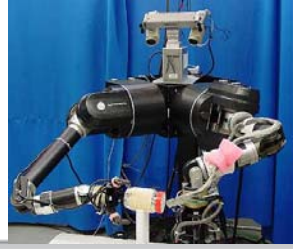
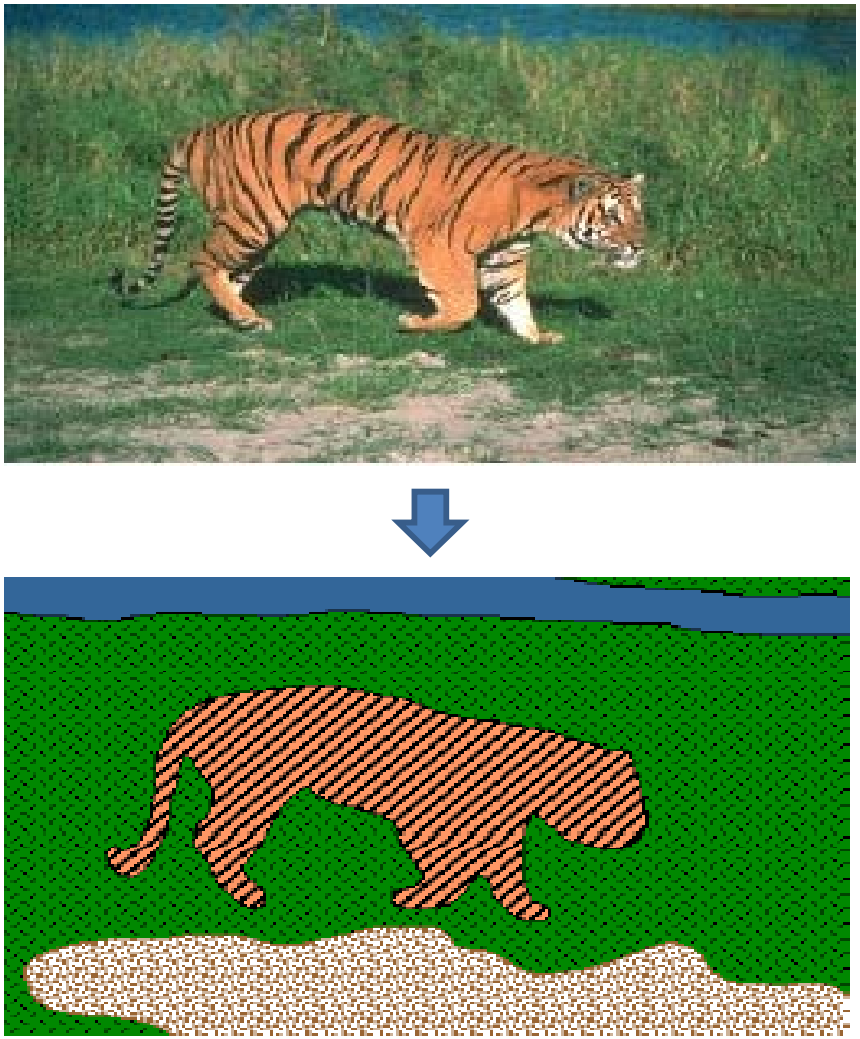
Einführung in Visual Computing

Unit 15: Image Segmentation



http://www.caa.tuwien.ac.at/cvl/teaching/sommersemester/evc

- Content:
  - Introduction
  - Greylevel Thresholding
  - Clustering
  - Relaxation Labelling
  - Region Growing
  - Split and Merge



Introduction to Image Segmentation

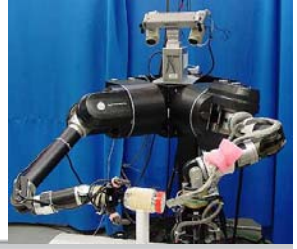
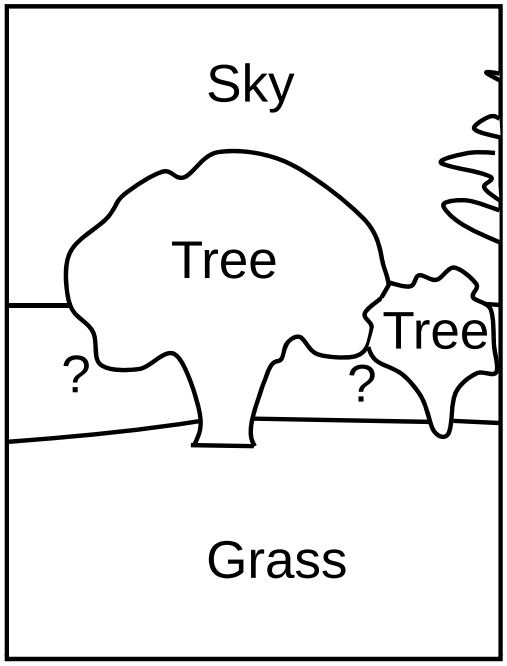
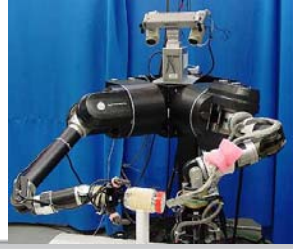


Image Segmentation



Introduction to Image Segmentation

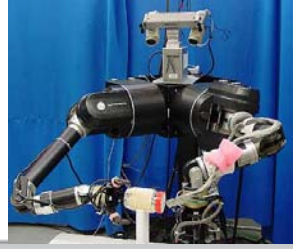


The **shape** of an object can be described in terms of:

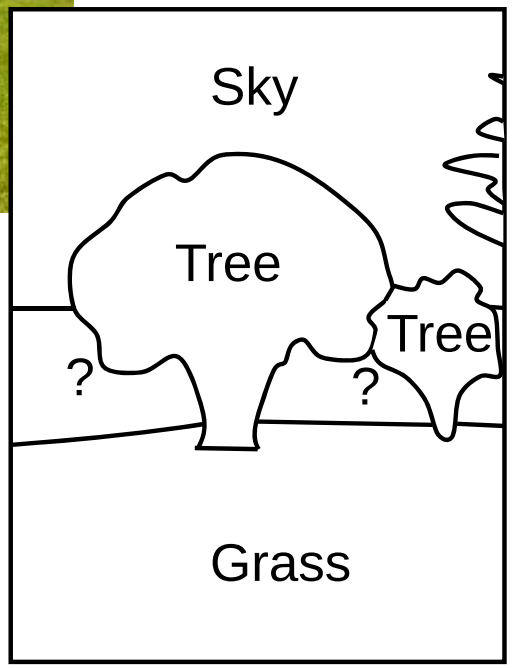
- Its **boundary** – requires image **edge detection**
- The **region** it occupies – requires **image segmentation** in homogeneous regions, Image regions generally have homogeneous characteristics (e.g. intensity, texture)

- The **purpose** of image segmentation is to partition an image into **meaningful regions** with respect to a particular application
- The segmentation is based on **measurements** taken from the image like
  - Greylevel, Color, Texture
  - Depth
  - Motion

Finding Objects in Images

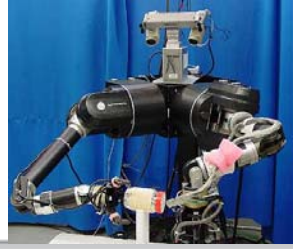


- To do this we need to divide the image into two parts
  - the object of interest (the **foreground**)
  - everything else (the **background**)



- The definition of foreground and background depends on the task at hand

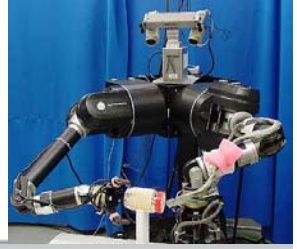
Introduction to Image Segmentation



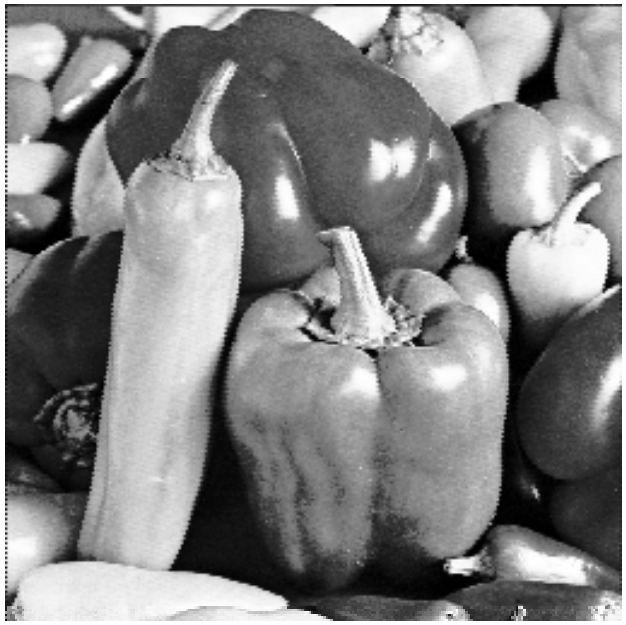
- Usually image segmentation is an **initial step** in a series of processes aimed at **image understanding**

- Applications of image segmentation include
  - **Identifying** objects in a scene for object-based **measurements** such as size and shape
  - Identifying objects in a moving scene for *object-based video* ***compression*** *(MPEG4)*
  - Identifying objects which are at different distances from a sensor using **depth measurements** from a laser range finder enabling **path planning** for a mobile robot

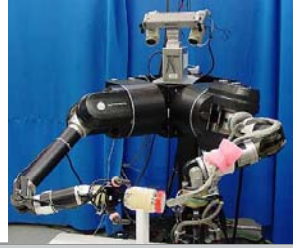
Introduction to Image Segmentation



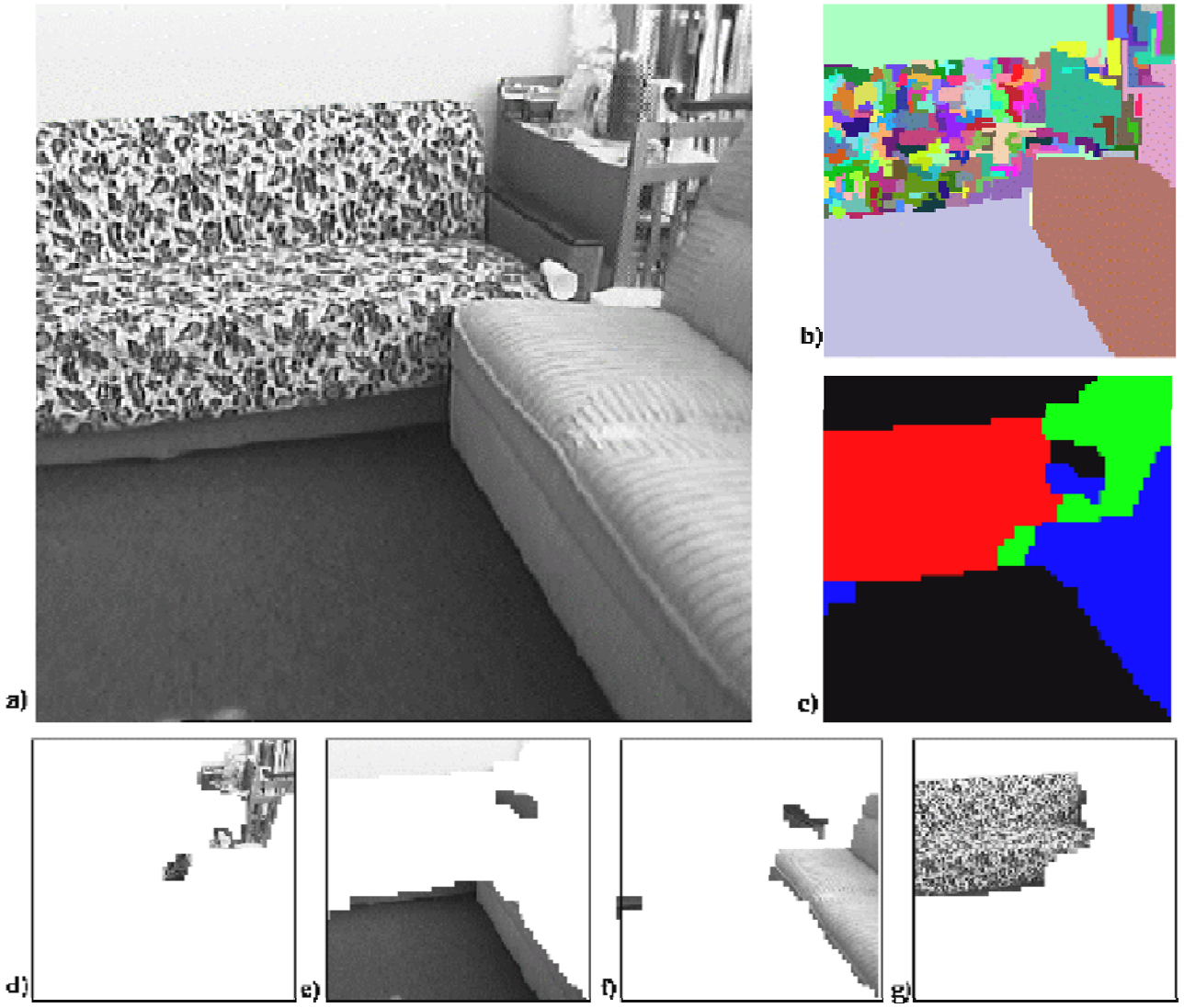
- Example 1
  - Segmentation based on **greyscale**
  - Very simple ‘model’ of greyscale leads to inaccuracies in object labelling



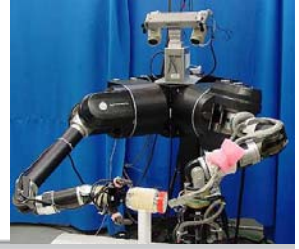
Introduction to Image Segmentation



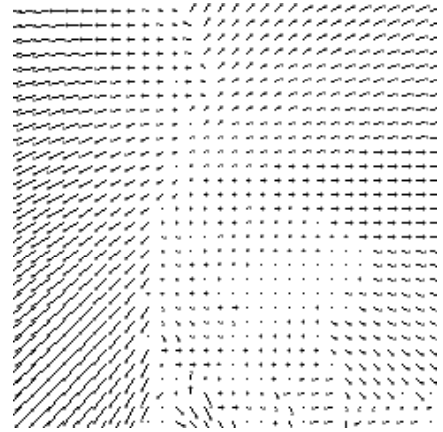
- Example 2
  - Segmentation based on **texture**
  - Enables object surfaces with varying patterns of grey to be segmented



