BLG453E COMPUTER VISION

Fall 2017 Term
Week 9

İstanbul Technical University
Computer Engineering Department

Instructor: Prof. Gözde ÜNAL

Teaching Assistant: Enes ALBAY

Thanks: Some Slides in this presentation are from the lecture notes of Prof. Greg Slabaugh at City University of London

Learning Outcomes of the Course

Students will be able to:

1. Discuss the main problems of computer (artificial) vision, its uses and applications
2. Design and implement various image transforms: point-wise transforms, neighborhood operation-based spatial filters, and geometric transforms over images
3. Define and construct segmentation, feature extraction, and visual motion estimation algorithms to extract relevant information from images
4. Construct least squares solutions to problems in computer vision
5. Describe the idea behind dimensionality reduction and how it is used in data processing
6. Apply object and shape recognition approaches to problems in computer vision

Week 8-9: Image Segmentation

At the end of Week : Students will be able to (Learning Objectives of the week):

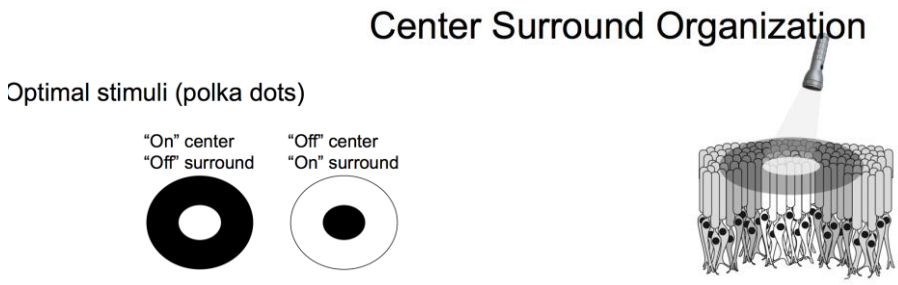
3. Define and construct segmentation, feature extraction, and visual motion estimation algorithms to extract relevant information from images

Prank: https://www.youtube.com/watch?v=wFa6NIgSRKQ

Green Screen

https://en.wikipedia.org/wiki/Chroma_key

Center Surround Organization in the Eye Neurons



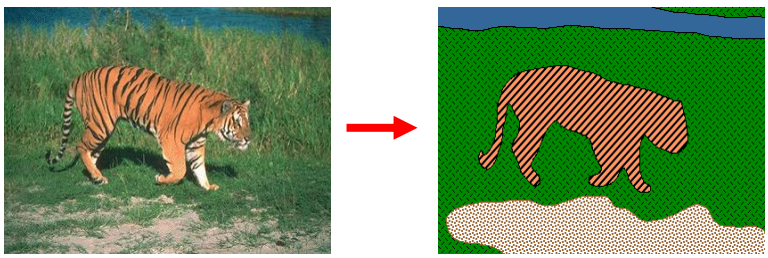
Some neurons receive net excitatory in the center and net inhibitory from the surround, and others receive net inhibitory in the center and net excitatory synapses in the surround.

Video min: 1:04: http://www.youtube.com/watch?v=KE952yueVLA

As a result: retinal ganglion cell are more sensitive to patterns of light that are not uniform, but are changing in a way that, that matches their center surround organization
That can make them most responsive to light spots on a dark background, which is referred to as having an on center and off surround, or dark spots on a light background, which is referred to as having an off center and on surround.
This is a step towards identifying boundaries of objects (happens later in visual cortex)

Prof. J. Groh, Duke University

From Images to Objects/Regions: Segmentation



- Segmentation partitions an image into regions of interest.
- The first stage in many automatic image analysis systems.
- A complete segmentation of an image domain is a finite set of regions R_1, ..., R_N, such that $W = \bigcup R_i$

Segmentation

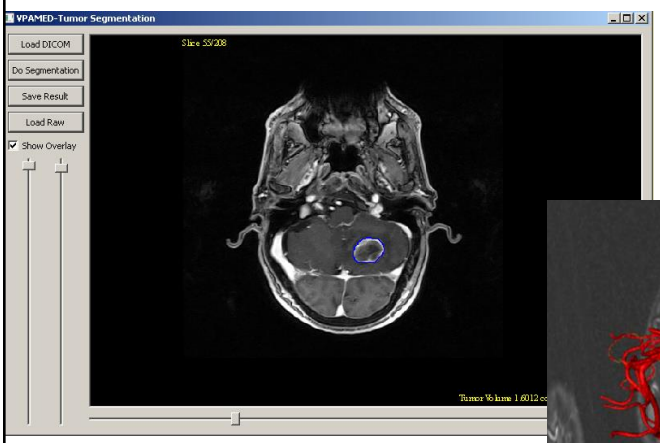
- Segmentation has a variety of practical uses

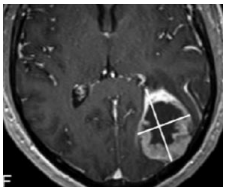
cut out

recolouring
(courtesy of Tommaso)

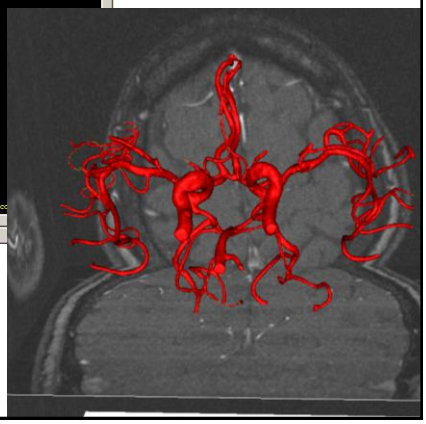
compositing

Segmenting anatomical structures





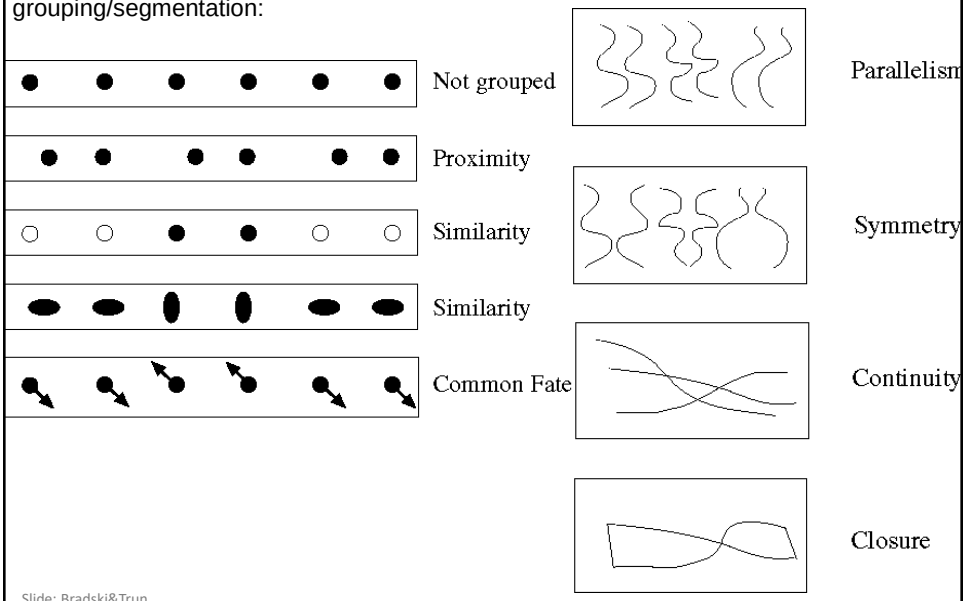
measurement



Some results from
Dr. Andac Hamamcı and Dr. Suheyla Cetin's PhD thesis results at Prof. G. Unal's group

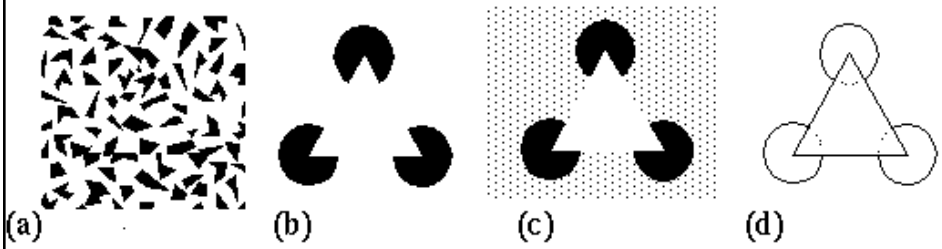
Biological basis:

For humans at least, Gestalt psychology identifies several properties that result in grouping/segmentation:

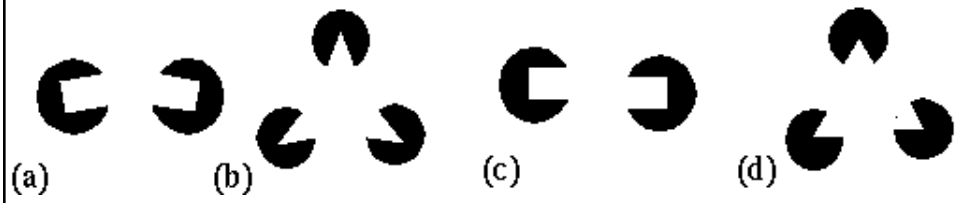


Slide: Bradski&Trun

Consequence:
Groupings by Invisible Completions



Stressing the invisible groupings:

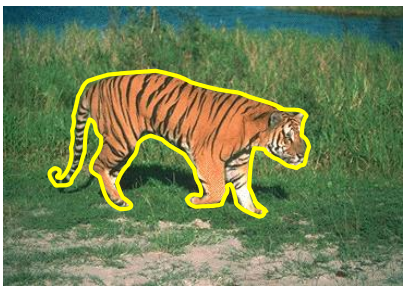


* Images from Steve Lehar's Gestalt papers: http://cns-alumni.bu.edu/pub/slehar/Lehar.html

And the famous invisible dog eating under a tree:

Slide: Bradski&Trun

Segmenting objects



How could this be done?

Image Segmentation

- Many approaches proposed
  - cues: color, contours, textural patterns
  - automatic vs. user-guided
  - no clear winner
- Is user-input required?
- * Our visual system is proof that automatic methods are possible, classical image segmentation methods are automatic
- * Argument for user-directed methods? only user knows desired scale/object of interest

Segmentation algorithms must be chosen and evaluated with an application in mind

Problem of Image Segmentation:

Segmentation: Labeling pixels in an image so that the image is partitioned into "meaningful" regions

Input: a grayscale or color image or image volume

Output: a binary (K-ary) image with labels

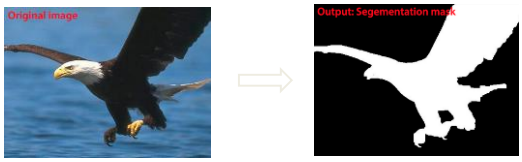
e.g. Labels = "foreground" vs "background"
or Labels = 0 or 1

Labels: 0, 1, 2, ... K (K objects) if more than two regions exist:

Image from Kohli, MICCAI 2009 tutorial

Segmentation

- Image segmentation partitions an image into regions (aka *segments*).

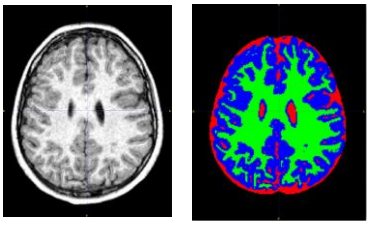


- You can think of image segmentation as assigning a label to each pixel in the image. In the case above, there are two labels: *bird* (foreground, white) and *non-bird* (background, black).

Slide: Dr. Greg Slabaugh, City University of London

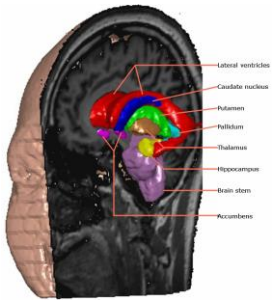
Segmentation

- There can be any number of labels



Four labels:
1. Non-brain
2. Grey matter
3. White matter
4. CSF

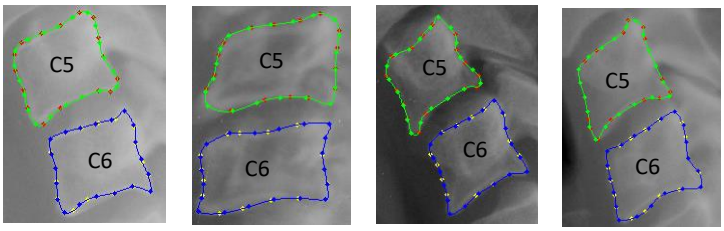
- Different types of data; more than two dimensions

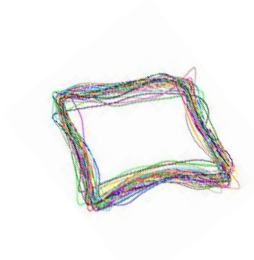


Slide: Dr. Greg Slabaugh, City University of London

Segmentation

- The boundary of a segment provides a *shape*.





