CM2208: Scientific Computing
3. Image Processing
3.2. Basic Image Processing: Applications and Examples

Prof. David Marshall

School of Computer Science & Informatics

MATLAB Image Processing Toolbox

Extensive Image Processing Facilites

- **MATLAB Image Processing Toolbox** — provides a **rich set** of **functions** for Image Processing this toolbox
  - We will look at only a **small** number here.
  - See `help/doc images` for more information.
- Lots of code builds on these for more advanced Image Processing
  - Search **MATLAB Central**
  - General Web Search for MATLAB code

Note: MATLAB Toolbox is an **add-on** toolbox. Make sure your version of MATLAB has this installed.

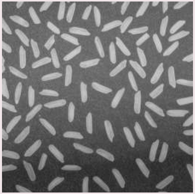
Thresholding

Basic Thresholding

Thresholding is one of the most simple Image Processing operations.

Simply stated, **Thresholding** is a way to binarise an image: Given an image$f(x,y)$ set all pixels **less than** a **threshold intensity value**, **T**, to **0**, set **all other pixels** to **1**.

$$f_{\text{thresh}}(x,y) = \begin{cases} 1 & \text{if } f(x,y) \geq T \\ 0 & \text{otherwise,} \end{cases}$$

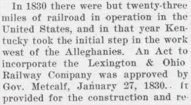


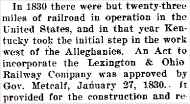
Applications of Thresholding

Examples of thresholding applications

Document image analysis — extract printed characters

- *E.g.* text, logos, graphical content, or musical scores


In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Kentucky took the initial step in the work west of the Alleghanies. An Act to incorporate the Lexington & Ohio Railway Company was approved by Gov. Metcalf, January 27, 1830. It provided for the construction and re-


In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Kentucky took the initial step in the work west of the Alleghanies. An Act to incorporate the Lexington & Ohio Railway Company was approved by Gov. Metcalf, January 27, 1830. It provided for the construction and re-

Map processing — find lines, legends, and characters

Target Detection — Recognise and track simple targets

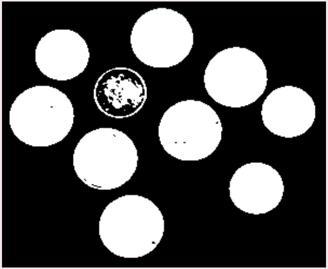
Quality inspection — simple shape measuring and delineated defective parts.

Segmentation — various image modalities for nondestructive testing

Medical Imaging

Issues with Thresholding

- **Simple cheap technique, easy implementation**
- **Output Binary image is cheaper to process**
- **How to choose the threshold?**
- **Not all images are easy to binarise**



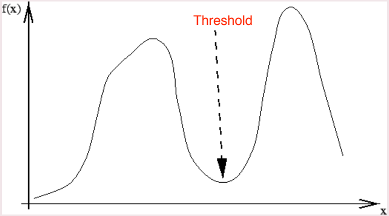
Choosing the Threshold Value (1)

How to choose the threshold values

- By Hand
- Automatically by computer

How to compute threshold value?

- Easy for bimodal distribution of pixel ntensities
  - Choose a value between peaks

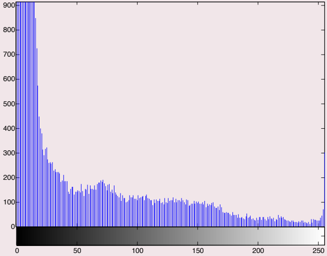


- **Harder** for other **distributions** — more typical of **interesting** images.

Choosing the Threshold Value (2)

Histogram

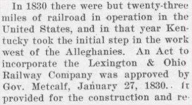
If we count the number of pixels of each intensity (or grey level) in an image, and display the result as a **histogram**

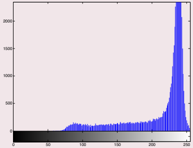


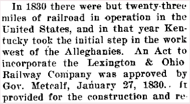
Choosing the Threshold Value (3)

Image Histogram and Threshold Examples

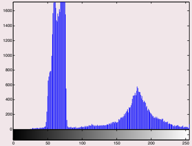
Easy Example:

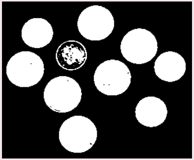

In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Kentucky took the initial step in the work west of the Alleghanies. An Act to incorporate the Lexington & Ohio Railway Company was approved by Gov. Metcalf, January 27, 1830. It provided for the construction and re-




