Vision Analysis and Deep Learning

Unit 4

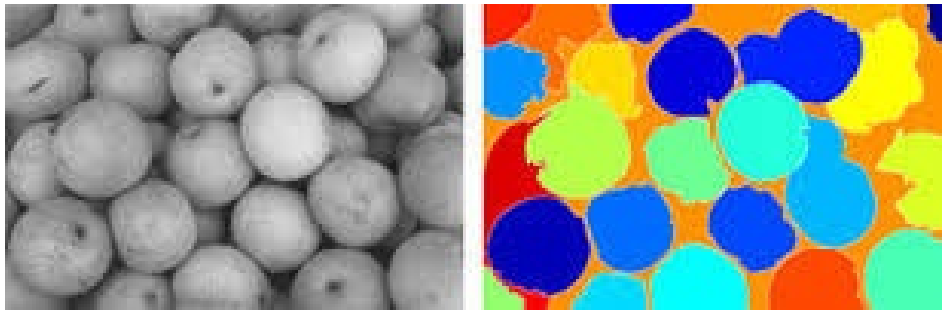
Image and video segmentation

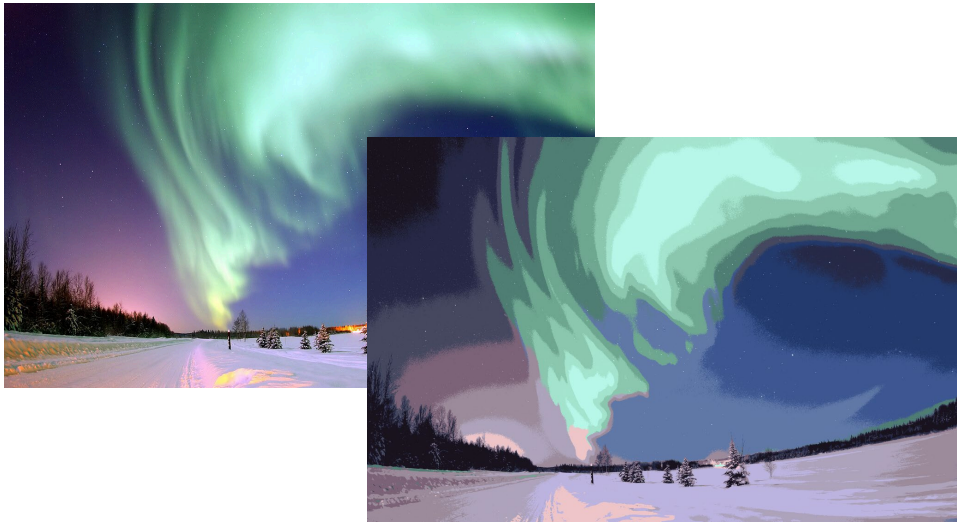
Index of content

4.0 - Introduction

4.0 - Introduction

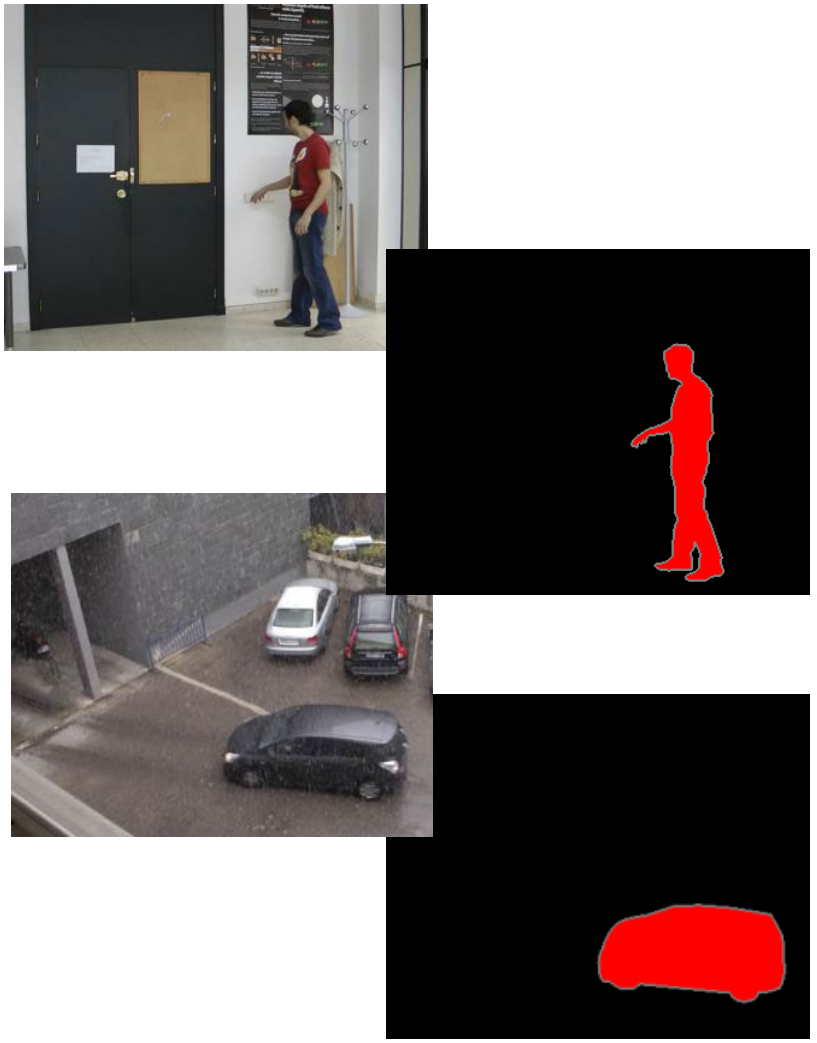
- **Image segmentation:**
  - It is the process of partitioning an image into multiple segments.
  - Its goal is to simplify and/or change the content of the image into something easier to analyze.
  - Is an essential step in many computer vision applications, such as:
    - Image analysis.
    - Object representation.
    - Visualization.
    - Etc.





4.0 - Introduction

- **Foreground segmentation (detection):**
  - It is one of the mayor tasks in the field of computer vision.
  - Its aim is to detect changes in image sequences.
  - These changes are used as input data by many high-level analysis tasks:
    - Video-surveillance.
    - Object tracking.
    - Augmented reality.
    - Classification.
    - Etc.



Unit 4

Image and video segmentation

4.1 - Image segmentation

4.1 Image segmentation

4.1.1 Introduction

4.1.2 Threshold-based segmentation

- Using histogram extrema
- Otsu's method
- K-means algorithm
- Optimal thresholding
- Histogram estimation
- Enhancing threshold segmentation

4.1.3 Edge-based segmentation

- The Hough transform
- Watershed segmentation
- Active contours (snakes)

4.1.1 Introduction

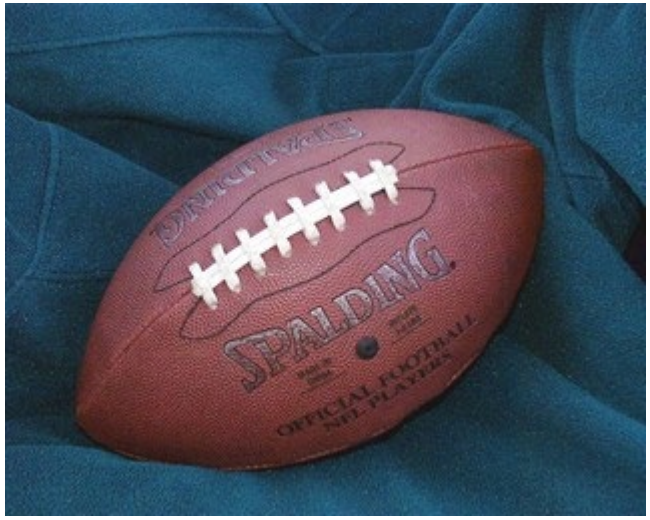
Definition: Image segmentation is the process of partitioning a digital image into multiple segments.

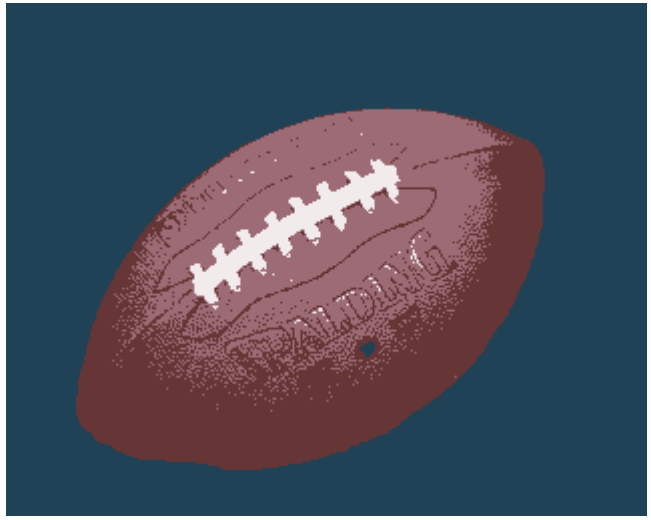
Goal: To simplify and/or change the representation of an image into something more meaningful and easier to analyze.

Result: A label is assigned to each pixel in the image. The pixels with the same label share certain characteristics.

Example:

Color-based segmentation





4.1.1 Introduction

Definition: Image segmentation is the process of partitioning a digital image into multiple segments.

Goal: To simplify and/or change the representation of an image into something more meaningful and easier to analyze.

Result: A label is assigned to each pixel in the image. The pixels with the same label share certain characteristics.

Example:

Edge-based segmentation

https://en.wikipedia.org/wiki/Edge_detection#/media/File:%C3%84%C3%A4retuvastuse_n%C3%A4ide.png

4.1.1 Introduction

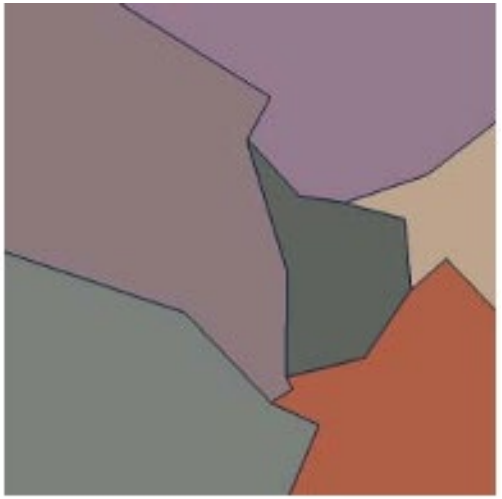
Definition: Image segmentation is the process of partitioning a digital image into multiple segments.

Goal: To simplify and/or change the representation of an image into something more meaningful and easier to analyze.

Result: A label is assigned to each pixel in the image. The pixels with the same label share certain characteristics.

Example:

Texture-based segmentation



S. Mike and M. Haindl, "Prague texture segmentationdata generator and benchmark", *ERCIM News*, vol. 64, pp. 67-68

4.1.1 Introduction

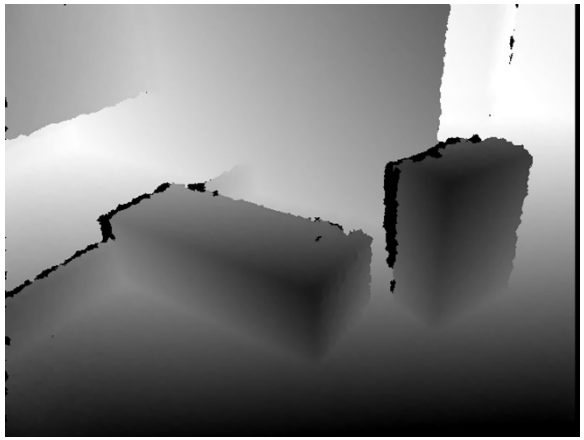
Definition: Image segmentation is the process of partitioning a digital image into multiple segments.

Goal: To simplify and/or change the representation of an image into something more meaningful and easier to analyze.

Result: A label is assigned to each pixel in the image. The pixels with the same label share certain characteristics.

Example:

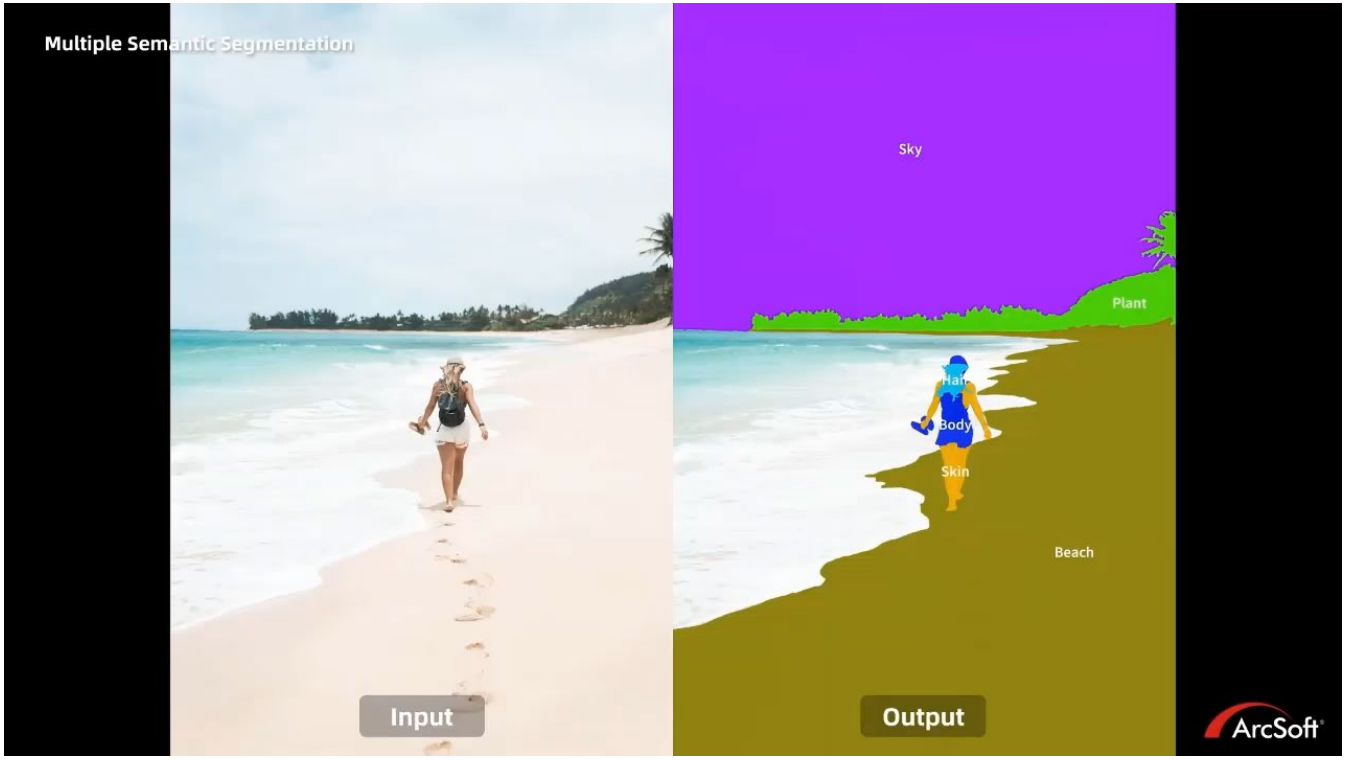
Depth-based segmentation



Gorte, B., & Sithole, G. (2012). Lookup table Hough Transform for real time range image segmentation and featureless co-registration.

4.1.1 Introduction

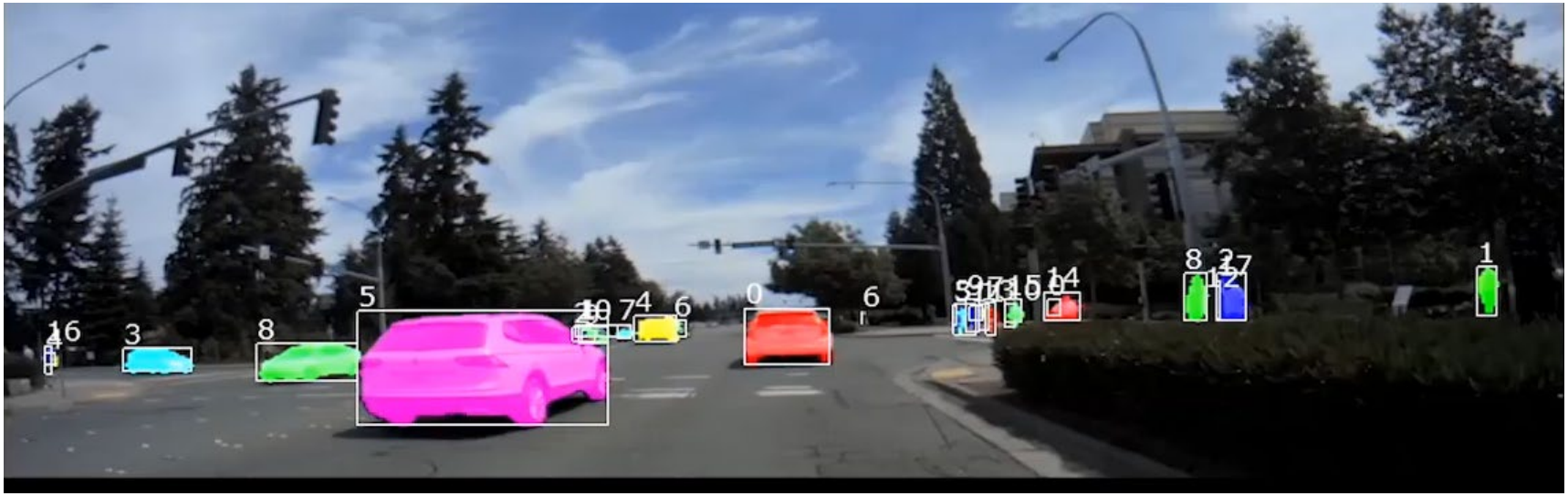
- **Semantic segmentation**.
  - It focuses on assigning class labels to each pixel, providing a high-level understanding of the scene.



https://www.youtube.com/watch?v=J3h8TX_qJ8Y

4.1.1 Introduction

- **Instance segmentation**.
  - It differentiates object instances, delineating each object's boundaries and generating separate masks for distinct objects.