A collection of aligned C4 vertebra shapes. From this, we can do all sorts of interesting things:

- Population statistics
- Mean, variation (principal component analysis)
- (more)

Slide: Dr. Greg Slabaugh, City University of London

Under and over segmentation

- An image that has been broken into too many segments is considered to be *over-segmented*

- Not enough segments is considered *under-segmented*

Pink flowers merged together

Slide: Dr. Greg Slabaugh, City University of London

Bird counting

- How would you determine the number of birds in this image?

http://350sav.fotomaps.ru/flock-of-birds.php

- One way is to count the dark blobs. First we need to *find* the dark blobs.
- Let's load the image and convert to grayscale

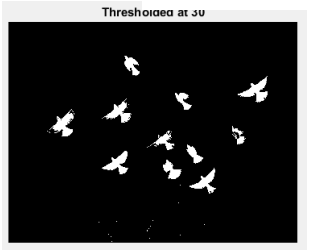
```
I = cv2.imread("birds.png",0)
```

Thresholding

- The simplest form of segmentation is to *threshold* an image. This assigns a value of 1 to any pixel that satisfies the threshold, otherwise, 0. A *binary* image is produced.

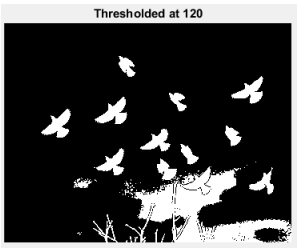
$$B(x,y) = \begin{cases} 1, & I(x,y) < T \\ 0, & \text{otherwise} \end{cases}$$

>T if looking for bright regions



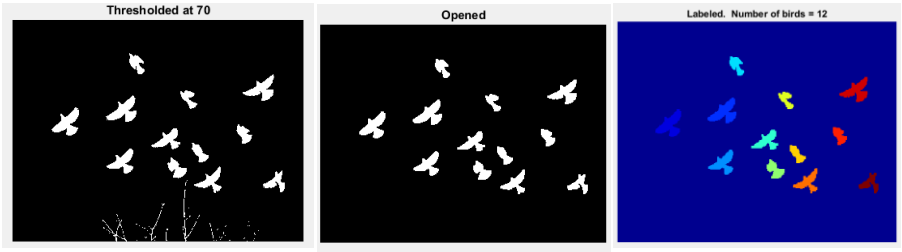

```
I = np.uint8((I < 30)
             * 255)
cv2.imshow('Thresholded'
           ' at 30', I)
cv2.waitKey(0)
```


```
I = np.uint8((I < 70)
             * 255)
cv2.imshow('Thresholded'
           ' at 70', I)
cv2.waitKey(0)
```



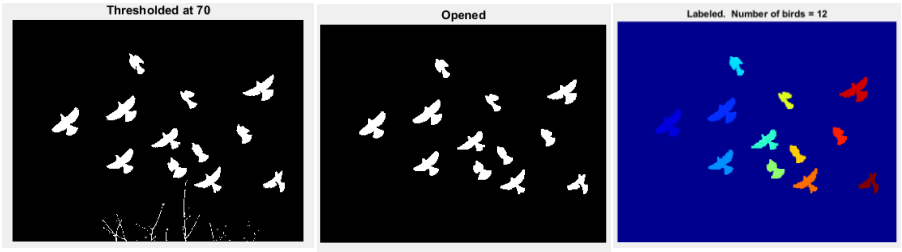

```
I = np.uint8((I < 120)
             * 255)
cv2.imshow('Thresholded'
           ' at 120', I)
cv2.waitKey(0)
```

Bird counting




```
I = cv2.imread("birds.png",0)
ret,B=cv2.threshold(I, 70, 255, cv2.THRESH_BINARY_INV)
cv2.imshow("Thresholded    at 70",B)
J = cv2.morphologyEx(B, cv2.MORPH_OPEN, np.ones((3,3),np.uint8))
cv2.imshow('Opened',J)
labelCount, labels = cv2.connectedComponents(J)
# Map component labels to hue val
label_hue = np.uint8(179*labels/np.max(labels))
blank_ch = 255*np.ones_like(label_hue)
labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])
```

Bird counting



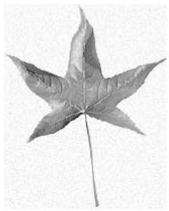

```
# cvt to BGR for display
L = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)
# set bachground label to black
L[label_hue==0] = 0
cv2.imshow('Labeled. Number of birds = ' + str(labelCount-1), L)
#backgorund is also labeled
cv2.waitKey(0)
cv2.destroyAllWindows()
```

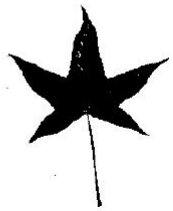
So, the simplest Segmentation approach is global thresholding

Simplest way to segment an image : create a binary image by separating the image domain into two regions

It labels each pixel **in** or **out** of the region of interest by comparison of the grey level with a threshold *T*:

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





Basic Thresholding Algorithm

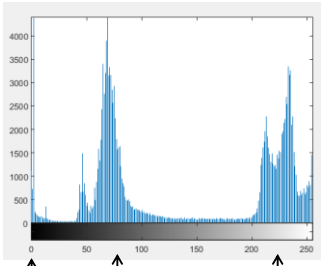
```
for x=1:X
    for y=1:Y
        B(x,y) = (I(x,y) >= T);
    end
end
```

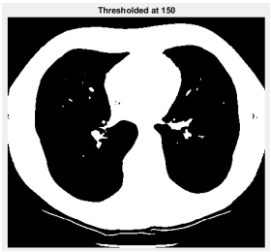
Don't write it like this at home!

Picture: Bryan Morse notes, BYU

Finding a good threshold

- The success of thresholding often depends on finding a good threshold *T*.
- Recall the histogram h(I) of an image. It counts how many pixels of a particular intensity are in the image.





air lungs tissue

a good *T*?

```
I = cv2.imread(lung.png",0)
cv2.calcHist([img],[0],None,
                    [256],[0,256])
J = I > 150;
cv2.imshow('Thresholded at 150', J);
```

Look at Histogram of the image (CT image slice from the brain)

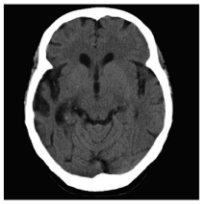
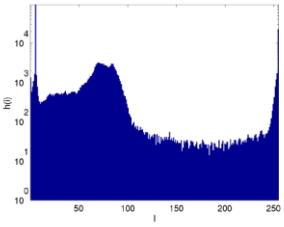
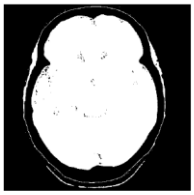


Image.



Histogram of CT brain image.
3 peaks: background, brain, skull.
Pick thresholds from histogram: 11 and 185.

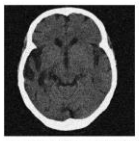
Background.



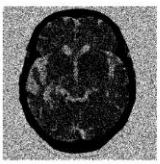
Brain.

Skull.

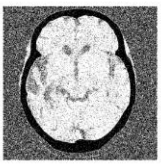
Slide: M. Niethammer



Thresholding
(noisy image)

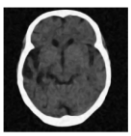


Background.



Brain.

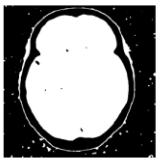
Skull.



Thresholding
(smoothed image)

Smoothing helps local thresholding.

Background.



Brain.

Skull.

M. Niethammer

Skin segmentation

* Segmentation of skin a notoriously difficult problem, given the variation observed in practice. One can perform simple skin detectors using *colour thresholding*.

Examples below: Work in a color space (other than (R,G,B)): Y, U, and V values; Y stores the brightness (luminance), and the color (chrominance) stored as U and V values.

[Chai et al.]

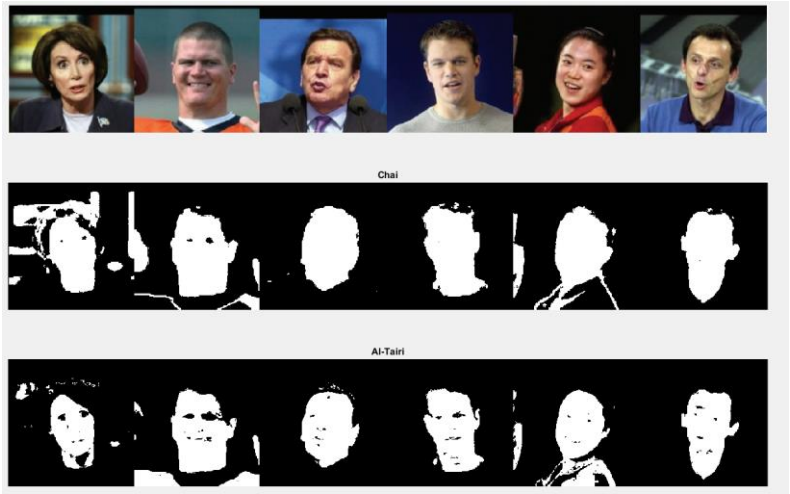
Convert to YUV
Skin = (77 <= U <= 127) and
(133 <= V <= 173)

[Al-Tairi et al.]

Convert to YUV
Skin = (80 < U < 130) and
(136 < V < 200) and
(V > U) & (R > 80) and
(G > 30) & (B > 15) and
|R-G| > 15

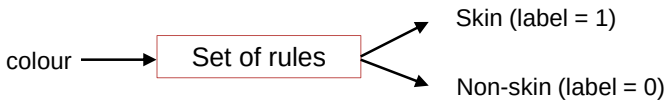
```
I = cv2.imread("faces.png");
cv2.imshow('Faces',I)
YUV= cv2.cvtColor(I, cv2.COLOR_RGB2YCR_CB);
U=YUV[:,:,1]; V=YUV[:,:,2]; R=I[:,:,2]
G=I[:,:,1]; B=I[:,:,0]
rows,cols,planes=I.shape

# Chai et al.
skin=np.zeros([rows,cols],dtype=np.uint8)
ind=(77<=U) & (U<=127) & (133<=V) & (V<= 173 )
skin[ind]=255; cv2.imshow('Chai',skin)

# Al-Tairi et al.
skin=np.zeros([rows,cols],dtype=np.uint8)
ind=(80    <  U) &  (U <  130)    &  \
    (136   <  V) &  (V <= 200    )& \
    (V >  U) &  (R >  80)     &  \
    (G >  30)     &  (B >  15 )& \
    (abs(R-G)  >  15)
skin[ind]=255; cv2.imshow('AL-Tairi',skin);
cv2.waitKey(0);cv2.destroyAllWindows()
```

Example



Rule-based

- We have a set of rules that label a pixel as skin or non-skin.
- More abstractly, we can think of this as



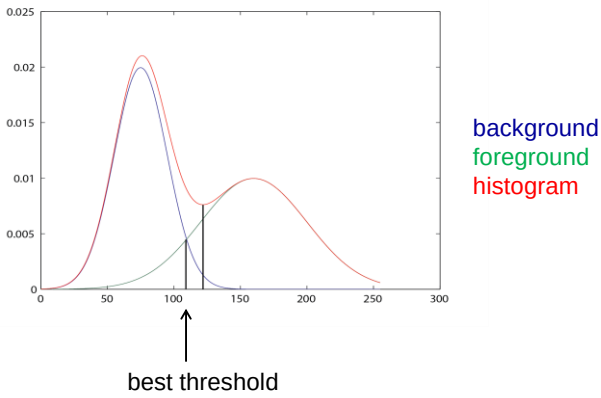
⇨ This is a rule-based classifier

In this case, the rules have been generated by a human by looking at many datasets.