In 1830 there were but twenty-three miles of railroad in operation in the United States, and in that year Kentucky took the initial step in the work west of the Alleghanies. An Act to incorporate the Lexington & Ohio Railway Company was approved by Gov. Metcalf, January 27, 1830. It provided for the construction and re-

Harder Example:





Thresholding in MATLAB

threshold_eg.m

```
im = imread('rice.png');
figure, imshow(im);

% Show Histogram

figure, imhist(im);

% Get MATLAB to estimate threshold value
thresh_level = graythresh(im);

% Threshold and show
im_thresh= im2bw(im,thresh_level);
figure, imshow(im_thresh);
```

See `help/doc` `imhist()`, `graythresh()` and `im2bw()` for more details.

Contrast Enhancement

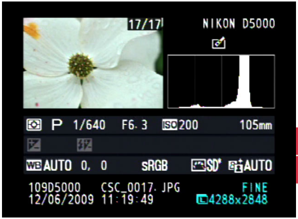
The need for Contrast Enhancement

- The human eye has a **limited sensitivity** to differences in intensity,
- Images **captured** display only a **limited range of intensities**:
  - **poor lighting**,
  - **incorrect** set-up of equipment, or
  - **various** other causes.

Contrast Enhancement

Make images more **understandable** by **increasing** the **range of differences** in **intensity** between pixel values.

- **Wider range of intensity values**



Histogram Equalisation

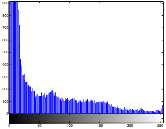
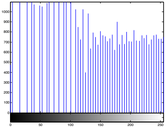
Histogram Equalisation: Basic Idea

Histogram (See previous)

- this gives us some overall information about the image.

If we now **'scale'** the histogram we can alter the intensities displayed **appropriately**.

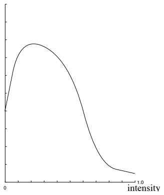
- **Changing** the **histogram appropriately** gives us an image with more **visible detail**.

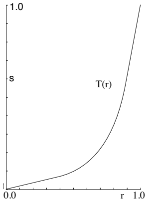

Histogram Equalisation: Algorithm (1)

Normalised Continuous Histogram Equalisation

- For simplicity for now we assume that intensities are **continuous** — We'll **modify** for discrete case later
- Also assume that the original pixel intensities lie in the range $0 \leq r \leq 1$ — **normalised range**
  - Trivial case to modify for **8-bit** **0-255** intensity ranges.
- We seek **transformations** of the type $s = T(r)$ to give a **new intensity** s for **each old intensity** r.
- For $T(r)$ to be a **reasonable** transformation, it should satisfy two conditions:
  - $T(r)$ should **increase monotonically** as r goes from 0 to 1.
    - ensures that the **order** from **black** to **white** is **preserved**.
  - $T(r)$ should **lie** between 0 and 1 for r between 0 and 1.
    - preserves range of intensities



Continuous Normalised Histogram



Illustrative Histogram Modification Function

Histogram Equalisation: Algorithm (2)

Normalised Continuous Histogram Equalisation Cont.

- Now, let $p_r(r)\,dr$ be the **probability** that an **original** pixel has **intensity** in the range r to $r + dr$
  - The **area** of a **vertical strip** of the histogram between r and $r + dr$, divided by the total area of the histogram
- Let the **equivalent probability** for a new pixel be $p_s(s)\,ds$.
- We have as each old pixel of intensity r is turned into a new pixel of intensity s , we have the relationship

$$p_s(s) = p_r(r) \left.\frac{dr}{ds}\right|_{r=T^{-1}(s)} .$$

  - The derivative dr/ds is to be evaluated at r such that $s = T(r)$.

Histogram Equalisation: Algorithm (2)

Normalised Continuous Histogram Equalisation Cont.

Aim of histogram equalisation:

Take whatever pixel intensity distribution we have initially, and to produce a new image which has equally many pixels of every shade of grey from black to white.

- This means that $p_s(s)$, the new pixel intensity distribution, must be a **constant**.

How can we choose $T(r)$ to make $p_s(s)$ constant?

Take

$$s = T(r) = \int_0^r p_r(\alpha)\, d\alpha.$$

Thus, $T(r)$ is the area under the graph of $p_r(r)$ from 0 up to r.

