Applied Computer Vision

David Vernon
Carnegie Mellon University Africa

vernon@cmu.edu
www.vernon.eu

Lecture 7

Segmentation

Region-based approaches, binary thresholding, connected component analysis

Segmentation

- Partitioning the image into its constituent parts

- Constituent parts depend on the task
  - Detect object, object class, foreground/background

Segmentation

- Partitioning the image into its constituent parts

- Constituent parts depend on the task
  - Detect object, object class, foreground/background

Segmentation

A grouping process:

- the components of a group are similar with respect to some feature or set of features
- This grouping should identify regions in the image which correspond to unique and distinct objects

Segmentation

Two complementary approaches:

1. Region Growing
   - Grouping elemental areas (in simple cases, individual image pixels)
   - That share a common feature
   - Into connected two-dimensional areas called regions
   - e.g. pixel grey-level, hue, or some textural pattern

2. Boundary Detection
   - Detecting or enhancing the boundary pixels of objects within the image
   - Edge detection … discontinuities is some feature between regions
   - Typical feature: image intensity

Segmentation

The usual approach to segmentation by boundary detection is to:

- Construct an edge image from the original grey-scale image
- Use this edge to construct the boundary image without reference to the original grey-scale data by edge linking to generate short curve segments

Segmentation

Boundary detection algorithms

- Use domain-dependent information or knowledge which they incorporate in associating or linking the edges

    - edge-thinning
    - gap-filling
    - curve segment linking

- Their effectiveness is dependent on the quality of the edge image

Binary Thresholding

Binary Thresholding

- Intensity or colour thresholding is a simple region based segmentation technique

- Works well where

  - an object exhibits a uniform grey-level or colour

  - and rests against a background of a different grey-level or colour

Binary Thresholding

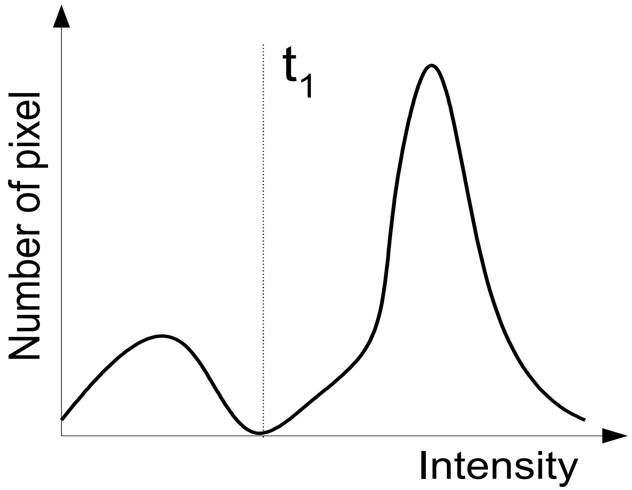
If $g(x, y)$ is a thresholded version of $f(x, y)$ for some global threshold T

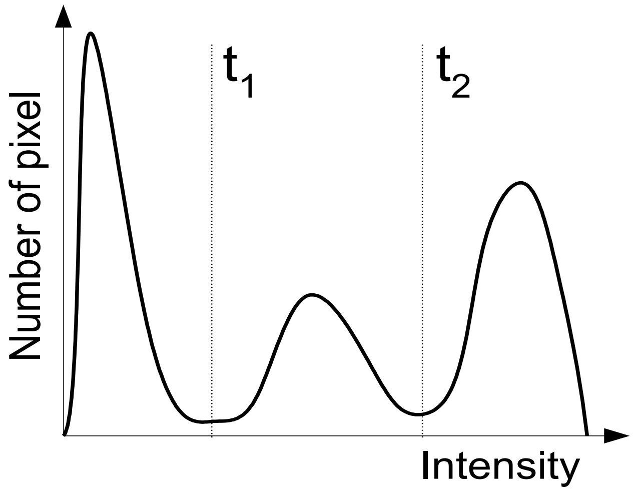
255

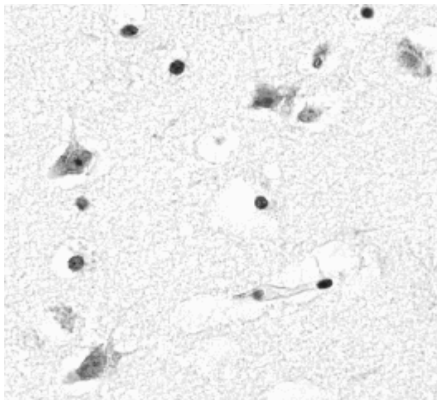
$$
\begin{aligned}
g(x, y) &= 1 \quad \text{if } f(x, y) \geq T \\
&= 0 \quad \text{otherwise}
\end{aligned}
$$