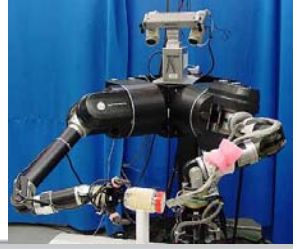
Introduction to Image Segmentation



- Example 3
  - Segmentation based on **motion**
  - The main difficulty of motion segmentation is that an intermediate step is required to (either implicitly or explicitly) estimate an *optical flow field*
  - The segmentation must be based on this estimate and not, in general, the true flow

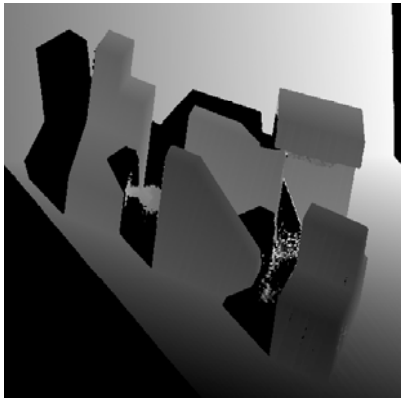


Introduction to Image Segmentation



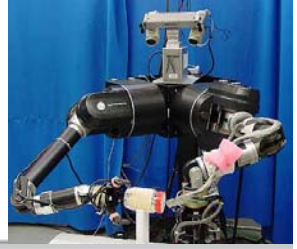
- Example 4
  - Segmentation based on **depth**
  - This example shows a range image, obtained with a laser range finder
  - A segmentation based on the range (the object distance from the sensor) is useful in guiding mobile robots

Range image



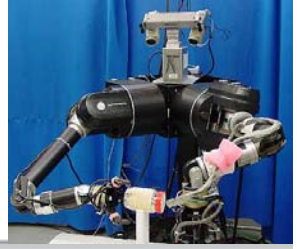
Segmented image

Image Segmentation



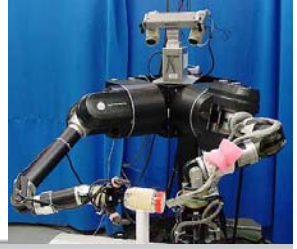
- In analysis of objects in images it is essential to distinguish between **objects of interest** and "the rest" = **background**.
- Techniques that are used to find objects of interest are referred to as **segmentation** techniques - segmenting the foreground from background.
- Image segmentation describes the division of the image in **homogenous segments** (no abrupt intensity changes within segments)
- **Edge detection** may (not necessarily) produce a segmentation

What do we mean by “Labeling” an Image?



- When we say we ”extract “ an object in an image , we mean that we **identify the pixels** that make it up.
- To express this information , we create an array of the same size as the original image and we give to **each pixel** a **label.**
- All pixels that make up the **object** are given the **same label** . The label is usually a number, but it could be anything: a letter, or color.
- Often **label images** are also referred to as classified images as they indicate the **class** to which **each pixel** belongs.

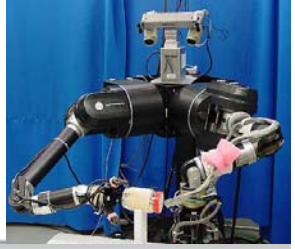
How can we divide an Image into Uniform Regions ?



- Segmentation techniques can be classified as either contextual or non-contextual .
- **Non-contextual techniques ignore** the relationships that exist between **features** in an image; pixels are simply **grouped** together on the basis of some **global attribute**, such as grey level.
- **Contextual techniques**, additionally **exploit** the relationships between image **features**. Thus, a contextual technique might group together pixels that have similar grey levels and are close to one another .

Greylevel Threshholding

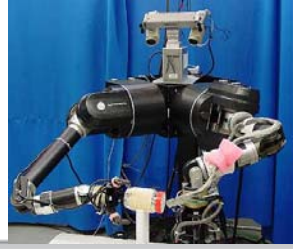
Greylevel Histogram-based Segmentation

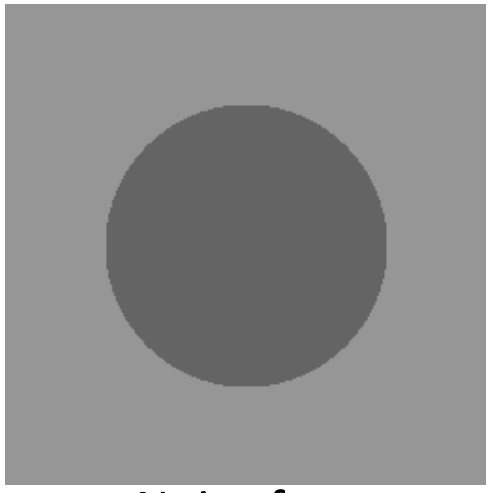


- First, we will look at two very simple **non-contextual** image segmentation techniques that are based on the **greylevel histogram** of an image:
  - Thresholding
  - Clustering

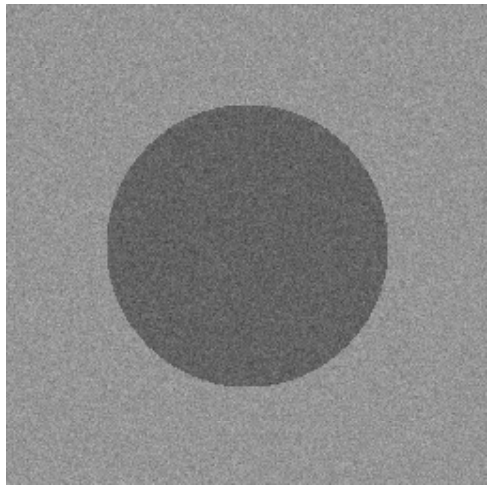
- We will use a very simple object-background test image
  - We will consider a zero, low and high noise image

Greylevel Histogram-based Segmentation

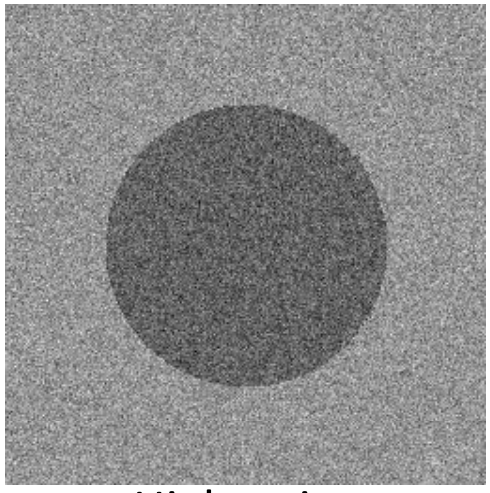




Noise free



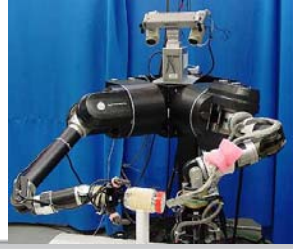
Low noise

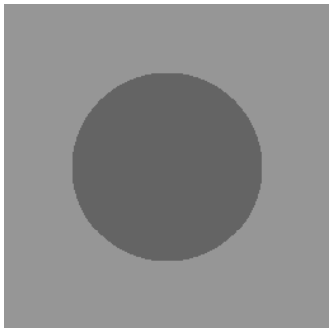


High noise

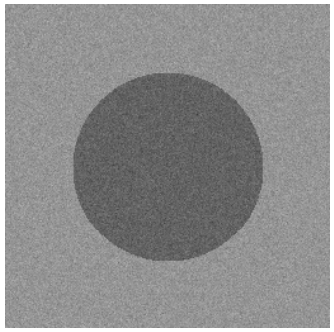
- How do we characterise low noise and high noise?

Greylevel Histogram-based Segmentation

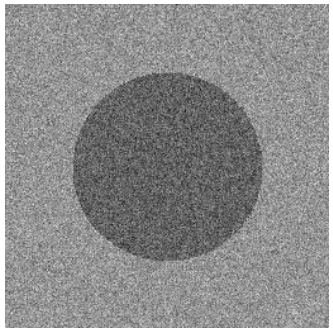




Noise free



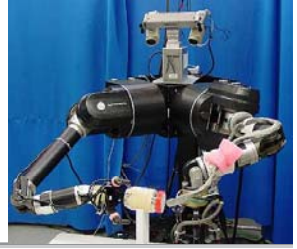
Low noise

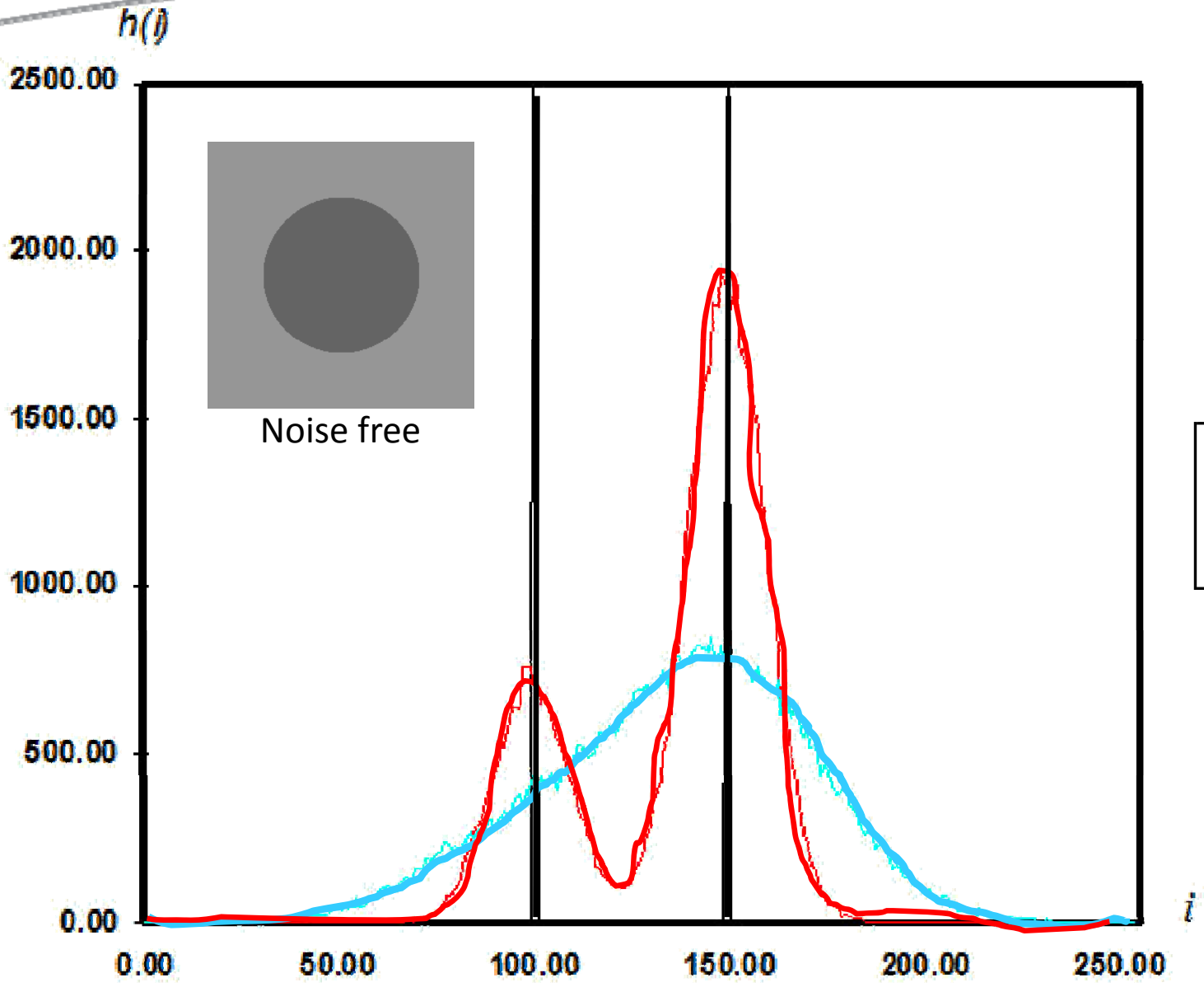


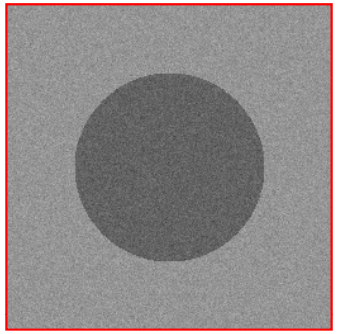
High noise

- We can consider the histograms of our images
  - For the noise free image, its simply **two spikes** at i=100, i=150
  - For the low noise image, there are **two clear peaks** centred on i=100, i=150
  - For the high noise image, there is a **single peak** – two greylevel populations corresponding to object and background have merged

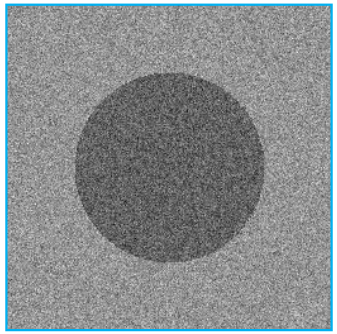
Greylevel Histogram-based Segmentation





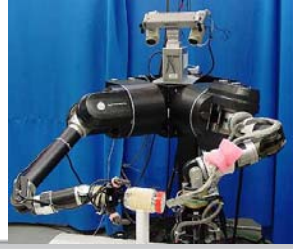


Low noise



High noise

Greylevel Histogram-based Segmentation

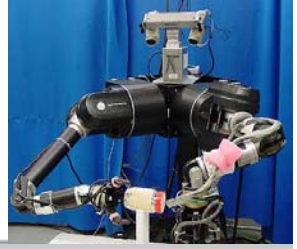


- We can define the input image ***Signal-to-Noise ratio*** in terms of the **mean greylevel** value of the **object** pixels and **background** pixels and the additive **noise standard deviation**

$$S / N = \frac{|\mu_b - \mu_o|}{\sigma}$$

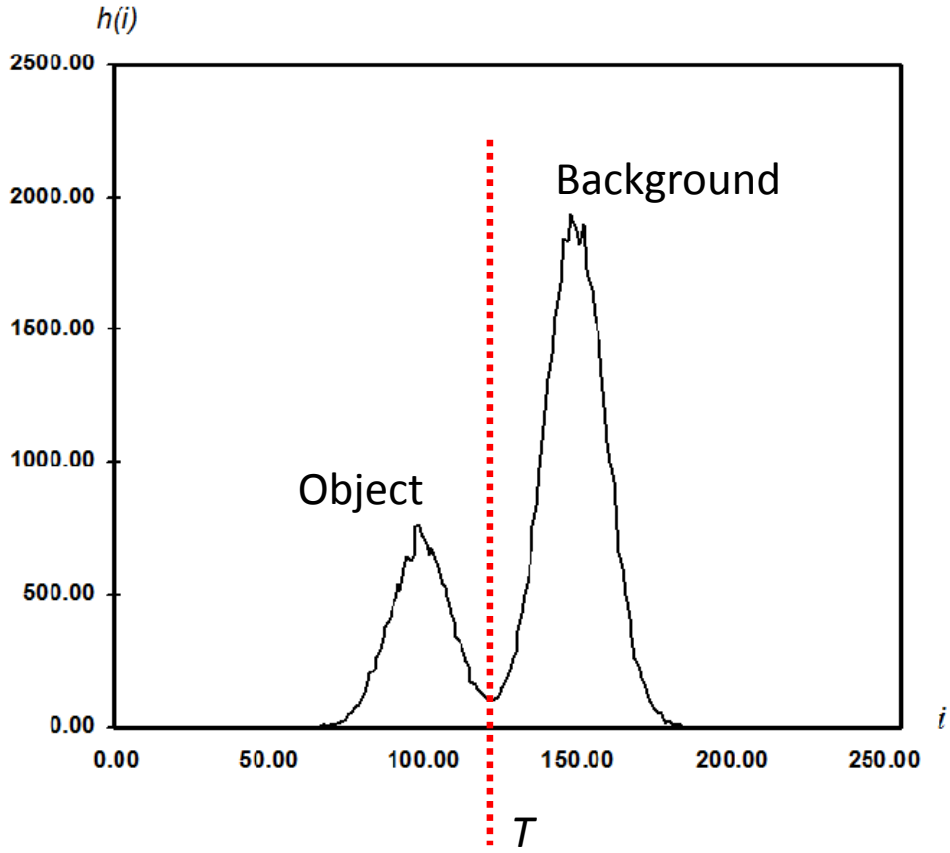
- For our test images :
  - *S/N* (noise free) = ∞
  - *S/N* (low noise) = 5
  - *S/N* (high noise) = 2

