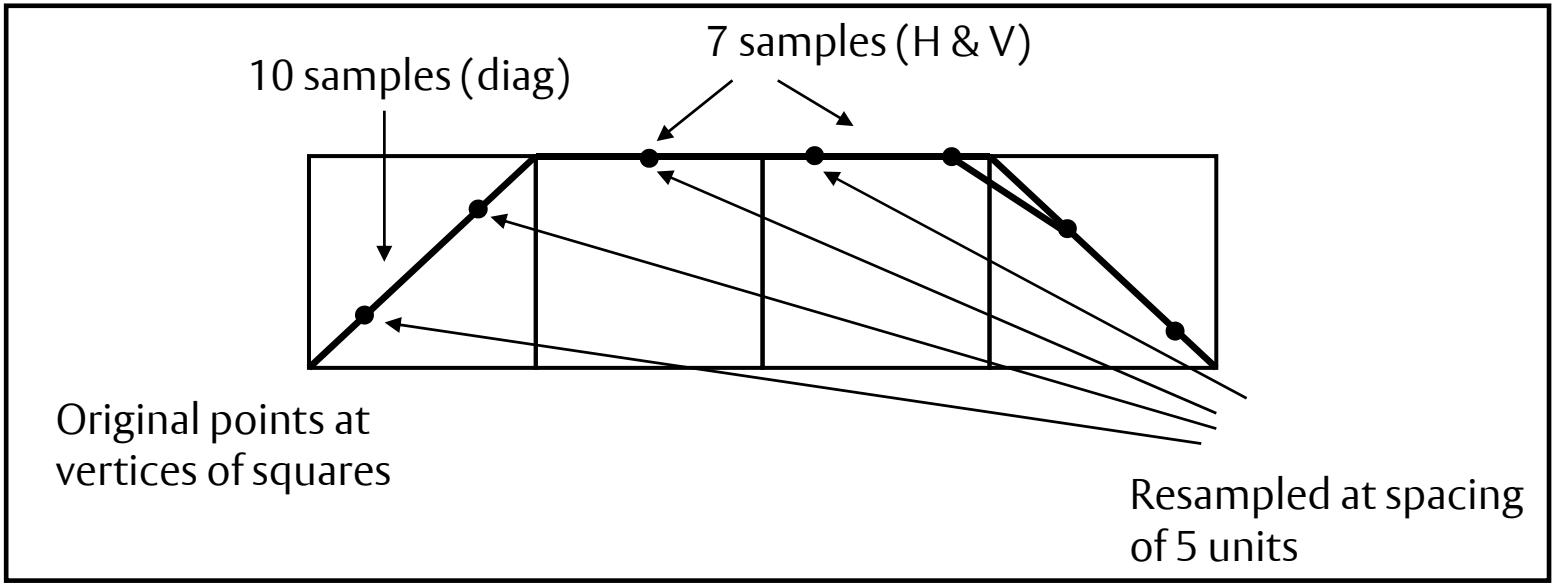
University of Reading

Intrinsic Representation of a Curve

- One way of overcoming this difficulty is to *resample* the line at more nearly equal intervals
  - this is relatively straightforward, for a connected curve (with square resampling) since we know that successive points must be at angles of multiplies of $\pi/4$.
- The distances between adjacent points are therefore 1 or sqrt(2), which are (approximately) equivalent to the ratio 7:10

Intrinsic Representation of a Curve

- So to produce a regular resampling: subsample the line fragments between pixel positions by 7 (H or V) or 10 (diagonals), then *resample* at a constant spacing (e.g. 5) to give the *s* parameter



