Computer Vision for Robotics

Thanks to Brad Nelson
For use of the slides

Image Processing Algorithm

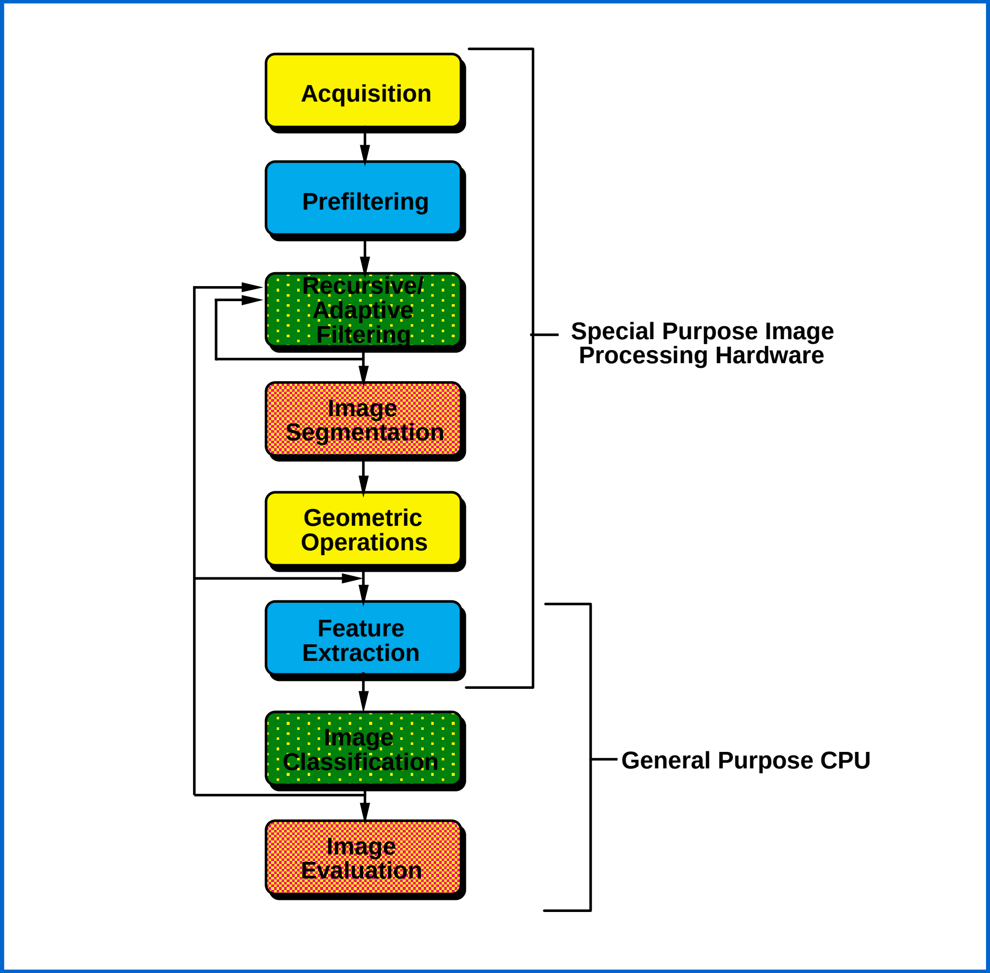
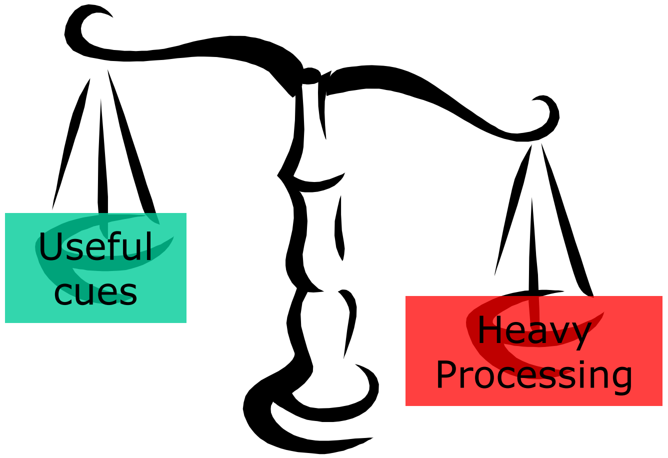


Image Intensities & Data reduction

- Monochrome image ⇨ matrix of intensity values
- Typical sizes:
  - 320 x 240 (QVGA)
  - 640 x 480 (VGA)
  - 1280 x 720 (HD)
- Intensities sampled to 256 grey levels ⇨ 8 bits
- Images capture a lot of information



Color Images: 3 channels (e.g. RGB)

8 bits per channel = 24 bits total

⇨Reduce the amount of input data:
preserving useful info & discarding redundant info

Commonly Used Algorithms

- Statistical Operations
- Segmentation and Edge Detection
- Finding Shapes
- Frequency Domain Techniques
- Spatial Operations and Transformations
- Morphological Operations
- Pattern Recognition
- Labeling

Spatial filters

- S_{xy} : neighborhood of pixels around the point (x,y) in an image I
- Spatial filtering operates on S_{xy} to generate a new value for the corresponding pixel at output image J

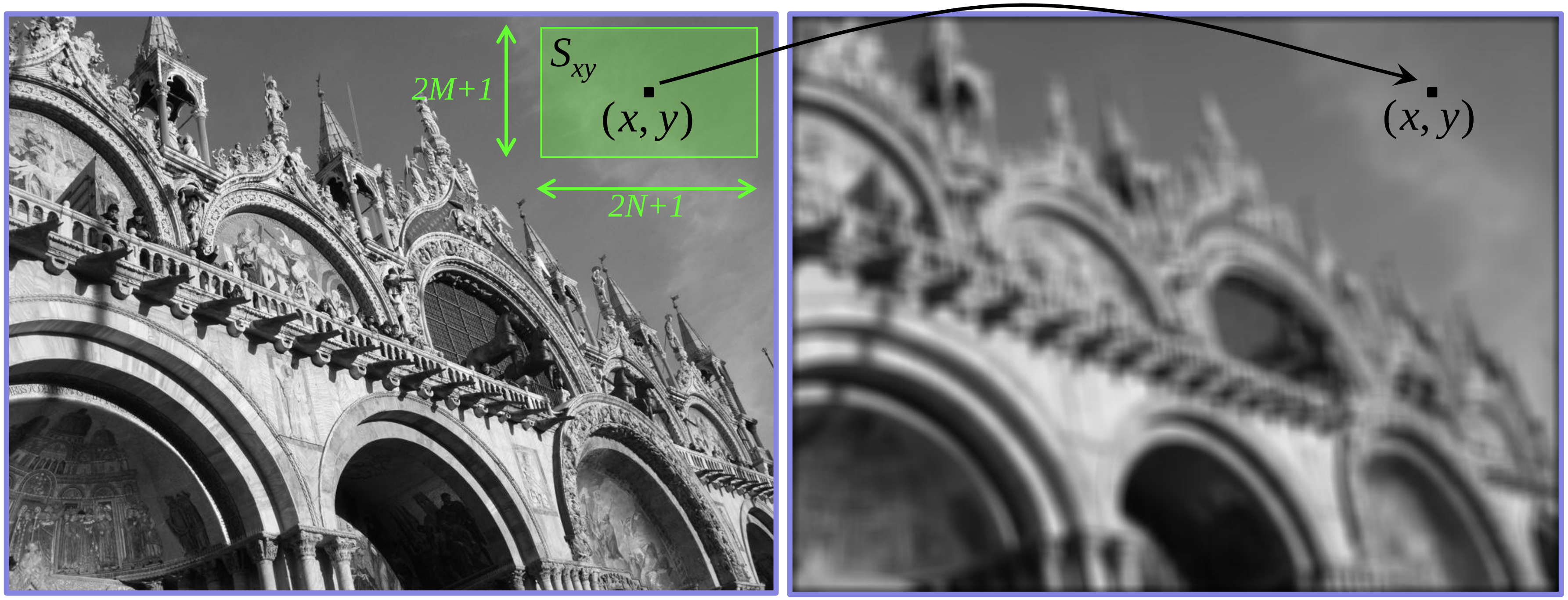


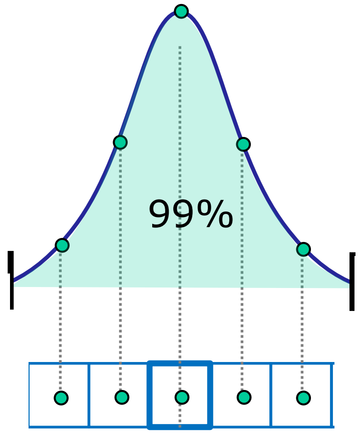
Image I

Filtered Image $J = F(I)$

- For example, an averaging filter is: $J(x,y) = \dfrac{\sum_{(r,c)\in S_{xy}} I(r,c)}{(2M+1)(2N+1)}$

Constructing Filter from a Continuous Fn

- Common practice for image smoothing: use a Gaussian $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

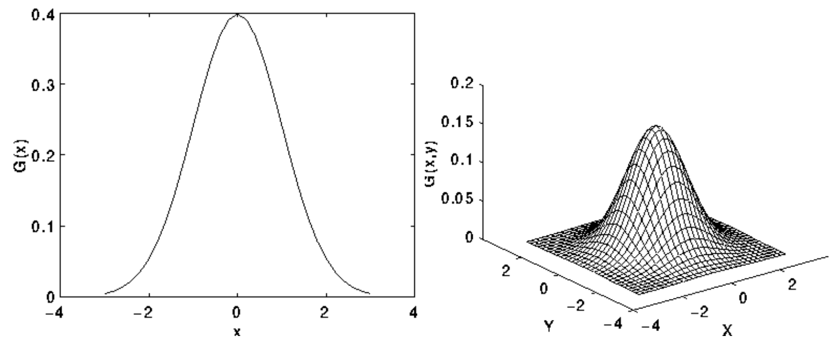


Normalize filter so that values always add up to 1

$\mu = 0$

σ : controls the amount of smoothing

- Near-by pixels have a bigger influence on the averaged value rather than more distant ones



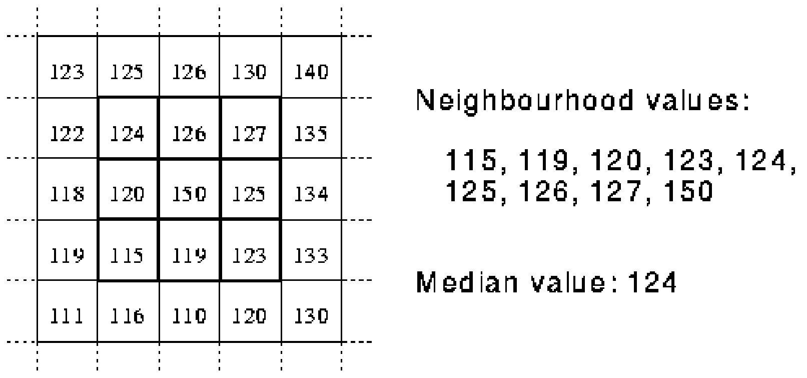
$$\frac{1}{273}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 7 & 4 & 1 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 7 & 26 & 41 & 26 & 7 \\ \hline 4 & 16 & 26 & 16 & 4 \\ \hline 1 & 4 & 7 & 4 & 1 \\ \hline \end{array}$$

Figure 1: Gaussian filter. Left: 1-D Gaussian with mean=0 and $\sigma = 1$. Middle: 2-D Gaussian with mean=0 and $\sigma = 1$. Right: $5x5$ convolution mask for Gaussian smoothing with mean=0 and $\sigma = 1$

- *Mean Averaging Filter:* This filter just averages the pixel values in a neighborhood around a pixel. Neighborhood sizes are variable, depending upon the spatial extent of the filter needed. Common sizes are 3x3, 5x5, 7x7 etc. A 3x3 mean filter uses the following set of local weights:

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

- *Gaussian Smoothing Filter:* Another smoothing filter is the Gaussian filter, which uses a neighborhood that approximates the fall-off of a Gaussian centered on the pixel of interest. This filter has larger weights for the central pixels and nearest neighbors rather than the mean filter which treats all pixels in the neighborhood with equal weights. See figure 1 above.



123	125	126	130	140
122	124	126	127	135
118	120	150	125	134
119	115	119	123	133
111	116	110	120	130

Neighbourhood values:

115, 119, 120, 123, 124,
125, 126, 127, 150

Median value: 124

Figure 2: Median filter. Noisy pixel in center (150) is removed by median of its neighborhood.

- Median Filter: This filter is used to remove outlier noise values in a region. It is based upon *order statistics*, and is a non-linear filter. In this filter, pixels in a neighborhood are sorted by value, and the *median* value of the pixel's in the neighborhood is taken to be the filter's response. If the pixel being processed is an outlier, it will be replaced by the median value. This filter is useful for "shot" or "salt-and-pepper" noise. See figure 2.

3.2 Enhancement

Often, most of the image values will be centered within a limited range of the full 256 gray levels of an image. *Contrast stretching* performs a linear remapping from the gray level range (I_{low}, I_{high}) to $(0, 255)$, effectively "stretching" the contrast in the image. See figure 3. Before the stretching can be performed it is necessary to specify the upper and lower pixel value limits over which the image is to be normalized. Often these limits will just be the minimum and maximum pixel values in the image. For example for 8-bit graylevel images the lower and upper limits might be 0 and 255. Call the lower and the upper limits a and b respectively.

