ECE5554 – Computer Vision
Lecture 6c – Region Analysis and Properties

Creed Jones, PhD

BRADLEY DEPARTMENT
OF ELECTRICAL & COMPUTER ENGINEERING

VIRGINIA TECH.

Today’s Objectives

Labeling Regions by Sequential Labeling (“Blobs”)
- Connected components
- blob statistics

Use of region descriptions

Chain code
- Differential chain code and shape number

Geometric features
- Compactness and Circularity

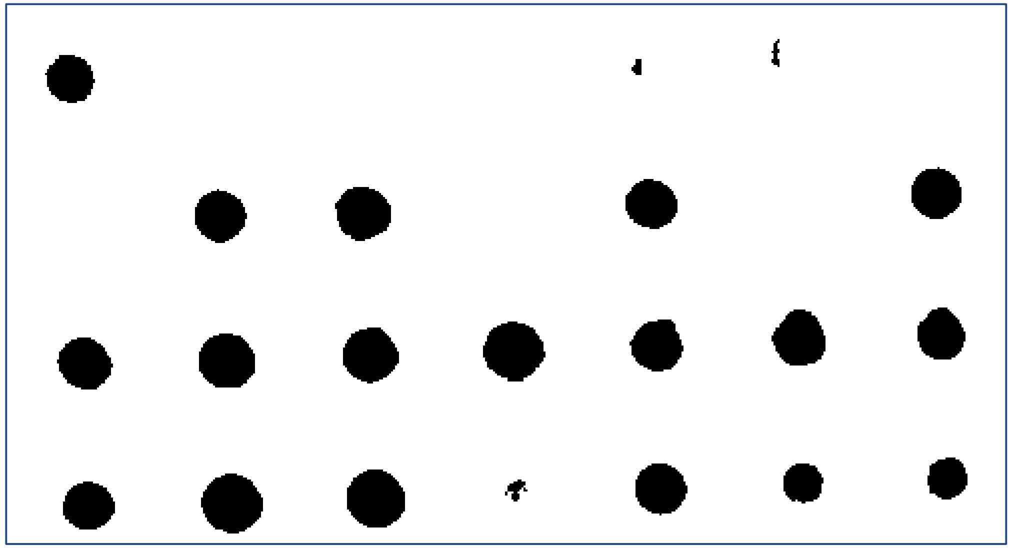
Statistical Shape Features
- Centroid
- Moments and Central Moments

Moment-based Features
- Orientation and Eccentricity
- Invariant moments - Hu’s and Zernike moments

The *Connected Components* or *Blobs* algorithm is used to locate and quantify connected regions in a binary image

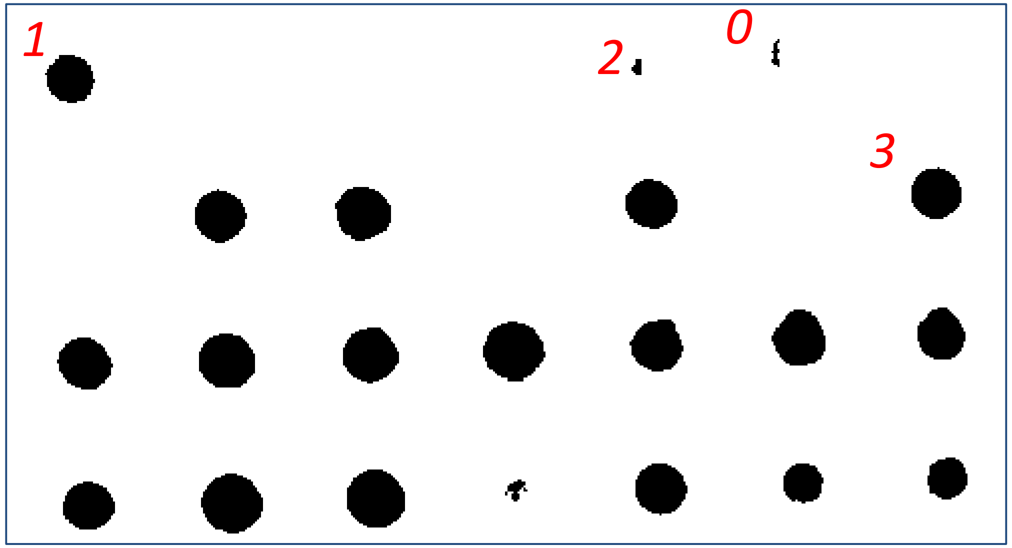
- Assuming that the image is divided into foreground and background
  - or can be binarized using a simple threshold
- Objects are defined as the connected foreground regions
  - objects can contain holes; holes can contain objects, etc…
- This algorithm will give us a whole set of descriptions for each object
- The algorithm examines each pixel to see which object it is in, and then updates the statistics for that object to incorporate that pixel
- Sometimes called the “blobs” algorithm

Here is a typical set of data from the blobs algorithm on the sample dot-blot image (binarized)



Blob #0, Area=34.0; BoundingBox=[382, 385; 385, 385] ; Perimeter=28; Circularity=26.308484235294117
Blob #1, Area=450.0; BoundingBox=[19, 43; 21, 26] ; Perimeter=72; Circularity=12.945228835555556
Blob #2, Area=30.0; BoundingBox=[312, 316; 314, 314] ; Perimeter=19; Circularity=11.57171413333333
Blob #3, Area=506.0; BoundingBox=[452, 476; 465, 465] ; Perimeter=71; Circularity=11.819875288537547
Blob #4, Area=500.0; BoundingBox=[309, 334; 318, 318] ; Perimeter=72; Circularity=12.134327328
Blob #5, Area=581.0; BoundingBox=[164, 191; 173, 173] ; Perimeter=78; Circularity=12.250059373493977
Blob #6, Area=519.0; BoundingBox=[94, 119; 105, 119] ; Perimeter=73; Circularity=12.336339606936415
Blob #7, Area=477.0; BoundingBox=[455, 478; 465, 465] ; Perimeter=69; Circularity=11.931223480083856
Blob #8, Area=569.0; BoundingBox=[382, 408; 393, 393] ; Perimeter=78; Circularity=12.508408604569421
Blob #9, Area=522.0; BoundingBox=[311, 337; 325, 325] ; Perimeter=73; Circularity=12.265441103448273
Blob #10, Area=688.0; BoundingBox=[238, 268; 250, 250] ; Perimeter=86; Circularity=12.71110793023256
Blob #11, Area=590.0; BoundingBox=[168, 195; 180, 193] ; Perimeter=76; Circularity=11.969575077966102
Blob #12, Area=619.0; BoundingBox=[96, 123; 106, 106] ; Perimeter=79; Circularity=11.65009111470113
Blob #13, Area=538.0; BoundingBox=[25, 52; 35, 35] ; Perimeter=74; Circularity=12.30259988104089
Blob #14, Area=330.0; BoundingBox=[460, 479; 469, 469] ; Perimeter=59; Circularity=11.960166593939393
Blob #15, Area=517.0; BoundingBox=[314, 339; 322, 322] ; Perimeter=72; Circularity=11.972752371373304
Blob #16, Area=315.0; BoundingBox=[388, 407; 394, 394] ; Perimeter=57; Circularity=12.09006648888889
Blob #17, Area=674.0; BoundingBox=[170, 198; 180, 180] ; Perimeter=82; Circularity=11.736385163204748
Blob #18, Area=703.0; BoundingBox=[97, 127; 109, 109] ; Perimeter=83; Circularity=12.000078384068278
Blob #19, Area=58.0; BoundingBox=[249, 259; 255, 255] ; Perimeter=37; Circularity=29.243480275862066
Blob #20, Area=486.0; BoundingBox=[28, 53; 38, 38] ; Perimeter=71; Circularity=12.058042600823043

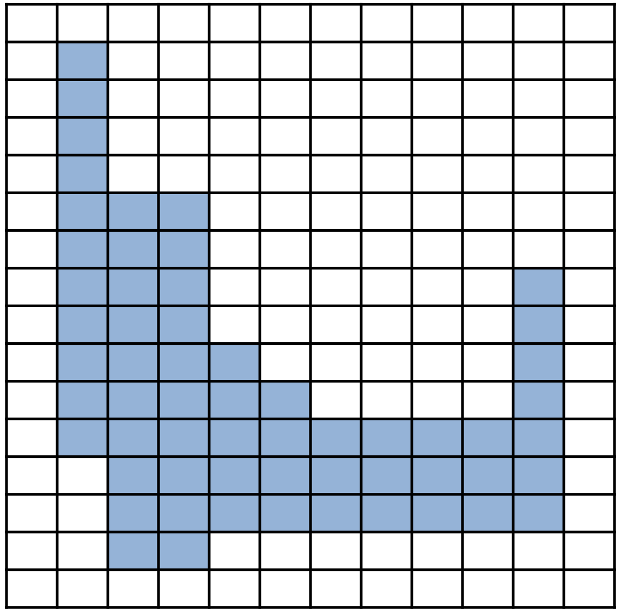
Here is a typical set of data from the blobs algorithm on the sample dot-blot image (binarized)



