Topic: 9 Edge and Line Detection

Contents:

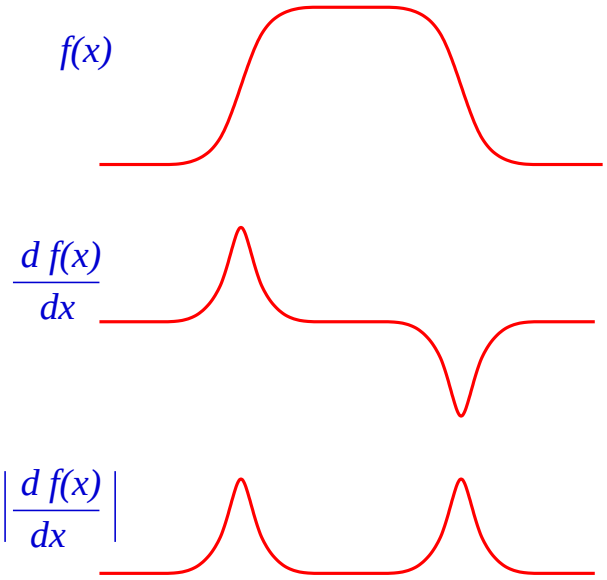
- First Order Differentials
- Post Processing of Edge Images
- Second Order Differentials.
- LoG and DoG filters
- Models in Images
- Least Square Line Fitting
- Cartesian and Polar Hough Transform
- Mathemactics of Hough Transform
- Implementation and Use of Hough Transform

Edge Detection

The aim of all edge detection techniques is to enhance or mark edges and then detect them.

All need some type of High-pass filter, which can be viewed as either First or Second order differentials.

First Order Differentials: In One-Dimension we have



We can then detect the edge by a simple threshold of

$$\left|\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right| > T \quad \Rightarrow \text{Edge}$$

Edge Detection I

but in two-dimensions things are more difficult:

$$\left|\frac{\partial f(x,y)}{\partial x}\right| \rightarrow \text{Vertical Edges} \qquad \left|\frac{\partial f(x,y)}{\partial y}\right| \rightarrow \text{Horizontal Edges}$$

But we really want to detect edges in **all** directions.

In two-dimensions the first order differential $\nabla f(x,y)$ is a vector, given by

$$\frac{\partial f(x,y)}{\partial x}\hat{i} + \frac{\partial f(x,y)}{\partial y}\hat{j}$$

so we need to calculate the modulus of the gradient, given by

$$|\nabla f(x,y) = \sqrt{\left|\frac{\partial f(x,y)}{\partial x}\right|^2 + \left|\frac{\partial f(x,y)}{\partial y}\right|^2}$$

which is a *non-linear* operation.

We can now threshold to give the edges, with

$$\begin{aligned} |\nabla f(x,y)| &> T \quad \text{Edge} \\ |\nabla f(x,y)| &< T \quad \text{no Edge} \end{aligned}$$

Digital Implementation

Real Space: From previous we know we can form the first order differentials by Convolution, with

$$\frac{\partial f(i,j)}{\partial i} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \odot f(i,j)$$

and

$$\frac{\partial f(i,j)}{\partial j} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \odot f(i,j)$$

so we can calculate the

$$|\nabla f(i,j)| = \sqrt{\left|\frac{\partial f(i,j)}{\partial i}\right|^2 + \left|\frac{\partial f(i,j)}{\partial j}\right|^2}$$

This implementation requires considerable numerical calculation. Note that square root must be calculated in floating point.

"Faster" approximation to the modulus of the gradient by

$$\left|\frac{\partial f(i,j)}{\partial i}\right| + \left|\frac{\partial f(i,j)}{\partial j}\right|$$

which is frequently a computational saving of 30%.

Sobel Filter

This is a slight variation on the first order differential filter where two differentials are given by:

$$\frac{\partial f(i,j)}{\partial i} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \odot f(i,j) \qquad \frac{\partial f(i,j)}{\partial j} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \odot f(i,j)$$

where the centre of the differentials is weighted by two.

Sobel filter is formed from the geometric sum, and the fast Sobel by the sum of the moduli.

This is the most common simple first order differential edge detector.

Selection of Threshold

Guess: for example set threshold at 30% of maximum $|\nabla f(i,j)|$.

Percentage of Edge: Set threshold so that x% of image classified as edge.

Histogram of $|\nabla f(i,j)|$ set threshold to set x% of pixels to on. [10% is a good guess].

Typical problems of:

- Arbitrary threshold value
- Thick edges
- Broken edges
- Noise points

Still a very useful simple edge detector for low noise, high contrast images.

Post Processing of Edge Image

Thick Edges: if threshold is too low edges frequently thick.