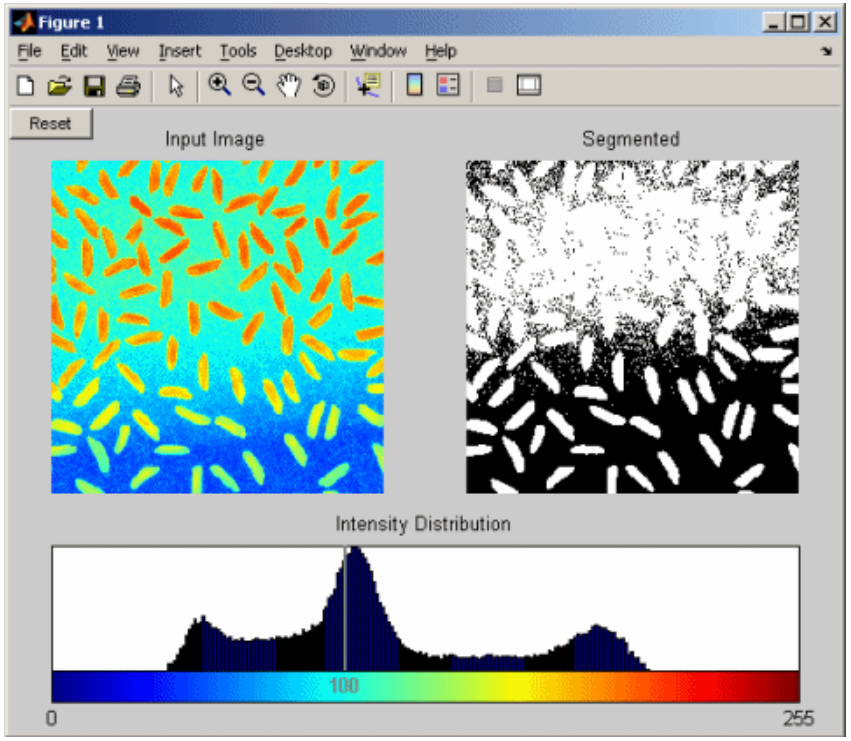
https://www.youtube.com/watch?v=HS1wV9NMLr8&list=PLAzJXCSg9-ZcVz5PeibgA8fY3nNFA4XCX

4.1.1 Introduction

- **Segmentation categories:**
  - There is a huge amount and variety of existing segmentation algorithms.
  - Typically, the segmentation algorithms can be classified into the following types:
    - Threshold-based segmentation.
    - Edge-based segmentation.
    - Region-based segmentation.
    - Clustering-based segmentation.
    - Matching-based segmentation.

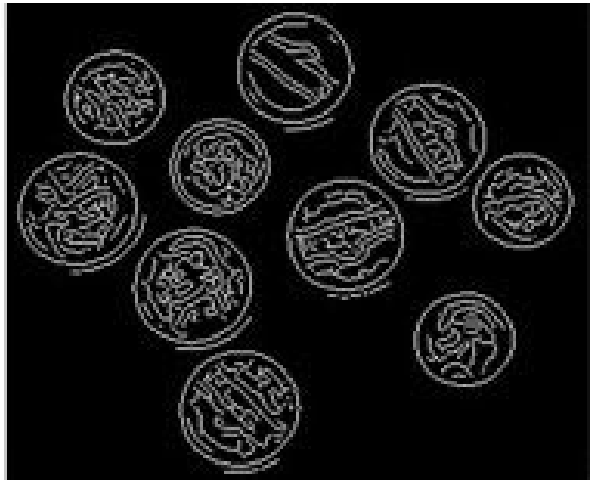
4.1.1 Introduction

- **Threshold-based segmentation:**
  - Color histogram thresholding techniques are used to segment the images.
  - They can be applied directly to an image, but they can also be combined with pre- and post-processing techniques.



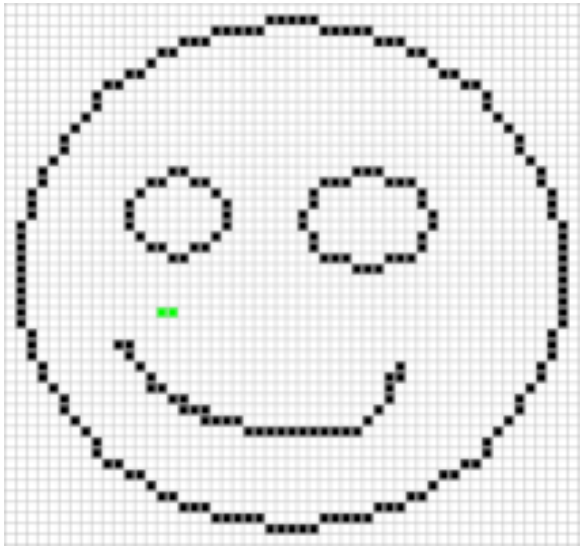
4.1.1 Introduction

- **Edge-based segmentation:**
  - The edges detected in an image are assumed to represent object boundaries.
  - These edges are used to identify the objects of interest.
  - Once the edges have been located, the objects are segmented by filling them in.



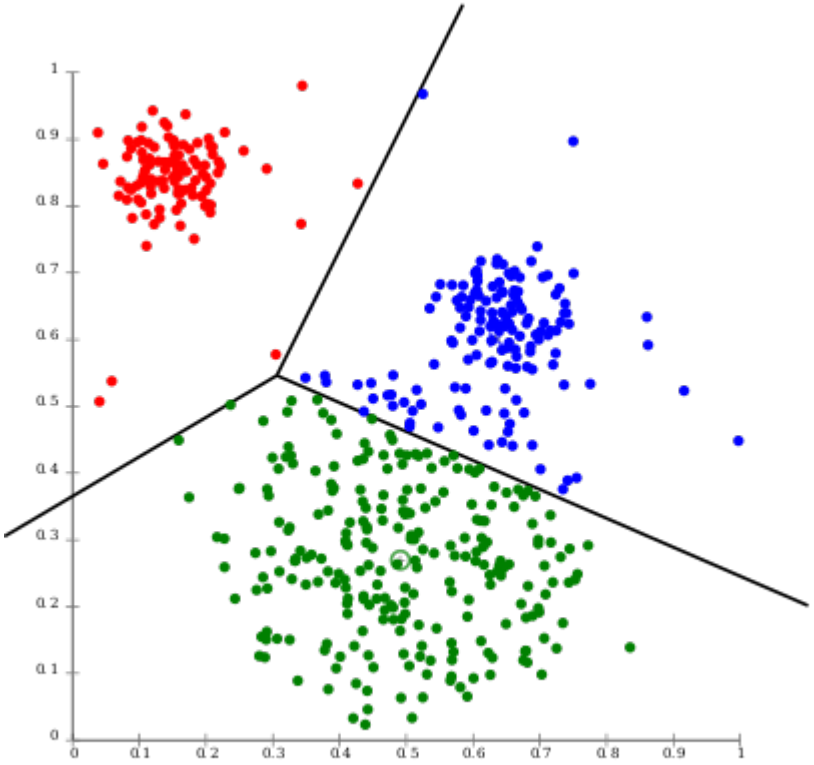
4.1.1 Introduction

- **Region-based segmentation:**
  - As opposite to edge-based segmentation, these techniques start in a small region inside an object and then grow until it meets the object boundaries.



4.1.1 Introduction

- **Clustering-based segmentation:**
  - These methods attempt to group patterns that are similar in some sense.



https://commons.wikimedia.org/wiki/File:KMeans-Gaussian-data.svg

4.1.1 Introduction

- **Matching-based segmentation:**
  - Once we know what an object we want to locate looks like, we use this knowledge to locate it in the image.

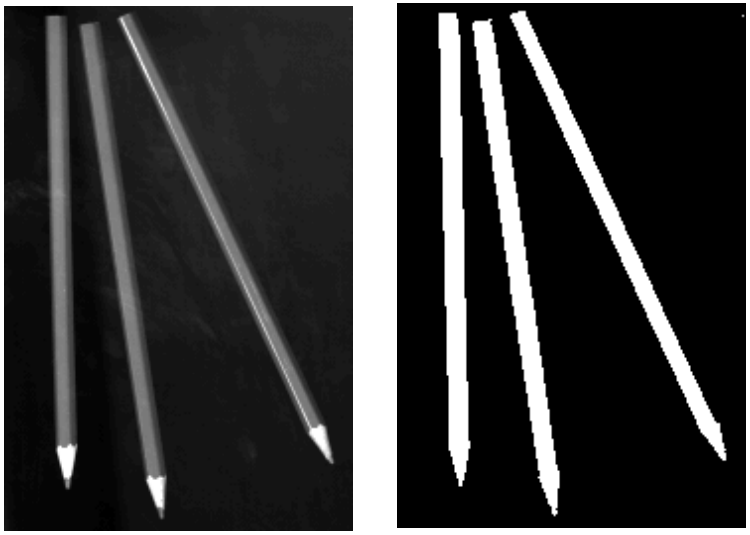
4.1.2 Threshold-based segmentation

- Thresholding is probably the simplest and most used method for segmenting images.
- Let $v(i,j)$ be the grey value of a pixel at position (i,j). The thresholding operation of such pixel can be defined as:

$$g(i,j) = \begin{cases} 0 & \text{if} \quad v(i,j) < t \\ 1 & \text{if} \quad v(i,j) \geq t \end{cases}$$

where t is the threshold value.

- After the thresholding operation the image has been segmented into two segments identified by the values 0 and 1.



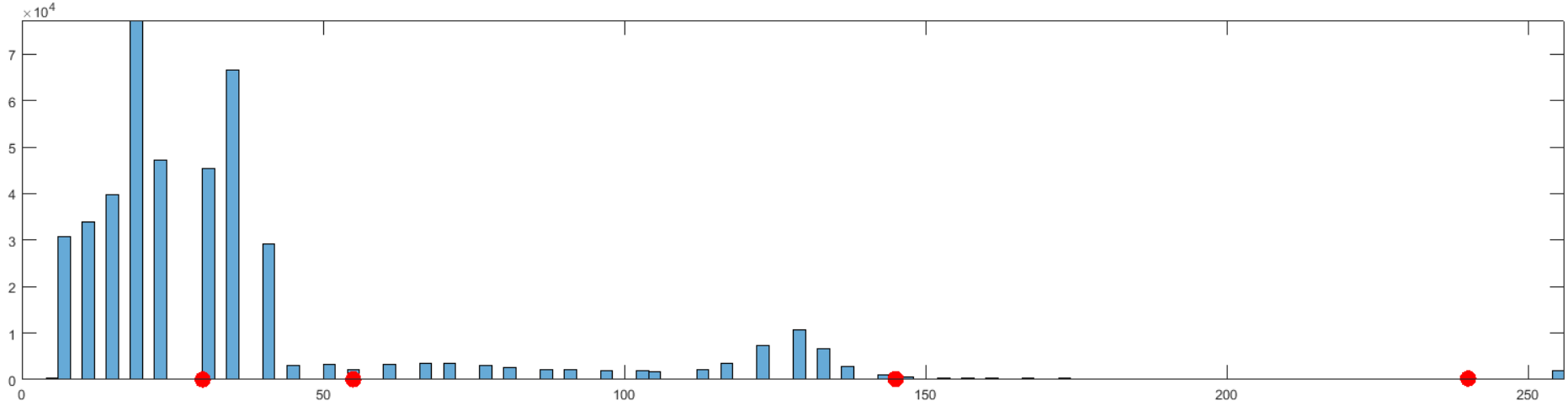
4.1.2 Threshold-based segmentation

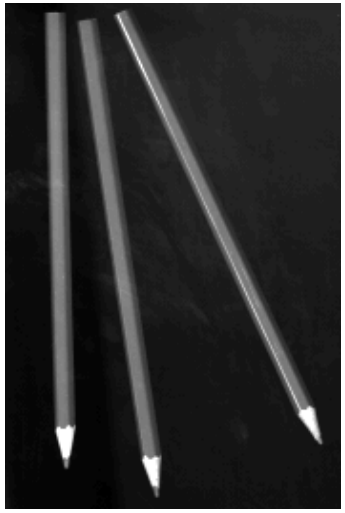
- What is the adequate threshold?

$t = 30$ $t = 55$ $t = 145$ $t = 254$

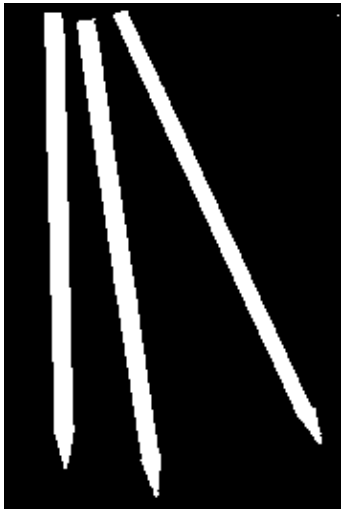
4.1.2 Threshold-based segmentation

- Histogram analysis:

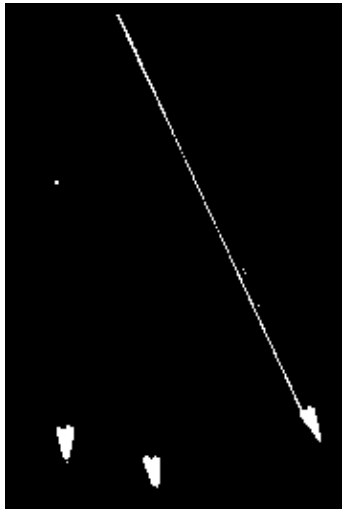




$t = 30$



$t = 55$



$t = 145$

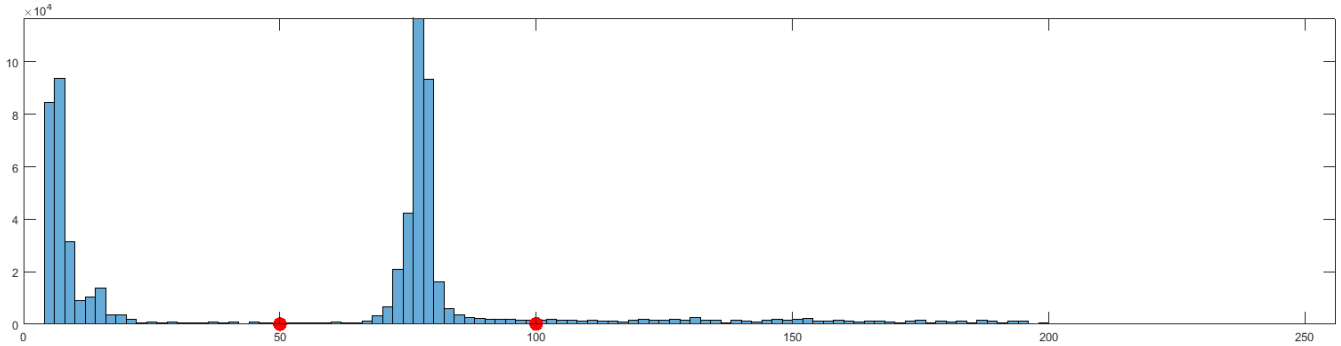
$t = 240$

4.1.2 Threshold-based segmentation

- When several desired segments can be distinguished by their grey values, several thresholds can be used as follows:

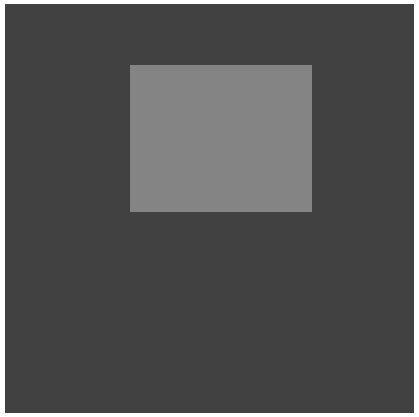
$$g(i,j) = \begin{cases} 0 & \text{if} \quad v(i,j) < t_1 \\ 1 & \text{if} \quad t_1 \leq v(i,j) < t_2 \\ 2 & \text{if} \quad t_2 \leq v(i,j) < t_3 \\ \vdots & \vdots \quad \vdots \\ n & \text{if} \quad t_n \leq v(i,j) \end{cases}$$

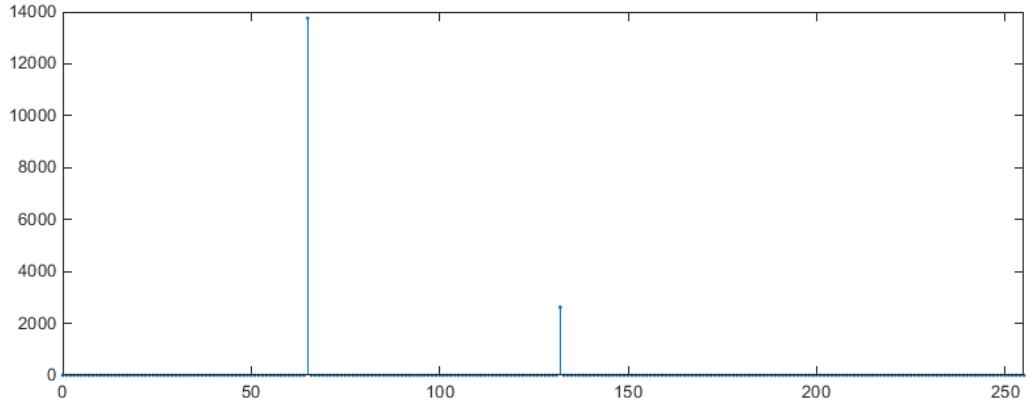
  - The image is segmented into $n + 1$ segments identified by the grey values 0 to n.