Binary Thresholding

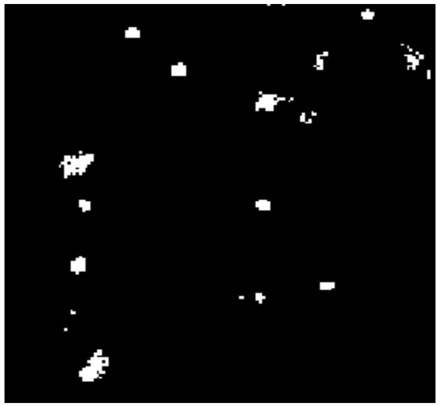
Histogram



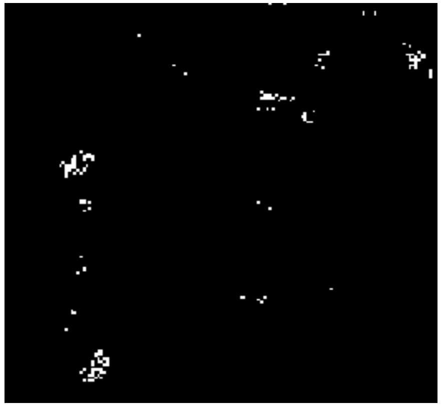




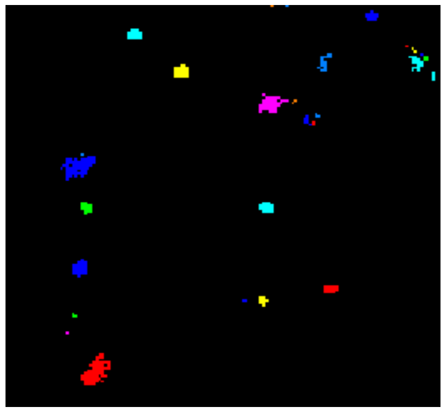
Original image



t_1=150



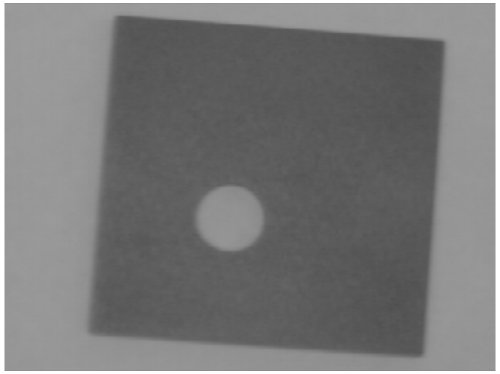
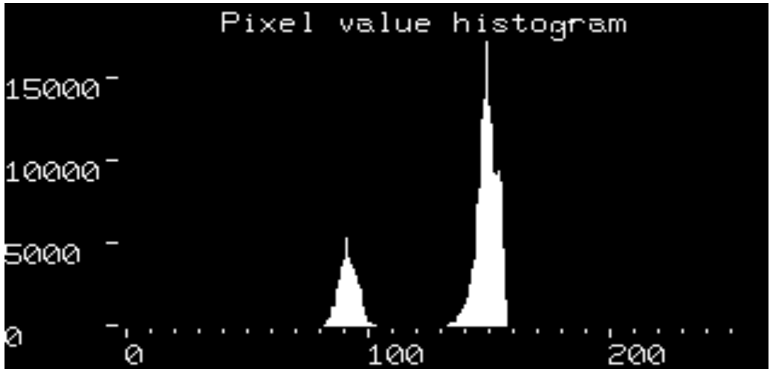
t_1=130, t_2=150

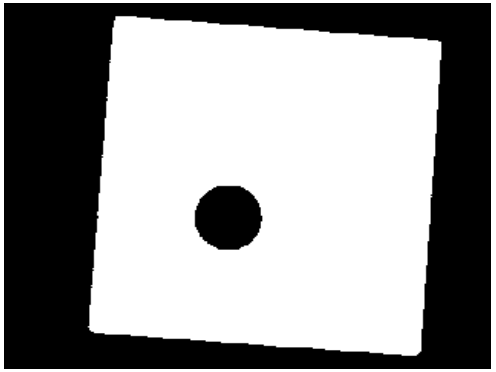


Regions (blobs)

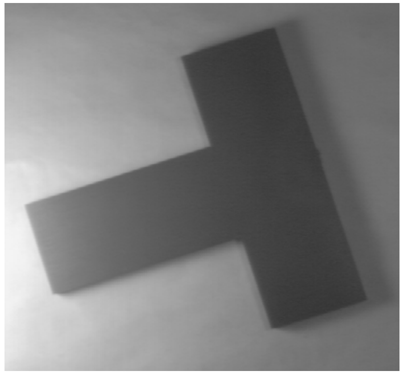
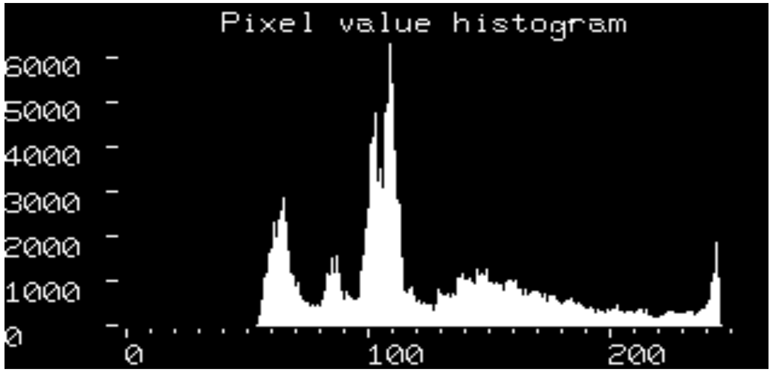
Credit: Markus Vincze, Technische Universität Wien

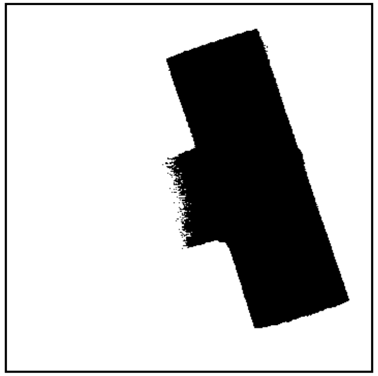
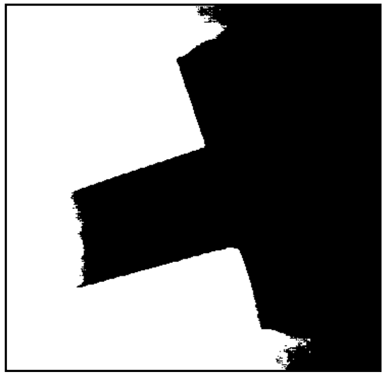
Binary Thresholding



