Topic 9: Edge and Line Detection

9.1 Introduction

One of the major tasks in image processing is the enhancement and detection of edges followed by the extraction of primitive features, mainly lines. This is used in multiple applications especially in remote sensing and robotic vision.

9.2 Edge Detection

The aim of all edge detection techniques is to either enhance or mark edges and then detect them. Edges are rapidly varying parts of the image so are represented by high spatial frequencies, so for the initial enhancement we need some type of high pass filter, which, as we have seen in previous sections, can be viewed as either *first* or *second* order differentials.

9.2.1 First Order Differentials

In one-dimension the differential of an edge is as shown in figure 1. The first order differential goes alternatively positive then negative, so the peak of its modulus locates the centre of the edge. We can then detect the edge by a simple threshold of

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

which will detect both positive and negative edges.

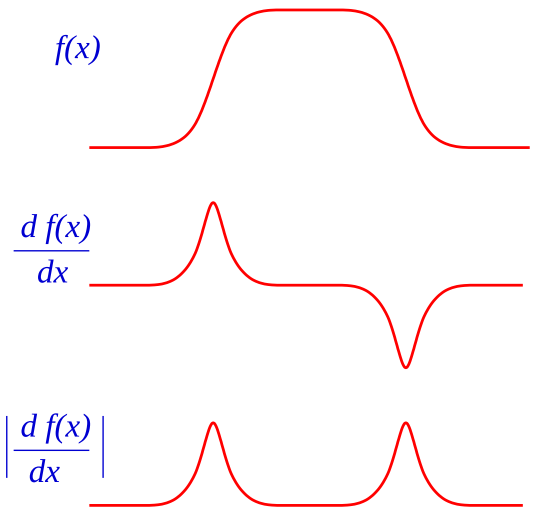


Figure 1: A one-dimensional edge and its first order differential.

In two-dimensional, in images, things are more complex since there are two derivatives being with respect to x and y which gives the vertical and horizontal edges with,

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

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

$$\frac{\partial f(x,y)}{\partial x}\hat{\imath}+\frac{\partial f(x,y)}{\partial y}\hat{\jmath}$$

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, so has no direct equivalent in Fourier space. We can now threshold to give the edges, with

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

To implement this for a digital image we know from previous that in real space first order differentials can be formed by convolution with the simple 3×3 mask of

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

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 since the that square root must be calculated in floating point and a *faster*[1] approximation to the modulus of the gradient can be formed 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%.

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

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

where the centre of the differentials is weighted by two. This is known as the *Sobel Filter* where the full filter is formed from the geometric sum while the *fast* Sobel by the sum of the moduli. This is the most common simple first order differential edge detector with typical results shown in figure 2 showing good clean edges especially when used on a low noise image.

To get the final *edge detected* image we must then threshold edge enhanced image. There are a range of possible scheme to set this threshold, for example

[1]known an Manhattan approximation.

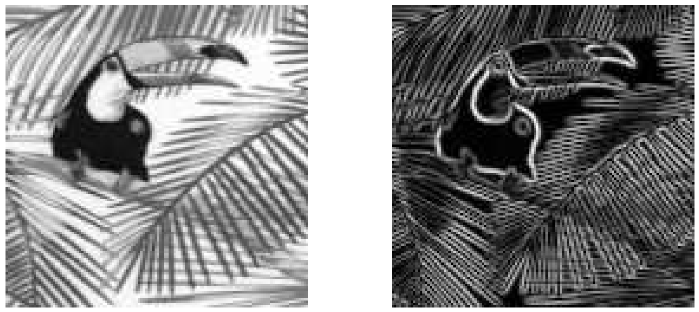


Figure 2: Effect of the sobel filter.

1. **Guess:** for example set threshold at 30% of maximum $|\nabla f(i,j)|$, this is often enough.
2. **Percentage of Edge:** Set threshold so that x% of image classified as edge. To form this take the histogram of $|\nabla f(i,j)|$ set threshold to set x% of pixels to *on*, here 10% is a good guess for many natural scenes.

