Chapter 4: Image Segmentation

Andre L. C. Barczak

Computer Science
Massey University, Albany

Contents

Feature Detection

Introduction

We want to detect geometric features: lines, corners, circles, edges
In some cases combine, i.e., find the edges and then find the circles

Edge Detection

Convolution masks help us: Sobel, Laplacian, etc. Lets see a special case: zero crossings

Zero Crossings

The second derivative of a function crosses zero when the slope changes suddenly. We can use this as a simple clue to where edges may be present.

a	b
d	c

Figure 1: Zero crossings 'kernel'.

if $(a - 128) < -T$ or.$(d - 128) < -T$ or $(b - 128).(c - 128) < -T$

then there is an edge at a.

T must be set for optimisation.

Zero Crossings

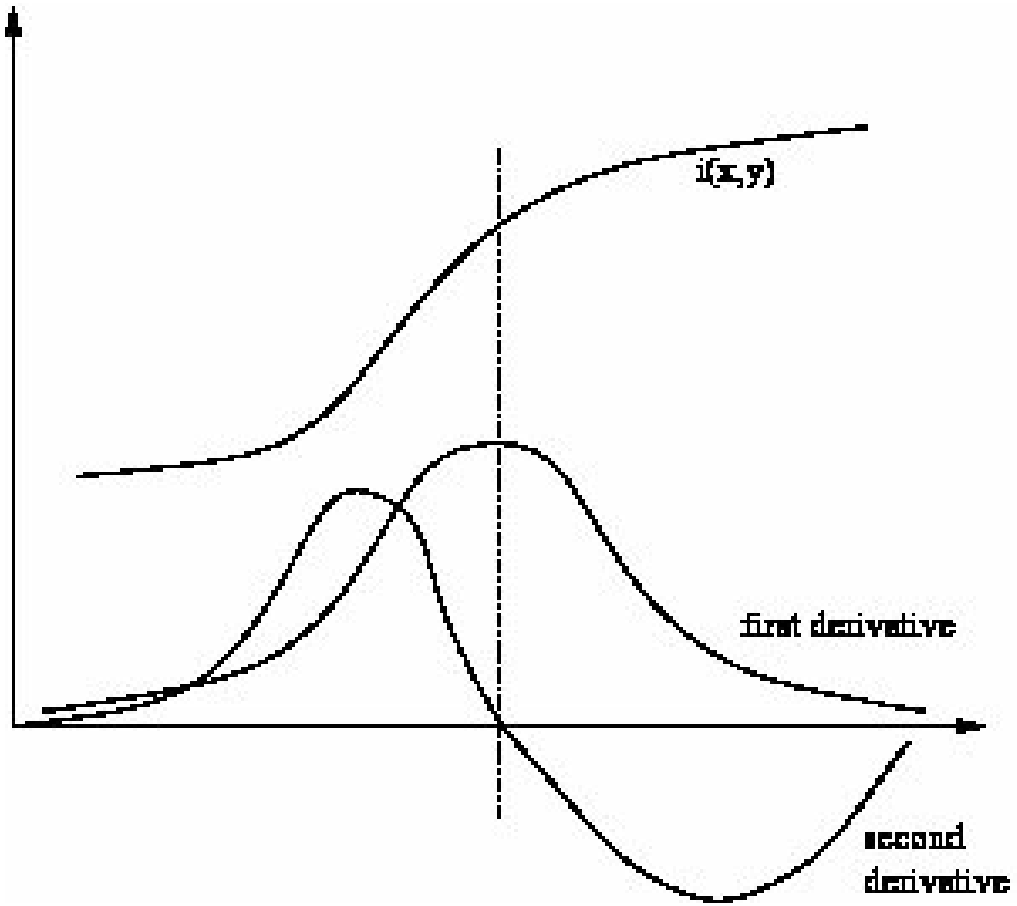

Figure 2: First derivative and Laplacian for a generic function.

Gaussian

A filter that mimics a Gaussian function blurs the image. We can a Gaussian as a 2D equation:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \tag{1}$$

0.118	0.235	0.118
0.235	0.469	0.235
0.118	0.235	0.118

$\xrightarrow{\text{round}}$ 1/16 $\times$

1	2	1
2	4	2
1	2	1

Figure 3: A Gaussian with a 3x3 kernel ($\sigma = 0.85$)

Laplacian of Gaussian (LoG)

- Compute the Gaussian of the image
- Compute the Laplacian of the image

This causes the weaker edges to "smooth" out, as the Gaussian blurs the image before the Laplacian is computed.