Greylevel Thresholding

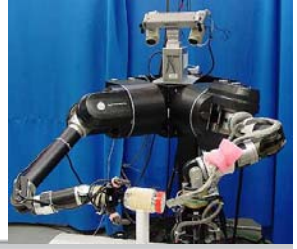


- We can easily understand segmentation based on thresholding by looking at the **histogram** of the low noise object/background image
  - There is a **clear 'valley'** between to **two peaks**
- We can define the **greylevel thresholding** algorithm as follows:
  - **If** the greylevel of pixel ***p*** **<= *T*** **then** pixel ***p*** **is** an **object** pixel

  else
  - Pixel *p* is a **background** pixel



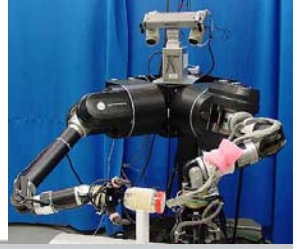
Foundation



- If **two dominant modes** characterize the image histogram, it is called a **bimodal histogram**. Only **one threshold** is enough for partitioning the image.

- If for example an image is composed of two types of light objects on a dark background, **three or more** dominant modes characterize the image histogram.

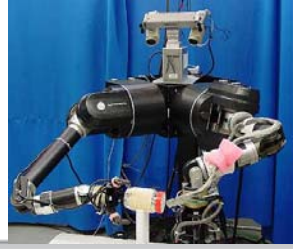
Foundation (contd.)



- In such a case the histogram has to be partitioned by **multiple thresholds**.

- Multilevel thresholding classifies a *point (x,y)* as belonging
  - to one object class if $T_1 < f(x,y) <= T_2$,
  - to the other object class if $f(x,y) > T_2$,
  - and to the background if $f(x,y) <= T_1$.

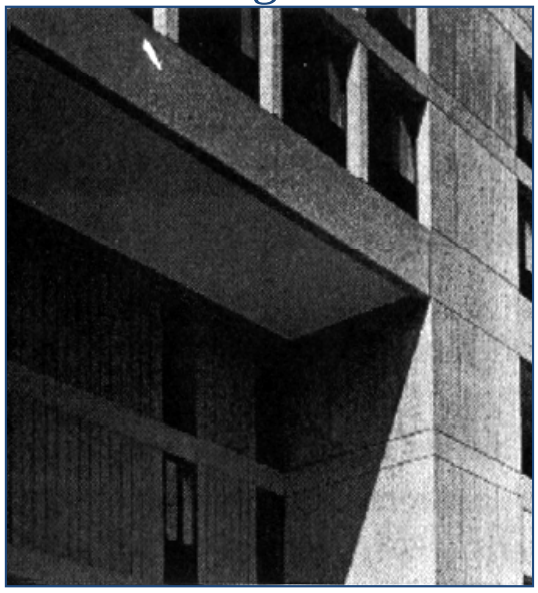
Pixel Classification by Threshold



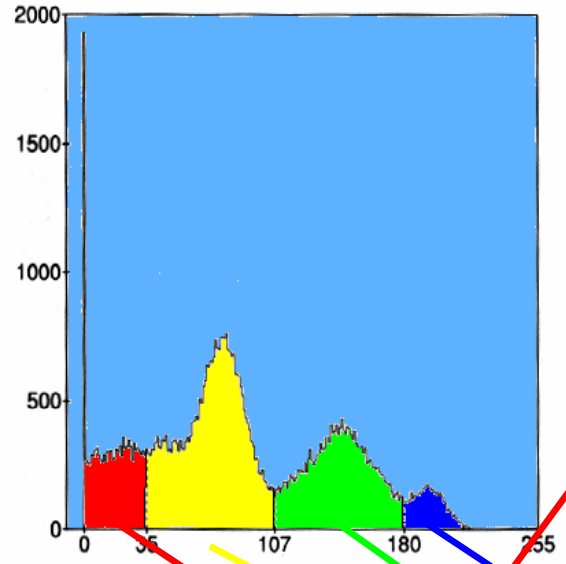
- Histogramm
  - **Problem**: Connected image regions do not always have the same intensity

    => missing **location relation** of pixels

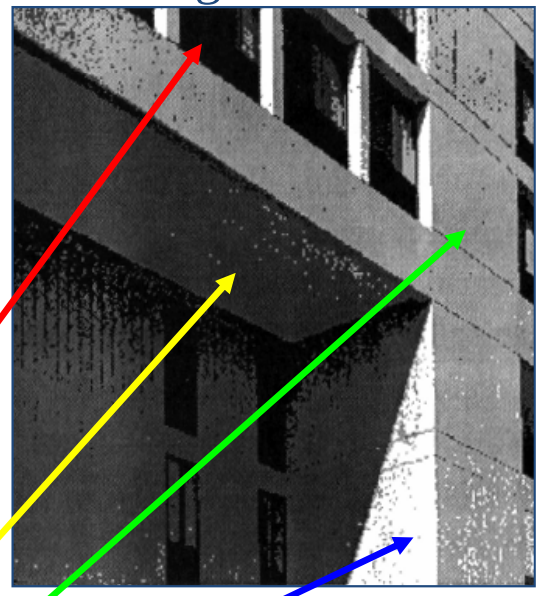
Original



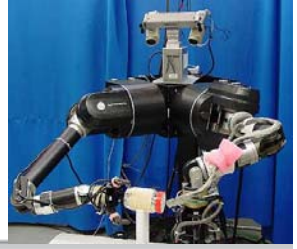
Histogram



Segmentation



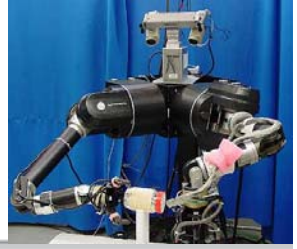
Greylevel Thresholding



- This simple threshold test begs the obvious question how do we determine the threshold ?

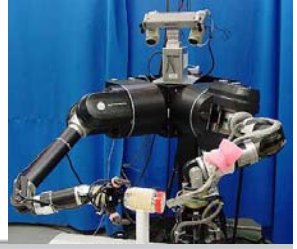
- Many approaches possible
  - Interactive threshold
  - Global threshold
  - Local threshold
  - Minimisation method
  - .....

Global Thresholding



- Basic Global Thresholding:
  1. Select an **initial estimate** for T
  2. **Segment** the image using T. This will produce **two groups** of pixels. G_1 consisting of all pixels with gray level values $>T$ and G_2 consisting of pixels with values $<=T$.
  3. Compute the **average gray level** values $mean_1$ and $mean_2$ for the pixels in regions G_1 and G_2.
  4. Compute a **new threshold** value $T=\frac{1}{2}(mean_1 + mean_2)$
  5. **Repeat** steps 2 through 4 until difference in T in successive iterations is **smaller than** a predefined parameter T_0.

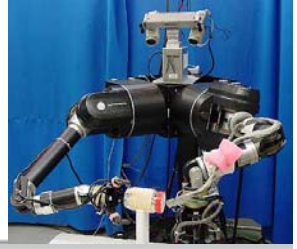
Local Thresholding



- A complex thresholding algorithm is to use a **spatially varying** threshold. This approach is very useful to compensate for the effects of non –uniform illumination. If *T* depends on coordinates *x* and *y*, this referred to as **Dynamic**, **Adaptive** or **Local Thresholding**.

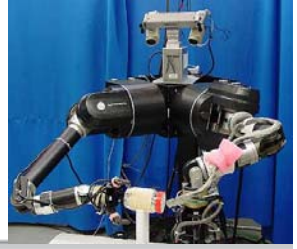
- A technique which provides good results is to use **edge points** when creating the grey level histogram .

Local Thresholding - How it Works



- There are two main approaches to find the threshold:
  - the **Chow and Kaneko** approach and
  - **local thresholding**.
- The assumption behind both methods is that **smaller image regions** are **more likely** to have approximately **uniform illumination**, thus being more suitable for thresholding.
- Chow and Kaneko divide an image into an array of overlapping **subimages** and then find the **optimum threshold** for each subimage by investigating its **histogram**. The threshold for each single pixel is found by interpolating the results of the subimages.

Local Thresholding



- Finding the local threshold is to **statistically examine** the intensity values of the **local neighborhood** of each pixel
- The **statistic** which is most appropriate **depends** largely **on** the **input image**. Simple and fast functions include:

1. The **mean** of the local intensity distribution

$$T = mean$$

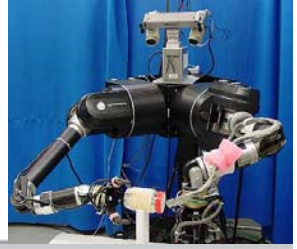
2. The **median** value

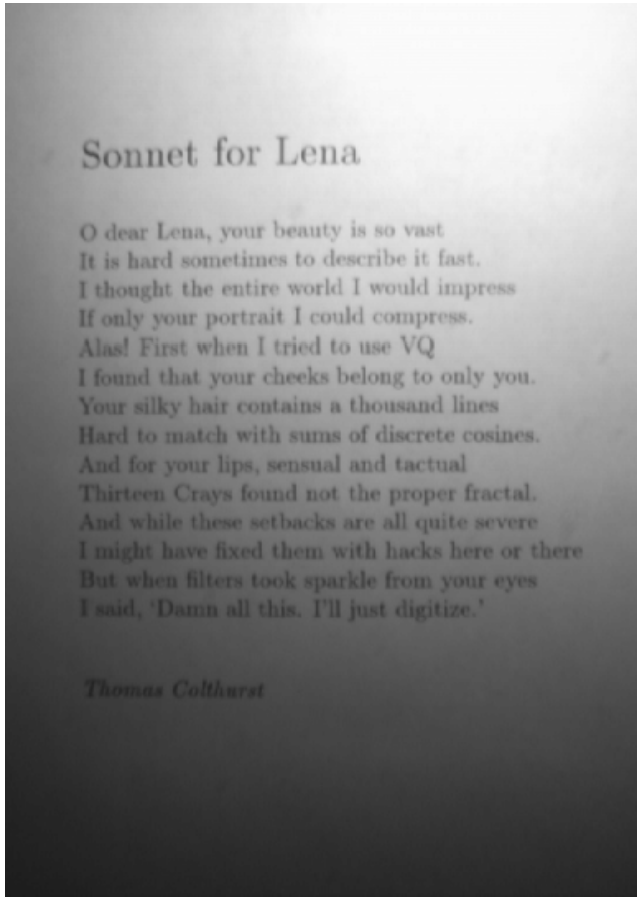
$$T = median$$

3. The mean of the **minimum** and **maximum** values,

$$T = \frac{\min + \max}{2}$$

Local Thresholding Example





Source Image

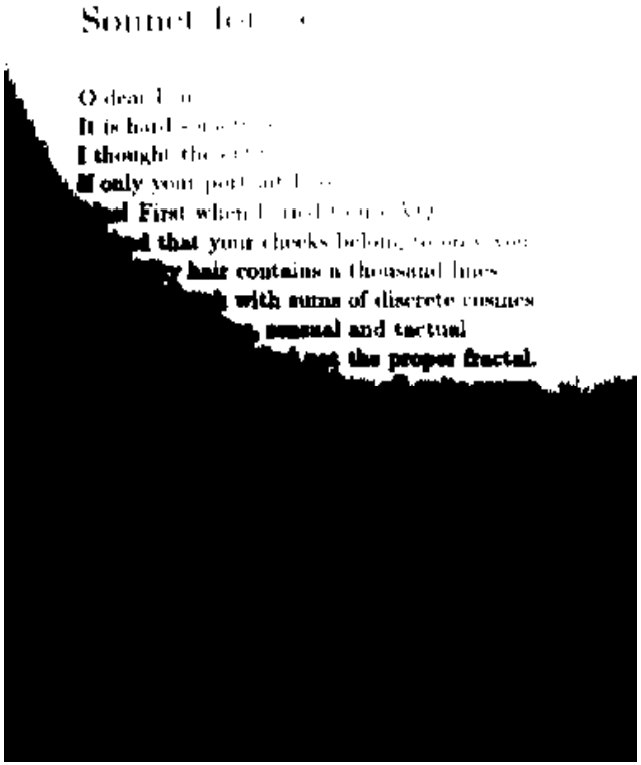
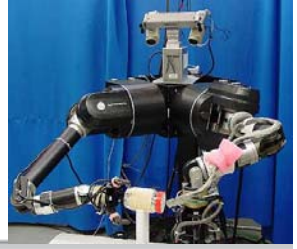


Image contains a strong illumination gradient, **global thresholding** produces a very poor result

Local Thresholding



- Improve: If threshold employed is not the mean, but (mean-**C**), where **C** is a constant

- Using this statistic, all pixels which exist in a uniform neighborhood (e.g. along the margins) are set to background