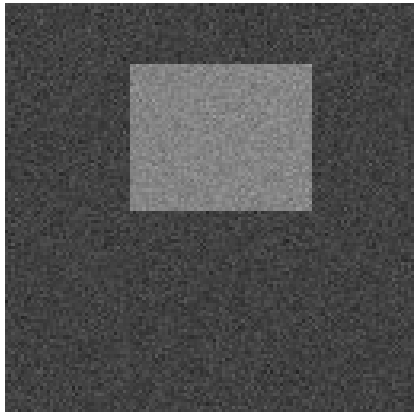


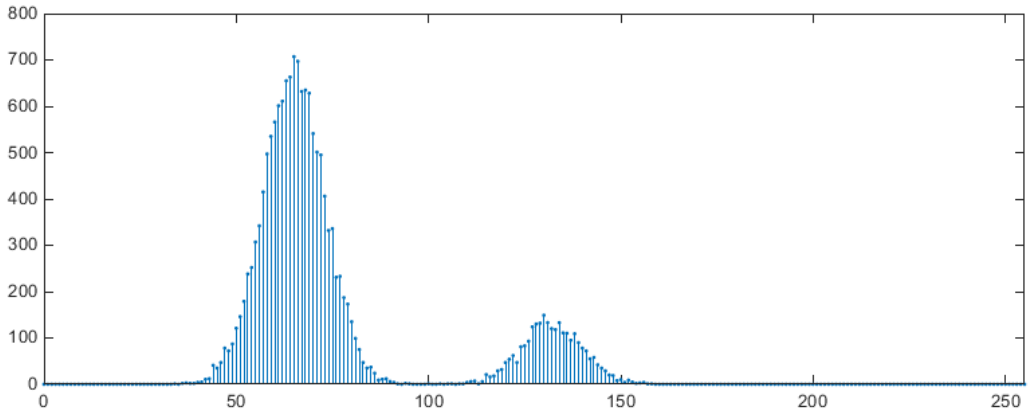
4.1.2.1 Using histogram extrema

- Typically, the images to segment will have a multimodal (multiple peaks) histogram.







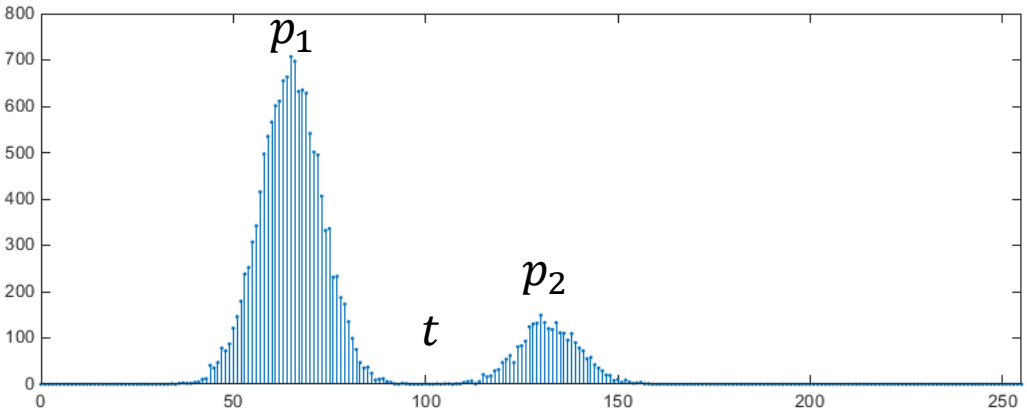


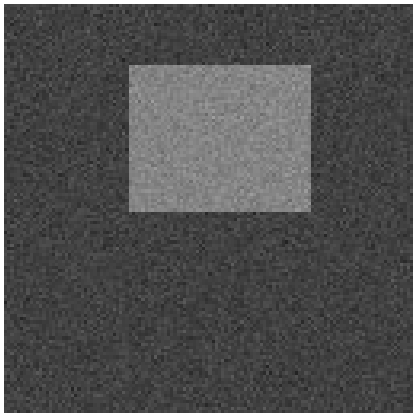
4.1.2.1 Using histogram extrema

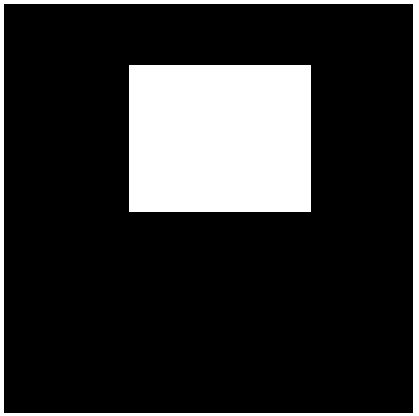
- Option 1: Use the maxima (peaks) to establish the segmentation threshold.
  - It can be set as the value between the two peaks.

$$t = \frac{p_1 + p_2}{2}$$

  - In this example: $t = \frac{65+135}{2} = 100$



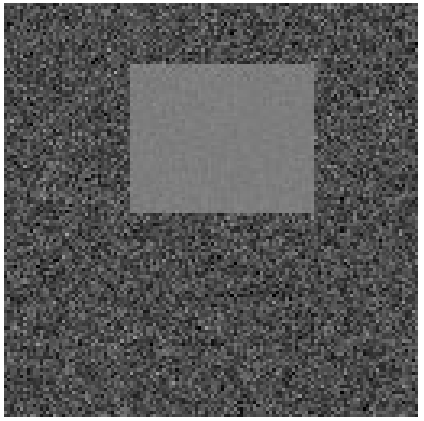




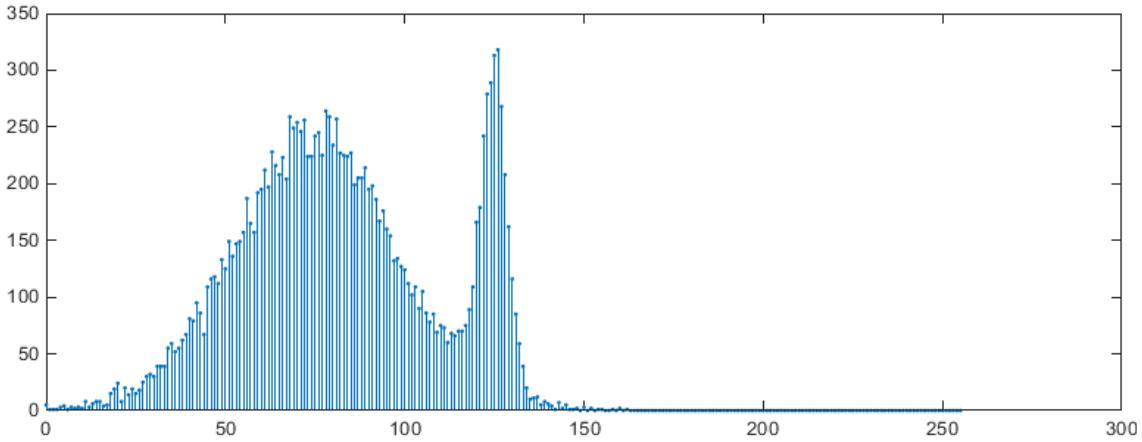
4.1.2.1 Using histogram extrema

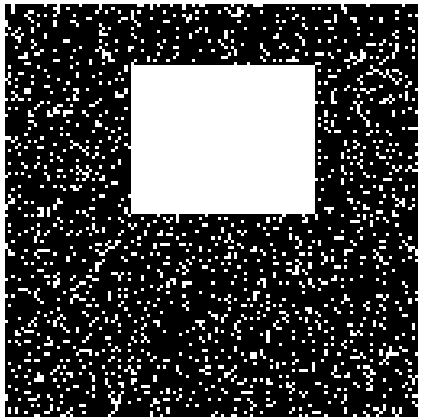
- Option 1: Use the maxima (peaks) to establish the segmentation threshold.
  - It can be set as the value between the two peaks.

$$t = \frac{p_1 + p_2}{2}$$



  - In this another example: $t = \frac{75+125}{2} = 100$





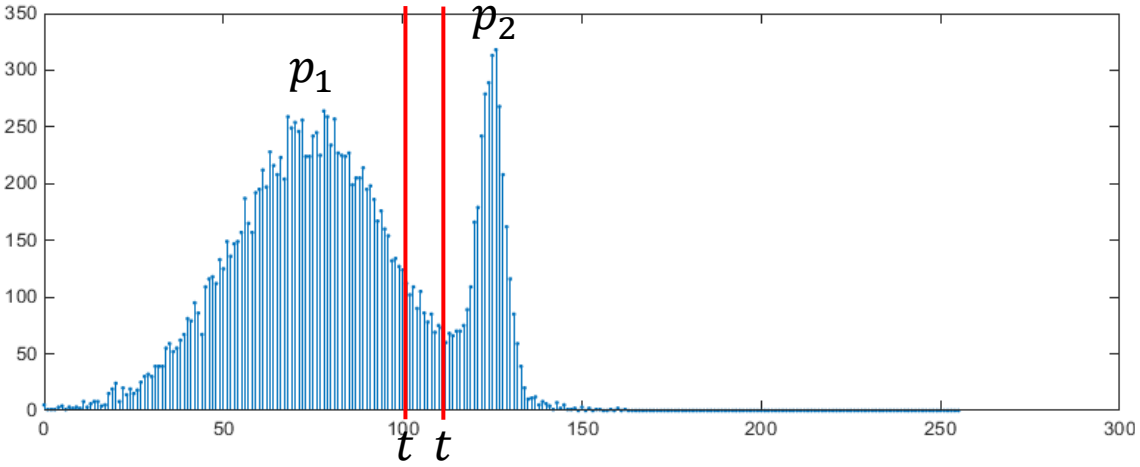
4.1.2.1 Using histogram extrema

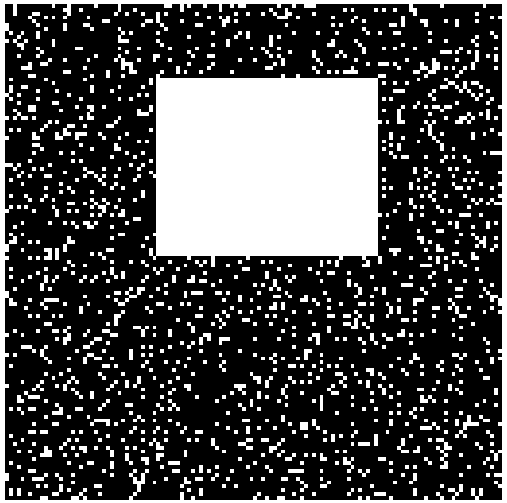
- Option 2: Use the minimum between the two peaks:

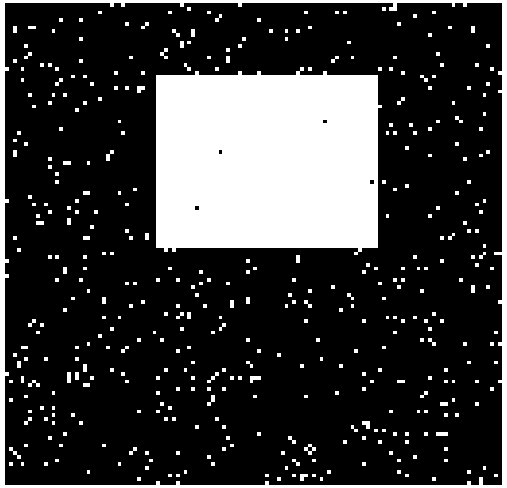
$$t = \arg \min_{v \in [p_1, p_2]} H(v)$$

  - Where $H(v)$ gives the histogram value at $v(i,j)$.
  - In this example:

$$t = \frac{75+125}{2} = 100 \qquad t = \arg \min_{v \in [p_1, p_2]} H(v) = 114$$







4.1.2.2 Otsu's method

- It assumes that the image contains two classes of pixels following a bi-modal histogram. Then, it calculates the optimum threshold separating the two classes according to any of the following two criteria (both criteria are equivalent):
  - **Intra-class (within-class) variance minimization:**
    - Segments should have relatively homogeneous grey values → The threshold must minimize the variance of the grey values within.
  - **Inter-class (between-class) variance maximization:**
    - Alternatively, a threshold that maximizes the variance between objects and background can be selected.

https://en.wikipedia.org/wiki/Otsu%27s_method

4.1.2.2 Otsu's method

- Let us assume that:
  - An $(M \times N)$ image can be represented in L gray levels: $[0,1,\cdots,L-1]$
  - The number of pixels of intensity i is n_i: $MN = \sum_{i=0}^{L-1} n_i$

- The probability of gray level i is denoted by: $p_i = \frac{n_i}{MN}$
  - Normalized histogram → $\sum_{i=0}^{L-1} p_i = 1$

- In a 2-level thresholding, the grey levels are divided into two classes by the threshold t:
  - $C_1(t)$ → With levels $[0,1,\cdots,t]$.
  - $C_2(t)$ → With levels $[t+1,\cdots,L-1]$.

4.1.2.2 Otsu's method

- The grey level probability distributions (weights) for the two classes are:

$$w_1(t) = \Pr\big(C_1(t)\big) = \sum_{i=0}^{t} p_i \qquad w_2(t) = \Pr\big(C_2(t)\big) = \sum_{i=t+1}^{L-1} p_i \qquad w_1(t) + w_2(t) = 1$$

- Their means are:

$$\mu_1(t) = \sum_{i=0}^{t} i\Pr\big(i|C_1(t)\big) = \left\{\begin{matrix}\text{Bayes}'\\ \text{rule}\end{matrix}\right\} = \sum_{i=0}^{t} i\,\frac{\Pr(C_1(t)|i)\Pr(i)}{\Pr\big(C_1(t)\big)} = \frac{1}{w_1(t)}\sum_{i=0}^{t} ip_i$$

$$\mu_2(t) = \sum_{i=t+1}^{L-1} i\Pr\big(i|C_2(t)\big) = \left\{\begin{matrix}\text{Bayes}'\\ \text{rule}\end{matrix}\right\} = \sum_{i=t+1}^{L-1} i\,\frac{\Pr(C_2(t)|i)\Pr(i)}{\Pr\big(C_2(t)\big)} = \frac{1}{w_2(t)}\sum_{i=t+1}^{L-1} ip_i$$

- The total mean of the histogram is: $\mu_T = w_1(t)\mu_1(t) + w_2(t)\mu_2(t) = \sum_{i=0}^{L-1} ip_i$

4.1.2.2 Otsu's method

- The variance of the two classes are:

$$\sigma_1^2(t) = \sum_{i=0}^{t} \big(i - \mu_1(t)\big)^2 \Pr\big(i|C_1(t)\big) = \cdots = \frac{1}{w_1(t)} \sum_{i=0}^{t} \big(i - \mu_1(t)\big)^2 p_i$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{L-1} \big(i - \mu_2(t)\big)^2 \Pr\big(i|C_2(t)\big) = \cdots = \frac{1}{w_2(t)} \sum_{i=t+1}^{L-1} \big(i - \mu_2(t)\big)^2 p_i$$

- The within-class variance (intra-class variance) is:

$$\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

- The between-class variance (inter-class variance) is:

$$\sigma_b^2(t) = \omega_1(t)(\mu_1(t) - \mu_T)^2 + \omega_2(t)(\mu_2(t) - \mu_T)^2$$

4.1.2.2 Otsu's method

- Otsu's method chooses the optimal threshold t by maximizing the between-class variance, which is equivalent to minimizing the within-class variance:

  - The total variance (the sum of the within-class variance and the between-class variance) is constant for different partitions.

$$t = \arg\left\{\max_t\{\sigma_b^2(t)\}\right\} = \arg\left\{\min_t\{\sigma_w^2(t)\}\right\}$$

4.1.2.2 Otsu's method

- **Otsu's method (algorithm):**
  1. Compute the image histogram and the probabilities of each grey level (normalized histogram), p_i.
  2. Step through all possible thresholds: $t = 0, \cdots L-1$
     a. Compute the weights of the classes → $w_i(t)$
     b. Compute the means of the classes → $\mu_i(t)$
     c. Compute the intra-class variance or the inter-class variance → $\sigma_w^2(t)$, $\sigma_b^2(t)$
  3. The desired threshold correspond to:
     - The minimum $\sigma_w^2(t)$
     - The maximum $\sigma_b^2(t)$

4.1.2.2 Otsu's method

- If more than two segments are required, the Otsu's method can be extended to use multiple thresholds.
- Let us assume that we are looking for K segments. Then, we have:
  - $K-1$ thresholds → $\{t_1, t_2, \cdots, t_{K-1}\}$
  - K classes → $\{C_1, C_2, \cdots, C_K\}$
- The solution of the method is as follows:

$$\{t_1, t_2, \cdots, t_{K-1}\} = \arg\left\{\max_t\{\sigma_b^2(t_1, t_2, \cdots, t_{K-1})\}\right\} = \arg\left\{\min_t\{\sigma_w^2(t_1, t_2, \cdots, t_{k-1})\}\right\}$$

$$w_k(t) = \sum_{i=t_{k-1}+1}^{t_K} p_i \qquad \mu_k(t) = \frac{1}{w_k(t)} \sum_{i=t_{k-1}+1}^{t_k} i p_i \qquad \sigma_k^2(t) = \frac{1}{w_k(t)} \sum_{i=t_{k-1}+1}^{t_k} \left(i - \mu_k(t)\right)^2 p_i$$

$$\sigma_\omega^2(t) = \sum_{k=1}^{K} \omega_k(t)\sigma_k^2(t)$$

$$\sigma_b^2(t) = \sum_{k=1}^{K} \omega_k(t)(\mu_k(t) - \mu_T)^2$$

4.1.2.2 Otsu's method

- If more than two segments are required, the Otsu's method can be extended to use multiple thresholds.
  - Multidimensional optimization is required → Complex and computationally expensive.
  - More practical solution → K-means algorithm.

4.1.2.3 K-means algorithm

- Its objective is to divide an image into K segments (using $K-1$ thresholds), minimizing the total within-segment variance.