A typical example of setting a threshold at 10% *edge pixels* is shown in figure 3. This scheme is very useful simple edge detector for low noise, high contrast images but there are problems of an arbitrary threshold value that has be set, the resultant edges are *thick*, possibly being many pixels wide, edges are often *broken* with gaps and there are often stray *noise* points where isolated points are classified as edge.

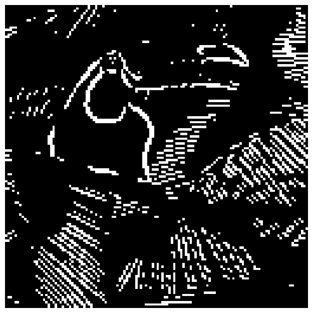


Figure 3: Threshold of Sobel image with 10% of images classified as *edge*.

The binary edge image can be further *post processing* to try and remove some of the problems, the most common being

1. **Thick Edges:** if threshold is too low edges frequently thick, being many pixels wide. There are a range of *edge thinning* techniques that try to thin edges to a single pixel by removing edge pixels while keeping the edges connected. See textbooks for details.
2. **Broken Edges:** Range or *edge-joining* techniques to try and bridge gaps (see *Computer Vision* literature). also the *Hough Transform* considered later that fits lines.
3. **Noise Points:** Modified threshold filter or *median* to remove isolated points or non-connected double points.

Range of these in the literature. All work to some extent, however if the *Sobel* filter fails to give acceptable results it is usually better to look for a better technique rather than to attempt extensive post processing.

9.3 Second Order Differentials

If we consider the second order differential of an edge in one dimension as shown in figure 4, then we see that the centre of the edge is located when this is zero, so the the edge is located at the *zero crossing* of the second order differential. This is therefore a scheme to detect edges *without* the arbitrary threshold require for the first odder schemes.

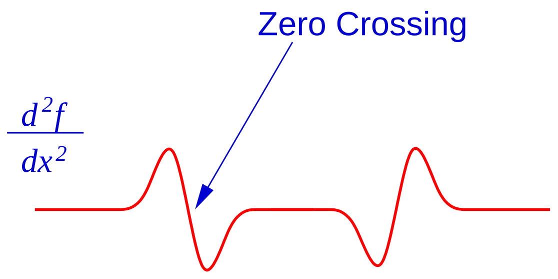


Figure 4: Second order differential of an edge in one-dimensions.

In two-dimensions the second order differential is the Laplacian given by

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

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

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

giving the a typical image as shown in figure 5.

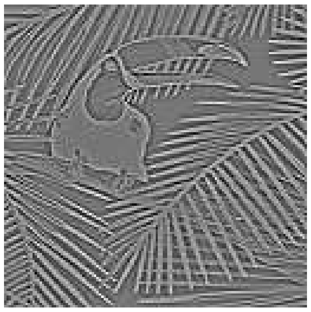


Figure 5: Laplacian of an image formed with a $[3 \times 3]$ mask.

Find the edges by location the zero crossings, or the *zero contour*. The resultant edges will be

- **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.

The resultant edges images are difficult to post process so we need 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 $[5 \times 5]$ convolution of:

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

9.4 Laplacian of Gaussian Filter

We have seen previously that a good way to smooth an image is to convolve it with a Gaussian. This results in a reduction in noise and most importantly does not introduce ringing artifacts that could be miss classified as as *edge*. The Laplacian of the smoothed image is them

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

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

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

We can then 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, in real space, as shown in figure 6.

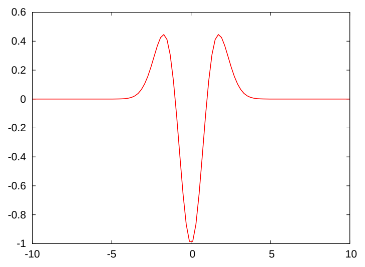


Figure 6: The Laplacian of Gaussian (LoG) filter in real space for $\sigma = 1$.