($p_r(r)$ is a actually **probability distribution function**)

Discreet Histogram Equalisation

Let's be discrete about things now ...

- In practice, we have only a **discrete** set of pixel **intensities**, rather than a **continuous range**,

Modified Discrete Method:

- The probability $p_r(r)$ that a pixel has a given value r, is now just n_r/n, where n_r is the **number** of pixels of **intensity** r, and n is the **total number of pixels**.
- The discrete equivalent is

$$s = T(r) = \sum_{j=0}^{r} p_r(j) = \sum_{j=0}^{r} \frac{n_j}{n}.$$

Summary

Basic Histogram Equalisation Algorithm

The **intensity** level r is in principle mapped to the new intensity level s where s is the **fraction** of **pixels** in the original image with intensities **less than** or **equal** to r.

- In practice s can must be **rounded** to the nearest permissible value (*e.g.* Integer).
- **Rescale** for range other than 0 to 1.

CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD

Equalisation Algorithm in MATLAB (1)

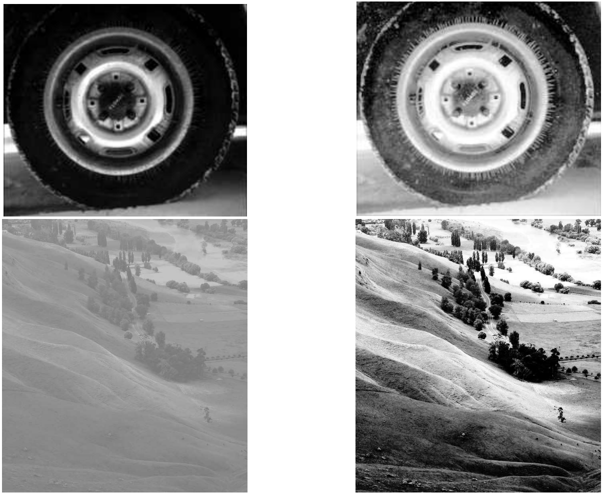
Ready Made Function: `histeq()`. Example, `hist_eq_eg.m`

```
im = imread('tire.tif'); % tire.tif is a MATLAB Example Image

% Histogram Equalise IMage
eqim = histeq(im);

% Show both Images
imshow(im)
figure, imshow(im)

% Compute and display histograms of Images
figure; imhist(im)
figure; imhist(eqim)

% do a similar process for
im = rgb2gray(imread('Unequalized_Hawkes_Bay_NZ.jpg'));
.......
```

See MATLAB `help/doc histeq()`, `imhist()` for more details.

Equalisation Algorithm in MATLAB (2)

`hist_eq_eg.m` output (partial)



Edge Detection

Definition of an Edge

An edge may be regarded as a **boundary** between two dissimilar regions in an image.

Here, **an edge** refers to an **image pixel** or **edge points** that we think **is likely to be an edge**.

The Need for Edges

Edges are Necessary

Edges are very important to any vision system (biological or machine).

- They are fairly cheap to compute.
- They do provide strong visual clues that can help the recognition process.
- **Caution**: Edges (and their robust detection) are affected by noise present in an image.

Extracting Edges from Images

Detecting an Edge

In principle an edge is **easy** to find since differences in pixel values between regions are relatively easy to calculate by considering gradients.

We have seen from our calculus lectures that:

- **First order derivatives** give us **gradients**.
- **Second Order Derivatives** show **points** of **maxima** or **minima**.

We will use **discrete approximations** of these derivatives.

Detecting Edge Points

Gradient based methods

An **edge point** can be regarded as a point in an image where a **discontinuity** (in **gradient**) occurs **across** some **line**.

Computing Gradients:
The **gradient** is a vector, whose components measure how rapidly pixel values are **changing with distance** in the x and y directions.

Thus, the **components** of the **gradient** may be found using the following **discrete approximation**:

$$\begin{aligned} \frac{\partial f(x,y)}{\partial x} = \Delta_x &= \frac{f(x+d_x,y)-f(x,y)}{d_x}, \\ \frac{\partial f(x,y)}{\partial y} = \Delta_y &= \frac{f(x,y+d_y)-f(x,y)}{d_y}, \end{aligned}$$

where d_x and d_y measure distance along the x and y directions respectively.

