Department of Computer Engineering
University of California at Santa Cruz

Line and curve detection

- Find lines, curves, or parts of lines or curves in an input image. Such an image might be the output of a edge detector discussed in the previous lectures
- Two problems
  - Grouping – Find the set of points compose each instance of the target curve in the image
  - Model fitting – Given a group, find the curve or line that explains the data points best

Department of Computer Engineering
University of California at Santa Cruz

Hough transform for lines

- Basic ideas
  - A line is represented as $y = mx + n$. Every line in the image corresponds to a point (m, n) in the parameter space
  - Every point in the image domain corresponds to a line in the parameter space (why ? Fix (x,y), (m,n) can change on the line $n = y - mx$. In other words, for all the lines passing through (x,y) in the image space, their parameters form a line in the parameter space)
  - Points along a line in the space correspond to lines passing through the same point in the parameter space (why ?)

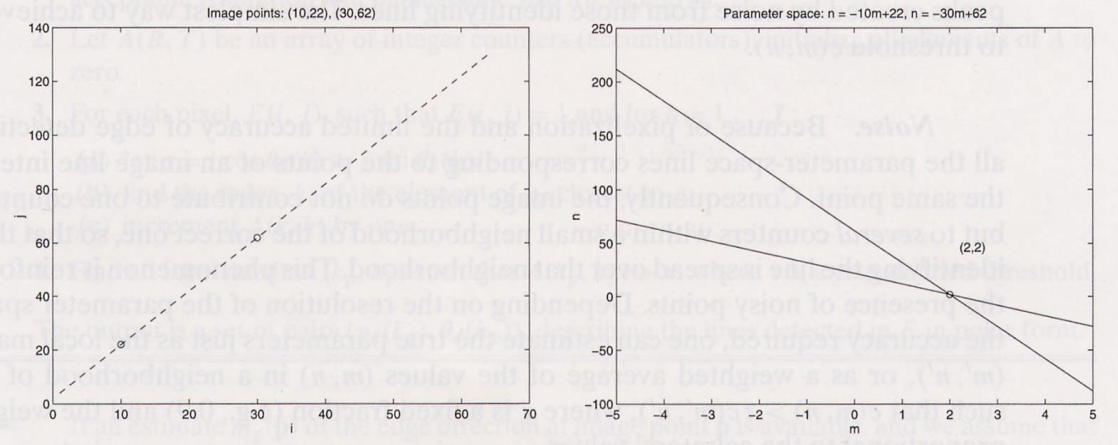
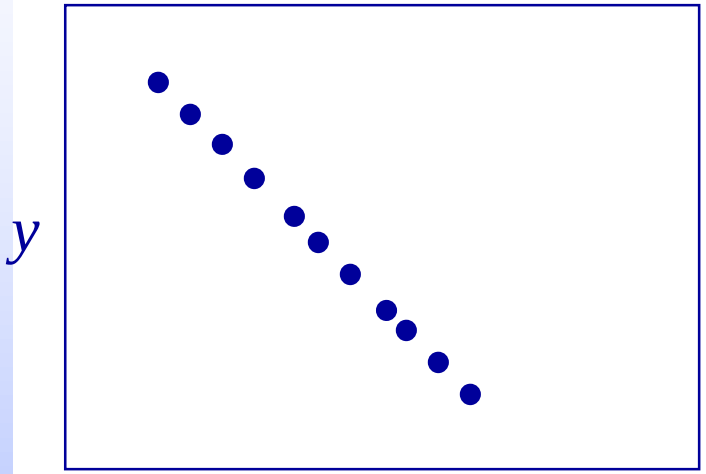
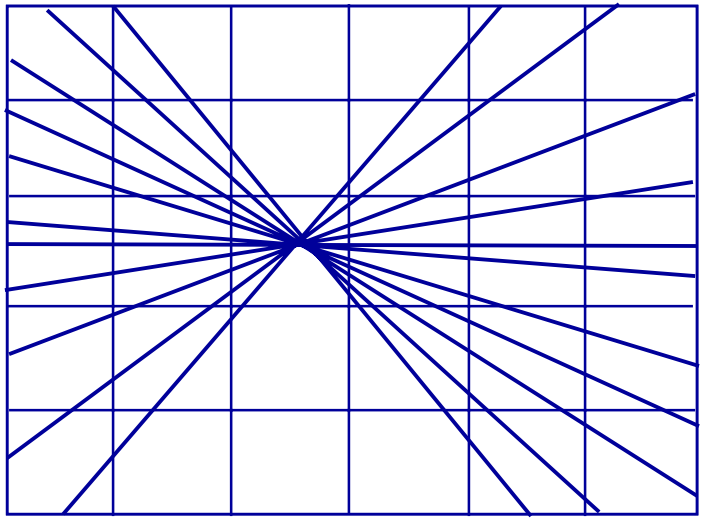


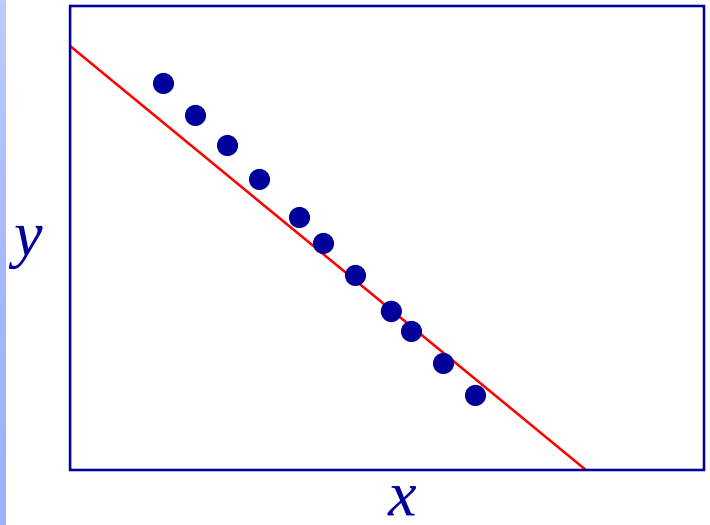
Figure 5.1 Illustration of the basic idea of the Hough transform for lines. The two image points (a) are mapped onto two lines in parameter space (b). The coordinates of the intersection of these lines are the parameters (m, n) of the image line through the two points.