Range of edge thinning techniques that try to thin edges to a single pixel by removing edge pixels while keeping the edges connected.

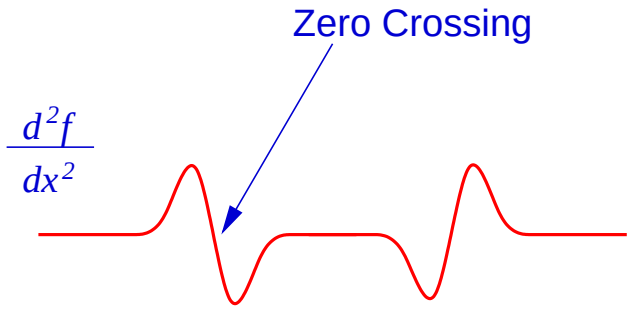
Broken Edges: Range or edge-joining techniques to try and bridge gaps (see *Computer Vision*). also Hough Transform (next lecture), to fit lines.

Noise Points: Modified threshold filter to remove isolated points or non-connected double points.

Range of these in the literature. All work to some extent, if the Sobel fails it is usually better to look for a better technique.

Second Order Differentials

Again in one dimension



the edge is then located by the zero crossing.

In two-dimensions we are required to calculate the Laplacian,

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Second Order Differentials I

Which, as seen previously, can be implemented by a single $[3 \times 3]$ convolution of

$$\nabla^2 f(i,j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot f(i,j)$$

Location of Edge

Find the edges by location the zero crossings.

- **Thin Edges:** the edges occur between pixels. Always get thin edges, but difficult to display on a digital image.
- **Closed Loops:** edges always form closed loops, reduces break-up of edges, but can cause problems as corners.
- **Noise Problems:** Laplacian is a high pass filter, so enhances high frequencies, and thus noise.

Difficult to post process edge image, so to reduce the effect of noise we typically want to smooth the image *before* we form the Laplacian.

For example use Nine point average, then Laplacian.

Noting that the convolution is a linear operation, the two $[3 \times 3]$ convolutions can be implemented as a singe $[5]$ convolution of:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & -2 & -1 & -2 & 1 \\ 1 & -1 & 0 & -1 & 1 \\ 1 & -2 & -1 & -2 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

LoG Filter

A good way to smooth an image is to convolve it with a Gaussian.

The Laplacian of the smoothed image is them

$$g(i,j) = \nabla^2 [h(i,j) \odot f(i,j)]$$

where $h(x,y)$ is a Gaussian of the form

$$h(i,j) = \exp\left(-\frac{r^2}{2\sigma^2}\right)$$

where $r^2 = i^2 + j^2$ and σ is the width of the Gaussian.

The convolution is linear, so we can write

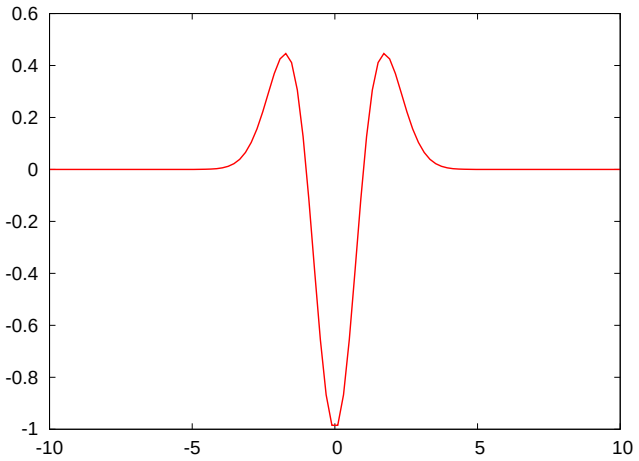
$$g(i,j) = \left[\nabla^2 h(i,j)\right] \odot f(i,j)$$

LoG Filter

We can differentiate the Gaussian to get

$$\nabla^2 h(r) = \frac{1}{\sigma^2}\left[\frac{r^2}{\sigma^2} - 1\right]\exp\left(-\frac{r^2}{2\sigma^2}\right)$$

which for $\sigma = 1$ has shape:



LoG Filter is rather large in real space,, so it is easier to implement in Fourier space.

Fourier Space Implementation

In Fourier space we have that

$$\frac{\partial h(i,j)}{\partial i} = \mathcal{F}^{-1}\left\{\imath\frac{2\pi k}{N}H(k,l)\right\}$$

where $H(k,l) = \mathcal{F}\{h(i,j)\}$, and $h(i,j)$ is of size $N \times N$ pixels, so that the LoG filter

$$\nabla^2 h(i,j)) = \mathcal{F}^{-1}\left\{-\left(\frac{2\pi w}{N}\right)^2 H(k,l)\right\}$$

where $w^2 = k^2 + l^2$.

