Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 16

Object Recognition

Hough transform for parametric curves: lines, circles, and ellipses

The Hough Transform

(pronounced Huff, to rhyme with Tough)

The Hough Transform

- However the Hough transform has been generalised so that it is capable of detecting arbitrary curved shapes
  - Very tolerant of gaps in the actual object boundaries or curves
  - Relatively unaffected by noise

The Hough Transform

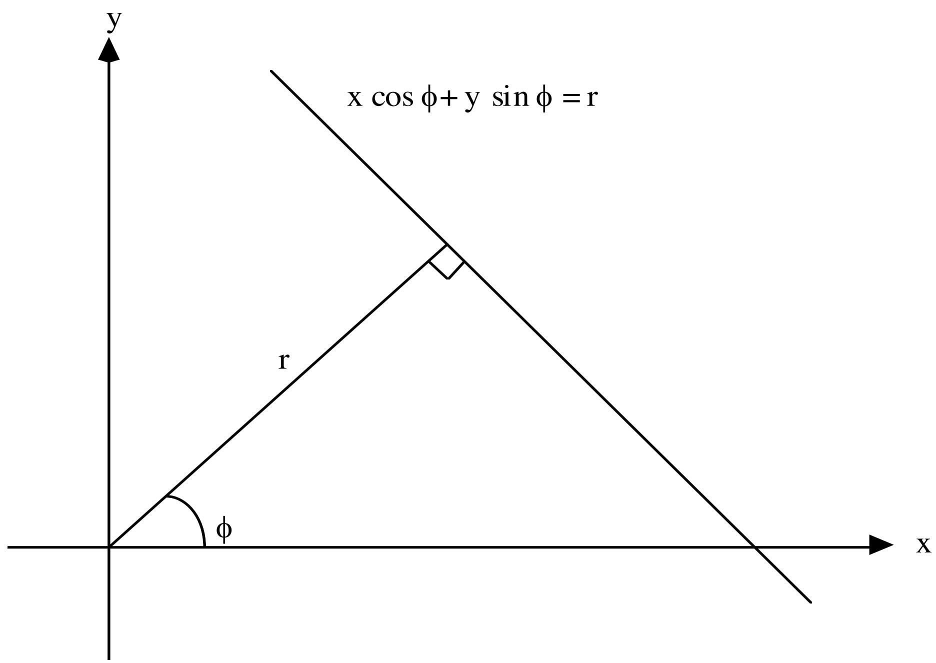
- The Hough Transform is a technique which is used to isolate curves of a given shape in an image
- The classical Hough transform requires that the curve be specified in some parametric form
  - Lines
  - Circles
  - Ellipses
- The number of parameters required to specify the curve determines the dimensionality of the Hough space

Hough Transform for Line Detection

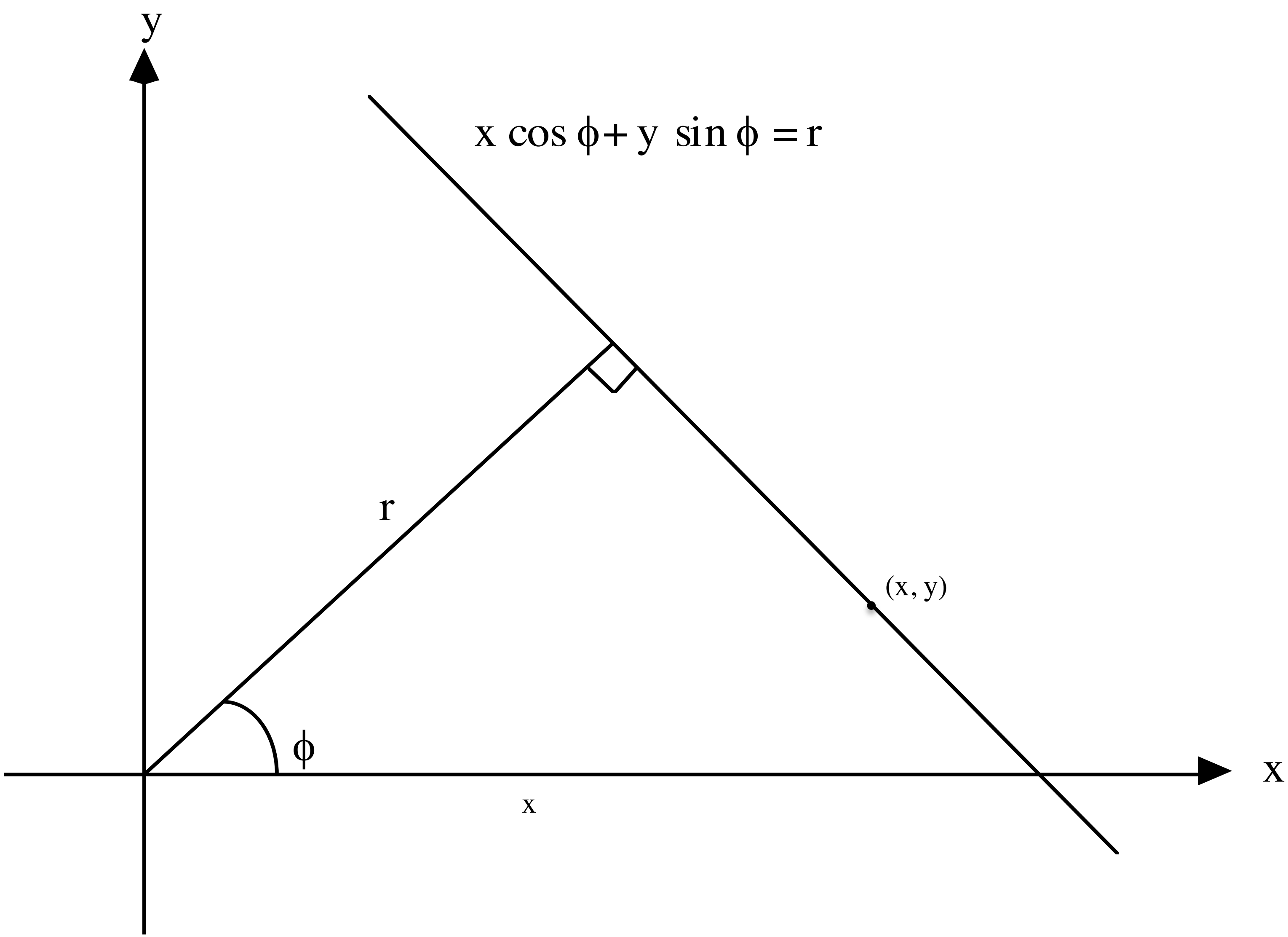
- We wish to detect a set of points lying on a straight line