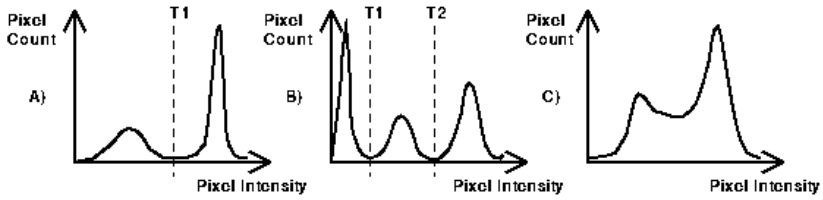
* An alternative is to use *machine learning* to develop a supervised approach to skin detection. Supervised learning is another way to perform segmentation, topic of another time

Automatic thresholding methods

- Many automatic methods to determine the threshold assume a *bimodal* histogram, meaning there are two peaks (one for foreground, the other for background)



Bimodal histogram Trimodal histogram:

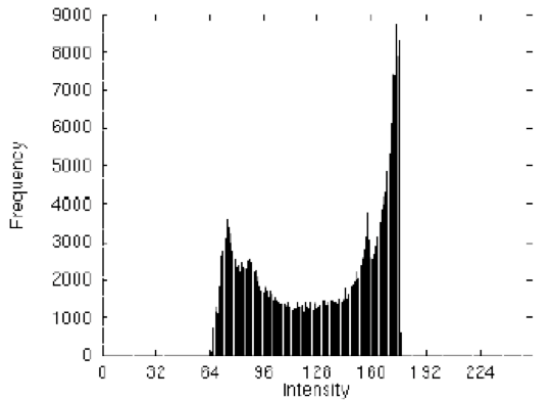


For instance, the segmentation thresholds T are set as the local minima of the histogram

Methods of threshold detection: Usually based on histogram analysis

Finding Modes of Histograms

Idea: find two peaks (modes) and the local minimum in between

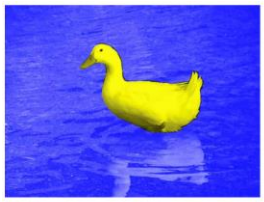


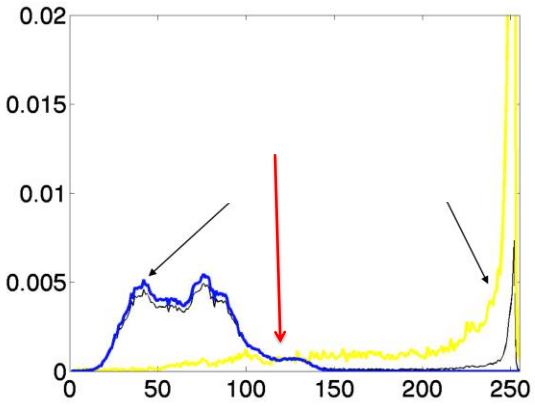
Problems :
Noisy Histograms
Are there better ways?

Bryan Morse, BYU

Optimal Thresholding

- Idea: the histogram is the sum of two overlapping distributions.
- Optimal threshold: the overlapping point of these distributions.





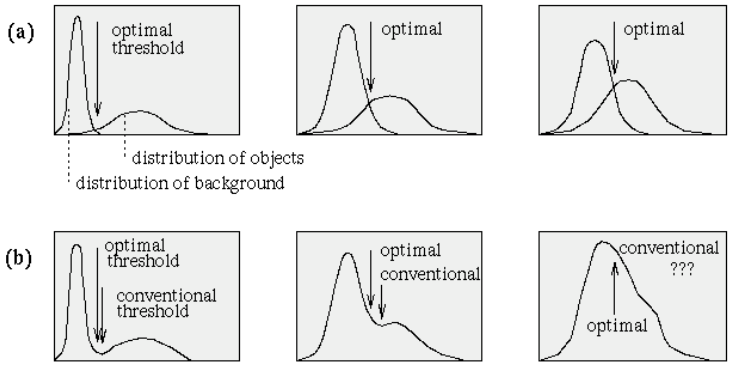


Figure 5.4 *Grey level histograms approximated by two normal distributions; the threshold is set to give minimum probability of segmentation error: (a) Probability distributions of background and objects, (b) corresponding histograms and optimal threshold.*

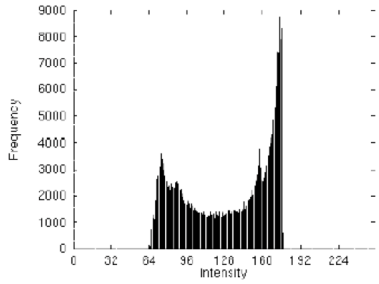
Problem 1: Don't know the distributions

Problem 2: "Optimal threshold" as the intersection of two distributions is not the same as the local minimum.

Figure: Bryan Morse, BYU

"Optimal" Thresholding
One approach was proposed by Otsu

One can express the total variance of image intensities as:



$$s^2(T) = s^2_{Within}(T) + s^2_{Between}(T)$$

T: unknown threshold

Within-class variance

Between-class variance

Since the total variance is constant and independent of *T*, the effect of changing the threshold is merely to move the contributions of the two terms back and forth.

So, *minimizing the within-class variance is the same as maximizing the between-class variance with respect to "optimal" threshold T to be estimated.*

The nice thing about this is that we can calculate the quantities in $s^2_{Between}(T)$ *recursively* as we run through the range of *T* values when we compute the variance

Optimal thresholding (Otsu)

- An approach was proposed by Otsu.
- Otsu's method finds the threshold *T* that minimises the within-class variance, defined as

$$\sigma^2 = w_f \sigma_f^2 + w_b \sigma_b^2$$

where σ_f^2 and σ_b^2 are the variances of the foreground and background, and

$$w_b = \sum_{I=0}^{T} p(I) \qquad w_f = \sum_{I=T+1}^{255} p(I)$$

are weights formed by summing the histogram pdf over the background and foreground intensities, respectively.

In OpenCV, Otsu thresholding is implemented with the function `threshold with cv2.THRESH_OTSU parameter`, which returns a threshold in the range of 0 to 1.

Optimal thresholding (Otsu)

Easier way to calculate:

- If you subtract the "within-class" variance from the total variance of the combined distribution, you get the "between-class" variance:

$$\sigma^2_{Between}(T) = \sigma^2_{total} - \sigma^2_{within}(T)$$

$$= w_B(T)[\mu_B(T) - \mu]^2 + w_F(T)[\mu_F(T) - \mu]^2$$

where σ^2 is the combined variance and μ is the combined mean.

* Since total variance is independent of the threshold, maximizing the between-class variance is the same as minimizing the within-class variance.

Substituting:

$$\mu = w_B(T)\mu_B(T) + w_F(T)\mu_F(T)$$

and simplifying, we get:

$$\sigma^2_{Between}(T) = w_B(T)w_F(T)[\mu_B(T) - \mu_F(T)]^2$$

"Optimal" Thresholding (Otsu's method)

Goal: Find T that maximizes the measure:

$$\sigma^2_{Between}(T) = w_B(T)w_F(T)[\mu_B(T) - \mu_F(T)]^2$$

ALGORITHM:

i. For each potential threshold T:

1. Separate the pixels into two clusters according to the threshold T
2. Find the mean of each cluster (don't have to calculate the variance in this way – if you minimize within-class variance instead, you have to calculate the variances of each cluster)
3. Calculate the above measure

ii. Output: Choose the threshold T that gives the maximum measure.

Extra Slide 1: OTSU'S THRESHOLDING Method: Details for an even more efficient implementation
Note the notation change from w_ weights to n_ and B(background) and O(object)

Idea: select T to minimize the *within-class* variance—the weighted sum of the variances of each cluster:

$$\sigma^2_{\text{Within}}(T) = n_B(T)\,\sigma^2_B(T) + n_O(T)\,\sigma^2_O(T)$$

where

$$\begin{aligned} n_B(T) &= \sum_{i=0}^{T-1} p(i) \\ n_O(T) &= \sum_{i=T}^{N-1} p(i) \\ \sigma^2_B(T) &= \text{the variance of the pixels in the background } (< T) \\ \sigma^2_O(T) &= \text{the variance of the pixels in the foreground } (\geq T) \end{aligned}$$

and $[0, N-1]$ is the range of intensity levels.

Extra Slide 2: A very efficient implementation

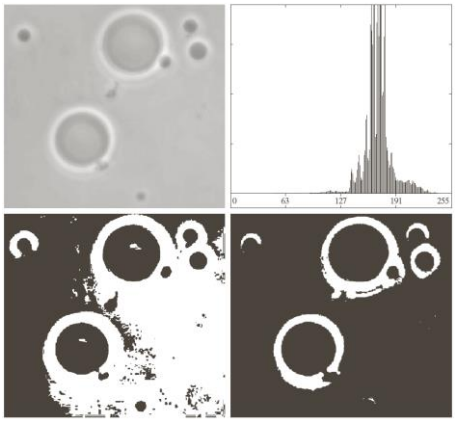
- Better Still: Update $n_B(T)$, $n_O(T)$, and the respective cluster means $\mu_B(T)$ and $\mu_O(T)$ with the pixels that move from one cluster to the other as T increases:

$$\begin{aligned} n_B(T+1) &= n_B(T) + n_T \\ n_O(T+1) &= n_O(T) - n_T \end{aligned}$$

$$\mu_B(T+1) = \frac{\mu_B(T)\,n_B(T) + n_T\,T}{n_B(T+1)}$$

$$\mu_O(T+1) = \frac{\mu_O(T)\,n_O(T) - n_T\,T}{n_O(T+1)}$$

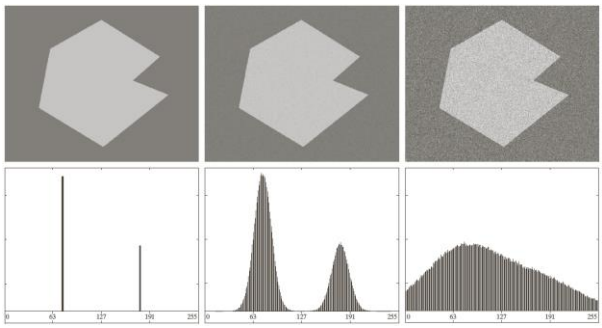
Result of Otsu



a b
c d

FIGURE 10.39 (a) Original image. (b) Histogram (high peaks were clipped to highlight details in the lower values). (c) Segmentation result using the basic global algorithm from Section 10.3.2. (d) Result obtained using Otsu's method. (Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

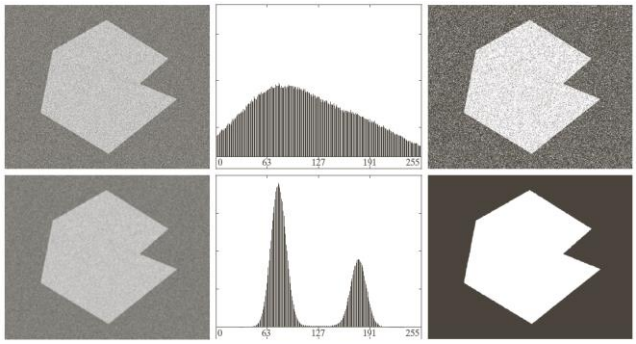
Effect of Noise in Thresholding



a b c
d e f

FIGURE 10.36 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d)–(f) Corresponding histograms.

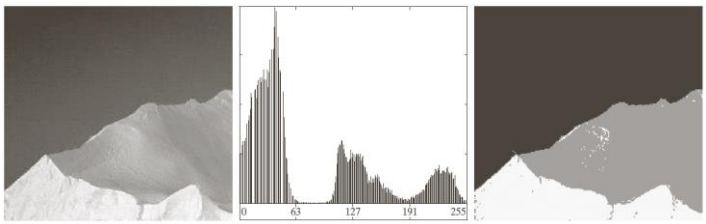
Effect of Noise and Smoothing in Thresholding



a b c
d e f

FIGURE 10.40 (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method.

Ternary Segmentation: i.e. Segment the image into 3 regions



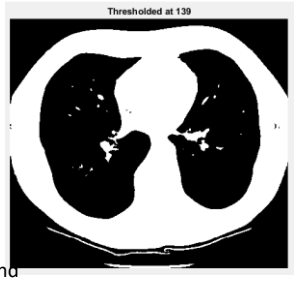
a b c

FIGURE 10.45 (a) Image of iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

Iterative Thresholding method (Another Method)

1. Set T to some initial value, and determine foreground and background pixels.
2. Compute m_b, the mean of the background, and m_f, the mean of the foreground, based on the current value of T.
3. Set $T = (m_b + m_f)/2$
4. Go to step 2 until convergence (when T no longer changes).

```
T=50
lastT=0
while abs(T-lastT) > 1 :
    lastT=T
    mf= np.mean(I[I>T])
    mb= np.mean(I[I<=T])
    T=0.5*(mb+mf)

cv2.imshow("Lungs",I)
```



In this example, the loop executes 5 times, with T starting at 50, and going to 81.9, 126.7, 138.2, 139.3, 139.4

An idea to set the initial Threshold: Start with m_b as the average of the four corner pixel, which is assumed to be the background, and m_f as the average of everything else. Go to step 3 of the algorithm above and continue.