  - As previously, the within-segment variance is defined as:

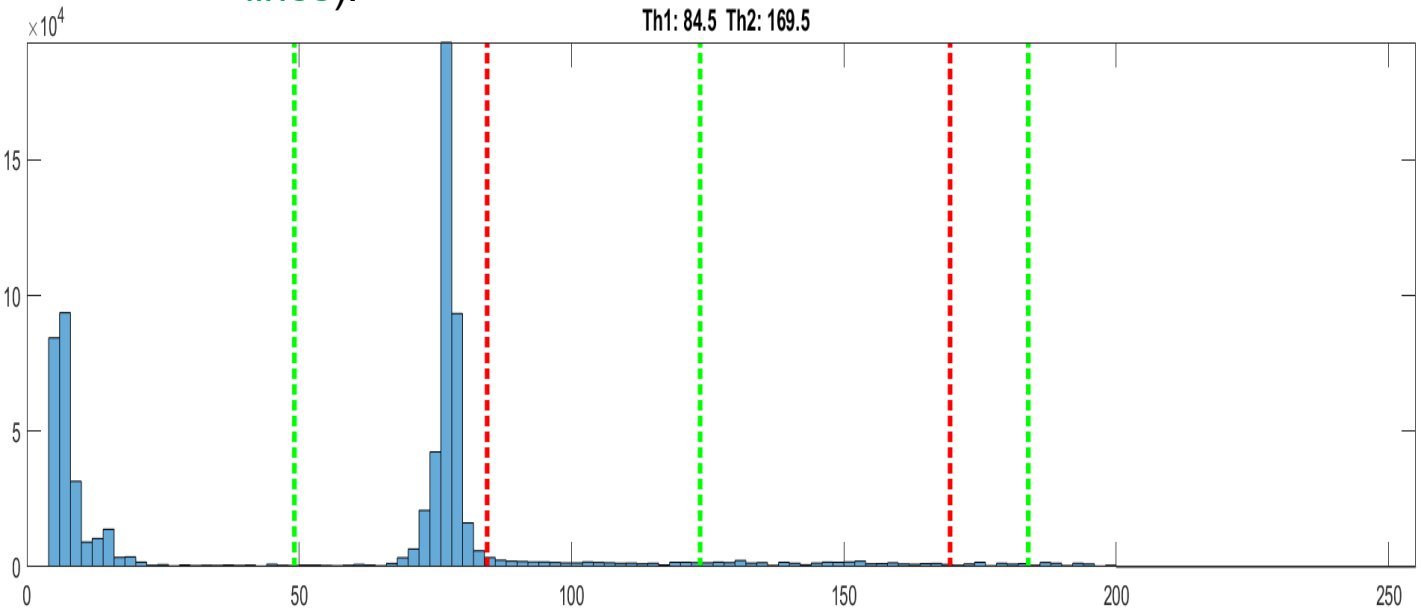
$$\sigma_\omega^2(t) = \sum_{k=1}^{K} \omega_k(t)\sigma_k^2(t)$$

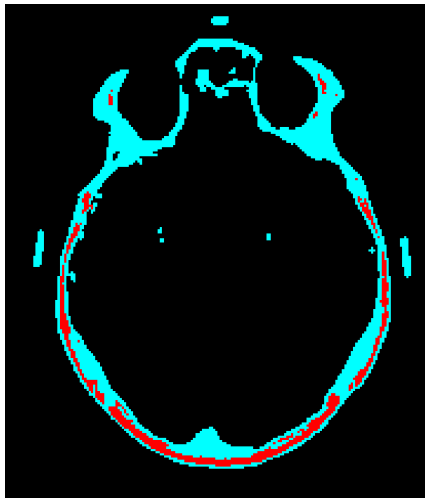
    - Where:

$$w_k(t) = \sum_{i=t_{k-1}+1}^{t_K} p_i \qquad \mu_k(t) = \frac{1}{w_k(t)} \sum_{i=t_{k-1}+1}^{t_k} i p_i \qquad \sigma_k^2(t) = \frac{1}{w_k(t)} \sum_{i=t_{k-1}+1}^{t_k} \left(i - \mu_k(t)\right)^2 p_i$$

4.1.2.3 K-means algorithm

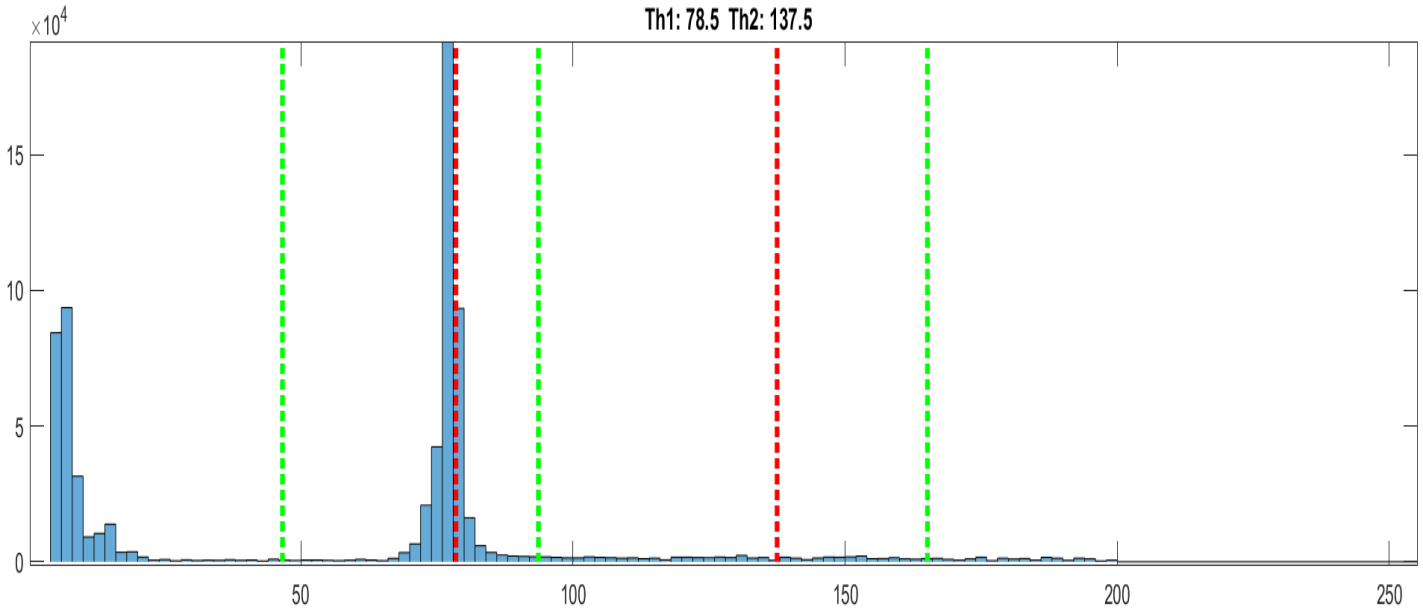
1. Initialization:
   a) Distribute the $K-1$ thresholds over the histogram (red lines) → $K=3$ in the example.
   b) Segment the image according such thresholds.
   c) For each segment, compute the mean pixel value (green lines).

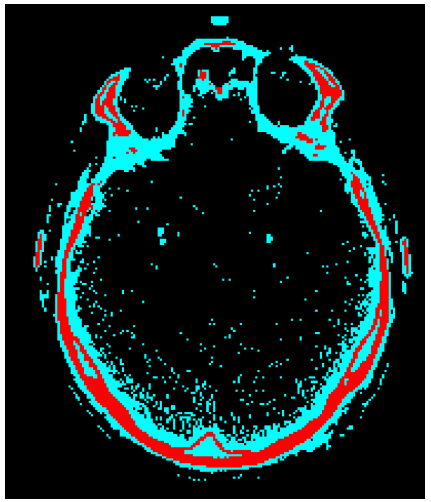




4.1.2.3 K-means algorithm

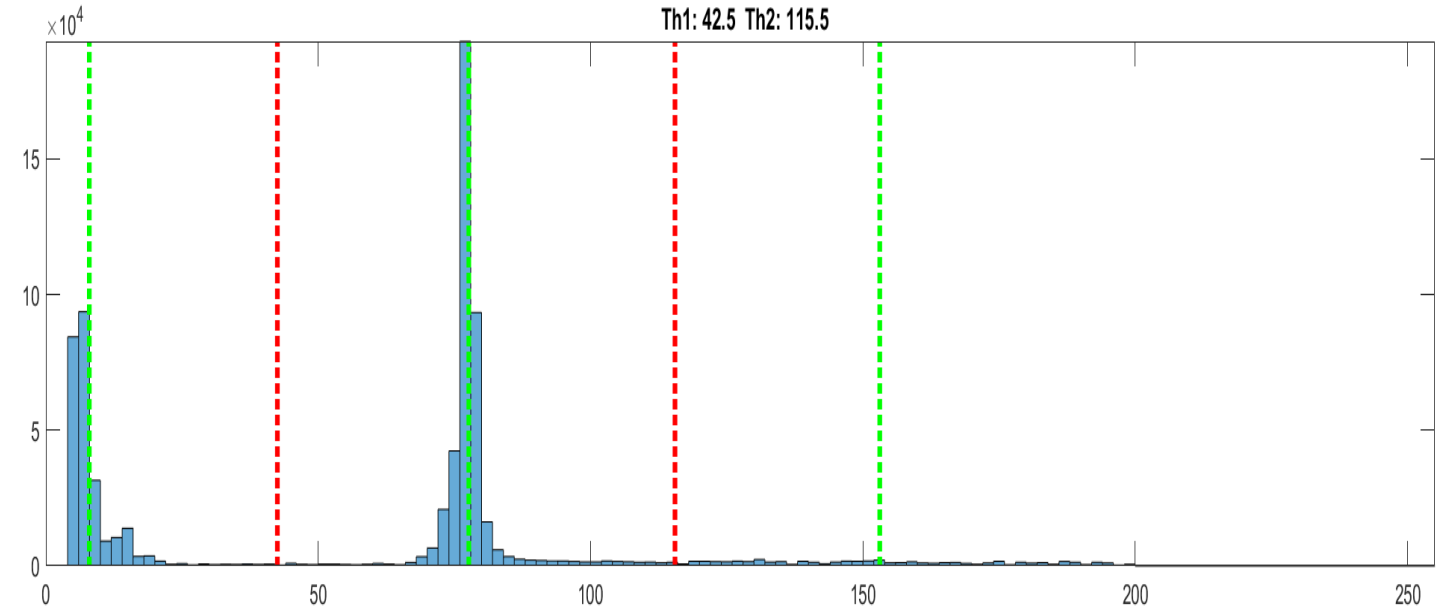
2. Reset the cluster centers to the computed mean values.
3. Reset the thresholds to be midway the cluster centers (red lines).
4. Segment the image and compute new means (green lines).
5. Go to step 2. Iterate until cluster centers do not move.





4.1.2.3 K-means algorithm

- In the illustrated example, the final segmentation has been reached in 9 iterations.
- It can be observed that the result is successful.

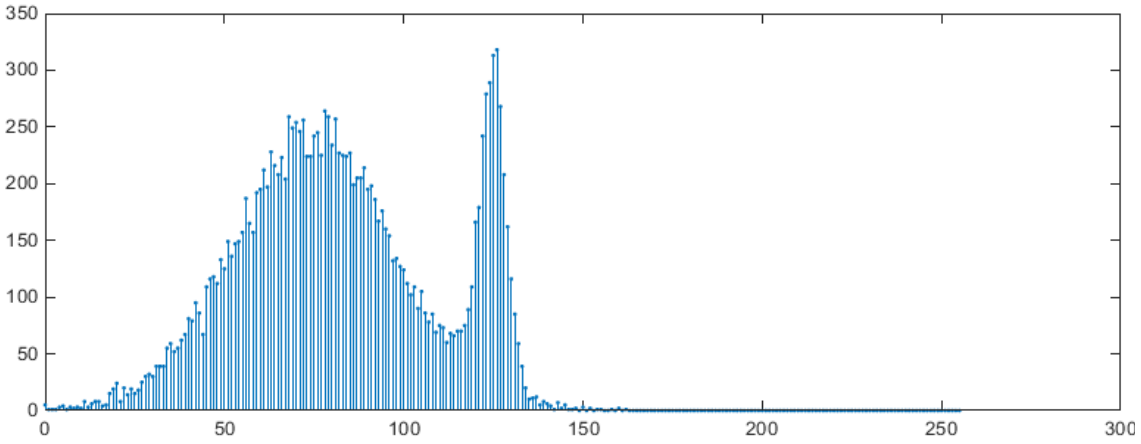


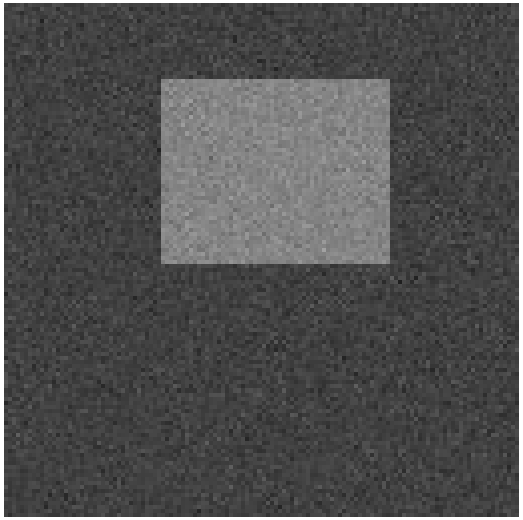
4.1.2.3 K-means algorithm

- **Drawbacks:**
  - The value of K must be set before running the algorithm.
    - What is the adequate one?
  - The result depends on the initialization.
    - What is the adequate one?

4.1.2.4 Optimal thresholding

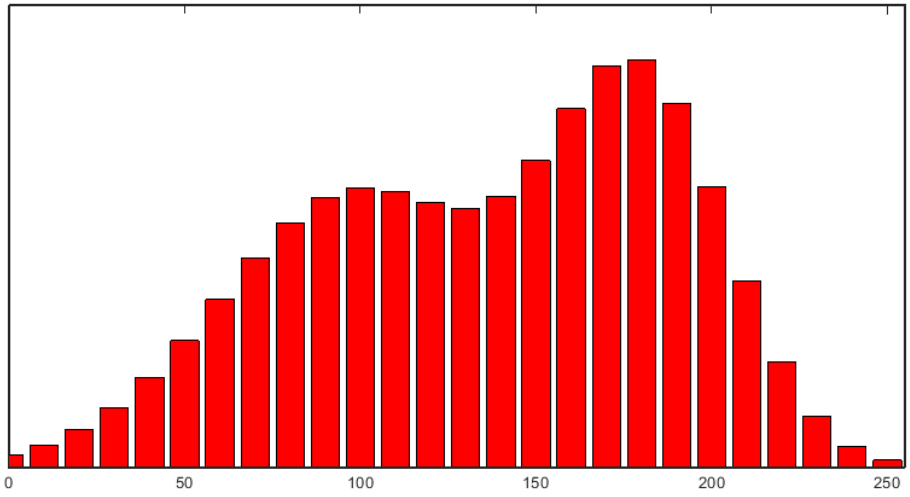
- The histogram of an object that theoretically has a uniform grey value typically shows a distribution of grey values.
- Often, this distribution can be approximated using a known statistical distributions (e.g., as the Gaussian).
- Optimal thresholding → It is a technique that approximates the histogram using a weighted sum of distribution functions.
    - Then, it sets a threshold to maximize the number of correctly segmented pixels.

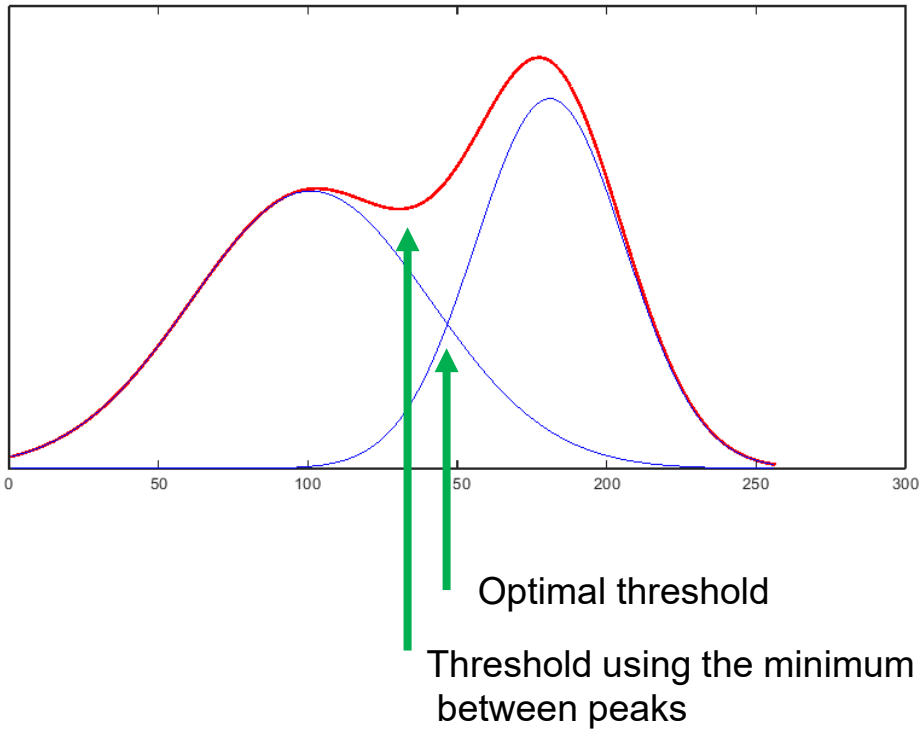




4.1.2.4 Optimal thresholding

- Example of optimal thresholding:
  - Left: Image histogram.
  - Right: Approximation of the histogram using two Gaussians.





4.1.2.4 Optimal thresholding

- Example of optimal thresholding:
  - The two Gaussians are obtained by minimizing the sum of squared distances between the histogram data and the fitted curve:

$$\text{minimize} \sum_i \big(h(v) - m(v)\big)^2$$

  - Where $h(v)$ is the normalized histogram and $m(v)$ is the chosen model for the histogram.
  - In this example, the model is:

$$m(v) = P_0 N(v, \mu_0, \sigma_0^2) + P_1 N(v, \mu_1, \sigma_1^2) = (1 - P_1) N(v, \mu_0, \sigma_0^2) + P_1 N(v, \mu_1, \sigma_1^2)$$

    - Where P_0 and P_1 are the probabilities that a pixel belongs to segment 0 or 1, respectively.