- The equation of a straight line is given in parametric form by

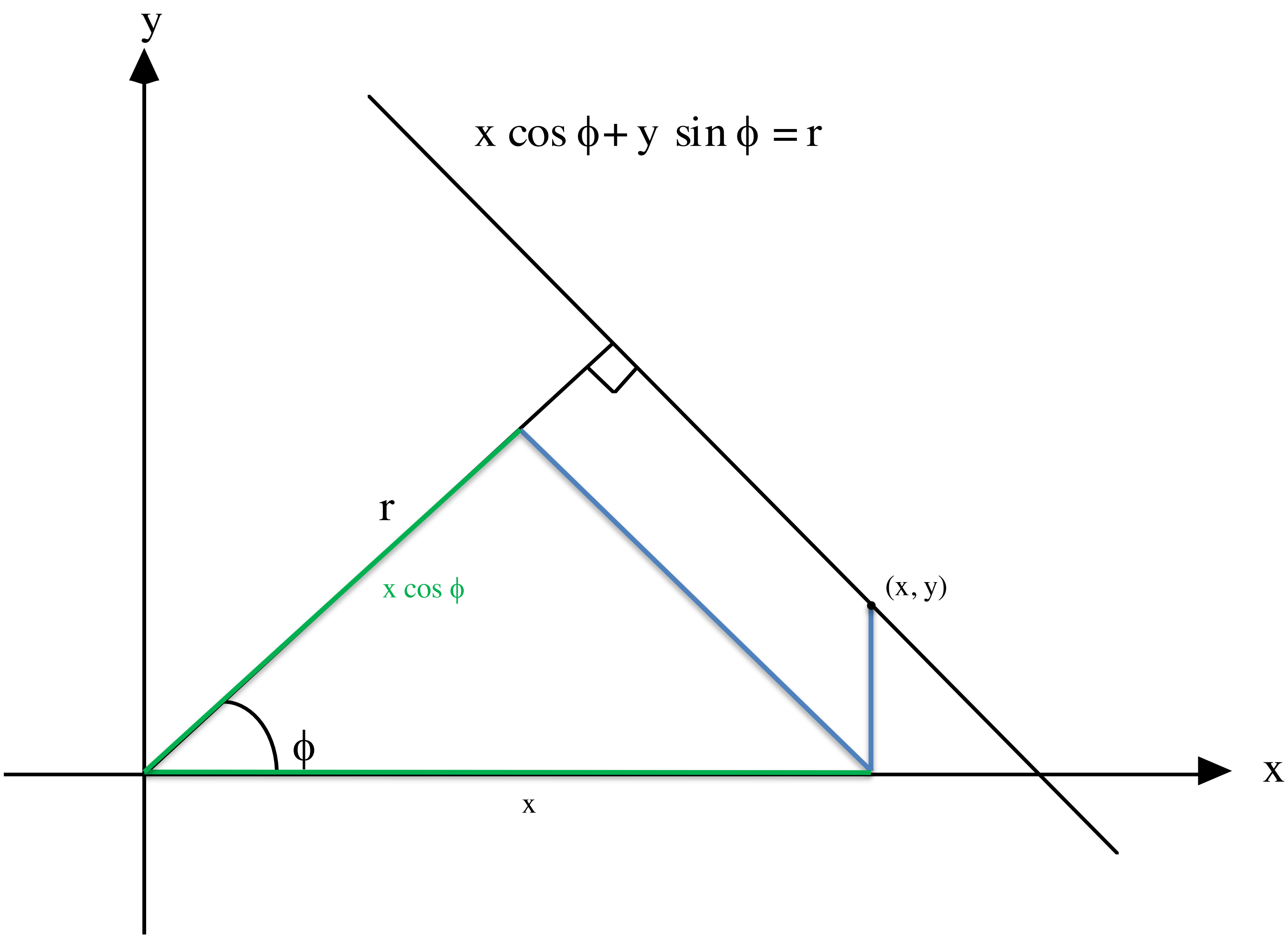
$$x\cos\phi + y\sin\phi = r$$



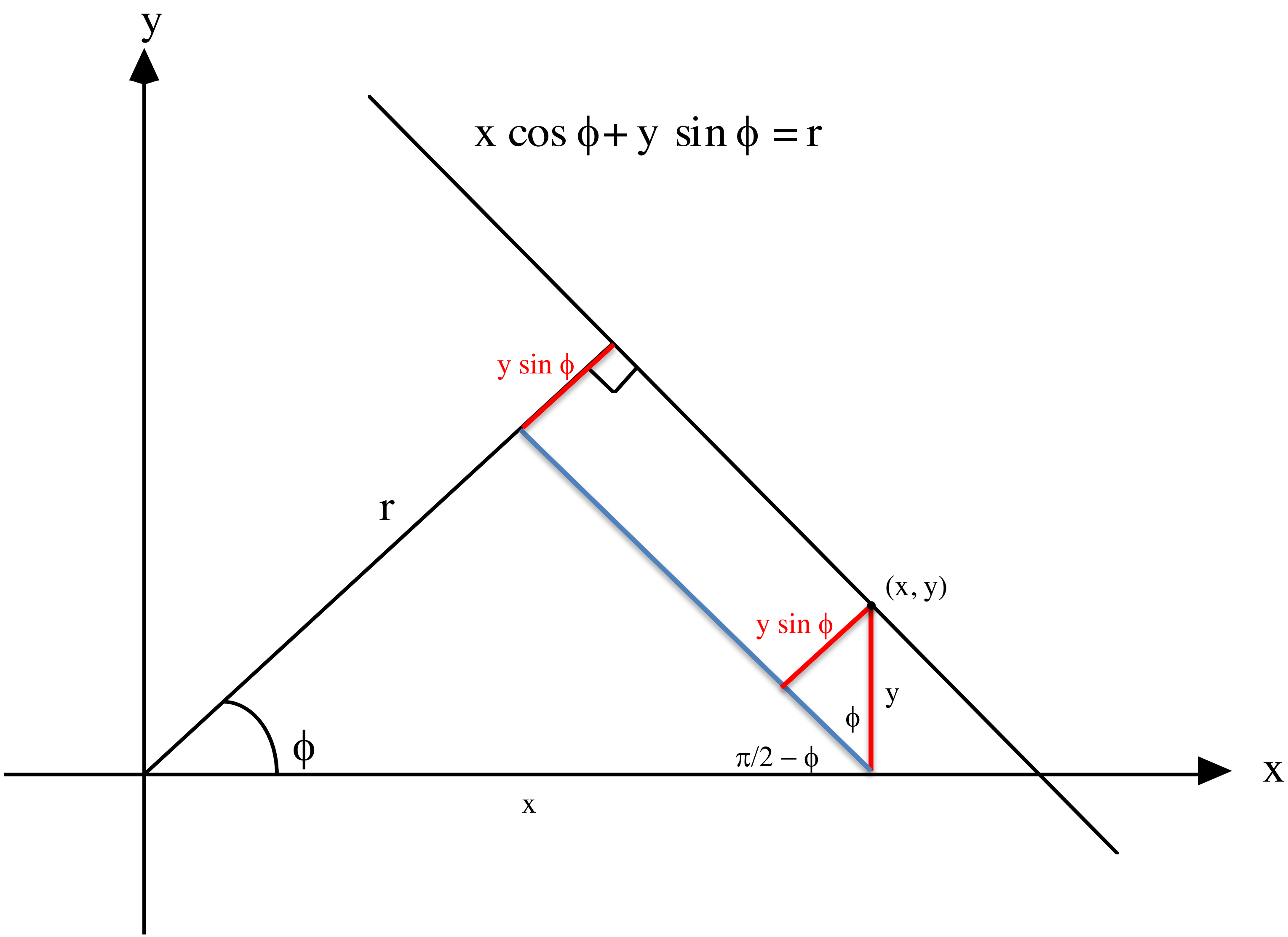
Hough Transform for Line Detection



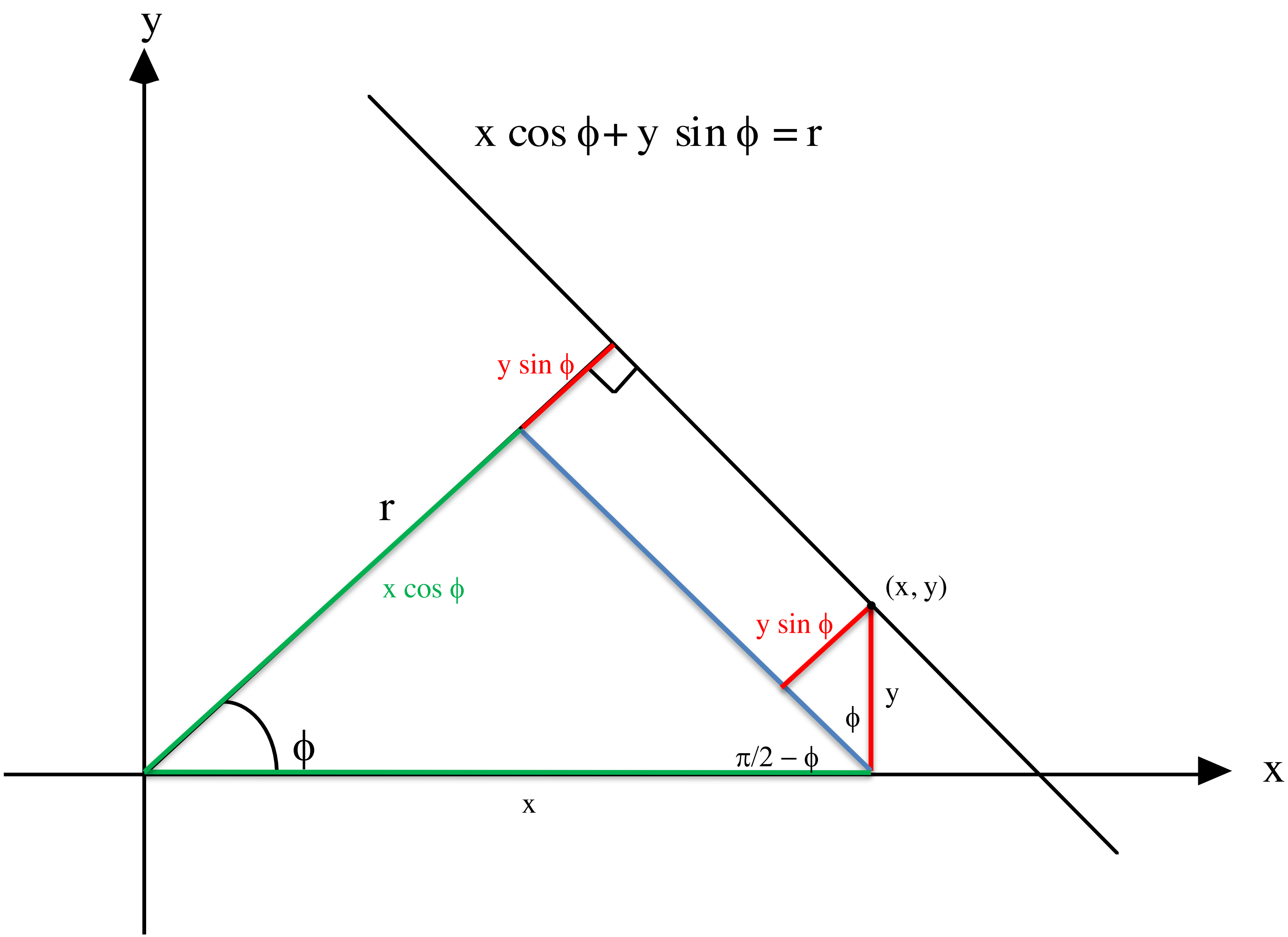
Hough Transform for Line Detection



Hough Transform for Line Detection



Hough Transform for Line Detection



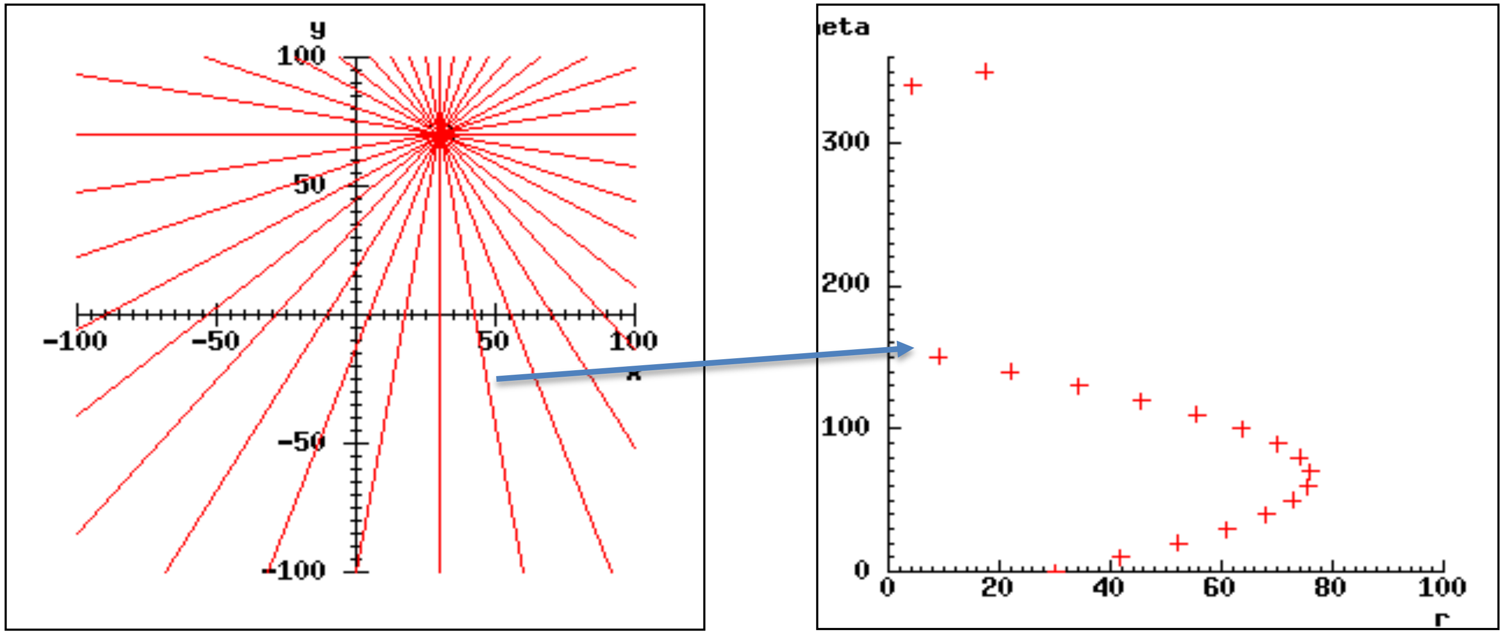
Hough Transform for Line Detection

- If we have a point (x_i, y_i) on this line then

$$x_i \cos\phi + y_i \sin\phi = r$$

- For a given line, r and ϕ are known and fixed

Hough Transform for Line Detection



One point (x_i, y_i) in image space generates a curve in (r, ϕ) Hough space

Source: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection

- Suppose, however, that we do not know what line exists (*i.e.* r and ϕ are unknown)
  but we do know the co-ordinates of the point(s) on the line

$$x_i \cos\phi + y_i \sin\phi = r$$

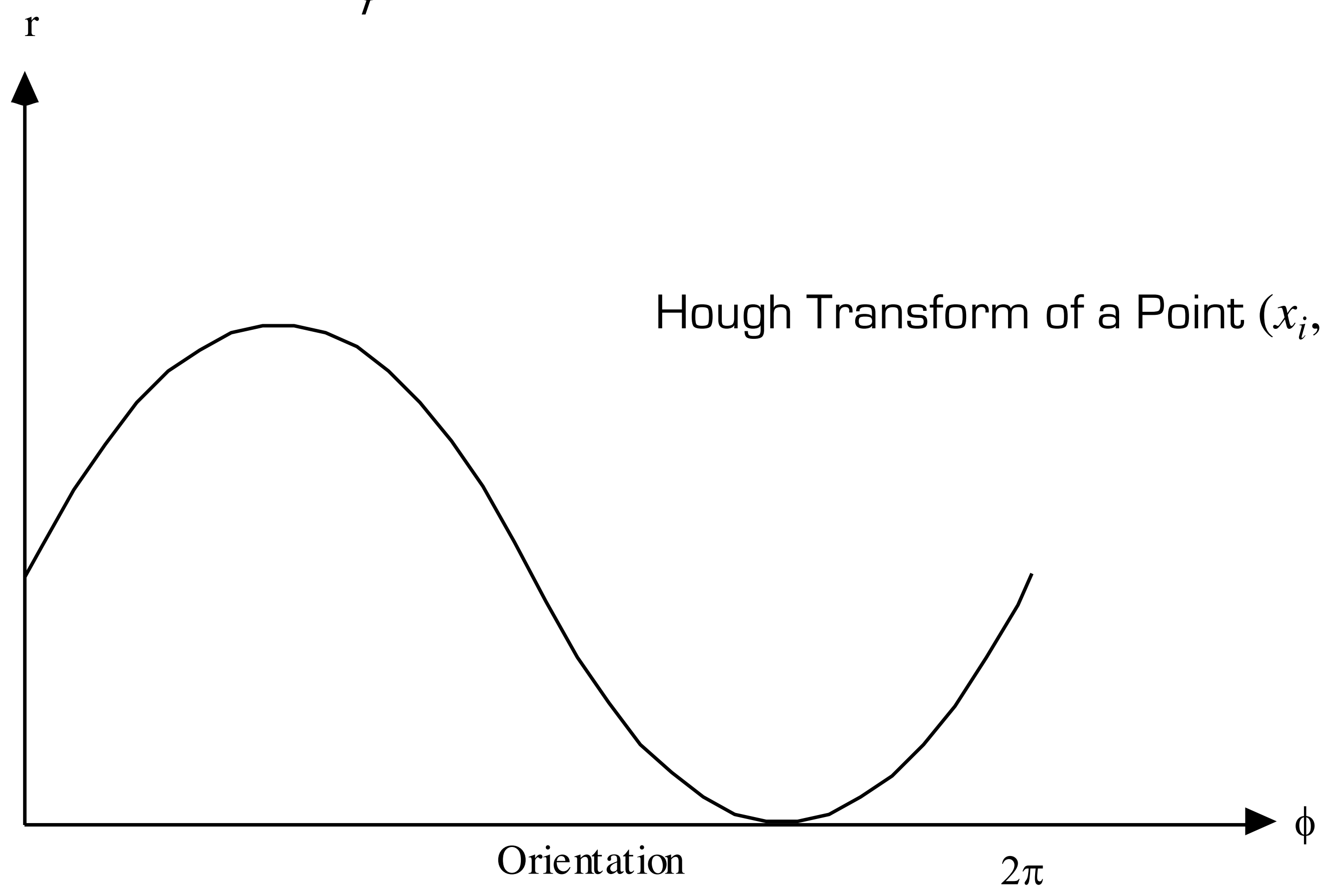
- We consider r and ϕ to be variables and x_i and y_i to be fixed
  In this case, the equation