Idea behind Iterative Threshold Selection

- Idea: pick a threshold such that each pixel on each side of the threshold is closer in intensity to the mean of all pixels on that side of the threshold than the mean of all pixels on the other side of the threshold.
- Let

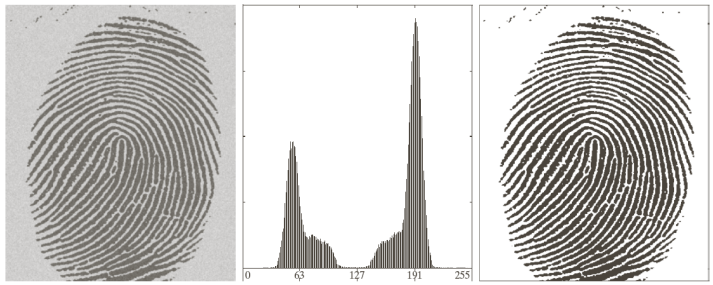
$\mu_B(T)$ = the mean of all pixels less than the threshold (background)
$\mu_O(T)$ = the mean of all pixels greater than the threshold (object)

We want to find a threshold such that the greylevels for the object are closest to the average of the object:

$$\forall g \geq T : |g - \mu_B(T)| > |g - \mu_O(T)|$$

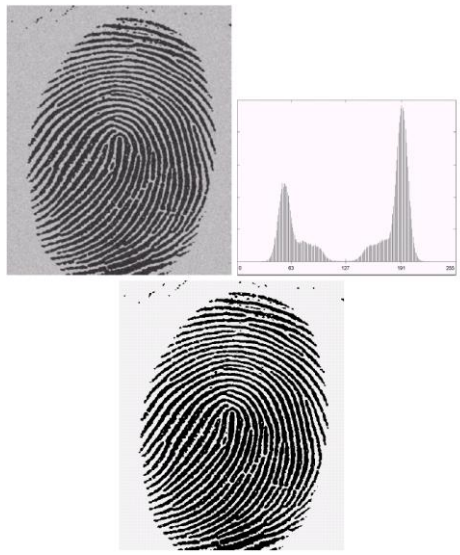
and the greylevels for the background are closest to the average of the background:

$$\forall g < T : |g - \mu_B(T)| < |g - \mu_O(T)|$$



a b c

FIGURE 10.38 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (the border was added for clarity). (Original courtesy of the National Institute of Standards and Technology.)



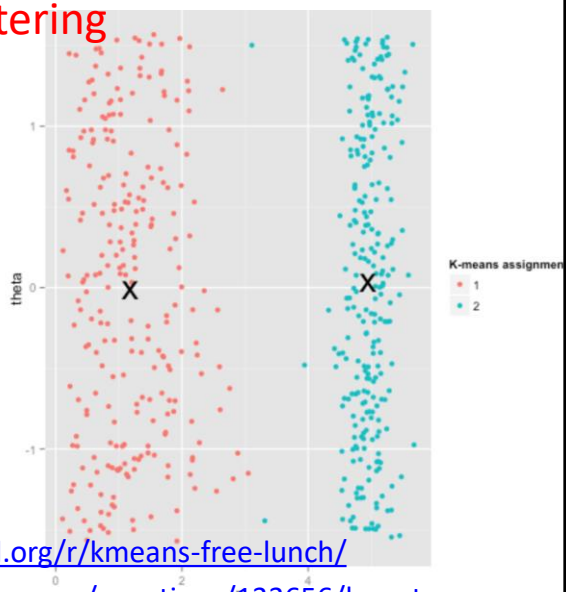
a b
c

FIGURE 10.29
(a) Original image. (b) Image histogram. (c) Result of segmentation with the threshold estimated by iteration. (Original courtesy of the National Institute of Standards and Technology.)

K means Clustering

- The optimal (iterative) thresholding method described on the previous slide is an example of the K means algorithm, where K = 2.
- K means is a clustering technique, in this case grouping pixels of similar intensity into two groups, or *clusters*.
- However, K means can be used to cluster data into any number (K) classes. The data can be based on intensity, colour, or any attribute associated with a pixel (e.g., texture, depth, etc.)
- It requires K to be provided in advance.

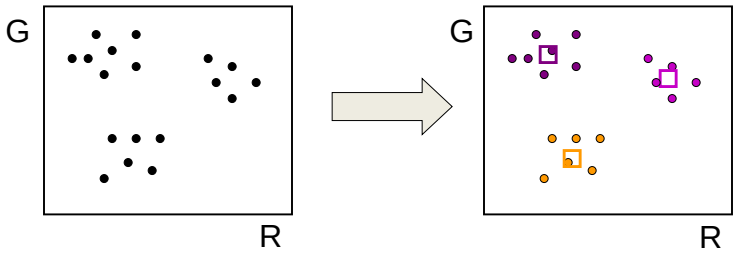
NEXT: K-means clustering



- http://varianceexplained.org/r/kmeans-free-lunch/
- http://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means

Segmentation as a Clustering Problem

- How to choose the representative colors?
  - This is a clustering problem!



Objective

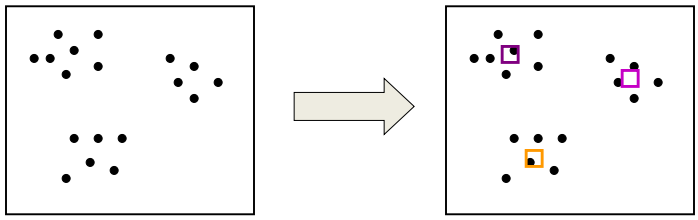
- Each point should be as close as possible to a cluster center
  - Minimize sum squared distance of each point to closest center

$$\sum_{\text{clusters } i} \quad \sum_{\text{points p in cluster } i} ||p - c_i||^2$$

R. Szeliski Slides

Break it down into subproblems

- Suppose I tell you the cluster centers c_i
  - Q: how to determine which points to associate with each c_i?
  - A: for each point p, choose closest c_i



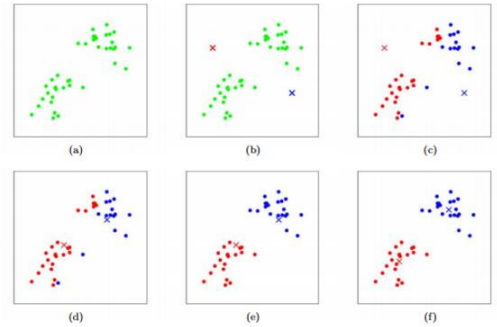
Suppose I tell you the points in each cluster

- Q: how to determine the cluster centers?
- A: choose c_i to be the mean of all points in the cluster

R. Szeliski Slides

Algorithm: K-means clustering

1. Start with a set of N points (green points below)
2. Randomly initialize the K cluster centers, c_1, ..., c_K (x's below)
3. Assignment: Given cluster centers, determine points in each cluster
   - For each point p, find the closest c_i. Put p into cluster i
4. Update: Given points in each cluster, calculate new means c_i
   - Set c_i to be the mean of assigned points in cluster i
5. Go to 3 until convergence (e.g. means c_i no longer change)



Java demo (please check if the link still works)
http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html

K-means clustering

K-means minimizes the objective function: Sum of Within-Cluster Variance, which is also Mean Squared Error within each cluster.
Equivalently: it is based on Sum of Squared Distances (SSD) between points in a cluster and the mean point

$$\sum_{\text{clusters } i} \sum_{\text{points p in cluster } i} \|p - c_i\|^2$$

Properties

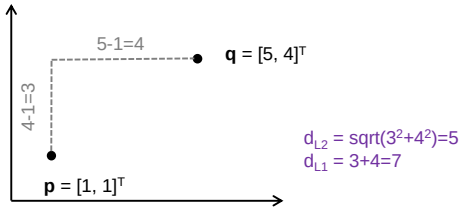
- Will always converge to *some* solution
- Can be a "local minimum"
  - does not always find the global minimum of objective function

Issues with K means

- What is K?
  - In some applications, there is an obvious K. In others, it may be unclear.
- How to initialise the means? Potentially different results each time the algorithm is run
  - One can rerun the algorithm multiple times and merge the results
- Dealing with high dimensional data
  - Finding nearest neighbours can be slow (e.g. kd tree can help)
- How do you measure distance?
  - Normally the L2 norm (Euclidean distance) is selected, but there are other options, for example, the L1 distance (aka CityBlock distance).

$$d_{\mathrm{L2}}(\mathbf{p},\mathbf{q}) = \sqrt{\sum_i (p_i - q_i)^2}$$

$$d_{\mathrm{L1}}(\mathbf{p},\mathbf{q}) = \sum_i |(p_i - q_i)|$$



K means

- OpenCV has support for K means using with the kmeans function. It requires an NxM matrix. Here, N is the number of pixels, and M are the features (colours) used in the grouping. This can be achieved using the `reshape` function.

```
img = cv2.imread('road_sky.jpg'); cv2.imshow('Original' ,img)
Z = img.reshape((-1,3))
# convert to np.float32
Z = np.float32(Z)
# define criteria, number of clusters(K) and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 3; attemps=3
ret,label,center=cv2.kmeans(Z,K,None,criteria,attemps,cv2.KMEANS_RANDOM_CENTERS
# Now convert back into uint8, and make original image
center = np.uint8(center)
res = center[label.flatten()]; res2 = res.reshape((img.shape))
cv2.imshow('K = ' + str(K),res2);cv2.waitKey(0);cv2.destroyAllWindows()
```

K means

- Different results for different K

K = 2

K = 3

K = 4

K = 5

K means

- Different results for different *runs* due to random initialisation.

K = 3

K = 3

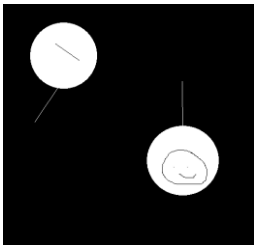
- Replicates (running the algorithm multiple times) can produce more consistent results

```
[clusterID, clusterCentre] = kmeans(R, 3, 'Replicates', 3);
```

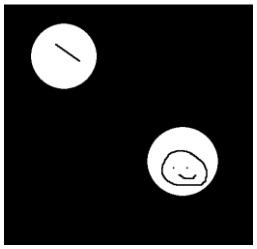
Mathematical morphology

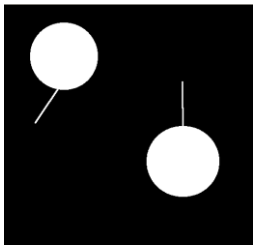
- A related class of nonlinear image processing techniques is known as mathematical morphology.
- These technique also use kernels, which are called *structuring elements* in the nomenclature of this branch of image processing.
- There are two basic operations, *erosion* and *dilation*. Erosion is like a min filter (over the structuring element extent on the image), and dilation is like a max filter. These operations are often used on binary images, but can be used on grayscale images too.
- A *closing* is dilation, followed by erosion. It fills holes smaller than the structuring element. An *opening* is erosion, followed by dilation. It removes objects smaller than the structuring element.
- Mathematical morphology may be useful in "cleaning up" segmentations.

Erosion and Dilation



Original image

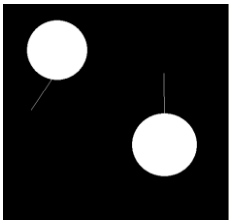




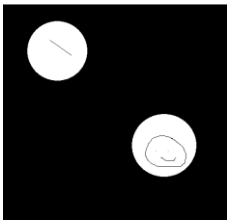
```
J = cv2.erode(I,np.ones((3,3),
              np.uint8))
cv2.imshow('erosion',J)
```

```
J = cv2.dilate(I,np.ones((3,3),
                         np.uint8)
cv2.imshow('dilation',J)
```

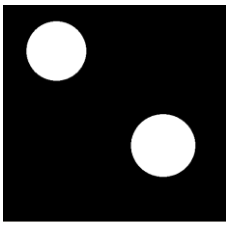
Closing and Opening



```
J = cv2.morphologyEx(I,
cv2.MORPH_CLOSE,
np.ones((3,3),np.uint8))
cv2.imshow('closing',J)
```



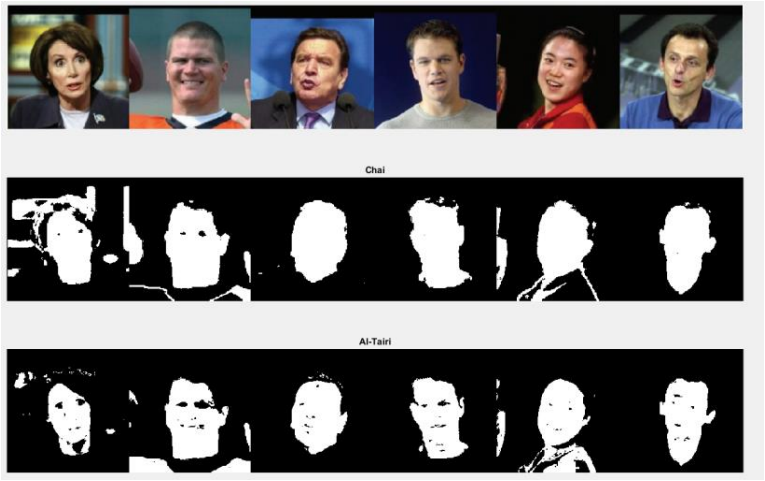
```
J = cv2.morphologyEx(I,
cv2.MORPH_OPEN,
np.ones((3,3),np.uint8))
cv2.imshow('opening',J)
```



```
K = np.ones((3,3),np.uint8)
J1 = cv2.morphologyEx(I,
cv2.MORPH_OPEN, K)
J2 = cv2.morphologyEx(J1,
cv2.MORPH_CLOSE, K)
cv2.imshow('opening-
closing',J2)
```

Evaluating segmentations

- What makes for a “good” segmentation?