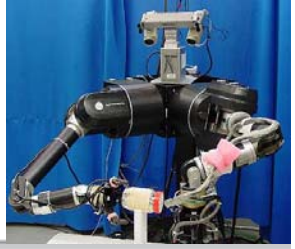
Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

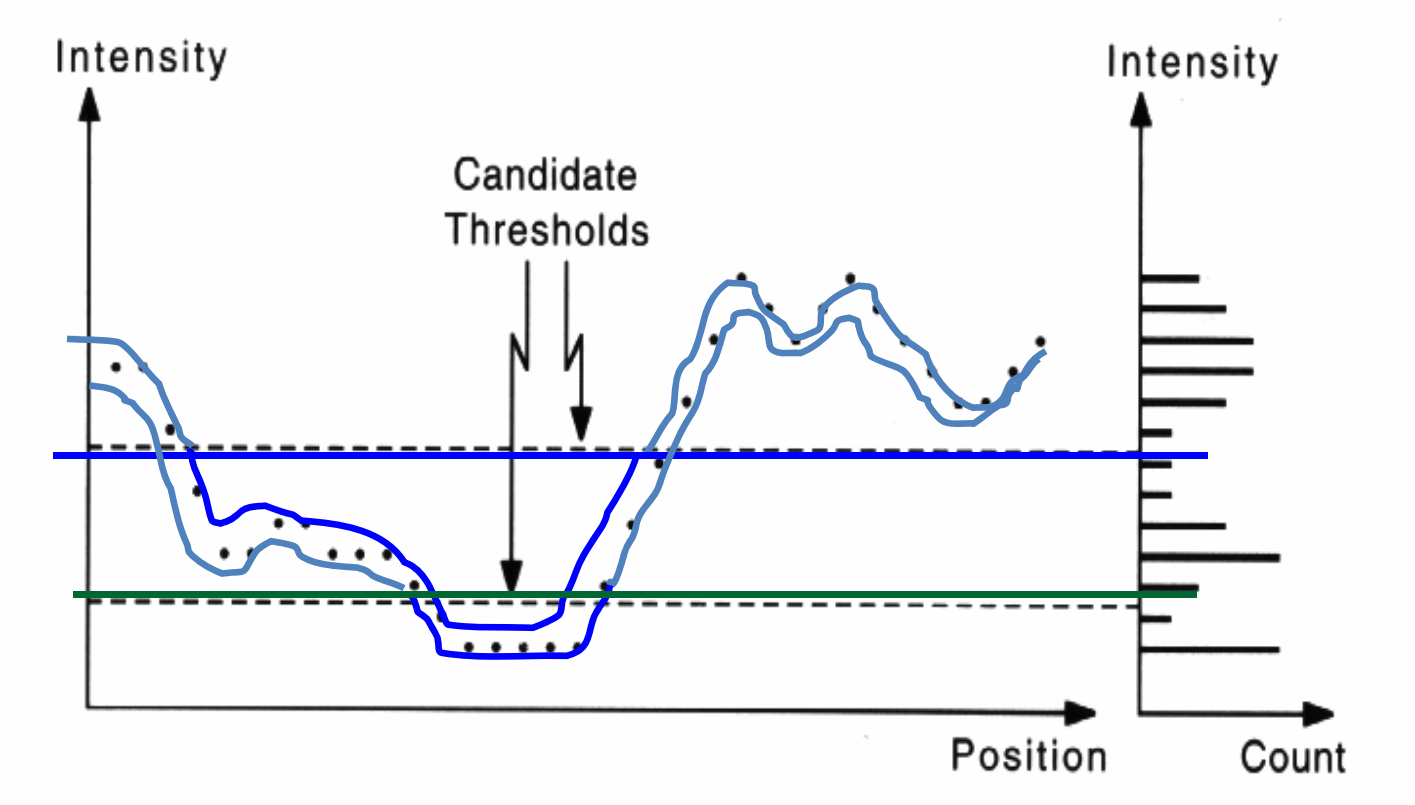
The result for a 7×7 neighborhood and C=7

Pixel Classification by Threshold

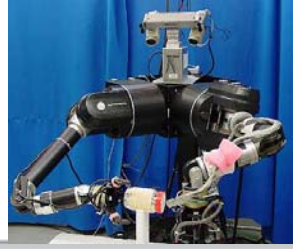


- Interactive, imprecise threshold

  => Threshold influences segmentation



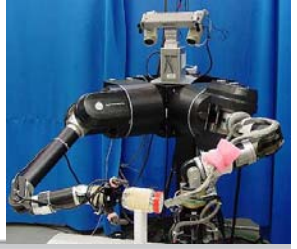
Thresholding Example



- Let‘s take a 7x7 image

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

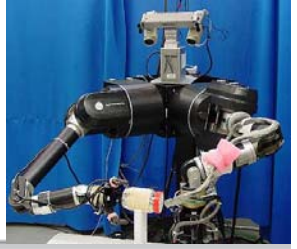
Thresholding Example



- And a threshold T= 7 (mean)

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

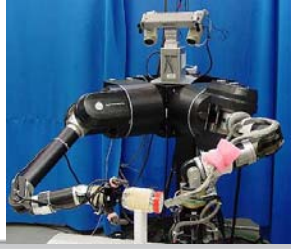
Thresholding Example



- Alternatively T= 6 (Median)

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

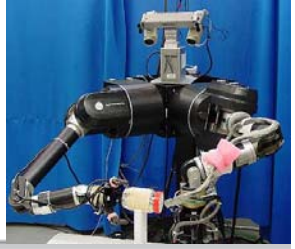
Thresholding Example



- Alternatively T= 12 ((min+max)/2)

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

Thresholding Example

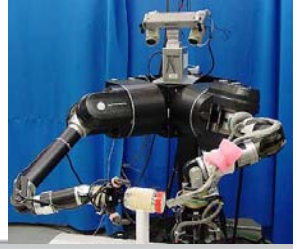


- Or T= 10

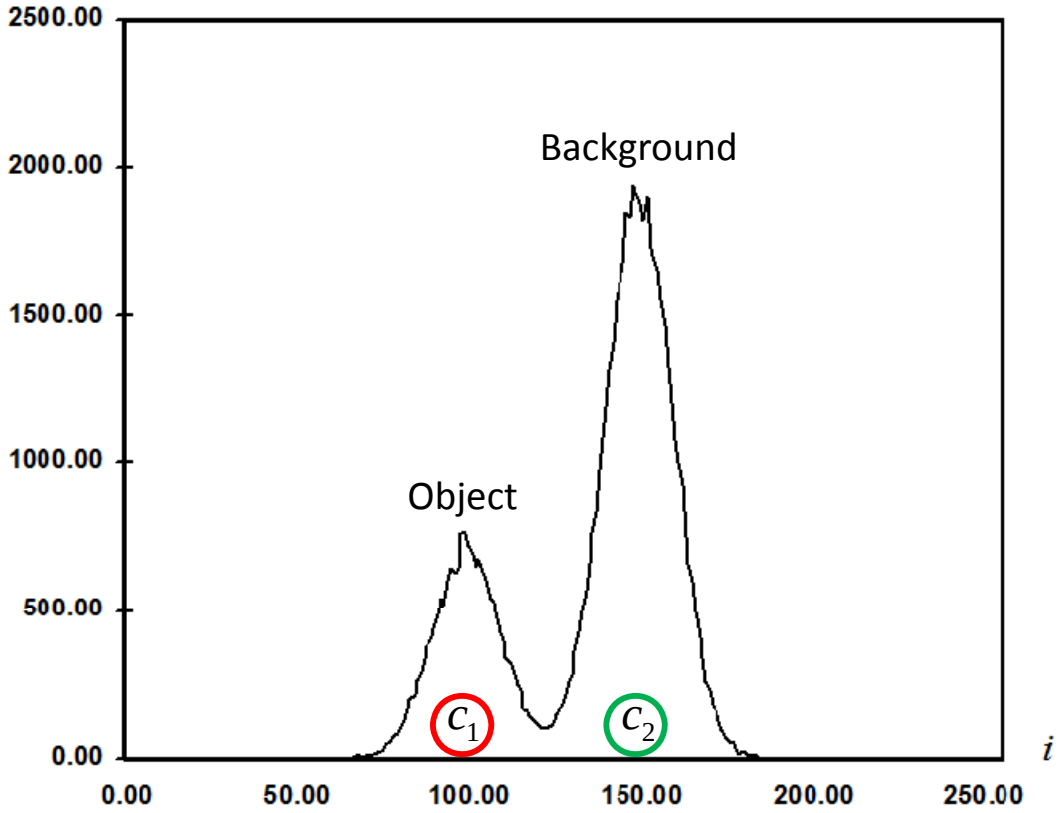
3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

Clustering

Greylevel Clustering

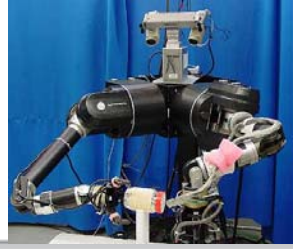


- Consider an idealized object/background histogram



- Clustering tries to separate the histogram into 2 groups defined by two cluster centres c_1 and c_2

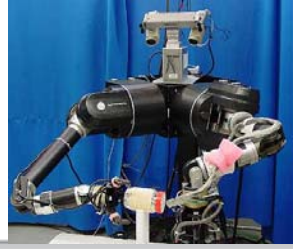
Greylevel Clustering



- A ***nearest neighbour*** clustering algorithm allows us perform a greylevel segmentation using clustering
  - A simple case of a more general and widely used ***K-means*** clustering
  - A simple iterative algorithm which has known convergence properties
- Given a set of greylevels: $\{g(1), g(2) \ldots\ldots g(N)\}$
- We can partition this set into two groups

$\{g_1(1), g_1(2) \ldots\ldots g_1(N_1)\}$ **and** $\{g_2(1), g_2(2) \ldots\ldots g_2(N_2)\}$

Greylevel Clustering



- Compute the local means of each group

$$c_1 = \frac{1}{N_1}\sum_{i=1}^{N_1} g_1(i) \quad \text{and} \quad c_2 = \frac{1}{N_2}\sum_{i=1}^{N_2} g_2(i)$$

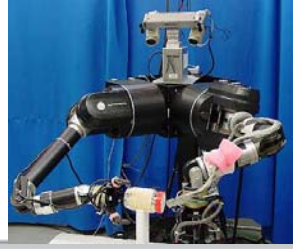
- Re-define the new groupings:

$$\left|g_1(k) - c_1\right| < \left|g_1(k) - c_2\right| \quad k = 1..N_1$$

$$\left|g_2(k) - c_2\right| < \left|g_2(k) - c_1\right| \quad k = 1..N_2$$

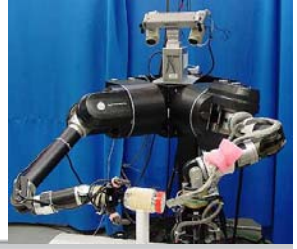
- In other words **all grey levels** in *Set 1* are **nearer to cluster center** c_1 and all grey levels in *Set 2* are nearer to cluster center c_2

Greylevel Clustering



- But, we have a ***chicken and egg*** situation
  - The problem with the above definition is that each group mean is defined in terms of the partitions and vice versa
  - The solution is to define an iterative algorithm and worry about the convergence of the algorithm later

Greylevel Clustering



- The iterative algorithm is as follows

Initialize the label of each pixel **randomly**

Repeat

c_1= mean of pixels assigned to object label

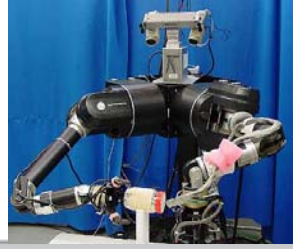
c_2= mean of pixels assigned to background label

Compute partition $\{g_1(1), g_1(2) \ldots\ldots g_1(N_1)\}$

Compute partition $\{g_2(1), g_2(2) \ldots\ldots g_2(N_2)\}$

Until **none** pixel **labelling changes**

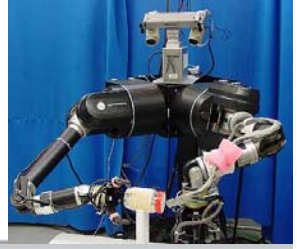
Greylevel Clustering

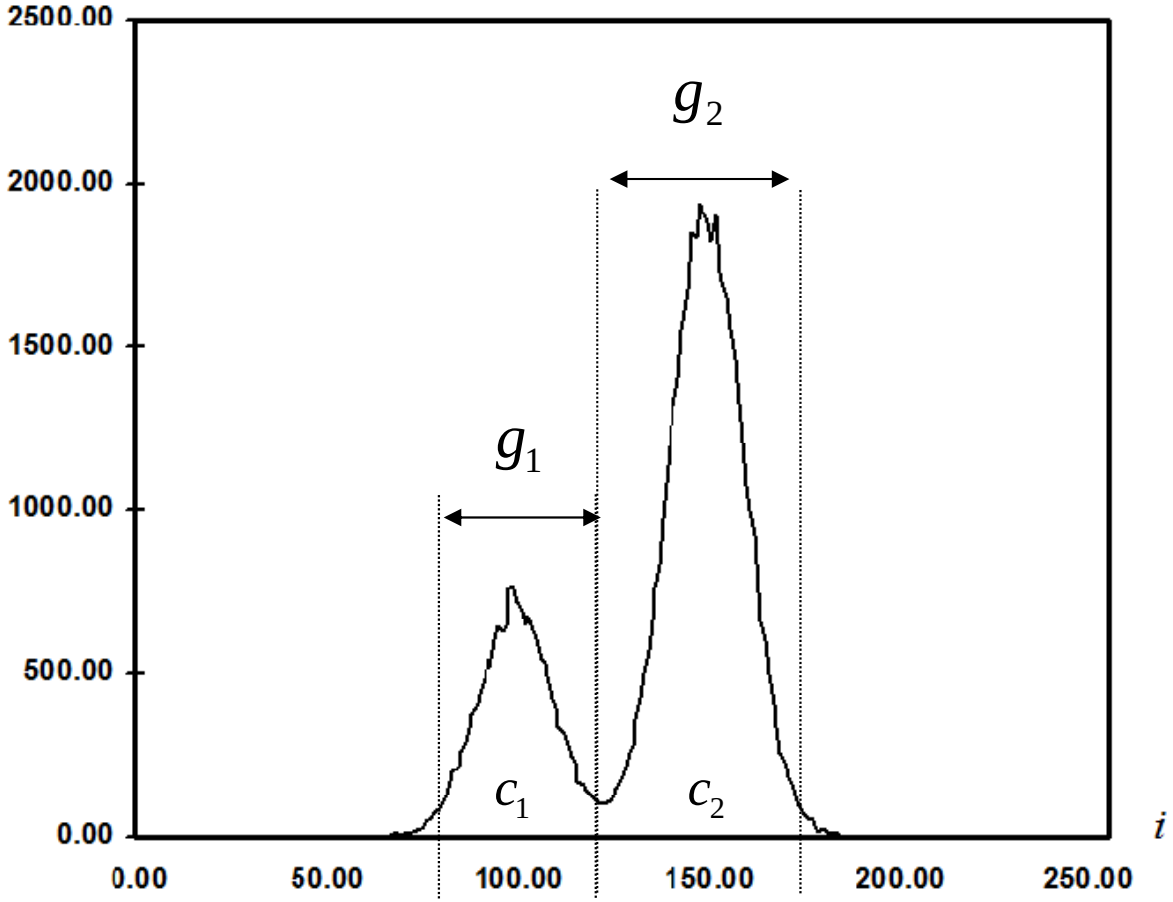


- Two questions to answer
  - Does this algorithm **converge**?
  - If so, **to what** does it converge?