$$x_i \cos\phi + y_i \sin\phi = r$$

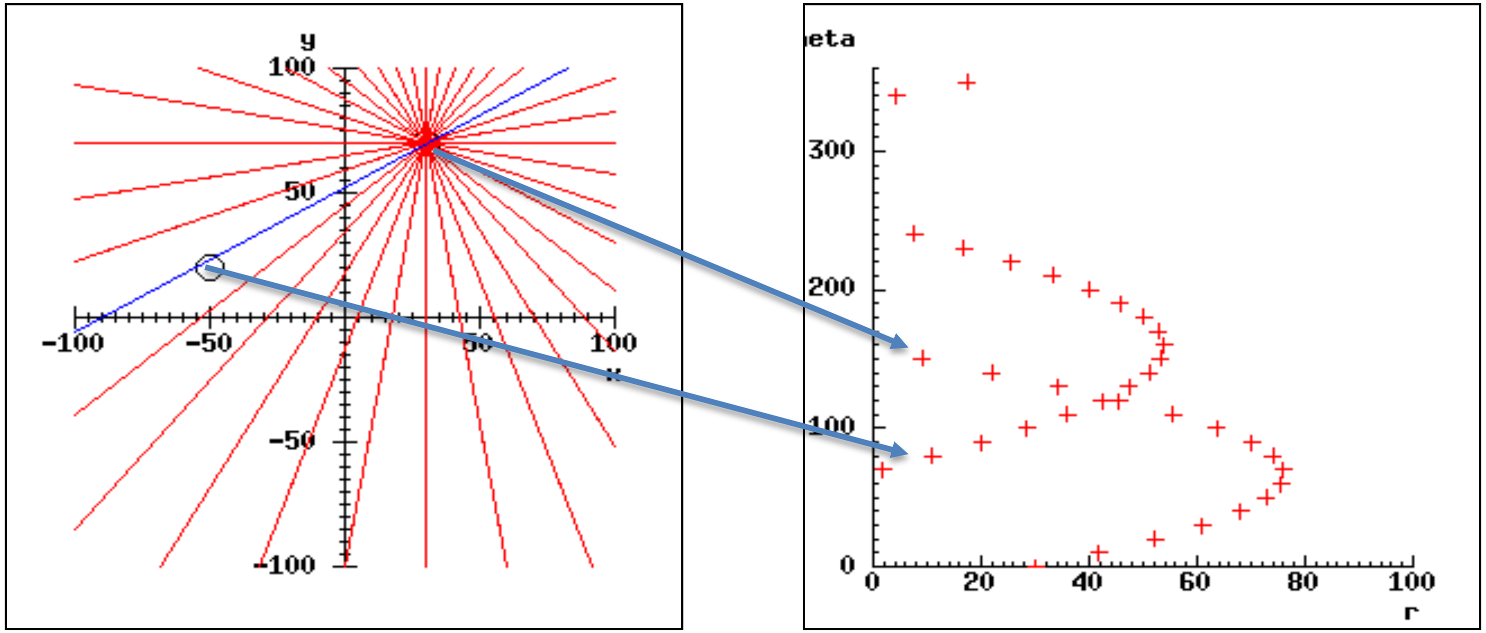
defines all the values of r and ϕ such that the line passes through the point (x_i, y_i)

Hough Transform for Line Detection

- If we plot these values of r and ϕ, for a given point (x_i, y_i), on a graph we get a sinusoidal curve in $(r - \phi)$ space, *i.e.* in a space where r and ϕ are the variables



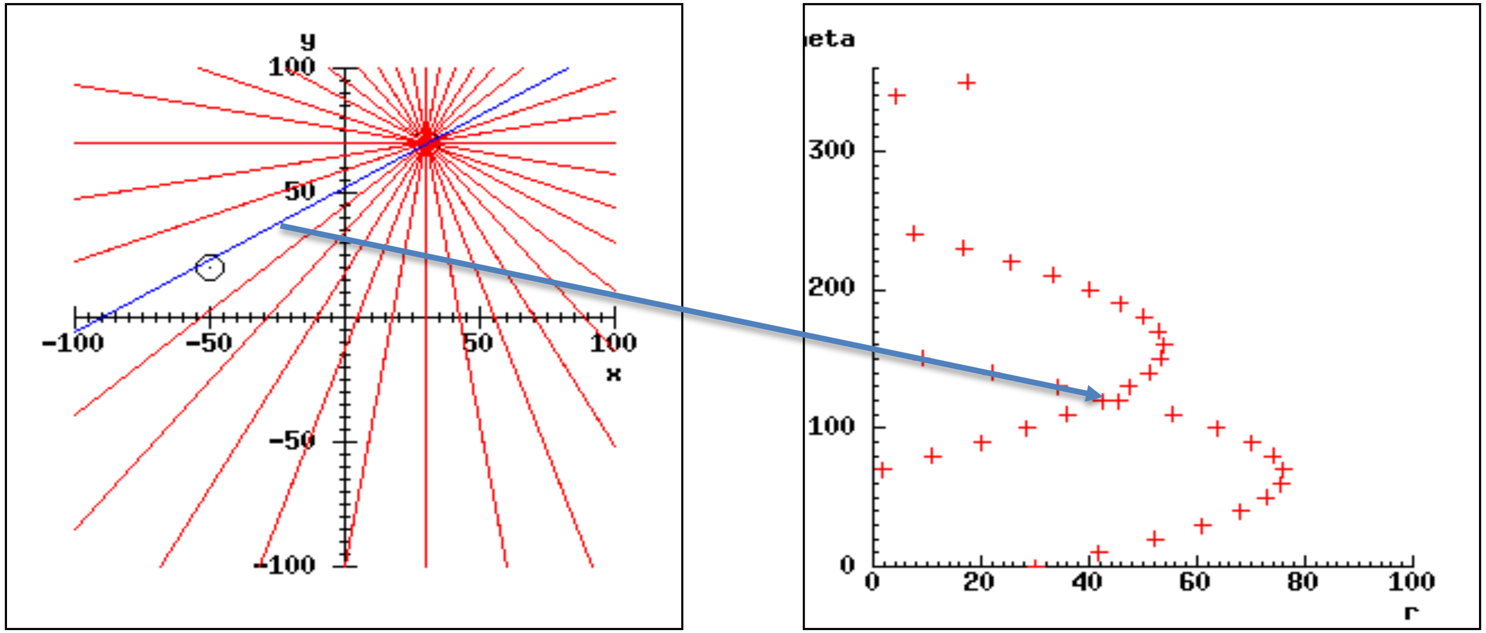
Hough Transform for Line Detection



Two points (x_i, y_i) in image space generate two curves in (r, ϕ) Hough space

Source: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection



A line of points (x_i, y_i) in image space generates a set of intersecting curves in Hough space
The point of intersection gives the values of (r, ϕ) for that particular line

Credit: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection

- The transformation between the image plane (x and y co-ordinates) and the parameter space (r and ϕ co-ordinates) is known as the Hough Transform
- The Hough transform of a point in the image plane is a sinusoidal curve in the Hough $(r - \phi)$ space

Hough Transform for Line Detection

- However, collinear points in the image plane will give rise to transform curves which all intersect in one point since they share common r_i and ϕ_i and they all belong to the line given by

$$x\cos\phi_i + y\sin\phi_i = r_i$$

- This, then, provides us with the means to detect collinear points, *i.e.* lines

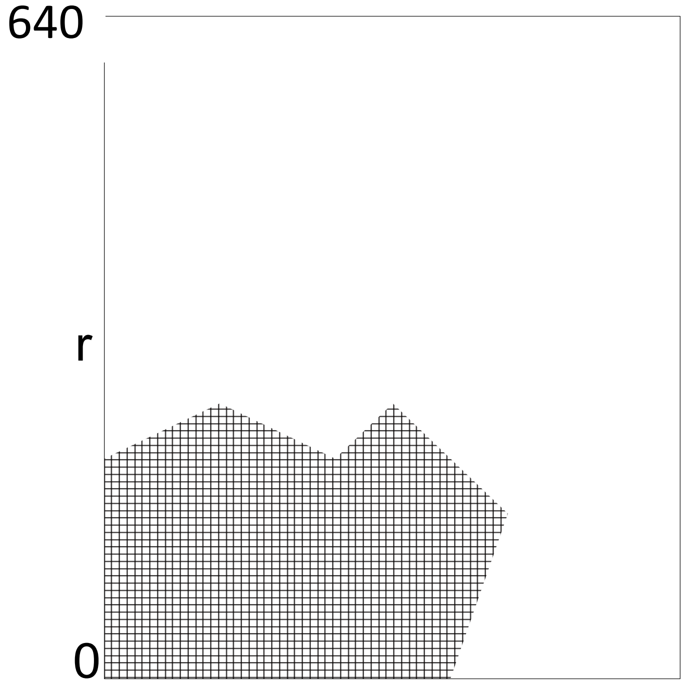
Hough Transform for Line Detection

- First of all, we must sample the Hough transform space, *i.e.* we require a discrete representation of $(r - \phi)$ space
  - Since ϕ varies between 0 and 2π radians, we need only decide on the required angular resolution to define the sampling
  - For example, a 6° resolution on the angle of the line might suffice, in which case we have a $360° / 6° = 60$ discrete values of ϕ
  - Similarly, we can limit r by deciding on the maximum distance from the origin (which is effectively going to be the maximum size of the image, e.g. 640 pixels)

Hough Transform for Line Detection

- Our representation of $(r - \phi)$ space is now simply a 2D array of size 640 * 60, each element corresponding to a particular value of r and ϕ

- This is called an **accumulator** since we are going to use it to collect or accumulate evidence of curves given by particular boundary points (x, y) in the image plane



Hough Transform for Line Detection

- For each boundary point (x_i, y_i) in the image, we increment all accumulator cells such that the cell co-ordinates (r, ϕ) satisfies the equation

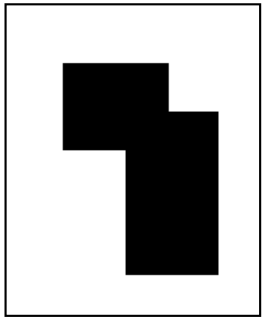
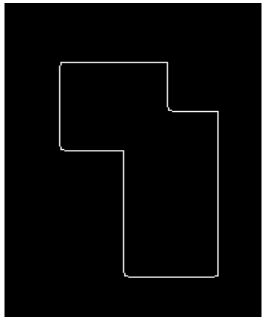
$$x_i \cos\phi + y_i \sin\phi = r$$

- When we have done this for all available (x_i, y_i) points, we scan the accumulator searching for cells which have a high count since these will correspond to lines for which there are many points in the image plane