Blob #0, Area=34.0; BoundingBox=[382, 385; 385, 385] ; Perimeter=28; Circularity=26.308484235294117
Blob #1, Area=450.0; BoundingBox=[19, 43; 21, 26] ; Perimeter=72; Circularity=12.945228835555556
Blob #2, Area=30.0; BoundingBox=[312, 316; 314, 314] ; Perimeter=19; Circularity=11.5717141333333
Blob #3, Area=506.0; BoundingBox=[452, 476; 465, 465] ; Perimeter=71; Circularity=11.819875288537547
Blob #4, Area=500.0; BoundingBox=[309, 334; 318, 318] ; Perimeter=72; Circularity=12.134327328
Blob #5, Area=581.0; BoundingBox=[164, 191; 173, 173] ; Perimeter=78; Circularity=12.250059373493977
Blob #6, Area=519.0; BoundingBox=[94, 119; 105, 119] ; Perimeter=73; Circularity=12.336339606936415
Blob #7, Area=477.0; BoundingBox=[455, 478; 465, 465] ; Perimeter=69; Circularity=11.931223480083856
Blob #8, Area=569.0; BoundingBox=[382, 408; 393, 393] ; Perimeter=78; Circularity=12.508408604569421
Blob #9, Area=522.0; BoundingBox=[311, 337; 325, 325] ; Perimeter=73; Circularity=12.265441103448273
Blob #10, Area=688.0; BoundingBox=[238, 268; 250, 250] ; Perimeter=86; Circularity=12.71110793023256
Blob #11, Area=590.0; BoundingBox=[168, 195; 180, 193] ; Perimeter=76; Circularity=11.969575077966102
Blob #12, Area=619.0; BoundingBox=[96, 123; 106, 106] ; Perimeter=79; Circularity=11.65009111470113
Blob #13, Area=538.0; BoundingBox=[25, 52; 35, 35] ; Perimeter=74; Circularity=12.30259988104089
Blob #14, Area=330.0; BoundingBox=[460, 479; 469, 469] ; Perimeter=59; Circularity=11.960166593939393
Blob #15, Area=517.0; BoundingBox=[314, 339; 322, 322] ; Perimeter=72; Circularity=11.972752371373304
Blob #16, Area=315.0; BoundingBox=[388, 407; 394, 394] ; Perimeter=57; Circularity=12.09006648888889
Blob #17, Area=674.0; BoundingBox=[170, 198; 180, 180] ; Perimeter=82; Circularity=11.736385163204748
Blob #18, Area=703.0; BoundingBox=[97, 127; 109, 109] ; Perimeter=83; Circularity=12.000078384068278
Blob #19, Area=58.0; BoundingBox=[249, 259; 255, 255] ; Perimeter=37; Circularity=29.243480275862066
Blob #20, Area=486.0; BoundingBox=[28, 53; 38, 38] ; Perimeter=71; Circularity=12.058042600823043

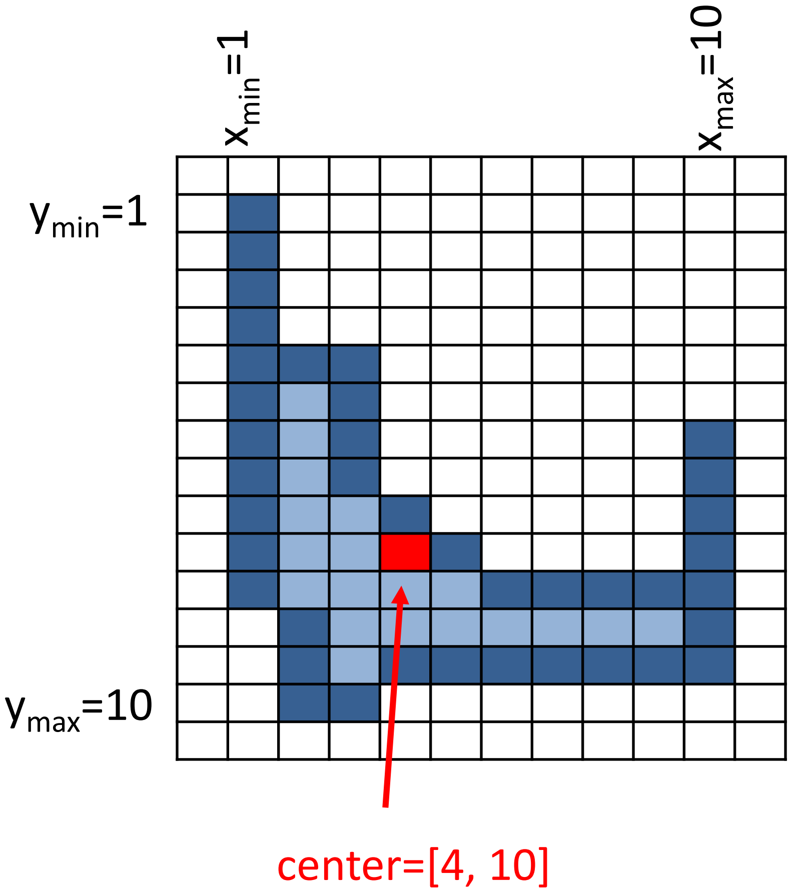
Common blob features describe the location, size and shape of the object – each image can contain many objects

- center
- "bounding box" –
  - xmin, xmax, ymin, ymax
- area – total # of pixels
- average gray-level
- perimeter - # boundary pixels
- # of holes – "Euler number"
- direction of "major axis"
- lots of other things…



Common blob features describe the location, size and shape of the object – each image can contain many objects

- area = 59
- $[x_{min}, x_{max}] = [1, 10]$
- $[y_{min}, y_{max}] = [1, 10]$
  - pct fill = 59/100 = 59%
- sum(x)= 253
  - $x_c = 4.288$
- sum(y)= 571
  - $y_c = 9.677$
- perimeter = 47

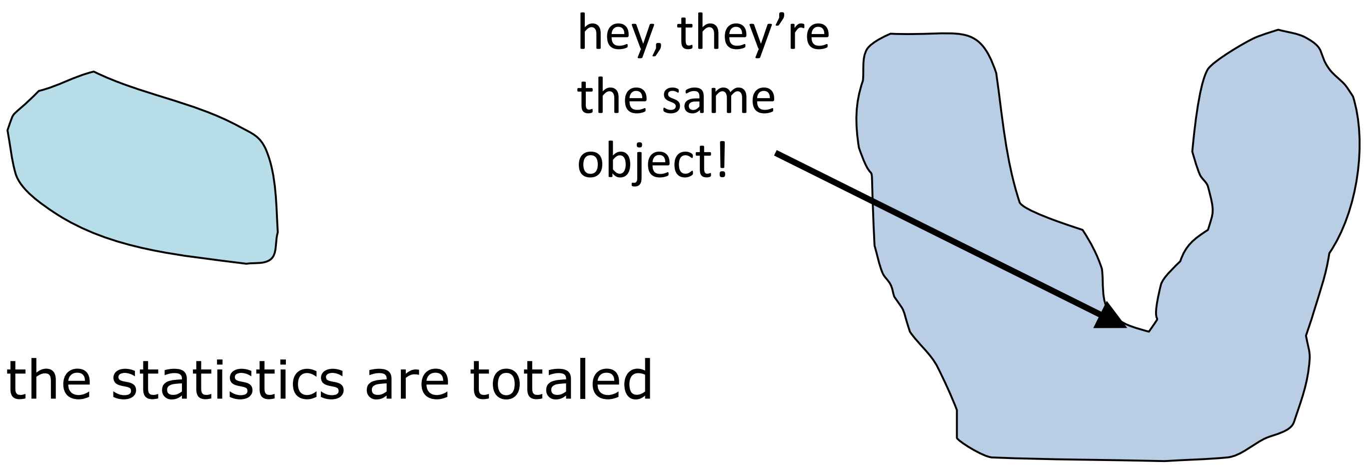


Here is a list of the features returned by MATLAB's connected components function

```
'Area'
'BoundingBox'
'Centroid'
'ConvexArea'
'ConvexHull'
'ConvexImage'
'Circularity'
'Eccentricity'
'EquivDiameter'
'EulerNumber'
'Extent'
'Extrema'
'FilledArea'
'FilledImage'
'Image'
'MaxFeretProperties'
'MinFeretProperties'
'MajorAxisLength'
'MinorAxisLength'
'Orientation'
'Perimeter'
'PixelIdxList'
'PixelList'
'Solidity'
'SubarrayIdx'
'MaxIntensity'
'MeanIntensity'
'MinIntensity'
'PixelValues'
'WeightedCentroid'
```

How does it work?