Department of Computer Engineering
University of California at Santa Cruz

Hough transform for lines







2	2	0	1	2	2
3	5	3	3	2	2
3	7	11	10	4	3
2	3	1	4	5	2
2	1	0	1	3	3

n

m

Department of Computer Engineering
University of California at Santa Cruz

Hough transform for lines

- A real example
  - Two lines correspond to two peaks in the parameter space

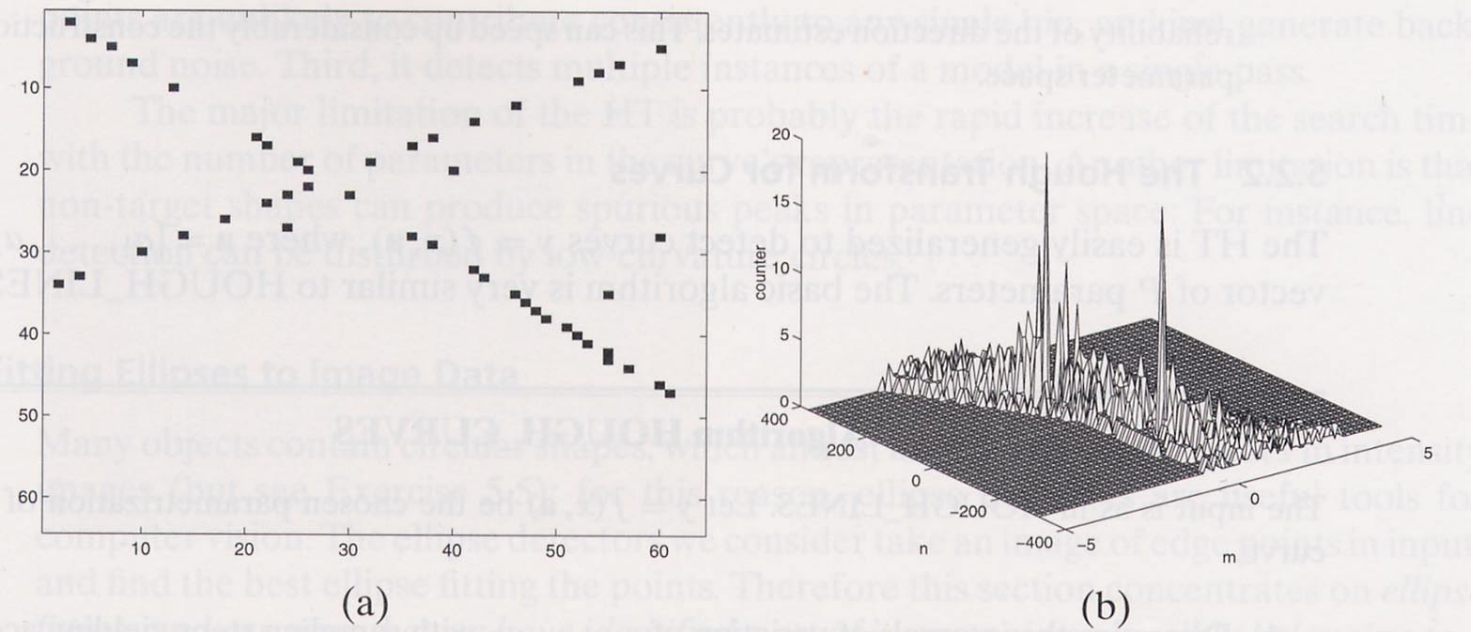


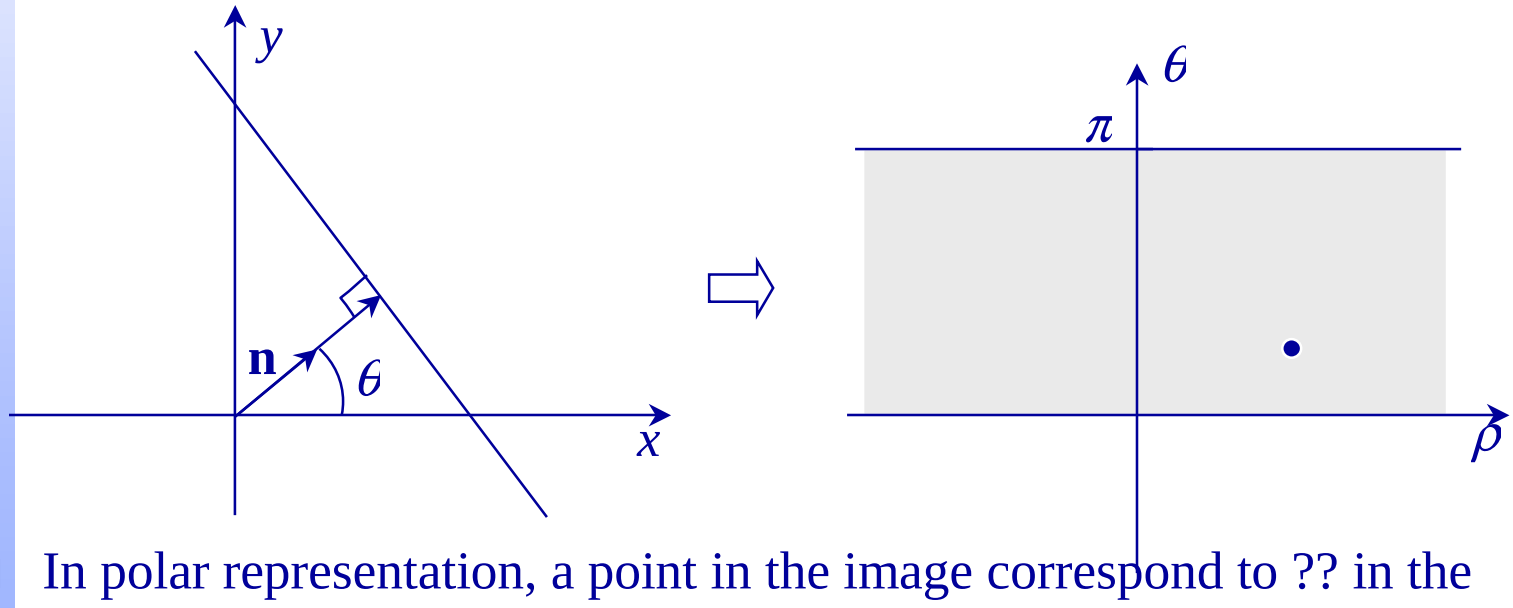
Figure 5.2 (a) An image containing two lines, sampled irregularly, and several random points. (b) Plot of the counters in the corresponding parameter space (how many points contribute to each cell (m, n)). Notice that the main peaks are obvious, but there are many secondary peaks.