Hough Transform for Line Detection

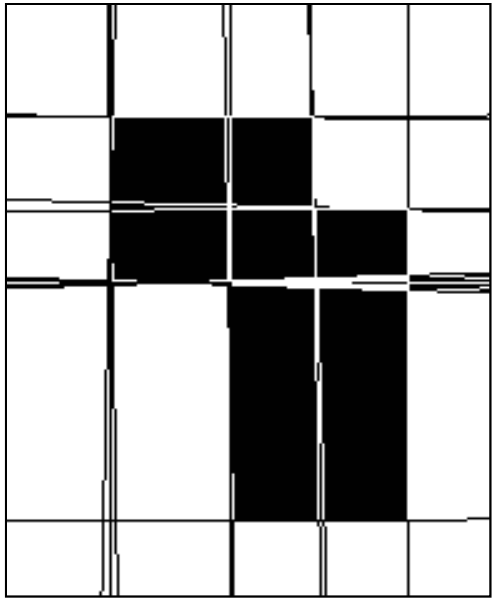
- Because there is likely to be some errors in the position of the x and y co-ordinates, giving rise to errors in r and ϕ, we search for clusters of points in the accumulator having high counts, rather than searching for isolated points

Hough Transform for Line Detection

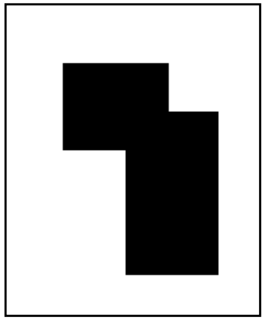
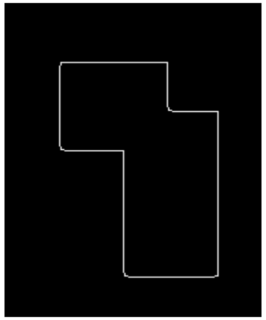
Accuracy depends on quantisation

Re-projected lines are infinite

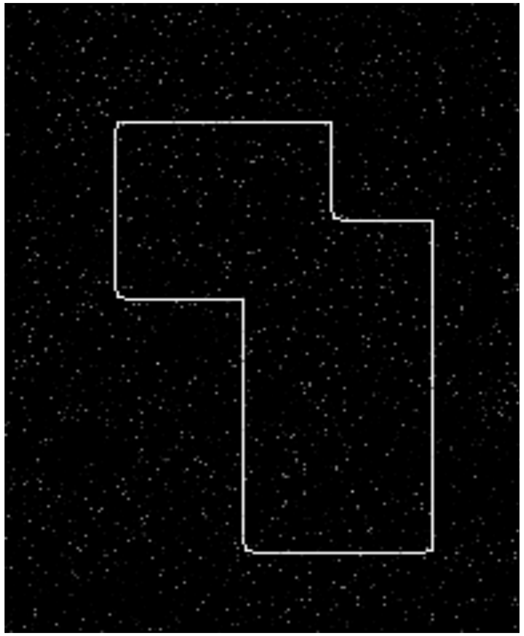
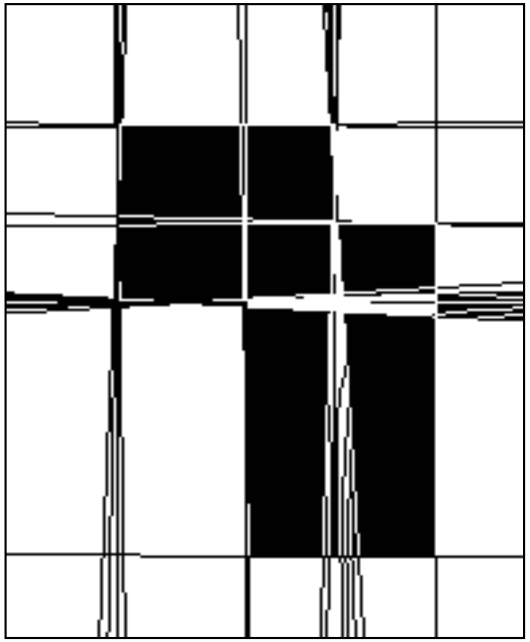
Credit: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection

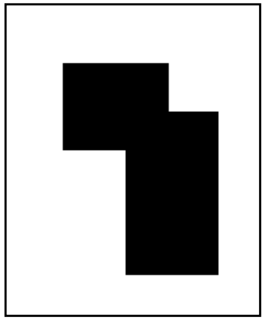
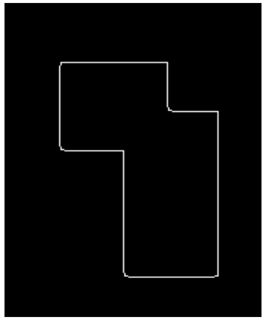
Accuracy depends on quantisation

Re-projected lines are infinite

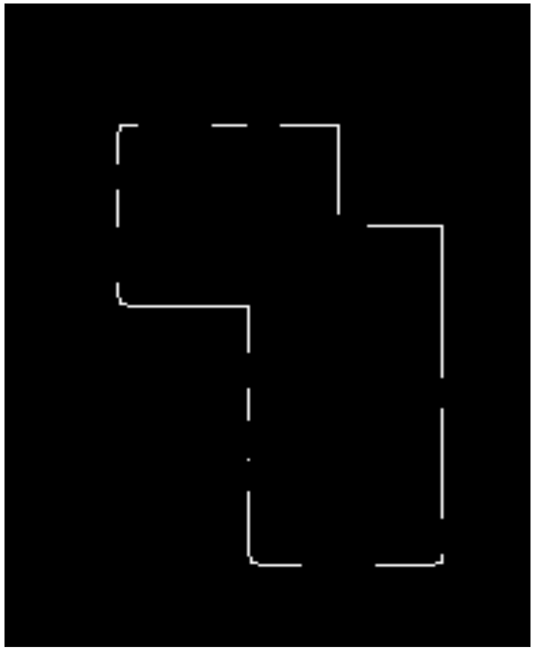
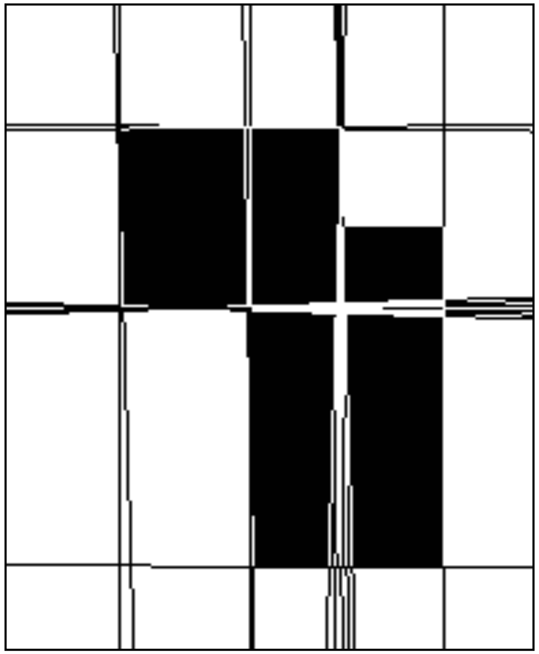
Credit: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection

Accuracy depends on quantisation

Re-projected lines are infinite

Credit: Markus Vincze, Technische Universität Wien

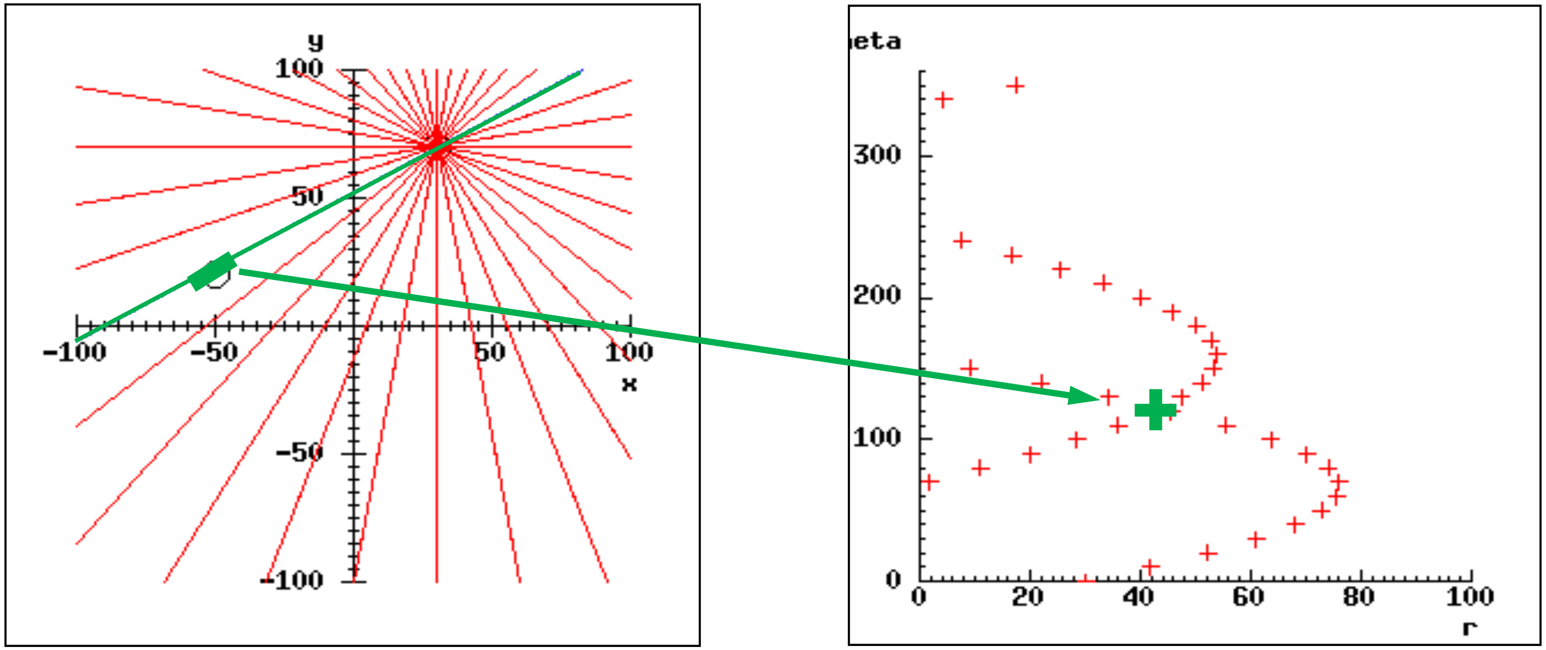
Hough Transform for Line Detection

- Edge detectors yield not only the position of the edge (x_i, y_i) but also its orientation θ, where $\theta = \phi + 90°$
- Some edge detectors, e.g. the Sobel operator, directly yield the gradient direction ϕ_i
- We can use this information to simplify the Hough Transform and, knowing x_i, y_i and ϕ_i, use

$$x_i \cos\phi_i + y_i \sin\phi_i = r$$

to compute r giving the co-ordinates of the appropriate accumulator cell to be incremented

Hough Transform for Line Detection



Credit: Markus Vincze, Technische Universität Wien

Hough Transform for Line Detection

```
/* Pseudo-code for Hough Transform: Line Detection */

Quantise the Hough transform space

Identify maximum and minimum values of r and φ
and the total number of r and φ values

Generate an accumulator array A(r, φ)
set all values to 0.
```