In real space the LoG Filter is rather large in real space, needing for $\sigma = 1$ a mask of approximately 11×11 pixel to get a good representation. It is therefore computationally advantageous to implement in Fourier space. 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 real space filter is given by

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

where $w^2 = k^2 + l^2$. Now $H(k,l)$ is a Gaussian of reciprocal width[2], 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 Laplacian of Gaussian filter is given by a Fourier Filter of

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

This is shown plotted in figure 7 for $N = 256$ and $w_0 = 32$ has shape of a band-pass filter with the *Gaussian* at high spatial frequencies giving the *noise reduction*, and the *parabola* at low spatial frequencies gives the *Laplacian*

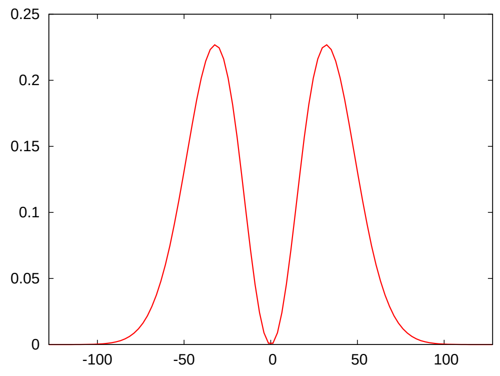


Figure 7: The Laplacian of Gaussian (LoG) filter in Fourier space.

The effect of this on an image is shown in figure 8 which shown a smoothed Laplacian image with no ringing or extra artifacts.

[2]See Fourier transform booklet.

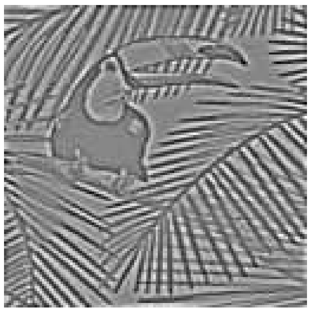


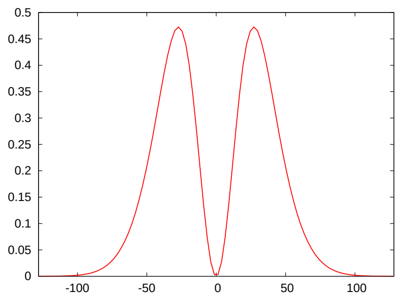
Figure 8: Laplacian of Gaussian filter applied to an image.

9.5 Difference of Gaussian (DoG) Filter.

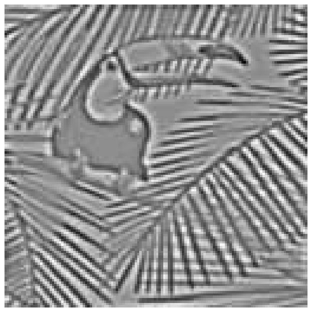
A very closely related Fourier filter is the *Difference of Gaussians*, (DoG) which is defined by

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

being the difference of two Gaussians. This is again a band-pass filter, which is plotted in figure 9 (a) with $w_0 = 40$ and $w_1 = 20$ for a 256×256 image. We can see by comparison with figure 7, that this will also take an approximate to a smoothed Laplacian of the image now with two parameters w_0 which controls to the extend of the smoothing and w_1 which controls the steepness of the filter at low spatial frequencies, and thus the extend of the edge enhancement. The effect on an image is shown in figure 9 (b), being almost identical to the Laplacian of Gaussian filter in figure 8.



(a) Filter profile



(b) Effect on image.

Figure 9: Difference of Gaussian (DoG) filter and its effect on an image.