Now $H(k,l)$ is a Gaussian of reciprocal width, which is given by

$$H(k,l) = \exp\left(-\frac{w^2}{w_0^2}\right)$$

where

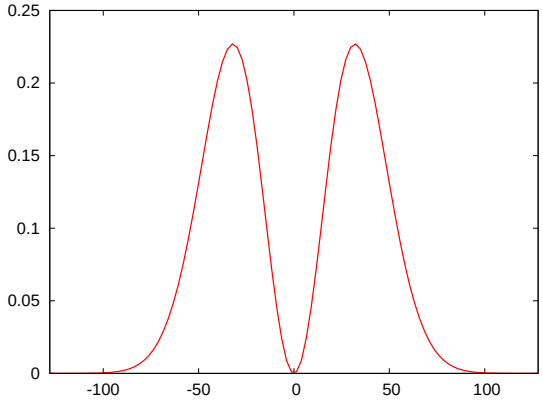
$$w_0 = \frac{N}{2\pi\sigma}$$

so that in Fourier space the LoG filter is given by a Fourier Filter of

$$\left(\frac{2\pi w}{N}\right)^2 \exp\left(-\frac{w^2}{w_0^2}\right)$$

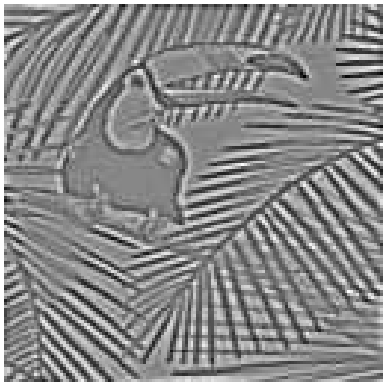
Fourier Space Implementation

So a typical LoG filter for $N = 256$ and $w_0 = 32$ has shape of:



which has the shape of a Band-Pass filter.

Gaussian at high spatial frequencies gives noise reduction, parabola at low spatial frequencies gives the Laplacian.



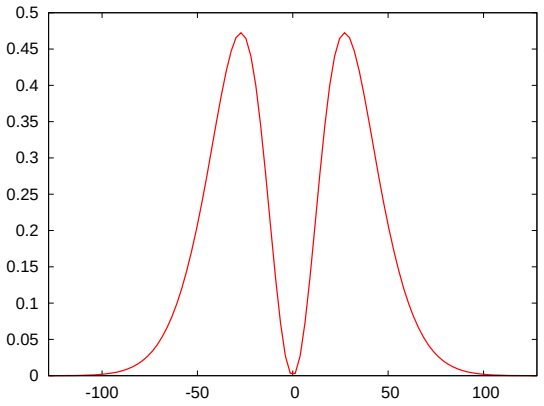
DoG Filter

Fourier filter of Difference of Guassians, (DoG) filter, which has the general shape of

$$H(w) = \exp\left(-\frac{w^2}{w_0^2}\right) - \exp\left(-\frac{w^2}{w_1^2}\right)$$

being the difference of two Gaussians.

This again is a Band-pass filter, which with $w_0 = 40$ and $w_1 = 20$ has shape:



Again the will approximate to a smoothed Laplacian of the image.

DoG in Real Space

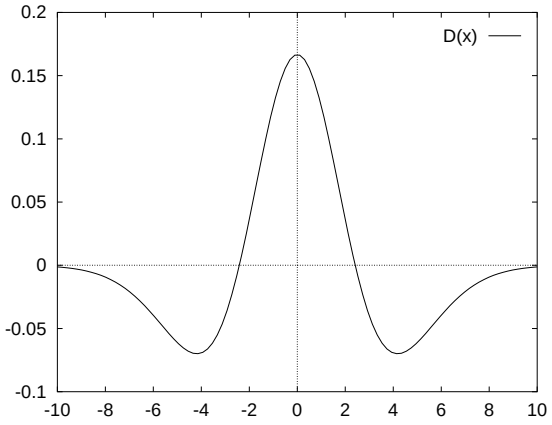
Fourier Transform of the difference of Gaussians, is again a difference of Gaussians,

$$h(r) = \left[\frac{1}{\sigma_0}\exp\left(-\frac{r^2}{2\sigma_0^2}\right) - \frac{1}{\sigma_1}\exp\left(-\frac{r^2}{2\sigma_1^2}\right)\right]$$

where we have that

$$\sigma_0 = \frac{N}{2\pi\, w_0} \quad \& \quad \sigma_1 = \frac{N}{2\pi\, w_1}$$