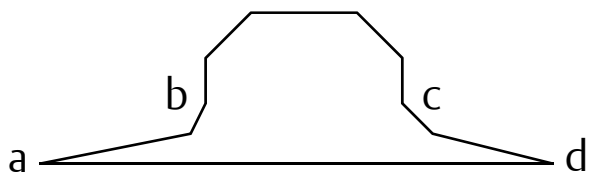
Curvature

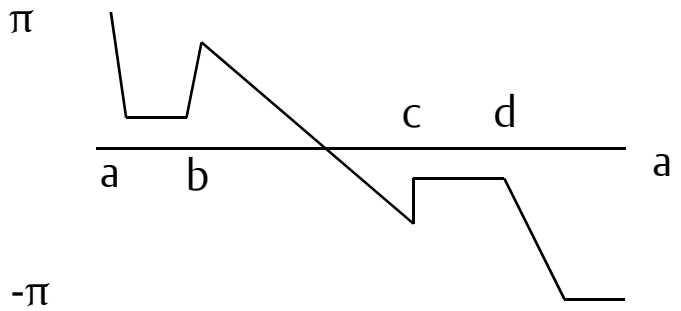
- The use of the intrinsic equation with regular sampling, allows concepts such as *curvature* to be introduced:
  - the derivative of the x(s) and y(s), w.r.t. s define the tangent to the curve at s: $\psi(s) = \arctan(\, dy/ds \,/\, dx/ds \,)$

    [tends to: arctan(dy/dx) for small ds]
- This representation is sometimes called the ψ - s function
- Note that the $d\psi/ds$ gives a measure of the *curvature*
- This is easy to approximate in the discrete sample, and provides an alternative means of segmenting a general curve into higher order components

Curvature

- E.g.: if the curvature is (locally):
  - 0, then the curve is a straight line
  - constant (non-zero), then the curve is circular
  - an extreme value, then there is a corner





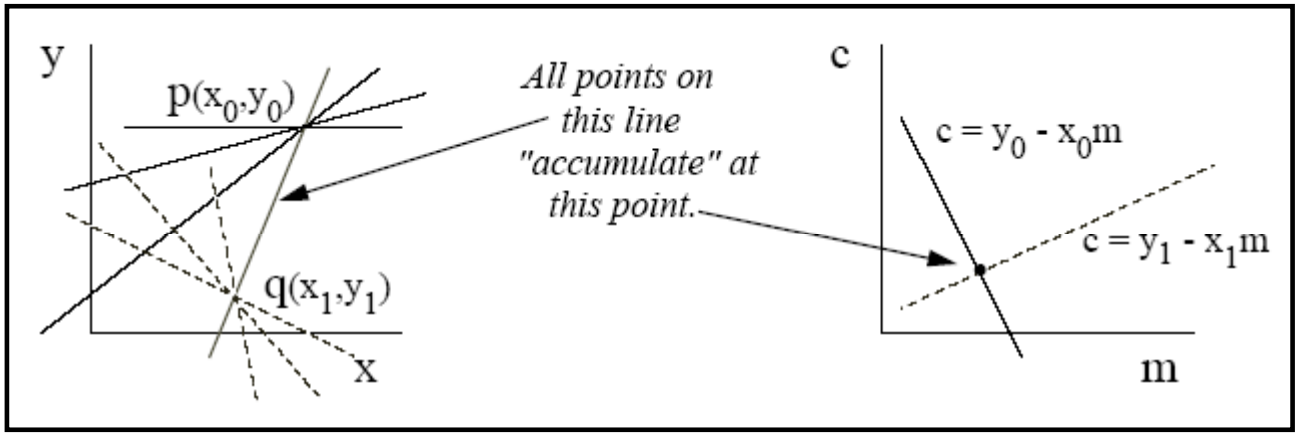
From bottom left, clockwise:
Convex, straight, concave, circle, concave, straight, convex, straight (xsvcvsxs)

- *A high level description of the curve has now been derived, as parts and their relationships along the curve (sometimes called a syntactic description)*
- *Allowing for cyclic permutations, it is independent of the orientation and position of the curve, but it is very dependent on the definitions of the components, and cannot distinguish distortions of size*

Hough Transform

- An alternative and powerful way to detect a known symbolic entity or pattern (e.g. a straight line, circle, etc.) works by searching the *parameter space* of the pattern
  - the idea was first proposed by P. Hough which is not called the Hough Transform
- The simplest (and most widely used) example looks for straight lines in an image
  - suppose we have applied an "edge-finder" to derive an edgel map
    - how can we find good, elongated lines?