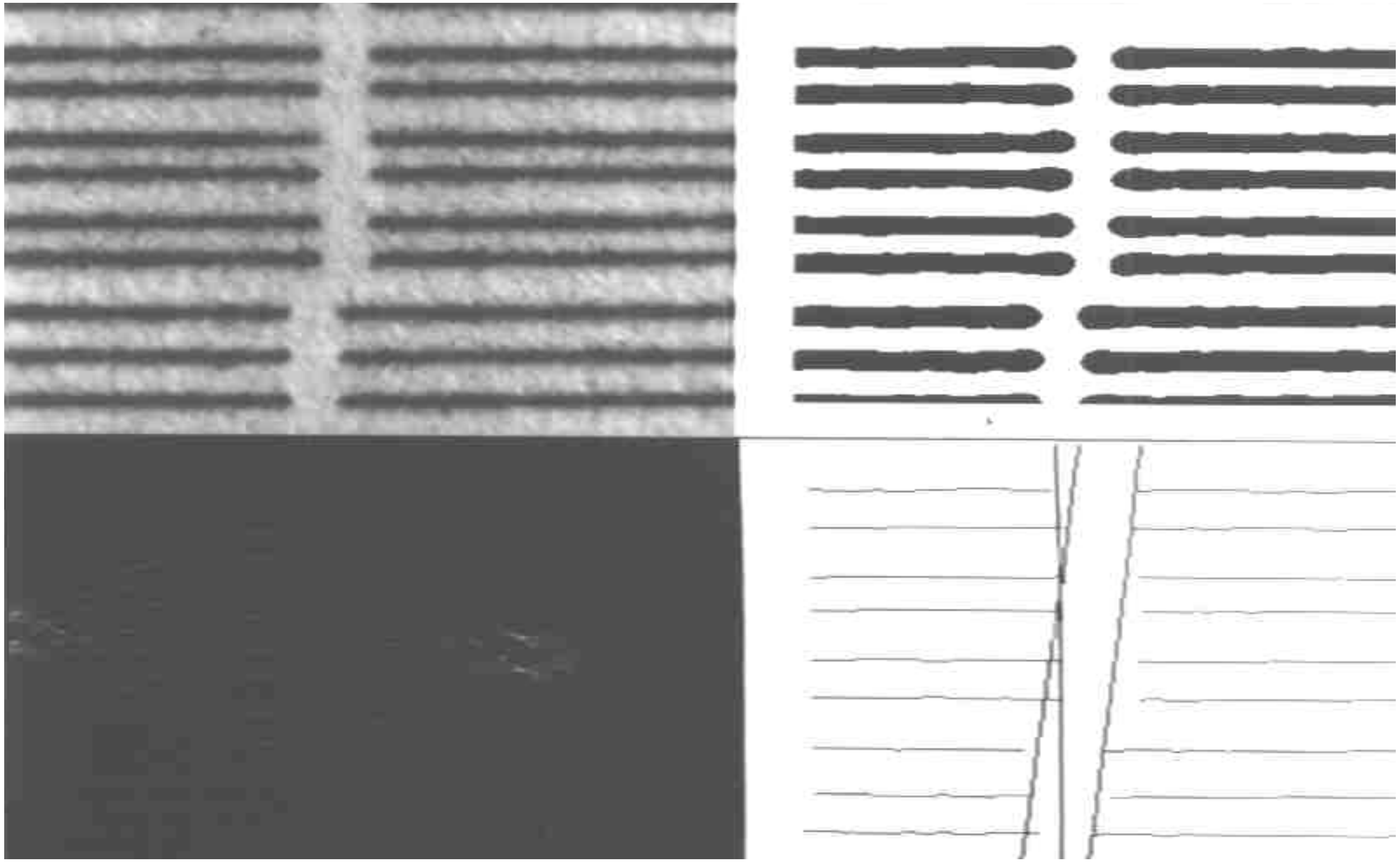
Hough Transform for Line Detection

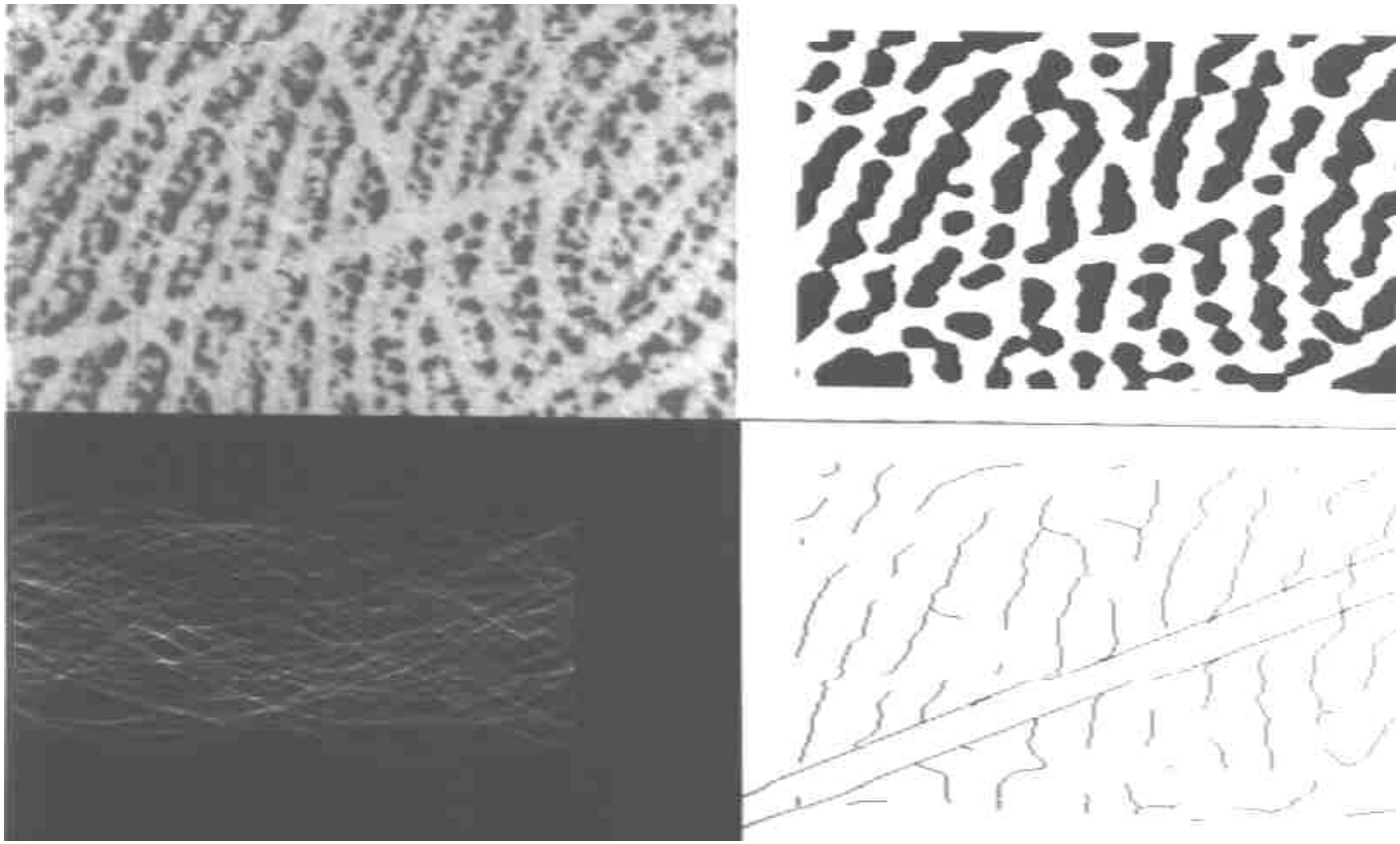
```
For all edge points (x_i, y_i) in the image
Do
   compute the normal direction φ
   i.e.(gradient direction) or (orientation - 90 degrees)†

   compute r from x_i cos φ + y_i sin φ = r

   increment A(r, φ)
```

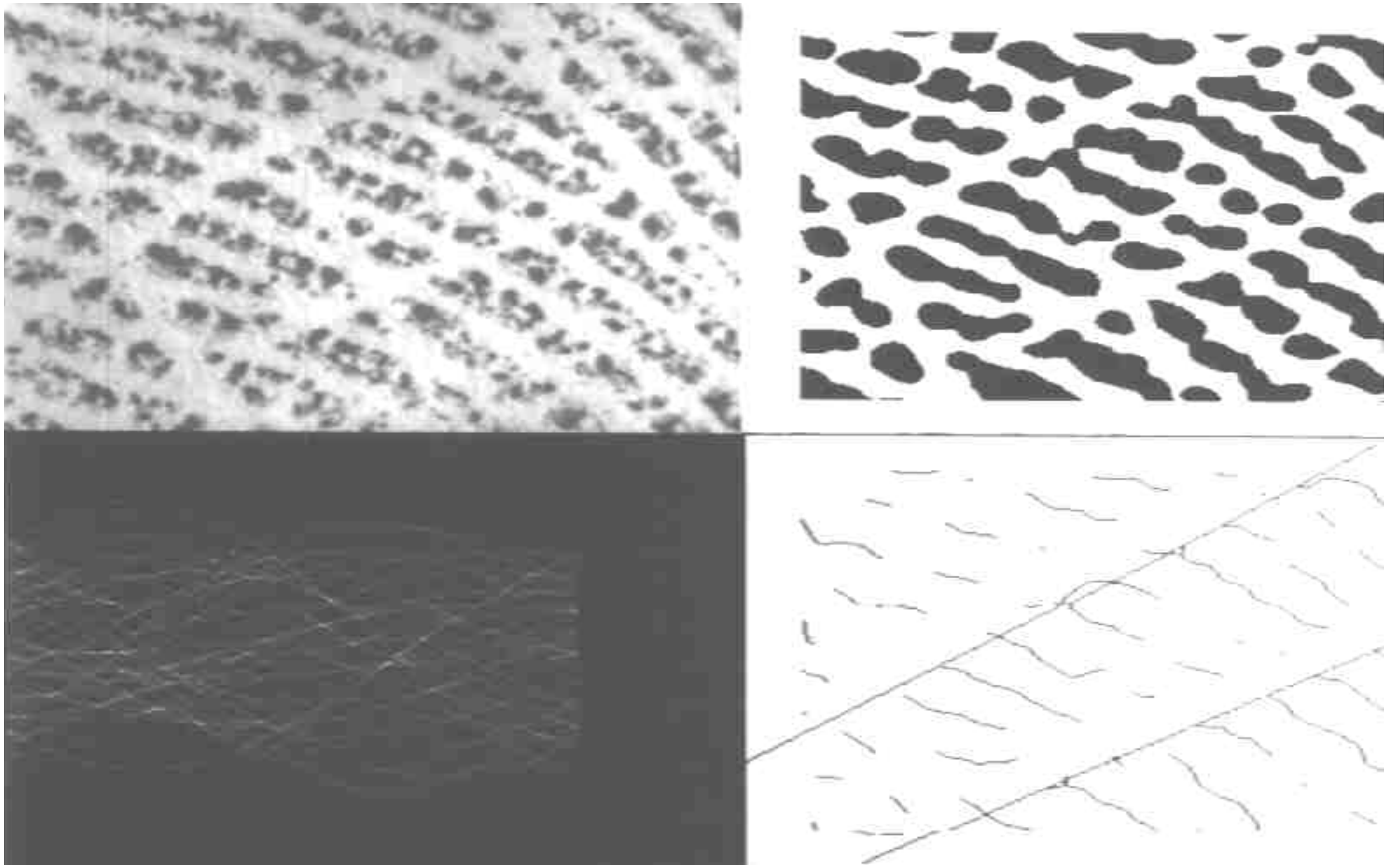
† Remember to normalise the result so that it lies in the interval 0 - 2π