The simplest sort of normalization then scans the image to find the lowest and highest pixel values currently present in the image. Call these c and d. Then each pixel P is scaled using the following function: $P_{out} = (P_{in} - c)(\frac{b-a}{d-c}) + a$

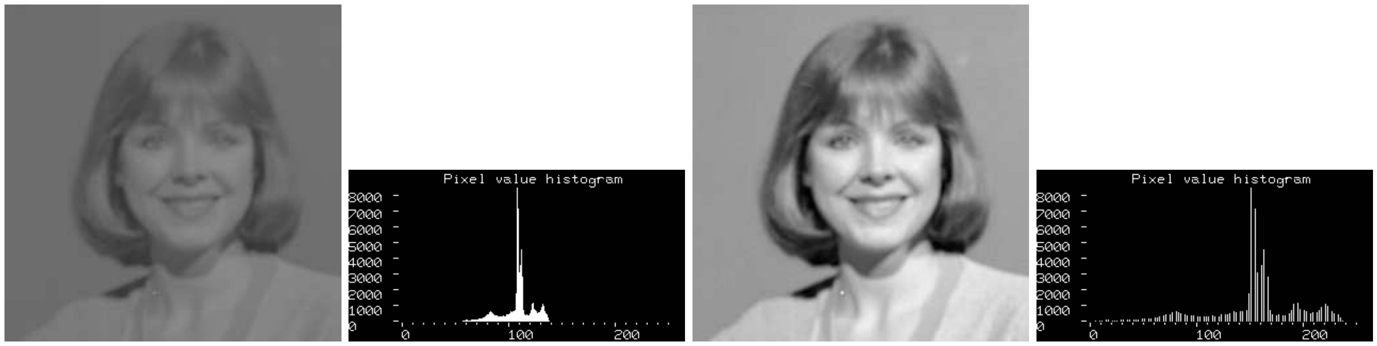


Figure 3: Contrast stretching. Original image and histogram and stretched image and histogram.

Histogram equalization is used to change the response over the entire range of gray values. Often, it is used to create a *uniform* histogram that has all gray values used at the same frequency. This may or may not be useful: large homogeneous regions can get remapped into many gray levels, introducing texture(see figure 4). If an image has R rows and C columns, and there are N gray levels $z_1, z_2, z_3, \ldots, z_n$ total (e.g. 256) then uniform histogram equalization requires each gray value to occur $q = \frac{R \times C}{N}$ times. Using the original histogram, we define $H_{in}[i]$ as the number of pixels in the original image having gray level z_i. The first gray level threshold t_1 is found by advancing i in the input image histogram until q pixels are used. All input image pixels with gray level $< t_1$ will be mapped to gray level z_1 in the output image:

$$\sum_{i=1}^{t_1 - 1} H_{in}[i] \leq q < \sum_{i=1}^{t_1} H_{in}[i]$$

This means that t_1 is the smallest gray level such that the original histogram contains no more thatn q pixels with lower gray values. The kth threshold t_k is defined by continuing the iteration:

$$\sum_{i=1}^{t_k - 1} H_{in}[i] \leq k \cdot q < \sum_{i=1}^{t_k} H_{in}[i]$$

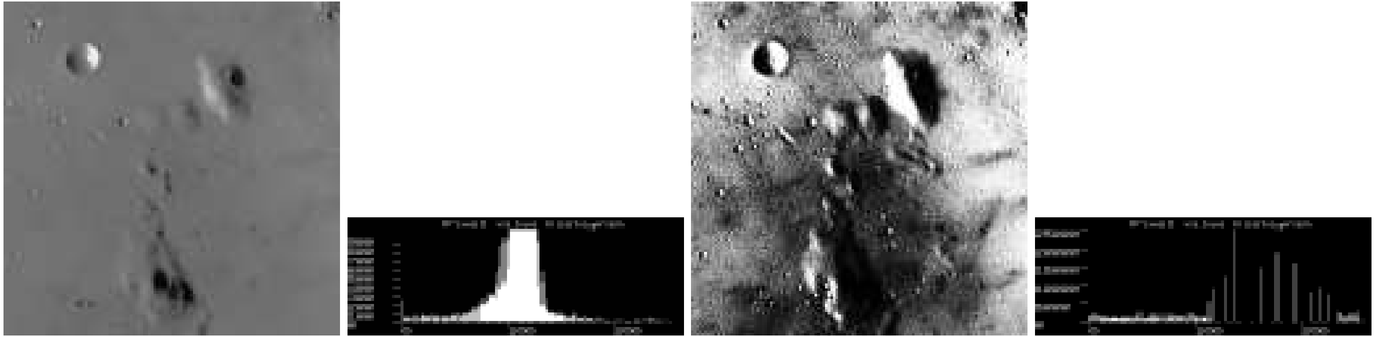
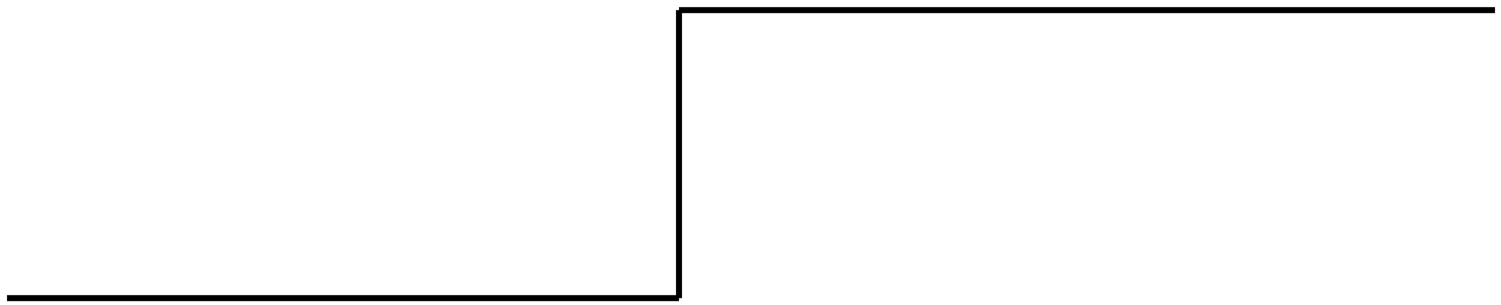


Figure 4: Histogram Equalization. Original image and histogram and equalized image and histogram. See http://www.dai.ed.ac.uk/HIPR2/histeq.htm.

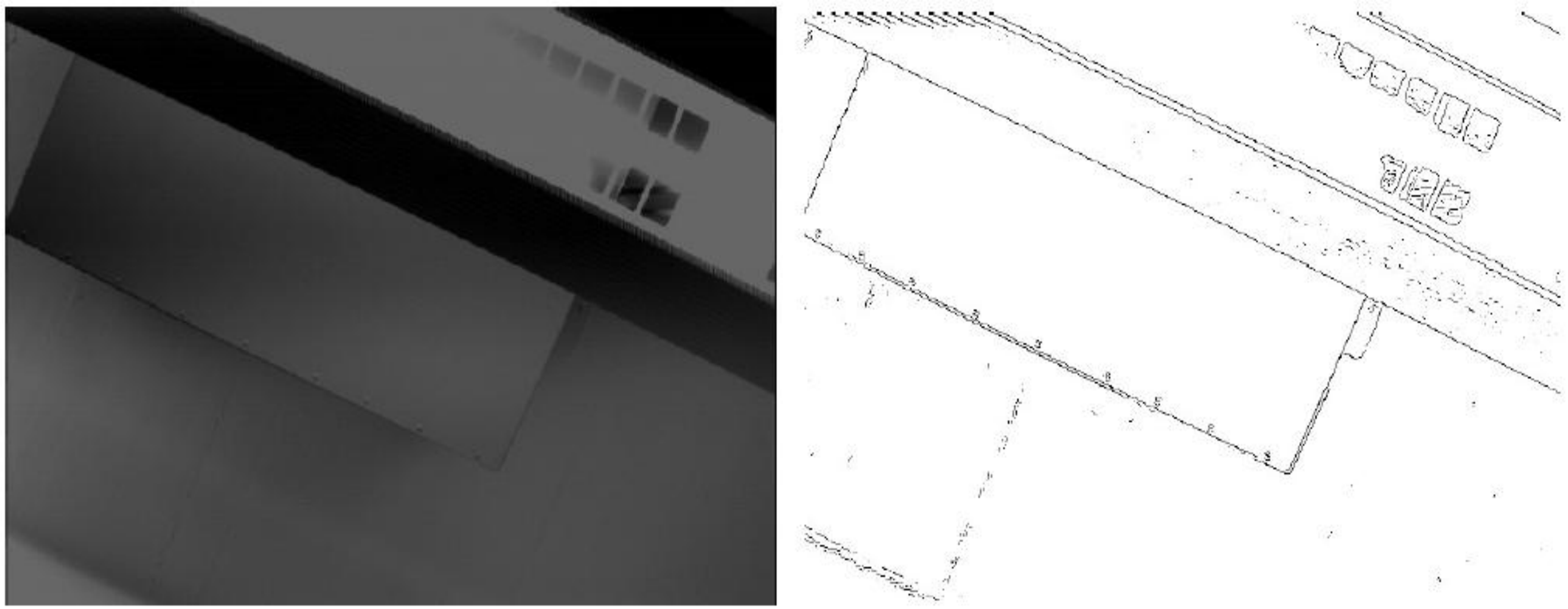
Edge = intensity discontinuity in one direction

- Edges correspond to sharp changes of intensity
- Change is measured by 1st order derivative in 1D
- Big intensity change ⇨ magnitude of derivative is large
- Or 2nd order derivative is zero.



Edge Detection

- Ultimate goal of edge detection: an idealized line drawing.
- Edge contours in the image correspond to important scene contours.



What is **USEFUL**, What is **REDUNDANT** ?