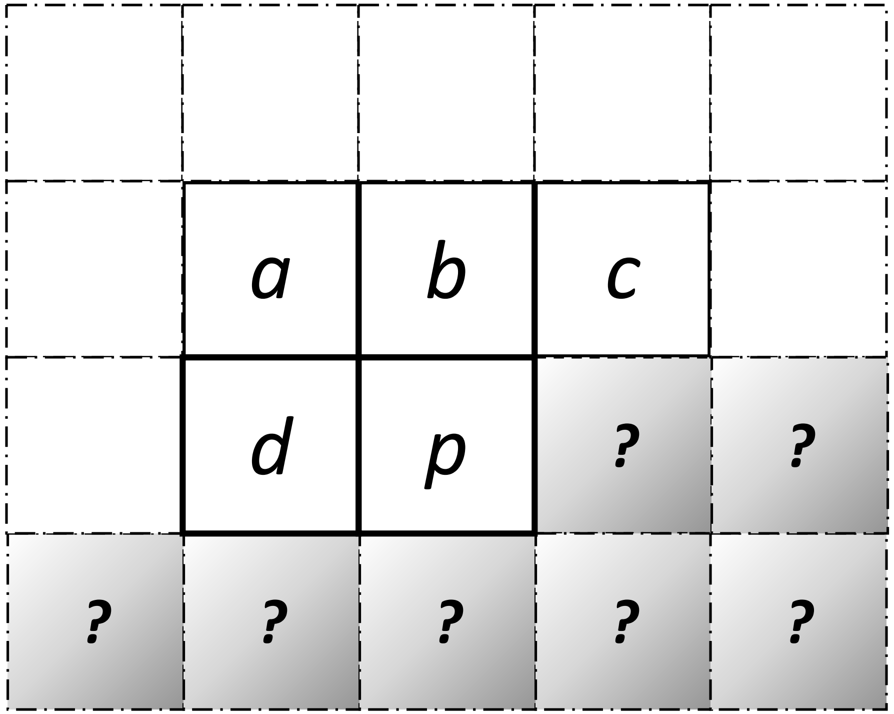
- The connected components or “blobs” algorithm builds a list of objects in the image
  - all foreground pixels
- If different pieces are connected, then the two objects are linked



- At the end, all of the statistics are totaled

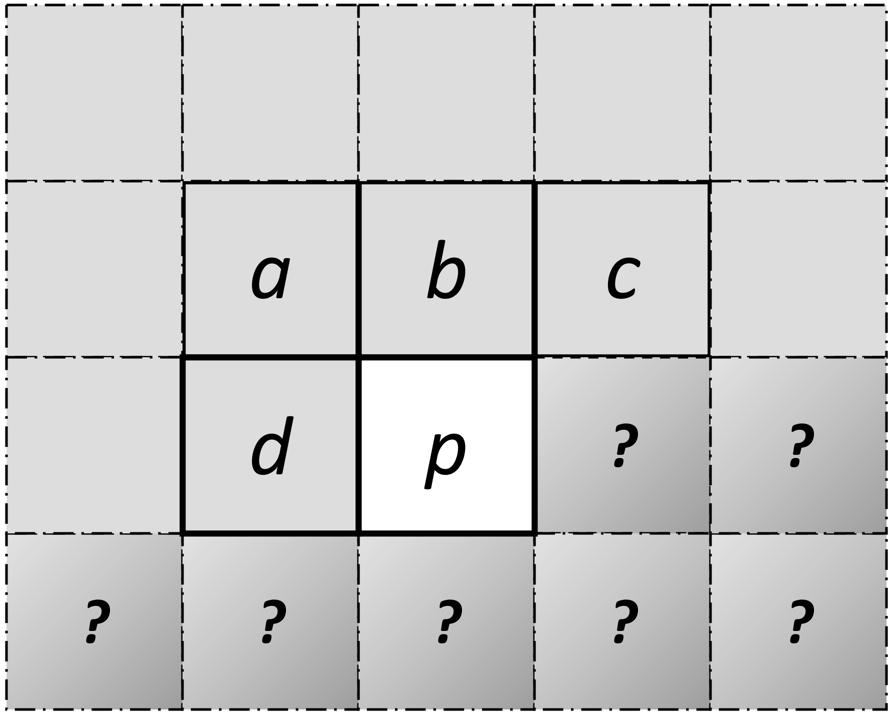
Each pixel is examined to update statistics in a list of image objects and holes

- The image is scanned in raster order
- In the crucial step, each pixel p is examined to see if it is part of an object above or to the left of it
- If so, it may actually combine two objects that had been seen as separate up to now
- Foreground pixels are "colored" with the blob number – the intensity is replaced by the number of the component that this pixel is part of
  - Note this may change as further linkages are developed!



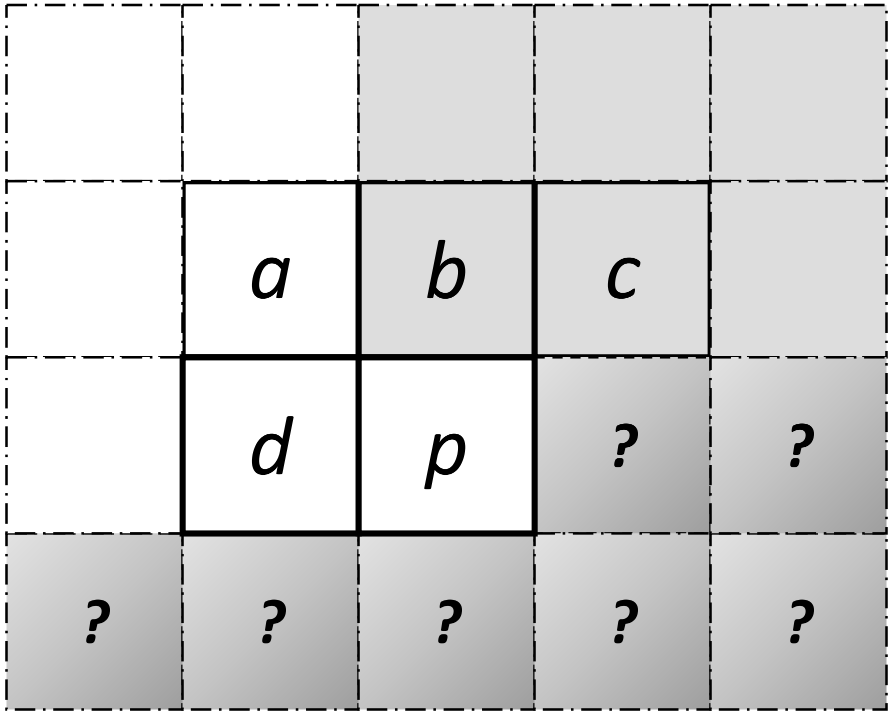
If p is the only foreground pixel in the neighborhood, we have found a new object (this pixel is along the top of it.

- We examine a 5-pixel neighborhood, as shown
- If only p is foreground, we assume it is the top of a new object – a blob
- Create a new object to store the blob information
- area = 1
  minx = b
  etc…



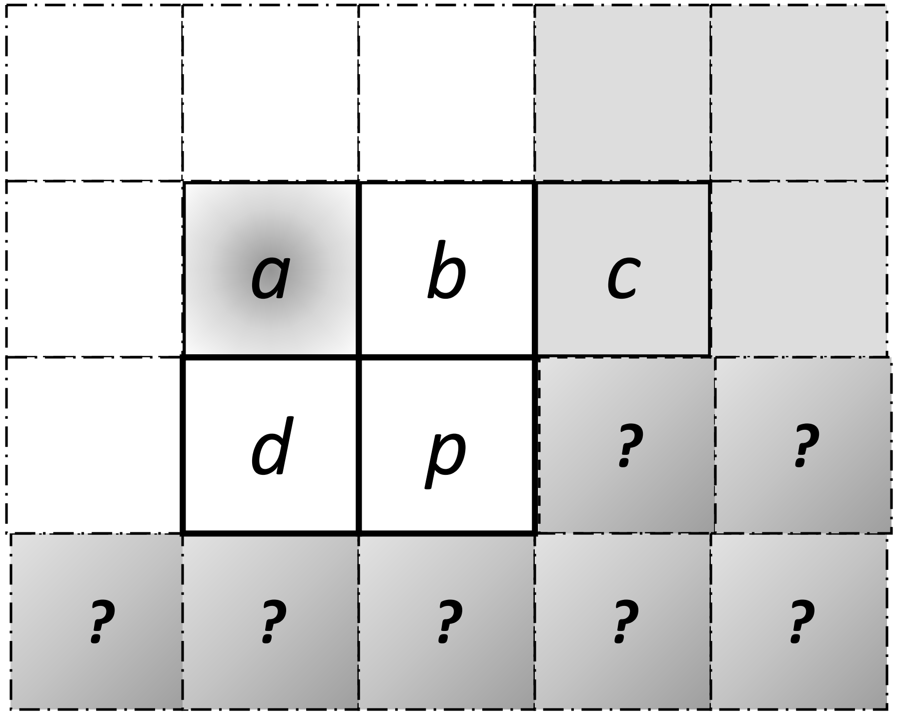
If p and a are foreground…

- If a is foreground, then don’t create a new blob, just add the pixel p to a’s object
- a.area++ a.maxy = y of p
- etc…