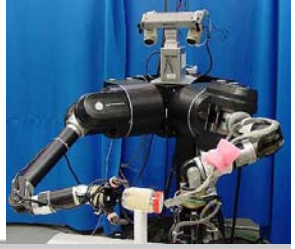
- We can show that the algorithm is guaranteed to converge and also that it converges to a sensible result

Greylevel Clustering





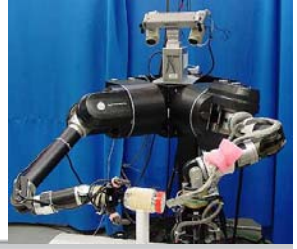
Greylevel Clustering



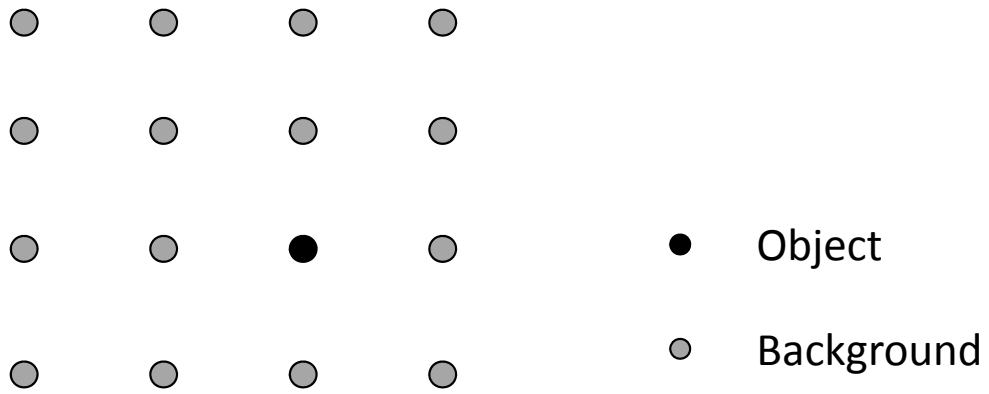
- Clustering
  - Finds groups of pixels with **similar properties**
  - Does **not guarantee** that these groups form **continuous areas** in the image
  - Even if it does, **edges** of these areas tend to be **uneven**

Relaxation Labelling

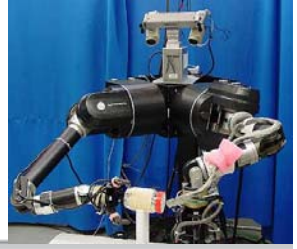
Relaxation Labelling



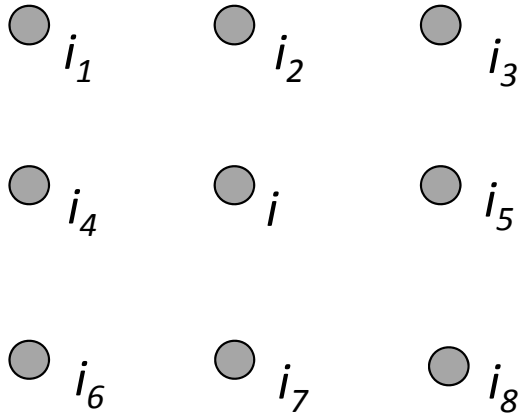
- All of segmentation algorithms considered so far are based on histogram of the image
  - This **ignores** the greylevels of each pixels' **neighbours** which will **strongly influence** the **classification** of each pixel
  - Objects are usually represented by a **spatially contiguous** set of pixels
- Trivial example of a likely pixel miss-classification:



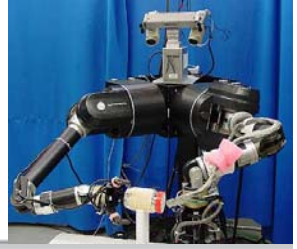
Relaxation Labelling: Probabilities



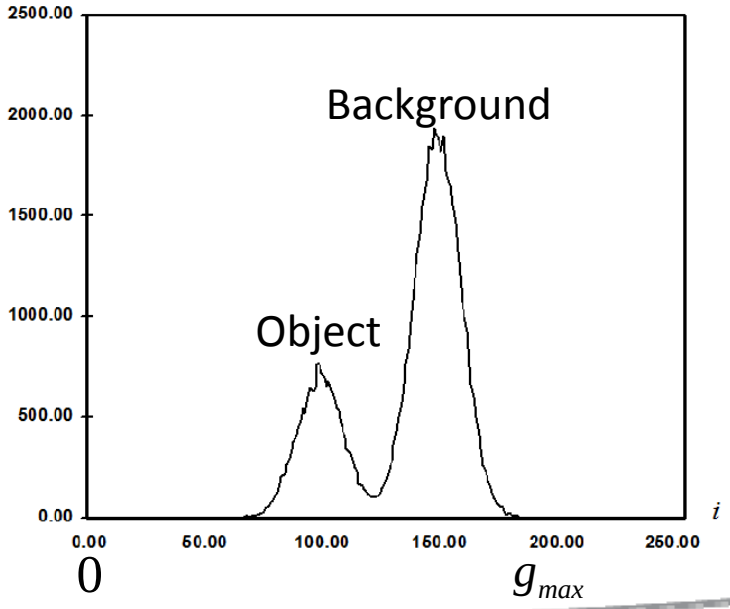
- ***Relaxation labelling*** is a general technique in computer vision which is able to incorporate **constraints** (such as spatial continuity) into image labelling problems
- Assume a simple object/background image
  - *p(i)* is the probability that pixel *i* is a background pixel
  - *(1- p(i))* is the probability that pixel *i* is a object pixel
- Define the 8-neighbourhood of pixel *i* as $\{i_1, i_2, \ldots . i_8\}$



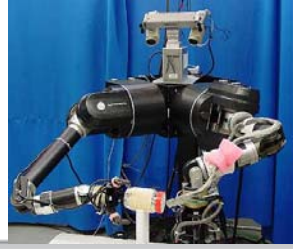
Relaxation Labelling: Consistencies



- Define ***consistencies*** c_p and c_n
  - Positive c_p and negative c_n encourages neighbouring pixels to have the same label
  - Setting these consistencies to appropriate values will encourage spatially contiguous object and background regions
- We assume again a bi-modal object/background histogram with maximum greylevel g_{max}



Relaxation Labelling: Algorithm

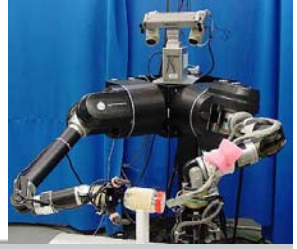


- We can initialize the probabilities

$$p^{(0)}(i) = g(i) / g_{max}$$

- Our relaxation algorithm must 'drive' the **background pixel** probabilities *p(i)* to **1** and the **object pixel** probabilities to **0**
- We want to take into account:
  - **Neighbouring probabilities** $p(i_1), p(i_2), \ldots\ldots p(i_8)$
  - The **consistency values** c_p and c_n
- We would like our algorithm to '**saturate**' such that *p(i)~1*
  - We can then convert the probabilities to labels by multiplying by 255

Relaxation Labelling: Algorithm

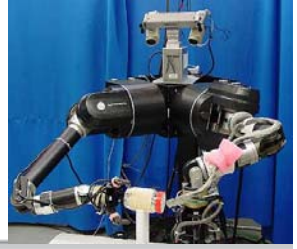


- We can derive the equation for relaxation labelling by first considering a neighbour i_1 of pixel i
  - We would like to evaluate the **contribution to** the **increment** in $p(i)$ from i_1
  - Let this increment be $q(i_1)$

    We can evaluate $q(i_1)$ by taking into account the consistencies

- We can apply a simple decision rule to determine the contribution to the increment $q(i_1)$ from pixel i_1
  - If $p(i_i)$ **>0.5** the contribution from pixel i_1 **increments** $p(i)$
  - If $p(i_i)$ <0.5 the contribution from pixel i_1 **decrements** $p(i)$

Relaxation Labelling: Algorithm



- Since $c_p > 0$ and $c_n < 0$ its easy to see that the following expression for $q(i_1)$ has the right properties

$$q(i_1) = c_p\, p(i_1) + c_n (1 - p(i_1))$$

- We can now **average all** the contributions from the **8-neighbours** of *i* to get the total increment to *p(i)*

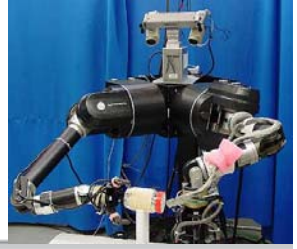
$$\Delta p(i) = \frac{1}{8} \sum_{h=1}^{8} (c_p\, p(i_h) + c_n (1 - p(i_h)))$$

- Easy to check that $-1 < \Delta p(i) < 1$ for $-1 < c_p, c_n < 1$
- Can update *p(i)* as follows:

$$p^{(r)}(i) \sim p^{(r-1)}(i)(1 + \Delta p(i))$$

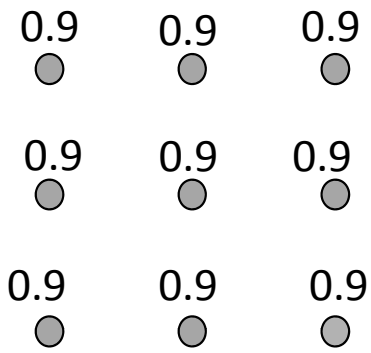
  - Ensures that *p(i)* remains positive
- Basic form of the relaxation equation

Relaxation Labelling: Normalization



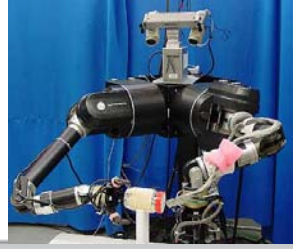
- We need to ***normalize*** the probabilities *p(i)* as they must stay in the range {0..1}
  - After every iteration $p^{(r)}(i)$ is ***rescaled*** to bring it back into the correct range
- Remember our requirement that likely background pixel probabilities are 'driven' to 1
- One possible approach is to use a ***constant normalisation*** factor

$$p^{(r)}(i) \rightarrow p^{(r)}(i) / max_i \; p^{(r)}(i)$$



  - the central background pixel probability may get stuck at 0.9 if max(*p(i)*)=1

Relaxation Labelling: Normalization

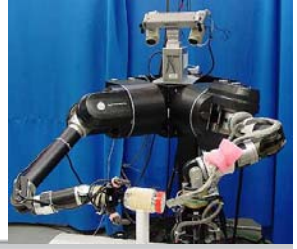


- The following normalisation equation has all the right properties
  - It can be derived from the general theory of relaxation labelling

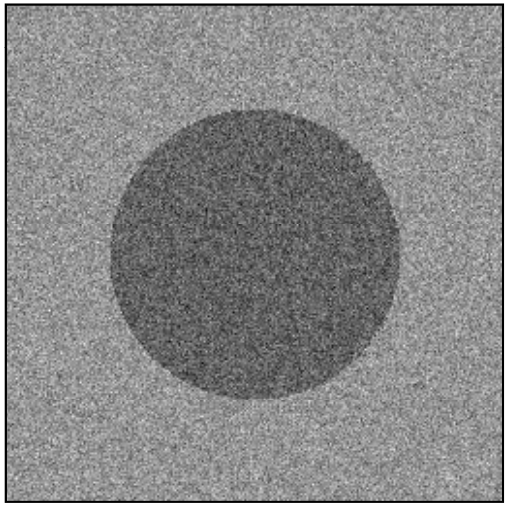
$$p^{(r)}(i) = \frac{p^{(r-1)}(i)(1+\Delta p(i))}{p^{(r-1)}(i)(1+\Delta p(i)) + (1-p^{(r-1)}(i))(1-\Delta p(i))}$$

- We can check to see if this normalisation equation has the correct 'saturation' properties
  - When $p^{(r-1)}(i)=1$, $p^{(r)}(i)=1$
  - When $p^{(r-1)}(i)=0$, $p^{(r)}(i)=0$
  - When $p^{(r-1)}(i)=0.9$ and $\Delta p(i)=0.9$, $p^{(r)}(i)=0.994$
  - When $p^{(r-1)}(i)=0.1$ and $\Delta p(i)=-0.9$, $p^{(r)}(i)=0.012$
- We can see that $p(i)$ converges to 0 or 1

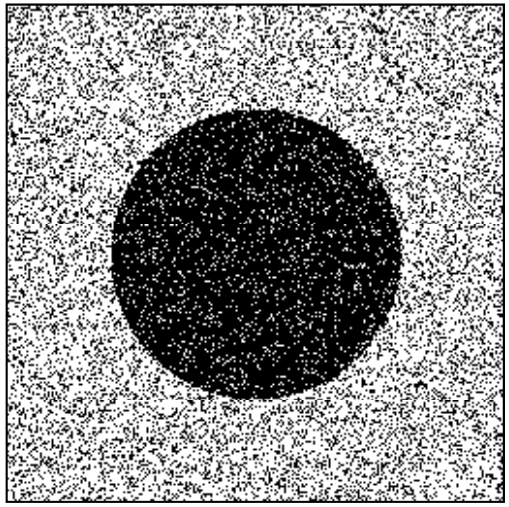
Relaxation Labelling: Results



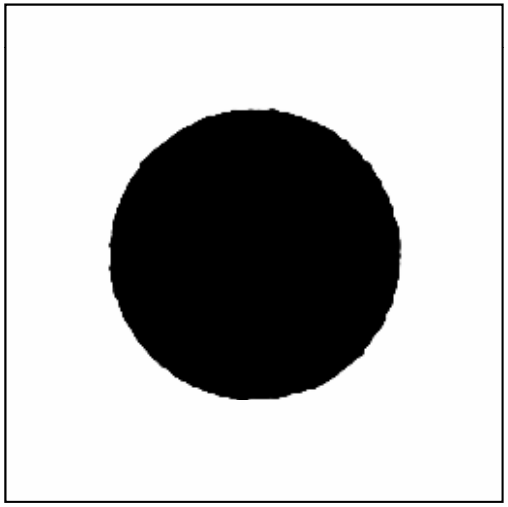
- Algorithm performance on the high noise image
  - Comparison with thresholding