Laplacian of Gaussian (LoG)

The LoG can be computed by a single kernel:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} * \begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix} \longrightarrow \frac{1}{16}\begin{bmatrix}1 & 3 & 4 & 3 & 1\\ 3 & 0 & -6 & 0 & 3\\ 4 & -6 & -20 & -6 & 4\\ 3 & 0 & -6 & 0 & 3\\ 1 & 3 & 4 & 3 & 1\end{bmatrix}$$

Figure 4: LoG 5x5 kernel

Laplacian of Gaussian (LoG)

Compare the two kernels. They look similar (one of them is scaled), but they are not equivalent.
Rounding up, as well as the difference in size, causes the results to be different.

Figure 5: a) Gaussian followed by Laplacian b) 5x5 kernel

Diffence of Gaussians (DoG)

The DoG is a filter where two Gaussians with different σs are subtracted from each other.
The original image is used for each Gaussian computation.
This is an approximation of the LoG, and it is widely used because it is vary fast.
E.g., a method called SIFT (can be a topic for the seminars!) uses the DoG.

Many OpenCV functions...

- cvLaplace()
- cvSobel()
- cvCanny()

cvLaplace()

cvLaplace(src,dst,aperture);
We need to convert the image to gray scale first.
e.g.,
cvCvtColour(image, reimage, CV_BGR2GRAY);
a single tunnable parameter.

uses the kernel $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

cvLaplace()

This function in OpenCV yields an unexpected result.
To avoid overflow, it uses a 16S pixel (or 32)
You need to convert back to 8U (1 byte per pixel)
use: cvScale(image2,auximage,1,0);

cvLaplace()

Image result with cvLaplace(auximage,image2,3);

Figure 6: Laplacian in OpenCV with kernel size 3x3.

cvSobel()

void cvSobel(src, dst, xorder, yorder, aperture);
order is the order of the derivative for either x or y.

cvSobel()

cvSobel(src,dst,1,0,3) corresponds to the kernel:

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

cvSobel(src,dst,0,1,3) corresponds to the kernel:

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

cvSobel()

Image result with cvSobel(auximage,image2,1,1,5);

Figure 7: Sobel in OpenCV with kernel size 5x5.

cvCanny()

```
cvCanny(src, dst, threshold1, threshold2, aperture);
```

in OpenCV, Canny produces a binary image.
The binary image's edges are as thin as possible.
Compare with `cvLaplace()` and `cvSobel()`.

cvCanny()

Canny is an algorithm, not a simple kernel.
6 simple steps:

1. smooth the image (Gaussian)
2. find edges (Sobel)
3. find gradient directions
4. find gradient with 90 degrees
5. find thin lines along edges
6. use hysteresis to make continuous lines (hence the two thresholds). Pixels above the high threshold are accepted. Pixels below the lower threshold are rejected. Pixels in between are accepted if they are connected to a strong (above the high threshold) gradient.

cvCanny()

Image result with cvCanny(auximage,image2,1,100,3)

Figure 8: Canny in OpenCV with kernel size 5x5.

Hough Transform

Straight lines

The idea behind this method can be explained in four steps (Gonzalez and Woods):

1. Obtain a binary image, the edges marked with max value (1 or 255)
2. Create a ρ,θ plane to represent the straight lines passing by point (x,y)
3. Extract the points of the ρ,θ plane that are above a certain threshold
4. Examine the continuity of the chosen lines

Hough Transform

Straight lines

Step 1: e.g., Canny detector...
Step 2: A straight line passing through point (x,y) can be represented by:

$$xcos(\theta) + ysin(\theta) = \rho \tag{2}$$

Step 3: use an accumulator to find possible lines.
Step 4: consider linking lines.