Hough Transform



- Consider an edgel at p at (x_0,y_0) – it could arise from any line whose parameters m and c satisfy $y_0 = m_x + c$.
- However, this equation can be taken as a line in (m,c) space – which defines all the (m,c) pairs of all possible lines in (x,y) constrained to pass through p
- This is true for any edgel
  - e.g. that at q (x_1,y_1), defining the line $c = y_1 - x_1m$.

University of Reading

Hough Transform

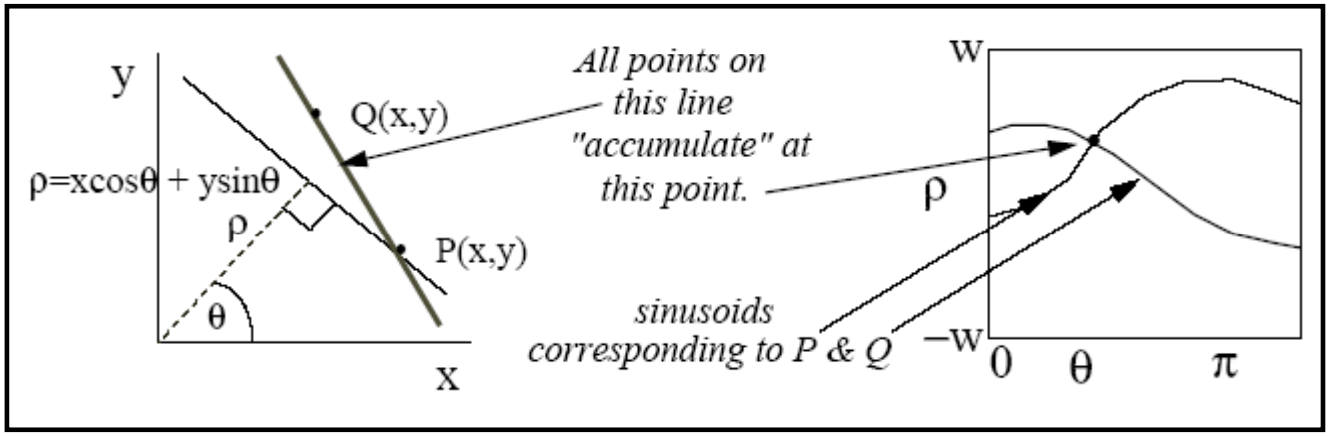
- Note the "dualism": the *line* which is common to p & q, maps into the *point* which is common to the two lines in (m,c) space
- So, to discover the straight lines in an image, we draw all the lines in the "parameter" space (m,c) corresponding to all the edgels discovered in the image, and look for multiple intersections
- This is done by representing (m,c) as a discretely-sampled "accumulator array" (initially zeroed), which is incremented by one everywhere a line passes through it – then look for peaks in the "votes"

Alternative Parameterisation

- In practice (m,c) space is very inconvenient
  - both are unbounded, i.e. to cope with all cases, both m & c must take values between ±∞
- This makes it very awkward to represent the (m,c) space in a finite array
- An alternative parameterisation of the straight line is commonly used