Department of Computer Engineering
University of California at Santa Cruz

Detecting lines using Hough transform

- *m* can be huge for near vertical lines, and there is no representation for a vertical line. Lines are not "evenly" distributed in the parameter space
- Solution – using polar representation for lines, more "evenly" distributed

$$\rho = p \cdot \mathbf{n} = x\cos\theta + y\sin\theta$$



- In polar representation, a point in the image correspond to ?? in the parameter space (Homework!)

Department of Computer Engineering
University of California at Santa Cruz

Detecting lines using Hough transform

- Algorithm HOUGH_LINES
  - The input image *E is* $M \times N$ binary array with edge pixels marked with ones and other pixels marked as zeroes. Let ρ_d, θ_d be the arrays containing the discretized intervals of the parameter space $\rho \in [0, \sqrt{N^2 + M^2}], \theta \in [0, \pi]$
  - Discretize the parameter spaces of ρ and θ using sampling steps $\delta\rho, \delta\theta$, yielding acceptable and manageable resolution of R,T in the parameter space
  - Let *A(R,T)* be the counter array, initialized as zeroes
  - For each pixel *E(i,j)*=1, and for $h = 1,...T$
    - Let $\rho = i \sin\theta_d(h) + j \cos\theta_d(h)$
    - Find index *k* so that $\rho_d(k)$ is closest to ρ
    - Increment *A(k,h)* by one
  - Find all local maxima (k_p, h_p) such that $(k_p, h_p) > \tau$, where τ is a user defined threshold
  - The output is a set of lines described by $(\rho_d(k_p), \theta_d(h_p))$

Department of Computer Engineering
University of California at Santa Cruz

Detecting curves using Hough transform

- Algorithm HOUGH_CURVES
  - Let $f(x, y, \mathbf{a}) = 0$ be the parametric form of the curves
  - Discretize the parameters $a_1, ..., a_p$ with sampling steps yielding acceptable and manageable resolution of $s_1, ..., s_p$
  - Let $A(s_1, ..., s_p)$ be the counter array, initialized as zeroes
  - For each pixel $E(i,j)=1$, increment all counters such that $f(i, j, \mathbf{a}) = 0$
  - Find all local maxima $\mathbf{a}_m$ such that $\mathbf{a}_m > \tau$, where τ is a user defined threshold
  - The output is a set of curves described by $\mathbf{a}_m$

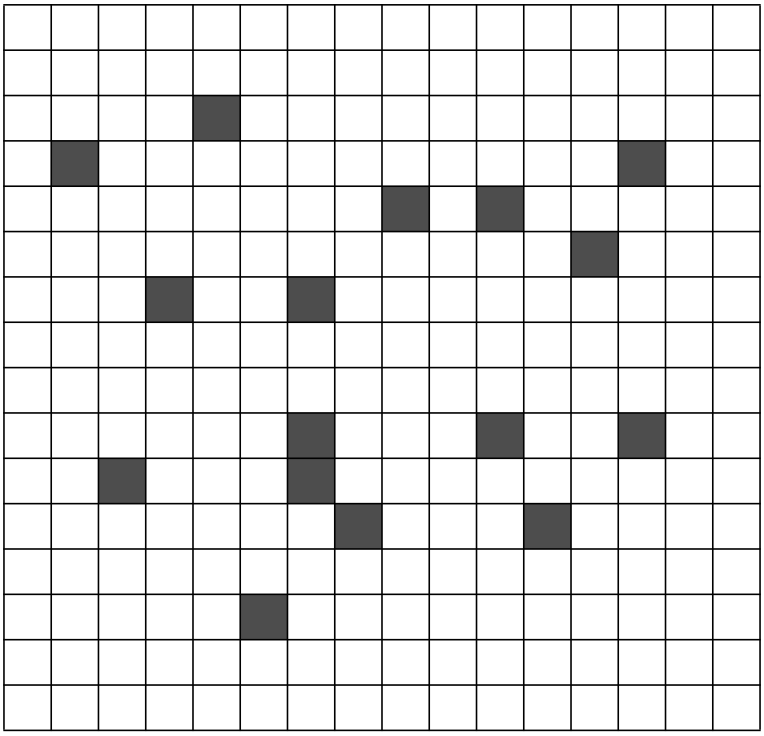
Summary

- Can detect lines even when they are partially occluded
- Hough Transform is a “voting” algorithm
- Since each point is handled independently, parallel implementations are possible
- It becomes difficult when the dimension of the parameter space is large

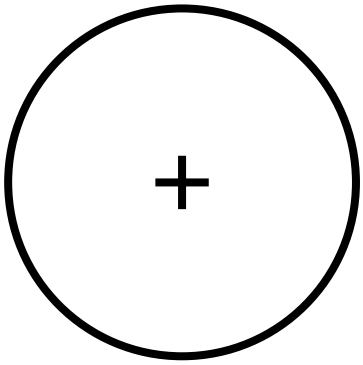
The Hough transform for fixed-size circles

Suppose we want to find circles of known radius in an image.