Hough Transform

Parametrised edges

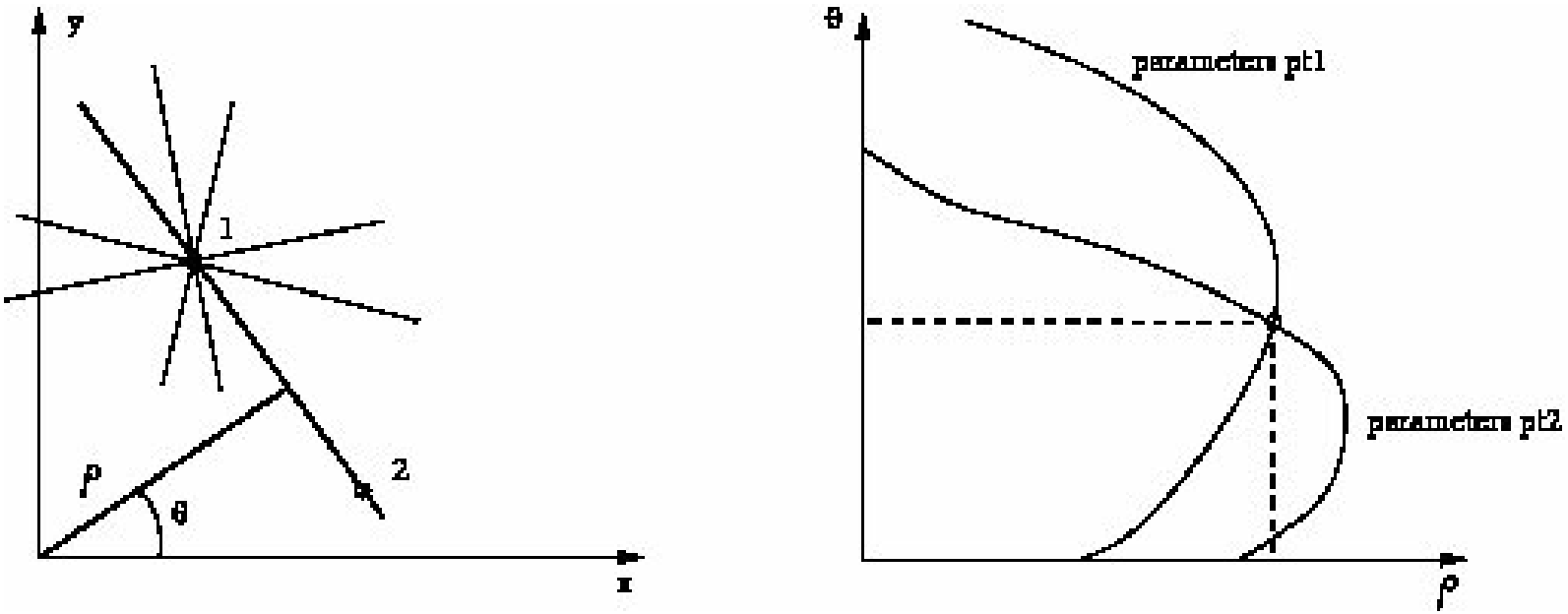

Figure 9: Lines for points 1 and 2 are plotted in polar coordinates. Where the curves meet there is a line to which both points belong to.

Hough Transform

Image example

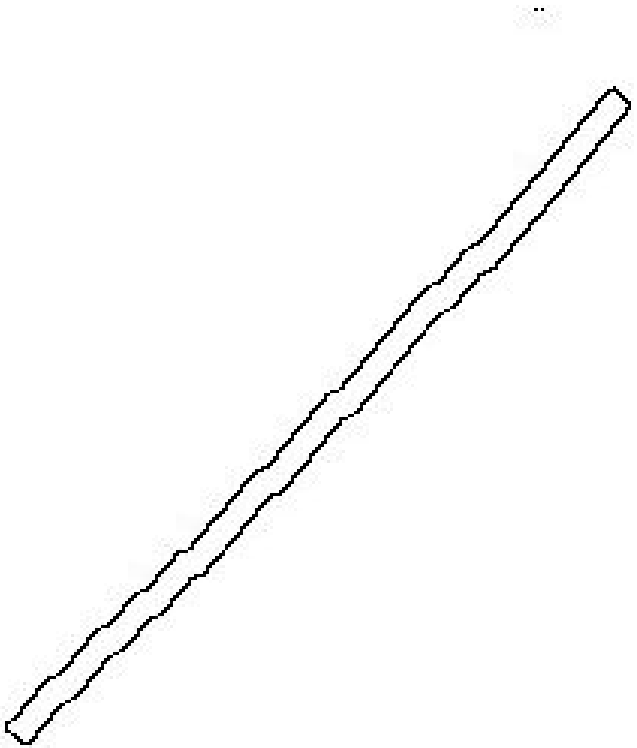

Figure 10: A binary image with two main straight lines.

Hough Transform

Accumulator

Figure 11: This accumulator uses an image to store the values.

Hough Transform in OpenCV

function

```
CvSeq* cvHoughLines2(image, accumulator, method, rho,
theta, threshold, param1, param2);
```

Returns a sequence of lines represented by rho,theta.
One needs to read the sequence and draw the lines.