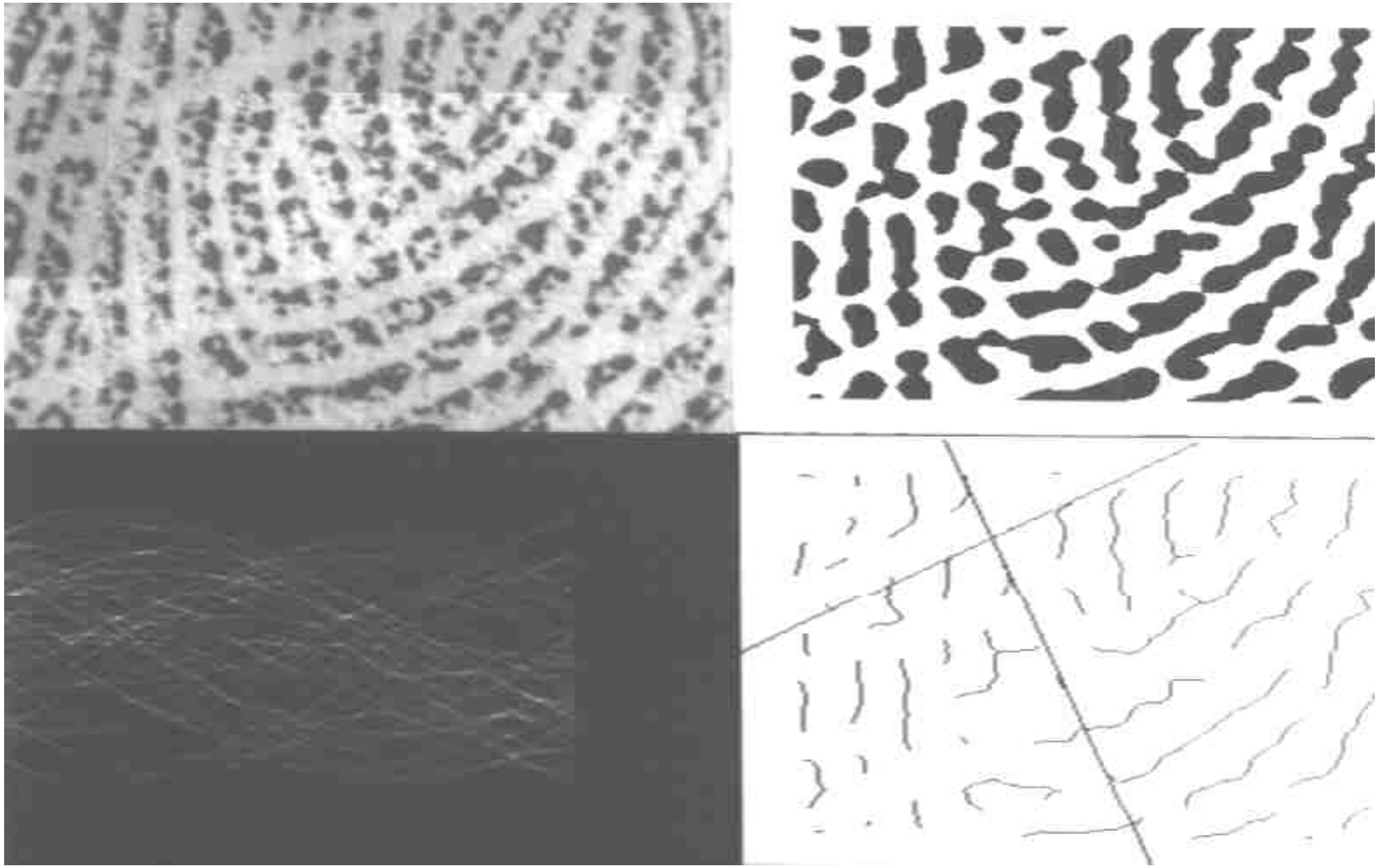
Hough Transform for Line Detection

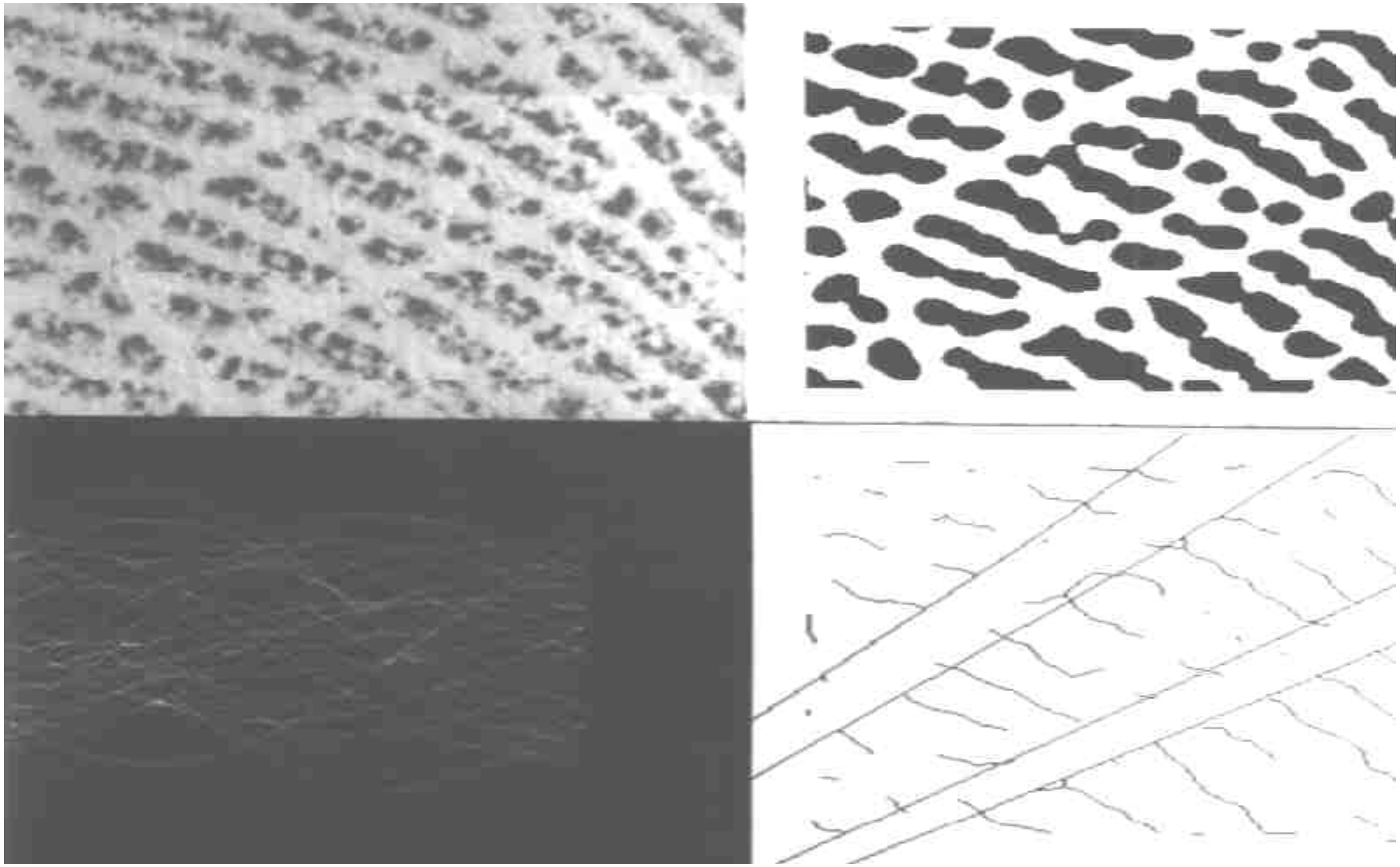
```
For all cells in the accumulator array
Do
   Search for maximum values
   The co-ordinates r and ϕ give the equation of
   the corresponding line in the image
```