Fourier Transform of the difference of Gaussians, is again a difference of Gaussians, to in real space, *it can be shown* that the filter is given by

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

where we have that

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

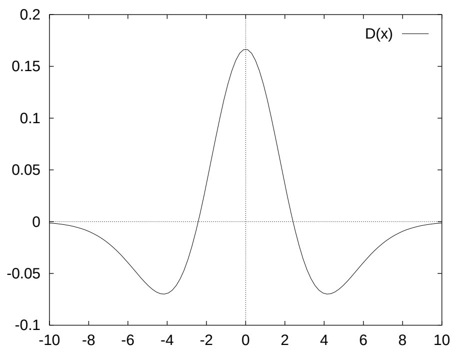
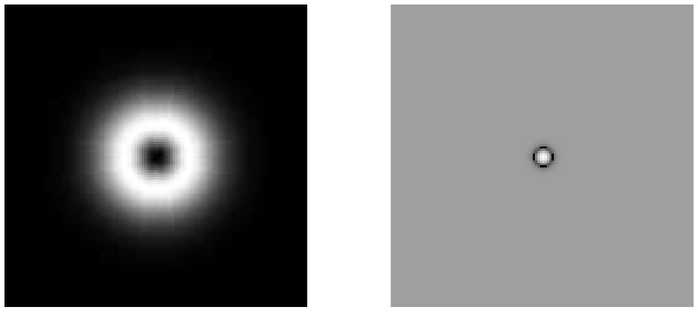


Figure 10: Difference of Gaussian filter in real space with $\sigma_0 = 3$ and $\sigma_1 = 2$.

where N is the size of the image. This filter is shown for $\sigma_0 = 3$ and $\sigma_1 = 2$ pixels in figure 10. This filter is also known as the Marr-Hildrith filter after the first people to use it is computer vision research and more informally as the *Mexican-Hat* filter for obvious reasons.

This is also shown in two-dimensions for $N = 128$, $w_0 = 20$ and $w_1 = 10$ in Fourier space in figure 11 (a) and in real space in figure 11 (b) where $\sigma_0 \approx 1$ and $\sigma_1 \approx 2$.



(a) DoG (Fourier)   (b) Real Space

Figure 11: Difference of Gaussians filter in (a) Fourier and (b) real space.

This is a very flexible edge detection filter which gives a good smoothed Laplacian even with moderately hight amounts of noise. It is particularly used in Computer Vision system for which is was originally designed. 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.

9.6 Fitting Models to Image

To analyse and interpret an image we need to fit or extract a set of simple shapes, the simples being *lines*. From *lines* it is then simple to extend to more complex shapes such as squares, rectangles and polygons all of which are simple made of lines. The process can then be extended to consider circles and Ellipses. In this course we will consider only the simples, being *lines*.

Before we fit any simple shape the first stage is the detect the edges by first applying an edge detector, for example the simple *Sobel* and thresholding to obtain a binary edge images. As we have seen in figure 3 the resultant edges are generally *not* complete but suffer from multiple small breaks due to noise in the image.

9.6.1 Fitting a Straight Line

To fit a single straight line to data, we must fit

$$y = mx + h$$

where m is the gradient and h is the intercept with the y-axis. This is a very common fitting problem and the simplest is a *least squares* fit. If we have n points y_i, x_i, being point on the line, then if we define the square error as

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

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

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

which has the effect of minimising the *vertical* distance between the points and the line as shown in figure 12.

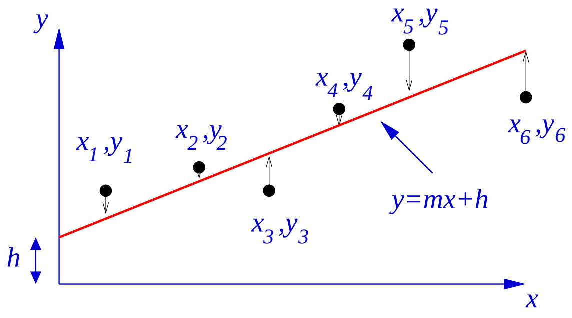


Figure 12: Least square fit to a single line of data points.

The works very well for a single line, *but* if there is *more* than one line, things get rather more complicated and, as shown in figure 13, the simple least squares simply gives the best *average* line 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.