Hough Transform example

Office image

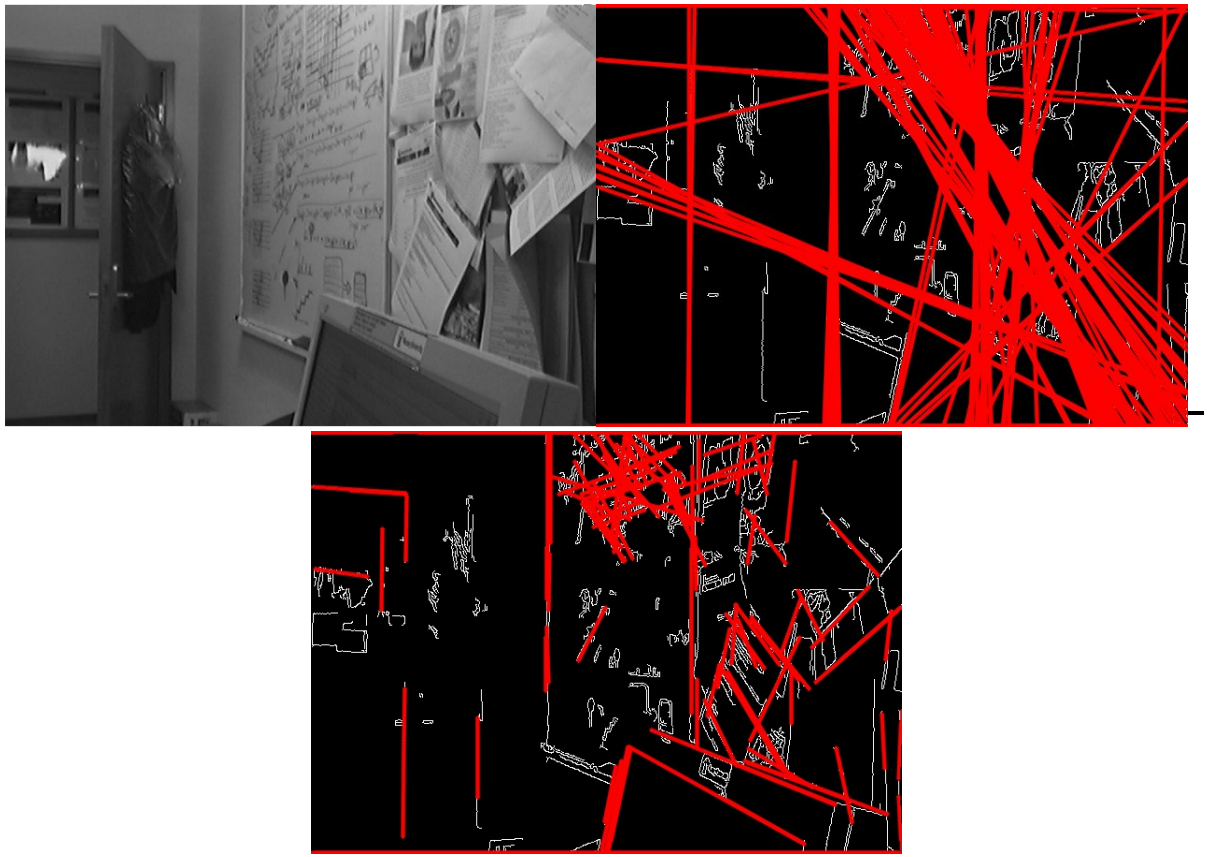

Figure 12: Houghlines2(): a) original b) infinite lines c) segments of lines

Hough Transform

Circles

Step 1: e.g., Canny detector...
Step 2: The family of circles with centre x_c, y_c and radius R:

$$(x - x_c)^2 + (y - y_c)^2 = R^2 \tag{3}$$

Step 3: use a 3D accumulator to find possible circles.
Step 4: choose strongest points in the accumulator.

Hough Transform in OpenCV

function

HoughCircles(image, circles, method, dp, minDist, param1, param2, minRadius, maxRadius);
Returns a vector with the circle parameters.

HoughCircles()