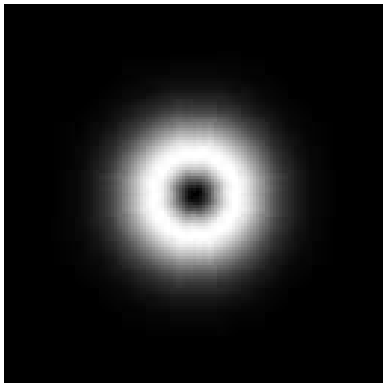
where N is the size of the image. This filter is shown for $\sigma_0 = 3$ and $\sigma_1 = 2$ pixels,



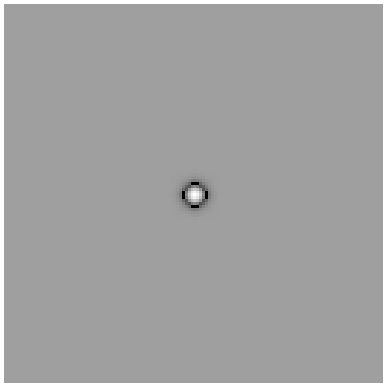
which is also known as the Marr-Hildrith filter, or “Mexican-Hat” filter.

Example in Two-Dimensions

For $N = 128$, $w_0 = 20$ and $w_1 = 10$.



DoG (Fourier)



Real Space

which gives $\sigma_0 \approx 1$ and $\sigma_1 \approx 2$.

Approximates the Laplacian of Gaussian filter, but with two controllable parameters.

Use: flexible edge detection filter, used extensively in Computer Vision.

Aside: Models of animal/human visual system suggest that DOG filter is fundamental to vision process as is essentially performed on the retina before information sent to brain for interpretation.

Fitting Model to Image

To start "*What do we want to fit?*"

In two dimensions we wish to fit a range of standard shapes to the image

- Lines: The simplest
- Squares, Rectangles, polygons: made-up of Lines
- Circles and Ellipses:

look at on the simplest; *lines*. Other are beyond this course.

Before we fit any simple shape,

1. Apply Edge Detection.
2. Threshold to get binary edges.

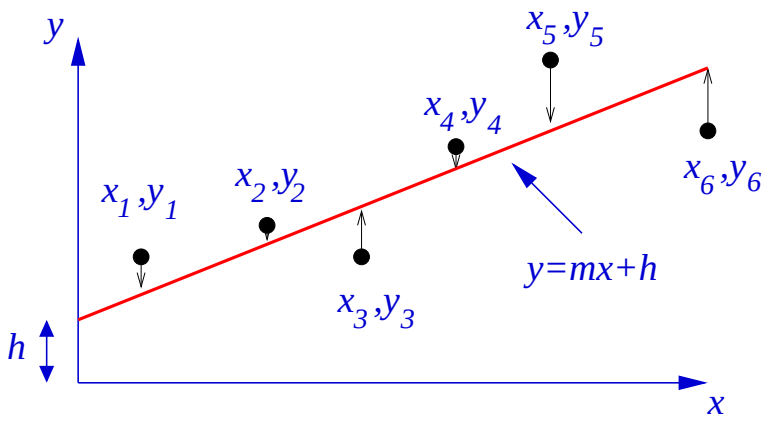
Main problem is that edges are generally *not* complete (breaks in the edges, as discussed above).

Fitting a Straight Line

To fit a single straight line, we must fit

$$y = mx + h$$

Simplest is a Least Squares fit.



If we have points y_i, x_i, being point on the line, define

$$e^2 = \sum_{i=1}^{n} (y_i - (mx_i + h))^2$$

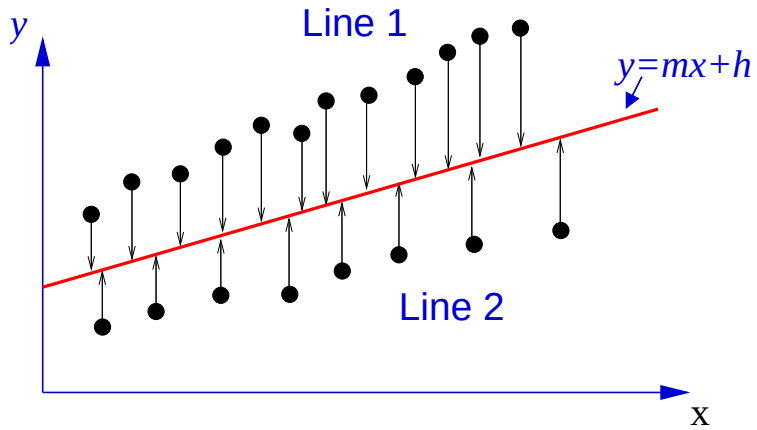
and we get the standard solution by minimising e^2 by setting

$$\frac{\partial e^2}{\partial m} = 0 \quad \frac{\partial e^2}{\partial h} = 0$$

we have minimised the *vertical* distance between the points and the line.