Original image Histogram Binary image: t_1=120

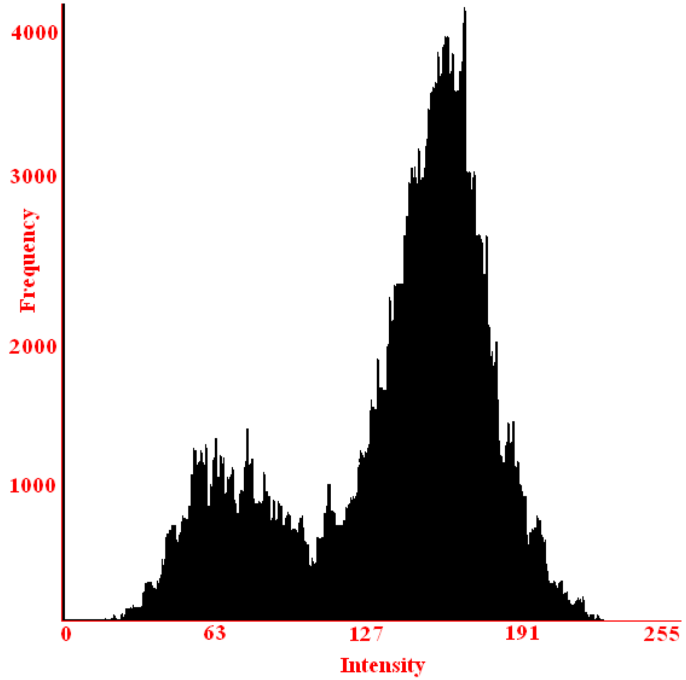



Original image Histogram Binary image for t_1=80 and t_1=120

Credit: Markus Vincze, Technische Universität Wien

Binary Thresholding

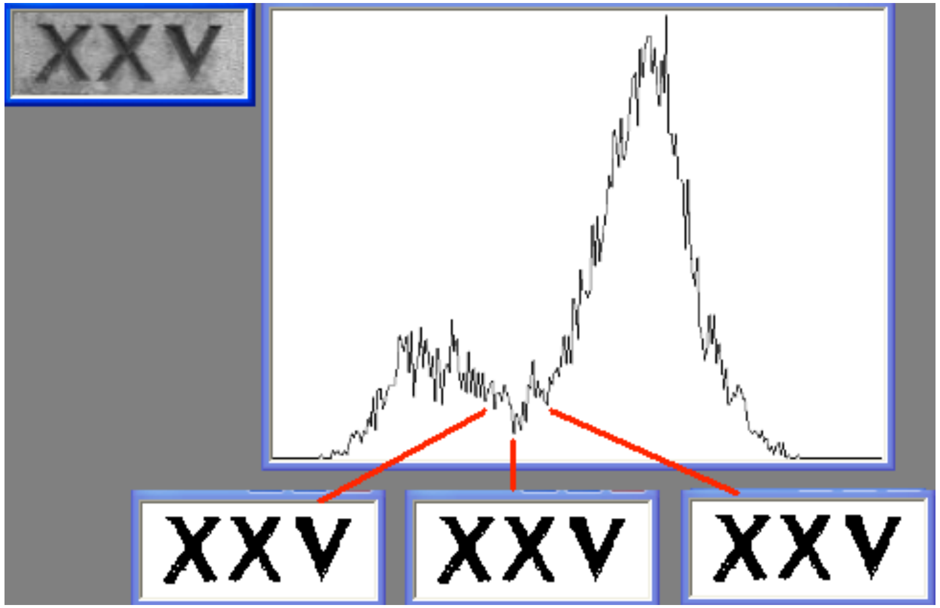


Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

Global, Local, and Dynamic Thresholding

A threshold operation may be viewed as a test T involving

- some function of the grey-level at a point
- some local property of the point
- the position of the point in the image

$$T(x, y, N(x, y), f(x, y))$$

- $f(x, y)$ is the grey-level at the point (x, y)
- $N(x, y)$ denotes some local property of the point (x, y)
- If $f(x, y) > T(x, y, N(x, y), f(x, y))$ label (x, y) object else label it background

Binary Thresholding

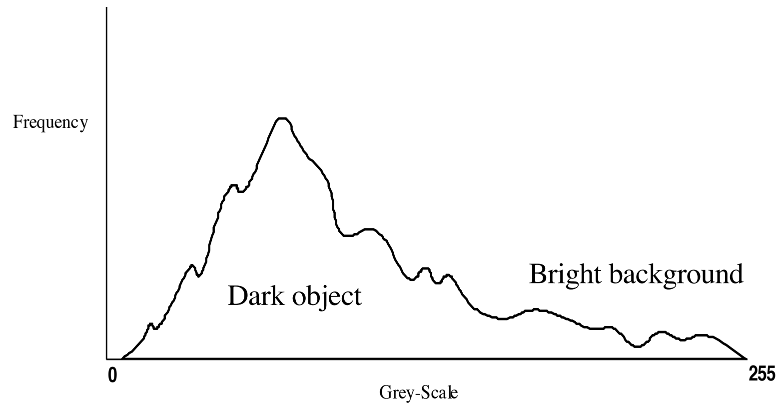
Global, Local, and Dynamic Thresholding

- $T = T(f(x, y))$ Global thresholding
  - The test is dependent only on the image value at that point
- $T = T(N(x, y), f(x, y))$ Local thresholding
  - The test is dependent on a neighbourhood property of the point and the image value at that point
- $T = T(x, y, N(x, y), f(x, y))$ Dynamic thresholding
  - The test is dependent on the coordinates of the point, a neighbourhood property of the point, and the image value at that point