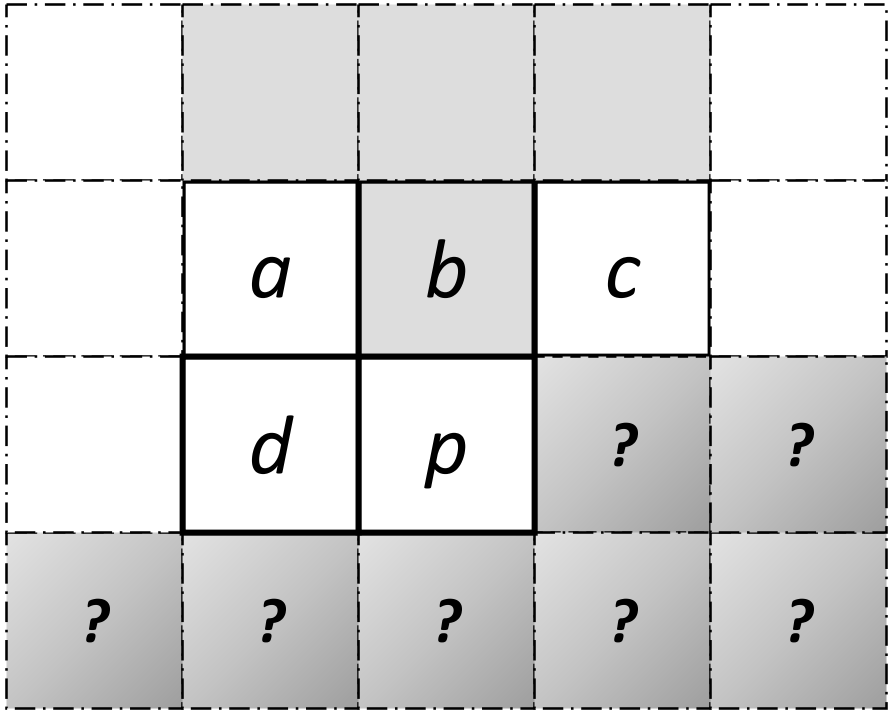
If p and b are foreground…

- If b is foreground, we have the same situation, just add the pixel p to b's object
- b.area++ b.maxy = y of p
- etc…



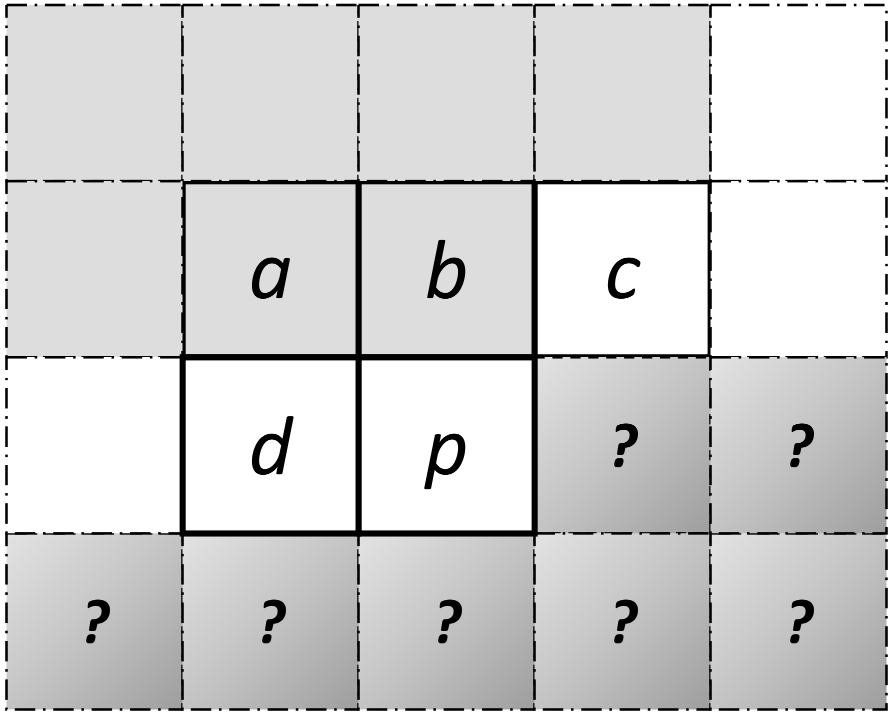
If p, a and c are all foreground, but b is background…

- Well, now we have an interesting situation
- We add p to the blob containing pixel c
- But, we also make a note that what we thought were two separate blobs are actually connected…



If p, d and c are all foreground (but a and b are background)…

- Same thing –
- not only does p have to be added to c’s blob
- We also combine the blobs of c and d
- When the whole image is processed, we come back and total up all the stats



When a pixel is examined that links two previously separate blobs, their statistics must be combined

- We also recall that these two blobs are, in fact, the same
  - So that we can recolor the pixels to have the same blob number (typically choose the lowest blob number contained
- These complex connections are the most difficult portion of the algorithm
  - Its not too bad…
- This is an extremely powerful algorithm for extracting some features from an image
- We start with an image, binarize it and find connected components, and we have numeric descriptors of the objects
  - location, size, color, …
- Of course, if the binarization had problems, we're in trouble…

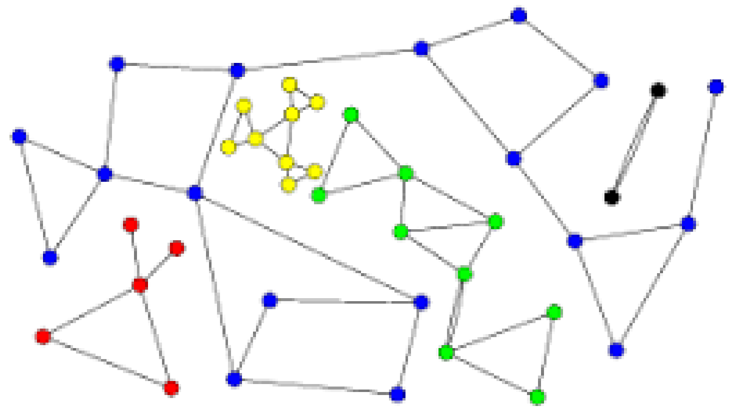
Graph theory also contains the concept of “connected components” – the two are related

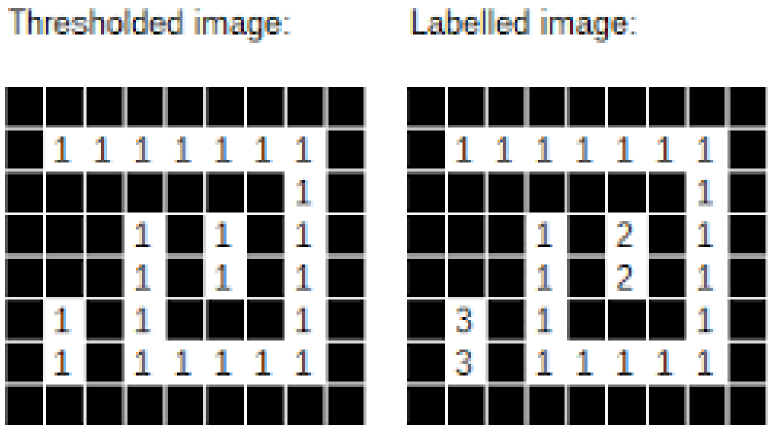
Graphs

- Determine regions of the graph that are connected
- Algorithms exist

Images

- Same, but consider adjacency (4- or 8-connected) as being “connected” in the graph sense

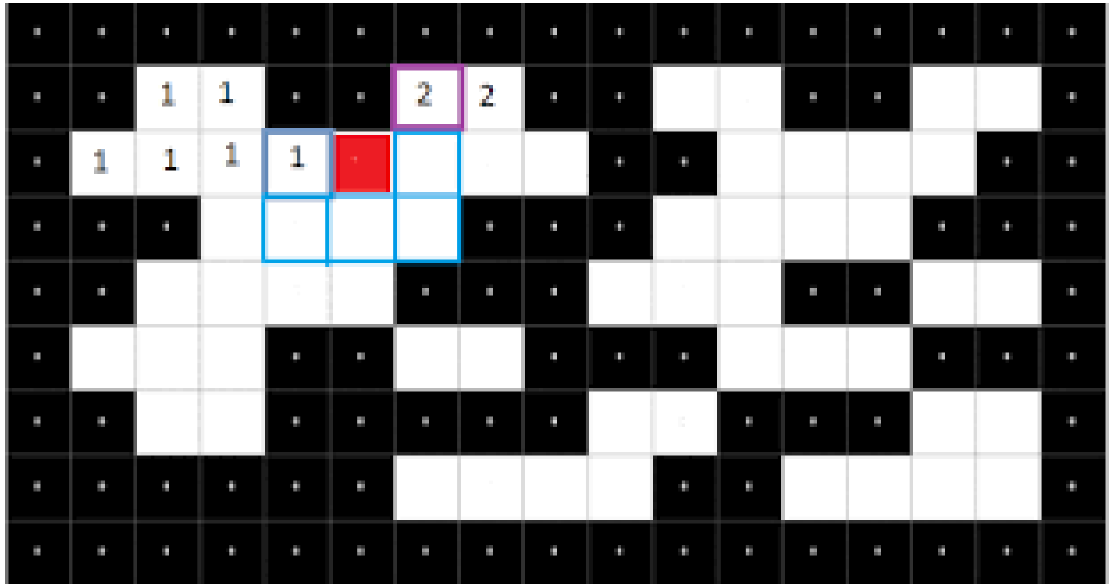




The aggregation (or agglomeration) step is a second pass through the image, to recolor all “first-pass” blobs that are connected to the same color

- In this example, we have been constructing two separate blobs - #1 and #2
- We now find that they are connected
- Add #2 to #1’s “equivalency” list
- Mark #2’s list as “merged”

- In the second pass, any pixel with color 2 will be recolored to 1
  – Same with any other color in 1’s equivalency list