Evaluating segmentations

- A common approach is to establish some ground truth. This can specify, for each pixel in the image if it should be part of the segmentation (or not).
- Example data for skin detection: http://web.fsktm.um.edu.my/~cschan/downloads_skin_dataset.html

The Segmentation Benchmark Datasets

http://host.robots.ox.ac.uk/pascal/VOC/

The PASCAL Visual Object Classes

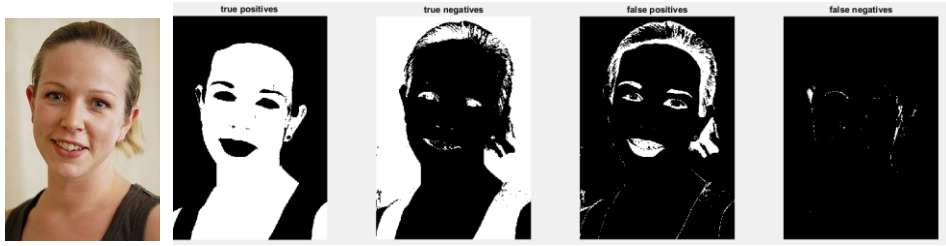
- Provides standardised image data sets for object class recognition and segmentation
- Enables evaluation and comparison of different methods

Figure from Arbelaez et al. Semantic Segmentation using Regions and Parts, 2012, CVPR. (figure 5)

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

Evaluating segmentations

- One can perform the segmentation, and using the ground truth, categorise pixels as:
  - True positives: correctly detected skin pixels
  - True negatives: correctly detected non-skin (background) pixels
  - False positives: non-skin pixels incorrectly detected as skin pixels
  - False negatives: skin pixels missed by the skin detector



Accuracy, sensitivity, specificity

- One can then produce quantitative measures
  - Accuracy

    ACC = (TP + TN) / (TP + FP + FN + TN)

    This measures the number of correctly labelled pixels in the image

  - Sensitivity, or True Positive Rate

    SENS = TP / P = TP / (TP + FN)

    This measures the percentage of positives correctly labelled

  - Specificity, or True Negative Rate

    SPEC = TN / N = (TN + FP)

    This measures the percentage of negatives correctly labelled

- Where TP and TN are the number of true positives and true negatives, and FP and FN are the number of false positives and false negatives.
- A perfect result would have 100% accuracy, sensitivity, and specificity.

Example

```
I = cv2.imread("face.jpg")
T = cv2.imread("groundTruth.png",0)

YUV= cv2.cvtColor(I, cv2.COLOR_RGB2YCR_CB);
U=YUV[:,:,1]; V=YUV[:,:,2]; R=I[:,:,2]; G=I[:,:,1]; B=I[:,:,0]
rows,cols,planes=I.shape

# Al-Tairi et al.
skin=np.zeros([rows,cols],dtype=np.uint8)
ind=(80 < U)   & (U < 130) &  (136 < V)  & (V <= 200) & (V > U) & (R > 80) & (G
> 30)    & (B>15) & (abs(R-G) >    15)
skin[ind]=255

cv2.imshow('Image',I)
cv2.imshow('Ground Truth',T)
cv2.imshow('AL-Tairi',skin)


tpInd = (skin  == 255) & (T == 255)
tnInd = (skin  == 0)  & (T == 0)
fpInd = (skin  == 255) & (T == 0)
fnInd = (skin  == 0)  & (T == 255)
```

Example, part 2

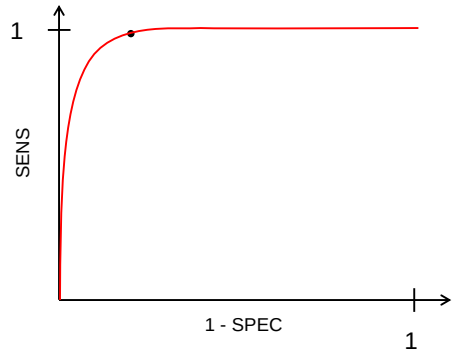
```
tpImage = np.zeros([rows, cols],dtype=np.uint8)
tpImage[tpInd] = 255
tnImage = np.zeros([rows, cols],dtype=np.uint8)
tnImage[tnInd] = 255
fpImage = np.zeros([rows, cols],dtype=np.uint8)
fpImage[fpInd] = 255
fnImage = np.zeros([rows, cols],dtype=np.uint8)
fnImage[fnInd] = 255
cv2.imshow('true   positives',tpImage)
cv2.imshow('true   negatives',tnImage)
cv2.imshow('false  positives',fpImage)
cv2.imshow('false  negatives',fnImage)
tp = len(tpInd); tn = len(tnInd); fp = len(fpInd); fn = len(fnInd)
#  Compute    measures
accuracy = (tp + tn)/ (tp + tn + fp + fn)
sens = tp/(tp + fn); spec = tn/(tn + fp)
print('Accuracy = ' + str(accuracy) + ', sensitivity = ' + str(sens) + ',
specificity = ' +    str(spec))
cv2.waitKey(0); cv2.destroyAllWindows()
```

>> Accuracy = 0.8979 , sensitivity = 0.98949, specificity = 0.82425

Interpretation: Sensitivity is high so many skin pixels correctly identified. However, specificity is lower, due primarily to the false positives. Overall, roughly 90% of the pixels are correctly labelled.

ROC curve

- A Receiver Operating Characteristic curve plots the sensitivity as a function of 1 – specificity.
- In our previous example, SENS = 0.99, 1 – SPEC = 0.18
- One can sweep a parameter that trades of sensitivity vs specificity to generate a curve. The area under the curve gives another way to characterise the quality of a classification method.



Confusion matrix

- The confusion matrix describes the number of predicted labels vs the ground truth. In this example, we have skin and non-skin pixels.

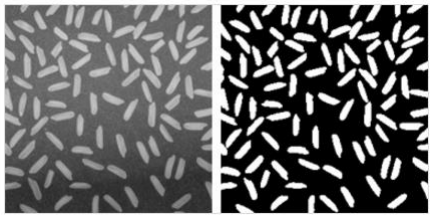
		Predicted	
		Skin	Non-skin
Actual	Skin	TP	FN
	Non-skin	FP	TN

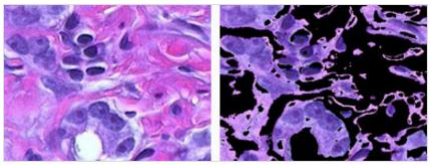
- A perfect result would be a diagonal matrix (no FPs or FNs)
- The confusion matrix generalises to multiple labels

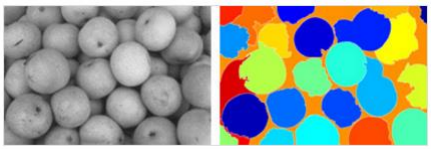
		Predicted		
		Label 1	Label 2	Label 3
Actual	Label 1			
	Label 2			
	Label 3			

OpenCV documentation and examples

- https://docs.opencv.org/3.4.3/d1/d5c/tutorial_py_kmeans_opencv.html







Segmentation based on Region Growing

https://www.youtube.com/watch?v=WJGcaSmVE0E

Region Growing labels pixels into segmentation labels based on:

- predefined criteria for growth
- starting from a set of seeds

- Important : growth or similarity criterion selected according to the problem under consideration

- Can be based on color, image statistics etc.

- Stop the region growing when no more pixels change label

Region Growing

- Start with a single *seed pixel* p and expand from that.
- Define a similarity measure $S(i, j)$ for pixels i and j.
- Add adjacent pixel q to pixel p's region iff $S(p, q) > T$ for some threshold T.
- Proceed to the other neighbors of p and do likewise.
- We can now similarly consider the neighbors of q and add them likewise if they are similar enough.

Similarity Measures

Can use intensity, texture, color (hue), etc.

- Compare to seed pixel
  Sensitive to noise, not transitive
- Compare to neighboring pixel
  Transitive, can drift easily
- Compare to region statistics
  Not transitive, doesn't drift as easily
- Multiple seed points
  Can get an idea not only of values but *variance*

from Bryan Morse, BYU

Region Growing

Design decisions:

1. How do we define similarity measure S?
2. What threshold T do we use? Does it change or stay constant?
3. If we wish to add q's neighbor r, do we use $S(p, r)$, $S(q, r)$, or something else?

Next: A list-based efficient processing in C++ for a fast implementation: recursive

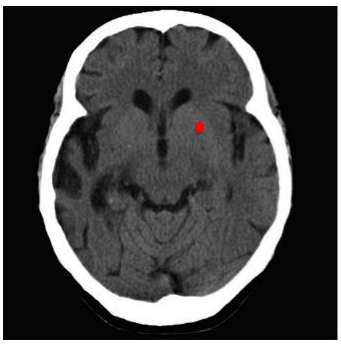
Region Growing Algorithm ~ Connected Components

```
// Nice Simple Example C++ code for 3D segmentation:
// check it out; for questions: send an email to me

struct vector3Df {float x, y, z;
                 };
//INITIALIZE YOUR LISTS
std::vector<vector3Df> *current;      //holds current region voxels
std::vector<vector3Df> *next;          //holds voxels to be analyzed on the next iteration
std::vector<vector3Df> *swap;         //pointer used in swapping the two lists
std::vector<vector3Df>::iterator it;
int Y = xSize;     int Z = xSize*ySize;          int x,y,z,i,j,k;
vector3Df  point3D;
point3D.x = seedX;  point3D.y = seedY;  point3D.z = seedZ; //  the seed location: a 3D vector

int OBJECT= 2;
int MARKED = 1;

next = new std::vector<vector3Df>;
current = new std::vector<vector3Df>;

next->clear();              //clear the two lists
current->clear();
//initialize the current list with the seed voxel
current->push_back(point3D);
```

Region Growing Algorithm (cont'd)~ Connected Comp

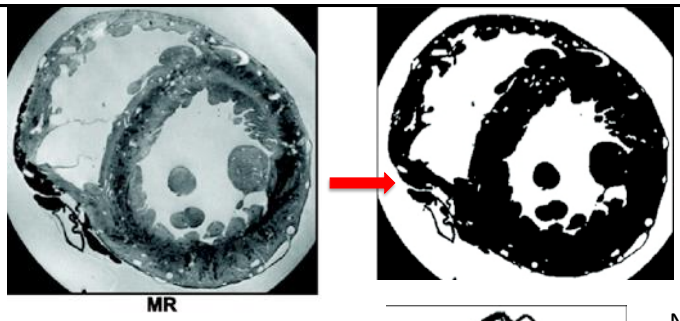
```
while(!current->empty()) //while there are still current voxels
{for (it = current->begin(); it != current->end(); it++) { //for each current voxel
      x = (int)it->x;  y = (int)it->y;  z = (int)it->z;
      segMap[x+y*Y+z*Z] = OBJECT; //mark current voxel as object in the seg. map

      for(i=-1;i<=1;i++)for(j=-1;j<=1;j++)for(k=-1;k<=1;k++)
       { //******* for each nbr of current voxel
            // check if its in the volume
            if ( x+i<0 || x+i>=xSize || y+j<0 ||y+j>=ySize || z+k<0 || z+k>=zSize)  continue;

           //DESIGN:  Similarity Criterion
            if( (segMap[(x+i)+(y+j)*Y+(z+k)*Z] ~= OBJECT && (fabs(Image[x+y*Y+z*Z] -
Image[(x+i)+(y+j)*Y+(z+k)*Z]) < epsilon) ) {
              segMap[(x+i)+(y+j)*Y+(z+k)*Z] =  OBJECT;  //mark the nbr as object/connected
              point3D.x = x+i; point3D.y = y+j; point3D.z = z+k;   //add it to the "next" list
               next->push_back(point3D);          }
       } //*********end for each nbr of current voxel
     }//********for each current voxel
  swap = current;    //swap "current" and "next" lists... old "next" is new "current"
  current = next;
   next = swap;    next->clear();
}// end while  //free allocated memory ;
delete next;  delete current;  .....
```