9.7 Hough Transform

Consider the idea of a *line-to-point* transform, as shown in figure 14 where the image data in x, y is transformed to a m, h space, so that each line is transformed to a point. Then as points are easy to detect, then a sharp peak in the m, h space would correspond to a line of the form

$$y = mx + h$$

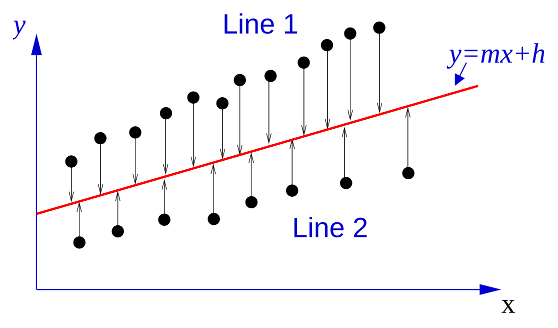


Figure 13: Least square fit two lines of data point by a single line.

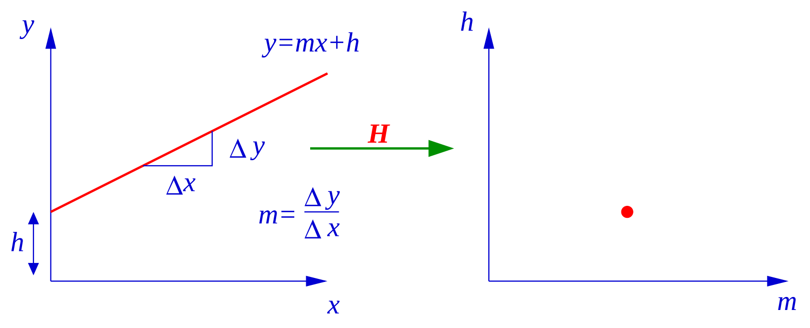


Figure 14: Concept of a *line-to-point* which a line is real space becomes point.

in the original image space.

If was have such a *transform* and we have multiple lines, then as shown in figure 15 we then simple get multiple points, one for each line. So if we can derive such a transform we have solves the general line extraction problem being able to extract any number of lines from an image. However there is a problem with the scheme as it stands since neither *m or h* are bounded, so that

$$\begin{array}{lcl}\text{Line} \parallel \text{to } x \text{ axis} & \Rightarrow & h \text{ not defined} \\ \text{Line} \parallel \text{to } y \text{ axis} & \Rightarrow & m \longrightarrow \infty\end{array}$$

which make the simple scheme computationally impractical, however this can be modified.

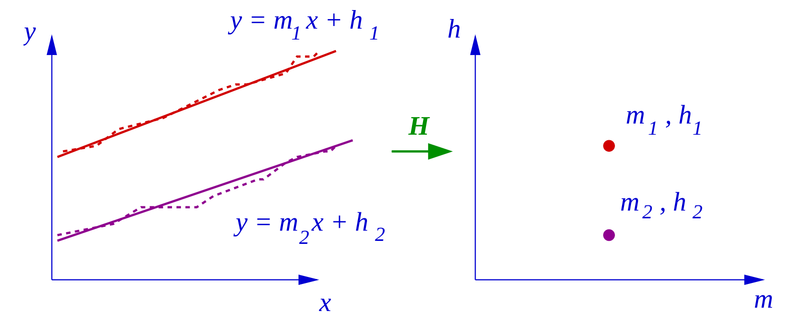


Figure 15: *Line-to-point* transform of multiple lines results is multiple points.

9.7.1 Polar Hough Transform

If alternatively we describe a line in polar coordinates by two variables r, θ as shown in figure 16, where r is the perpendicular distance to the image centre and θ is angle that r line makes with the positive x-axis. when, if we put the origin at the centre, then we have

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

so that both r and θ and bounded and much more practical to calculate.