Binary Thresholding

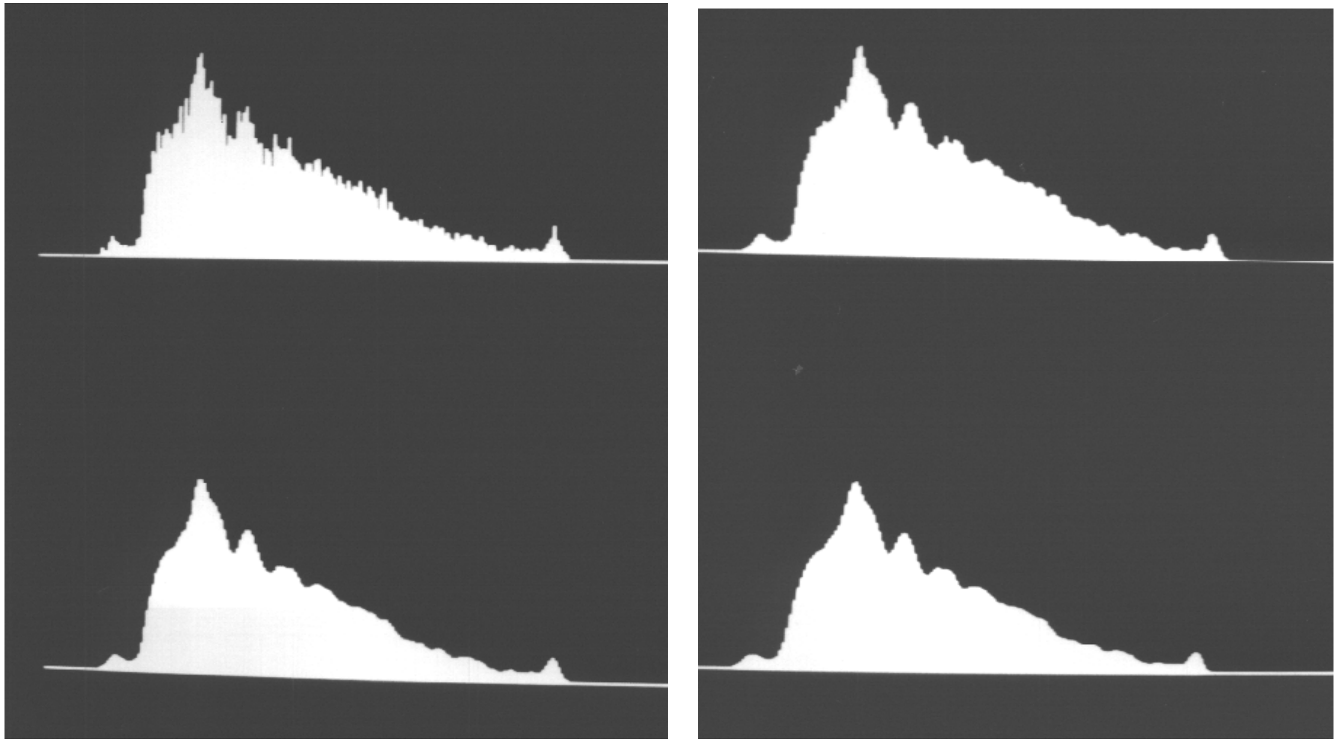
Threshold Selection

- Most techniques are based on histogram analysis
  - select thresholds which lie in the region between the modes
- Assumption of bi-modal histogram may not be valid
  - Histograms are noisy
  - Histograms may be uni-modal
  - Histogram smoothing is often required



Binary Thresholding

Histogram smoothing



Top-left: no smoothing; top-right: one application of a 3x1 neighbourhood average operator;
Bottom-left: two applications; bottom-right: three applications

Binary Thresholding

Threshold Selection

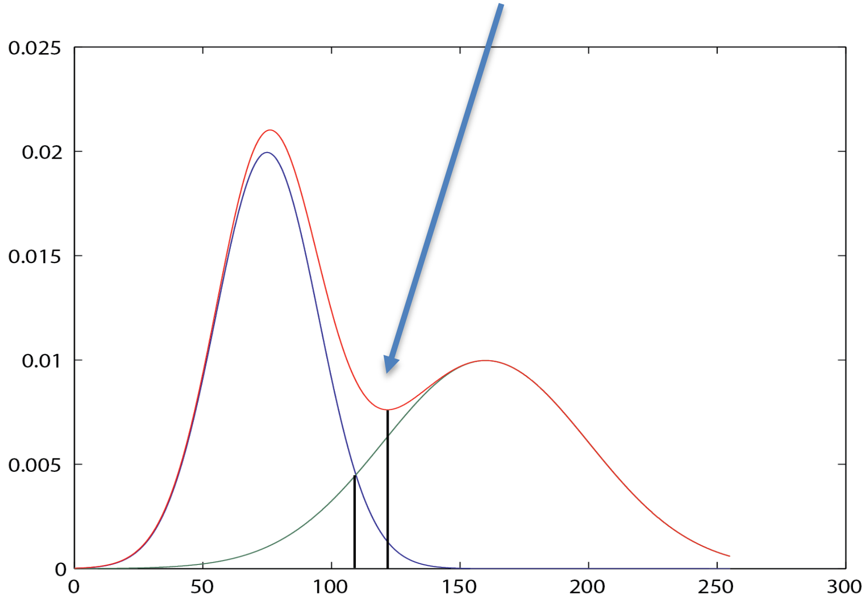
- Use the average image value of those pixels which are on the boundary between the object and the background as an estimate of the threshold value
  - Use an edge detector (e.g. Marr-Hildreth Laplacian of Gaussian operator or Canny operator) to locate edges in the image
  - Compute the mean grey-level of the image pixels at these edge locations is computed
  - This mean represents the global threshold value

Binary Thresholding

Threshold Selection

- Model as two normal distributions
- What if they overlap?

The position of minimum overlap
(i.e. the position where the misclassified
areas of the distributions are equal)
is not necessarily where the valley occurs



Binary Thresholding

Threshold Selection

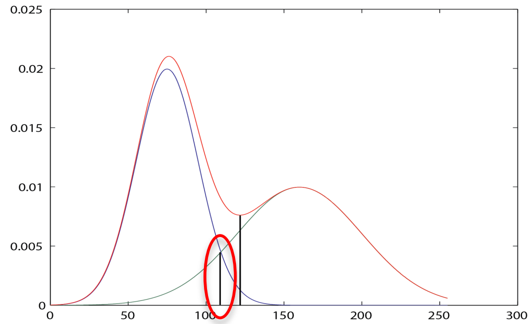
For the techniques which follow:

- Image $f(i, j)$
- Histogram $h(g)$
- Probability Distribution $p(g) = h(g) \,/\, \Sigma_g\, h(g)$

Binary Thresholding

Threshold Selection

Clustering – Variation of K-Means



Minimize the error of classifying a background pixel as a foreground pixel and vice versa

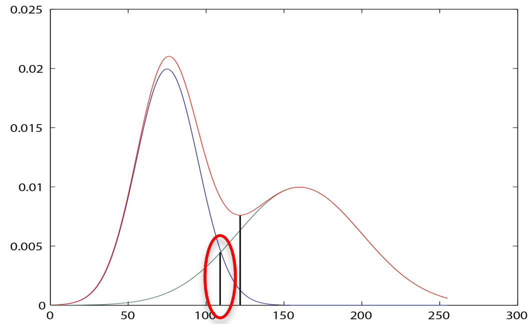
To do this we try to minimize the area under the histogram for one region that lies on the other region's side of the threshold

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

Binary Thresholding

Threshold Selection

Clustering – Variation of K-Means



Let $\mu_b(T)$ be the mean of all pixels less than the threshold

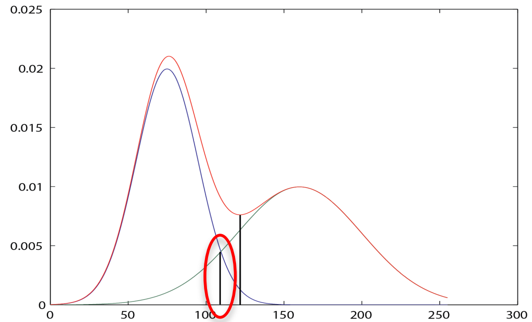
Let $\mu_f(T)$ be the mean of all pixels greater than the threshold

We want to find a threshold such that the following holds:

$$|f(i,j) - \mu_b(T)| > |f(i,j) - \mu_f(T)| \qquad \text{for all } f(i,j) \geq T$$
$$|f(i,j) - \mu_b(T)| < |f(i,j) - \mu_f(T)| \qquad \text{for all } f(i,j) < T$$