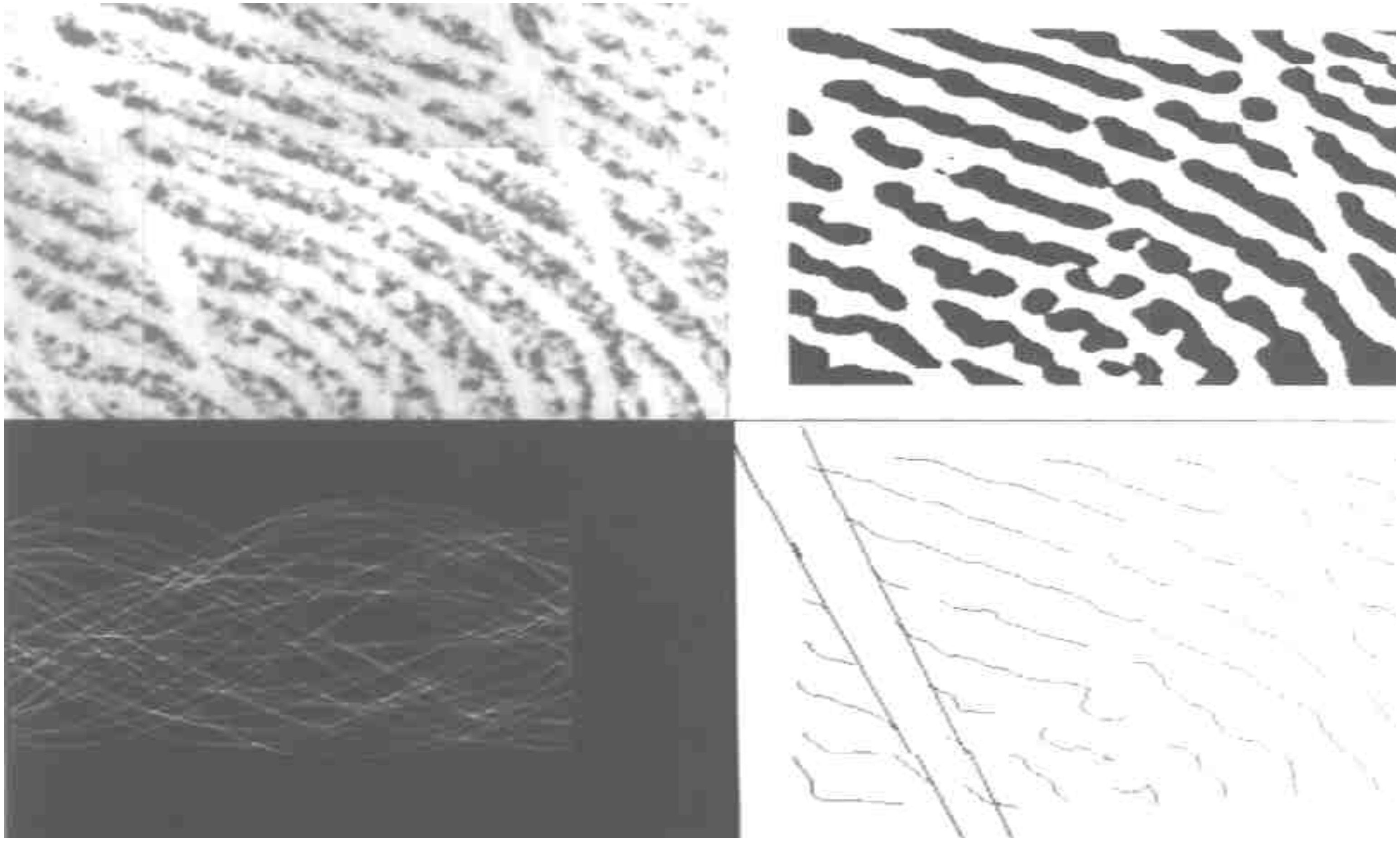


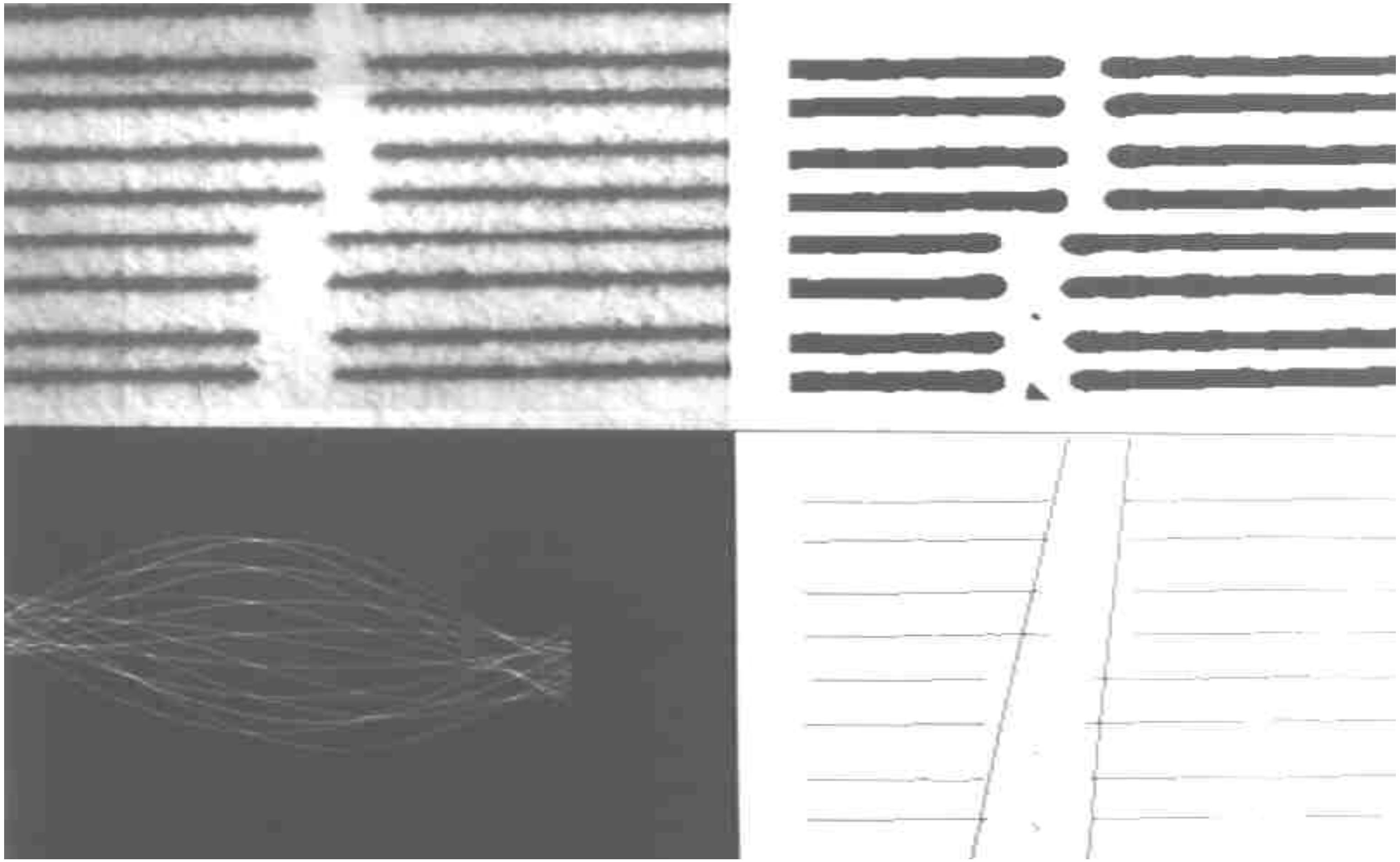


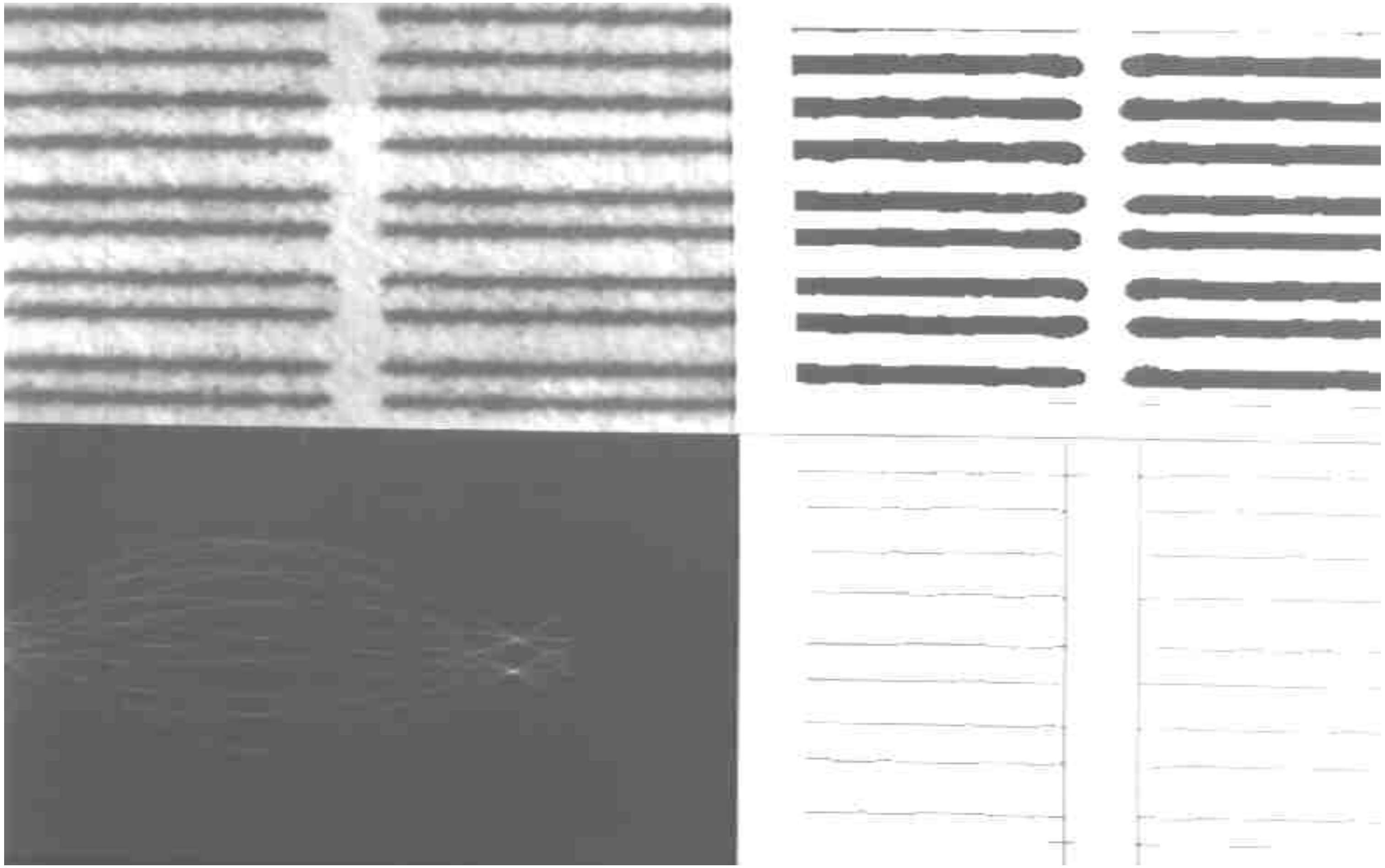












Hough Transform for Circle Detection

- Just as a straight lines can be defined parametrically, so can a circle

- The equation of a circle is given by

$$(x-a)^2+(y-b)^2=r^2$$

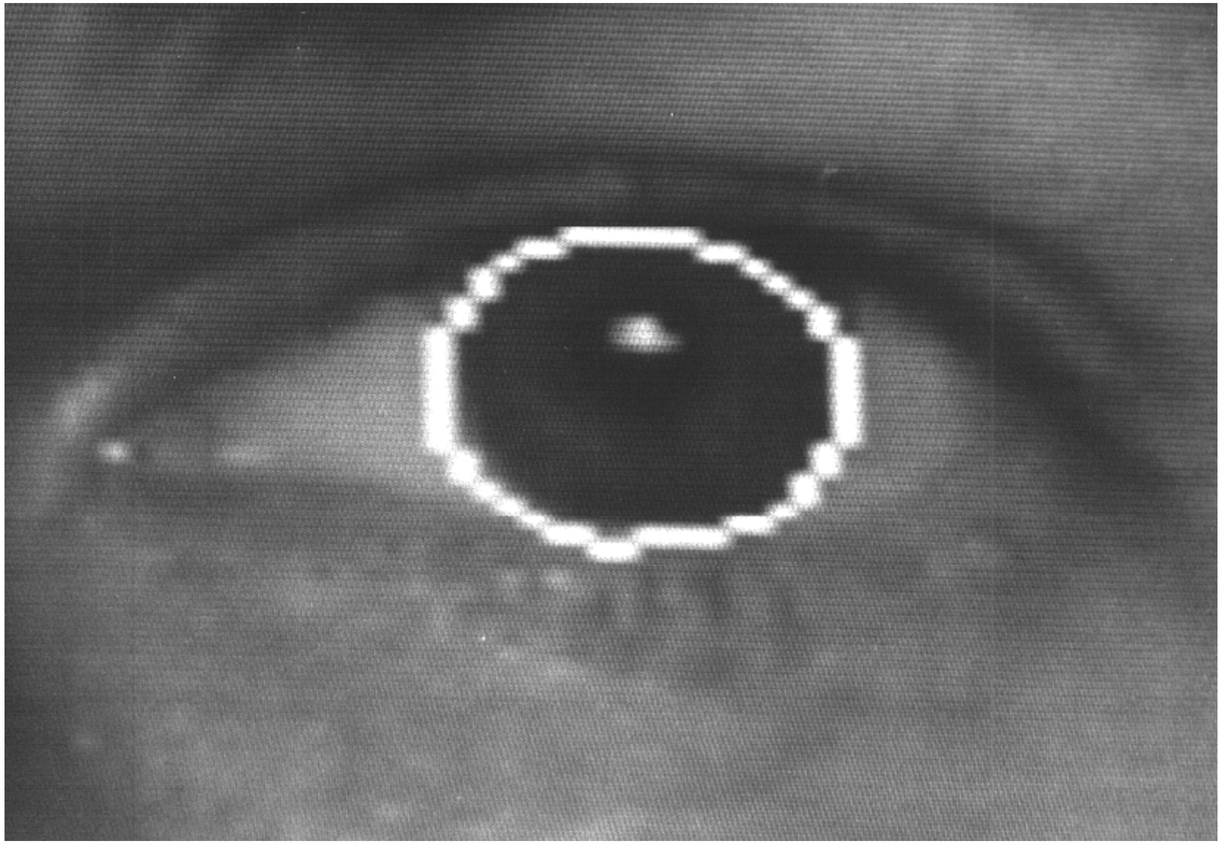
  where (a, b) are the co-ordinates of the centre of the circle