Note: $\frac{\partial f(x,y)}{\partial x}$ denotes the **partial differentiation** of $f(x,y)$ with respect to x **only**. Sim. for $\frac{\partial f(x,y)}{\partial y}$

CARDIFF UNIVERSITY PRIFYSGOL CAERDYD

Computing Gradients

Computing Gradients: Simplifying

In (discrete) images we can consider d_x and d_y in terms of numbers of pixels between two points. Thus, when $d_x = d_y = 1$ (pixel spacing) and we are at the point whose pixel coordinates are (i, j) we have

$$\begin{aligned} \Delta_x &= f(i+1, j) - f(i, j), \\ \Delta_y &= f(i, j+1) - f(i, j). \end{aligned}$$

Gradient Magnitude and Gradient Direction

Gradient Magnitude and Gradient Direction

In order to detect the presence of a gradient discontinuity we must calculate the **change in gradient** at (i, j).

- We can do this by finding:

  Gradient magnitude, M :

$$M = \sqrt{\Delta_x^2 + \Delta_y^2},$$

  Gradient direction, θ :

$$\theta = \tan^{-1}\left[\frac{\Delta_y}{\Delta_x}\right].$$

Implementation (1)

Convolution Masks

The difference operators, Δ_x and Δ_y correspond to convolving the image with the two **edge masks** in

-1	1
0	0

Δ_x

-1	0
1	0

Δ_y

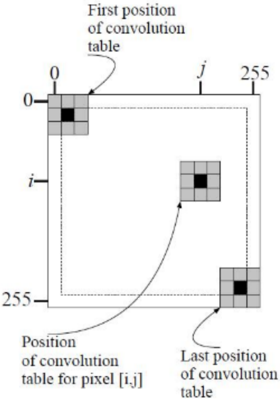
- These masks are referred to as **convolution masks** or **convolution kernels**.

Implementation (2)

Computation

This is easy to compute:

- The **top left-hand** corner of the appropriate mask is **superimposed** over **each pixel** of the image **in turn**,
- A **value** is **calculated** for Δ_x or Δ_y by using the **mask coefficients** in a weighted sum of the value of pixel (i, j) and its **neighbours**.
- Example (left) show as 3x3 mask — 3x3 (or odd size mask) usually preferred (See later)



Roberts Edge Operator

Rotate the Edge Masks by 45°

Instead of finding approximate gradient components **along** the x and y directions we can also approximate gradient components along directions at 45° and 135° to the axes respectively.
In this case the following equations are used:

$$\begin{aligned}\Delta_1 &= f(i+1, j+1) - f(i, j), \\ \Delta_2 &= f(i, j+1) - f(i+1, j).\end{aligned}$$

The corresponding **convolution masks** are given by:

Δ_1:

0	1
-1	0

Δ_2:

1	0
0	-1

This form of operator is known as the **Roberts edge operator** and was one of the first operators used to detect edges in images.

Derivatives over a 3x3 Grid

3x3 Masks

3x3 Masks are preferred:

- Larger Masks (even bigger than 3x3) are less **prone** to **noise** (**Local Averaging**)
- Values can be computed around a **central pixel**.

Consider the arrangement of pixels about the pixel (i, j):

a_0	a_1	a_2
a_7	**[i,j]**	a_3
a_6	a_5	a_4

The partial derivatives can be computed by:

$$\begin{aligned} \Delta_x &= (a_2 + \mathbf{c}a_3 + a_4) - (a_0 + \mathbf{c}a_7 + a_6) \\ \Delta_y &= (a_6 + \mathbf{c}a_5 + a_4) - (a_0 + \mathbf{c}a_1 + a_2) \end{aligned}$$

- The constant **c** implies the emphasis given to pixels closer to the center of the mask.

Sobel Edge Operator

Sobel Edge Operator

One important edge operator of this type is the **Sobel edge operator**.

Setting $c = 2$ the **Sobel edge operator masks** are given by:

$$\Delta_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \Delta_y = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

Note: Setting $c = 1$ gives the **Prewitt edge operator**

Second Order Methods

The *Laplacian* operator

All of the previous edge detectors have approximated the **first order derivatives** of pixel values in an image.

It is also possible to use **second order derivatives** to detect edges.