Binary Thresholding

Threshold Selection



1. Set T^0 = <some initial value>, $t = 0$

2. Compute μ^t_B and μ^t_O using T^t

$$w_b(T^t) = \sum_{g=0}^{T^t-1} p(g)$$

$$w_f(T^t) = \sum_{g=T^t}^{255} p(g) = 1 - w_b(T^t)$$

$$\mu_b(T^t) = \sum_{g=0}^{T^t-1} p(g).g \Big/ w_b(T^t)$$

$$\mu_f(T^t) = \sum_{g=T^t}^{255} p(g).g \Big/ w_f(T^t)$$

3. Update the threshold:

   Set $T^{t+1} = (\mu^t_b + \mu^t_f) / 2$

   Increment t

4. Go back to 2 until:

   $T^{t+1} = T^t$

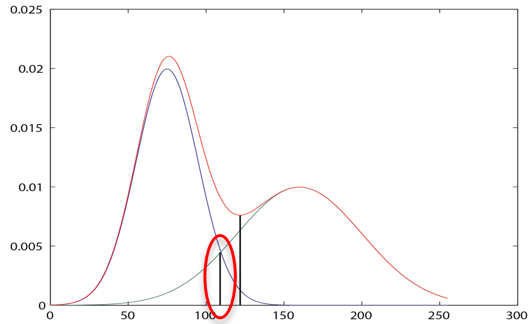
"There is no intuitive way to explain why it returns the optimal answer"
K. Dawson-Howe, 2014

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

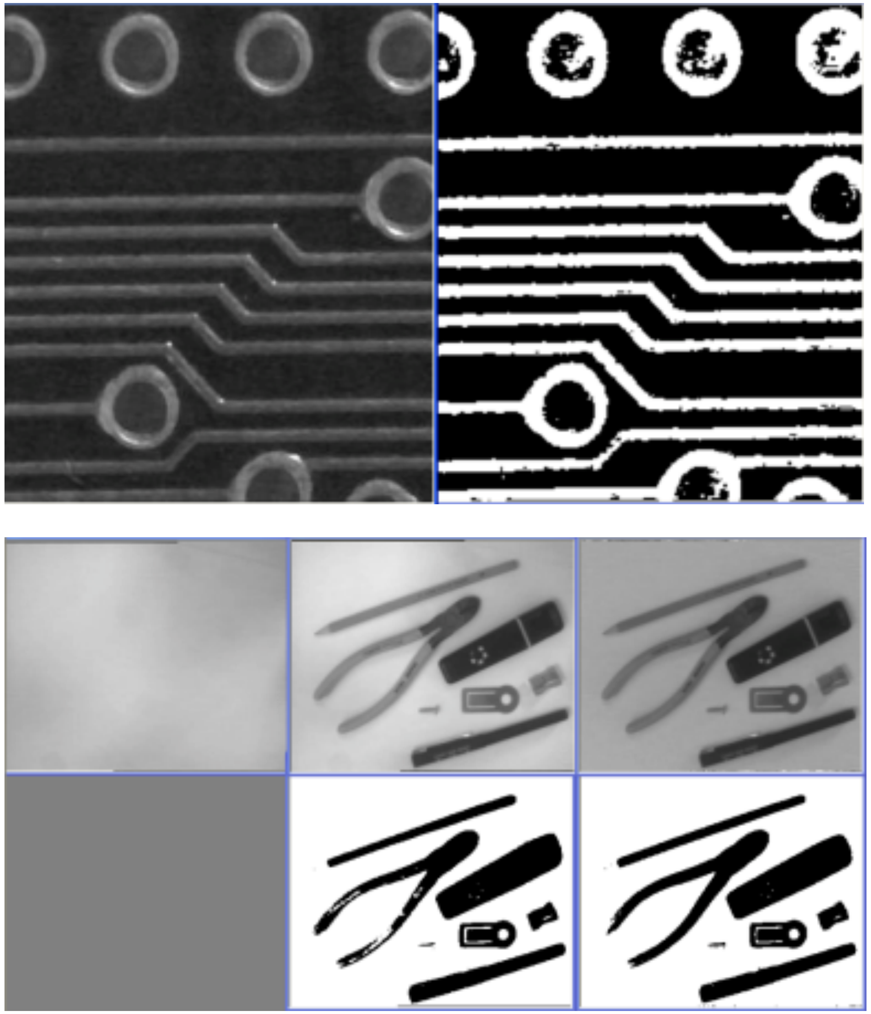
Threshold Selection

Clustering – Variation of K-Means



Works well if the variances of the distributions are approximately equal

Binary Thresholding



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

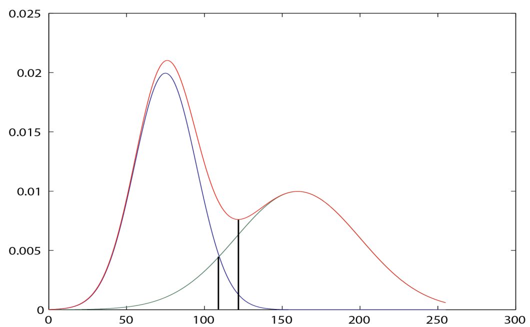
Threshold Selection – Otsu Algorithm

- What if the histogram not a mixture of two normal distributions?

- Use a technique that finds the threshold separating the two classes (background and foreground) so that

  – their combined spread (intra-class variance) is minimal,

  – or, equivalently, so that their inter-class variance is maximal



N. Otsu, "A threshold selection method from gray-level histograms", IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66, 1979.