Multi-line problem

But if there is *more* than one line,



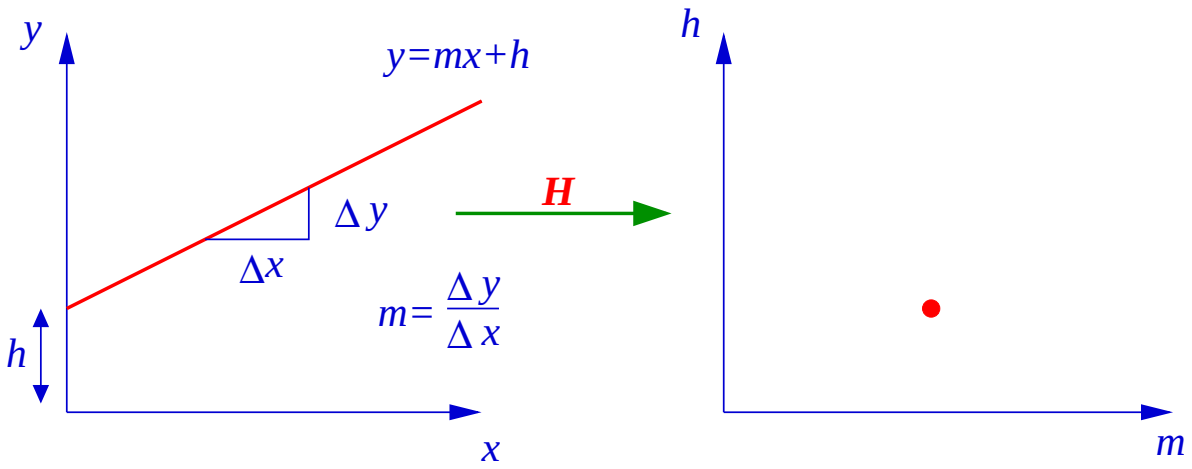
You get the average best-fit, single line, which is usually wrong.

Least-Square only works if you have a *single* line, or are able to segment out a segment of the image that contains a single line.

We need to look for something a lot more general than this.

Hough Transform

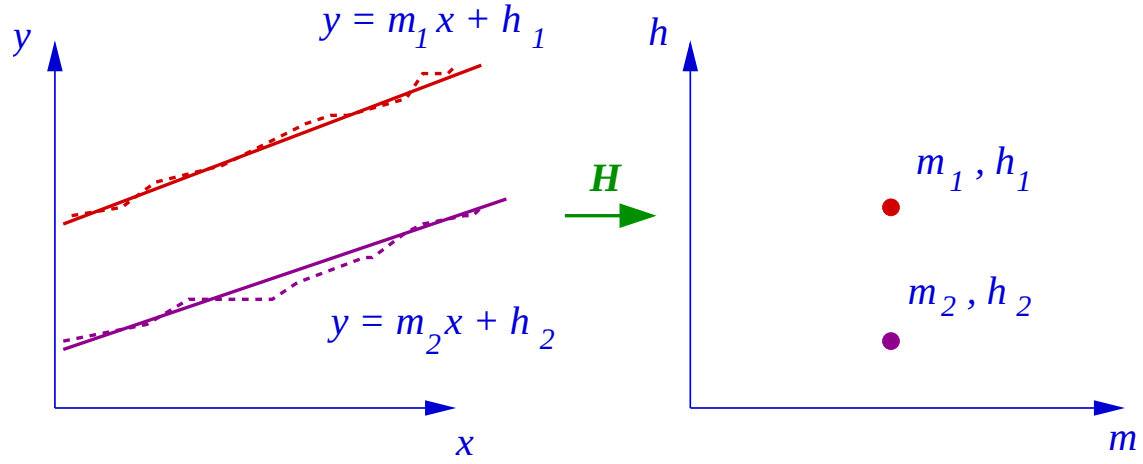
Consider a line-to-point transform,



that transforms the data from x, y space to m, h space, where each line is transformed to a point.

Hough Transform

If we have multiple lines, we then get multiple points,



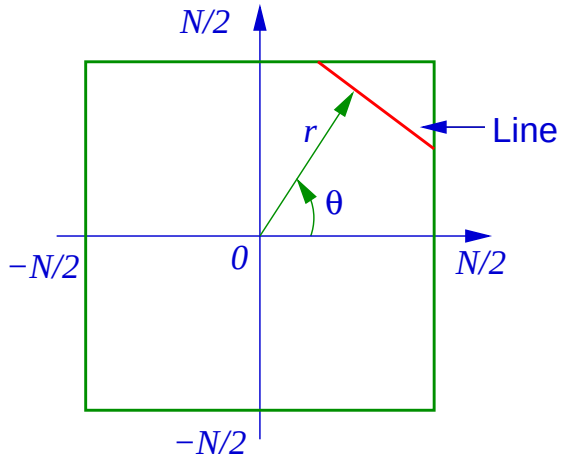
which is valid for any number of lines.