ECE5554 FA19 HW2 part 1.py | ECE5554 FA19 HW2.py | ExpBlobs.py | ExpMeanShift.py

```python
# -*- coding: utf-8 -*-
"""
ExpBlobs.py
Created on Thu Sept 26 15:17:05 2019
@author: crjones4
"""

import numpy as np
import cv2

def showImage(img, name):
    cv2.imshow(name,img)
    return

##############################

def imshow_components(labels):
    # 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])

    # cvt to BGR for display
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

    # set bg label to black
    labeled_img[label_hue==0] = 0

    cv2.imshow('labeled.png', labeled_img)
    cv2.waitKey()

##############################


img = cv2.imread('C:\\Data\\airplanes.png', cv2.IMREAD_GRAYSCALE)
showImage(img, "Original")
thr, binary = cv2.threshold(255-img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  # ens
ret, labels = cv2.connectedComponents(binary)
for idx in range(5):
    area = (labels == idx).sum()
    print("Blob #" + str(idx) + ", Area=" + str(area))

#contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
#for cnt in contours:
#    ROI = cv2.boundingRect(cnt)
#    print(ROI)
#    momentArray = cv2.moments(cnt)
#    Hu = cv2.HuMoments(momentArray)
#    print(Hu)

imshow_components(labels)
```

Name	Type	Size	Value

Variable explorer | File explore

IPython console

Console 3/A

```
Python 3.7.4 (default, Aug
Type "copyright", "credits

IPython 7.7.0 -- An enhanc

In [1]: runfile('C:/Users/ jones4/.spyder-py3')
Blob #0, Area=197669
Blob #1, Area=939
Blob #2, Area=915
Blob #3, Area=896
Blob #4, Area=881
```

#	Surface (mm²)	Aspect ratio	Weighted surface	Sign	Span length (mm)	x span length (mm)	y span length (mm)
0	3.713	3.306	8.658	-	4.627	4.382	1.485
1	1.390	1.297	1.585	-	1.998	1.485	1.337
2	6.333	1.105	7.642	-	4.098	2.971	2.822
3	7.315	1.916	8.628	-	5.095	4.085	3.045
4	8.093	1.916	9.479	-	4.982	4.456	2.228
5	.1379	1.190	.1390	-	.5801	.4456	.3714
6	.1931	1.975	.1941	-	1.200	1.114	.4456
7	.6454	1.234	.6627	-	1.314	.8913	.9656
8	66.58	1.439	509.0	-	14.24	9.136	10.92
9	4.496	1.110	11.57	-	3.468	2.525	2.377
10	8.352	2.950	8.968	-	7.012	6.165	3.342
11	4.485	1.261	4.948	-	3.468	2.525	2.377

Z - Defects characterization AV = .3882, STD = .1708 , MIN = .1515 , MAX = 97.52 85.04 x 64.54 mm 49 %

Connected components (or blobs) can contain holes

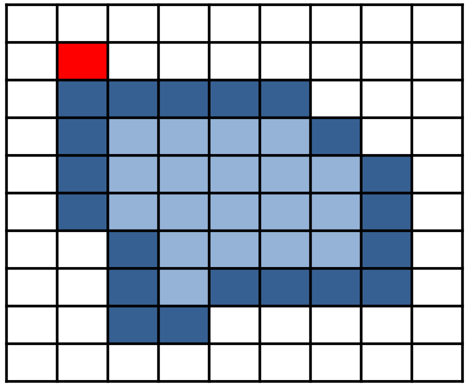
To characterize these, invert the polarity and trace the hole as if it were a foreground object

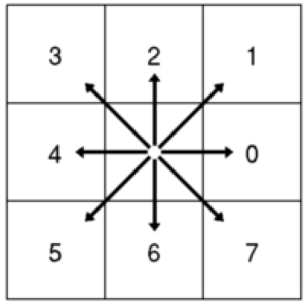
Or, maintain two lists while scanning – foreground pixels aggregate into blobs, while background pixels aggregate into holes

Keep track of “nesting” information – which hole is within which blob

One metric of interest is the *Euler number:* the total number of objects in the image minus the total number of holes in those objects.

We have briefly discussed the chain code form of representing an object; if the object has holes, we must trace them as well





- Also called the Freeman Chain Code of Eight Directions – FCCE
- Chain code:
  7, 0, 0, 0, 7, 7, 6, 6, 6, 4, 4, 4, 5, 4, 2, 2, 3, 2, 2, 2, 2
- The differential chain code encodes the change in direction at any point
- Differential chain code:
  5, 1, 0, 0, 7, 0, 7, 0, 0, 6, 0, 0, 1, 7, 6, 0, 1, 7, 0, 0, 0

To completely specify a binary image using chain codes, what do we need to record? How can we process the result to derive useful information

- For an absolute chain code, store the following for each object:
  - X and Y of a starting point
  - chain of directions of motion
  - the same for all holes in the object
- For a differential chain code, store the same for each object, as well as the default starting direction

- Some features are available from the chain code representation
  - number and perimeter of objects
  - number and perimeter of holes
  - bounding box

To enable comparison of chain codes in the presence of rotation, we can compute the *shape number*

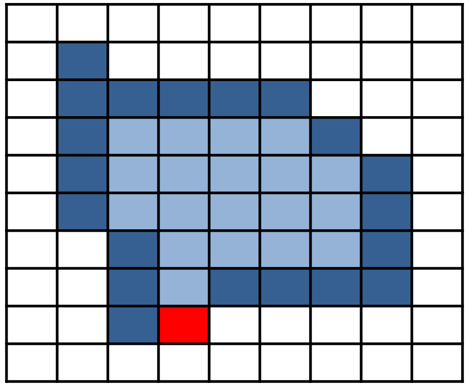
1. Form the differential chain code: 5,1,0,0,7,0,7,0,0,6,0,0,1,7,6,0,1,7,0,0,0
2. Form the set of all possible shifts of the code
3. Choose the numerically largest
4. This is the shape number – invariant over rotation

5 1 0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0
1 0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0 5
0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0 5 1
…
7 6 0 1 7 0 0 0 5 1 0 0 7 0 7 0 0 6 0 0 1
…

To enable comparison of chain codes in the presence of rotation, we can compute the *shape number*

1. Form the differential chain code
2. Form the set of all possible shifts of the code
3. Choose the numerically largest
4. This is the shape number – invariant over rotation

5 1 0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0
1 0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0 5
0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0 5 1
…
7 6 0 1 7 0 0 0 5 1 0 0 7 0 7 0 0 6 0 0 1
…

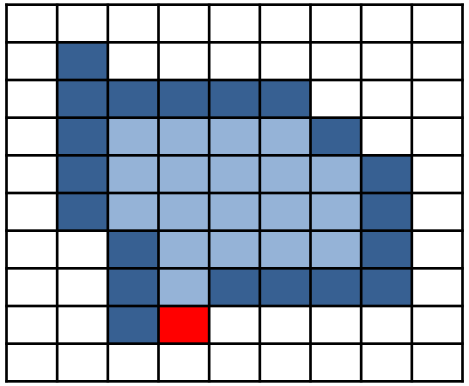


An approximate version of the object perimeter is the number of steps in the contour – but we can do better

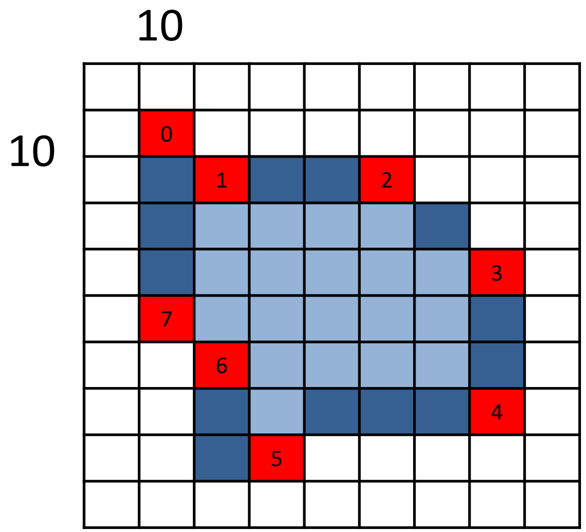
- We can correct for the diagonal nature of some contour steps
- Add 1 to the perimeter for orthogonal steps, $\sqrt{2}$ for diagonal ones
  - Diagonal ones have odd numbers in the FCCE
    5 1 0 0 7 0 7 0 0 6 0 0 1 7 6 0 1 7 0 0 0

$$Perimeter = \sum orthogonal\ steps + \sqrt{2} \sum diagonal\ steps = 13 + 8\sqrt{2} = 24.313$$

- Even this tends to overestimate the perimeter
- It's common to correct by multiplying by 0.95
- Perimeter = 23.75



Area can be roughly computed from the points of the polygon using the following formula (Gauss' formula) – NOTE: not all contour points need be considered!