Binary Thresholding

Threshold Selection – Otsu Algorithm

- Minimize the spread of the pixels … compute the threshold that achieves
  - Smallest within-class (intra-class) variance $\sigma_W^2(T) = w_f(T)\sigma_f^2(T) + w_b(T)\sigma_b^2(T)$

$$w_f(T) = \sum_{g=T}^{255} p(g) \qquad \sigma_f^2(T) = {\sum_{g=T}^{255} p(g).\left(g - \mu_f(T)\right)^2} \Big/ {w_f(T)}$$

$$w_b(T) = \sum_{g=0}^{T-1} p(g) \qquad \sigma_b^2(T) = {\sum_{g=0}^{T-1} p(g).\left(g - \mu_b(T)\right)^2} \Big/ {w_b(T)}$$

within-class

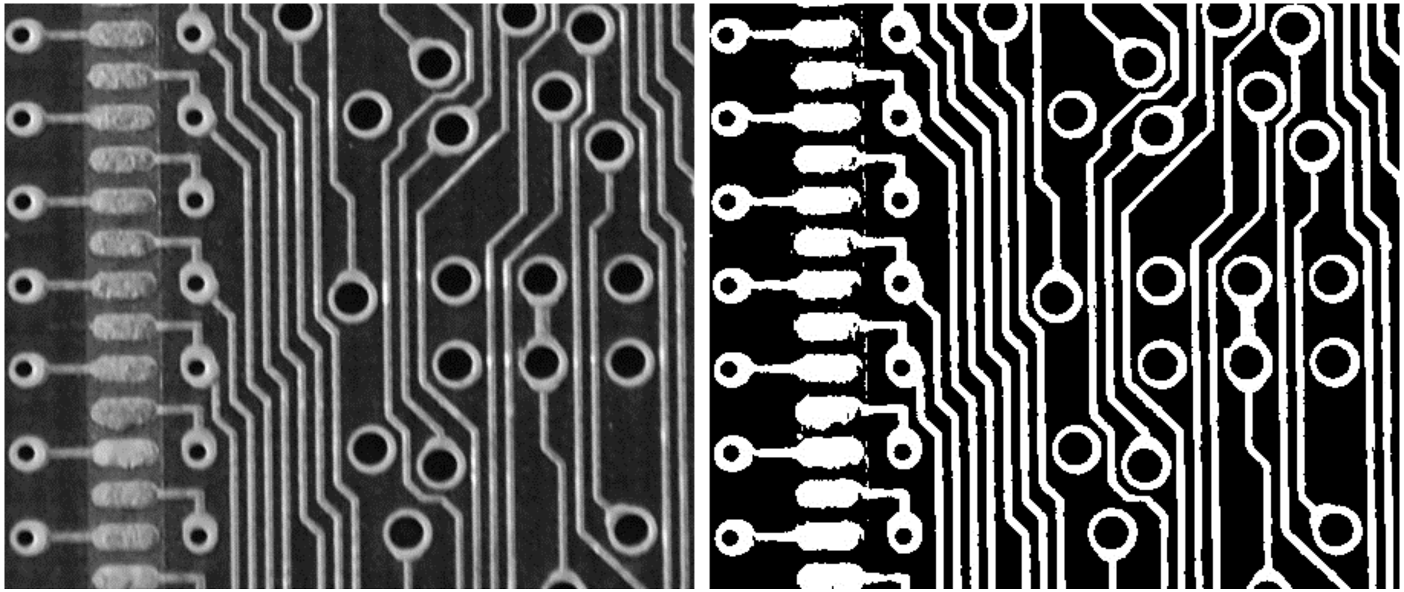
$$\mu_f(T) = {\sum_{g=T}^{255} p(g).g} \Big/ {w_f(T)} \qquad \mu_b(T) = {\sum_{g=0}^{T-1} p(g).g} \Big/ {w_b(T)}$$

  - Largest between-class (inter-class) variance $\sigma_B^2(T) = w_f(T)w_b(T)\left(\mu_f(T) - \mu_b(T)\right)^2$

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

Threshold Selection – Otsu Algorithm



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

Threshold Selection – Adaptive Algorithm

- Divide the image into sub-images
- Compute thresholds for all sub-images
- Interpolate thresholds for every point using bilinear interpolation

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Connected Component Analysis

Connected Component Analysis

Adjacency conventions

- This problem is one of defining exactly which are the neighbours of a given pixel
- Consider the 3*3 neighbourhood in an image where the pixels are labelled 0 through 8

3	2	1
4	8	0
5	6	7

Which pixels does pixel 8 touch?

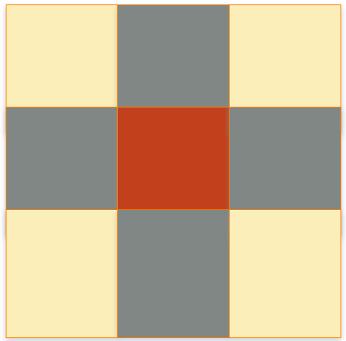
Connected Component Analysis

Adjacency conventions

- A pixel p at coordinates (i, j) has four horizontal and vertical neighbors at coordinates
  $(i\text{-}1, j)$ $(i\text{+}1, \mathrm{j})$ $(i, j\text{-}1)$ $(i, j\text{+}1)$

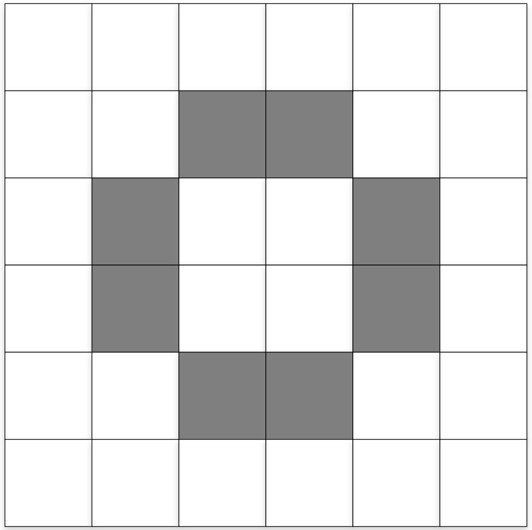
- This set is called 4-neighborhood $N_4(p)$

- The pixel also has four diagonal neighbors:
  $(i\text{-}1, j\text{-}1)$ $(i\text{+}1, j\text{-}1)$ $(i\text{+}1, j\ \text{-}1)$ $(i\text{+}1, j\text{+}1)$



- The 8 points together form a 8-neighborhood $N_8(p)$

Connected Component Analysis

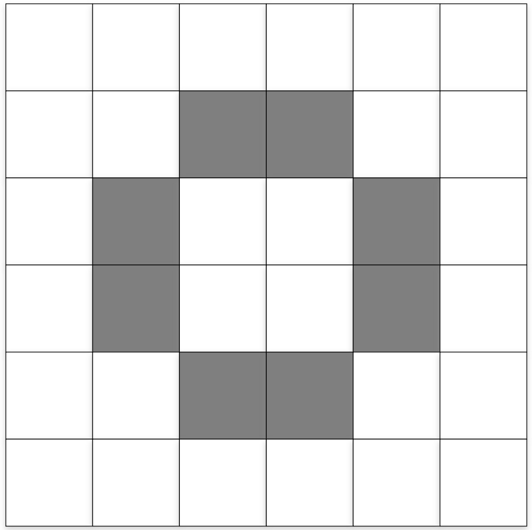


If figure and ground are both 8-connected it means the hole in the 'ring' is connected to the region surrounding the 'ring'