4.1.2.4 Optimal thresholding

- Example of optimal thresholding:
  - The optimal threshold, t, is the grey value where the two Gaussian curves intersect:

$$(1 - P_1)N(t, \mu_0, \sigma_0^2) = P_1 N(t, \mu_1, \sigma_1^2)$$

  - <u>Demonstration:</u>
    - The normalized histogram is approximated by:

$$h(v) \approx (1 - P_1)N(v, \mu_0, \sigma_0^2) + P_1 N(v, \mu_1, \sigma_1^2)$$

    - Where:
      - $N(v, \mu_0, \sigma_0^2)$ → Distribution of pixels in segment 0.
      - $N(v, \mu_1, \sigma_1^2)$ → Distribution of pixels in segment 1.
      - $1 - P_1$ → Fraction of pixels in segment 0.
      - P_1 → Fraction of pixels in segment 1.

4.1.2.4 Optimal thresholding

- If the image is segmented using a certain threshold t:
  - The fraction of pixels in segment 0 misclassified as part of segment 1 is:

$$(1 - P_1) \int_t^{\infty} N(v, \mu_0, \sigma_0^2) dv$$

  - The fraction of pixels in segment 1 misclassified as part of segment 0 is:

$$P_1 \int_{-\infty}^{t} N(v, \mu_1, \sigma_1^2) dv$$

- The total error is the sum of these partial errors:

$$E(t) = (1 - P_1) \int_t^{\infty} N(v, \mu_0, \sigma_0^2) dv + P_1 \int_{-\infty}^{t} N(v, \mu_1, \sigma_1^2) dv$$

4.1.2.4 Optimal thresholding

- This error is minimal when its derivative is zero:

$$E'(t) = 0$$

$$E'(t) = (1 - P_1)[N(v, \mu_0, \sigma_0^2)]_t^{\infty} + P_1[N(v, \mu_1, \sigma_1^2)]_{-\infty}^{t}$$

$$E'(t) = (1 - P_1)\left(N(\infty, \mu_0, \sigma_0^2) - N(t, \mu_0, \sigma_0^2)\right) + P_1\left(N(t, \mu_1, \sigma_1^2) - N(-\infty, \mu_1, \sigma_1^2)\right)$$

$$E'(t) = (1 - P_1)\left(0 - N(t, \mu_0, \sigma_0^2)\right) + P_1(N(t, \mu_1, \sigma_1^2) - 0)$$

$$E'(t) = -(1 - P_1)N(t, \mu_0, \sigma_0^2) + P_1 N(t, \mu_1, \sigma_1^2) = 0$$

$$\boxed{(1 - P_1)N(t, \mu_0, \sigma_0^2) = P_1 N(t, \mu_1, \sigma_1^2)}$$

4.1.2.4 Optimal thresholding

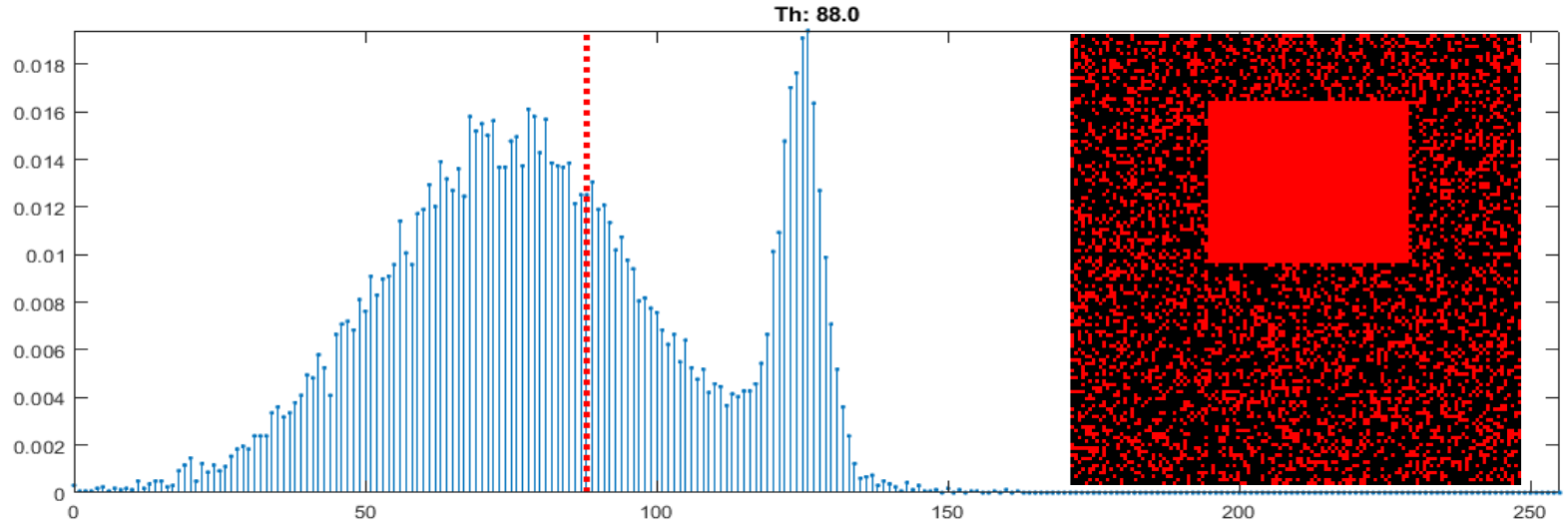
- Optimal thresholding can be extended to use multiple thresholds:
  - The histogram is approximated using a sum of more than two Gaussians.

$$h(v) \approx \sum_{i=0}^{k-1} P_i N\left(v, \mu_i, \sigma_i^2\right)$$

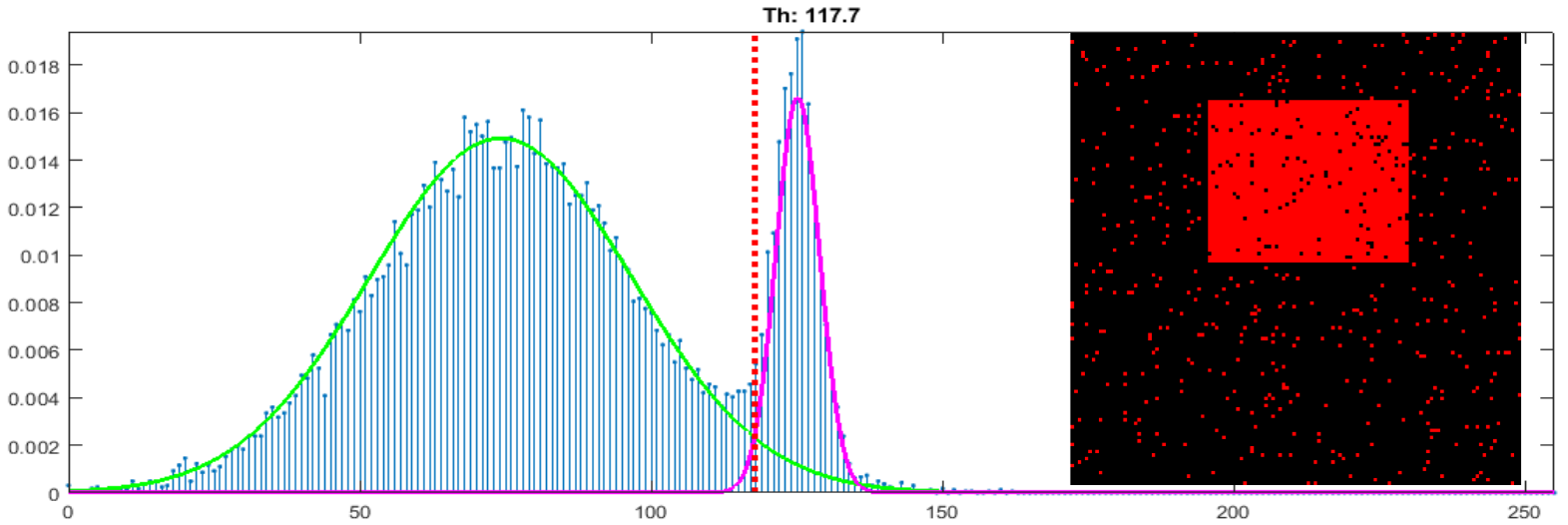
  - Problems:
    - The addition of each Gaussian adds three parameters to be estimated.
    - Selecting an adequate number of Gaussians is a difficult problem.

4.1.2.4 Optimal thresholding

Otsu's method



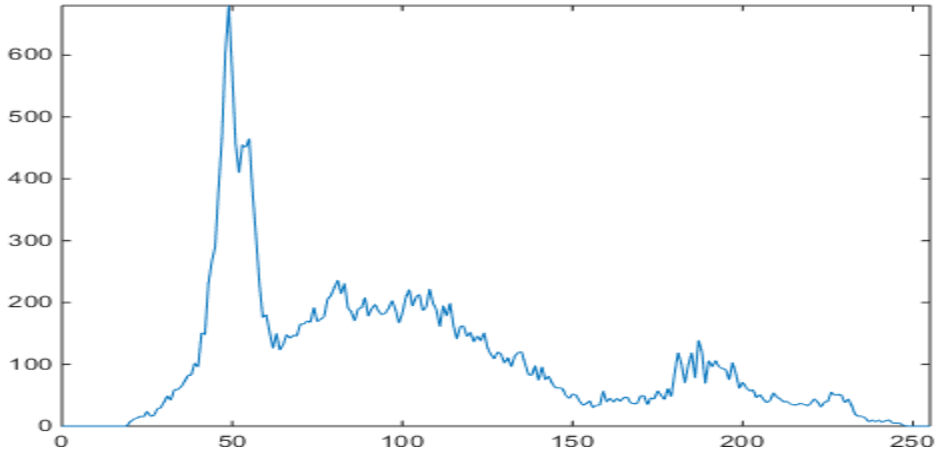
Optimal thresholding



4.1.2.5 Histogram estimation

- Estimation of the histogram → A smooth curve that best fits the real image histogram.
- Optimal thresholding method → It uses an estimation of the histogram.
- Other methods (e.g., those requiring finding extrema) → They also benefit from using a smooth estimate.
  - The chance of getting stuck in local extrema is decreased.
  - The process is less susceptible to noise.

Conclusion: The histogram estimation is very useful in many cases.



4.1.2.5 Histogram estimation

- **Parametric estimation:**
  - We assume that the histogram follows a specific distribution that depends on a fixed number of parameters.
  - Example: The mixture of two Gaussians used to describe the optimal thresholding method.
  - Problem: Not all the histograms can be successfully approximated using mixtures of known distributions.

- **Nonparametric estimation:**
  - The histogram is estimated directly from the data, without any assumptions about the underlying distribution.
  - It avoids having to choose a model and estimating its parameters.

4.1.2.5 Histogram estimation

- **Kernel Density Estimation (KDE):**
  - It is the most frequently used nonparametric modeling method.
  - It estimates the histogram by a sum of local functions (kernels, K) centered at each sample.
  - Typical kernels: Gaussian, triangular and rectangular.
  - Each distribution is identical to all others.
  - The only parameter that must be stablished is their variance.

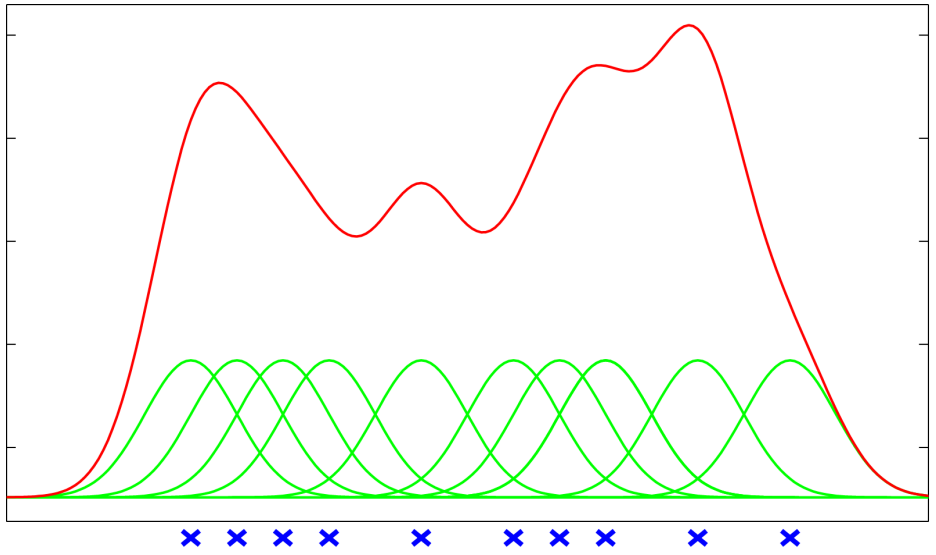
$$f(v) = \frac{1}{N}\sum_{i=1}^{N} K(v - v_i)$$

$$f(v) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(v - v_i)^2}{2\sigma^2}\right)$$

N → Number of samples.

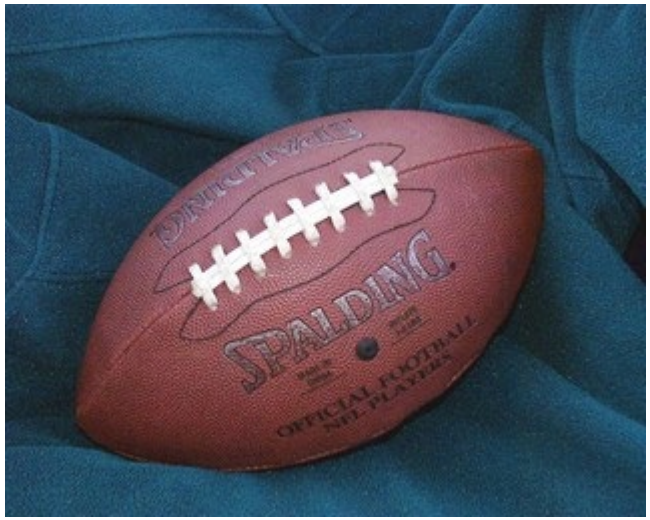
σ^2 → Variance of the kernels.

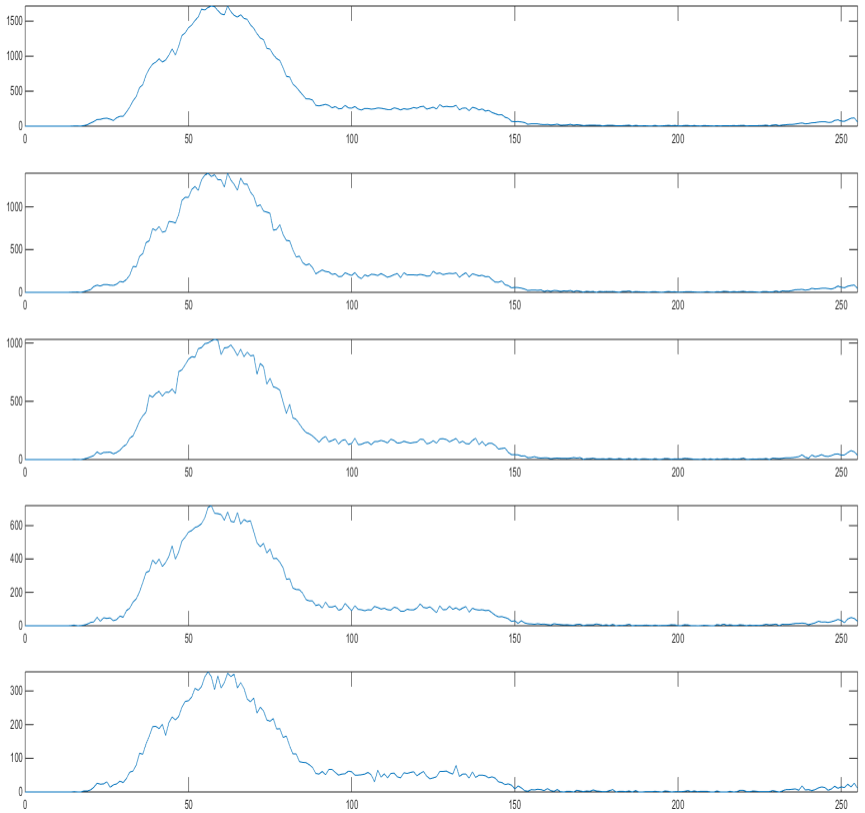
v_i → Grey value of each sample.



4.1.2.5 Histogram estimation

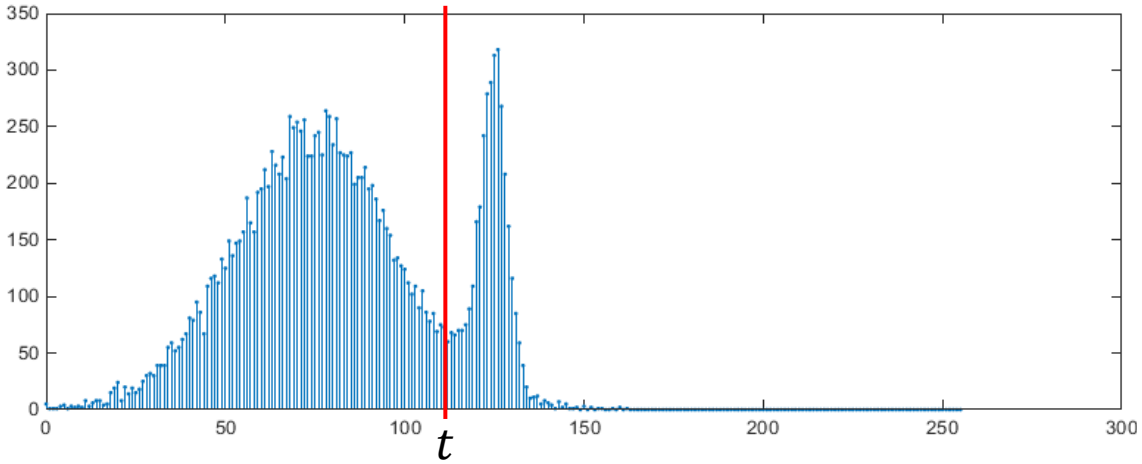
- The estimation of the histogram often does not require to use all the available data.
  - A small fraction of the data is usually sufficient for a good estimation.
  - This will allow reducing the computational cost in the estimation.