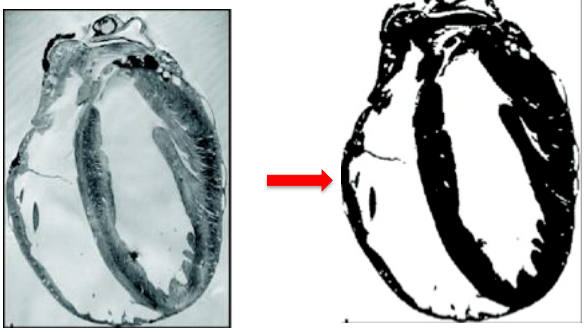
Segmentation based on Region Growing



Region growing started with a seed point in the brain (e.g. the red dot shown)

For example, here adds neighboring points to the region if they are less than a threshold away from a running (re-calculated or updated every step) mean intensity



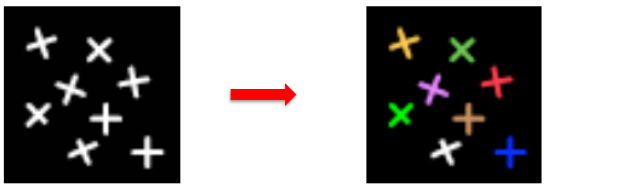


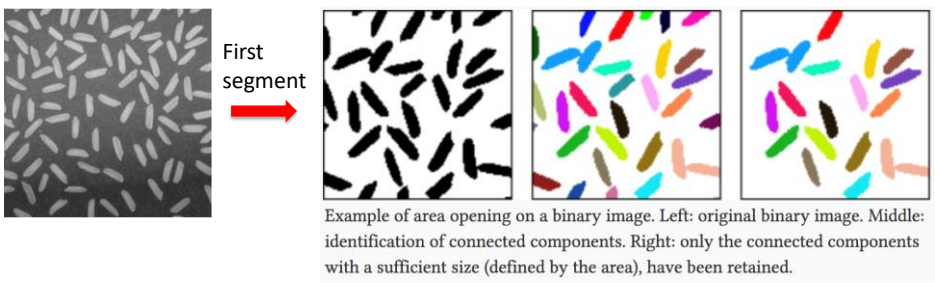
Next step after segmentation of multi-region scenes or objects is to label different parts separately. How? We can use connected component labeling

Figure from: "Development of an anatomically detailed MRI-derived rabbit ventricular model and assessment of its impact on simulations of electrophysiological function", American J Physiology, MJ Bishop et al 2010

Connected Component Labeling

How to label (or color) separately connected segmented regions in an image ?

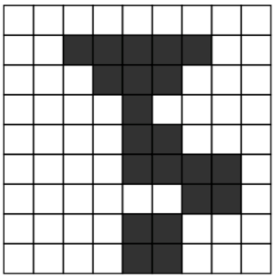




Example of area opening on a binary image. Left: original binary image. Middle: identification of connected components. Right: only the connected components with a sufficient size (defined by the area), have been retained.

http://imagej.net/MorphoLibJ

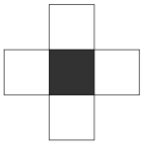
Pixel Connectivity



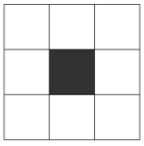
Q: are the dark pixels in this image connected?

To answer that: We need to define which pixels are counted as neighbors

Pixel Neighbourhoods



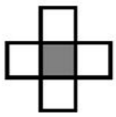
4-neighbourhood



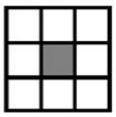
8-neighbourhood

Spatial Connectedness: going beyond color

- Colour (or intensity) is a powerful feature and typically used in image segmentation. However, typically there is additional contextual information we can use, often encoded *spatially*.
- When segmenting a pixel **p**, we might consider pixels in **p**'s neighbourhood.
- This require a definition what a neighbouring pixel is.



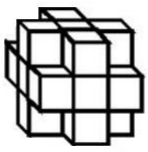
4-neighbourhood



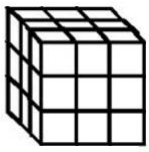
8-neighbourhood

Common 2D neighbourhoods

6-neighbourhood



18-neighbourhood



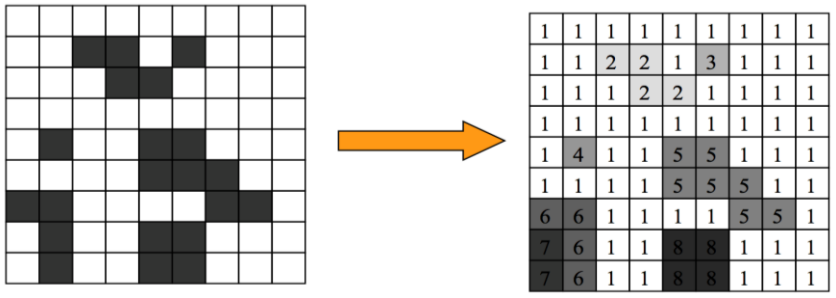
26-neighbourhood

Common 3D neighbourhoods

https://www.linkedin.com/pulse/20140506202304-14648565-3d-is-not-1-5-times-2d

Connected Components Labeling

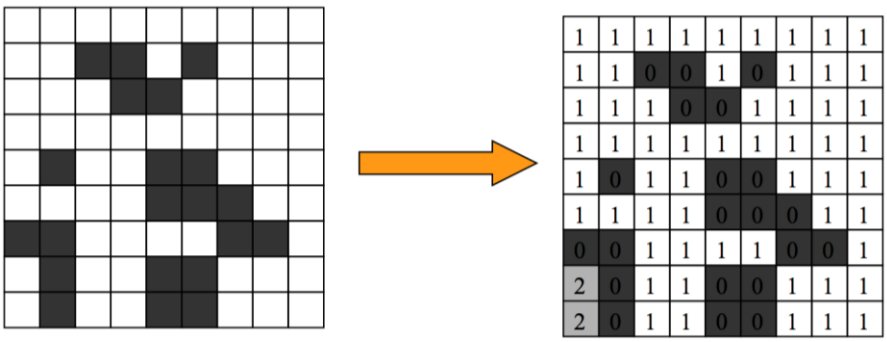
- Labels each connected component of a binary image with a separate number.



E.g. Use region growing starting from the top left corner of the image, set it to label L=1. Then find the first unlabelled pixel, set to L=2, use region growing, and continue in this way. Here, region growing algo, growing criteria: Neighbor pixel has the same label as the current pixel.

Foreground Connected Components Labeling

- Only extract the connected components of the foreground



Assuming that dark pixels are the background, others are foreground

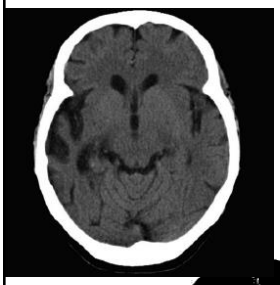
Check this algorithm on connected component labeling

```
#B is the binary image input.
#L is the labeled image output
def ConnectedComponents(B):
    X,Y = B.shape
    L = np.zeros([X,Y])
    n=0
    for (y,x) in B:
        if B[y, x] and L[y, x]==0:
            label(x,y,n,B,L)
            n = n + 1
    return L
```

```
# Recursively give label to this pixel
# and all it foreground neighbours.
def label(x_start,y_start,n,B,L):
    L[y_start, x_start]= n
    for (y,x) in N[y_start, x_start]:
        if L[y, x]==0 and B[y, x]:
            label(x,y,n,B,L)
```

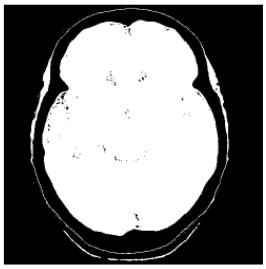
You need to define your neighborhood function N(.,.). E.g. 4 nbhd: x,y should be one of: (xs+1,ys), (xs-1,ys), (xs,ys+1), (xs,ys-1)

Recall Segmentation based on Thresholding



… needs to be cleaned up -> e.g., morphological operations.

Background.

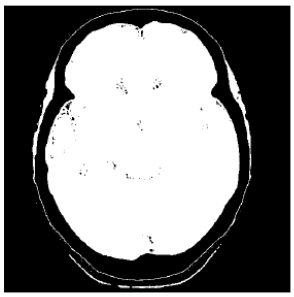


Brain.

Skull.

M. Niethammer

Extracting Connected Component

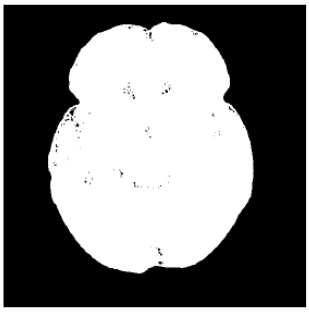


Original brain mask.

Largest connected component.

M. Niethammer

Dilation

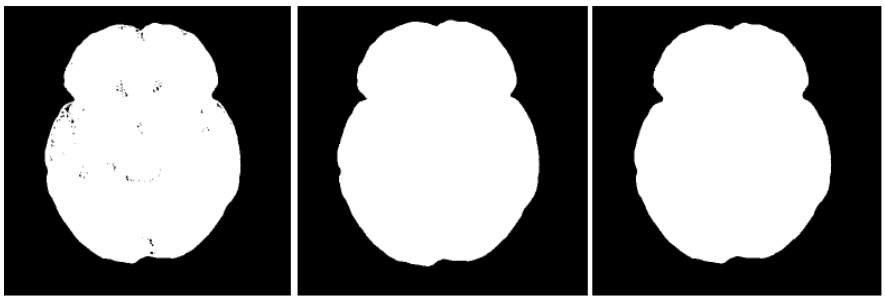


Original brain mask.

Dilated brain mask.
Fills in holes (5x5, SE).

M. Niethammer

Erode to Recover Original Size



Original. Dilated. Closed.

Dilation followed by Erosion = Closing

M. Niethammer

END OF LECTURE

Note that Segmentation has a vast literature, there are many more methods currently used in Computer Vision.
We just studied the basic methods, but they are important in understanding the idea of segmentation and data clustering.

Recall Learning objectives of the week: Students are able to:

3. Define and construct segmentation, feature extraction, and visual motion estimation algorithms to extract relevant information from images

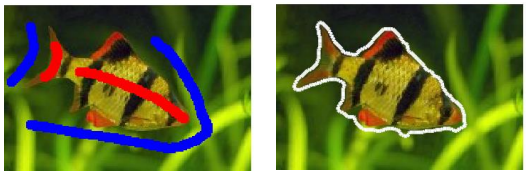
Reading Assignments:
[Klette Book]

[Gonzalez and Woods] 10.3 : very limited
R. Szeliski Comp Vision Book: Section 5.3

Extra Material for those interested

Interactive segmentation

- Recently a number of interactive segmentation methods have been presented in the literature. These methods often ask the user to provide some strokes providing seeds.

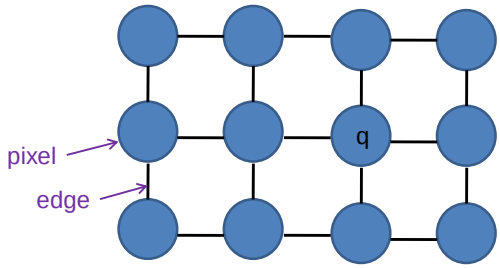


http://jgmalcolm.com/pubs/research.html

- The strokes:
  - Are used to form a colour model of different regions.
  - Also provide spatial context.

Markov Random Field

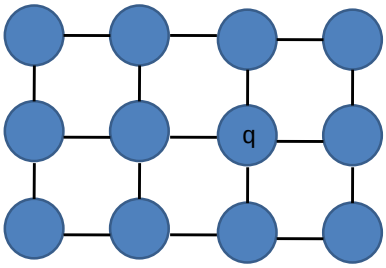
- Often, one employs a Markov Random Field to solve this problem.
- In a Markov Random Field (MRF), each pixel in the image is considered a random variable. It is connected to its adjacent pixels through edges.
- The segmentation label for a pixel depends on two terms:
  - A *unary* term, the cost of assigning a label to a pixel, independent of the neighbours.
  - A *pairwise* term, the cost of a assigning a label to a pixel based on a neighbour in the neighbourhood.