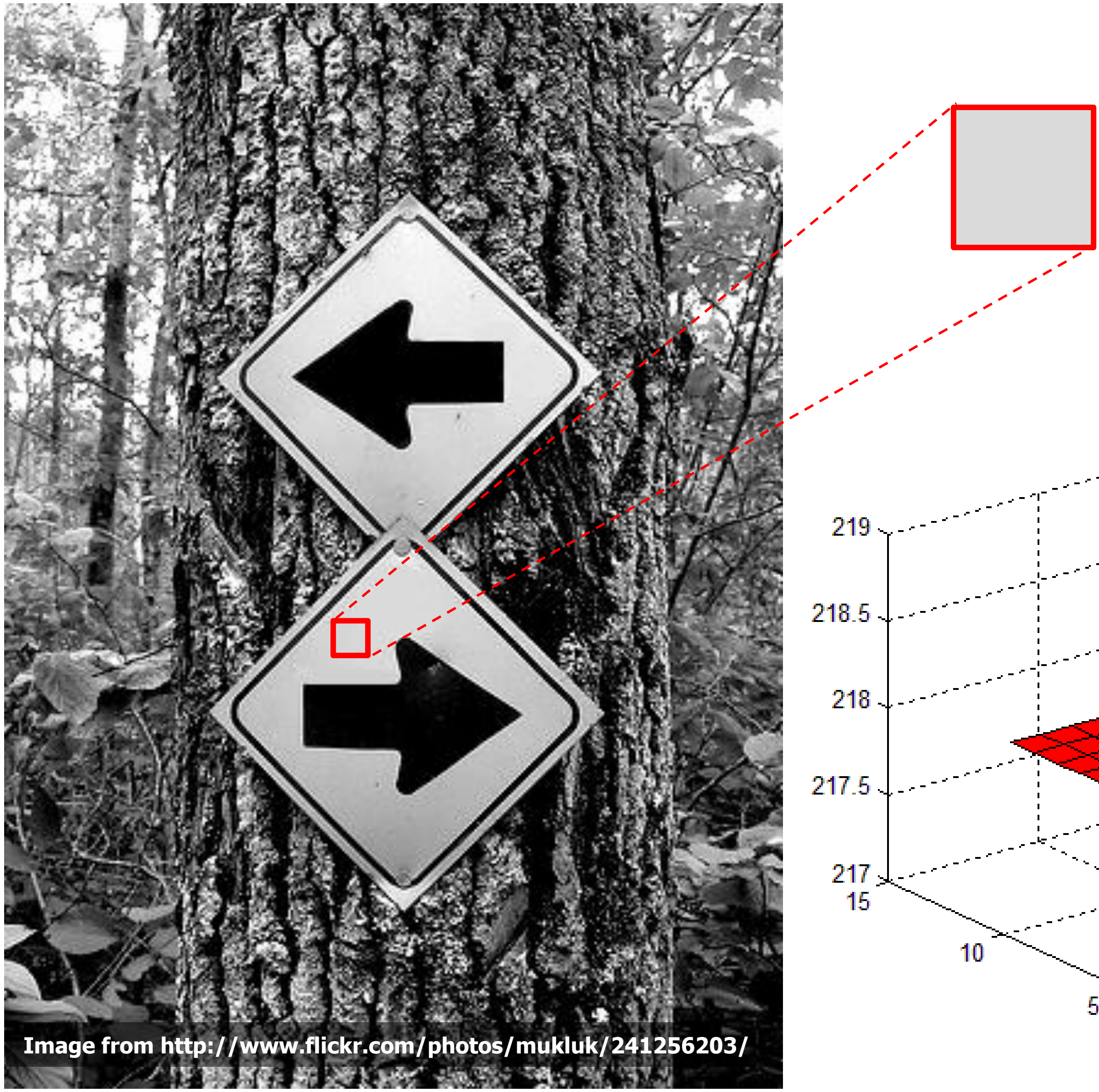
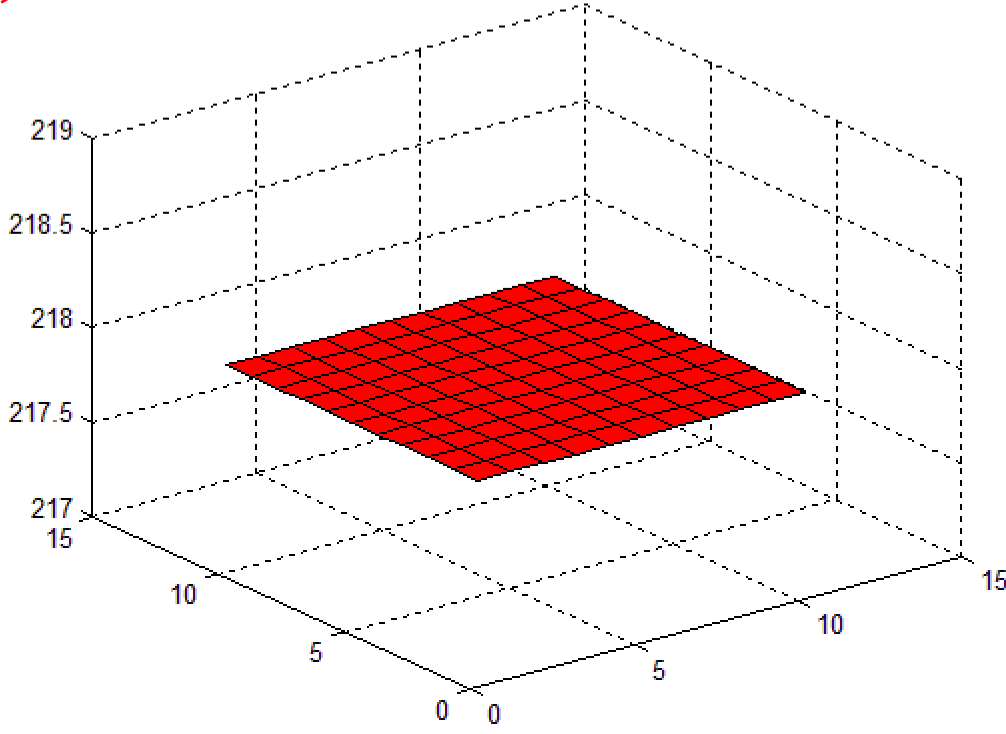
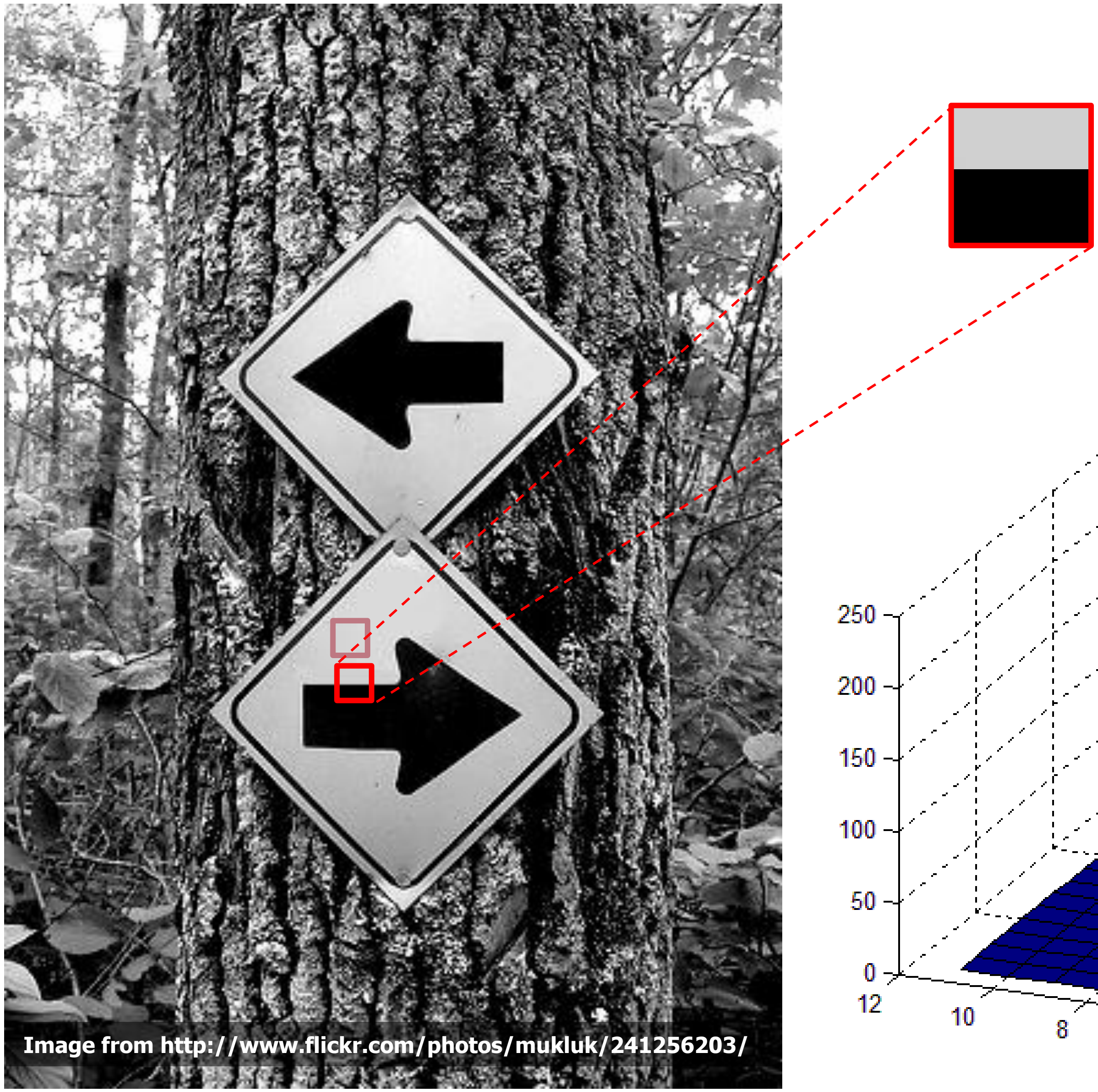


Image from http://www.flickr.com/photos/mukluk/241256203/

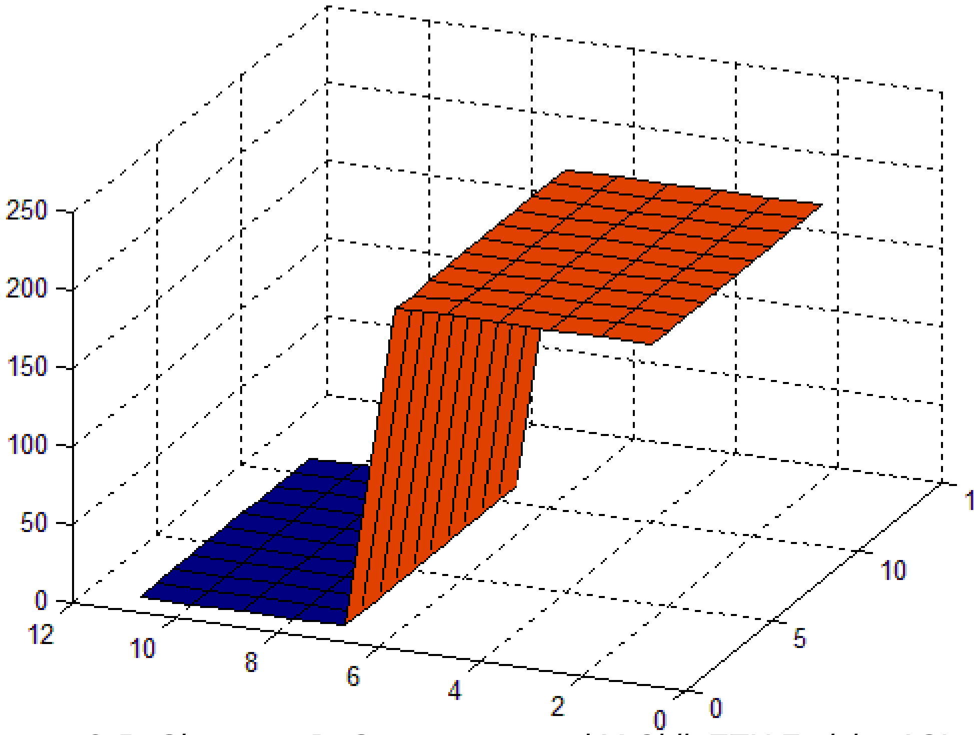
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218



What is **USEFUL**, What is **REDUNDANT** ?



208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	207	208	208	208	208	208	208	208	208	208
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0



What is **USEFUL**, What is **REDUNDANT** ?

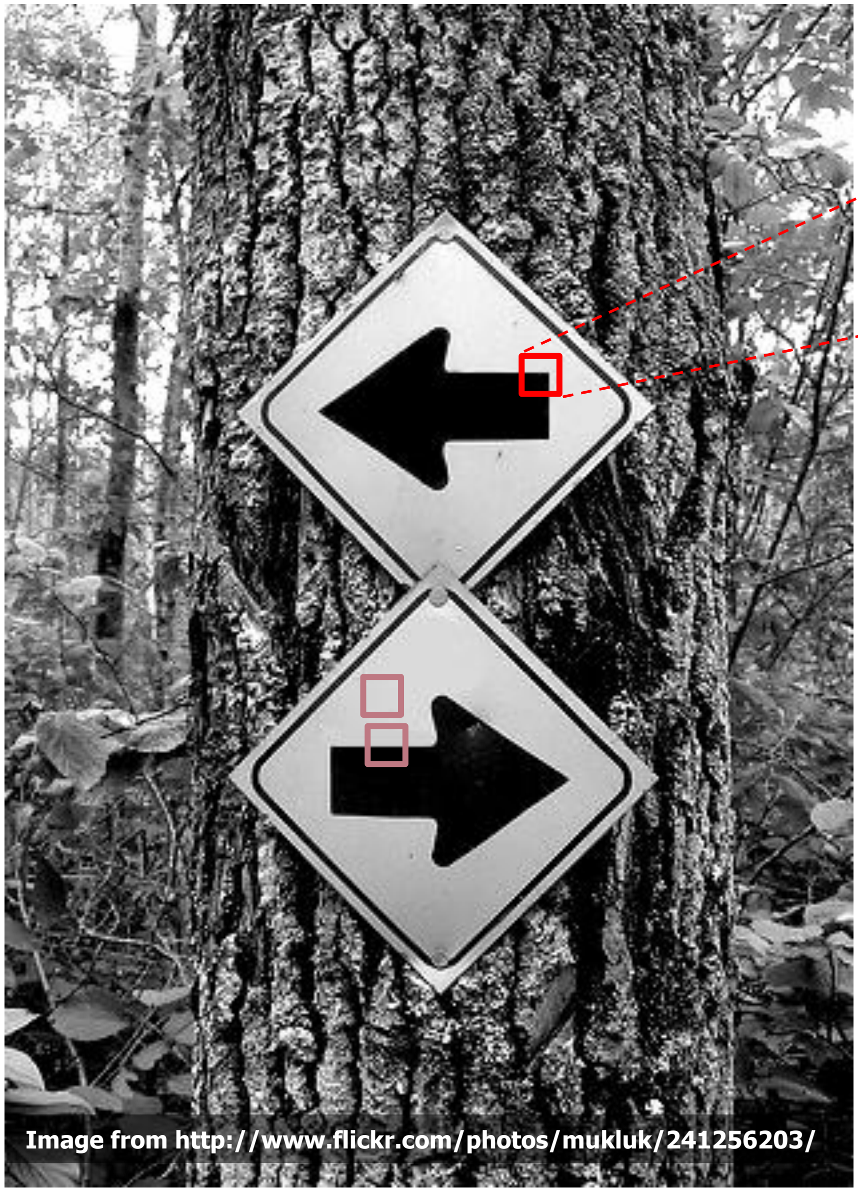
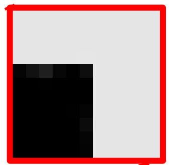
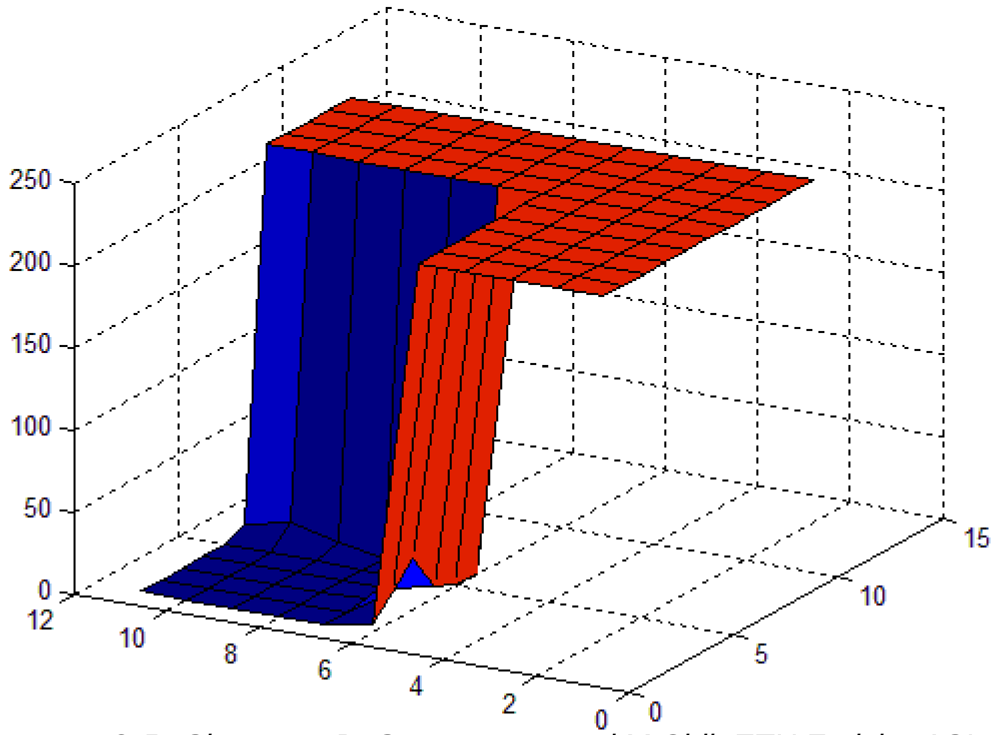