This is also known as the *shoelace formula*

$$pts = \{(10,10),(11,11),(14,11),(16,13),(16,16),(12,17),(11,15),(10,14)\}$$

$$A(R) \approx= \frac{1}{2}\sum_{i=0}^{M-1}\begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix} = \frac{1}{2}\sum_{i=0}^{M-1} x_i y_{i+1} - y_i x_{i+1}$$

$$= \frac{1}{2}\begin{pmatrix} 10\cdot 11 - 10\cdot 11 + 11\cdot 11 - 14\cdot 11 + 14\cdot 13 - 16\cdot 11 + 16\cdot 16 - 16\cdot 13 \\ +16\cdot 17 - 12\cdot 16 + 12\cdot 15 - 11\cdot 17 + 11\cdot 14 - 10\cdot 15 \end{pmatrix}$$

$$= \frac{1}{2}(0 - 33 + 6 + 48 + 80 - 7 + 4) = 49$$

The actual pixel count is 40

There are adjustments to use Gauss' formula for small objects

The *isoperimetric quotient* is roughly invariant to changes in size; it's often computed relative to a circle with the same perimeter

A circle's area in terms of its perimeter:

$A = \pi r^2,\ P = 2\pi r,\ r = \frac{P}{2\pi},\ A = \frac{P^2}{4\pi}$

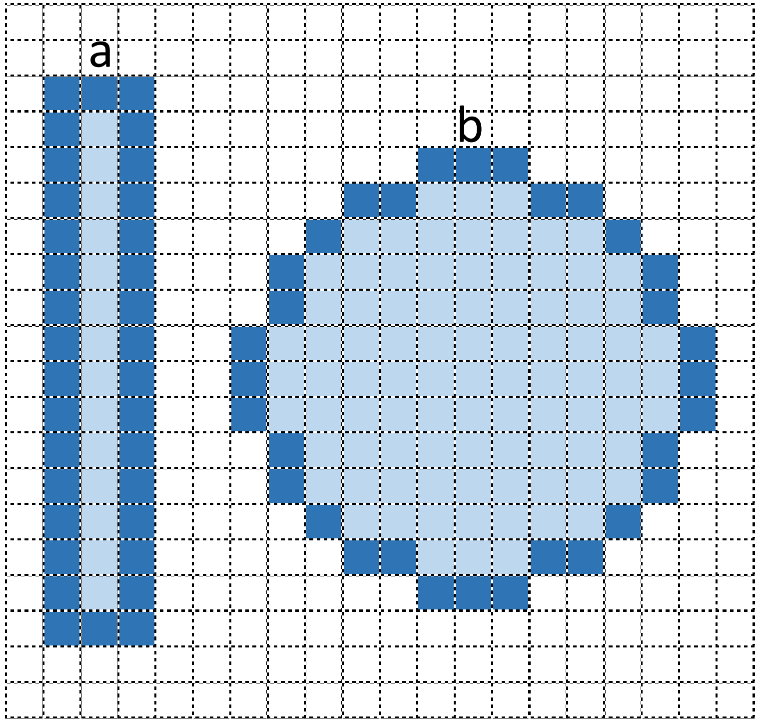
Isoperimetric quotient (ratio of area to the area of a circle with the same perimeter

Longer, thinner objects will have a lower isoperimetric quotient:

$$IQ_a = \frac{A}{\frac{P^2}{4\pi}} = \frac{A4\pi}{P^2} = \frac{48(4\pi)}{32.3^2} = 0.5781$$

$$IQ_b = \frac{A}{\frac{P^2}{4\pi}} = \frac{A4\pi}{P^2} = \frac{121(4\pi)}{36.696^2} = 1.129$$

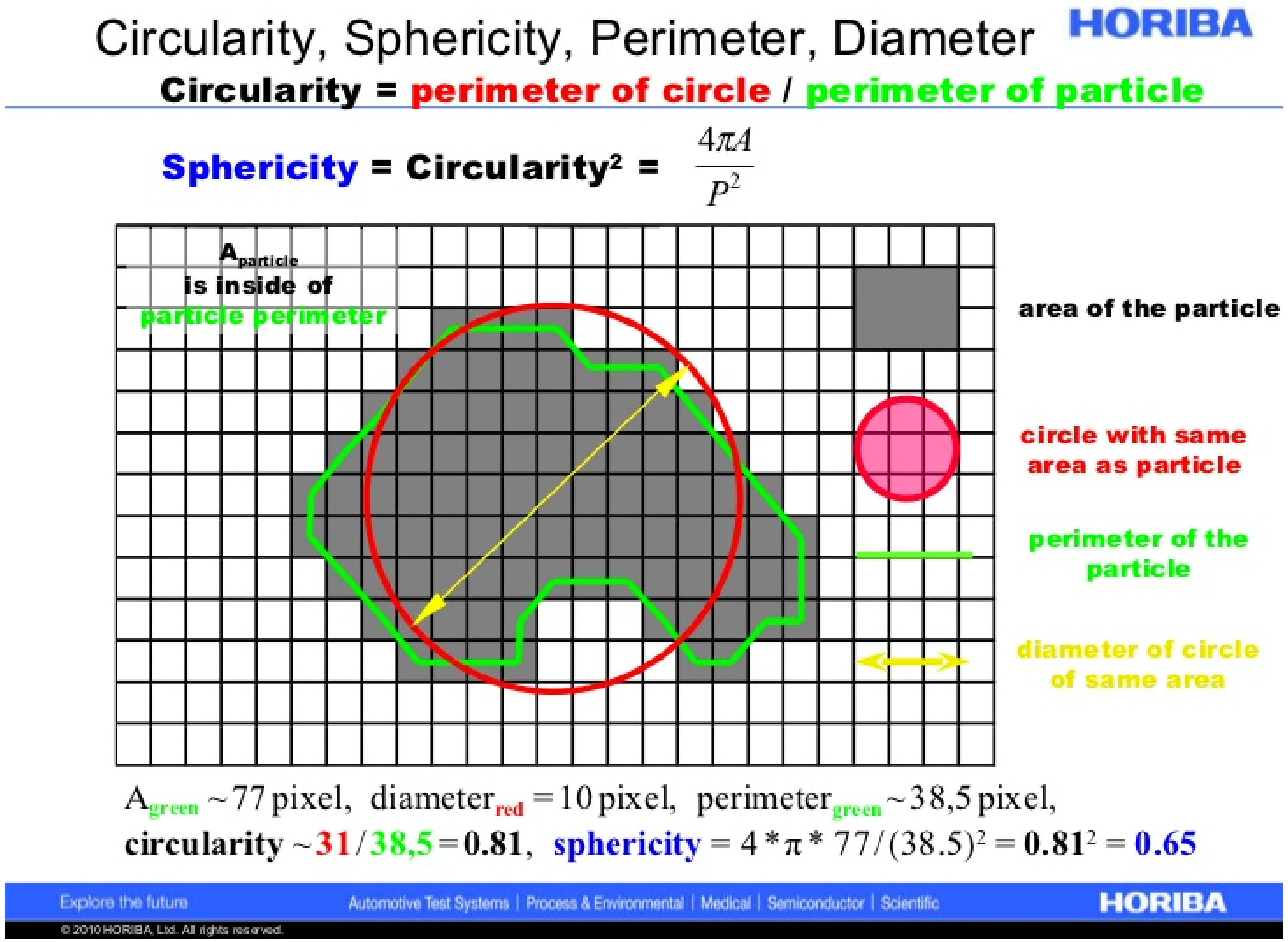
The IQ for the circle is not exactly 1 because of quantization effects



$$P_a = 0.95(34) = 32.3$$

$$P_b = 0.95\left(16 + 16\sqrt{2}\right) = 36.696$$

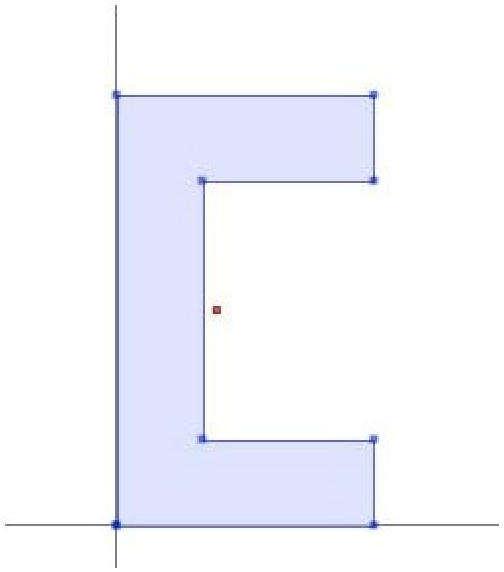
Another name for isoperimetric quotient is *circularity*; the maximum circularity is 1 (for a circle, of course)



The centroid of a binary region is the center point calculated as the average of all of the points' coordinates

- centroid = $\left[\frac{1}{N}\sum_{i=1}^{N} x_i \quad \frac{1}{N}\sum_{i=1}^{N} y_i\right]$

- The centroid can be outside the perimeter of the object



- The object need not be connected…

The intensity-weighted centroid also considers the original image intensity when computing the centroid (if the binary image came from a gray-scale image)

- intensity centroid =

$$\left[\frac{1}{N\sum I(x_i,y_i)}\sum_{i=1}^{N} x_i I(x_i,y_i) \quad \frac{1}{N\sum I(x_i,y_i)}\sum_{i=1}^{N} y_i I(x_i,y_i)\right]$$

- The intensity centroid will be offset in the direction of brighter pixels

- If the intensity centroid is above and to the left of the centroid, then the object is brighter on the upper left

Image moments are computed by summing pixel intensity values multiplied by powers of the coordinate values

- A family of moments are defined, for different values of p and q

$$M_{pq}(R) = \sum_{i=1}^{N} I(x, y) x^p y^q$$

- For a binary image:
  - $M_{00}(R)$ is the average intensity
  - $\frac{1}{M_{00}(R)}[M_{10}(R), M_{01}(R)]$ is the centroid
- Higher order moments are useful, but they are dependent on the location of the object

Central moments are computed around or *centered* on the centroid of the object and are insensitive to location of the object