The cost of assigning a label ℓ to pixel q depends on the image and the values of labels of the connected pixels as well.

Markov Random Field

- Example

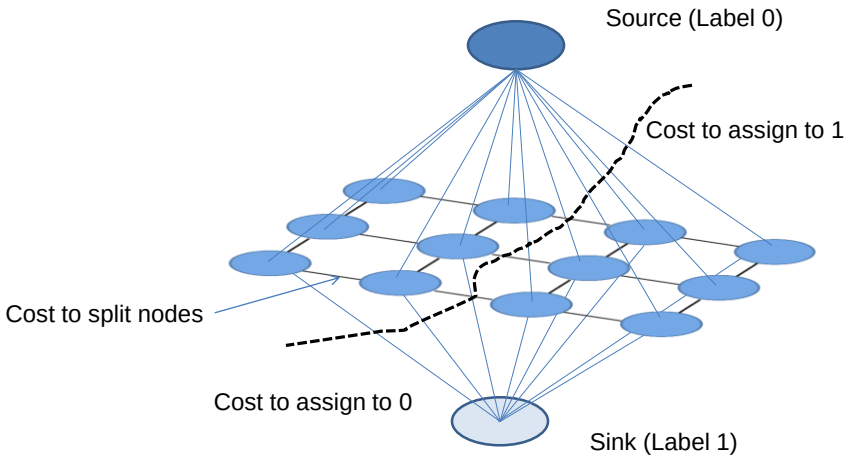


- Unary
  - Label 0: -log p(ℓ(q) = 0| image)
  - Label 1: -log p(ℓ(q) = 1| image)
- Pairwise

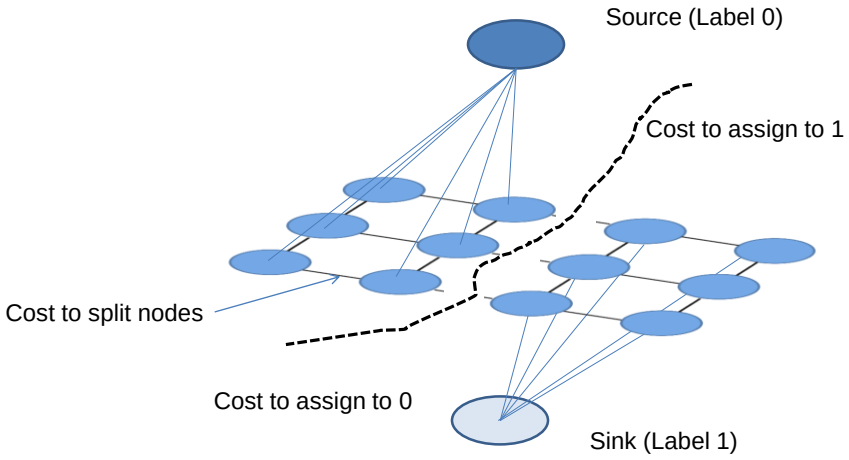
	0	1
0	0	K
1	K	0

K>0

Graph cuts



Graph cuts



Superpixels

- A popular approach currently in image segmentation is to first group pixels together into superpixels to form an oversegmentation of the image.
- A superpixel consists of a connected group of pixels that have similar properties, like colour.
- Then, one can group superpixels together to form the final segmentation. This can be more efficient than processing each pixel in the image.

SLIC

- SLIC (Simple Linear Iterative Clustering) is a popular superpixel segmentation technique, recently published by [Achanta et al. 2012]
- It performs a local grouping of pixels in 5D space defined by the LAB colours and x, y positions of the pixels [LABXY]. A distance measure in this 5D space combines distance measured in colour and the image itself
- It defines K regularly spaced cluster centres, and moves them to locations where the edge response is smallest, and assigns pixels to clusters based on distance.

Example

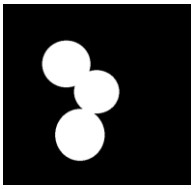
- Using following imports
  - `from skimage.segmentation import slic`
  - `from skimage.segmentation import mark_boundaries`
- The syntax is

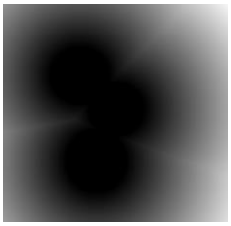
```
segments = slic(image, n_segments = numSegments, sigma = 5)
```

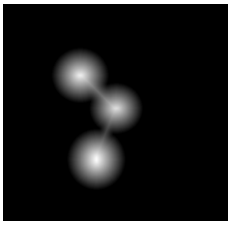
```
from skimage import io
image = img_as_float(io.imread("Sucess-Kid.png"))
fig = plt.figure("Superpixels -- %d segments" % (numSegments))
ax = fig.add_subplot(1, 1, 1)
ax.imshow(mark_boundaries(image, segments))
```

Distance transform

- A visual example





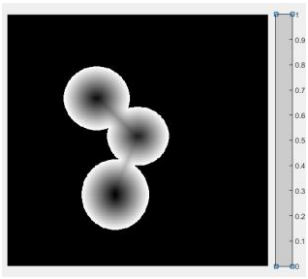


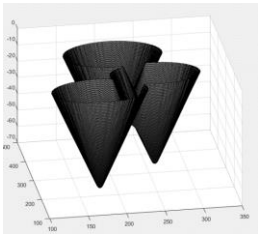
```
dist = cv2. distanceTransform(np.uint8(~(I.astype(np.bool_))),
cv2.DIST_L2, 3)
dist = dist / dist.max()
cv2.imshow('dist image', dist)
cv2.waitKey(0)
```

```
dist = cv2.distanceTransform(I, cv2.DIST_L2, 3)
dist = dist / dist.max()
cv2.imshow('dist image', dist)
cv2.waitKey(0)
```

Watershed segmentation

- Consider an image D as a topographical surface, where the intensity of the image is mapped to a height.

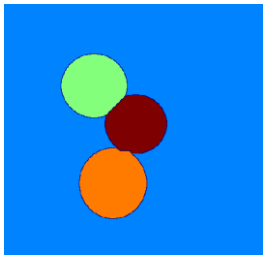




- Imagine it starts raining. Excluding the background, water will start to accumulate in the basins (lowest points). We can think of each basin as a region. As water is continually added, eventually regions will collide at *ridge lines*.

Watershed segmentation

- The watershed function in OpenCV computes these ridge lines, and returns a matrix L with labels grouping pixels into regions based on the catchment basin.

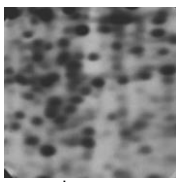


```
# Marker labelling
ret, markers = cv.connectedComponents(sure_fg)imshow(L, []
markers = cv.watershed(img,markers)
```

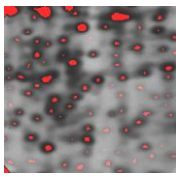
- Here, multiple regions are detected, even though circles are connected.

Marker-controlled watershed

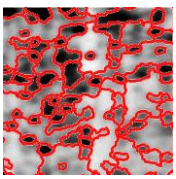
- On a grayscale image, one typically applies watershed to the distance transform of an edge map. However, this will result in *over-segmentation*.
- Instead, one can provide markers (also known as seeds). Each seed defines a catchment basin (region in the final segmentation).
- Nearby regions grow until they collide with other regions.



image



markers (on image)



segmentation

```
I = cv2.imread('cells2.png', 0)
I = (I / 255.)
K = 0.5* np.asarray([[-1, 0, 1]])
Ix = cv2.filter2D(I, -1, K)
Ix = np.clip(Ix, 0, 1)
Ix = Ix / Ix.max()
Iy = cv2.filter2D(I, -1, K.T)
Iy = np.clip(Iy, 0, 1)
Iy = Iy / Iy.max()
E = np.sqrt(Ix**2 + Iy**2)
```

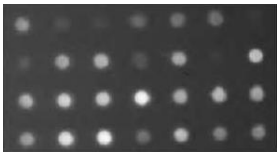
```
B = imregionalmin(I);
```

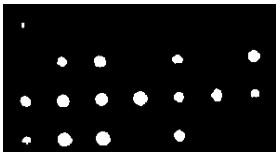
```
Enew = imimposemin(E, B);
L = cv2.watershed(Enew);
I[L == -1] = [255,0,0]
cv2.imshow('result', I);
```

* imregoionalmin, imimposemin does not exist in opencv built-in. You have to implement on your own if you want to run this code.

Hysteresis thresholding

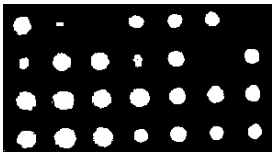
- Hysteresis thresholding applies *two thresholds* to achieve a segmentation.
- An region will be labelled as foreground if
  - It contains connected to a pixel with intensity greater than T_{high} *and*
  - All its pixels have an intensity greater then T_{low},
- Goal: Perform a segmentation of the brighter blobs





$I > T_{high}$

Finds brighter blobs, but shapes are not correct



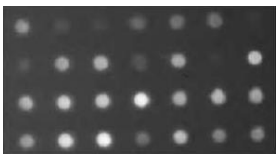
$I > T_{low}$

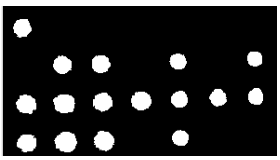
Shapes are (mostly) correct but darker blobs included

- Hysteresis thresholding will select brighter blobs using T_{high}, and for those blobs, use the shape from T_{low}.

Hysteresis thresholding

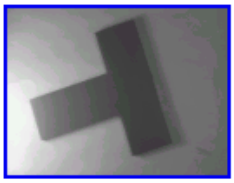
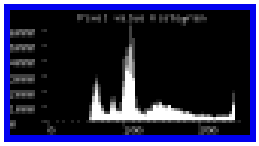
```
I = cv2.imread("dots.png",0)
cv2.imshow('image', I); cv2.waitKey(0)

Thigh = 150
Tlow = 80
H = np.uint8((I > Thigh) * 255)
cv2.imshow('High threshold', H); cv2.waitKey(0)

L = np.uint8((I > Tlow) * 255)
cv2.imshow('Low threshold', L); cv2.waitKey(0)

ret, markers = cv2.connectedComponents(L)
valid_markers = np.unique(markers*(H/255))
isin = np.isin(markers, valid_markers[1:])
cv2.imshow('Hysteresis tresholding result', np.uint8(isin
* 255))
cv2.waitKey(0)
```





This code labels the regions detected using the low thresholded image (L), and uses the high thresholded image (H) to select valid regions. The pixels from L that are part of a valid region are retained.

Note: If the image does not have a bimodal histogram (i.e. either has no distinct peak or it has more than 1 peak) then thresholding won't work!

Threshold = 80 Threshold = 120

You have to go with other segmentation methods, e.g. A local threshold approach
Or if possible remove the background illumination effects, …

Otsu's Thresholding Method (1979)

- Based on a very simple idea: Find the threshold that *minimizes the weighted within-class variance.*
- This turns out to be the same as *maximizing the between-class variance.*

- Operates directly on the gray level histogram [*e.g.* I= 256 numbers, P(i)], so it's fast (once the histogram is computed).
- used with considerable success in various situations.

Otsu: Assumptions

- Histogram (and the image) are *bimodal.*
- Assumes uniform illumination (implicitly), so the bimodal brightness behavior arises from object appearance differences only.
- No use of *spatial coherence,* nor any other notion of object structure.

The *weighted within-class variance* is:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Where the class probabilities are estimated as:

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

And the class means are given by:

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

Finally, the individual class variances are:

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

Now, we could actually stop here. All we need to do is just run through the full range of t values [1,l=256] and pick the value that minimizes $\sigma_w^2(t)$.

But the relationship between the within-class and between-class variances can be exploited to generate a recursion relation that permits a much faster calculation.

Between/Within/Total Variance

- The basic idea is that the total variance does not depend on threshold (obviously).
- For any given threshold, the total variance is the sum of the within-class variances (weighted) and the *between class variance*, which is the sum of weighted squared distances between the class means and the grand mean.