Image from http://www.flickr.com/photos/mukluk/241256203/



229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	230	229	229	229	229	229
5	17	31	7	1	0	229	229	229	229	229
0	0	1	0	0	0	229	229	229	229	229
0	0	0	0	0	0	229	229	229	229	229
0	0	0	0	1	4	229	229	229	229	229
0	0	0	0	0	11	229	229	229	229	229
0	0	0	0	0	5	229	229	229	229	229



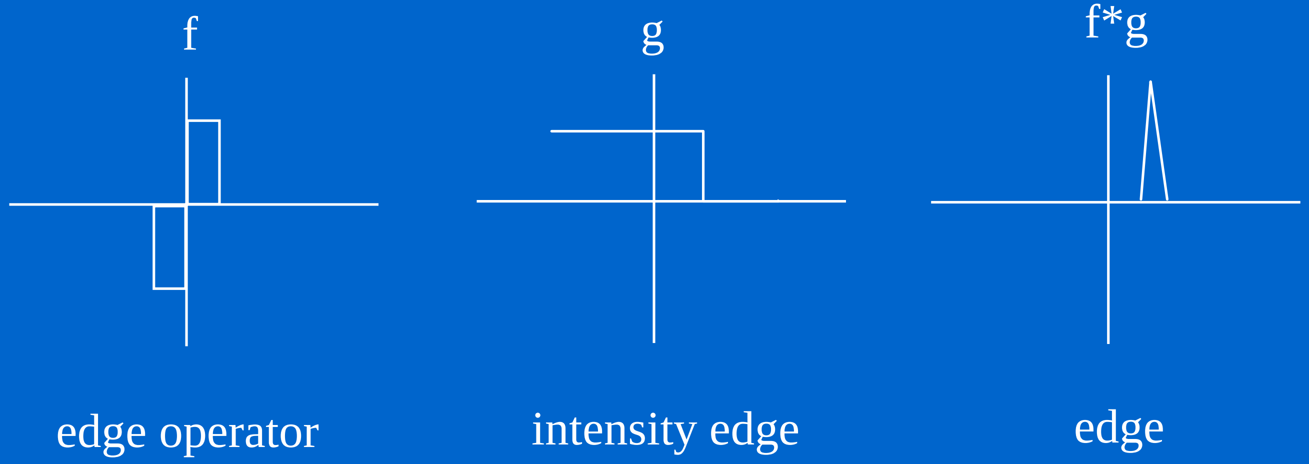
Edge Detection

	x	y
• **Sobel**	$\begin{matrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{matrix}$	$\begin{matrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{matrix}$
• **Roberts**	$\begin{matrix}0 & 1\\ -1 & 0\end{matrix}$	$\begin{matrix}1 & 0\\ 0 & -1\end{matrix}$
• **Prewitt**	$\begin{matrix}-1 & 0 & 1\\ -1 & 0 & 1\\ -1 & 0 & 1\end{matrix}$	$\begin{matrix}-1 & -1 & -1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{matrix}$

- **Laplacian of Gaussian (LOG)**

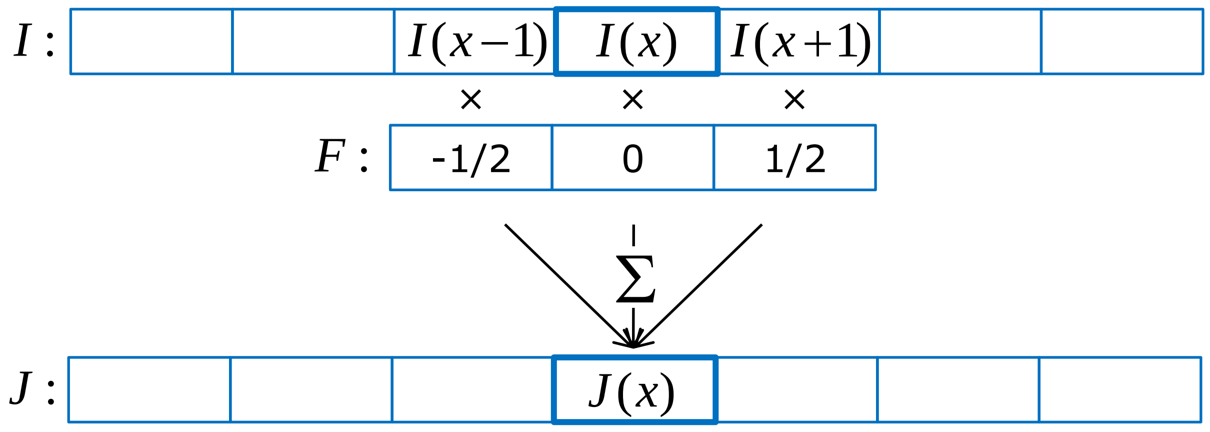
Edge Detection

- **1-D edge detection**



Taking derivatives with Correlation

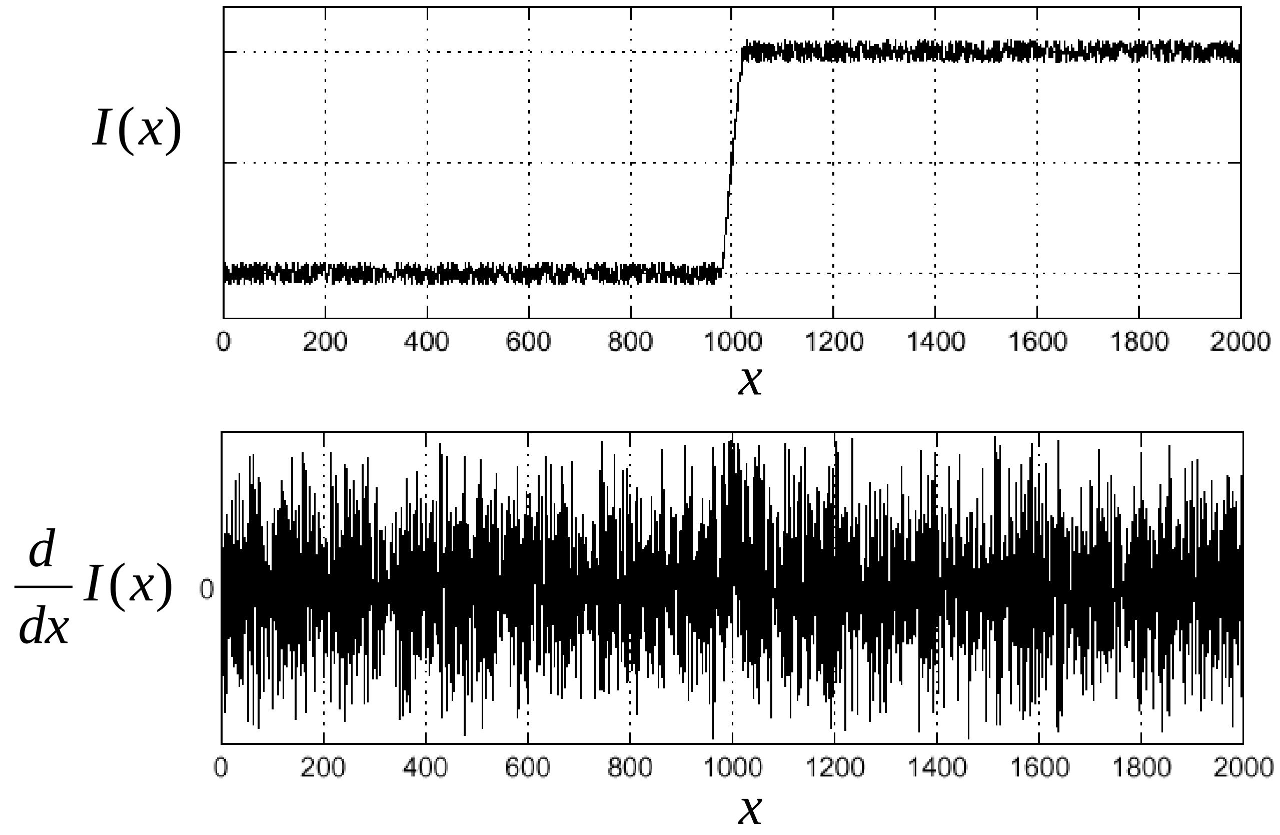
- Derivative of an image: quantifies how quickly intensities change (along the direction of the derivative)
- Approximate a derivative operator:



$$J(x) = \frac{I(x+1) - I(x-1)}{2}$$

1D Edge detection

- Image intensity shows an obvious change



- Where is the edge? ⇨ image noise cannot be ignored

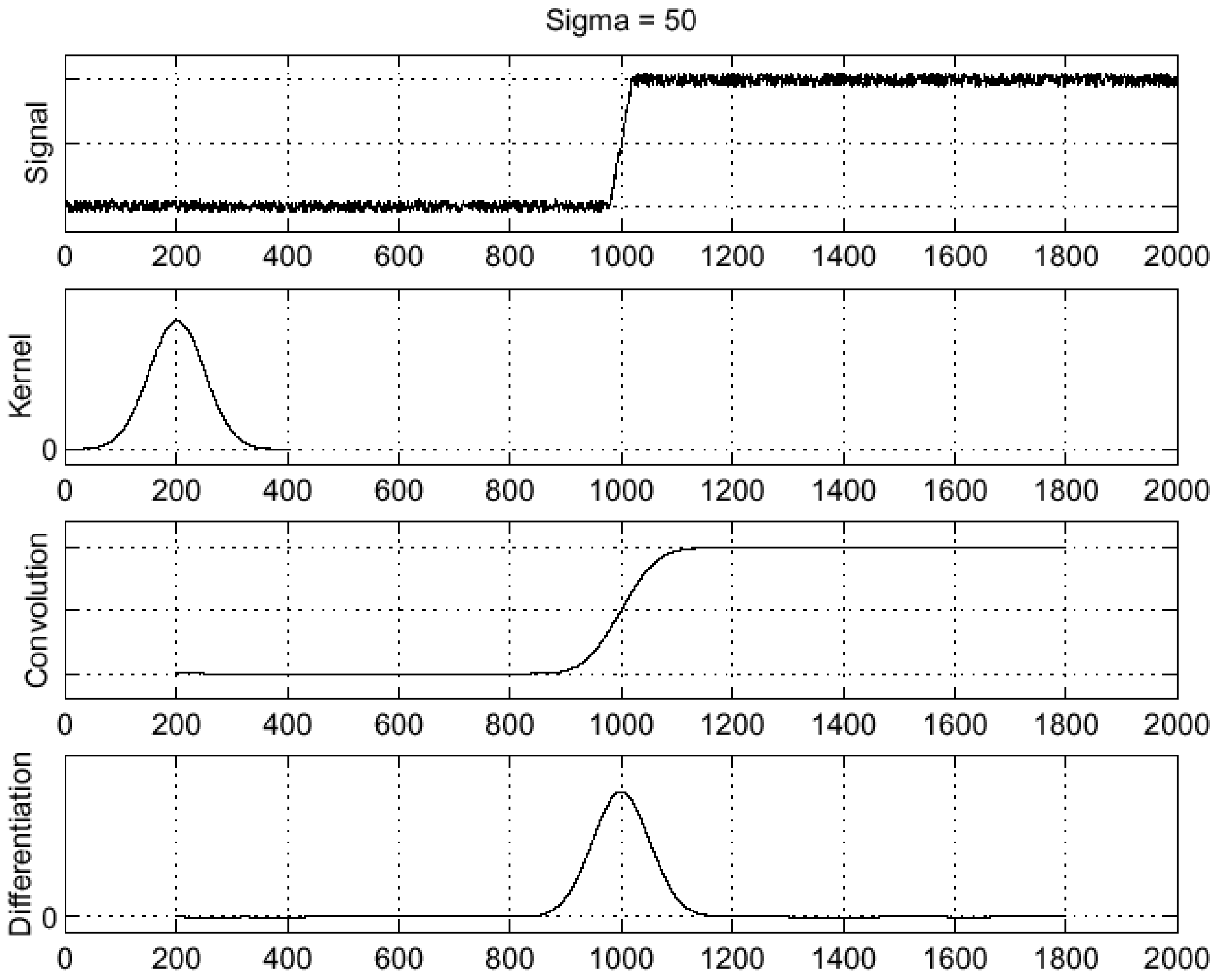
Solution: smooth first

$$I(x)$$

$$G_\sigma(x)$$

$$s(x) = I(x) * G_\sigma(x)$$

$$s'(x) = \frac{d}{dx}\left(s(x)\right)$$



- Where is the edge? At the extrema of $s'(x)$

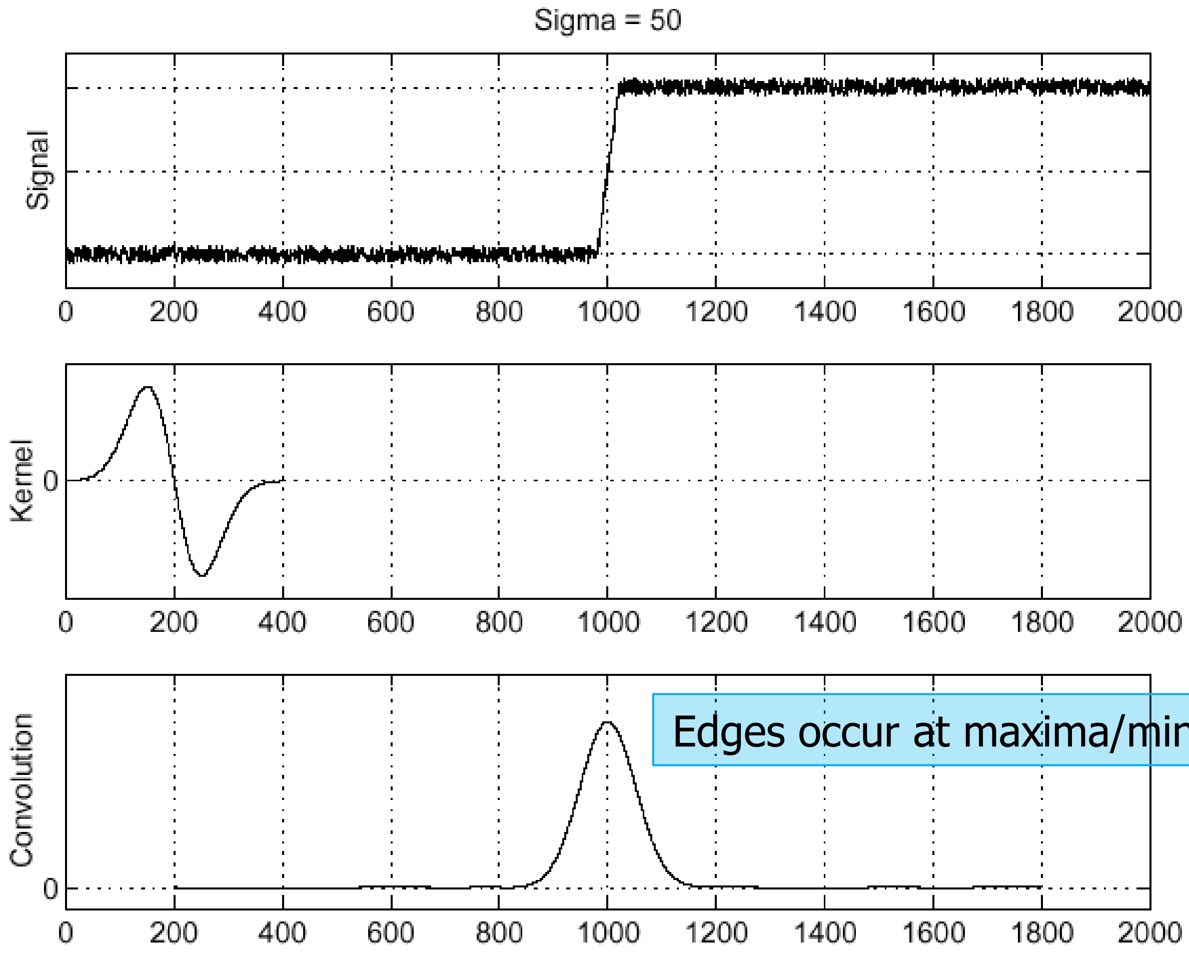
Derivative theorem of convolution

- $$s'(x) = \frac{d}{dx}\left(G_\sigma(x) * I(x)\right) = G'_\sigma(x) * I(x)$$
- This saves us one operation:

$$I(x)$$

$$G'_\sigma(x) = \frac{d}{dx} G_\sigma(x)$$

$$s'(x) = G'_\sigma(x) * I(x)$$



Edges occur at maxima/minima of $s'(x)$

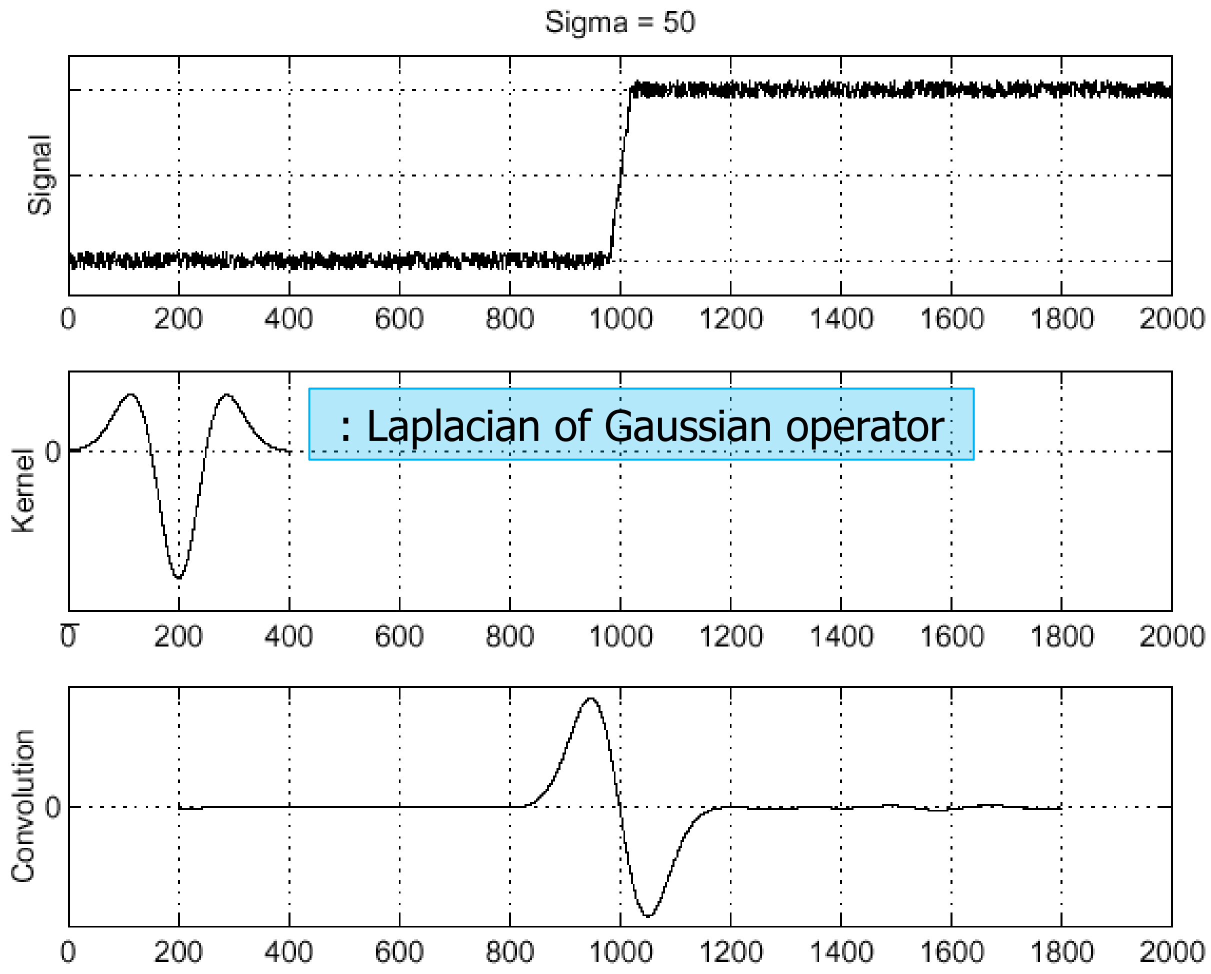
Zero-crossings

- Locations of Maxima/minima in $s'(x)$ are equivalent to zero-crossings in $s''(x)$

$$I(x)$$

$$G''_\sigma(x) = \frac{d^2}{dx^2} G_\sigma(x)$$

$$s''(x) = G''_\sigma(x) * I(x)$$



2D Edge detection

- Find gradient of smoothed image in both directions

Usually use a separable filter such that: $G_\sigma(x,y) = G_\sigma(x)G_\sigma(y)$

$$\nabla S = \nabla(G_\sigma * I) = \begin{bmatrix} \frac{\partial(G_\sigma * I)}{\partial x} \\ \frac{\partial(G_\sigma * I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix} = \begin{bmatrix} G'_\sigma(x)G_\sigma(y) * I \\ G_\sigma(x)G'_\sigma(y) * I \end{bmatrix}$$

- Discard pixels with $|\nabla S|$ below a certain below a certain threshold

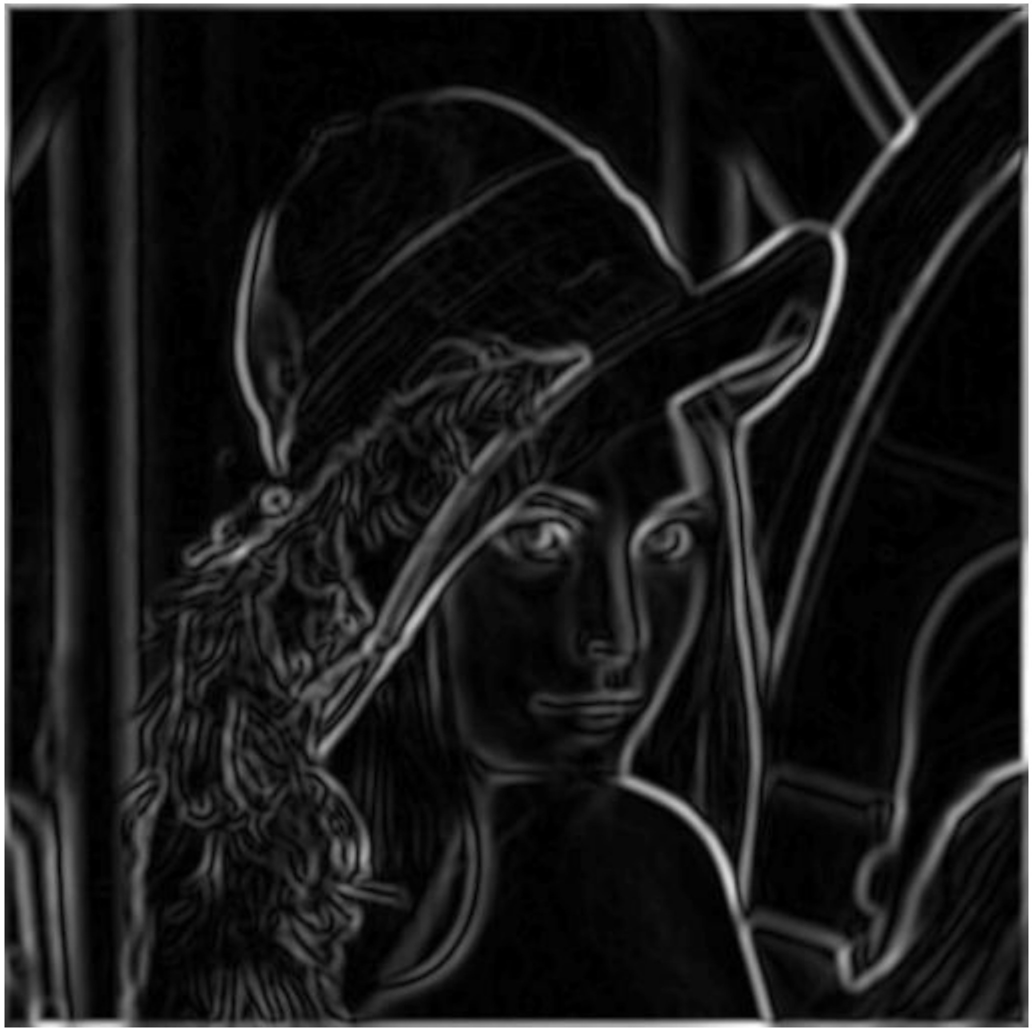
- **Non-maximal suppression**: identify local maxima of $|\nabla S|$ along the directions $\pm|\nabla S|$

2D Edge detection: Example

I : original image (Lena image)