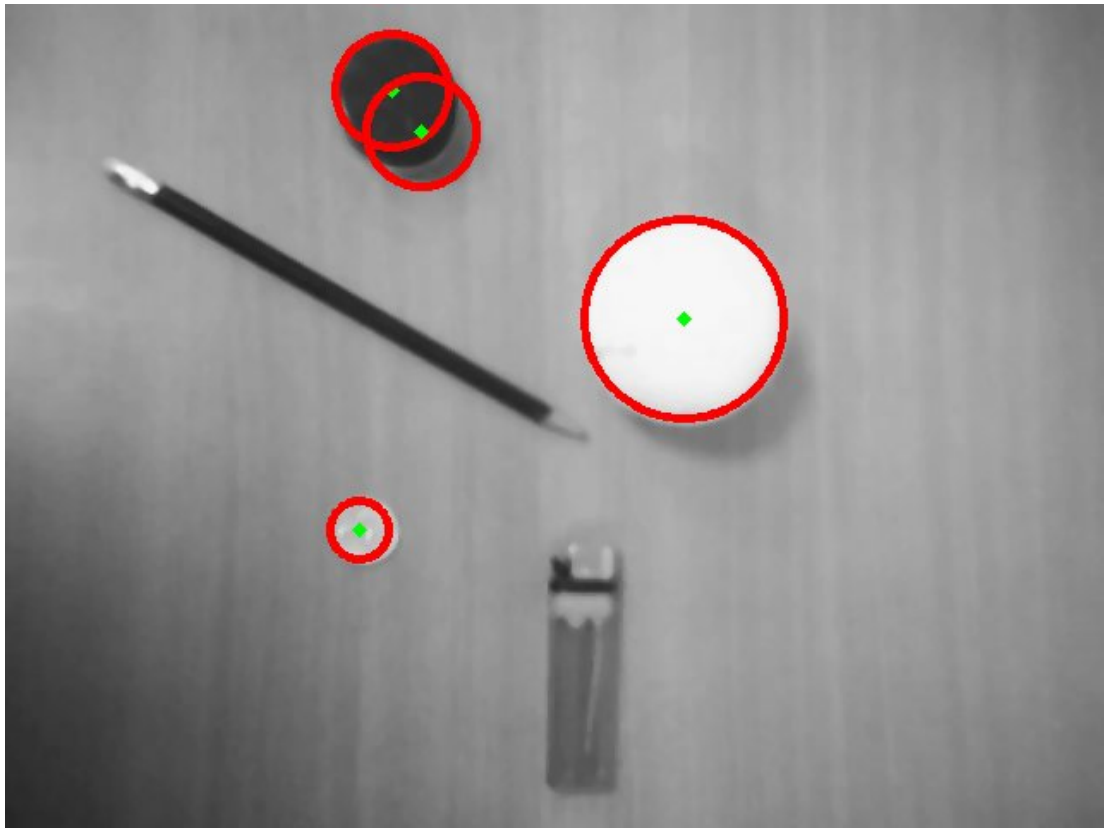

Figure 13: Example of false positive.

cvFindContours()

Blob finder again...

OpenCV offers an alternative for the connectivity finder:
cvFindContours(image, storage, seq, sizeof, mode, method, offset)
Of all parameters, *mode* and *method* are important to choose.

cvFindContours()

MODE

How the contours are presented and stored:
CV_RETR_EXTERNAL: outer contour par CV_RETR_LIST: all of them (holes)
CV_RETR_CCOMP: two level hierarchy (holes)
CV_RETR_TREE: nested contours (complex objects)

cvFindContours()

METHODS

The method used to link the points:
CV_CHAIN_CODE: Freeman chain code
CV_CHAIN_APPROX_NONE: points
CV_CHAIN_APPROX_SIMPLE: compresses segments
CV_LINK_RUNS: link horizontal segments

cvFindContours()

Examples:

A hand image (binary):