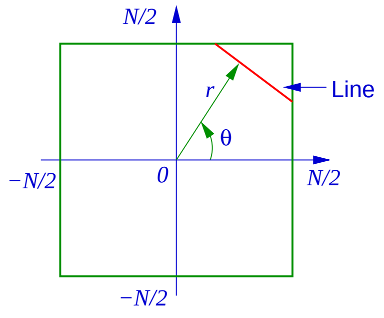


Figure 16: Description of a line in polar coordinates.

Again is we have multiple line, then we will have multiple points in r, θ space as shown in figure 17. This is the polar Hough transform for the detection of lines.

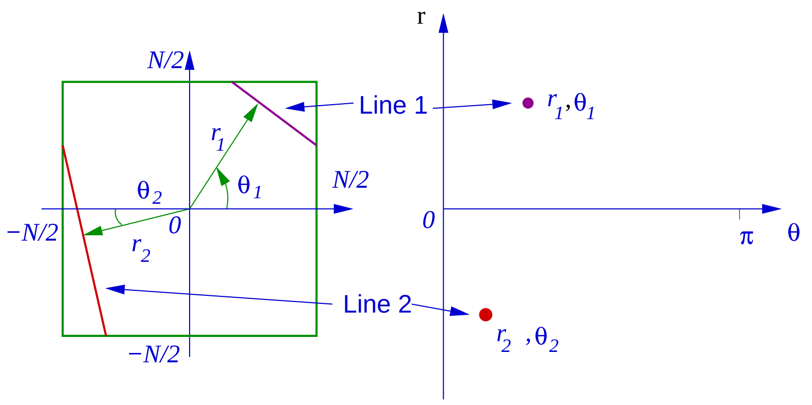


Figure 17: Polar Hough transform for multiple line detection.

First think of implementation polar Hough Transforms as a series of projections at various angles using the same projection as in collimated beam tomography as shown in figure 18. If this is then repeated at each angle to get the full Hough Transform as shown for a rectangle in figure 19. From this we see that the Hough transform is actually identical to the Radon Transform used in tomography.

Now consider the mathematics of the Hough transform rather more carefully. We know that the equation of a line at angle θ and position r is given by

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

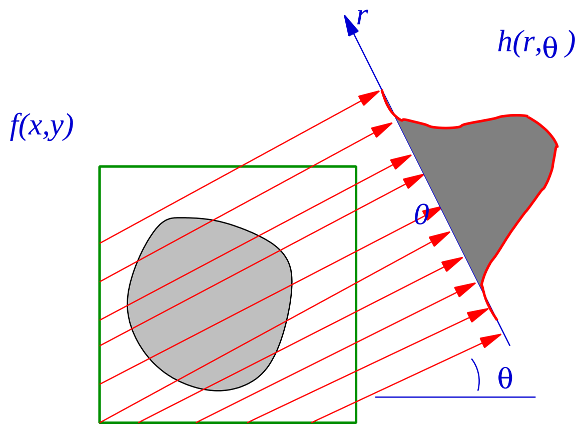


Figure 18: Polar Hough transform as a series of projections.

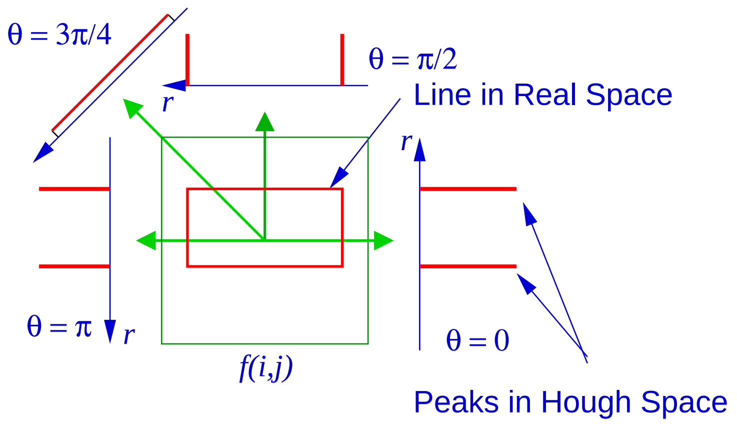


Figure 19: Polar Hough of a rectangular object.