A very popular second order operator is the **Laplacian operator.**

The **Laplacian** of a function $f(x, y)$, denoted by $\nabla^2 f(x, y)$, is defined by:

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

Using **discrete difference approximations** to estimate the derivatives gives the **Laplacian operator convolution mask**:

0	1	0
1	-4	1
0	1	0

Edge Detectors and Noise Smoothing/Filtering

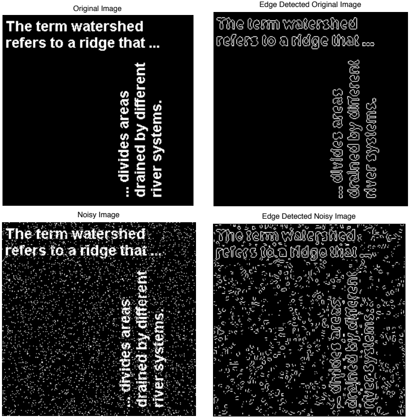
Noisy Edges and No Direction

However there are disadvantages to the use of second order derivatives.

- We should note that first derivative operators **exaggerate** the effects of **noise**.
- Second derivatives will **exaggerated noise twice** as much.
- **No** directional information about the edge is given.

Laplacian of a Gaussian (LOG) Edge Detector

- Employ **Noise Smoothing** — **Gaussian Kernel**.
- **Then apply** the **Laplacian**.



MATLAB: noisy_edge_eg

MATLAB Edge Detection

The edge() Function

This function implements all the edge filters with have described above plus another one

Simple examples:

```
im = imread('text.png');

edge_im = edge(im,'roberts');
edge_im = edge(im,'sobel');
edge_im = edge(im,'prewitt');
edge_im = edge(im,'log');
edge_im = edge(im,'zerocross');
```

See MATLAB `help/doc edge` for more details.

The Hough Transform

Joining Edge Pixels

Having detected edge pixels it is useful the to try and **join** these **together** to form true edge **lines** or **contours**.

One powerful method for detecting edges is called the **Hough transform**.

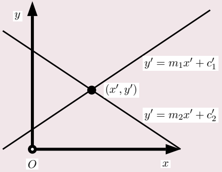
The Hough Transform Concept

Let us suppose that we are looking for **straight lines** in an image.

If we take a point (x', y') in the image, **all lines** which **pass** through that **pixel** have the form

$$y' = mx' + c$$

for varying values of m and c.



Let's Turn this Equation Around

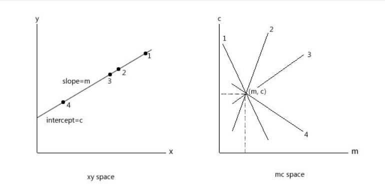
(m, c) space: A *function* in m and c?

However, this equation can **also** be written as

$$c = -x'm + y'$$

where we now consider x' and y' to be **constants**, and m and c as **varying**.

This is a straight line on a graph of c against m as shown.



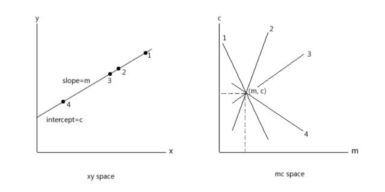
Each different line through the **point** (x', y') corresponds to **one** of the **points** on the **line** in (m, c) **space**.

Hough Transform Algorithm (1)

Developing an Algorithm

Now consider **two** pixels p and q in (x, y) space which lie on the **same line**.

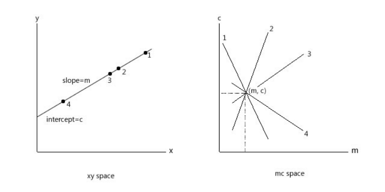
- For **each** pixel, **all of the possible lines** through it are represented by a **single line** in (m, c) **space**.
- Thus the **single line** in (x, y) **space** which goes through **both pixels lies** on the **intersection** of the **two lines** representing p and q in (m, c) **space**:



Hough Transform Algorithm (2)

Taking this one step further

- **All** pixels which **lie** on the **same line** in (x, y) **space** are represented by lines which **all pass** through a **single** point in (m, c) **space**.
- The **single point** through which they **all pass** gives the values of m and c in the equation of the line $y = mx + c$.
- So for every pixel on a line we can accumulate points in (m, c) **space**
  - Every valid pixels will add one vote.



Hough Transform Straight Line Detection

To detect **straight lines** in an image

1. **Quantise** (m, c) **space** into a **two-dimensional array** A for appropriate steps of m and c.
2. **Initialise all** elements of $A(m, c)$ to **zero**.
3. For **each pixel** (x', y') which **lies** on **some edge** in the image, we **add 1** to **all elements** of $A(m, c)$ whose indices m and c satisfy $y' = mx' + c$.
4. **Search** for **elements** of $A(m, c)$ which have **large accumulated values** — **Each one found** corresponds to a **line** in the **original image**.