Connected Component Analysis

It is normal practice to use both conventions

- one for an object
- one for the background on which it rests



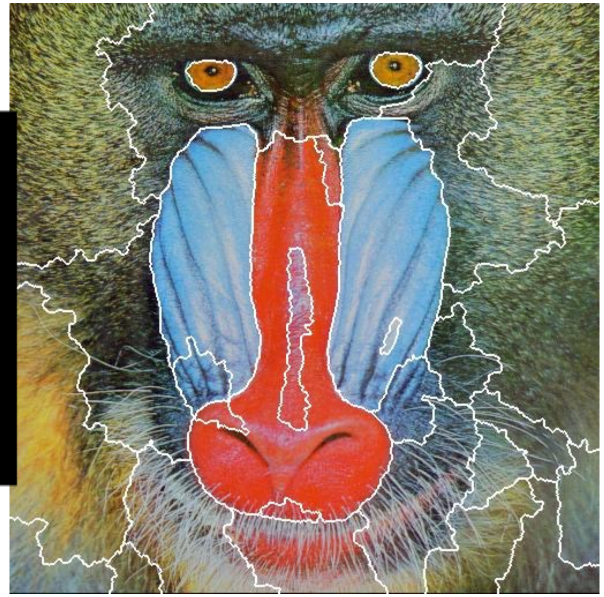
This can be extended quite generally

- adjacency conventions are applied alternatively to image regions which are recursively nested (or embedded) within other regions as one goes from level to level in the nesting

Connected Component Analysis

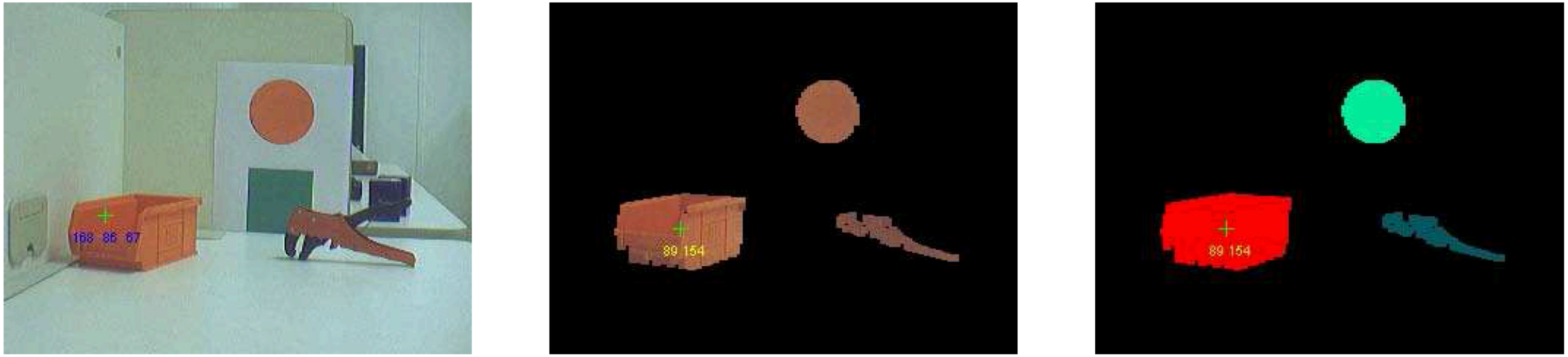
Connected components

- groups of connected pixels with common properties
- the properties could be a similar color, texture, or motion pattern, …



Credit: Francesca Odone, University of Genova

Connected Component Analysis



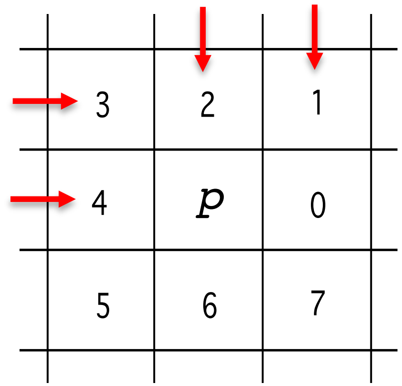
Connected Component Analysis

```
Assume a binary input images and 8-connectivity

scan the image, row by row examining pixels p

    if pixel p is a foreground pixel
       examine the four neighbours of p already encountered in the scan

       if all four neighbours are 0
          assign a new label to p
       else
          if only one neighbour is a foreground pixel
             assign its label to p
          else
             assign one of the labels to p &
             make a note of the label equivalences
```



Connected Component Analysis

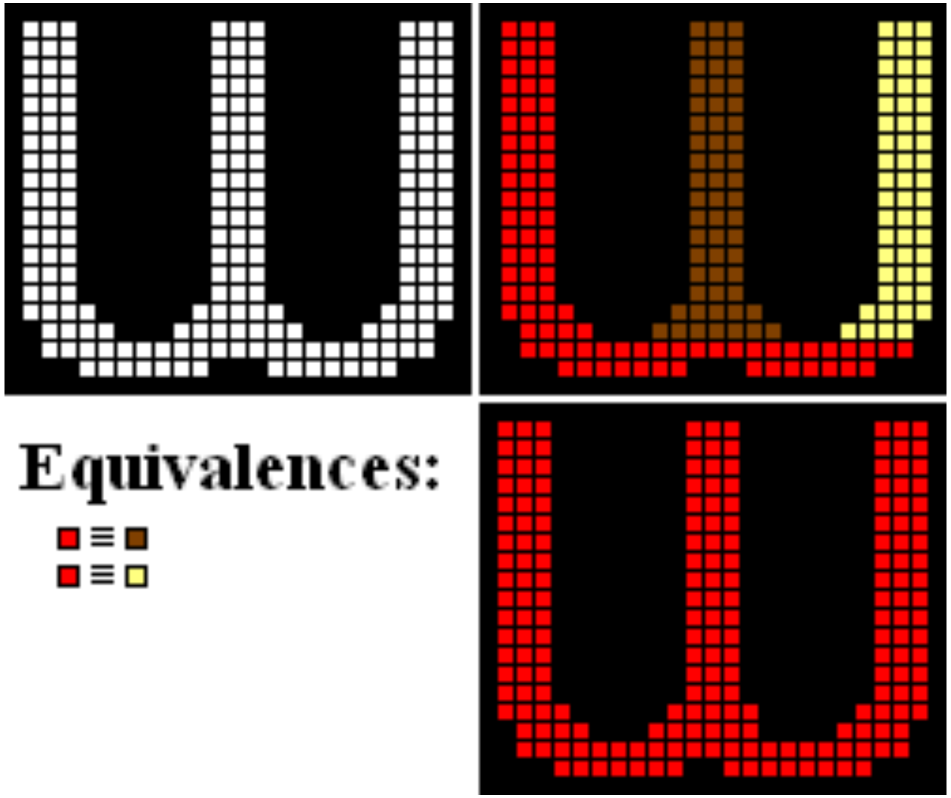
```
sort the equivalent label pairs into equivalence classes

assign a unique label to each class

scan through the image
   replace each label with the label assigned to its equivalence class

for display, encode the labels as a distinct grey-levels or colour
```