Problem: neither m **or** h are bounded, so that

$$\text{Line} \parallel \text{to } x \text{ axis} \quad \Rightarrow \quad h \text{ not defined}$$
$$\text{Line} \parallel \text{to } y \text{ axis} \quad \Rightarrow \quad m \rightarrow \infty$$

Polar Hough Transform

We can describe a line in Polar Coordinates by two variables r, θ

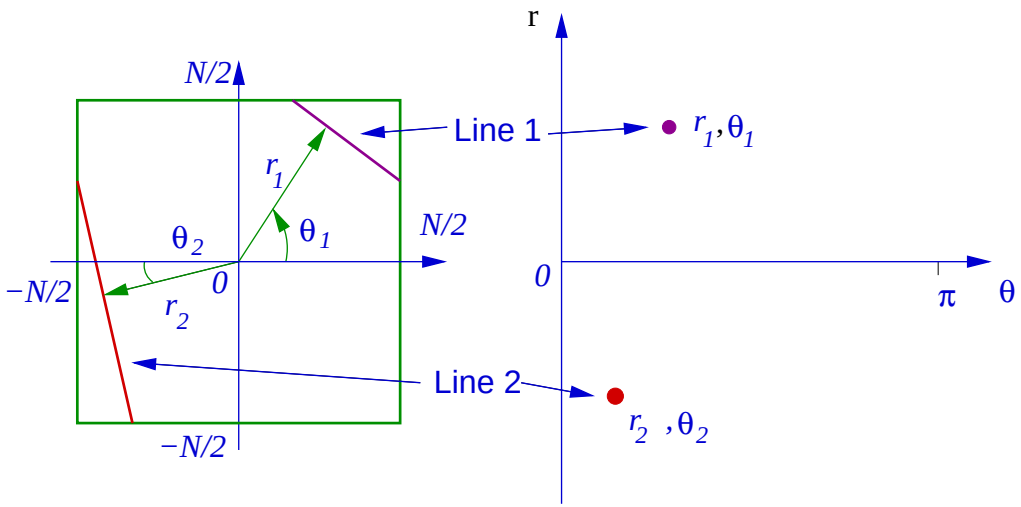


where, if we put the origin at the centre, then we have

$$-\frac{N}{\sqrt{2}} < r \leq \frac{N}{\sqrt{2}}$$
$$0 \leq \theta < \pi$$

Polar Hough Transform

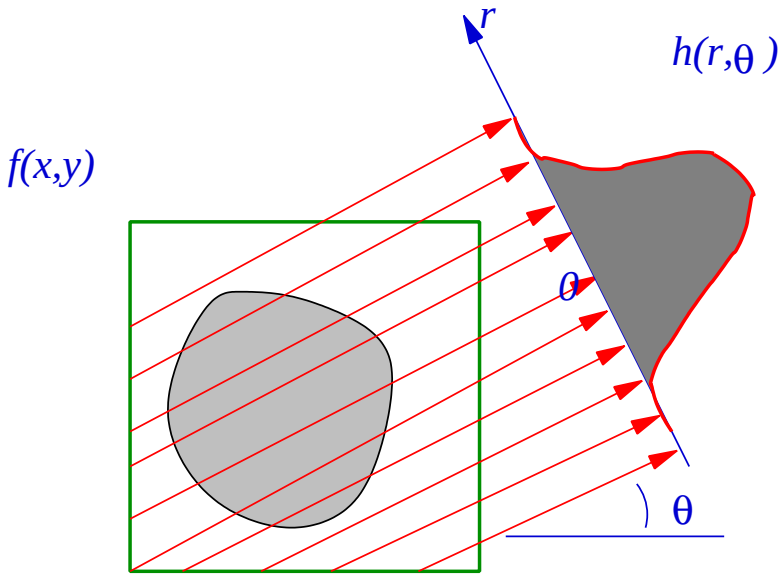
Which are **both** bounded, so we get a finite range in both x, y and r, θ space.



and again multiple lines are transformed to multiple points.