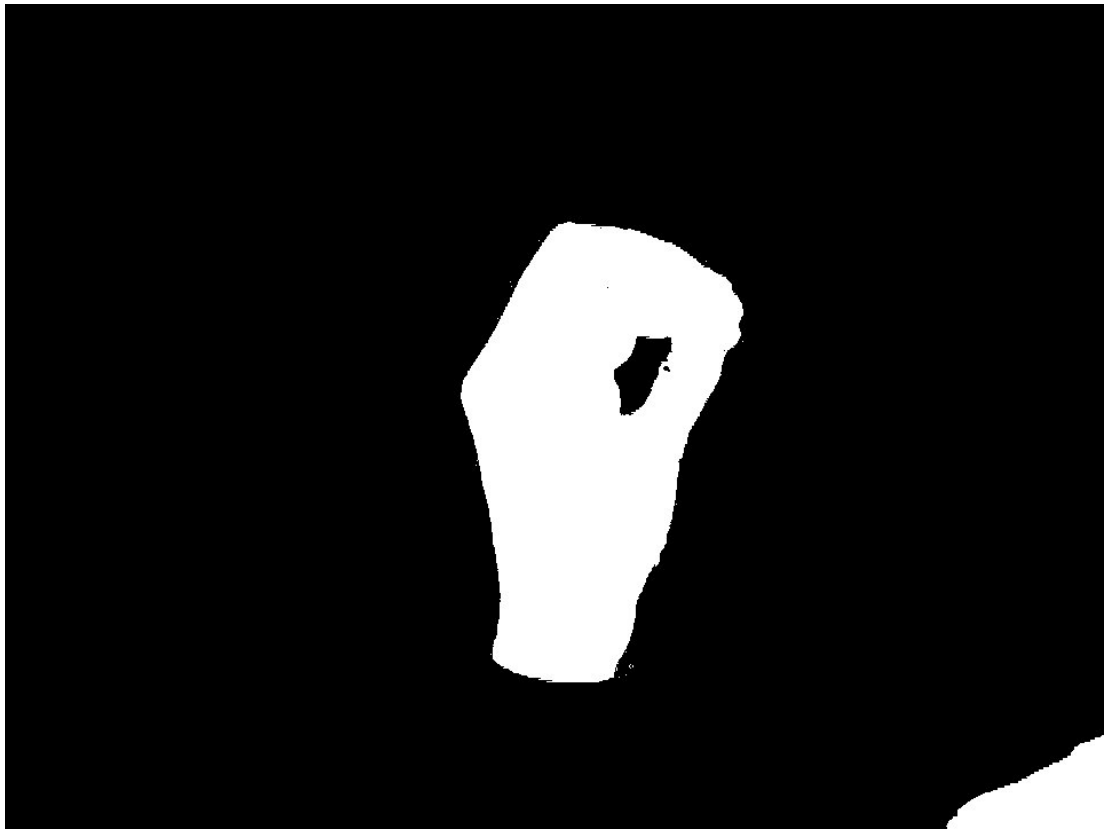

Figure 14: Binary image.

cvFindContours()

Examples:

CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE

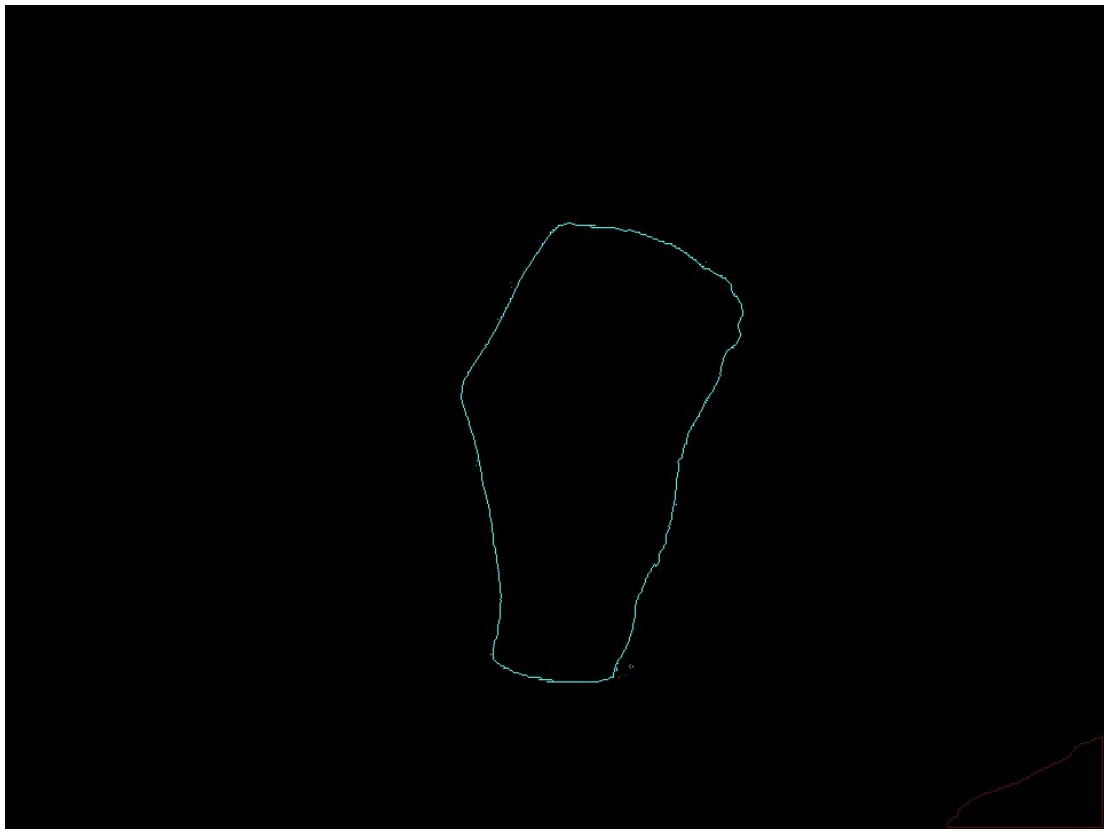

Figure 15: Contour 1.

cvFindContours()

Examples:

CV_RETR_LIST, CV_CHAIN_APPROX_NONE

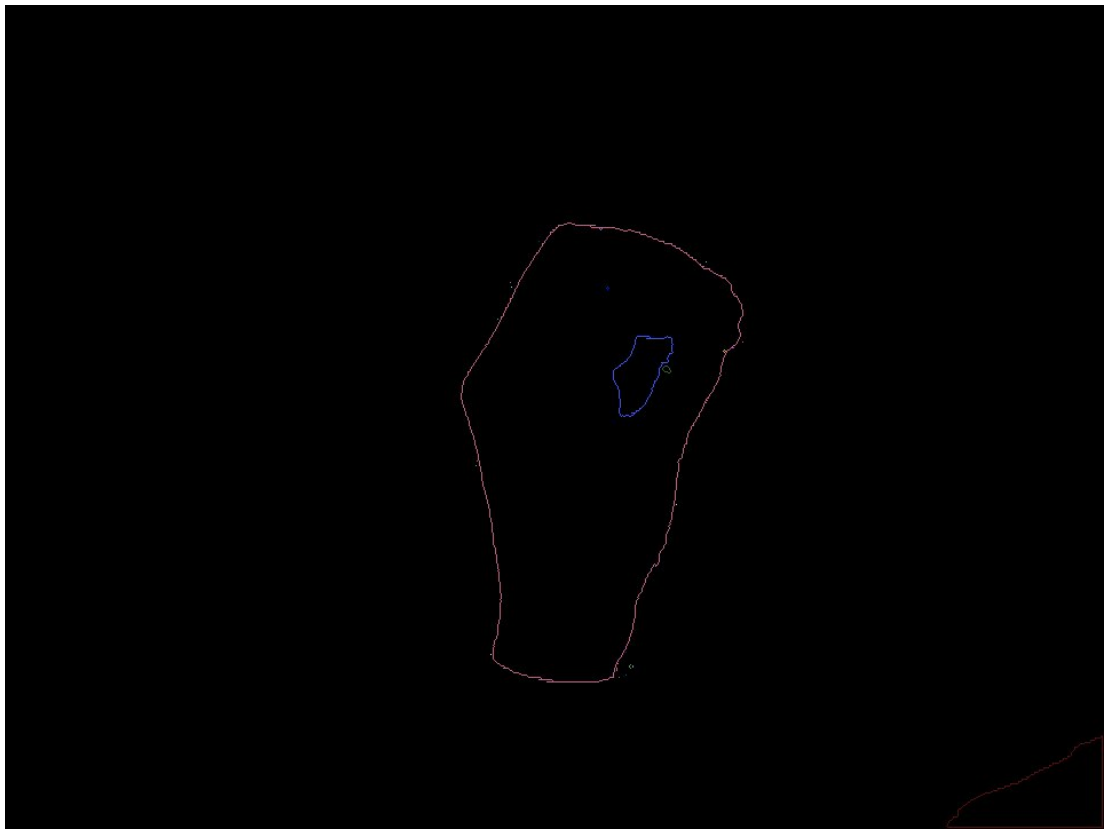

Figure 16: Contour 2.

Watershed Segmentation

The idea.

Segmentation is difficult without knowledge of the objects.
Watershed: imagine the image being a 3D topographic map.
Fill it with imaginary water...
and check for watersheds.
cvWatershed() implements Meyer's method.

Watershed Segmentation

The idea.