Note: The Hough Transform is **global** as it will find **all lines** in an image

A Practical Problem

We have a slight problem with our formulation of a line

One **practical detail** is that the $y = mx + c$ form for representing a straight line **breaks down** for **vertical lines**, when m becomes **infinite**.

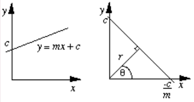
- We have to loop over values of $-\infty \leq m \leq +\infty$!

An alternative representation of a line

An **alternative representation** of a line is

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

where r is the perpendicular distance from the line to the origin and θ is the angle the line makes with the **x-axis**:



(a) $y = mx + c$ equation of line (b) Alternative form of line equation

Hough Transform in (r, θ) Space

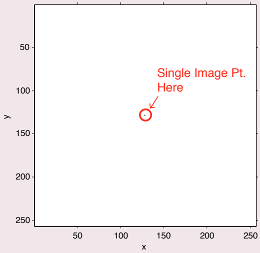
Lines in (r, θ) Space

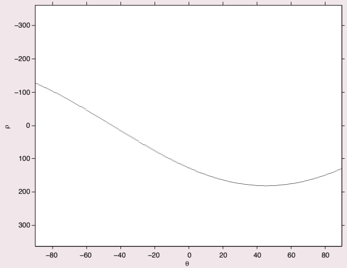
The $r = x\cos\theta + y\sin\theta$ form has the **advantage** that the **gradient** m, with a range $-\infty \leq m \leq +\infty$ has been replaced by the range of angles $0 \leq \theta \leq \pi$.

- This is **much easier** to deal with **computationally**.

So we now have an **efficient Hough Transform Algorithm**.

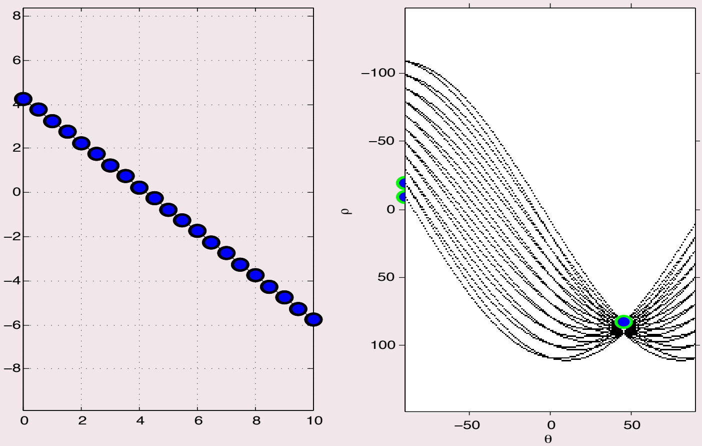
- Note, however, that a **point** in (x, y) space is now represented by a **curve** in (r, θ) space **rather** than a **straight line**.
- Otherwise, the method is unchanged.





Hough Transform Animation

hough_line_anim_demo.m



Hough Transform Circle Detector

The Hough transform can be used to detect **other shapes** in an image as well as straight lines.

Hough Transform Circle Detection

For example, if we wish to find circles, with equation

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

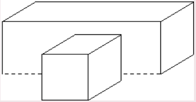
Now:

- Every point in (x, y) space corresponds to a surface in (a, b, r) space (as we can vary any two of a, b and r, but the third is determined by above equation.
- The basic Hough Transform method is, thus, **modified** to use a **three-dimensional array** $A(a, b, r)$,
- **All** points in it which **satisfy** the **equation for a circle** are **incremented**.

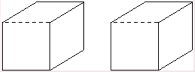
Issues with Hough Transform

Advantages/Disadvantages

- The technique takes rapidly increasing amounts of time for more complicated curves as the **number of variables** (and hence the number of **dimensions** of A) **increases**
  - Really only of use for simple algebraic curves.
- **Global Shape Detector** — Detects **whole** shapes
  - The line need not be **contiguous**



  - On the other hand, it can also give misleading results when objects happen to be **aligned** by chance, as shown by the two dotted lines



MATLAB Hough Transform Line Detections

Three MATLAB Functions for the Complete Line Detection

`hough()` — **compute** the **Hough transform**

`houghpeaks()` — **detect** the **peaks** in the **Hough transform** in **(r,θ)** space

`houghlines()` — **Extract line segments** based on **Hough transform**

See `help` **hough()**, **houghpeaks()** and **houghlines()** for full details.