Assume the image is pre-processed (e.g. with an edge or corner detector) to yield a binary image giving *feature locations* of some sort. The boundary of the circle is marked by some of these features.

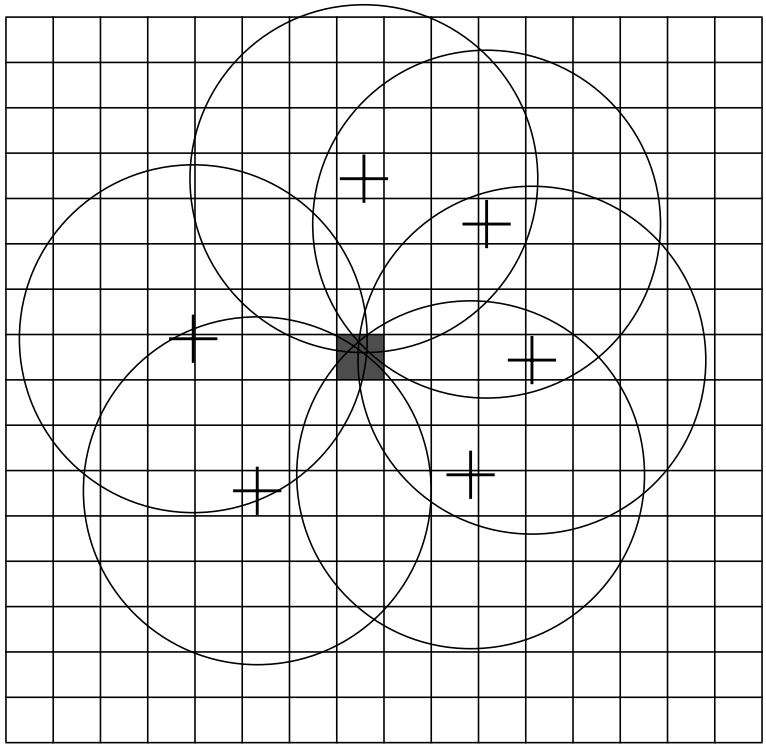


The “1” pixels showing feature positions are dark above. We want to find pixels forming a circle like the one on the right.

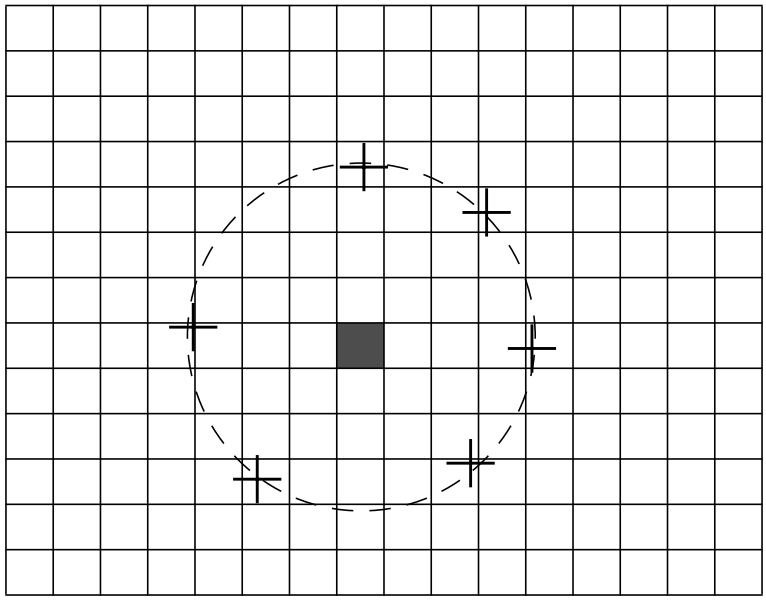


This amounts to identifying the position of the circle’s centre when the circle lines up with the dark pixels.

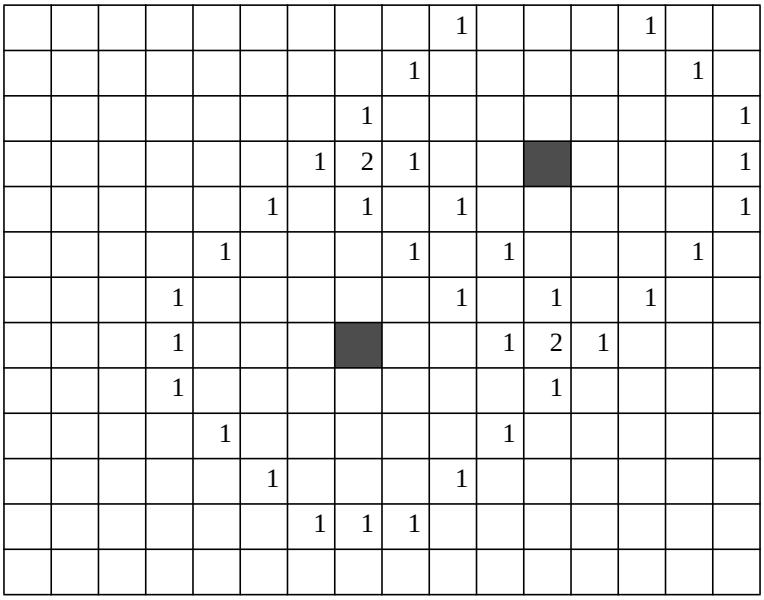
A solitary feature could lie on the boundary of many possible circles.



But the centres of these all lie themselves on a circle round the feature:



So a given feature can *vote* for the centres of all the circles that it might lie on, by adding values to the elements of an *accumulator array.* Each feature increments the places where a centre might lie — i.e. the cells on a circle around itself.



Zero cells shown blank.

Filled cells are feature positions.