Connected Component Analysis



Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Connected Component Analysis

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Demos

The following code is taken from the binaryThresholding project in the lectures directory of the ACV repository

See:

```
binaryThresholding.h
binaryThresholdingImplementation.cpp
binaryThresholdingApplication.cpp
```

```
void binaryThresholding(int, void*) {

   extern Mat inputImage;
   extern int thresholdValue;
   extern char* thresholded_window_name;
   Mat greyscaleImage;
   Mat thresholdedImage;
   int row, col;

   if (thresholdValue < 1)  // the trackbar has a lower value of 0 which is invalid
      thresholdValue = 1;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }

   thresholdedImage.create(greyscaleImage.size(), CV_8UC1);

   for (row=0; row < greyscaleImage.rows; row++) {
      for (col=0; col < greyscaleImage.cols; col++) {
         if(greyscaleImage.at<uchar>(row,col) < thresholdValue) {
            thresholdedImage.at<uchar>(row,col) = (uchar) 0;
         }
         else {
            thresholdedImage.at<uchar>(row,col) = (uchar) 255;
         }
      }
   }

   /* alternatively, use OpenCV */

   // threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY);
   // threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY  | THRESH_OTSU); // automatic threshold selection

   imshow(thresholded_window_name, thresholdedImage);
}
```

Demos

The following code is taken from the binaryThresholdingAdaptive project in the lectures directory of the ACV repository

See:

```
binaryThresholdingAdaptive.h
binaryThresholdingAdaptiveImplementation.cpp
binaryThresholdingAdaptiveApplication.cpp
```

```
/*
 * function binaryThresholding
 * Trackbar callback - block size user input
*/

void binaryThresholdingAdaptive(int, void*) {

   extern Mat inputImage;
   extern int blockSizeValue;
   extern char* thresholded_window_name;
   Mat greyscaleImage;
   Mat thresholdedImage;

   if (blockSizeValue < 1)  // the trackbar has a lower value of 0 which is invalid
      blockSizeValue = 2;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }

   thresholdedImage.create(greyscaleImage.size(), CV_8UC1);

   blockSizeValue = 2*(blockSizeValue/2)+1; // blocksize has to be odd

   adaptiveThreshold(greyscaleImage,thresholdedImage,255.0,ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, blockSizeValue, 0 );

   imshow(thresholded_window_name, thresholdedImage);
}
```

Demos

The following code is taken from the binaryThresholdingOtsu project in the lectures directory of the ACV repository

See:

```
binaryThresholdingOtsu.h
binaryThresholdingOtsuImplementation.cpp
binaryThresholdingOtsuApplication.cpp
```

```
void binaryThresholdingOtsu(char *filename) {

   Mat inputImage;
   Mat greyscaleImage;
   Mat thresholdedImage;

   int thresholdValue            = 128; // default threshold

   char* input_window_name       = "Input Image";
   char* thresholded_window_name = "Thresholded Image";

   inputImage = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
   if (inputImage.empty()) {
      cout << "can not open " << filename << endl;
      prompt_and_exit(-1);
   }

   printf("Press any key to continue ...\n");

   // Create a window for input and display it
   namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
   imshow(input_window_name, inputImage);

   // Create a window for thresholded image
   namedWindow(thresholded_window_name, CV_WINDOW_AUTOSIZE );

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }
```

```
    //thresholdedImage.create(greyscaleImage.size(), CV_8UC1);

    threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY  | THRESH_OTSU); // automatic threshold selection

    imshow(thresholded_window_name, thresholdedImage);

    do {
       waitKey(30);                                 // Must call this to allow openCV to display the images
    } while (!_kbhit());                            // We call it repeatedly to allow the user to move the windows
                                                   // (if we don't the window process hangs when you try to click and drag

    getchar(); // flush the buffer from the keyboard hit

    destroyWindow(input_window_name);
    destroyWindow(thresholded_window_name);
}
```

Demos

The following code is taken from the connectedComponents project in the lectures directory of the ACV repository

See:

```
connectedComponents.h
connectedComponentsImplementation.cpp
connectedComponentsApplication.cpp
```

```
/*
 * function connectedComponents
 * Trackbar callback - threshold user input
*/

void connectedComponents(int, void*) {

   extern Mat inputImage;
   extern int thresholdValue;
   extern char* thresholded_window_name;
   extern char* components_window_name;

   Mat greyscaleImage;
   Mat thresholdedImage;

   vector<vector<Point>> contours;
   vector<Vec4i> hierarchy;

   if (thresholdValue < 1)  // the trackbar has a lower value of 0 which is invalid
      thresholdValue = 1;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }

   threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY);

   imshow(thresholded_window_name, thresholdedImage);

   findContours(thresholdedImage,contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_NONE);
   Mat contours_image = Mat::zeros(inputImage.size(), CV_8UC3);
   for (int contour_number=0; (contour_number<(int)contours.size()); contour_number++)
   {
        Scalar colour( rand()&0xFF, rand()&0xFF, rand()&0xFF );
        drawContours( contours_image, contours, contour_number, colour, CV_FILLED, 8, hierarchy );
   }

   imshow(components_window_name, contours_image);

}
```

CV_RETR_EXTERNAL
retrieves only the extreme outer contours

Demos

Connected component analysis

Also see standalone application:

```
connectedComponentsApp.exe
```

Exercises

- Read OpenCV documentation for all OpenCV functions in sample code

  e.g. findContours

  https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours

  (also see https://docs.opencv.org/trunk/d9/d8b/tutorial_py_contours_hierarchy.html)

- Study utility functions in sample code

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 3.3 More neighborhood operations

Section 3.3.4 Connected components