$$\rho = x\cos\theta + y\sin\theta$$

Alternative Parameterisation



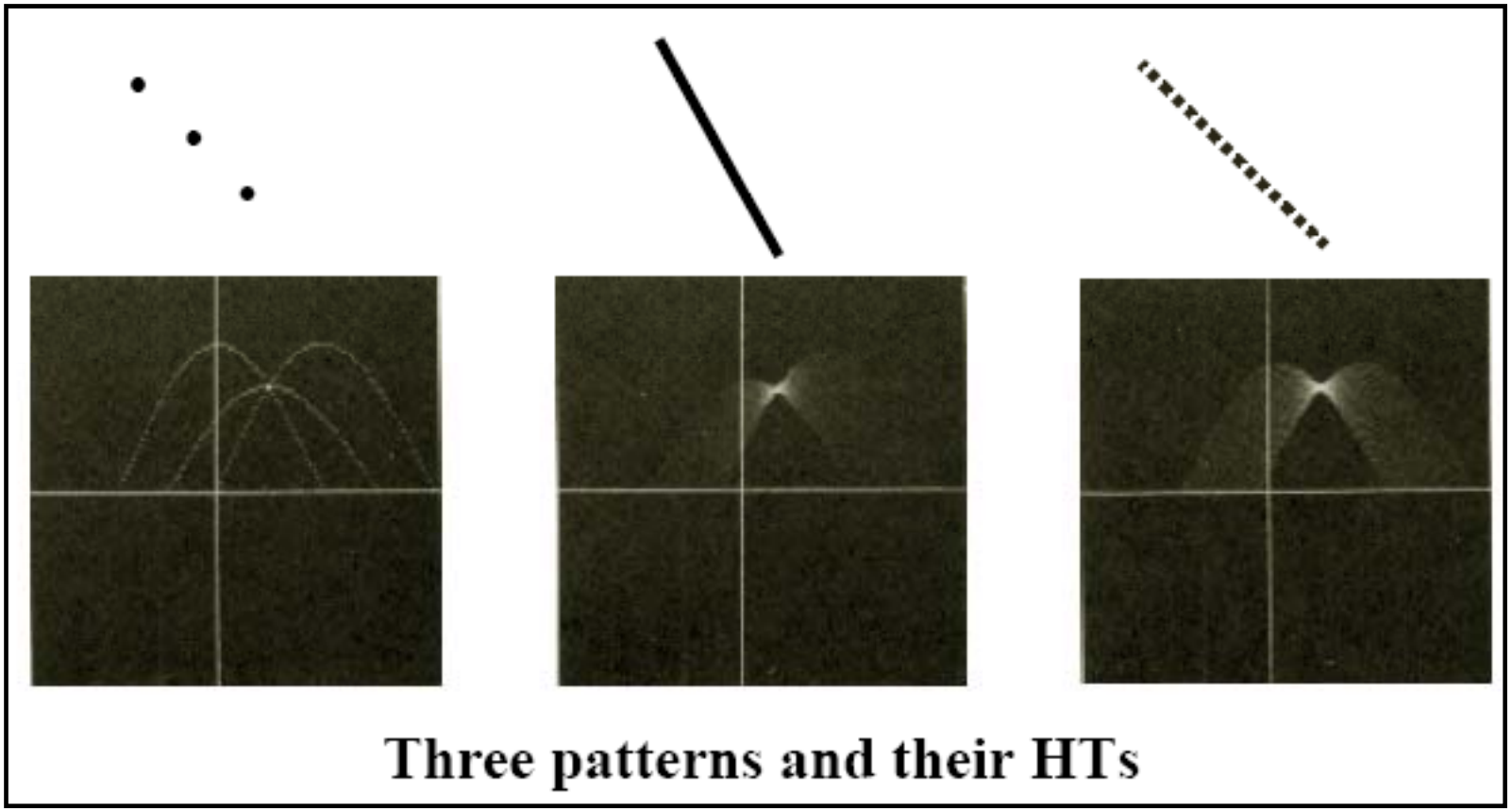
- The set of all lines passing through a single edgel in (x,y) space now creates a sinusoid in (p,θ) space; however we now need only be concerned with values of (p,θ) in the bounded region:

  $0 \leq \theta \leq \pi$ (we only need ½ a sinusoid, since (p,θ) is (-p,-θ))

  $-w \leq \rho \leq w$ (where w is the diagonal diameter of the image)
- This is far easier to store as an array (though the accumulation process is somewhat harder – drawing a sine wave vs. a line)