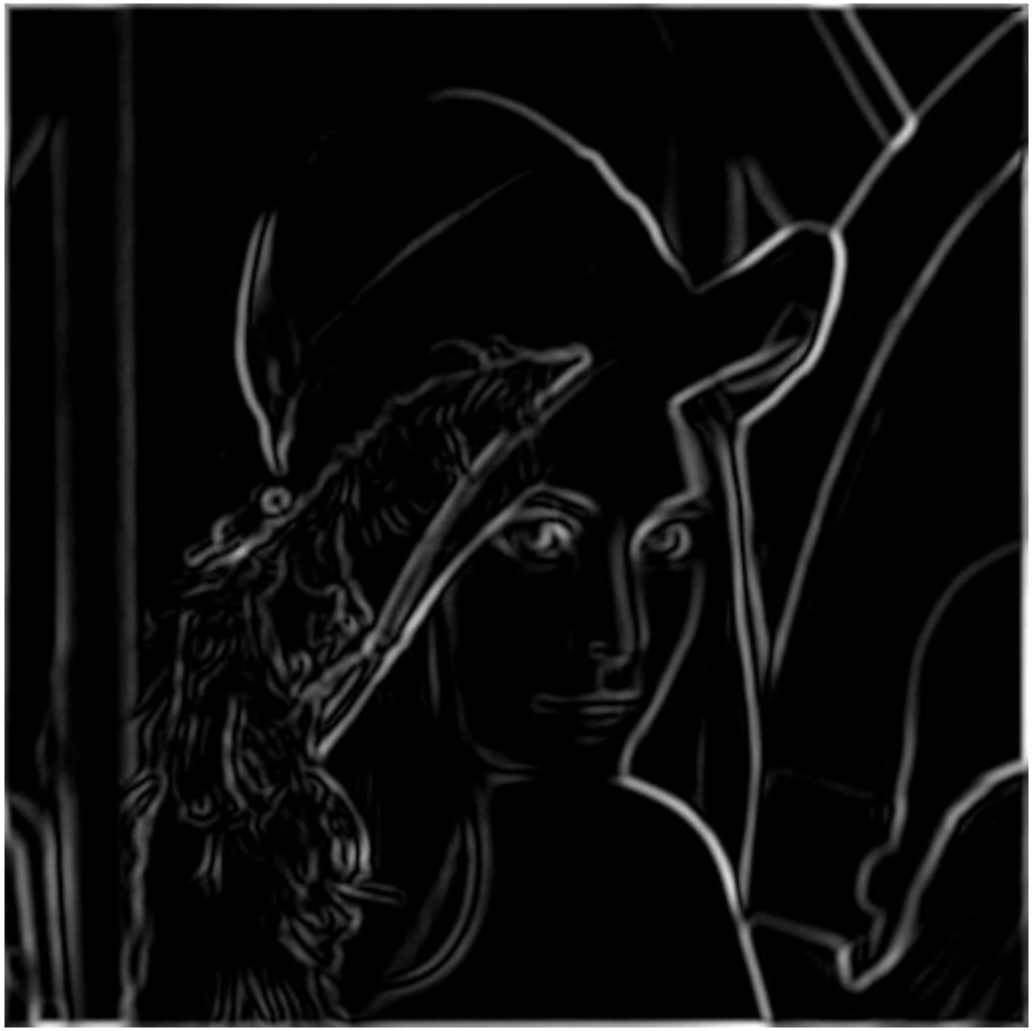
2D Edge detection: Example

$$\nabla S = \nabla\left(G_\sigma * I\right)$$



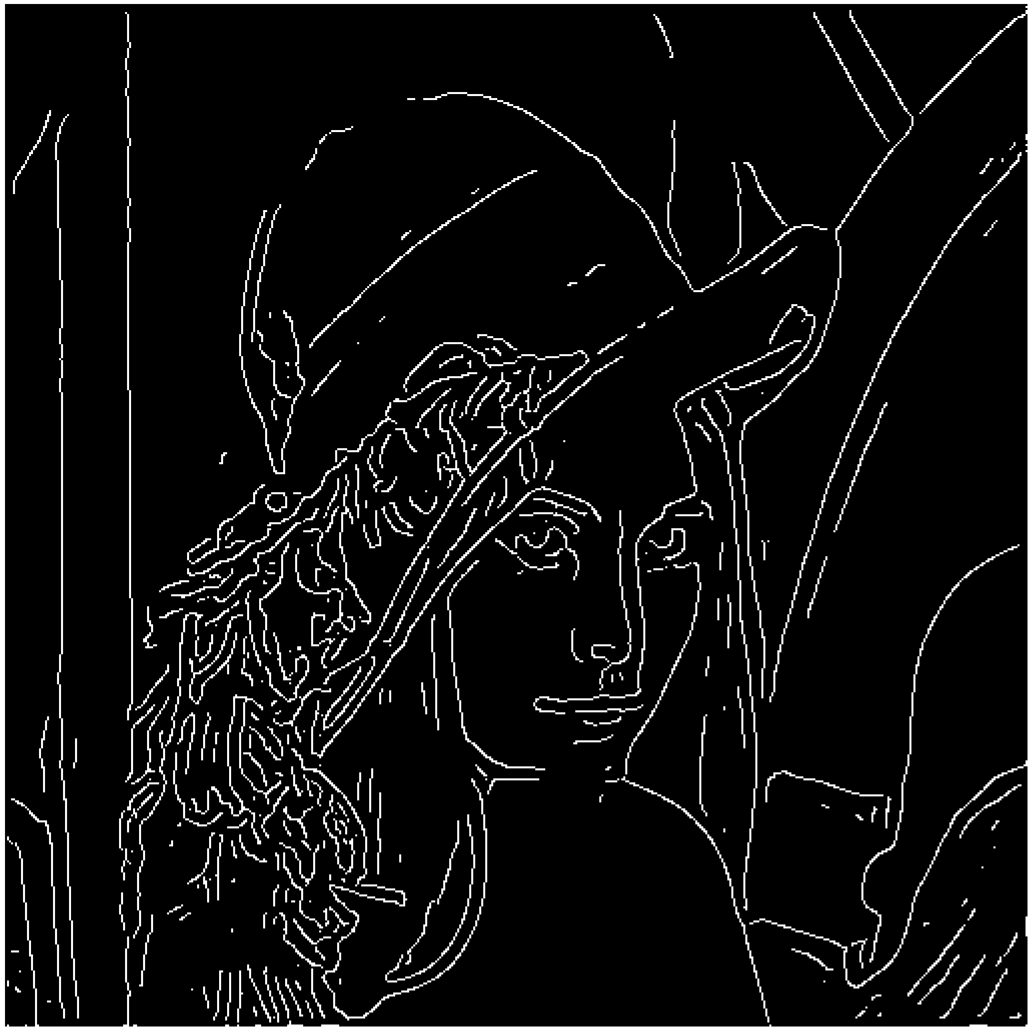
$|\nabla S|$: Edge strength

2D Edge detection: Example



Thresholding $|\nabla S|$

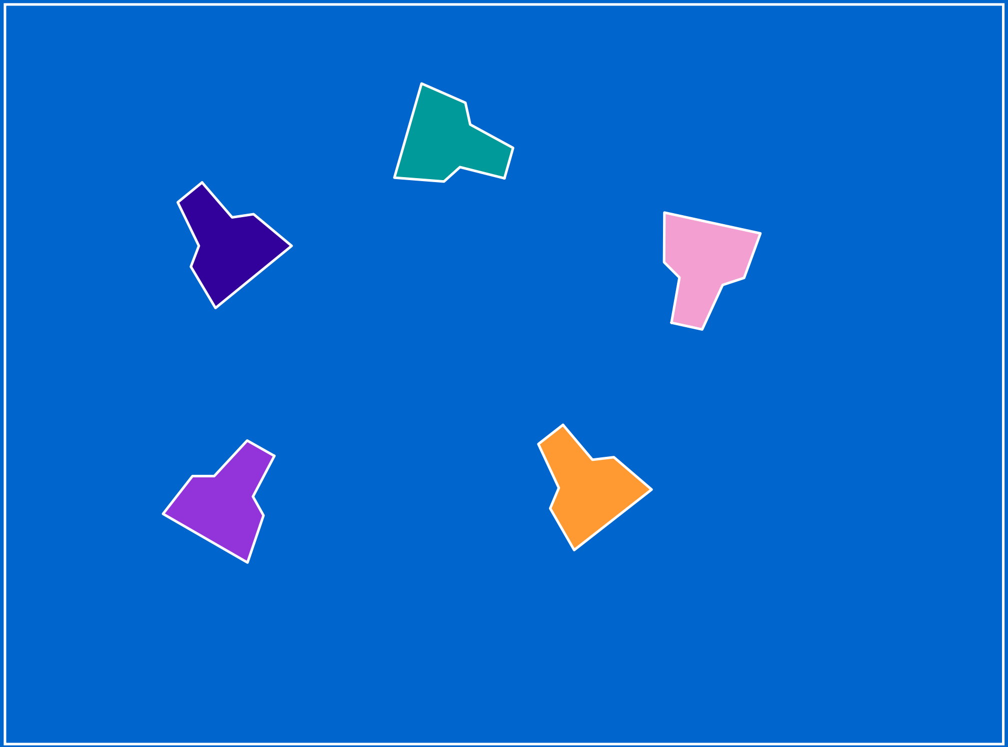
2D Edge detection: Example



Thinning: non-maximal suppression

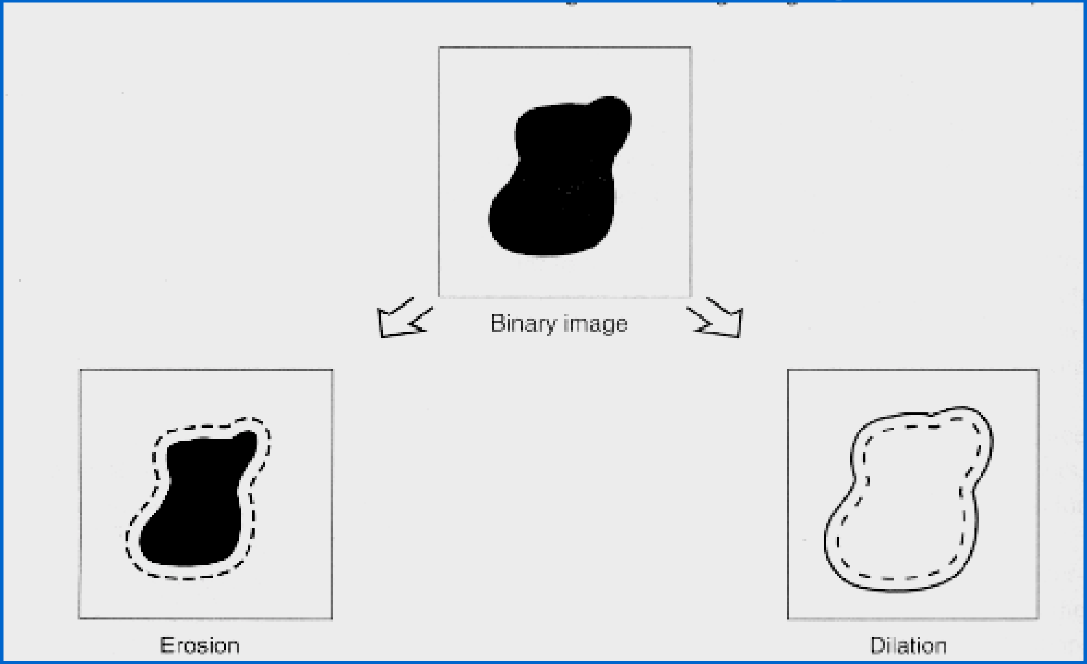
Labeling

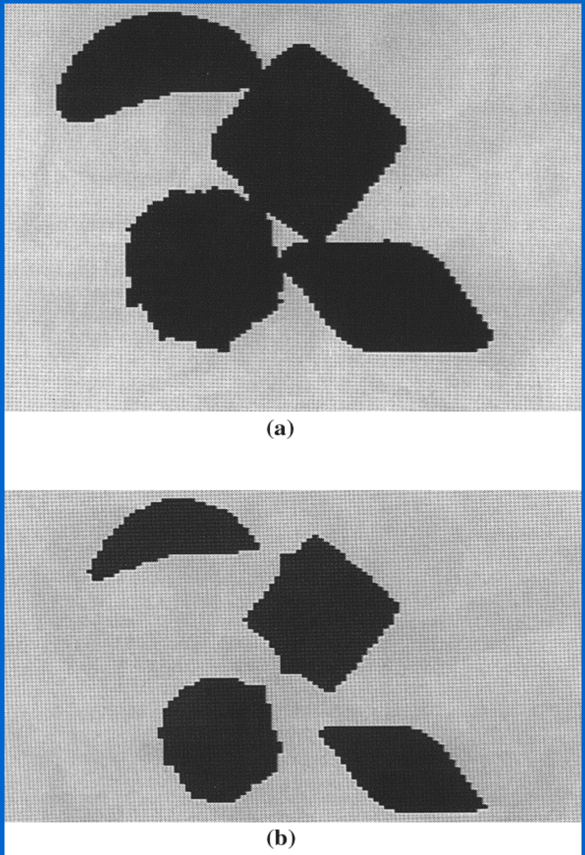
- **Labeling parts and regions**



Segmentation: Morphological Operations

- **Erosion - Shrinking**
- **Dilation - Growing**





Separation of touching objects by SHRINK operations. Here objects (chocolates) in (a) are shrunk (b) in order to separate them so that they may be counted reliably.

Middle Level Vision

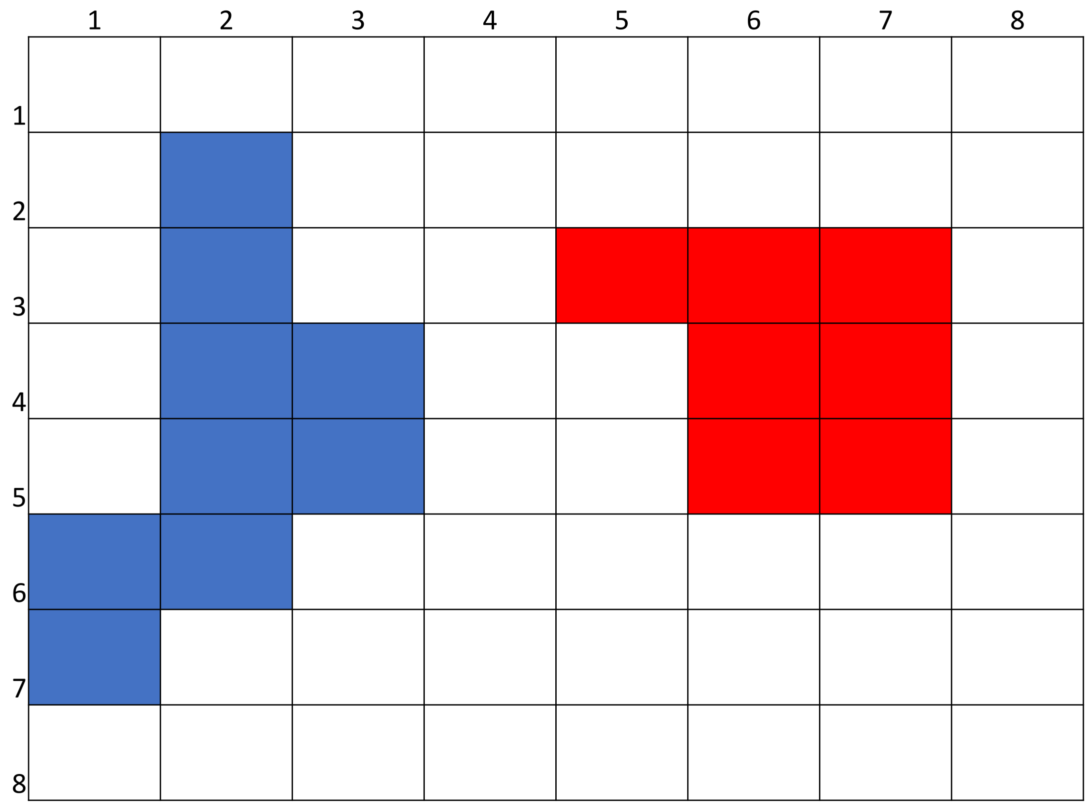
Middle level vision tries to move beyond the pixel level to larger abstractions including shape and geometry.