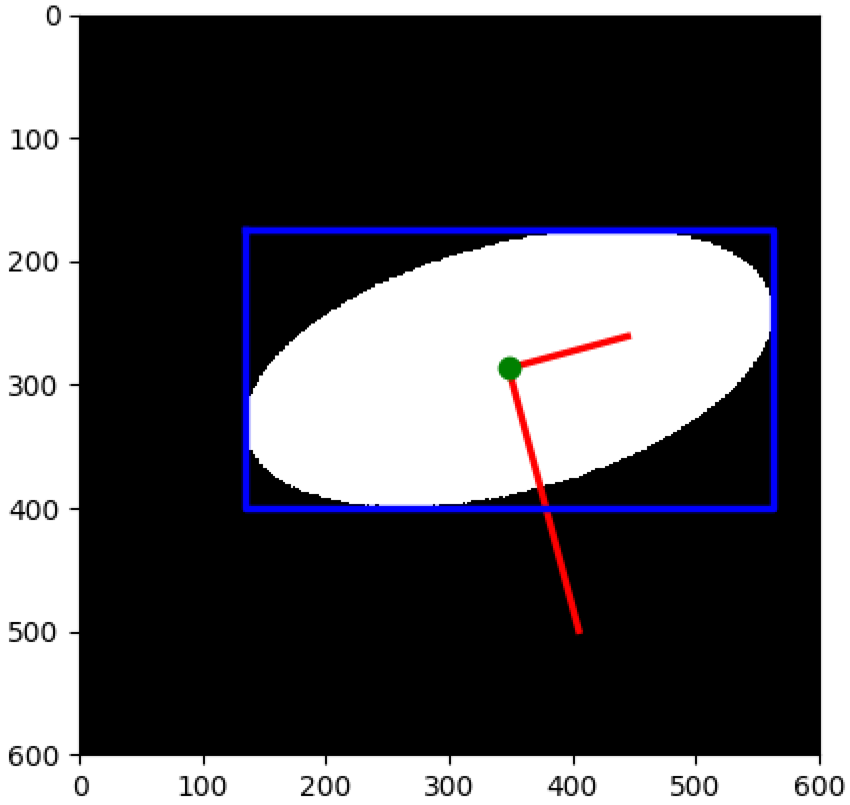
$$\mu_{pq}(R) = \sum_{i=1}^{N} I(x,y)(x-\bar{x})^p(y-\bar{y})^q$$

For binary images:

- $\mu_{00}(R) = \sum_{i=1}^{N} I(x,y)(x-\bar{x})^0(y-\bar{y})^0 = \sum_{i=1}^{N} I(x,y) = M_{00}(R)$ (average intensity)
- $\mu_{10}(R) = \sum_{i=1}^{N} I(x,y)(x-\bar{x})^1 = 0$
- $\mu_{01}(R) = \sum_{i=1}^{N} I(x,y)\,(y-\bar{y})^1 = 0$

Using central-based moments, we can compute a number of useful geometric features for a shape

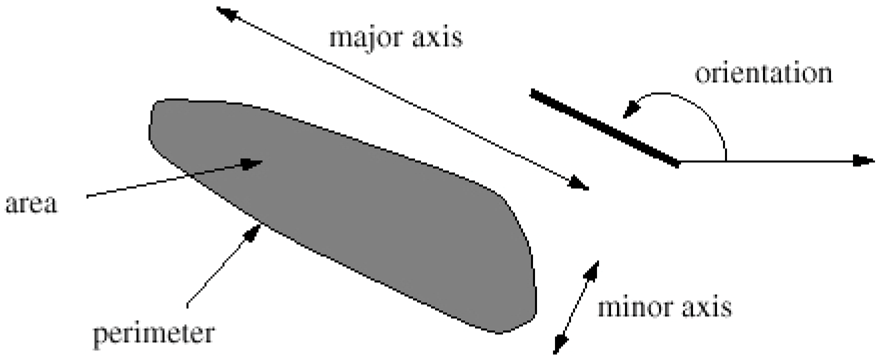
- Orientation
- Eccentricity
- Invariant (Hu’s) moments



Object orientation – the angle of the major axis – is computed from the central moments

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\mu_{11}(R)}{\mu_{20}(R) - \mu_{02}(R)}\right)$$

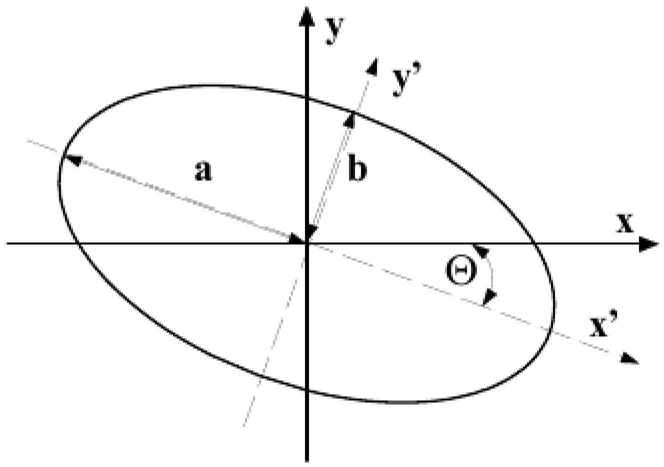
- If there is no major axis (if the object is round, for example), the calculation is not possible (the denominator is zero)



The eccentricity ε of an object is related to its aspect ratio; a circle has $\varepsilon = 1$, while other, longer objects have $\varepsilon > 1$

$$\varepsilon(R) = \frac{\mu_{20} + \mu_{02} + \sqrt{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}}{\mu_{20} + \mu_{02} - \sqrt{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}}$$

- Eccentricity is not dependent on the location, orientation or size of an object



There is a full set of object moments that are independent of location, scale and rotation, called *Hu's moments* – they don't all have easy interpretation

- Expressed in terms of central moments, they are:

$$H_1 = \mu_{20} + \mu_{02}$$

$$H_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2$$

$$H_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$$

$$H_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$

$$H_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$

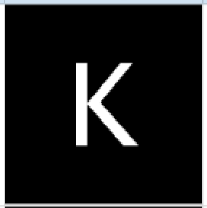
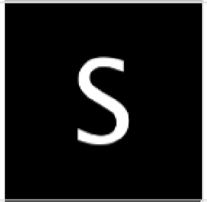
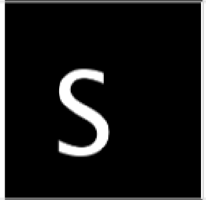
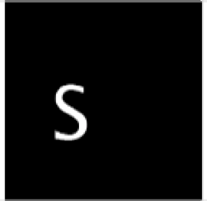
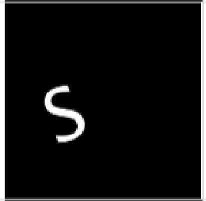
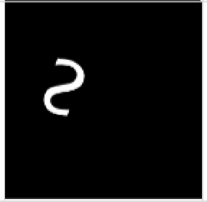
$$H_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$$

$$H_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{12} - \mu_{30})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$

There is a full set of object moments that are independent of location, scale and rotation, called *Hu's moments* – they don't all have easy interpretation

- Expressed in terms of central moments, they are:

$$H_1 = \mu_{20} + \mu_{02}$$
$$H_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2$$
$$H_3 = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$$
$$H_4 = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$$
$$H_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{21} - \mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$
$$H_6 = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$$
$$H_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] + (3\mu_{12} - \mu_{30})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$$

id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

Editor - C:\Users\crjones4\.spyder-py3\ExpBlobs.py

ECE5554 FA19 HW2 part 1.py | ECE5554 FA19 HW2.py | ExpBlobs.py | ExpMeanShift.py

```python
# -*- coding: utf-8 -*-
"""
ExpBlobs.py
Created on Thu Sept 26 15:17:05 2019
@author: crjones4
"""

import numpy as np
import cv2

def showImage(img, name):
    cv2.imshow(name,img)
    return

##############################

def imshow_components(labels):
    # 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])

    # cvt to BGR for display
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

    # set bg label to black
    labeled_img[label_hue==0] = 0

    cv2.imshow('labeled.png', labeled_img)
    cv2.waitKey()

##############################


img = cv2.imread('C:\\Data\\airplanes.png', cv2.IMREAD_GRAYSCALE)
showImage(img, "Original")
thr, binary = cv2.threshold(255-img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  # ens
ret, labels = cv2.connectedComponents(binary)
for idx in range(5):
    area = (labels == idx).sum()
    print("Blob #" + str(idx) + ", Area=" + str(area))

contours, hierarchy = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPL
for cnt in contours:
    ROI = cv2.boundingRect(cnt)
    print(ROI)
    momentArray = cv2.moments(cnt)
    Hu = cv2.HuMoments(momentArray)
    print(Hu)

imshow_components(labels)
```