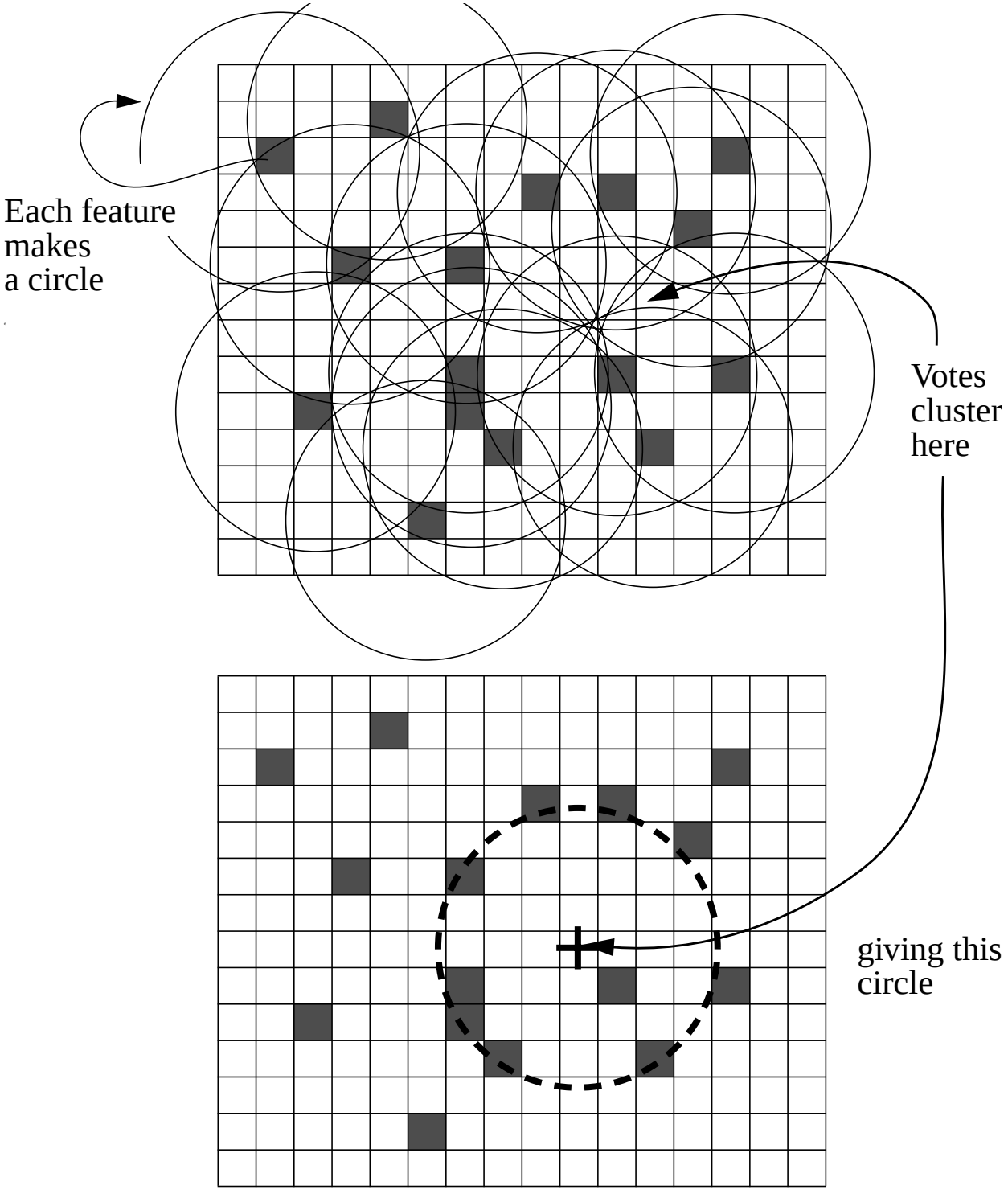
The circles do not look smooth because of *quantisation* — the need to split the plane up into discrete cells. The cells are often called *bins*.

However, the two cells that get 2 votes each are the candidates for the centres of circles that pass through the two features.

This is repeated for each feature.

Note that this is different from convolution, where the basic operation is carried out for every element of the output array. Here, the main loop is over features in the *input*. This can be very fast. Parallel implementation is also possible.

When there are many features, the votes build up for the centres of circles.

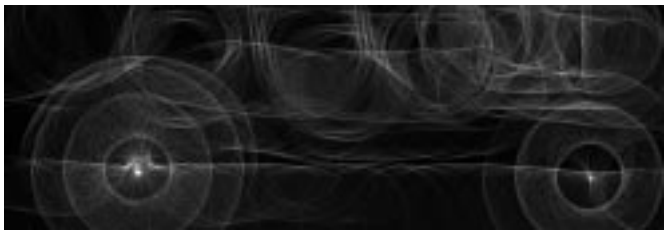


Thus the Hough transform *accumulates evidence* for a given geometrical structure.

In practice, assuming the radius of the wheel is 20 pixels:

Grey-level image

Canny edges
(Edge orientation ignored.)