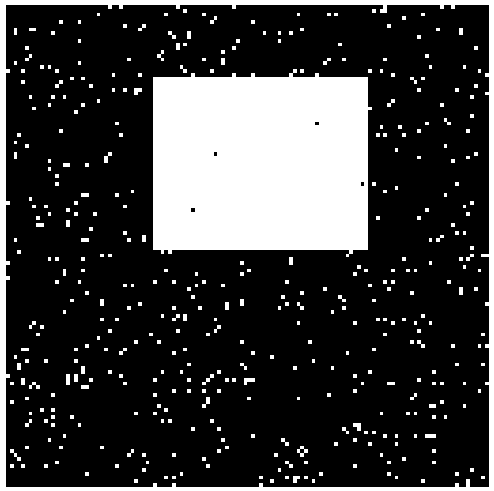




4.1.2.6 Enhancing threshold segmentation

- In all the previously described thresholding methods we have not considered the spatial context of the data.
- However, there are many cases where the humans segment the observed objects based on their spatial context.
- Example:
  - The noise can cause many small segments that can not physically exists (see erroneous segments and object holes in the image).





4.1.2.6 Enhancing threshold segmentation

- There exist many techniques to address these problems, which can be classified into three categories:
  - Pre-processing → Processing the original image prior segmentation.
  - Post-processing → Processing the segmented image.
  - Adapt the segmentation process.

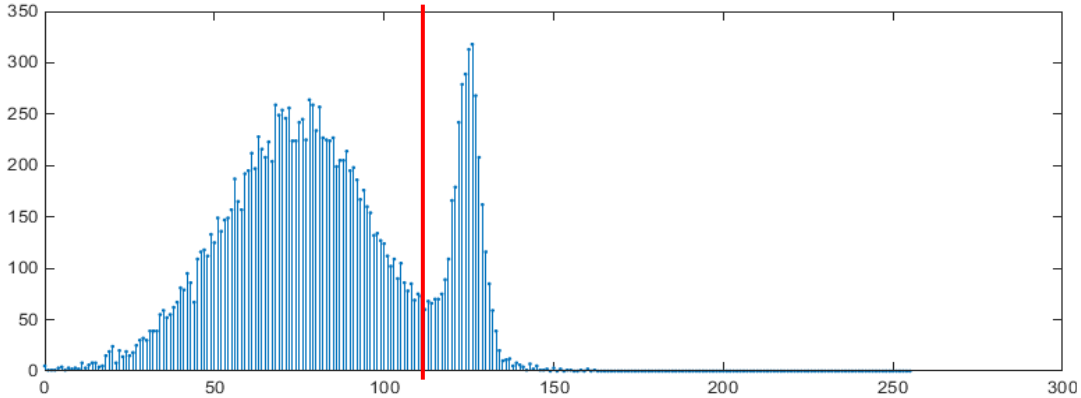
- It is also possible to combine more than one of these techniques in a single application.

- Examples of typical approaches:
  - Non-linear filters, such as the median filter.
  - Morphological operators: closing to remove holes and opening to remove small segments.
  - Search for isolated pixels (or segments below a certain area) and set their grey value to the value of their neighborhood.

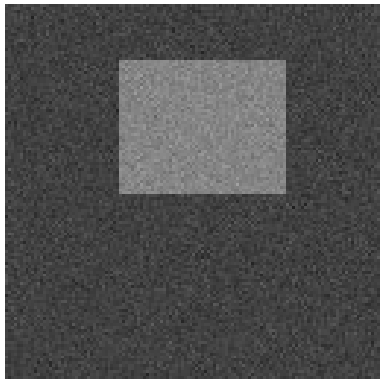
4.1.2.6 Enhancing threshold segmentation

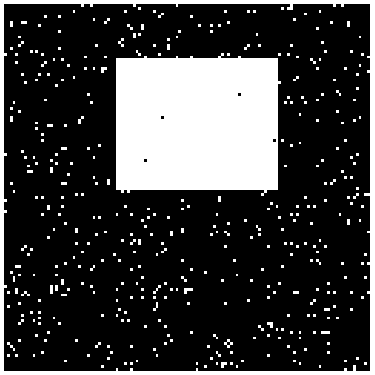
- **Post processing using a median filter:**

  - Each output pixel contains the median value in a $n \times n$ neighborhood around the corresponding pixel in the input image.

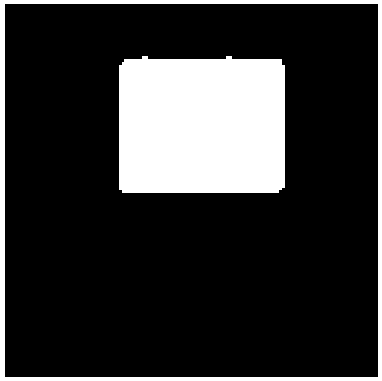


$t = 114$





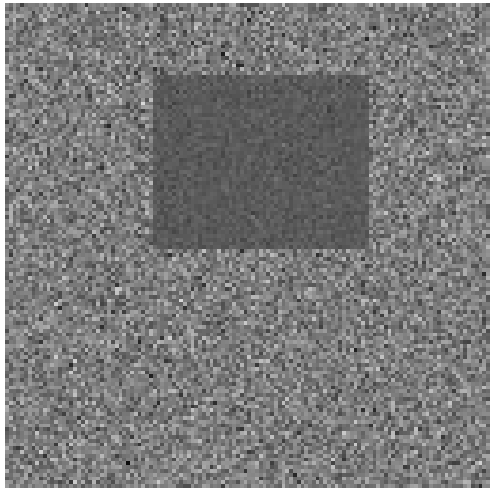
$n = 3$

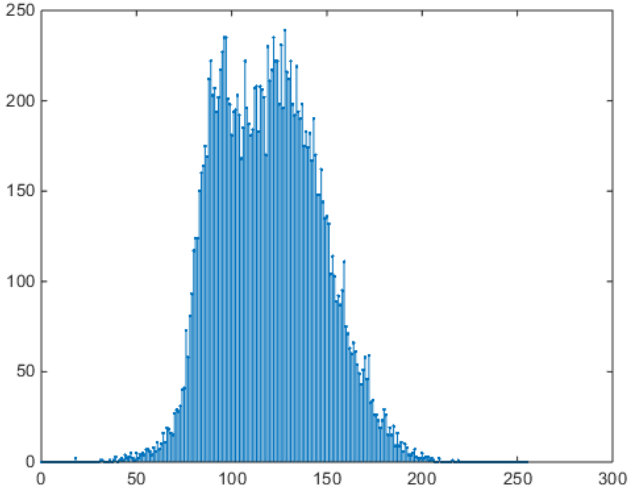


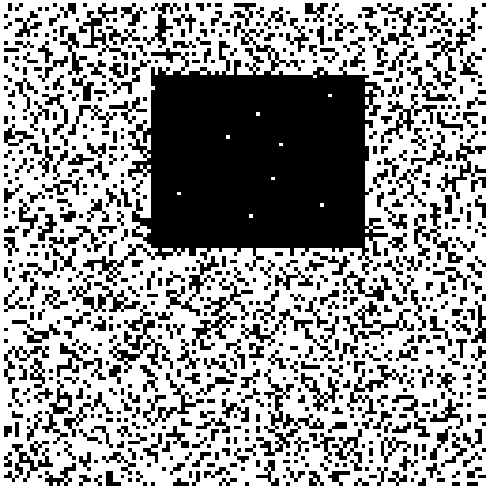
$n = 5$

4.1.2.6 Enhancing threshold segmentation

- **Pre and post processing using morphology:**



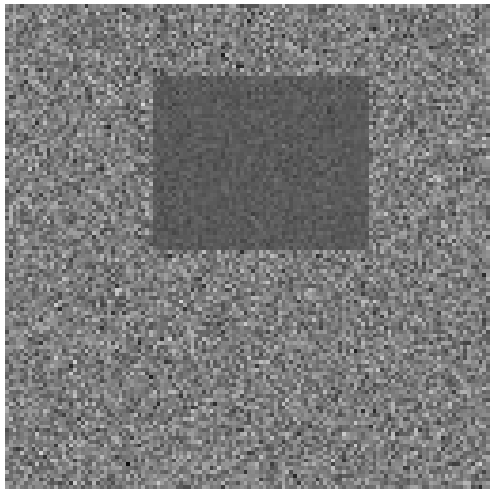


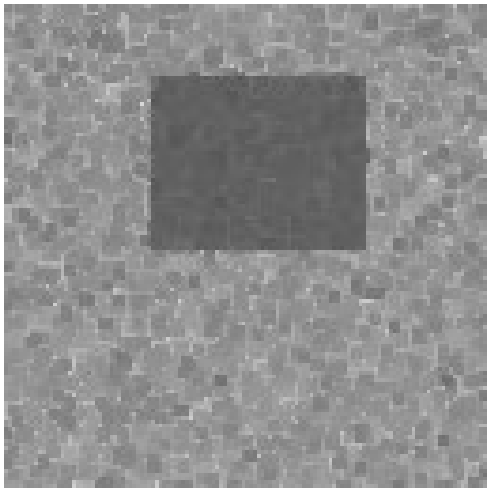


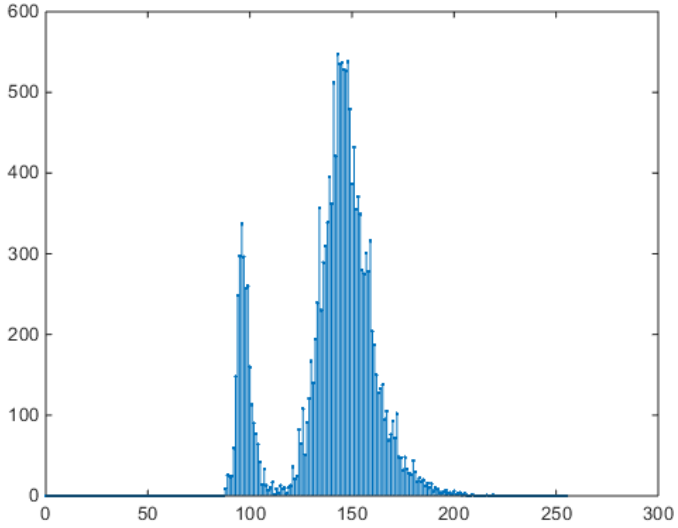
  - This is the result if we apply a 3×3 median filter in a post processing stage.

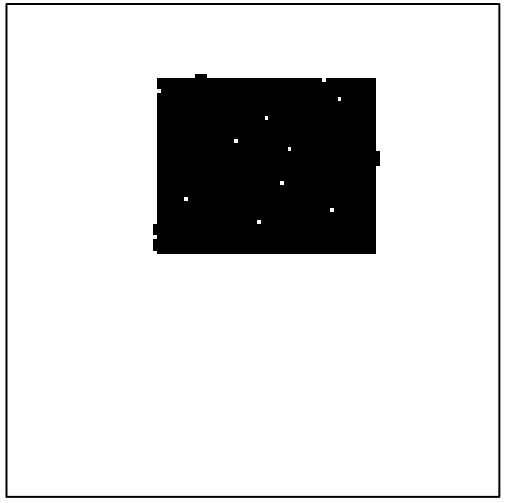
4.1.2.6 Enhancing threshold segmentation

- **Pre and post processing using morphology:**





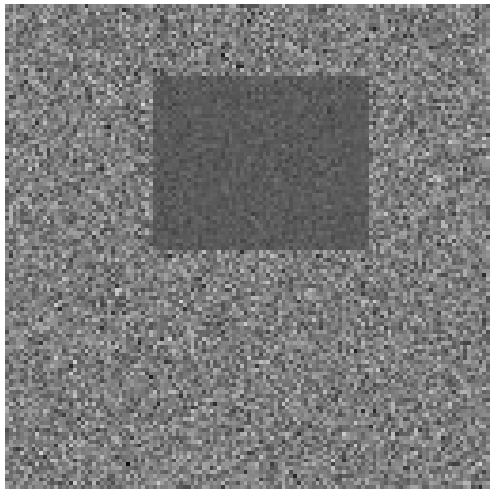


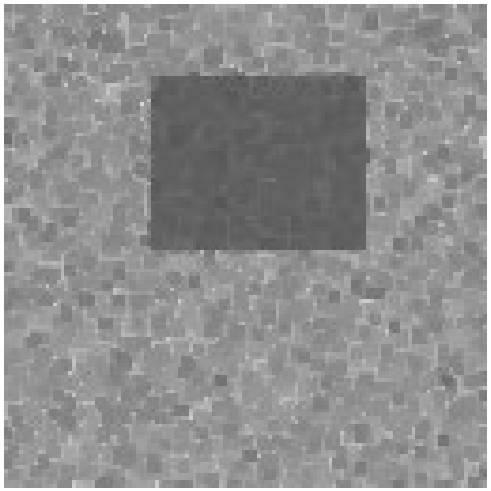


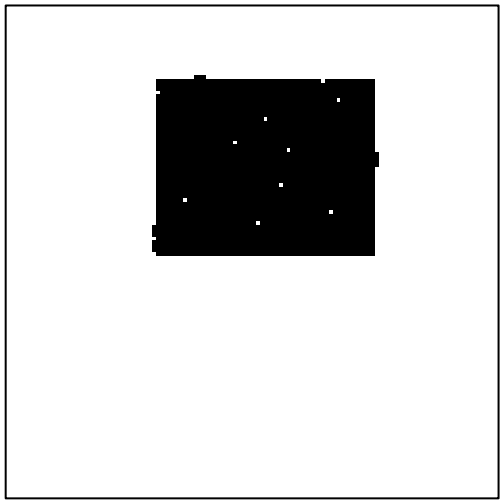
- Pre-processing: First, we apply a 3×3 closing operation (dilation followed by an erosion) on the original image.
- The result is a new image with a histogram easier to analyze.
- The noisy segments have been removed.

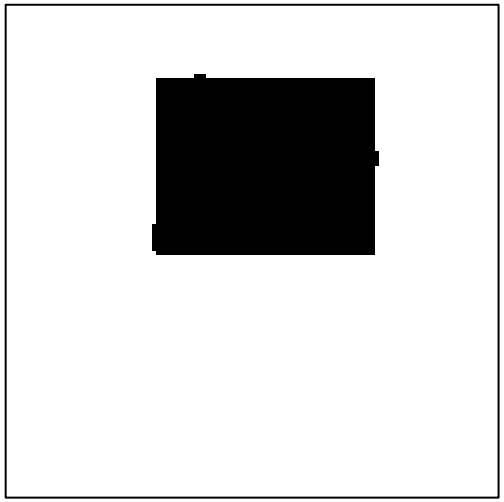
4.1.2.6 Enhancing threshold segmentation

- **Pre and post processing using morphology:**





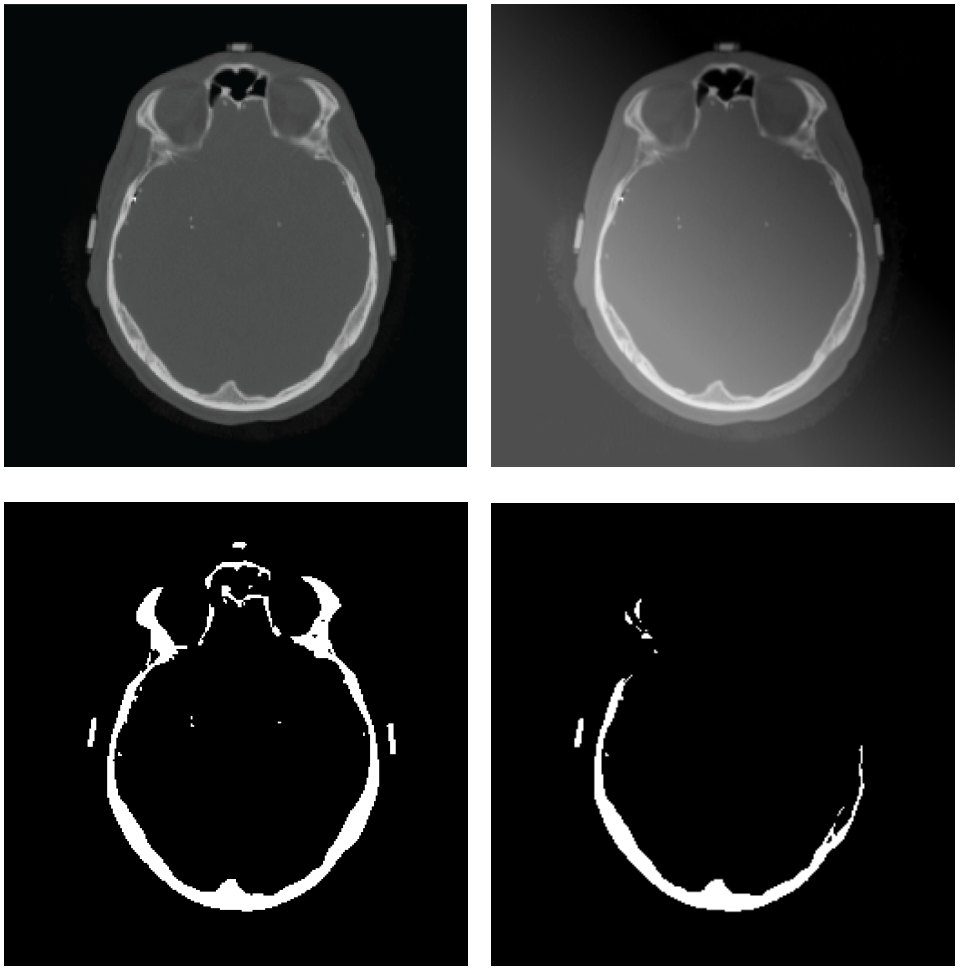




  - Post processing: Finally, we apply a 3×3 opening operation (erosion followed by a dilation) **on the segmented image**.
  - The result is a segmented image in which the holes have been removed.