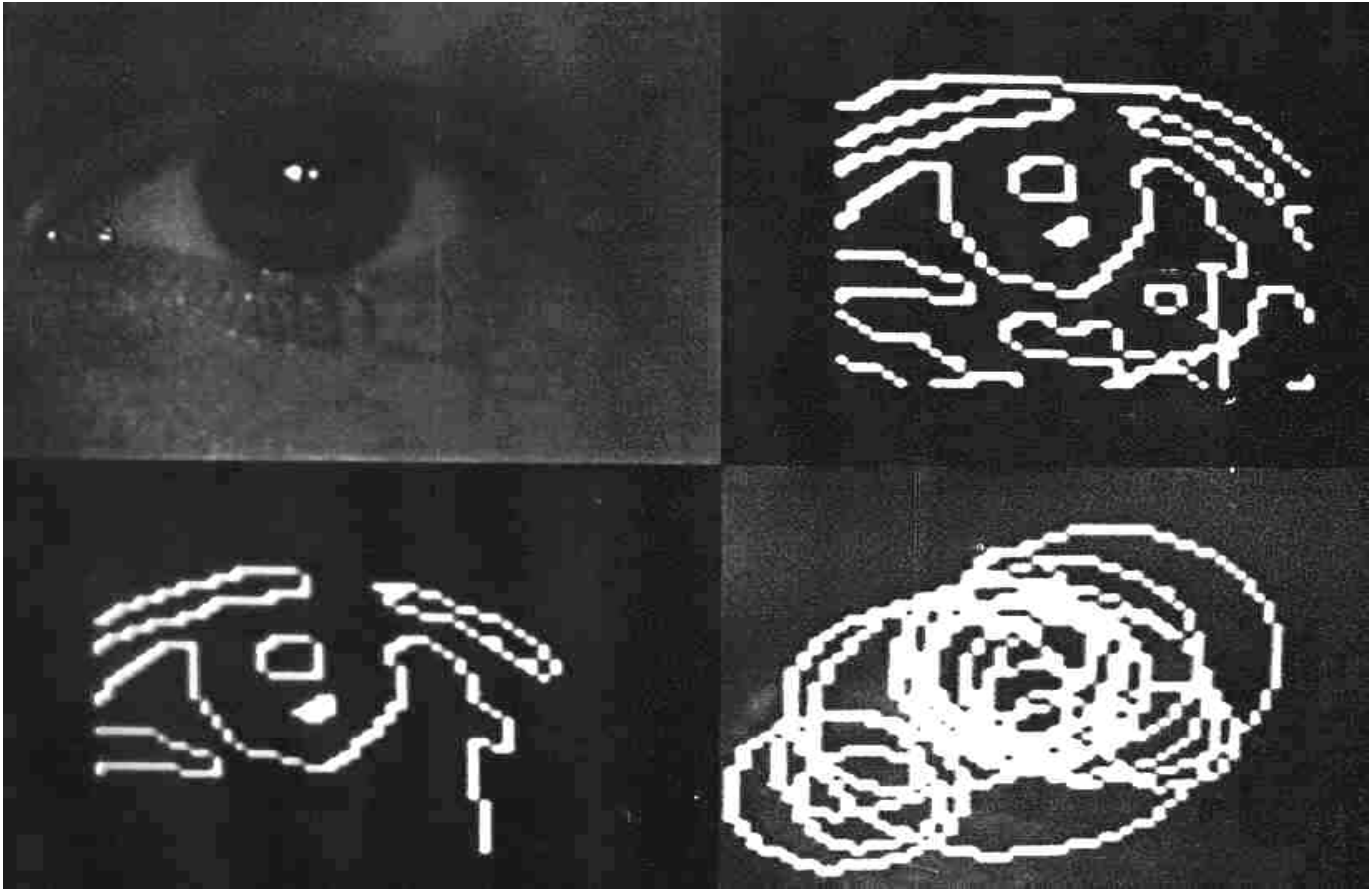
  r is its radius

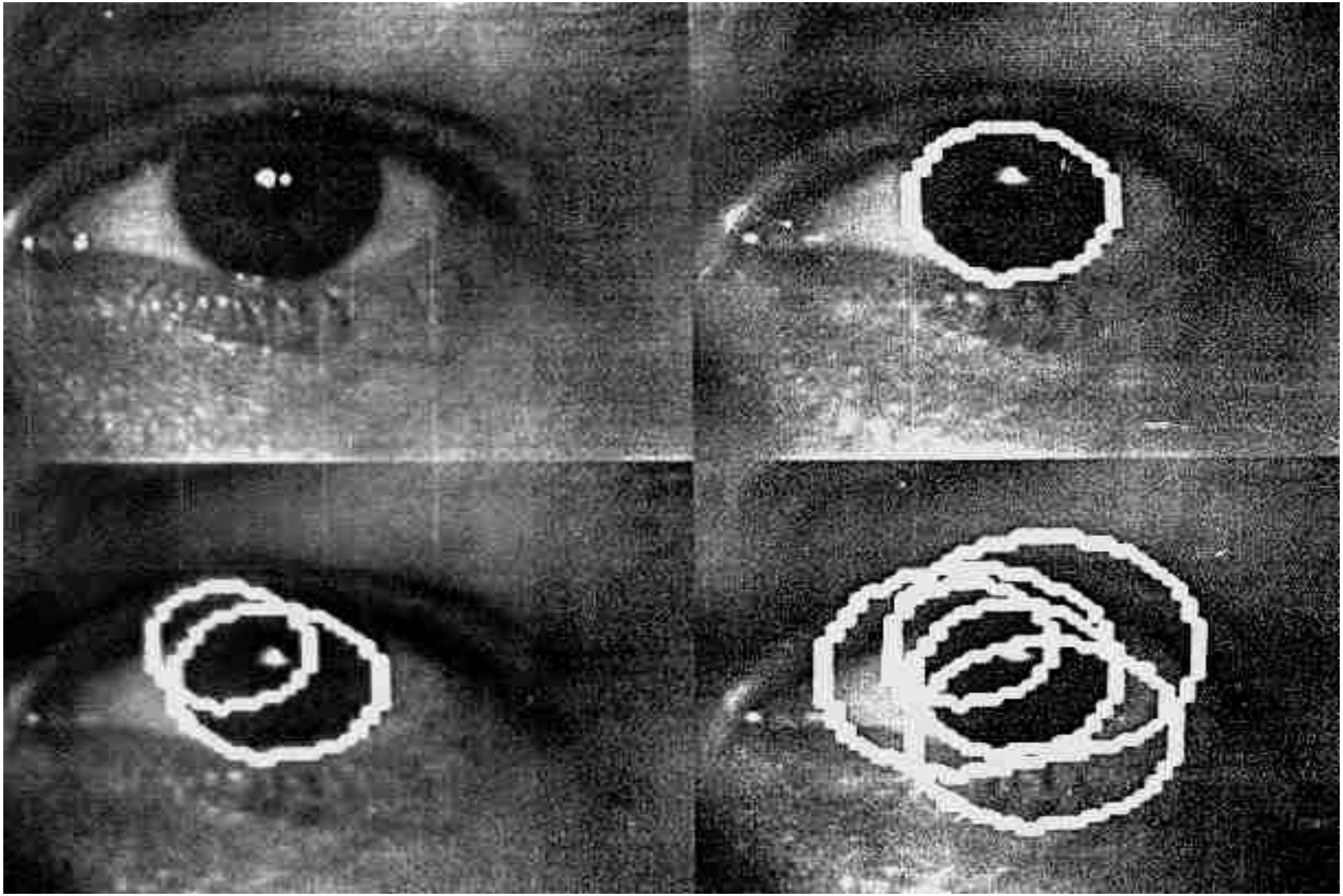
- In this case, we have three co-ordinates in the parameter space: a, b and r

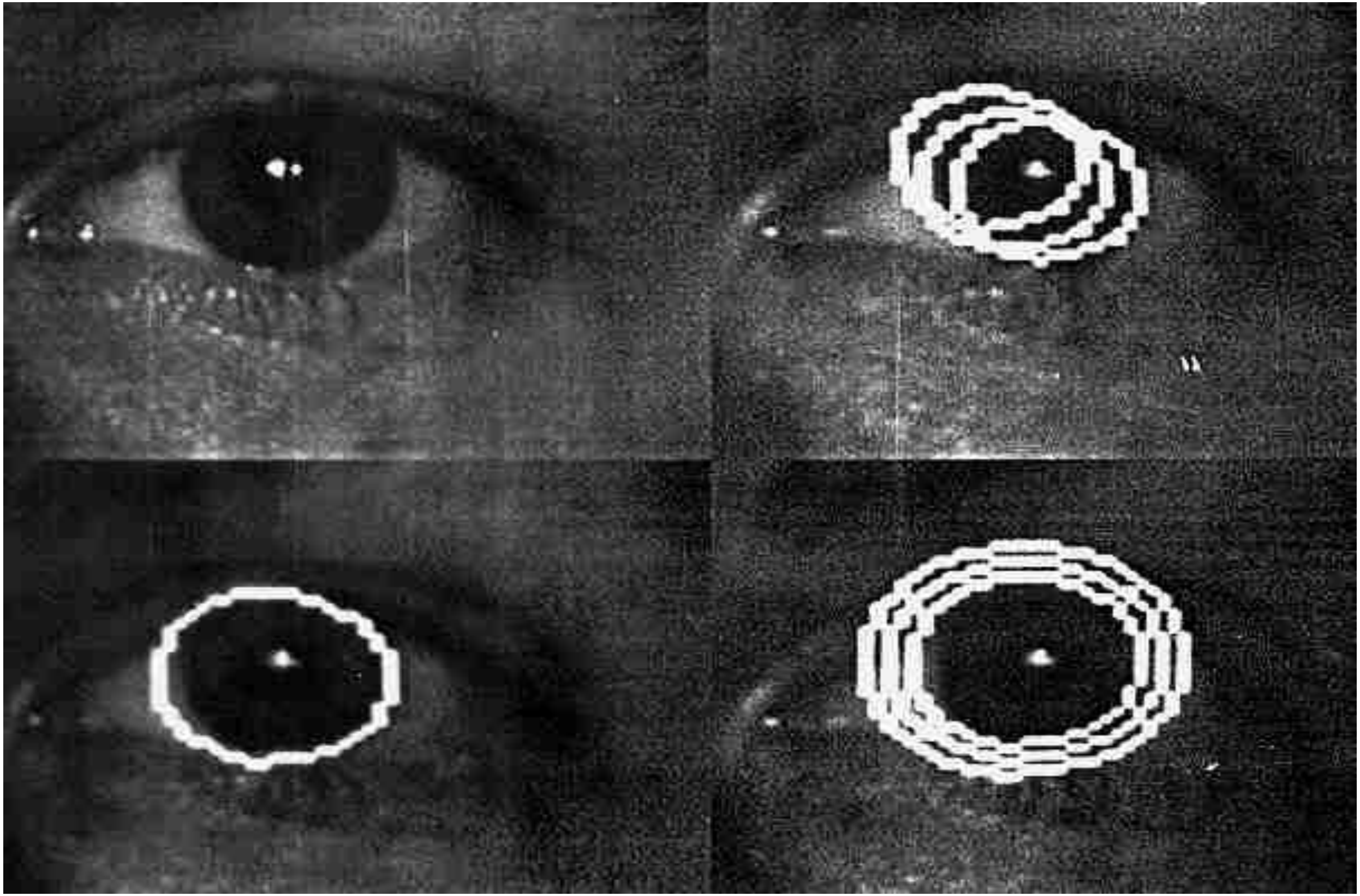
- Hence, we require a 3D accumulator (with an attendant increase in the computational complexity of the algorithm)

Hough Transform for Circle Detection









Hough Transform for Ellipse Detection

- Ellipse: 5 parameters
  - 5D Hough-space is very expensive
- Hough-space for centre point (2D), length of axes (2D) and orientation (1D)

[Aguado, Nixon]: Image

Edges with Canny

2 best ellipses

Credit: Markus Vincze, Technische Universität Wien

Hough Transform

- One further point is worth noting

  The Hough Transform identifies the parameter of the curve (or line) which best fits the data (the set of edge points).

- However, the circles that are generated are complete circles and the lines are infinite

- If you want to identify the line segments or curve segments which generated these transform parameters, further image analysis will be required

Demos

The following code is taken from the houghTransformLines project in the lectures directory of the ACV repository

See:

```
houghTransformLines.h
houghTransformLinesImplementation.cpp
houghTransformLinesApplication.cpp
```

```
void HoughThreshold(int, void*) {

   extern Mat    src;
   extern Mat    src_gray;
   extern Mat    src_blur;
   extern Mat    detected_edges;
   extern Mat    hough;
   extern int    houghThreshold;
   extern char*  hough_window_name;
   vector<Vec2f> lines;
   float         rho;
   float         theta;
   Point         pt1, pt2;
   double        a, b;
   double        x0, y0;
   double        rho_resolution        = 1;    // pixels
   double        theta_resolution      = 2;    // degrees

   src.copyTo(hough);

   HoughLines(detected_edges, lines, rho_resolution, theta_resolution * CV_PI/180, houghThreshold, 0, 0 );

   for (size_t i = 0; i < lines.size(); i++ ) { // now draw the lines
      rho = lines[i][0];
      theta = lines[i][1];

      a = cos(theta);
      b = sin(theta);
      x0 = a*rho;
      y0 = b*rho;
      pt1.x = cvRound(x0 + 1000*(-b));
      pt1.y = cvRound(y0 + 1000*(a));
      pt2.x = cvRound(x0 - 1000*(-b));
      pt2.y = cvRound(y0 - 1000*(a));
      line( hough, pt1, pt2, Scalar(0,0,255), 1, CV_AA);
   }

   imshow (hough_window_name, hough);
}
```

Demos

The following code is taken from the houghTransformCircles project in the lectures directory of the ACV repository

See:

```
houghTransformCircles.h
houghTransformCirclesImplementation.cpp
houghTransformCirclesApplication.cpp
```

```
void HoughThreshold(int, void*) {

   extern Mat    src;
   extern Mat    src_gray;
   extern Mat    src_blur;
   extern Mat    detected_edges;
   extern Mat    hough;
   extern int    houghThreshold;
   extern int    cannyThreshold;
   extern char*  hough_window_name;
   vector<Vec3f> circles;
   Point         pt;
   int           r0;
   double        sigma = 1.0;

   src.copyTo(hough);

   /* condition image for use with Canny edge detector */
   cvtColor(src, src_gray, CV_BGR2GRAY);
   GaussianBlur(src_gray, src_blur, Size(31,31), sigma);

   /* check for invalid low thresholds */
   if (cannyThreshold < 1) cannyThreshold = 1;
   if (houghThreshold < 1) houghThreshold = 1;

   HoughCircles(src_blur, circles, CV_HOUGH_GRADIENT,
                1.0,                     // inverse of resolution
                8.0,                     // minimum distane between circle centres
                (double) cannyThreshold, // upper threshold
                (double) houghThreshold, // for centres
                20,                       // minimum radius
                100                       // maximum radius
                );

   for (size_t i = 0; i < circles.size(); i++ ) {
      pt.x = (int) circles[i][0];
      pt.y = (int) circles[i][1];
      r0 =   (int) circles[i][2];
      circle(hough, pt, r0, Scalar(0,255,255), 1, CV_AA); // draw circles in yellow
   }

   imshow (hough_window_name, hough);
}
```

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 4.3 Lines

Section 4.3.2 Hough Transforms