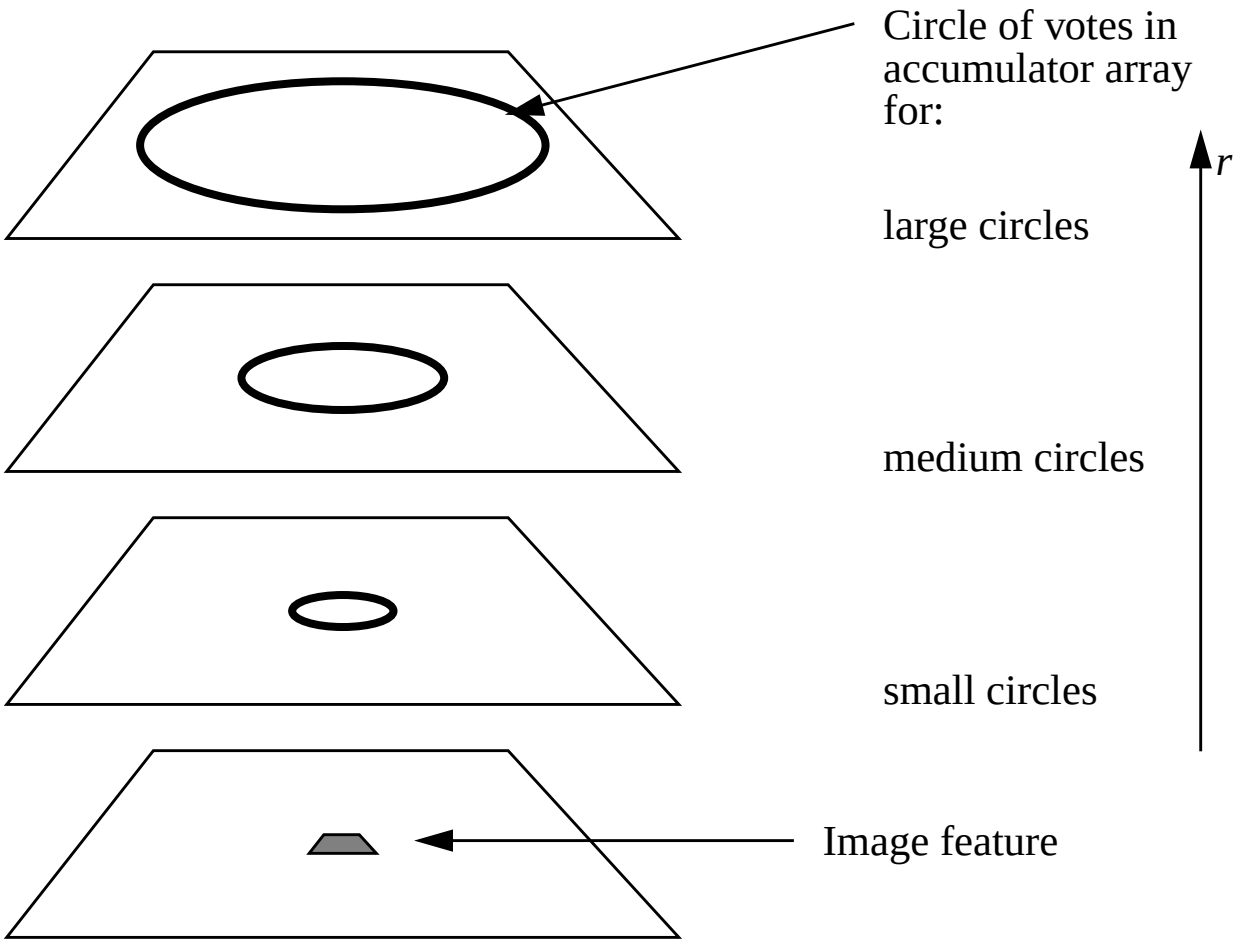
Accumulator array — the transform itself

Most prominent circle

The Hough transform for circles of unknown size

A circle of unknown radius has 3 *parameters*: the coordinates of its centre, x_c and $y_{c,}$ and its radius, r.

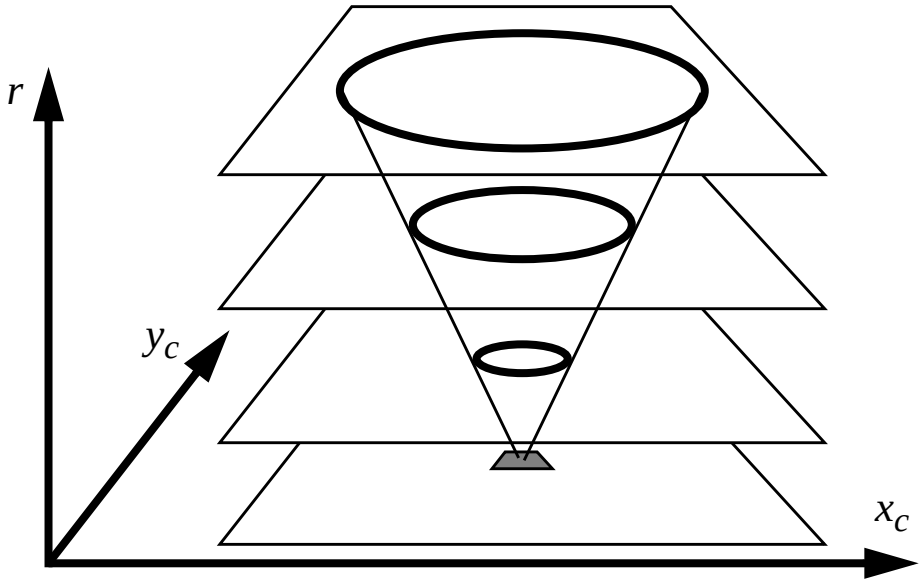
One way to find them is to do the previous HT for circles with many different radii, and see which one produces the biggest peak in the accumulator array.



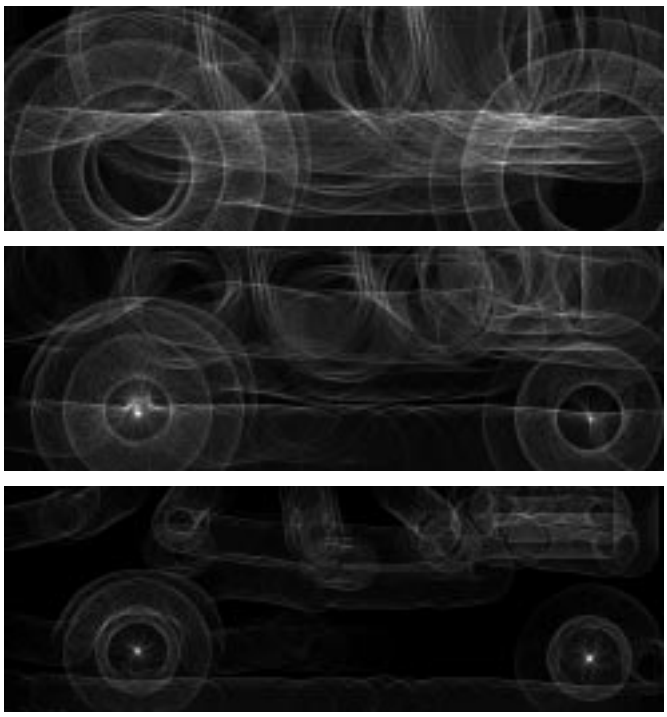
In practice, might use many different values of r.