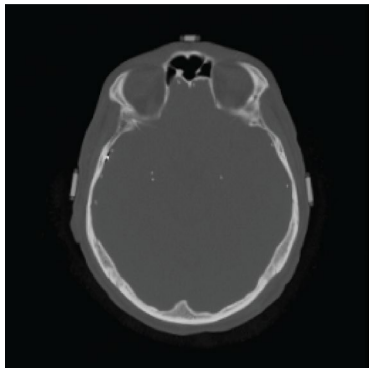
4.1.2.6 Enhancing threshold segmentation

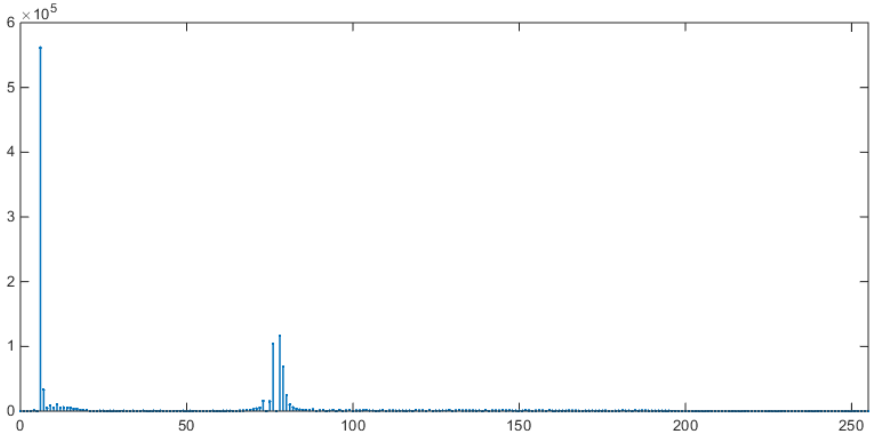
- **Local thresholding:**
  - All the thresholding techniques previously presented apply global thresholds.
    - That is, the thresholds are the same for the entire images.
  - However, this does not provide successful results in some images.
    - For example, images that do not have a uniform illumination.

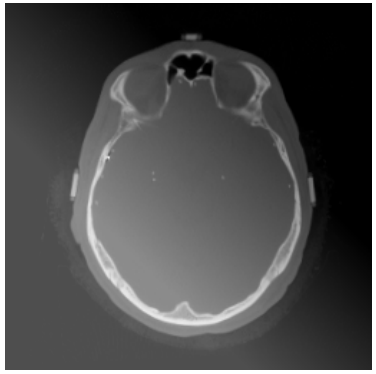


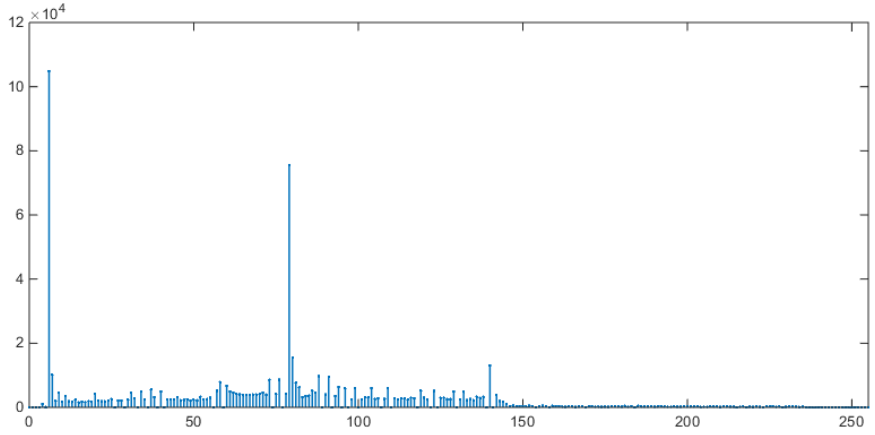
4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**



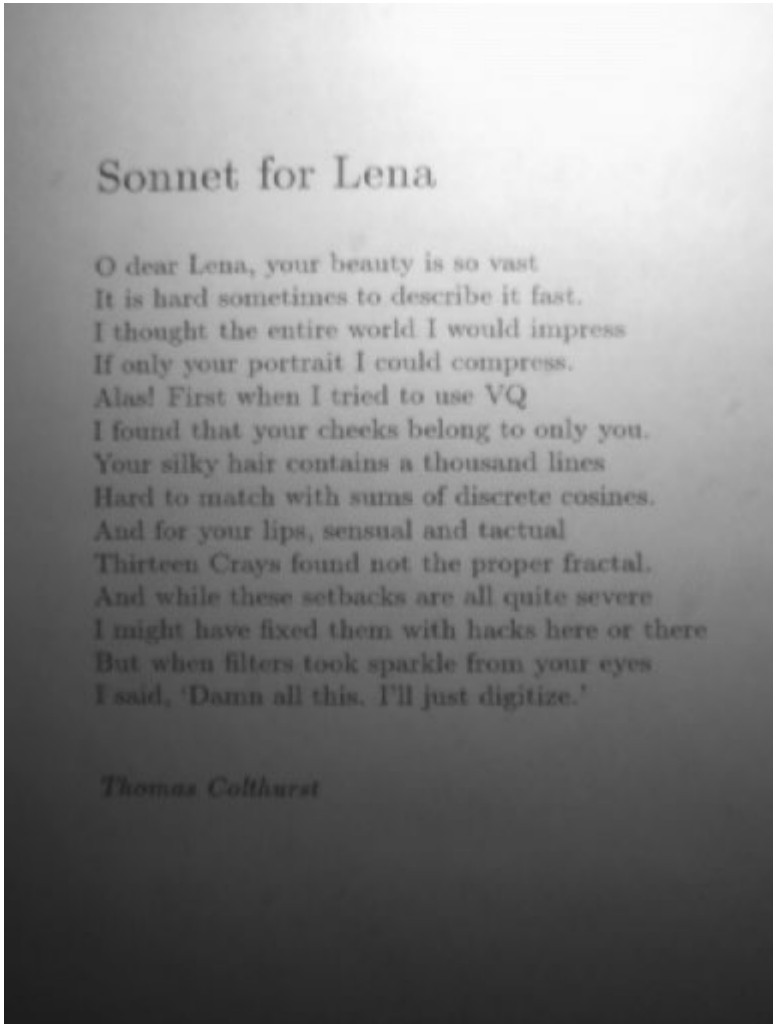


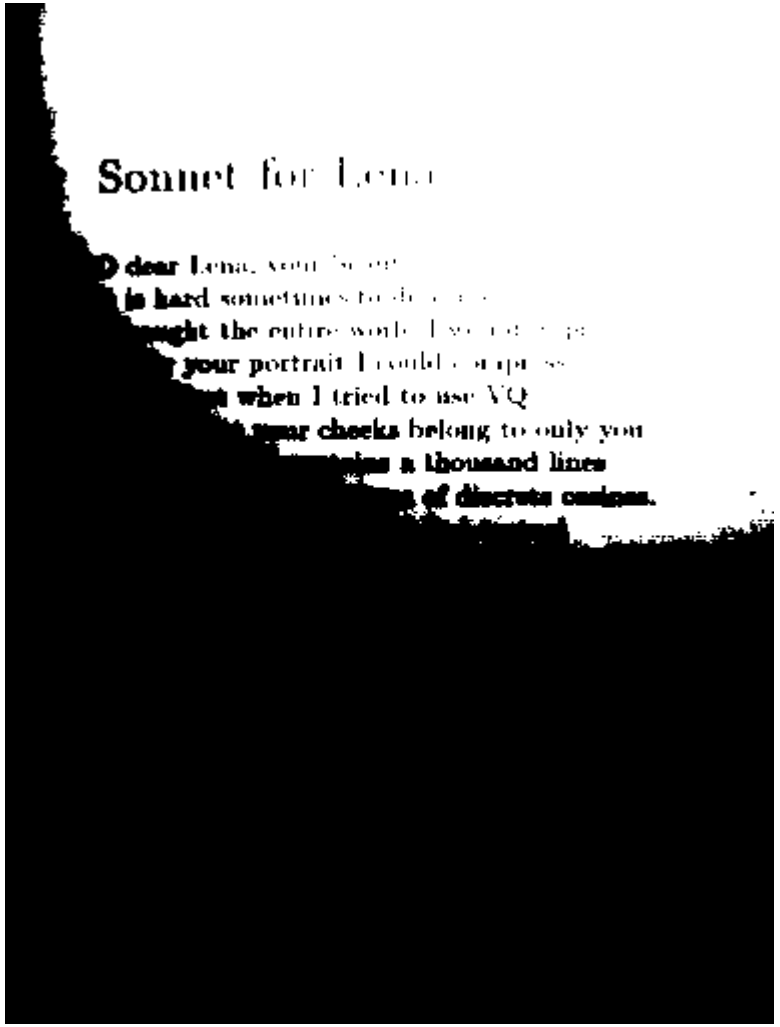




4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**



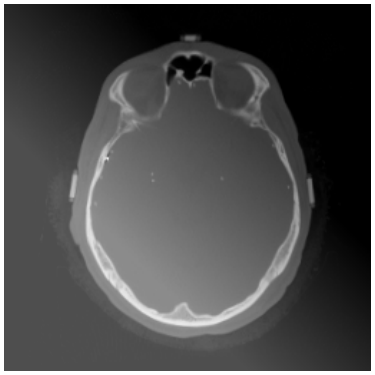


4.1.2.6 Enhancing threshold segmentation

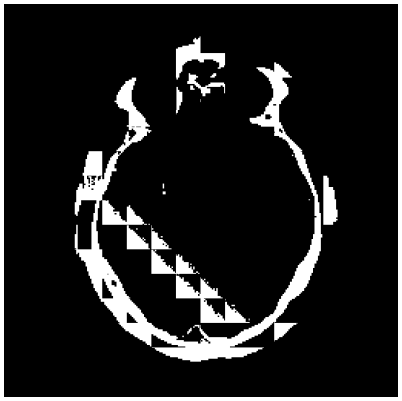
- **Local thresholding:**
  - This problem can be solved by tiling the images into subimages.
    - The grey value of the object to segment must be relatively constant in each subimage.
    - An appropriate threshold could be found for each subimage.
    - The final segmentation will be the result of merging all the segmentation results.
  - Problems:
    - How many subimages?
    - How many thresholds per subimage?
      - It could be necessary to apply different techniques in the different subimages.

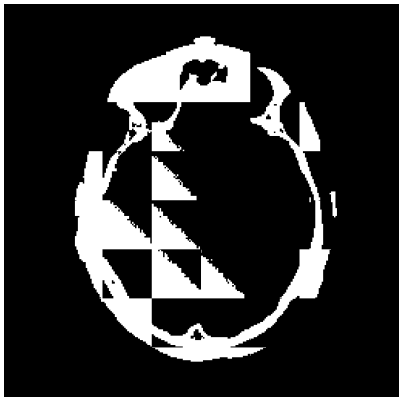
4.1.2.6 Enhancing threshold segmentation

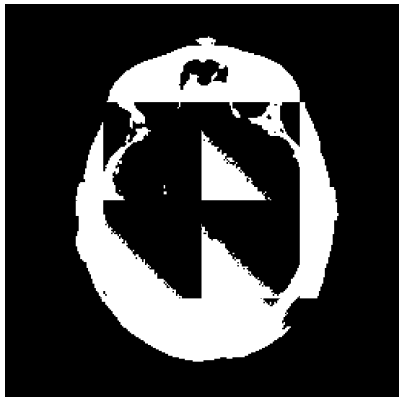
- **Local thresholding:**
  - Example - Image with 3 gray levels and large segments:



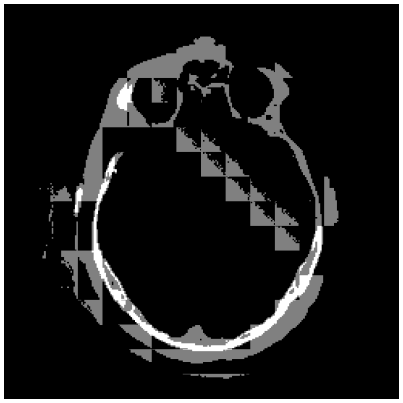
$k = 2$

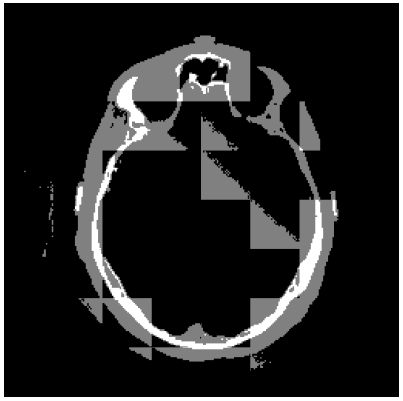


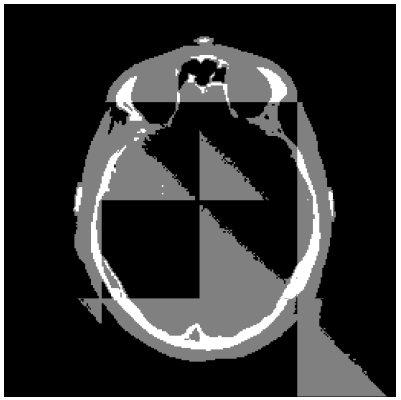




$k = 3$



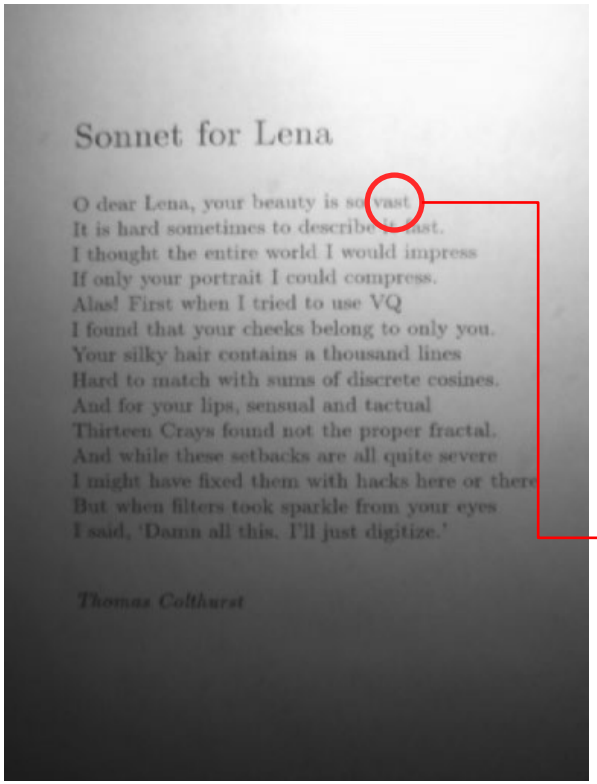


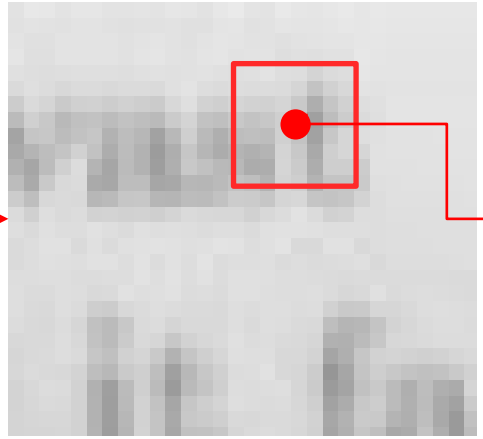


50×50 100×100 200×200

4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**
  - Example - Image with 2 gray levels and small segments:

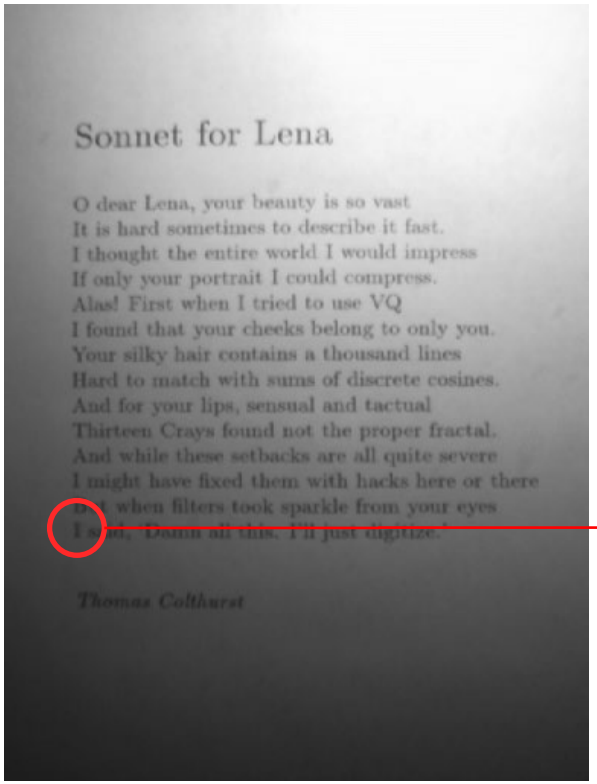




  - Let us analyze one pixel on a character in the darker area of the image:
    - Pixel value: $p = 161$
    - Mean value (7×7 neighborhood): $\mu = 203$
    - ➢ Final classification:
      $p < \mu \rightarrow 0$
      $p \geq \mu \rightarrow 1$