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,as noted above, is the *Radon Transform* seen in tomography. This formulation allows us to look at and alternative visualisation and implementation.
Look at the Hough Transform of a single point at x_0, y_0, so that the *image* becomes

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

so that the Hough Transform of this is just,

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

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 as shown in figure 20.

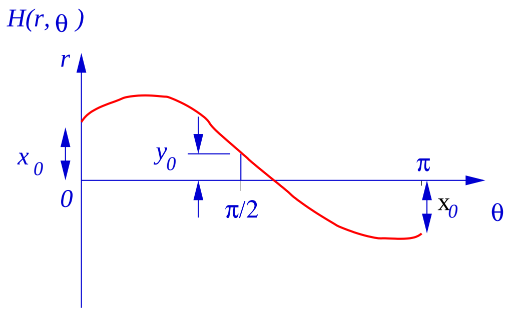


Figure 20: Hough transform of a single point.

If we now consider a line of points, then if the points form a *line* at a particular r_0, θ_0, then each $\cos()$ line crosses at one point, giving the r_0, θ_0 of the line as shown in figure 21. 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 speed-up by using a pre-calculated table to return the $\cos()$ and $\sin()$.

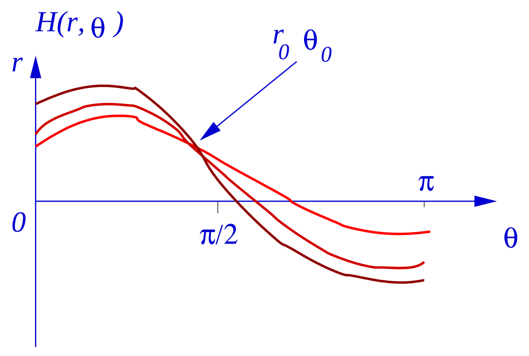


Figure 21: Hough transform of a series of points that form a line.

9.7.2 Example of Hough Transform to find lines

Firstly take the image, form the *Sobel* and threshold to get *broken lines* where the roads are as shown in figure 22. This image shows the typical problem of edge detection in the presence of noise. The roads in the original image are clearly visible to a human observer, but when detected automatically become broken and much less distinct than expected.

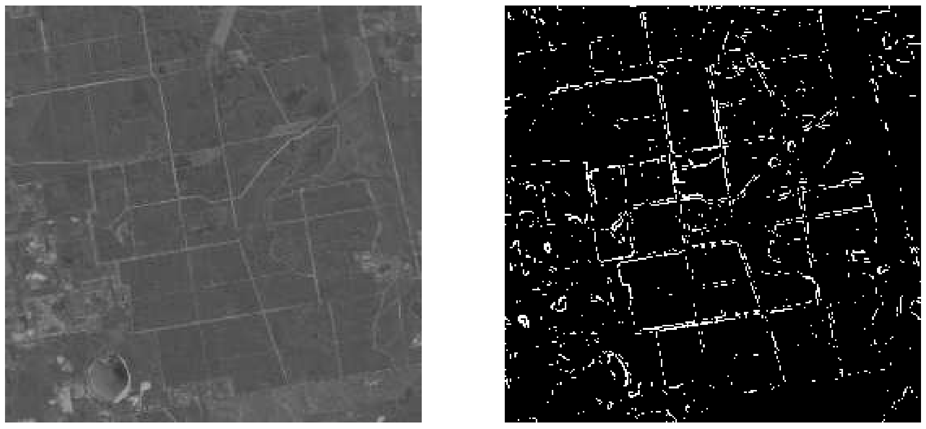

(a) Image (b) Binary edges

Figure 22: Original image and binary threshold of *Sobel* filtered to initially detect the edges.

Then form the Hough, with r on the vertical axis and θ on the horizontal is shown in figure 23. Here the lines show up a very distinct peaks where the peaks then give the *equations* of the lines in the image. This example shows that the Hough transform work very well even with rather noisy images are broken lines.