Region Labeling: Recursive Region Growing

Recursive region growing is a simple method. Starting from a binary image, it scans the image for any foreground pixels (not black). For each foreground pixel, it labels that pixel with a unique label, “grows” the pixel by coloring any of its non-black 4-neighbors with this unique color label, and pushing these pixels on a queue. The queue is then processed until empty. All 4-connected pixels in the region will be labeled consistently. Recursive method can be slow however, and may need large memory for recursive calls.

```
# Recursive Region Grower
# Do the following for every univisited SEED pixel…..

Input: Binary image – White(255) = foreground,  Black(0) = background. Output: labeled regions

Choose a foreground SEED pixel ( pixel whose value =  White),  I(c,r)
Enqueue(c,r), and mark (c,r) as Visited
Label = K   # random color
While Queue !Empty do
        (c,r) = Dequeue
        Out_image(c,r) = K
        if I(c-1,r) !Visited && I(c-1,r) == White # WEST neighbor pixel, hasn’t been Visited and is foreground pixel
              enqueuer(c-1,r), mark (c-1,r) as Visited
        if I(c+1,r) !Visited && I(c+1,r) == White # EAST neighbor pixel, hasn’t been Visited and is foreground pixel
              enqueuer(c+1,r), mark (c+1,r) as Visited
         if I(c,r-1) !Visited && I(c,r-1) == White # NORTH neighbor pixel, hasn’t been Visited and is foreground pixel
              enqueuer(c,r-1), mark (c,r-1) as Visited
         if I(c,r+1) !Visited && I(c,r+1) == White # SOUTH neighbor pixel, hasn’t been Visited and is foreground pixel
              enqueuer(c,r+1), mark (c,r+1) as Visited
```

Recursive Region Grower - Seed Pixel is (2,2)



Queue	Queue	Queue	Queue
2,2	2,3	2,4	3,4 2,5

Queue	Queue	Queue	Queue
2,5 3,5	3,5 2,6	2,6	1,6

Queue	Queue
1,7	empty

4.2 Region Labeling: Blob Coloring

This algorithm uses 2 passes. The first pass labels each pixel and the second pass merges the labels into a consistent labeling.

Let the initial color, $k = init_{color}$, and choose a color_increment to change the color each time a new blob is found. Scan the image from left to right and top to bottom. Assign colors to each non-zero pixel in pass 1. In pass2, we merge the regions whose colors are equivalent. To maintain the equivalence table between merged colors, we can use a standard disjoint set Union-Find data structure.

```
If $I(x_C) = 0$ then continue
else begin

        if$I(x_U) = 1$ and $I(x_L) = 0$
        then color $(x_C)$: = color $(x_U)$
        if$I(x_L) = 1$ and $I(x_U) = 0$
        then color $(x_C)$: = color $(x_L)$
        if$I(x_L) = 1$ and $I(x_U) = 1$
        then begin /* two colors are equivalent. */
            color $(x_C)$: = color $(x_L)$
            color $(x_L)$ is equivalent to color $(x_U)$
            end
        if$I(x_L) = 0$ and $I(x_U) = 0$ /* new color */
        then color $(x_C)$: = k; k: = k + color_increment

end
```

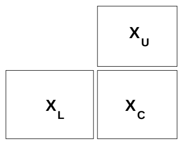


Figure 6: Image topology of x_u, x_c, x_l for region growing

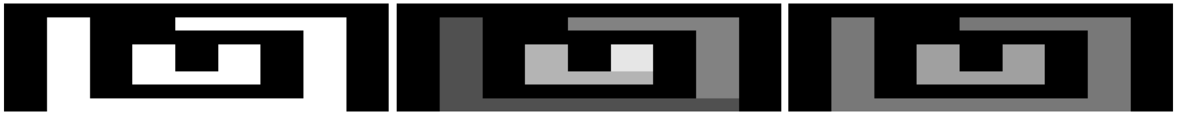


Figure 7: Blob coloring. Left: original binary image. Middle: blob color assignment after first pass. Right: Blob color assignment after merging colors.

Below are 3 ascii images, showing the original test pattern in figure 6, the first pass results, and the final image after region labels are merged. The initial color=80 and the color increment is 50.

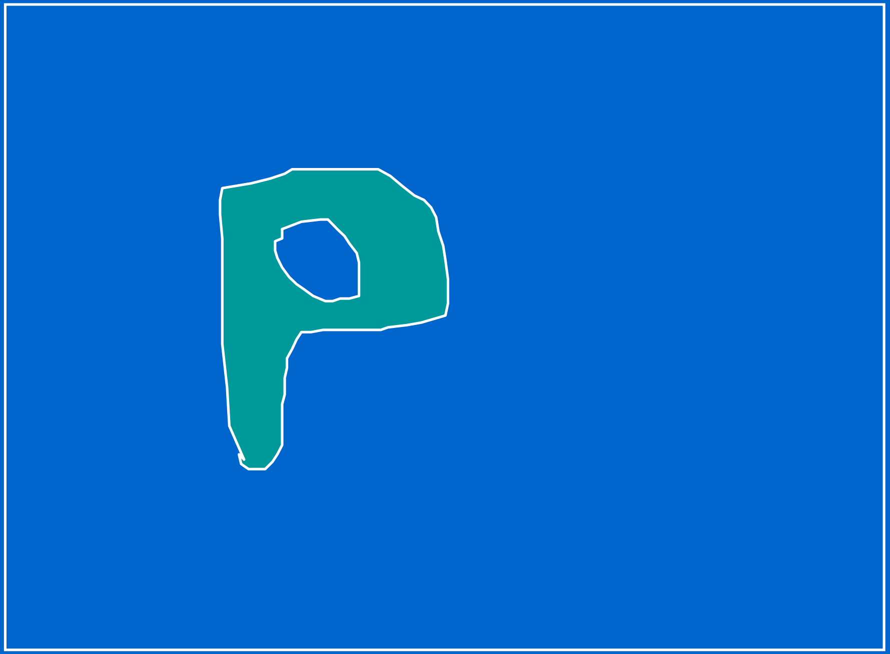
```
0    0    0    0    0    0    0    0    0
0    1    0    0    1    1    1    1    0
0    1    0    0    0    0    0    1    0
0    1    0    1    0    1    0    1    0
0    1    0    1    0    1    0    1    0
0    1    0    1    1    1    0    1    0
0    1    0    0    0    0    0    1    0
0    1    1    1    1    1    1    1    0

0    0    0    0    0    0    0    0    0
0   80    0    0  130  130  130  130    0
0   80    0    0    0    0    0  130    0
0   80    0  180    0  230    0  130    0
0   80    0  180    0  230    0  130    0
0   80    0  180  180  180    0  130    0
0   80    0    0    0    0    0  130    0
0   80   80   80   80   80   80   80    0

0    0    0    0    0    0    0    0    0
0  130    0    0  130  130  130  130    0
0  130    0    0    0    0    0  130    0
0  130    0  230    0  230    0  130    0
0  130    0  230    0  230    0  130    0
0  130    0  230  230  230    0  130    0
0  130    0    0    0    0    0  130    0
0  130  130  130  130  130  130  130    0
```

Object Classification

- **Object measures**
  - area
  - perimeter
  - length
  - width
  - shape analysis
    - rectangularity
    - circularity

Pattern Recognition

- **Optical Character Recognition**



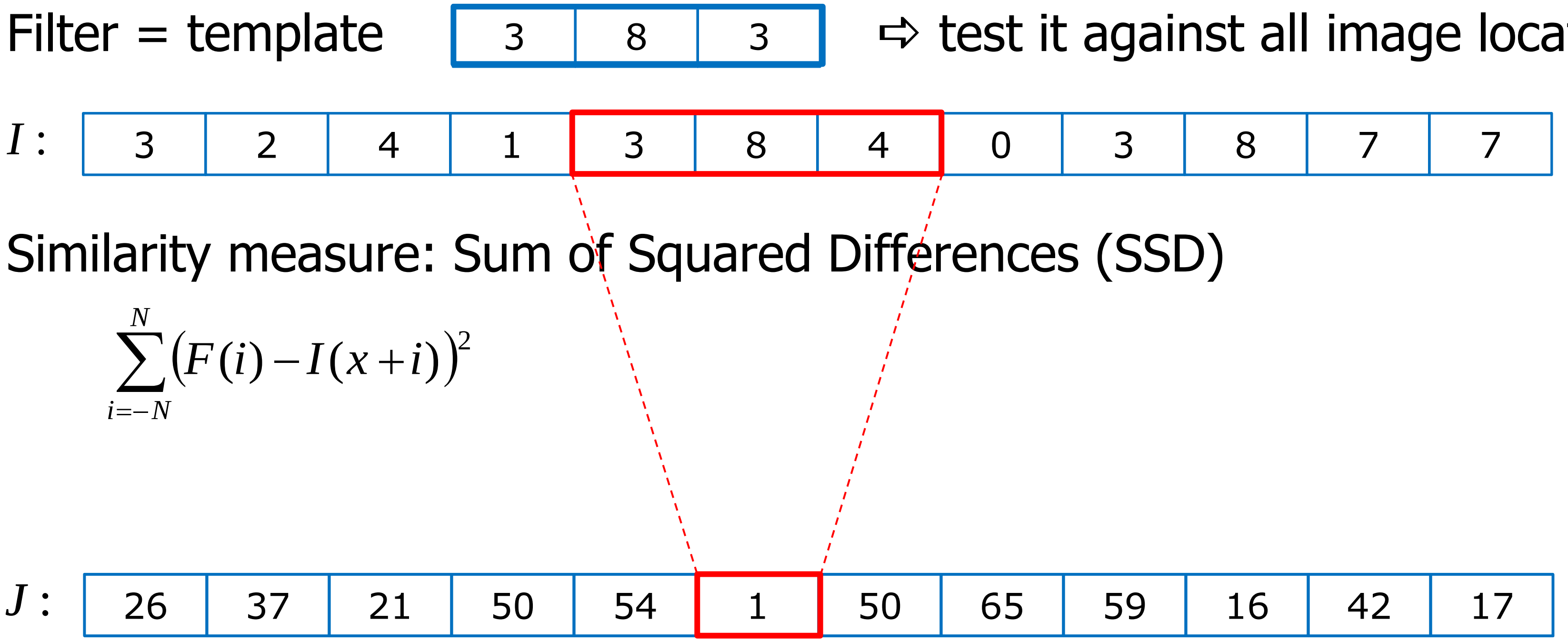
P

Simple Shape Matching

- Template Matching: Simple matching of masks (templates) that contain object’s image structure
- Object is represented as a region of pixels. Region is compared against all other positions in the image.
- Measure is absolute value of difference between template pixels and image pixels - zero means exact match. Find minimum response for template operator and this is best match
- Problems: Translation, Rotation, Scaling, Lighting changes between image and template
- Translation is handled by applying template everywhere in image
- Rotation handled by using a set of templates oriented every few degrees. Increases cost

Matching using Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template ⇨ test it against all image locations



- Similarity measure: Sum of Squared Differences (SSD)

$$\sum_{i=-N}^{N}\left(F(i)-I(x+i)\right)^2$$

- Scaling is more difficult. Can scale templates but not easily. Not clear how many scales to use.
- Lighting changes can be alleviated by using normalized correlation. Use correlation operator and scale template responses by average intensities of image and template.
- Method of Moments: Use statistical properties of object to match.

$$Continuous: \; M_{ij} = \int\int x^i \, y^j \, f(x,y) dx dy \, ; \;\; Discrete: \; M_{ij} = \sum\sum x^i \, y^j \, f(x,y)$$

- If we assume $f(x,y)$ is a mass function that calculates object mass at each point of the object x, y, then these are the moments of inertia from physics.
- If we further assume $f(x,y)$ is binary valued (1= object present in image, 0= no object at x, y) then we can use these moments as shape descriptors
- M_{00} is simply the area of the object in the image. Counts the pixels that contain the object.
- We can calculate the *centroid* of the object. This is equivalent to the point where an object of uniform mass balances. The mass is equally distributed in all directions.

$$X_c \;= \frac{M_{10}}{M_{00}} \;\; . \;\; Y_c \;= \frac{M_{01}}{M_{00}}$$

- By translating the object coordinates by X_c, Y_c, we can move the object to a known coordinate system. These are *central moments*. Creates translational invariance in moment computation.
- Rotational Invariance can be achieved by finding princiapl axis of object. This is the axis of the moment of least inertia. We can always align an object's principal axis with $X \, Y$ or Z axis.
- Scaling invariance is posible using *normalized moments* which scales by an area measure.
- Higher order moments can be used as unique shape descriptors for an object. Problem: simple scalar measures like this are not robust.

5.1 Finding the Principal Axis

Assume a point set centered on the origin: $(x - x_c, y - y_c)$, where the centroid of the points is (x_c, y_c). To find the principal axis we want to find the rotation angle that will align the axis of minimum intertia with the X axis:

We rotate the points by $-\theta$ to align the dataset with the x axis:

$$ROT(Z, -\theta) \; \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \; ; \; \Rightarrow \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos\theta - ysin\theta \\ -xsin\theta + ycos\theta \end{bmatrix}$$

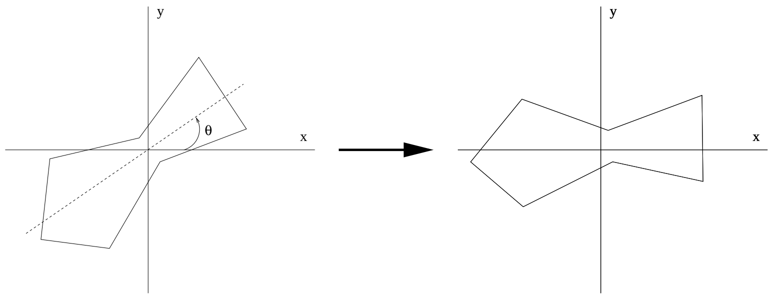


Figure 8: Left: Principal Axis of a 2D object whose centriod is at the origin. Right: rotated object so principal axis is aligned with X axis.