The between-class variance is defined as:

$$\sigma_B^2(t) = q_1(t)(\mu_1(t) - \mu_g)^2 + q_2(t)(\mu_2(t) - \mu_g)^2$$

Global mean: $\mu_g = \sum_{t=1}^{l} i \; P(i)$

Note that $q_2(t) = 1 - q_1(t)$

One can express the total variance as:

$$\sigma^2(t) = \underbrace{\sigma_w^2(t)}_{\text{Within-class variance}} + \underbrace{\sigma_B^2(t)}_{\text{Between-class variance}}$$

Since the total is constant and independent of *t*, the effect of changing the threshold is merely to move the contributions of the two terms back and forth.

So, *minimizing the within-class variance* $\sigma_w^2(t)$ *is the same as maximizing the between-class variance* $\sigma_B^2(t)$

The nice thing about this is that we can compute the quantities in $\sigma_B^2(t)$ *recursively* as we run through the range of *t* values when we compute $\sigma_B^2(t)$

Algorithm:

Initialization... $q_1(1) = P(1);\ m_1(0) = 0$

Recursion... {i.e. compute $s_B^2(t)$ *for all t (all intensity values)using the following:}*

$$q_1(t+1) = q_1(t) + P(t+1)$$

$$m_1(t+1) = \frac{q_1(t)\,m_1(t) + (t+1)P(t+1)}{q_1(t+1)}$$

$$m_2(t+1) = \frac{m - q_1(t+1)\,m_1(t+1)}{1 - q_1(t+1)}$$

End... Pick t for which $s_B^2(t)$ *is maximized as the threshold*

Gaussian Mixture Modeling

Based on approximation of the histogram of an image using a weighted sum of two or more probability densities with normal distribution.

- Suppose that region intensities are each normal distributions (Gaussians).

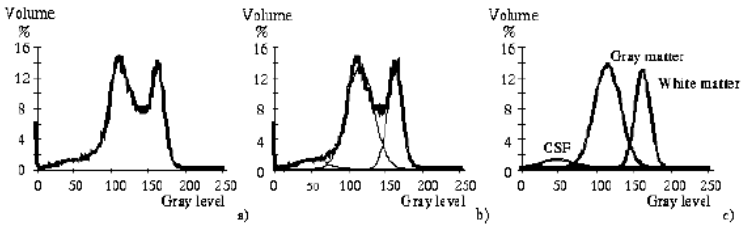


Figure 5.5 *Segmentation of 3-D T1-weighted MR brain image data using optimal thresholding: (a) Local gray level histogram, (b) fitted Gaussian distributions; global 3-D image fit, (c) Gaussian distributions corresponding to WM, GM, and CSF.*

Gaussian Mixture Modeling

- Each of the distributions has a mean (μ_i) and a standard deviation (σ_i) *independent of the threshold we choose.*
- Foreground/background case:

$$h_{\text{model}}(g) = n_B \; e^{-(g-\mu_B)^2/2\sigma_B^2} + n_O \; e^{-(g-\mu_O)^2/2\sigma_O^2}$$

- *Assumes that there exists two distributions and we must find them.* (Once we know parameters, determining the best threshold is easy.)
- Six unknown parameters: $\vec{\mathbf{x}} = n_B, n_O, \mu_B, \mu_O, \sigma_B$, and σ_O
- How well does the sum of the distributions approximate the histogram?

$$F(\vec{\mathbf{x}}) = \sum_{0}^{N-1} \left[h_{\text{model}}(g) - h_{\text{image}}(g) \right]^2$$

- Choose parameters to minimize the error in the fit (F). [We won't cover the algorithm for doing this, just the basic idea.]

Bryan Morse, BYU

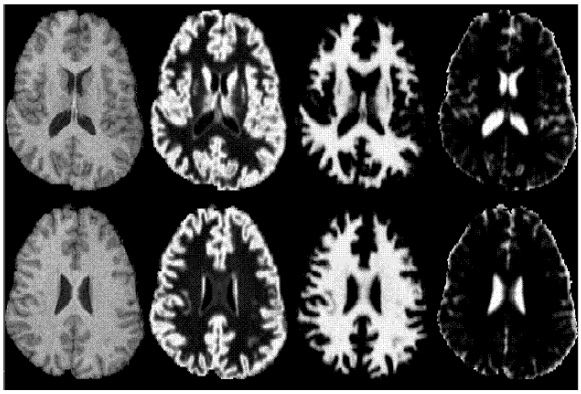
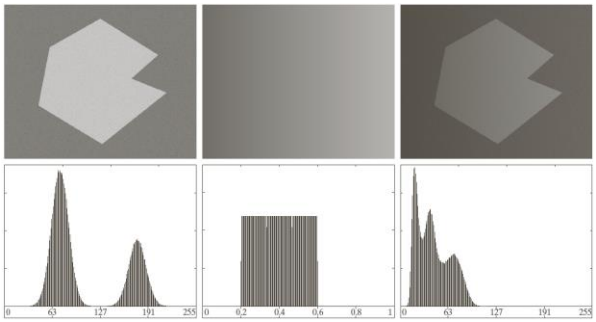


Figure 5.6 *Optimal MR brain image segmentation. Left column: Original T1-weighted MR images, two of 120 slices of the 3-D volume. Middle left: Partial-volume maps of gray matter. The brighter the voxel, the higher is the partial volume percentage of gray matter in the voxel. Middle right: Partial-volume maps of white matter. Right column: Partial-volume maps of cerebro-spinal fluid. Courtesy R.J. Frank, T.J. Grabowski, Human Neuroanatomy and Neuroimaging Laboratory, Department of Neurology, The University of Iowa.*

Effect of Intensity Inhomogeneity in the Background

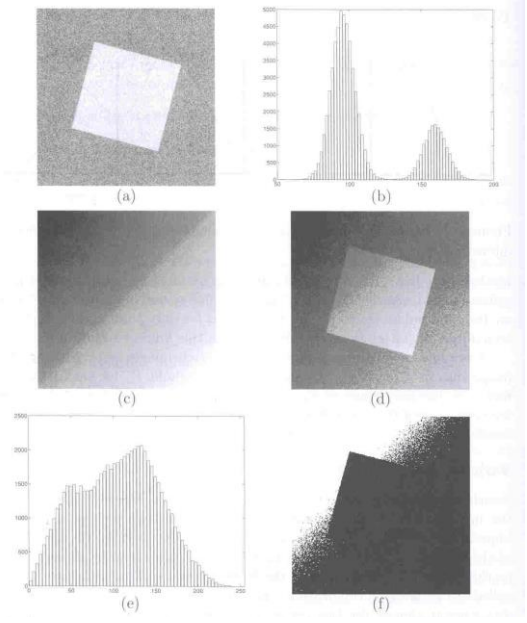


a b c
d e f

FIGURE 10.37 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d)–(f) Corresponding histograms.

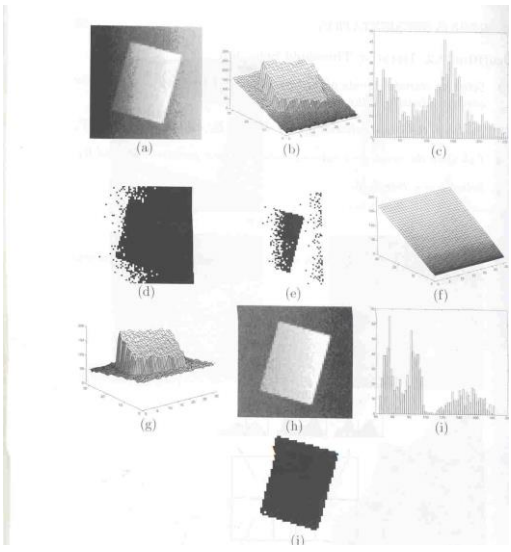
Iterative threshold won't work!

Basic segmentation with e.g. İterative thresholding won't work when there is background intensity variation in the image



Background Normalization for better Thresholding

- Approximate the intensity values of the image by as simple function such as a plane.
- The function fit is determined in large part by the gray value of the background.
- Hence the name Background normalization: one obtains estimate of the variable background.
- Correct the image by subtracting the background estimate
- Now one can threshold the image appropriately



Background Light Intensity Problem in Historic Documents

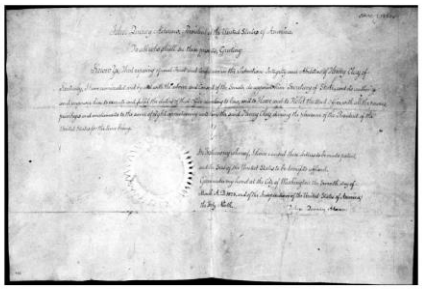


Figure 2: Historical handwritten document image with uneven background.

Figure 7: It is impossible for the original document image in Figure 2 to be segmented using a global threshold. The best global threshold we could manually find is at 200 and the binarized image shows significant parts of text obliterated.

"Historical Handwritten Document Image Segmentation Using Background Light Intensity Normalization" by Zhixin Shi and Venu Govindaraju

Background Intensity Normalization for historic documents

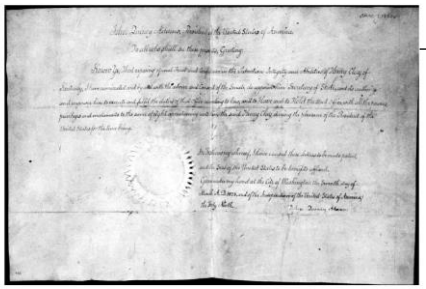
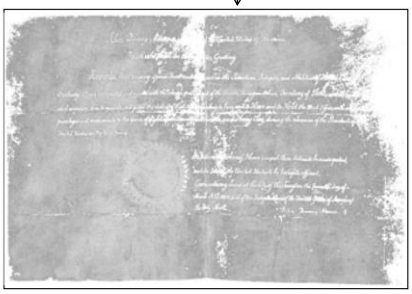


Figure 2: Historical handwritten document image with uneven background.

Threshold to find a mainly background image



Plane fit to background image ←

Figure 3: Pixels below the thresholding plane cover most of the background.

"Historical Handwritten Document Image Segmentation Using Background Light Intensity Normalization" by Zhixin Shi and Venu Govindaraju

Background Intensity Normalization

Let us treat an image function as a 3D object (x,y,z): i.e. as a graph of (x,y) pixel coordinates, and z=I(x,y) image gray value.

Goal: find a plane H with an orientation over the x-y plane using the pixel coordinates (x_i,y_i) and their intensity values (z_i) coming mainly from the background (x_i,y_i,z_i) (e.g. One can do a simple tresholding to select mainly bg)

Using a plane equation: $Ax + By - z + d = 0$

We can minimize: $\min \sum_i (Ax_i + By_i - z_i + d)^2$

where the sum is for all the available points in the background image (thresholding result)

Background Intensity Normalization

"The best fit" linear plane is found (Note: one can also partition the image into smaller regions and find different planes for each sub-region)

That is, the parameters A, B, d.

Now the approximate background image value for each pixel coordinate (x_i,y_i) is:

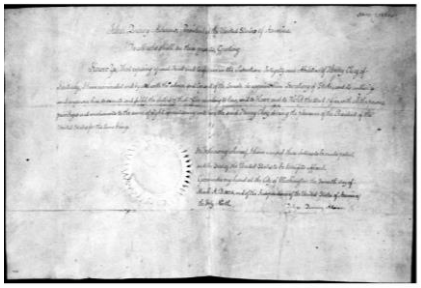
$$z(x_i, y_i) = Ax_i + By_i + d$$

Background subtracted image can be found by:

$$I_{corrected}(x, y) = I_{orig}(x, y) - z(x, y) + const$$

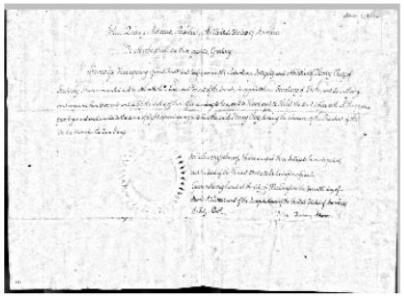
where one can appropriately scale the image (e.g. Either add a const such as 255 and normalize the estimated z values to (0,255).

Background Intensity Normalization for historic documents



I_{orig}

$I_{corrected}$



Now binarized with a global threshold

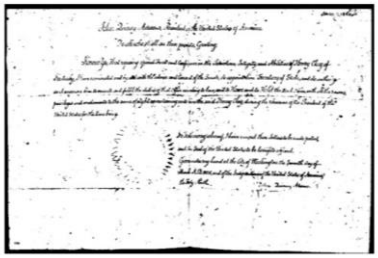


Figure 5: Normalized historical document image showing an even background.

Historical Handwritten Document Image Segmentation Using Background Light Intensity Normalization, Zhixin Shi and Venu Govindaraju