Rather than doing each one as a separate HT, do all the r's for each feature together.

We can think of the stacked-up 2-D transforms forming a 3-D *parameter space*, in which each image feature votes on a cone.



Accumulator arrays for r = 8, 20 and 40 applied to the car image:



peak at r = 20

peak at r = 8

The Hough transform for ellipses

An ellipses has two additional parameters: the *aspect ratio* which specifies how flattened it is, and the *orientation* of its major axis.

If the size and shape of the ellipse is known, the procedure is much as for the simple circle-finding HT.

If all 5 parameters have to be found, the HT needs a 5-dimensional parameter space — i.e. a 5-D accumulator array.

This is computationally expensive. Various tricks are used to reduce the cost. For example the hierarchical HT starts with *coarse quantisation* (or big bins) and then does a finer division on bins with enough votes. (A kind of scale-space approach.)

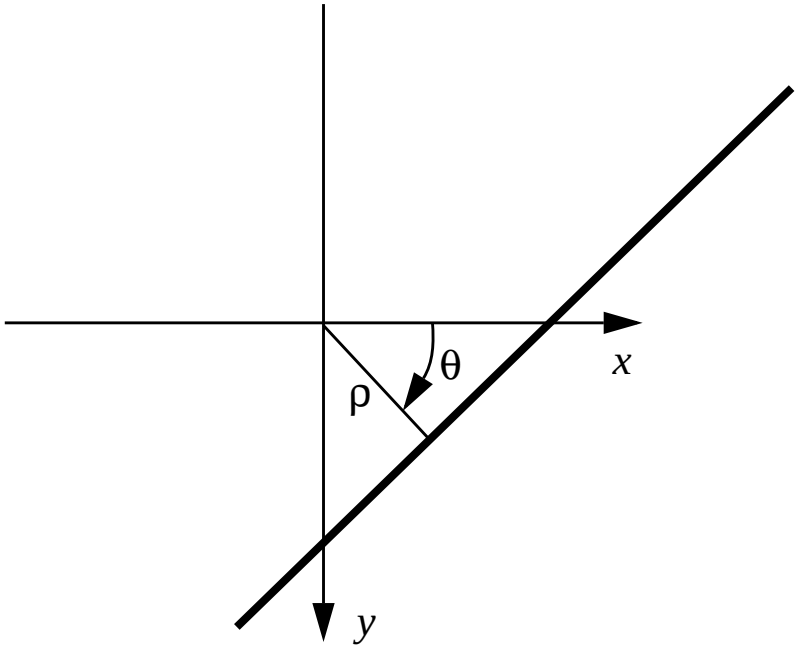
Smoothing of the accumulator array, and recognition of special patterns in it, is also done.

The HT is robust. It is not badly affected if the shape is broken up, or partly hidden as above.

The Hough transform for straight lines

This is the original HT.

A straight line has two parameters (we are not at present interested in its end points). There are various possibilities for these — mathematicians might use slope and gradient. A better choice for our purpose is this:

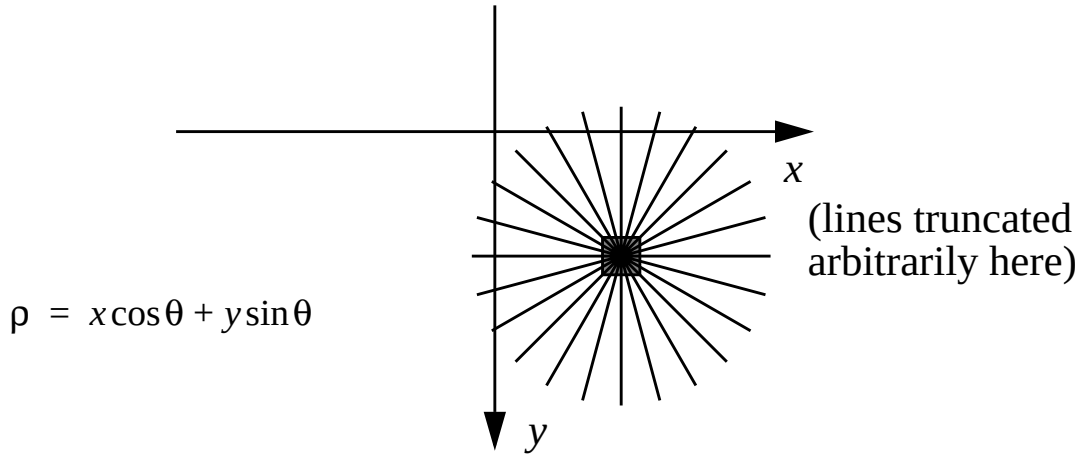


where θ (theta) is the angle from the *x* axis to the perpendicular to the line, and ρ (rho) is the distance of the line from the origin, measured along its perpendicular.