High noise *circle* image

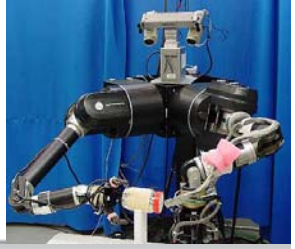


’ Optimum’ threshold

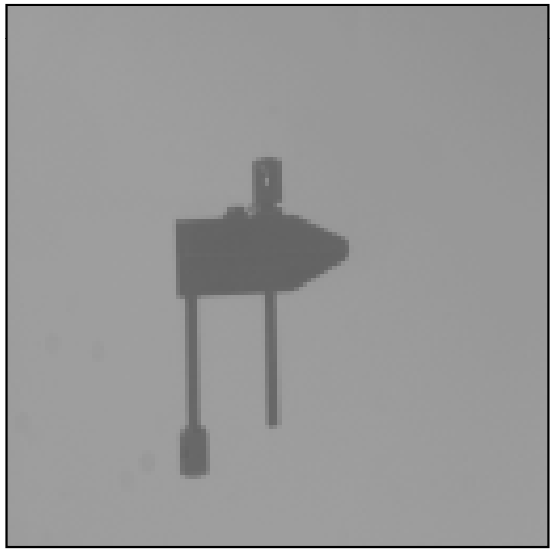


Relaxation labeling
20 iterations

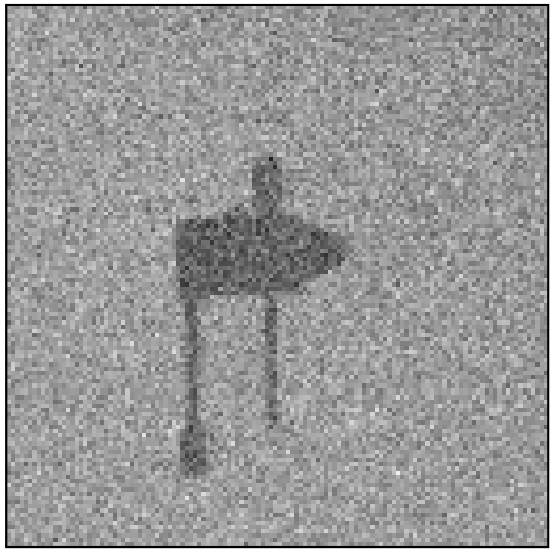
Relaxation Labelling: Results



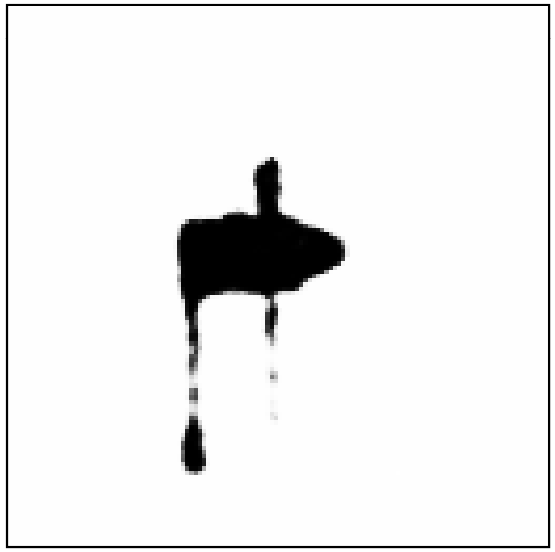
- The following is an example of a case where the algorithm has problems due to the thin structure in the *clamp* image



clamp image
original

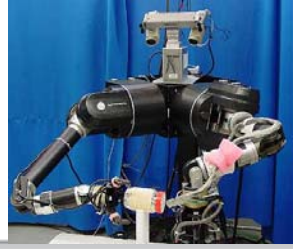


clamp image
noise added



segmented *clamp* image
10 iterations

Relaxation Labelling: Results



- Applying the algorithm to normal greylevel images we can see a clear separation into *light* and *dark* areas

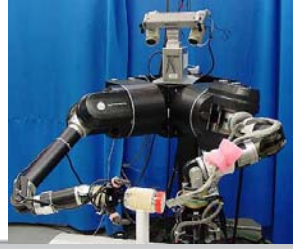
Original

2 iterations

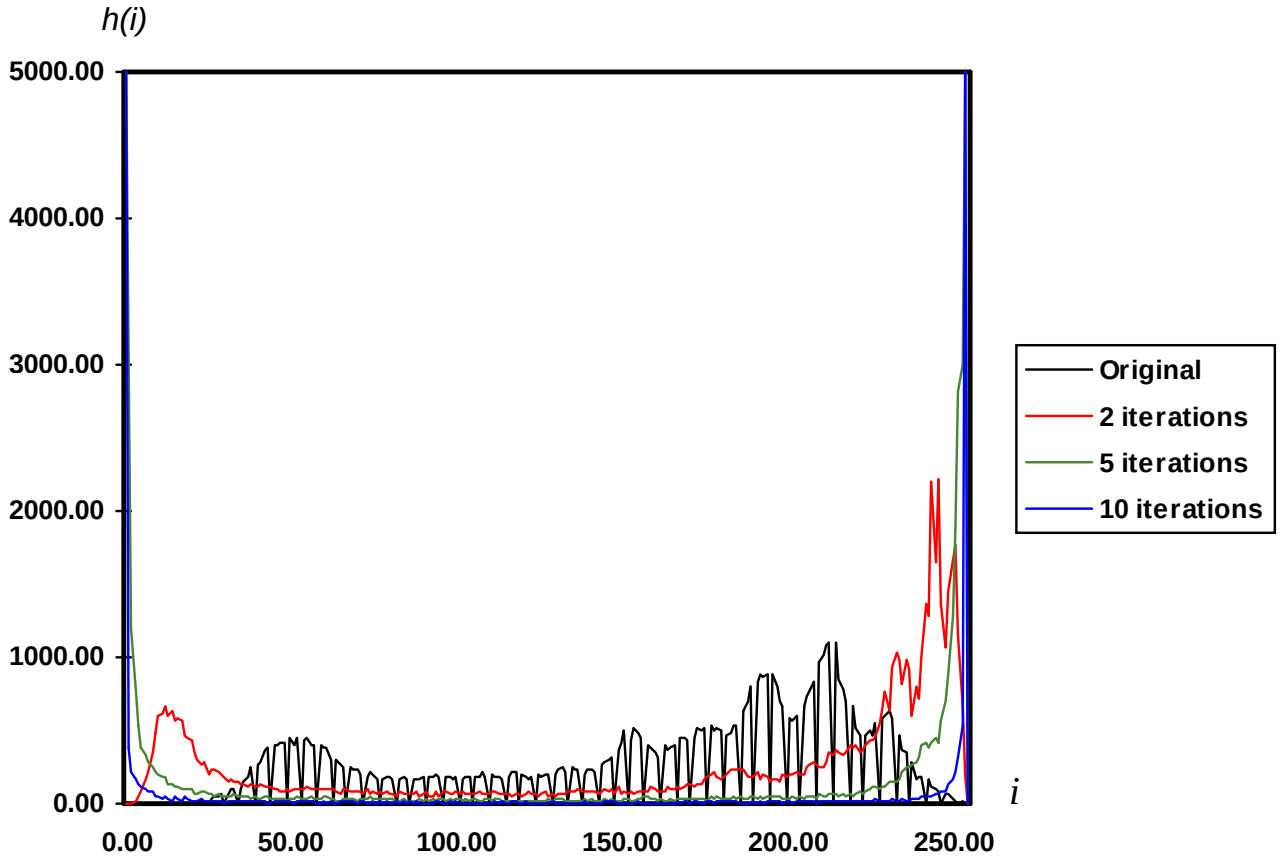
5 iterations

10 iterations

Relaxation Labelling: Results

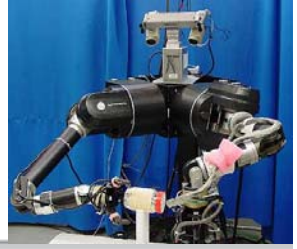


- The histogram of each image shows the clear saturation to 0 and 255



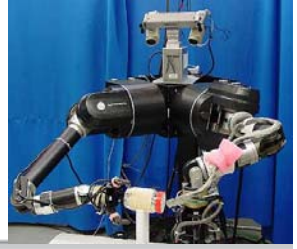
Region Growing

Region-Based Segmentation



- We want **smooth regions** in the image
  - We still want the pixels in each region to be **similar**, and those in adjacent regions to be **different**
  - One way to do this is to work with **regions** rather than pixels
- Region growing
  - Start with a **small 'seed'** and expand by adding similar pixels
- Split and merge
  - Splitting **divides** regions that are **inconsistent**
  - Merging **combines** adjacent regions that are **consistent**

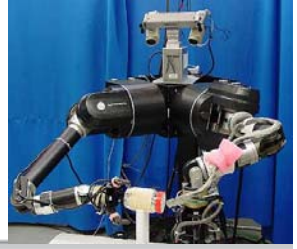
Region Growing



- *Simple approach: start from some pixels (**seeds**) representing distinct image regions and to **grow** them, until they cover the **entire image***

- *For region growing we need a rule describing a **growth mechanism** and a rule **checking the homogeneity** of the regions after each growth step*
  - ***Growth mechanism**: at each stage k and for each region $R_i(k)$, $i = 1,\ldots,N$, check if there are **unclassified pixels** in the 8-neigh-bourhood of each pixel of the region border*
  - *Before assigning such a pixel x to a region $R_i(k)$,we check if the **region homogeneity**: $P(R_i(k) \cup \{x\}) = TRUE$, is **valid***

Growth Mechanism

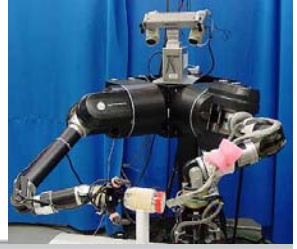


- The arithmetic mean *m* and standard deviation *sd* of a class R_i having *n* pixels:

$$m = \frac{1}{n}\sum_{i=1}^{n} I(x,y) \qquad sd = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[I(x,y)-m]^2}$$

  - can be used to decide if the merging of the two regions R_1, R_2 is allowed
  - if $|m_1 - m_2| < k \cdot sd(i), i = 1,2$ two regions are merged

Region Homogeneity



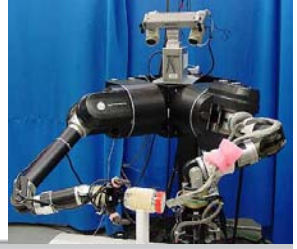
- Homogeneity test: if the pixel intensity is close to the region mean value

$$m = \left| I(x,y) - m_i \right| \leq T_i$$

- Threshold T_i varies depending on the region R_n and the intensity of the pixel $I(x,y)$ and can be chosen by:

$$T_i = \left(1 - \frac{sd(i)}{m(i)} \right) T$$

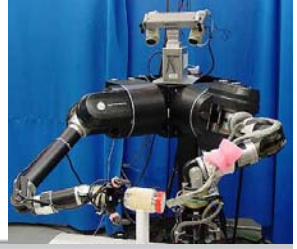
Region Growing Summary



- Region growing starts with a **small patch of seed** pixels
  - Compute statistics about the region
  - Check neighbors to see if they can be added
  - Re-compute the statistics

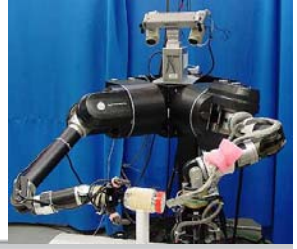
- This procedure repeats until the region stops growing
  - Simple example: We compute the mean grey level of the pixels in the region
  - Neighbors are added if their grey level is near the average

Region Growing Example



3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

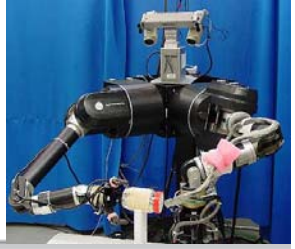
Region Growing Example



- We start with T=10

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

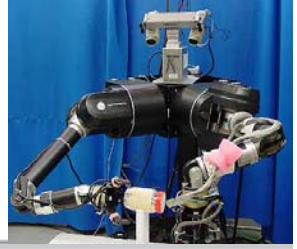
Region Growing Example



- Check neighbors in line 4 (11,17,19) in 8-neighborhood

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

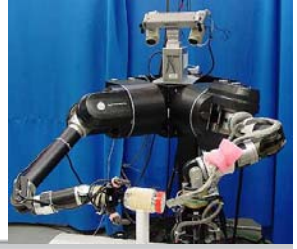
Region Growing Example



- Check neighbors in line 5 (11,12,18,16) in 8-neighborhood

3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

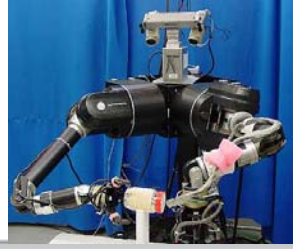
Region Growing Example

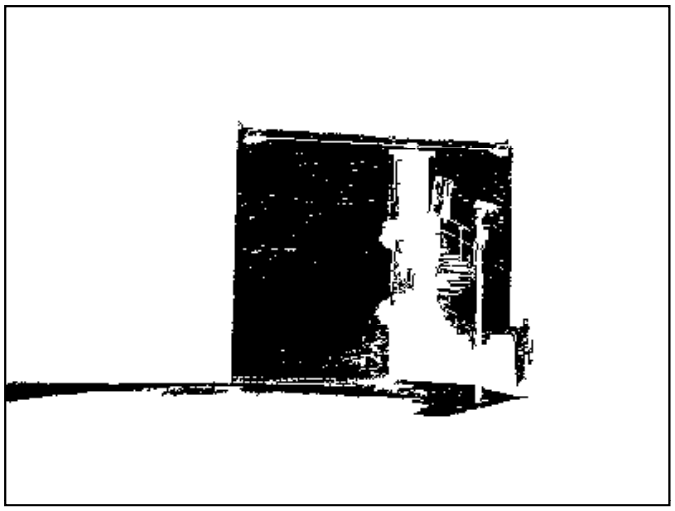


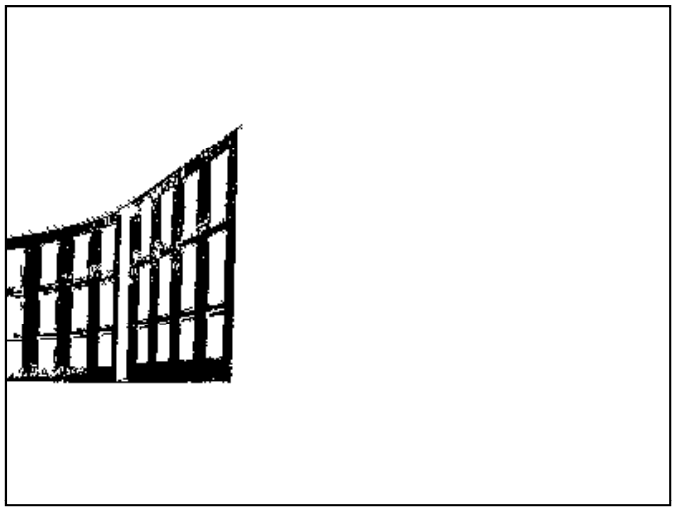
- Check neighbors in line 3 (12,11,15,10) in 8-neighborhood