University of Reading

Example of Hough Transforms



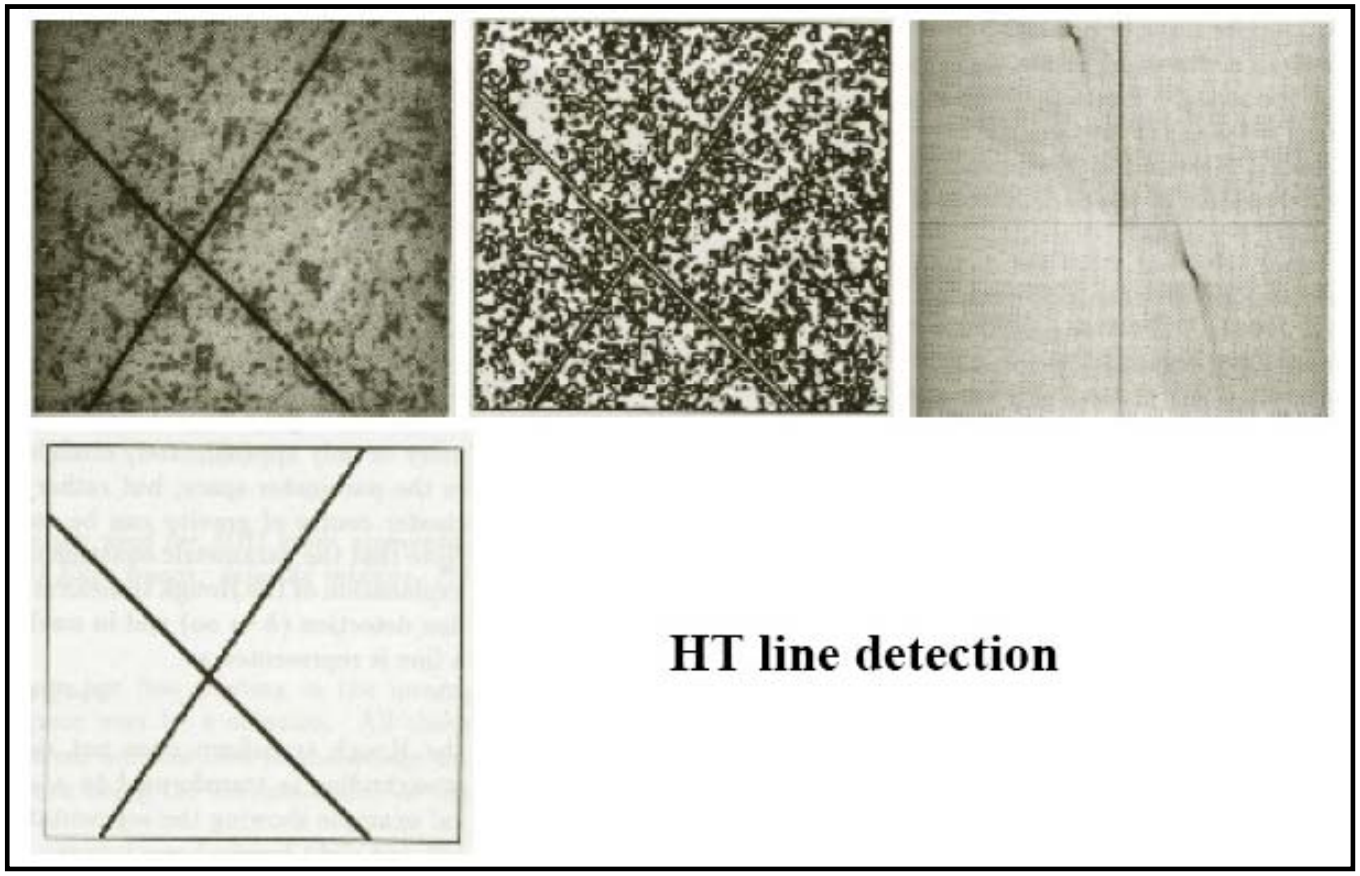
Three patterns and their HTs

University of Reading

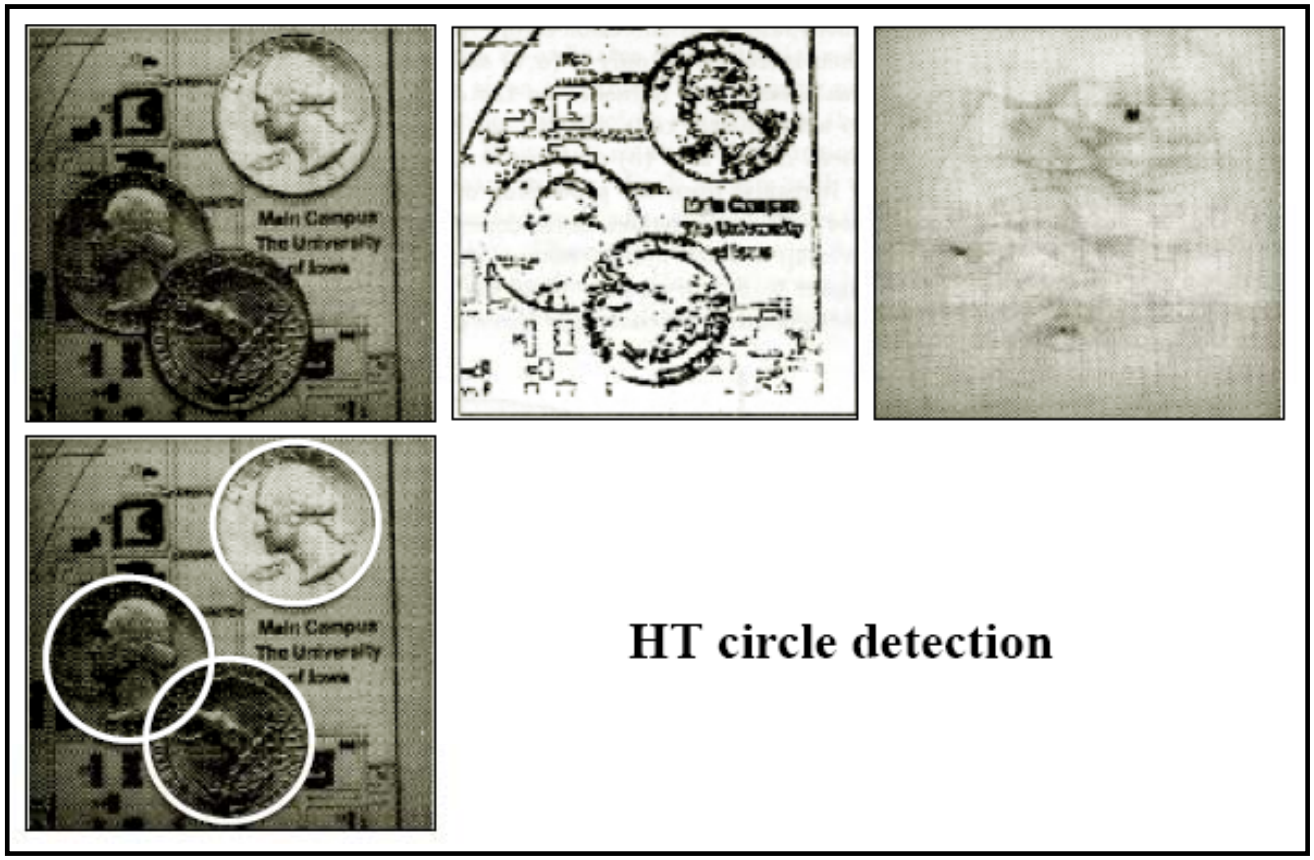
Generalised Hough Transform

- The Hough Transform can easily be generalised to any pattern which can be specified by an analytical equation f(x,a)=0, where x is an image point and a is the parameter vector
- The general procedure is as follows:
  - Initialise accumulator array A(a) to zero
  - For each edgel x, compute all a such that f(x,a)=0 and increase A(a) by 1
  - Local maxima in A correspond to instances of the pattern in the image
- The Hough transform may be further extended to arbitrary shapes specified by a sequence of boundary points

Examples of Hough Transform



Examples of Hough Transform



University of Reading

Summary

- Overview provided of symbolic feature extraction
- The lecture next week covers morphological image processing