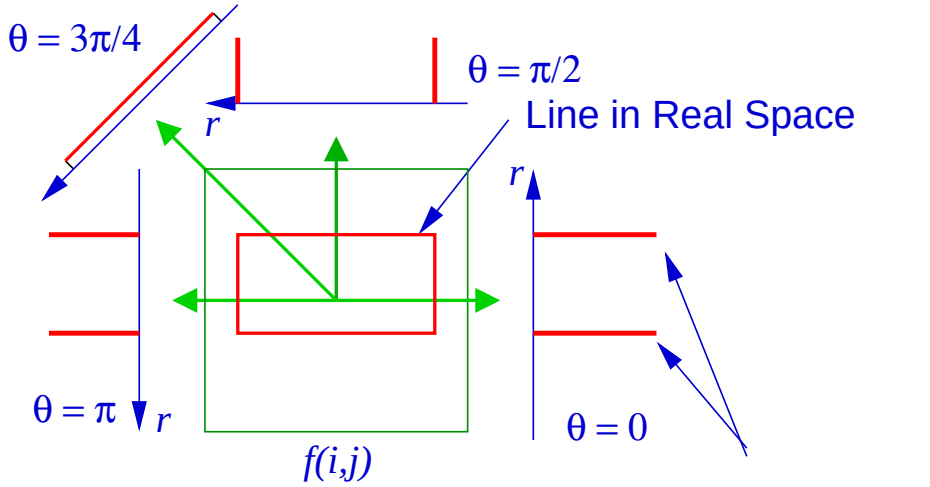
Projection Implementation

Implementation polar Hough Transforms as a series of projections at various angles,



Projection Implementation

which can be repeated at each angle to get the full Hough Transform.



with each projection giving one θ line.

It is the same projection as the Radon Transform found in tomography.

Mathematics of Transform

The equation of a line at angle θ and position r is given by

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

which can be written in more familiar notation as:

$$y = \frac{r}{\sin\theta} - \frac{x}{\tan\theta} = h + mx$$

so to form the Hough Transform we need to integrate along *each line*, so in integral form as

$$H(r,\theta) = \iint f(x,y)\delta(r - x\cos\theta - y\sin\theta)\mathrm{d}x\mathrm{d}y$$

which is the Radon Tranform seen in tomography.

This does not help much, but does allow to look at and alternative visualisation and implementation.

Point Implementation

Look at the Hough Transform of a single point at x_0, y_0, so that

$$f(x,y) = \delta(x - x_0, y - y_0)$$

so that the Hough Transform is

$$H(r,\theta) = \iint \delta(x - x_0, y - y_0)\delta(r - x\cos\theta - y\sin\theta)\mathrm{d}x\mathrm{d}y$$

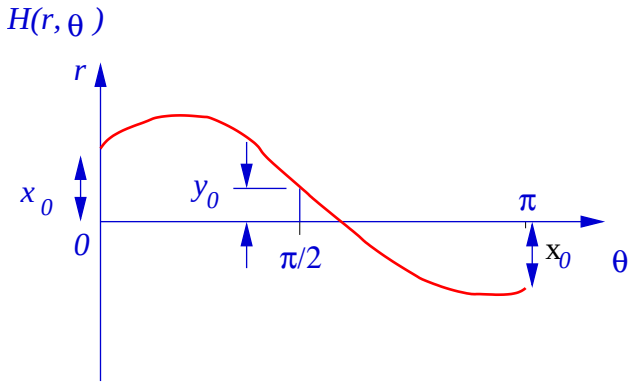
which if we apply the shifting properties of δ-fn, just gives that

$$H(r,\theta) = \delta(r - x_0\cos\theta - y_0\sin\theta)$$

which is just a curve in r, θ space of the form

$$r = x_0\cos\theta + y_0\sin\theta$$

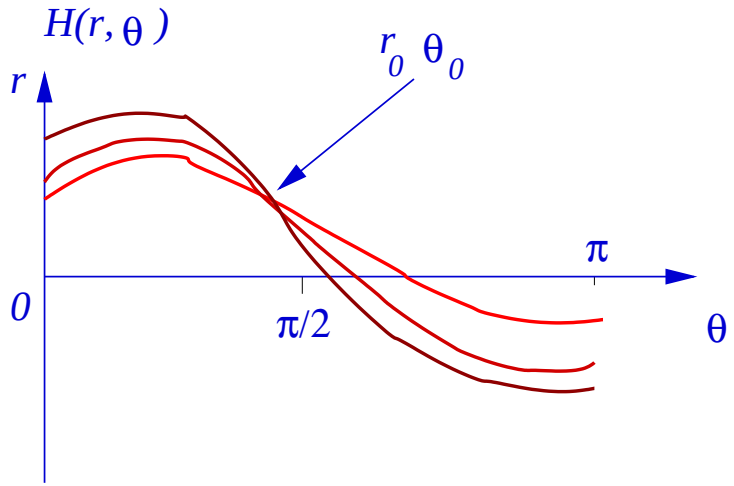
which gives



Line of Points

If we consider a line of points, then if the points form a line at a particular r_0, θ_0:

Each cos line crosses at one point, giving the r_0, θ_0 of the line,



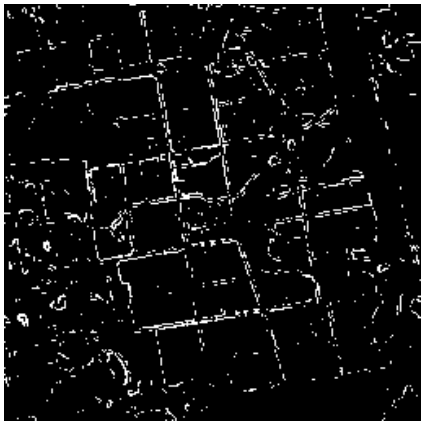
So if the image to be transformed is a small number of binary points, (edge detected image), implementation is just:

- Start with blank image.
- For each edge point in the input image, **add** "cos-line" to Hough image.

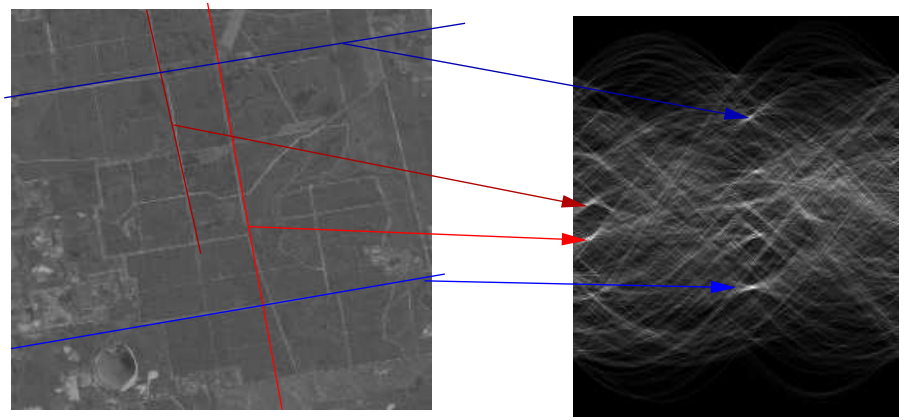
This can be significantly speeded-up by using a pre-calculated table to return the $\cos()$ and $\sin()$.

Example of Hough

Firstly take the image, form Sobel and threshold to get broken lines where the roads are:



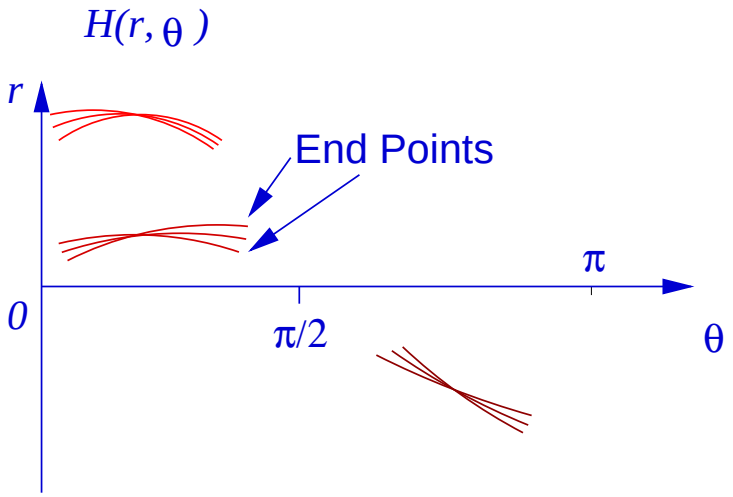
Then form the Hough, with r on the vertical axis and θ on the horizontal.



The peaks then give the *equations* of the lines in the image.

Shape of Hough Peak

Hough peak is *not* sharp, but is made-up from a set of crossing cos-curves. The shape of the peak will depend on the location,



but all have a butterfly shape.

To detect the peak,

- Threshold if clear peaks (from distinct, long lines).
- Filter to enhance the butterfly shape, then threshold.

Actually fairly *easy* for good low-noise images.

Uses of Hough

The Polar Hough transform is the only for actually used:

- Gives equations of lines (but not “end-points”).
- Works well with simple images that contain good straight lines. (Good for Robot Vision)
- Deal with broken lines very well.
- Reasonably efficiently if there are “few” edge points.

Problems:

- Very slow of there are many edge points.
- Hough Space is non-linear, different edge sensitivities in different directions.
- Poor for short lines.

Extensions

Range of recent extensions of the Hough Transform, for example:

- Circle and Ellipse detection by “Double Hough”.
- Image transform plus Hough for general shape detection.

Summary

In this section we have considered

- First Order Differentials
- Post Processing of Edge Images
- Second Order Differentials.
- LoG and DoG filters
- Models in Images
- Least Square Line Fitting
- Cartesian and Polar Hough Transform
- Mathemactics of Hough Transform
- Implementation and Use of Hough Transform