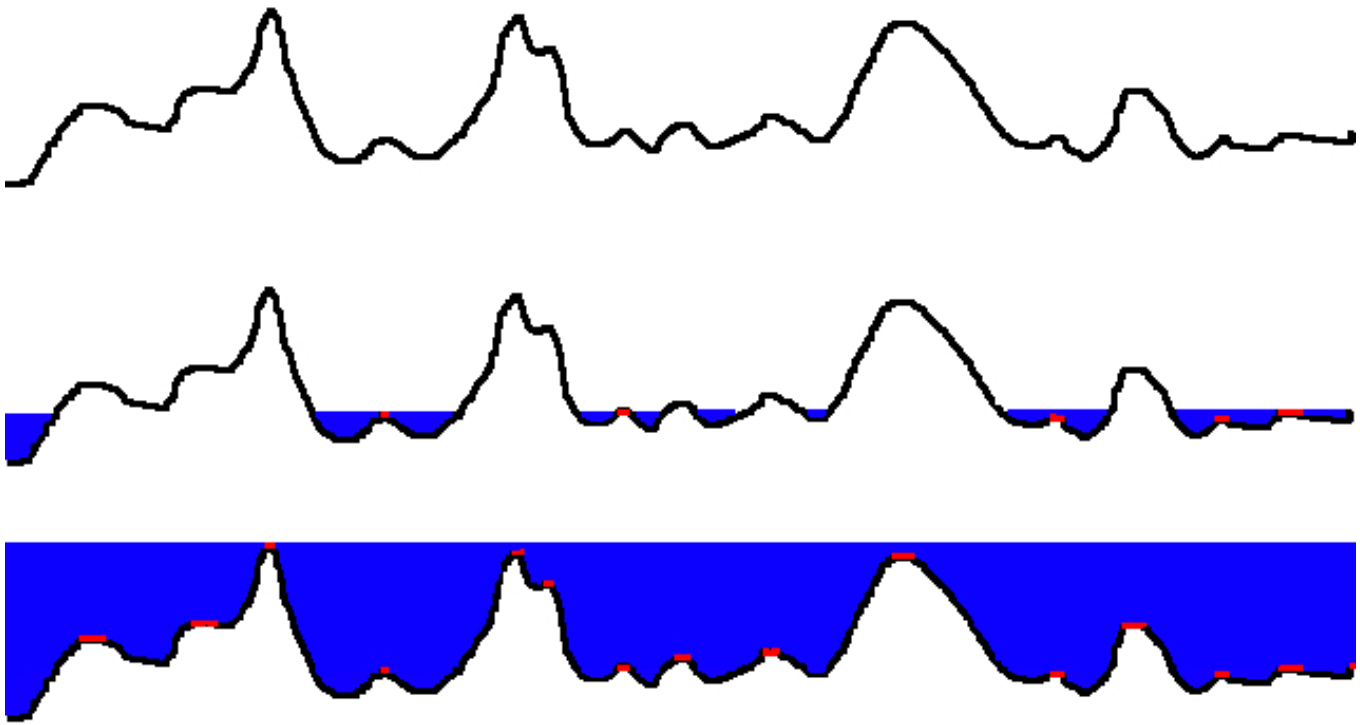

Figure 17: Marking the edges as the imaginary water fills up the regions.

Watershed Segmentation

Examples:

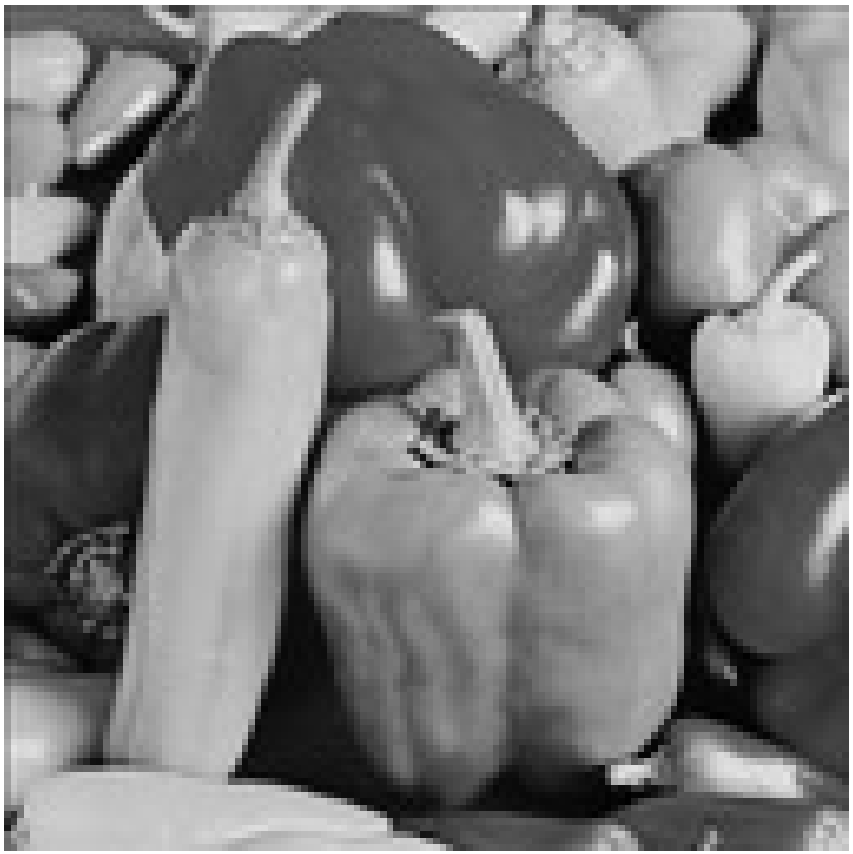

Figure 18: Watershed segmentation example.

Watershed Segmentation

Examples:

Figure 19: Watershed segmentation example.

1. Write (or just modify) code to compute the LoG of a greyscale image using two options: a) a single 5x5 kernel. b) two 3x3 kernels. Compare the images. Are they identical? Compare the runtime for the two approaches using a large image (use the time utility in Linux).
2. Write (or modify) the code for edge detection. Take an argument to choose from: Sobel, Laplacian or Canny. Note that the resulting image may be of different nChannels for different OpenCV functions.
3. Test the Hough transform using cvHoughLines (or the equivalent) using different images. What happens when the thresholds are changed?
4. Write your own Hough transform code for a straight line. What considerations are you making regarding the size of the accumulators? How these decisions affect the performance of the code?

Bibliography

(partial)


Gary Bradski and Adrian Kaehler, Learning OpenCV, O'Reilly, 2008.


Gonzales and Woods, Digital Image Processing, Prentice Hall, 2002.