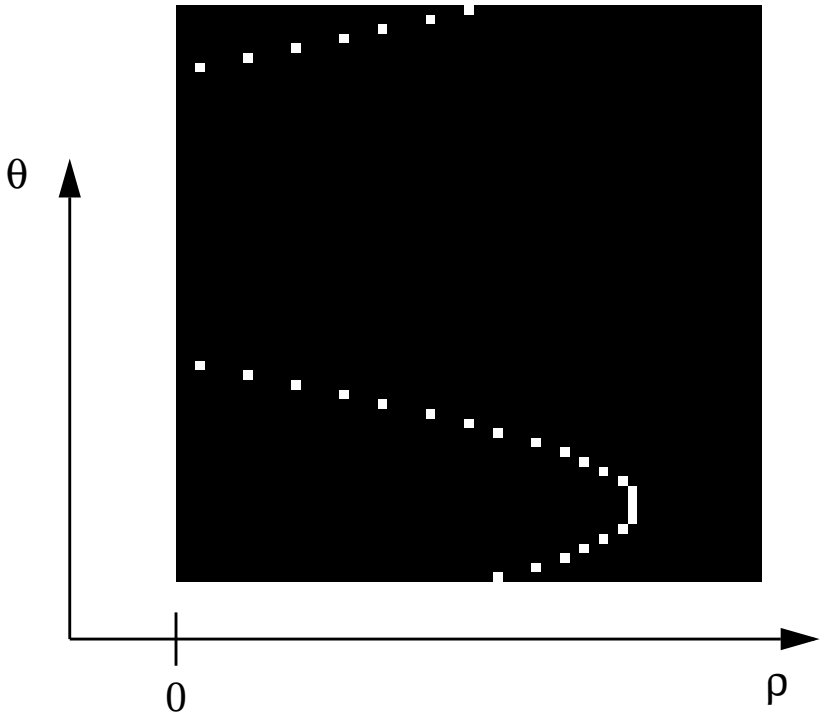
Any line in the plane can be represented by a pair of values for ρ and θ.

Thus each point in the (ρ, θ) parameter space corresponds to a single line, and *vice versa.*

A given point feature lies on many different lines:



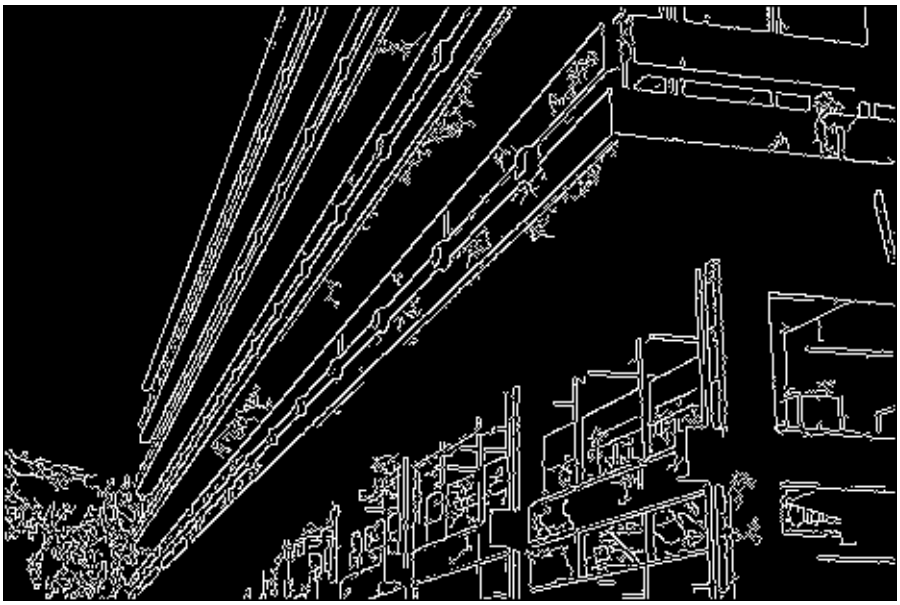
and so has many votes in parameter space:



Features lying on the same line accumulate votes in one bin.

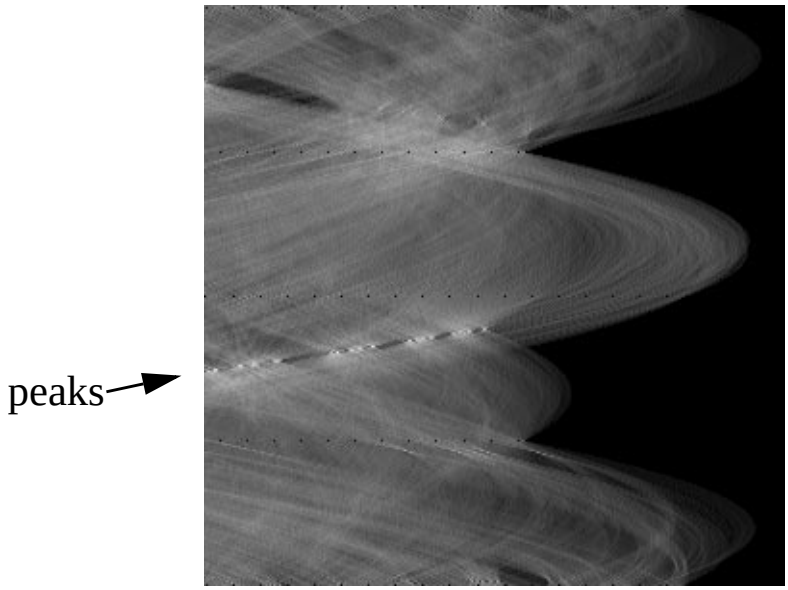
A scene with some prominent straight lines

and the edges found by the Canny edge detector.



Each white pixel in the figure above votes for 300 possible lines through it ...

... to produce the accumulator array



from which the 18 largest peaks give the strongest lines:

General Hough transforms

The HT can be applied to any *parametrised* shape.

These include

- shapes described by equations (lines, ellipses, circles etc.)
- shapes described by tables, with orientation, position and maybe scale as the parameters to be found — this is the *Generalised Hough Transform*.
- shapes that deform depending on their position (e.g. for finding iris position in eye images).

The HT can be used to detect *symmetries* — e.g. axis of brain images, shear symmetry caused by perspective projection.

Significant problems include

- how to determine the correct quantisation
- how to deal with large numbers of parameters
- how to decide what shapes to detect.