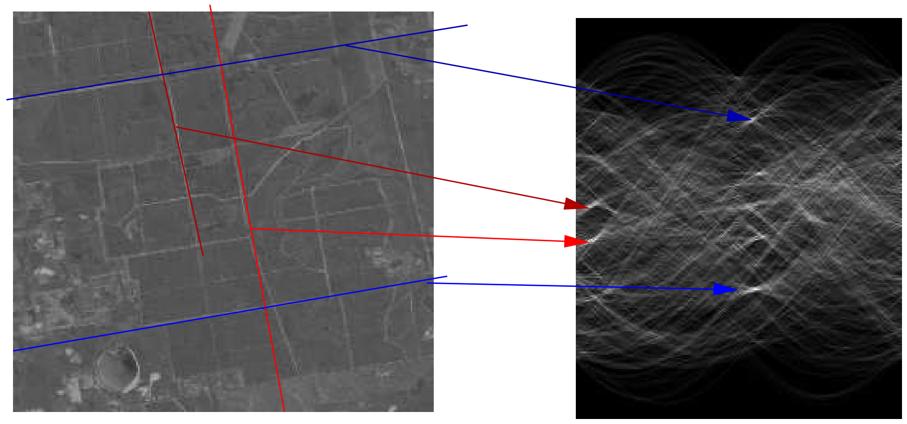


Figure 23: Hough transform of the binary edge image with the Hough peaks identifying line in the original image.

As can be seen from the above example, the 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 as shown in figure 24. The peak will have a *butterfly* shape where the extend of the *wings* is given by the length of the line and the orientation given by the location of the line in the image. It is *in principle* possible to extract the length of the line from the shape of the peak, but *in practise* this is very numerically unstable.

To detect the peak in Hough space it is usually sufficient to threshold and locate the centre. It is also possible to filer to enhance the *butterfly* shape, then threshold, or to use simple *template matching*, as discussed later. In practise this is actually fairly *easy* for good low-noise images giving a robust line detection scheme. It gives the equations of lines, but not *end-points*, so works well with simple images that contain good straight lines. It us particularly useful in robot vision where you want to detect simple geometric objects, for example to follow a track

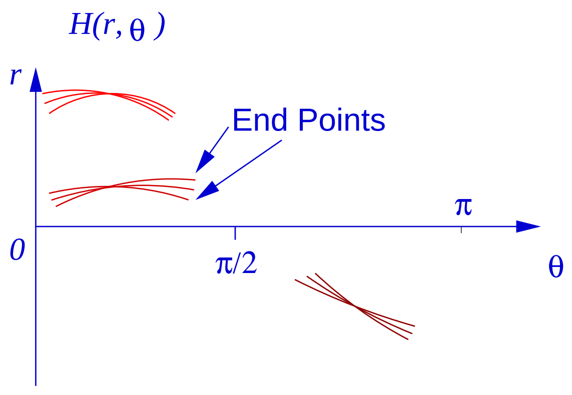


Figure 24: Shape of the Hough peak being a summation of cos() curves.

or line. It deal with broken lines very well and is also reasonably efficiently if there are "few" edge points.

There are however some problems, in that it becomes very slow of there are many edge points as you find in complex natural scenes. This also results in very complex Hough space images where are very difficult to analyse, and in particular short lines tend to get *lost* in the noise. The Hough space is also non-linear so you get different edge sensitivities in different directions and in different parts of the image.

There are also a range of extensions of the Hough Transform, for example, circle and ellipse detection by *Double Hough*, and image transform plus Hough for general shape detection, all of which are beyond this course; they also tend to be very computationally expensive.

9.8 Summary

In this section we have considered

1. First Order Differentials
2. Post Processing of Edge Images
3. Second Order Differentials.
4. LoG and DoG filters
5. Models in Images
6. Least Square Line Fitting
7. Cartesian and Polar Hough Transform
8. Mathematics of Hough Transform
9. Implementation and Use of Hough Transform