Original

Variable explorer

Name	Type	Size	Value

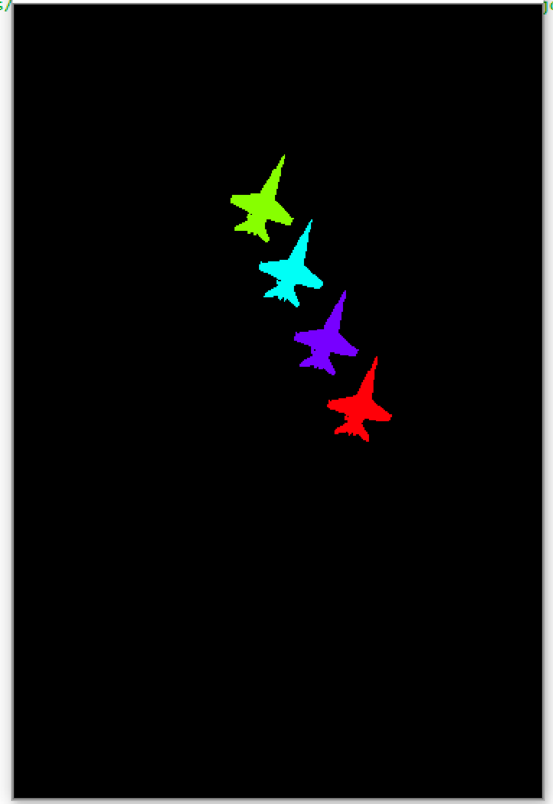
Variable explorer | File explorer | Help

IPython console

Console 4/A

```
In [1]: runfile('C:/Users/                                    jones4
Blob #0, Area=197669
Blob #1, Area=939
Blob #2, Area=915
Blob #3, Area=896
Blob #4, Area=881
(217, 244, 45, 59)
[[ 2.53192404e-01]
 [ 5.33760002e-03]
 [ 3.85005219e-03]
 [ 9.75039624e-04]
 [ 1.88544431e-06]
 [ 6.95762823e-05]
 [-1.18286931e-07]]
(194, 198, 45, 59)
[[ 2.54069415e-01]
 [ 6.15341443e-03]
 [ 4.08207717e-03]
 [ 1.03783867e-03]
 [ 2.13030888e-06]
 [ 8.04644344e-05]
 [-1.58105140e-07]]
(170, 149, 44, 61)
[[ 2.55171039e-01]
 [ 5.71926018e-03]
 [ 4.14525073e-03]
 [ 1.07970638e-03]
 [ 2.27168637e-06]
 [ 8.13311966e-05]
 [-2.38757742e-07]]
(150, 104, 44, 61)
[[ 2.52388681e-01]
 [ 5.94140267e-03]
 [ 4.09932003e-03]
 [ 1.13595974e-03]
 [ 2.45095541e-06]
 [ 8.74091474e-05]
 [-4.23791031e-08]]
```

labeled.png



Hu's moments can be calculated for gray-scale objects or regions in the image as well as binary

	H_1	H_2	H_3	H_4	H_5	H_6	H_7
Original image	0.00178421388598880	3.15834695516106e-07	6.44951125465700e-10	4.44289716670280e-11	-3.81454242506225e-21	1.4477162212496 6e-14	6.48161819262287e-21
Half Sized	0.0017852753728 2729	3.21930915835298e-07	6.48490784713280e-10	4.46081541961950e-11	-3.73352023893018e-21	1.4915591954299 4e-14	6.60486517698611e-21
Rotated (45°)	0.00178641593439320	3.10399124609950e-07	6.4461803271969 0e-10	4.48944767438045e-11	-3.87613252119480e-21	1.45895498993993e-14	6.58058875363834e-21

There are other sets of (partially) invariant object moments – for example, the Zernike moments

Rotational Invariance

The magnitude of each Zernike moment is invariant under rotation.

Coefficient of variance income example by chocobar vs airplane

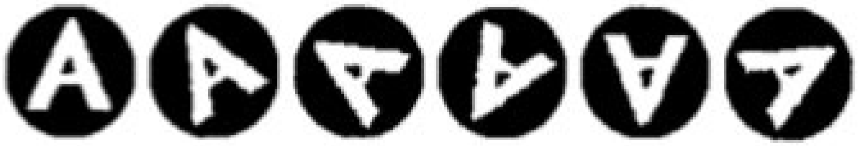


Fig. 2. The image of character A and five rotated versions of it. From left to right rotation angles are: 0°, 30°, 60°, 150°, 180°, and 300°.

TABLE II
MAGNITUDES OF SOME OF THE ZERNIKE MOMENTS FOR ROTATED IMAGES SHOWN IN FIG. 2 AND THEIR CORRESPONDING STATISTICS

	$\lvert A_{20} \rvert$	$\lvert A_{22} \rvert$	$\lvert A_{31} \rvert$	$\lvert A_{33} \rvert$
0°	439.62	41.79	57.97	172.57
30°	436.70	40.20	63.82	171.96
60°	440.63	40.08	66.28	169.41
150°	438.53	41.55	65.47	170.83
180°	439.01	46.85	62.39	168.47
300°	438.43	39.19	65.77	170.84
μ	438.82	41.61	63.62	170.68
σ	1.32	2.74	3.12	1.53
$\sigma/\mu\%$	0.30	6.57	4.90	0.90

Zernike polynomials are a set of orthogonal functions of x and y, defined on the domain $\sqrt{x^2 + y^2} \leq 1$

$$Z_{nm}(\rho, \theta) = R_{nm}(\rho) e^{jm\theta}, \text{ where}$$

$$R_{nm}(\rho) = \begin{cases} \displaystyle\sum_{i=0}^{(n-m)/2} \frac{(-1)^i (n-i)!}{i! \left[\frac{1}{2}(n+m) - i\right]! \left[\frac{1}{2}(n-m) - i\right]!} \rho^{n-2i} & \text{for } n - m \text{ even} \\ 0 & \text{for } n - m \text{ odd} \end{cases}$$

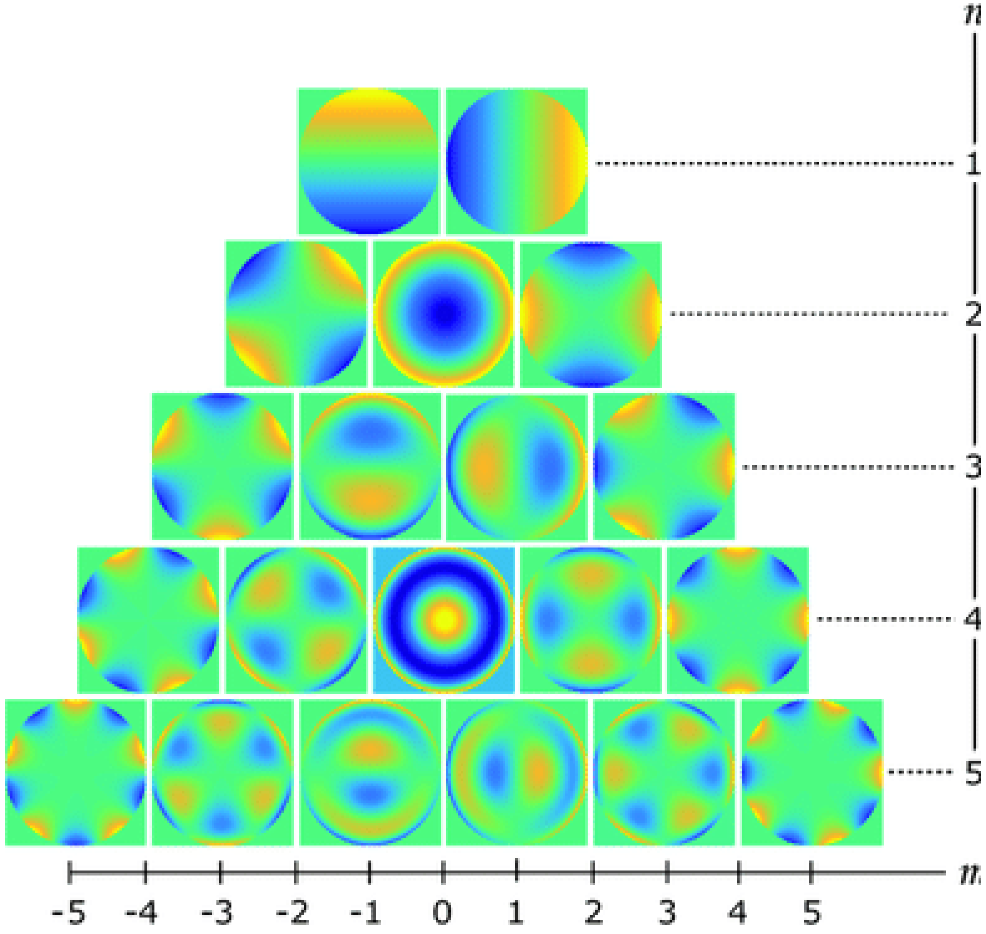
- The Zernike moments are then just projections of an object's intensity surface onto these basis functions

$$A_{mn} = \frac{m+1}{\pi} \int_x \int_y f(x, y) [Z_{nm}(x, y)]^* dx\, dy$$

This family of polynomials can be used to efficiently represent a 2D function; the Zernike moments of a centered object are also used for compact object description

- Zernike moments are naturally rotation independent
- If the object is centered, naturally we have translation independence
- The Zernike moments are not scale independent

- Zernike polynomials are also used to describe optical phenomena such as output of lasers and laser diodes



Today's Objectives

Labeling Regions by Sequential Labeling ("Blobs")
- Connected components
- blob statistics

Use of region descriptions

Chain code
- Differential chain code and shape number

Geometric features
- Compactness and Circularity

Statistical Shape Features
- Centroid
- Moments and Central Moments

Moment-based Features
- Orientation and Eccentricity
- Invariant moments - Hu's and Zernike moments