So we can calculate the moments of order 2 for a rotated point set by: $\sum\sum(-xsin\theta + ycos\theta)^2$

These are the moments of order 2 about the X axis for the rotated point set. The term $(-xsin\theta + ycos\theta)$ is the vertical distance from the X axis (i.e. the Y coordinate value) of the transformed point set.

Now, find the value of θ that minimizes that measure. We do this by differentiating with respect to θ, and setting the resulting measure equal to zero:

$$\sum\sum 2(-xsin\theta + ycos\theta)(-xcos\theta - ysin\theta) \ = 0$$

$$2\sum\sum(x^2sin\theta cos\theta + xysin^2\theta - xycos^2\theta - y^2sin\theta cos\theta) \ = 0$$

$$2sin\theta cos\theta\sum\sum x^2 + 2(sin^2\theta - cos^2\theta)\sum\sum xy - 2cos\theta sin\theta\sum\sum y^2 \ = 0$$

Using the definition of discrete moments M_{ij}:

$$2sin\theta cos\theta\overline{M_{20}} + 2(sin^2\theta - cos^2\theta)\overline{M_{11}} - 2cos\theta sin\theta\overline{M_{02}} \ = 0$$

where $\overline{M_{ij}}$ refers to *Central Moments*, moments where the centroid is translated to the origin.

Since $sin2\theta \ = \ 2sin\theta cos\theta$ and $cos2\theta \ = \ cos^2\theta - sin^2\theta$, we can substitute to get:

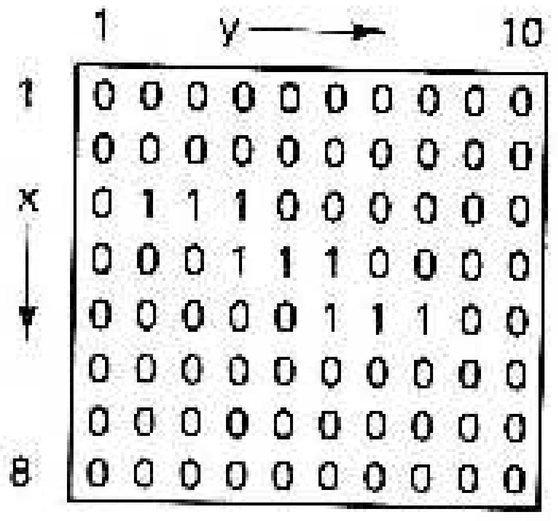
$$sin2\theta\overline{M_{20}} - 2cos2\theta\overline{M_{11}} - sin2\theta\overline{M_{02}} \ = 0$$

and

$$\frac{sin2\theta}{cos2\theta} \ = \ \frac{2\overline{M_{11}}}{\overline{M_{20}} - \overline{M_{02}}}$$

The principal angle is: $2\theta = atan2(2\overline{M_{11}}, \overline{M_{20}} - \overline{M_{02}})$

5.2 Example: Moments



	1		y ——►							10
1	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
x	0	1	1	1	0	0	0	0	0	0
│	0	0	0	1	1	1	0	0	0	0
▼	0	0	0	0	0	1	1	1	0	0
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0

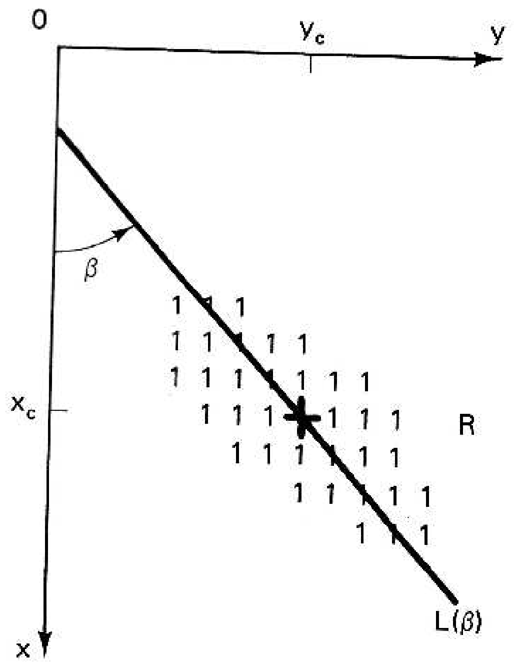


Figure 9: Left: Image used in example below. Right: Idea of Principal angle computation: rotate blob by $-\beta$ so its coincident with X axis

From the image above, we have a region R denoted by values = 1. We can calculate the discrete moments for the region as:

$$M_{ij} = \sum \sum x^i \, y^j \, f(x, y)$$

and $m_{00} = 9$, $m_{01} = 45$, $m_{10} = 36$, $m_{11} = 192$, $m_{02} = 255$, $m_{20} = 150$.

We can create *central moments* by finding the centroid and translating the region so that the origin is the centroid, (x_c, y_c) :

$$Area \ = \ m_{00} \ = \ 9 \quad ; \quad x_c \ = \ \frac{m_{10}}{m_{00}} \ = \ 4 \quad ; \quad y_c \ = \ \frac{m_{01}}{m_{00}} \ = \ 5$$

Finally, the principal angle for the image on the left is computed as $\beta = \frac{atan2(2\overline{M_{11}}, \overline{M_{20}} - \overline{M_{02}}}{2}$:

$$\beta \ = \ \frac{atan2(24, -24)}{2} \ = \ \frac{135}{2} \ = \ 67.5°$$

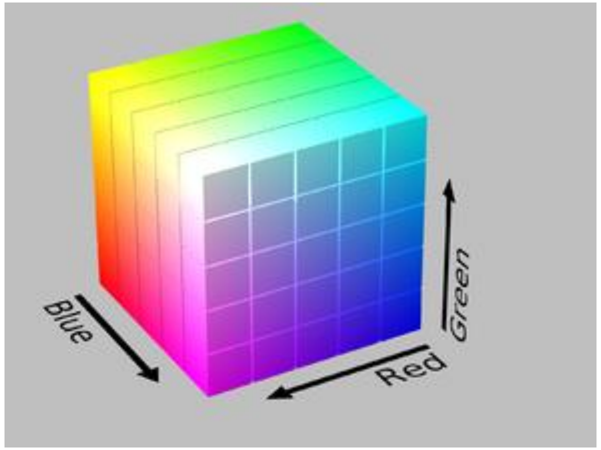
Color Tracking

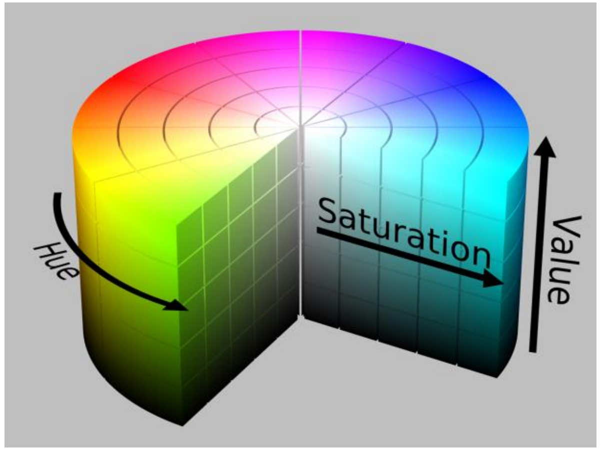
A Quick Overview

- Color Representations
- Choosing a Color to Track
- How to Find the Target

RGB vs HSV

- RGB is very sensitive to brightness
- HSV (Hue, Saturation, Value) is less sensitive
- [Color Space Visualizer: http://colorizer.org/](http://colorizer.org/)

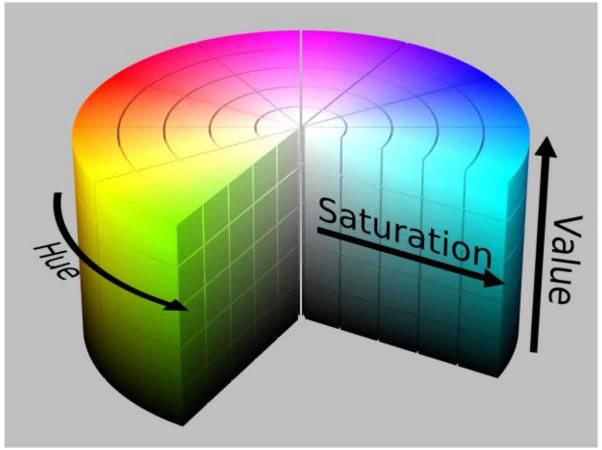




HSV Color Space

- **Hue:** expressed as a number from 0 to 179 when using OpenCV image operations
- **Saturation:** How "pure" the color is. The closer to 0, the more grey the color looks. Range 0-255
- **Value:** (or Brightness) works in conjunction with saturation and describes the brightness or intensity of the color from 0 to 255.
- Color conversion: `cv2.cvtColor(input_image, flag)`

Where `flag` determines the type of conversion.
For BGR→Gray, flag is `cv2.COLOR_BGR2GRAY`
For BGR→HSV, flag is `cv2.COLOR_BGR2HSV`
Note:
For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. You will have to experiment to find the right settings for your lab.

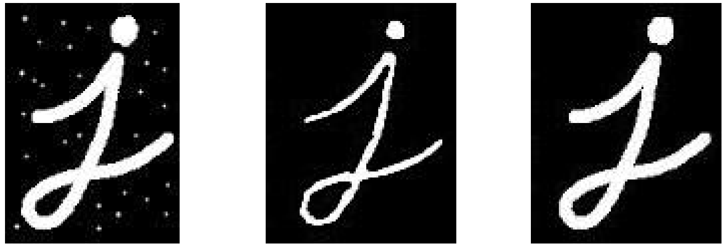


Convert to HSV, Find HSV values

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while (1):
     # Take each frame
     _, frame = cap.read()
    # Convert BGR to HSV
   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
-----------------------------------------------------------------------------------------------
#print out the HSV values for color green
   green = np.uint8([[[0,255,0 ]]])
    hsv_green = cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
    print  hsv_green
    [[[ 60 255 255]]]
```

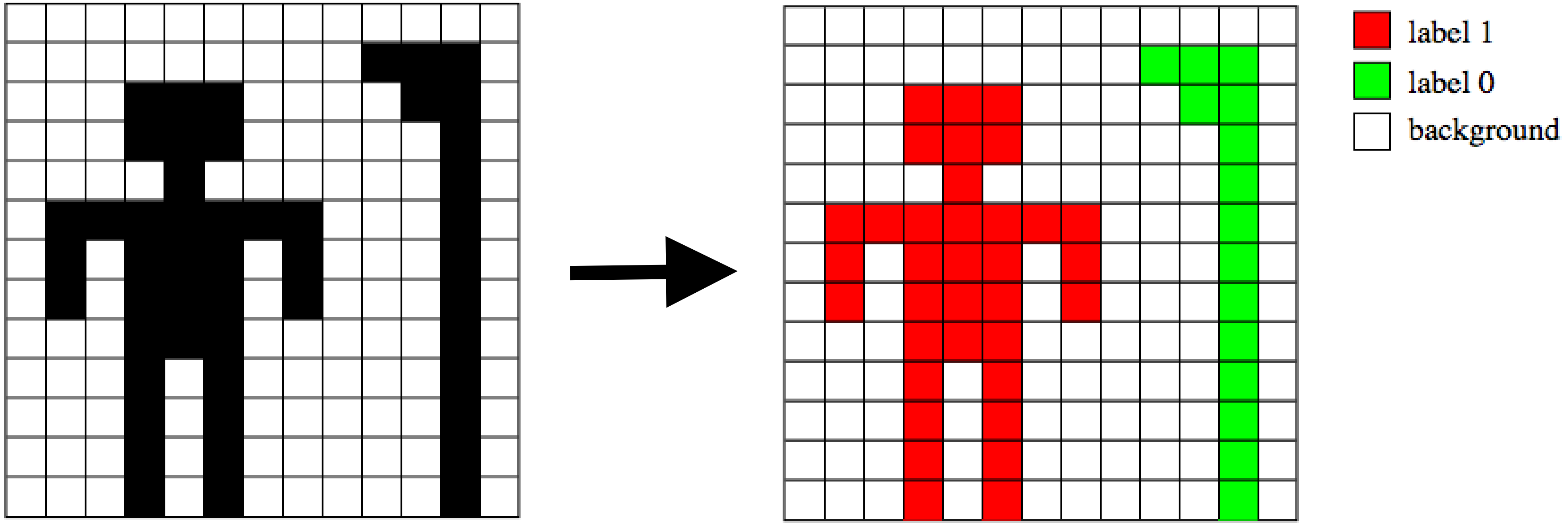
Use Morphology to “clean up” image

erode:
The value of the output pixel is the *minimum* value of all the pixels in the input pixel's neighborhood
dilate:
The value of the output pixel is the *maximum* value of all the pixels in the input pixel's neighborhood



Tutorial: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

Connected-Component Labeling (a.k.a. Blob Extraction)



Finding the Target-Pseudocode

```
def get_target(hsv_image):

#get pixels within threshold of target patch
        masked_image = mask_image(hsv_image, h_thresh, s_thresh, v_thresh)

        #morphologically erode and dilate the image
         eroded_image = erosion_filter(masked_image)
        cleaned_image = dilate_filter(erodeed_image)

        #find the largest connected component (largest blob)
        big_blob = get_largest_blob(cleaned_image)

# Compute centroid and area of big_blob to move the robot forward, back, left,, right

       centroid = get_centroid(big_blob)
       area = get_area(big_blob)

return centroid, area
```

Example

- www.cs.columbia.edu/~allen/F19/NOTES/tracker_with_video_output_clean.py
- www.cs.columbia.edu/~allen/F19/NOTES/green_tracker_output.avi (video)