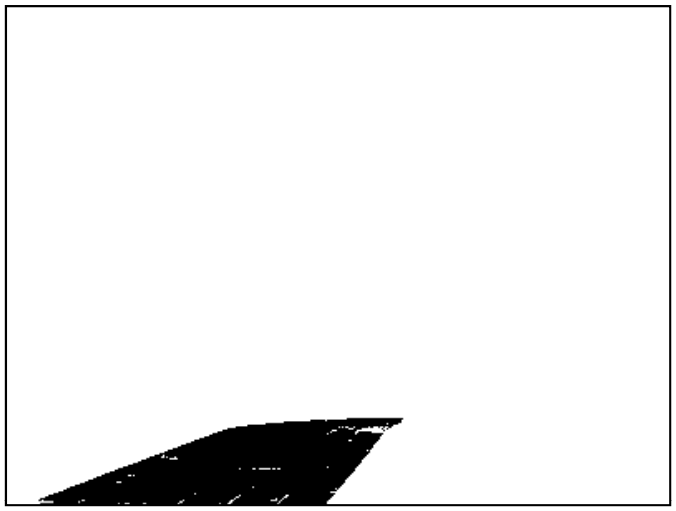
3	5	7	3	4	2	1
2	4	9	10	22	9	3
3	5	12	11	15	10	3
5	6	11	9	17	19	1
2	3	11	12	18	16	2
3	6	8	10	18	9	5
4	6	7	8	3	3	1

Region Growing Example



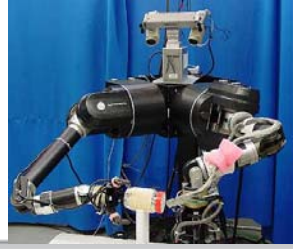






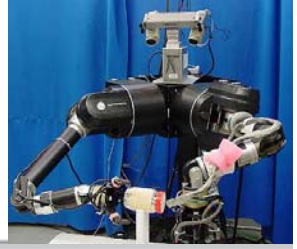
Split and Merge

Split and Merge



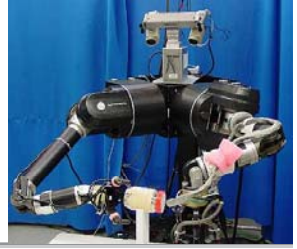
- The opposite approach to region growing is region shrinking (**Splitting**)
- It is a **top-down** approach and it starts with the assumption that the entire image is **homogeneous**
- If this is not true, the image is split into **four sub images**
- This splitting procedure is repeated **recursively** until we split the image into **homogeneous regions**

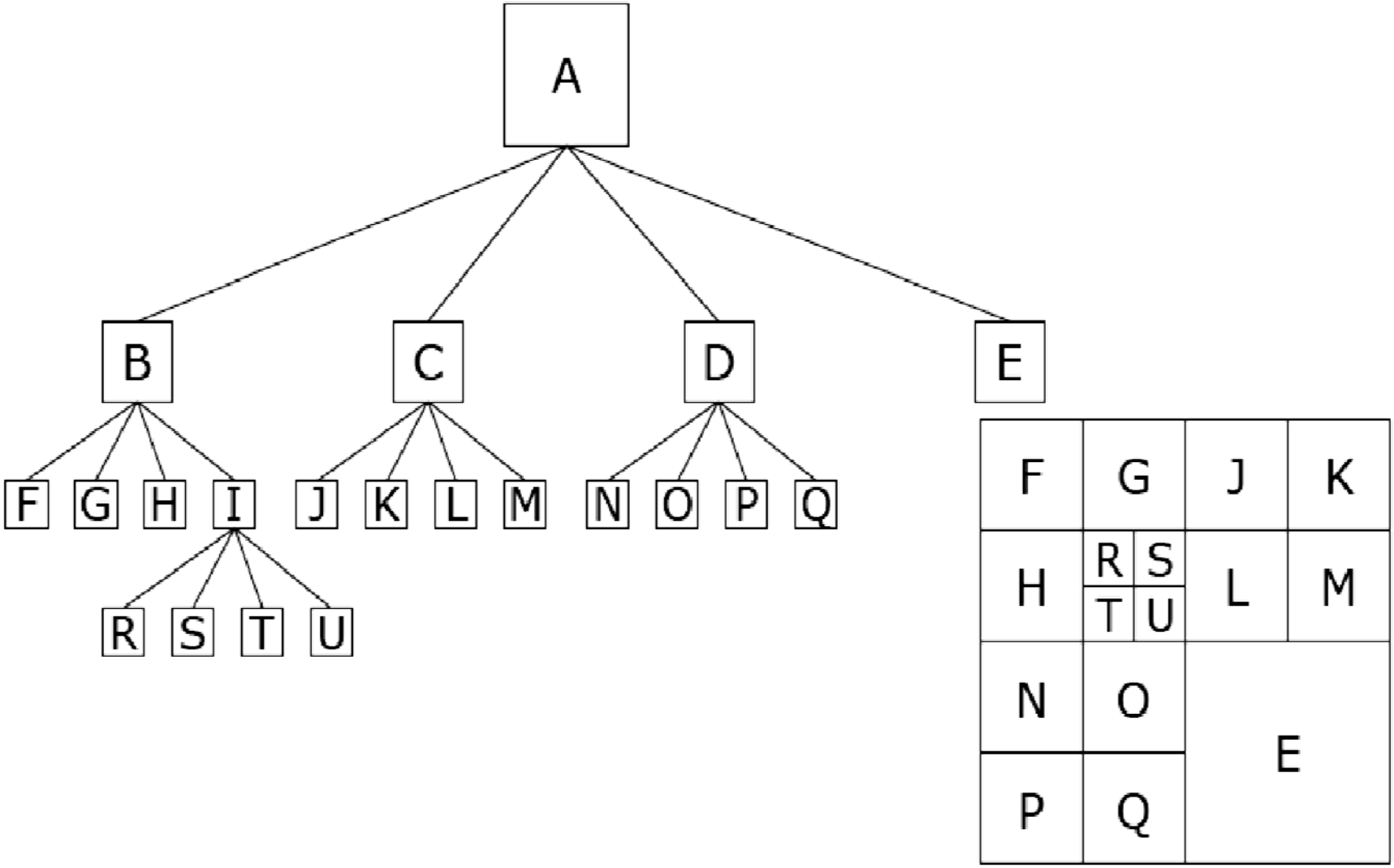
Split



- If the original image is square N x N, having dimensions that are powers of 2($N = 2^n$):
  - All regions produced but the splitting algorithm are squares having dimensions M x M , where M is a power of 2 as well ($M=2^m, M <= n$).
- Since the procedure is recursive, it produces an image representation that can be described by a tree whose nodes have four sons each
- Such a tree is called a **Quadtree**.

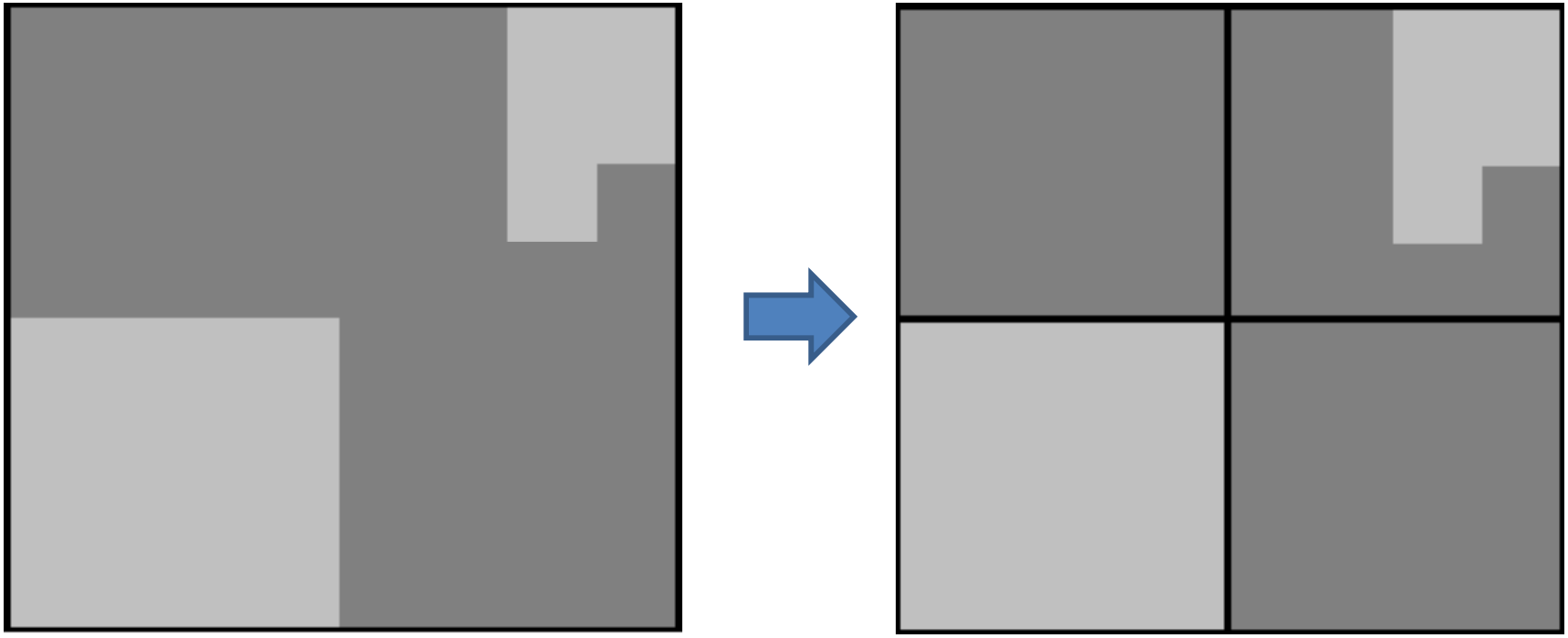
Quadtrees



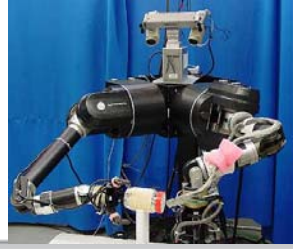


Split Example

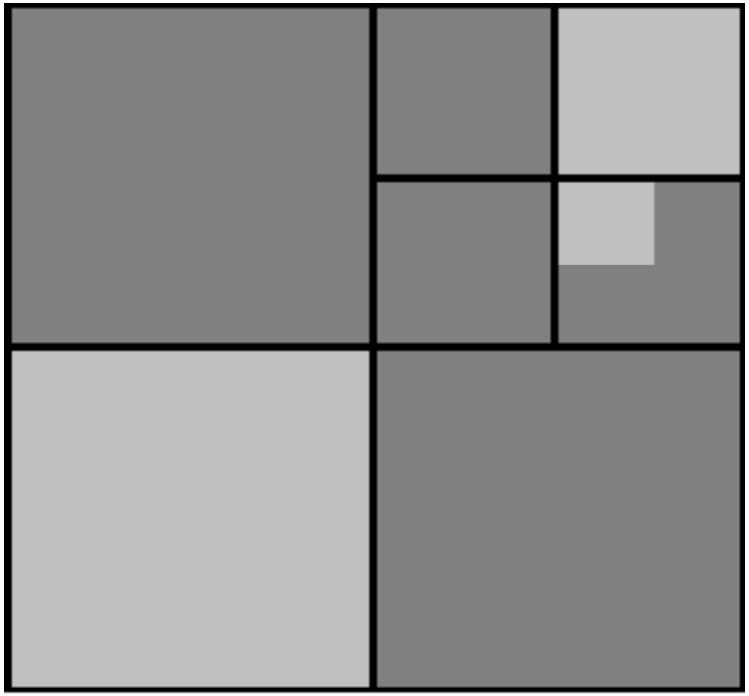
- Splits original image into 4 sub-images



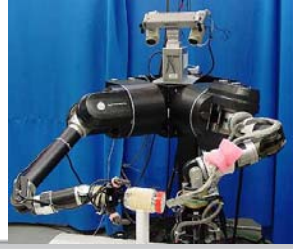
Split Example



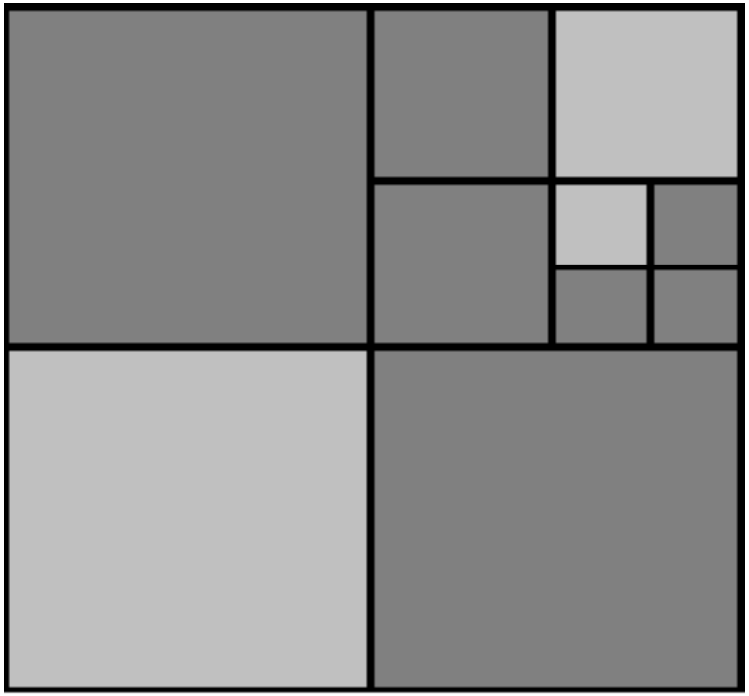
- 3 sub-images are homogeneous
  - Upper right sub-image hast to be split