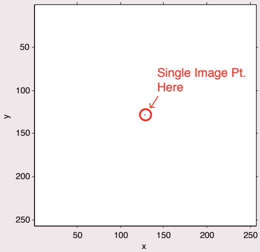
Performing the Hough Transform

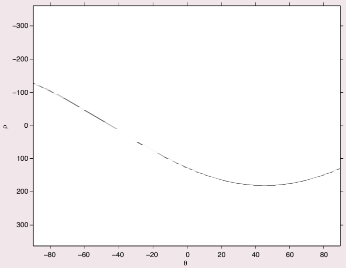
hough_rhotheta_pt_eg.m

```
% Hough Transform of a Point
% Create an image with a Point
M= 256; N = 256;
im = zeros(M,N);
x = 128, y = 128
im(x,y) = 1;

imshow(1-im); % Invert image for plots
axis on, axis normal;
xlabel('x'), ylabel('y');

[H,T,R] = hough(im);

% Display the Hough Transform
figure
imshow(imadjust(mat2gray(1-H)),'XData',T,'YData',R,...
'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');

axis on, axis normal, hold on;
colormap(hot);
```

hough_rhotheta_pt_eg.m Output

hough_rhotheta_pt_eg.m Output

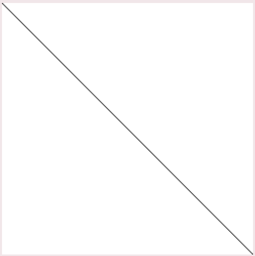


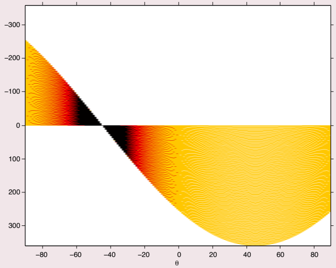


Another Basic Hough Example: A Single Line

See also hough_line_eg.m

```
% Create an 'image' with a diagonal line
M= 256; N = 256;
im = zeros(M,N);
figure, imshow(im), hold on

plot([0 M],[0 N],'w');

F=getframe;
im = F.cdata(2:M-1,2:N-1);

..... do Hough Transform as before
```





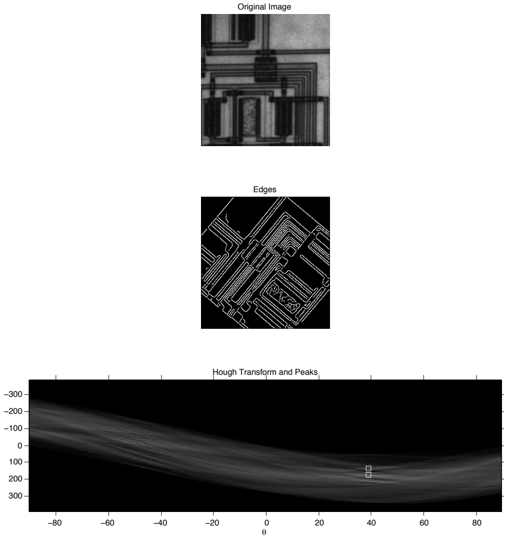
Detecting and Plotting Peaks in the Hough Transform

hough_peak_eg.m

```
.... Compute Hough Transform as before
[H,T,R] = hough(BW);
P  = houghpeaks(H,2);


... Display the original image and edges
% Display the Hough Peaks.

imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
plot(T(P(:,2)),R(P(:,1)),'s','color','white');
title('Hough Transform and Peaks');
```

- `[H,T,R] = hough(BW)` — Returns the **values** of θ, `T` and r, `T`.
- `P = houghpeaks(H,2)` returns, in the example, the `(T,R)` locations of the highest **2 peaks**
- Simply `plot` these locations out — use a square box to show location on plot.

hough_peak_eg.m Output



Extracting Lines from The Hough Transform

Finally some Edge Lines, houghlines_eg.m

```
... Do Hough Transform and Calculate Peaks as before

lines = houghlines(BW,T,R,P,'FillGap',5,'MinLength',7);

.... Plot out detected lines
```