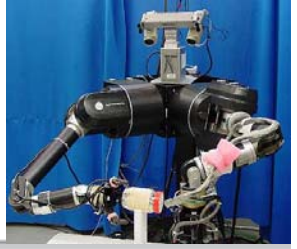
Split Example

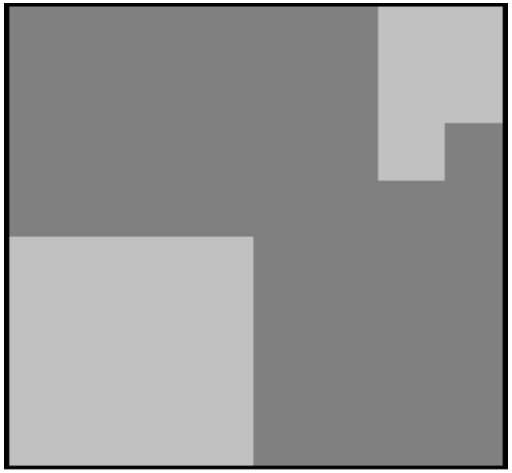


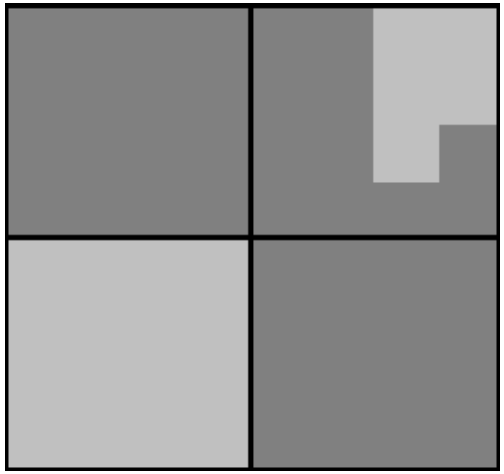
- 3 sub-sub-images are homogeneous
  - Lower right sub-sub-image hast to be split

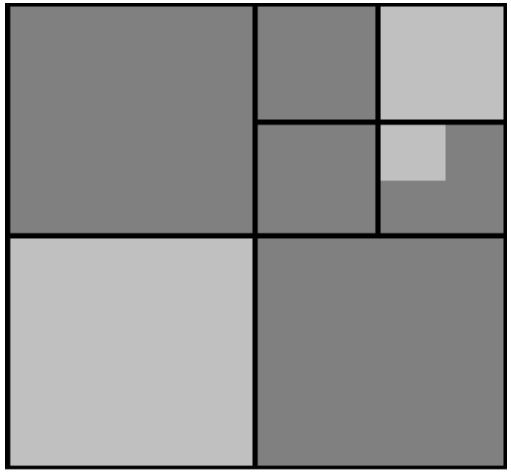


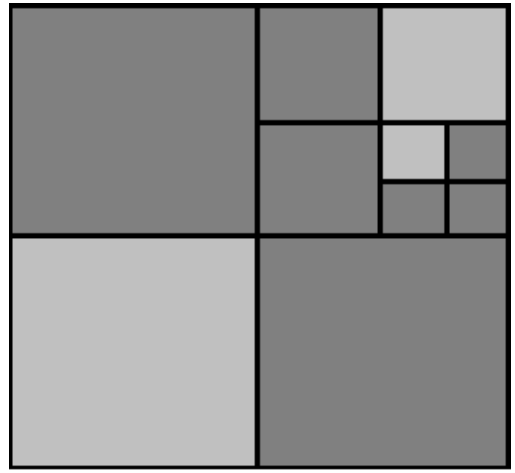
Split Example

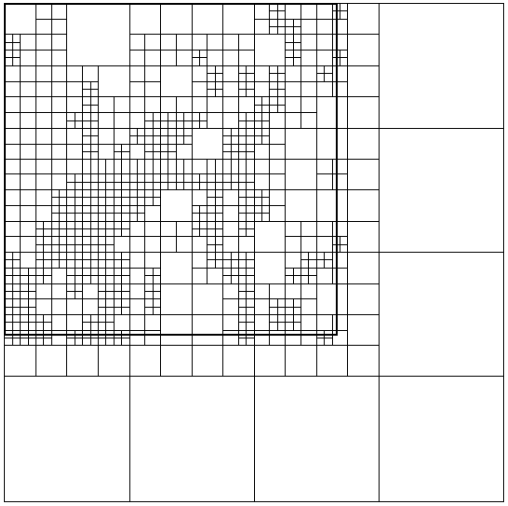


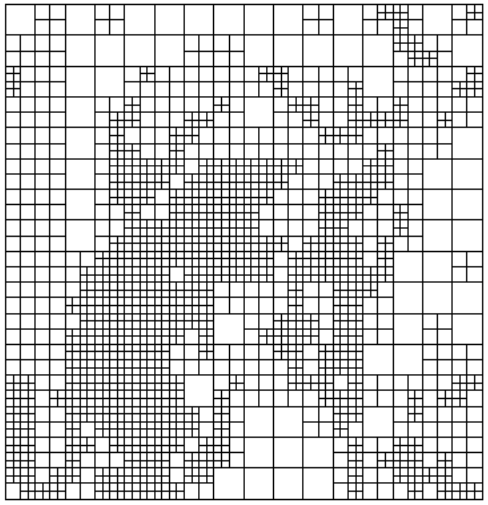


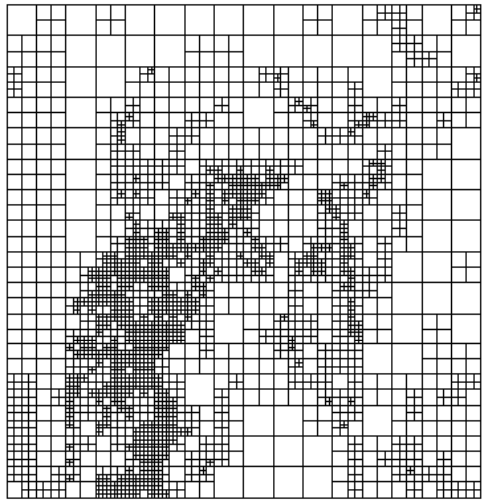




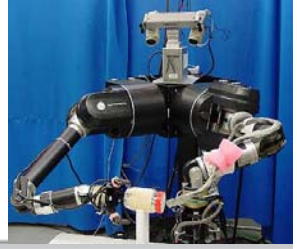






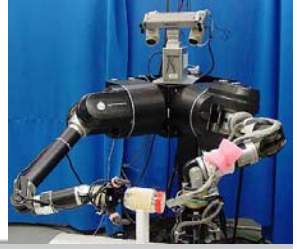


Split and Merge



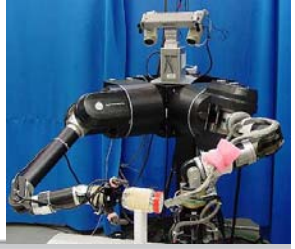
- Splitting **disadvantage**: creates regions that may be adjacent and homogeneous, but **not merged**
- Split and Merge method: iterative algorithm that includes both splitting and merging at **each iteration**:
  - *(P(R)= False):* if a region *R* is inhomogeneous then split into four sub regions
  - *(P(R_i U R_j) = TRUE):* if two adjacent regions R_i,R_j are homogeneous, they are merged
  - The algorithm stops when no further splitting or merging is possible
- Split and Merge algorithm produces **more compact regions** than the pure splitting algorithm

Applications

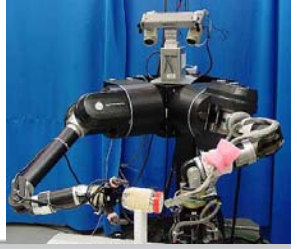


- 3D – Imaging : A basic task in 3-D image processing is segmentation of images which classifies voxels/pixels into objects or groups.
- 3-D image segmentation makes it possible to create 3-D rendering for multiple objects and performs quantitative analysis for size, density or other parameters of detected objects.
- Several applications in the field of Medicine like magnetic resonance imaging (MRI).

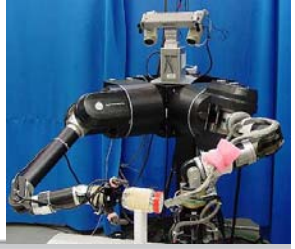
Results – Region Growing



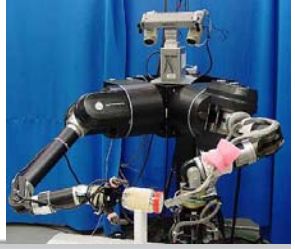
Results – Region Split

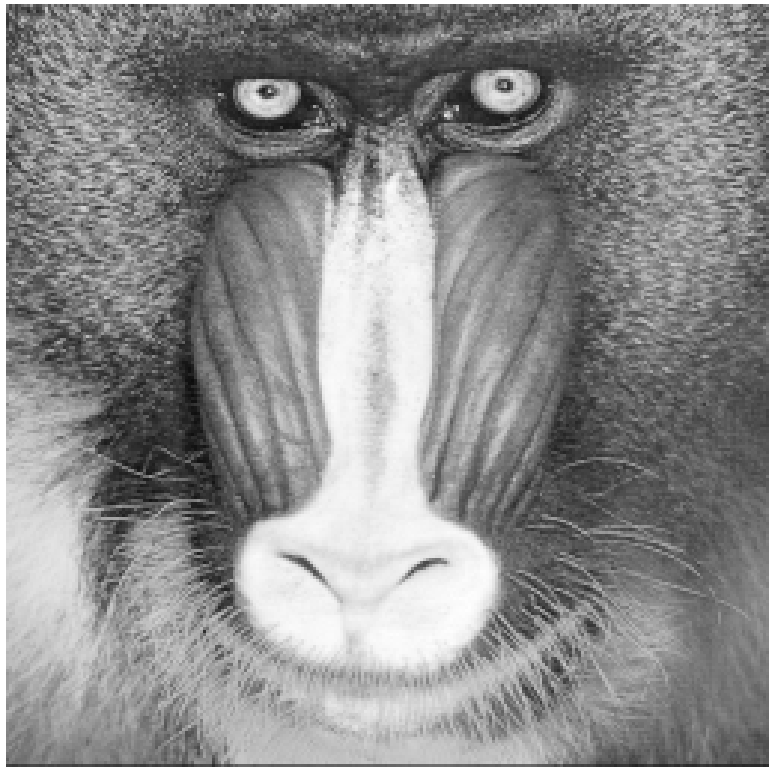


Results – Region Split and Merge

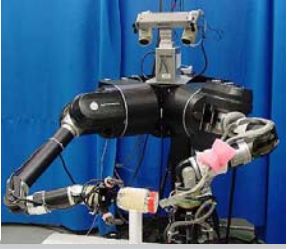


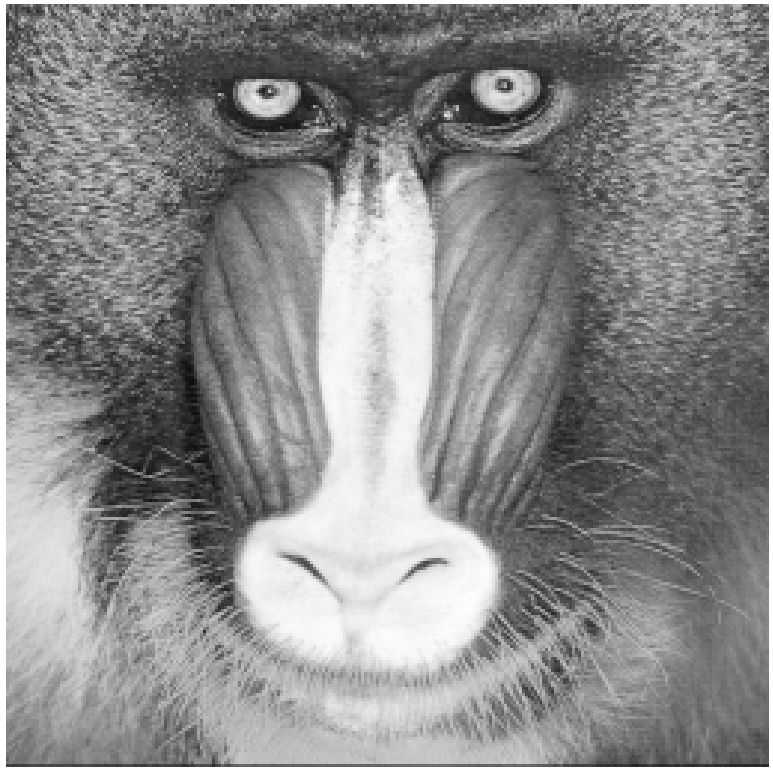
Results – Region Growing

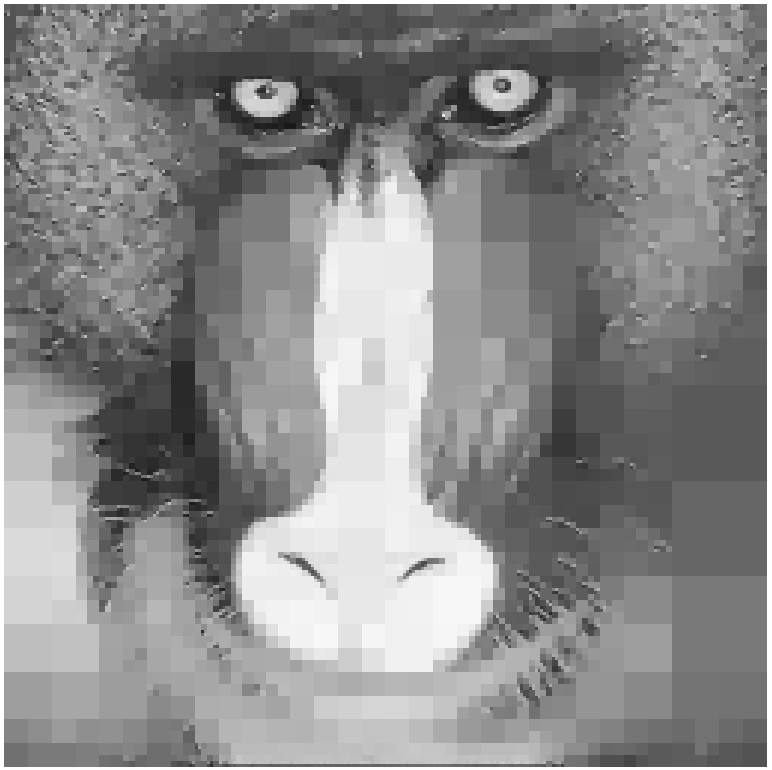




Results – Region Split







Results – Region Split and Merge