4.1.2.6 Enhancing threshold segmentation

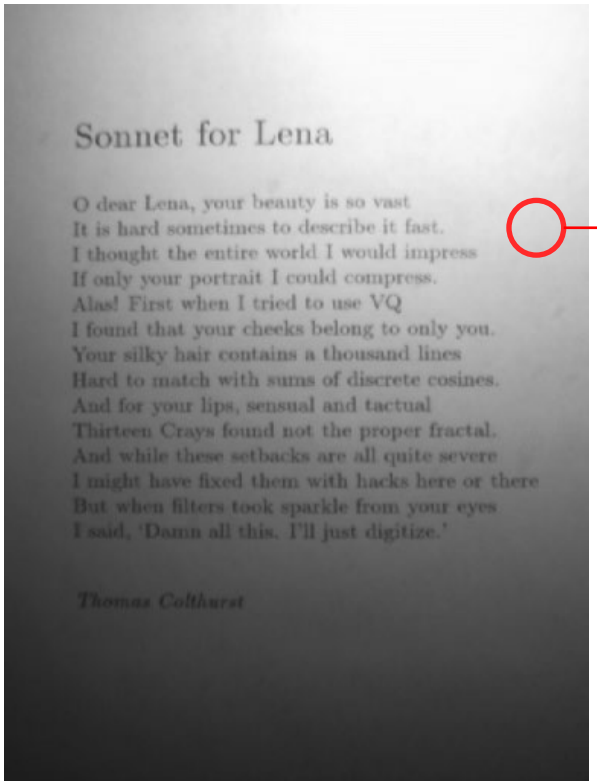
- **Local thresholding:**
  - Example - Image with 2 gray levels and small segments:

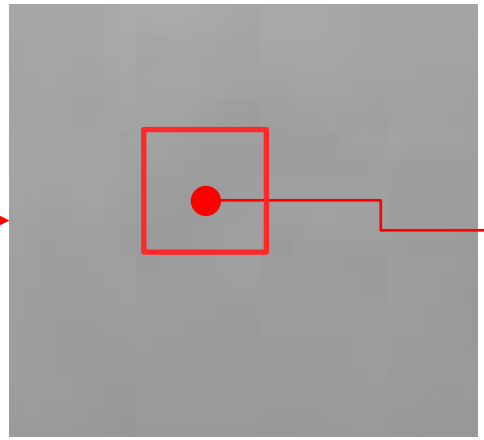


  - Let us analyze one pixel on a character in the lighter area of the image:
    - Pixel value: $p = 51$
    - Mean value (7×7 neighborhood): $\mu = 61$
    - ➢ Final classification:
      $p < \mu \rightarrow 0$
      $p \geq \mu \rightarrow 1$

4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**
  - Example - Image with 2 gray levels and small segments:



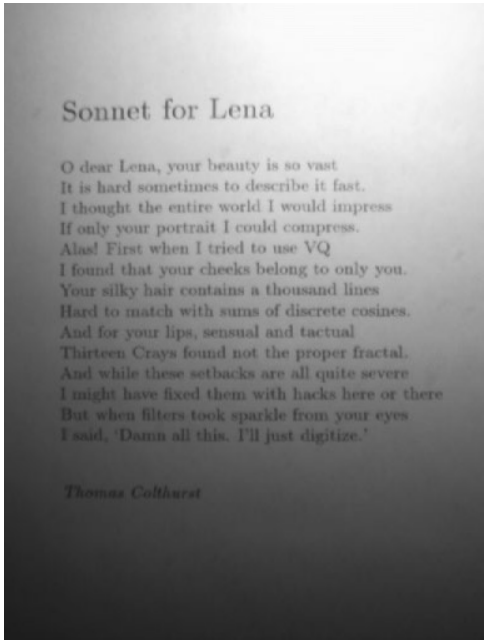


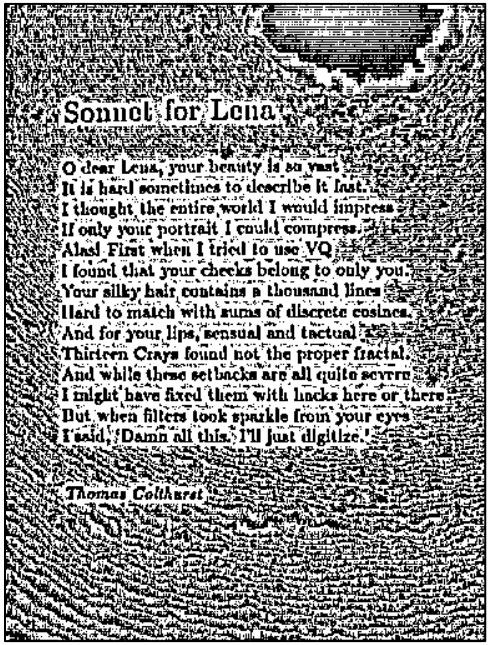
  - What is about a background pixel?
    - Pixel value: $p = 239$
    - Mean value (7×7 neighborhood): $\mu = 239.8$
  - ➢ Final classification:
    - $p < \mu \rightarrow 0$
    - $p \geq \mu \rightarrow 1$

Random classification!!!

4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**
  - Example - Image with 2 gray levels and small segments:

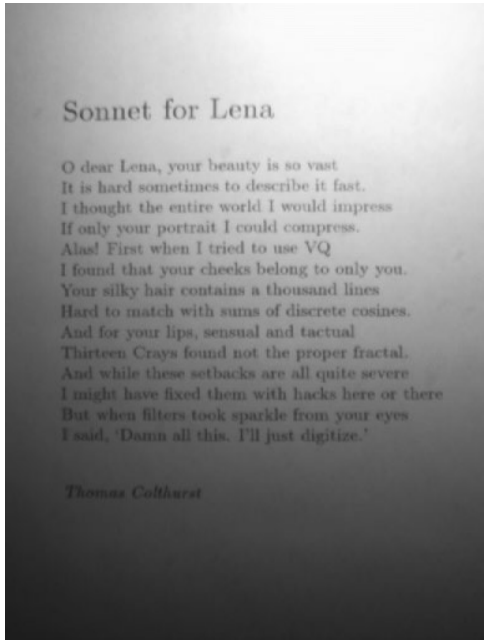


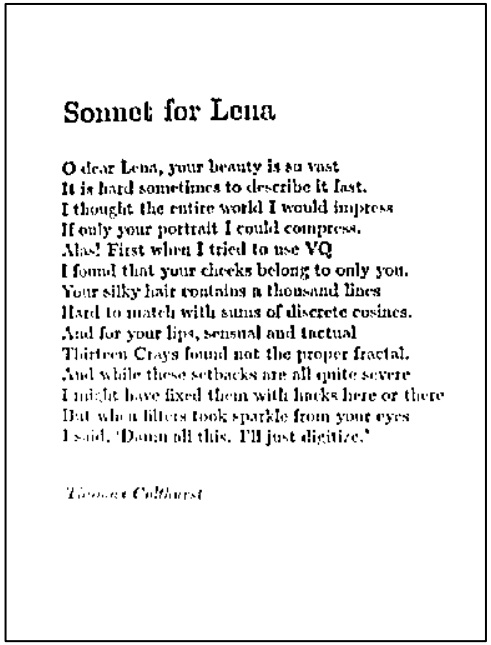


  - The threshold applied to each pixel is the mean of a 7×7 neighborhood.
  - Good results in the area surrounding the text → There are enough foreground and background pixels in the local neighborhood of each pixel.
  - Bad results in too uniform areas → The range of intensity values is very small, and their mean is close to the value of the center pixel

4.1.2.6 Enhancing threshold segmentation

- **Local thresholding:**
  - Example - Image with 2 gray levels and small segments:





POSSIBLE SOLUTION:

- These errors can be improved if the threshold employed is not the mean, but **the mean minus a constant**, C.
- In this way, all the pixels in uniform areas will be set as background.

- This result has been obtained by thresholding each pixel with the mean of its 7×7 neighborhood minus $C = 7$.

4.1.2.7 Case of use – Soccer field segmentation

- **Objective:**
  - Separate the soccer field from the rest of elements in the image (stands and other non-relevant areas) → This will make it easier the accomplishment of the subsequent stages (e.g., field modeling, players tracking…).

More info in: C. Cuevas, D. Berjón, N. García,
"A fully automatic method for segmentation of soccer playing fields",
Scientific Reports, vol. 13, article 1464, pp. 1-17, Jan. 2023.
(doi: 10.1038/s41598-023-28658-1)

4.1.2.7 Case of use – Soccer field segmentation

- **Initial analysis:**
  - What characteristic of the field is more representative for the segmentation?

    Its color

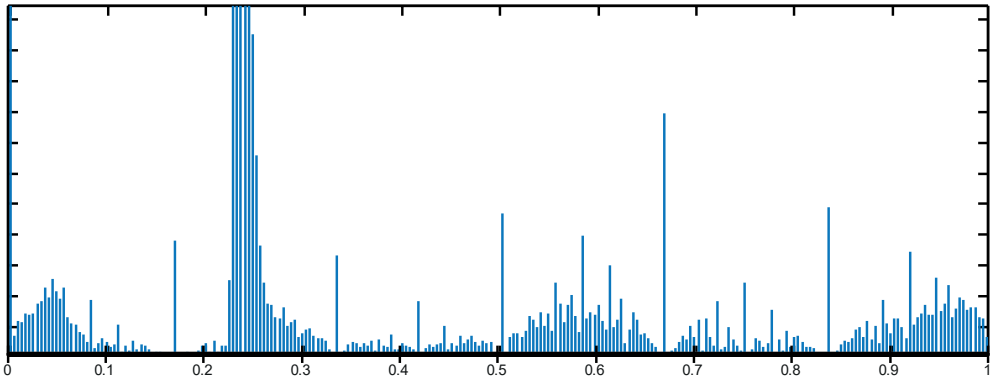
  - Many sets of color components can be used for this analysis: RGB, HSI, …
  - We are going to use the green chromaticity, which is computed as:

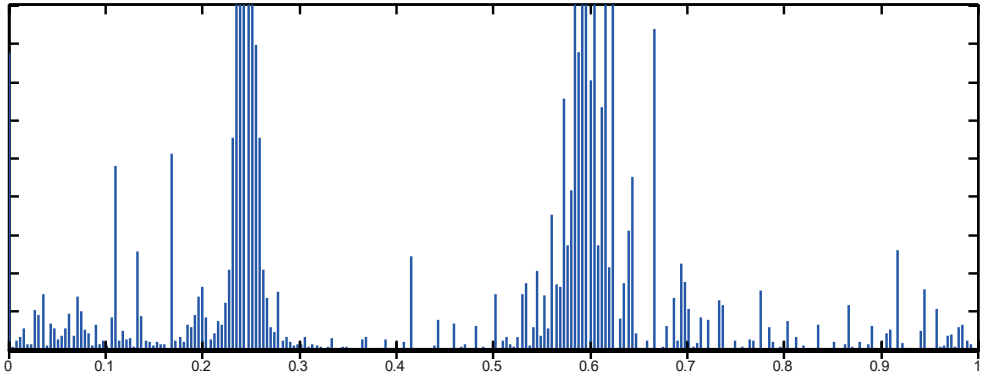
$$g = \frac{G}{R + G + B}$$

    - Values between 0 and 1.
    - Pure green pixels result in $g = 1$
    - Green is dominant when $g > 1/3$
    - Therefore → The highest values of the histograms will correspond to Green values, not to pixels with the highest luminance (as until now).

4.1.2.7 Case of use – Soccer field segmentation

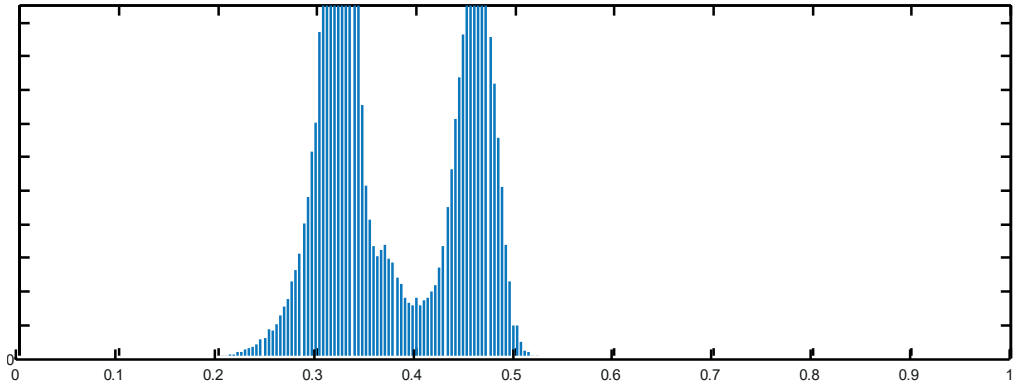
- **Initial analysis:**
  - Examples of histograms using the H channel of HSI:

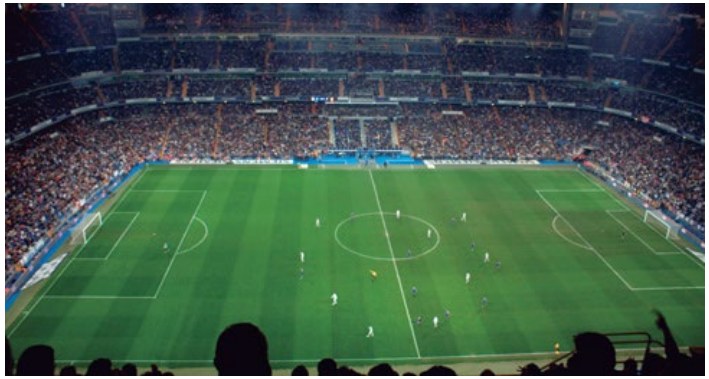


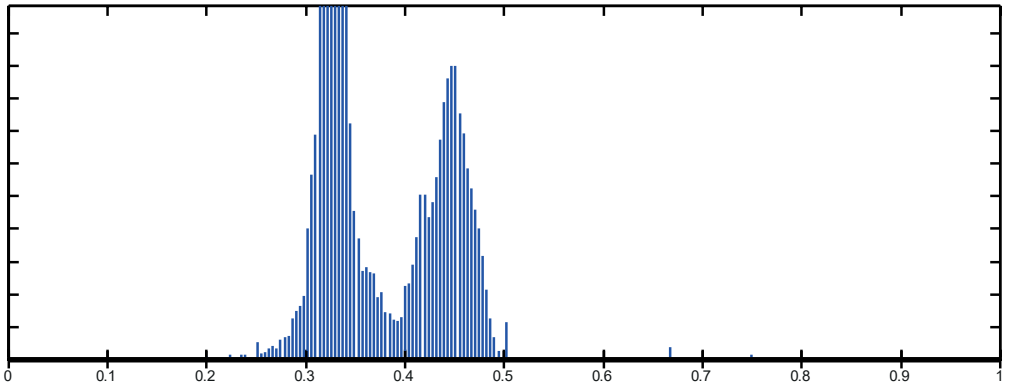


4.1.2.7 Case of use – Soccer field segmentation

- **Initial analysis:**
  - Examples of histograms using the *g* channel:

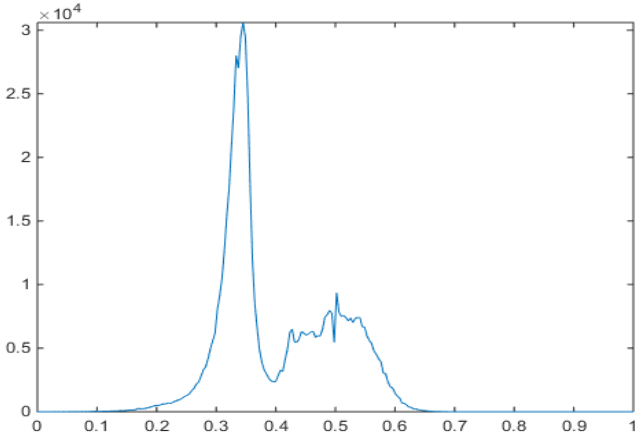




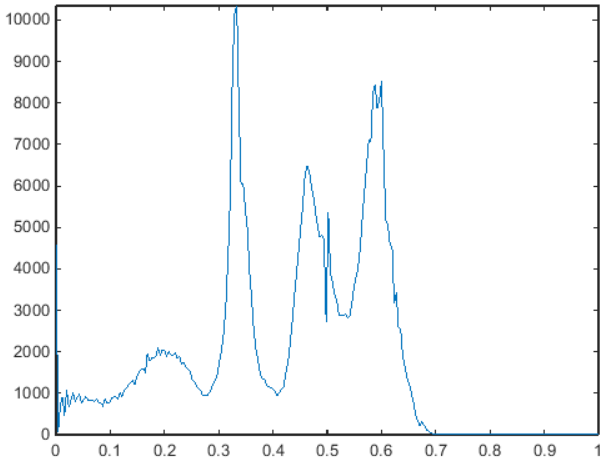


4.1.2.7 Case of use – Soccer field segmentation

- **Typical problems:**
  - Images with shadows:



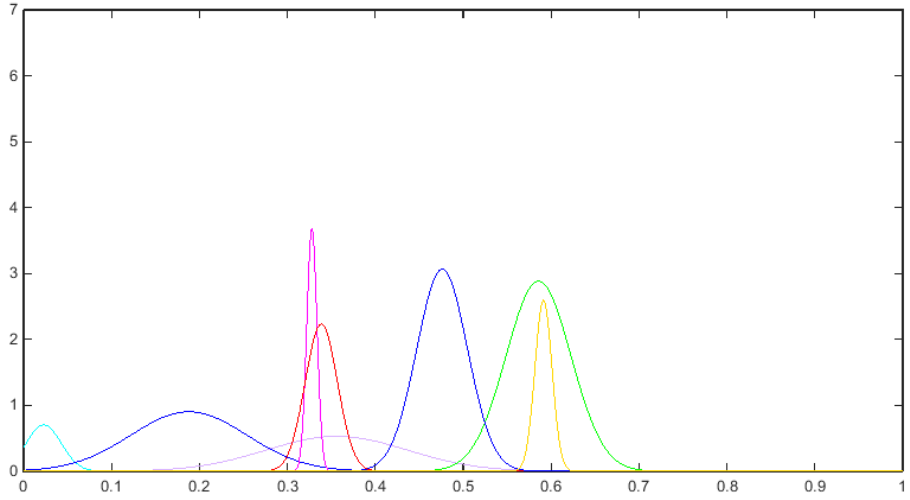
  - Multiple green levels:

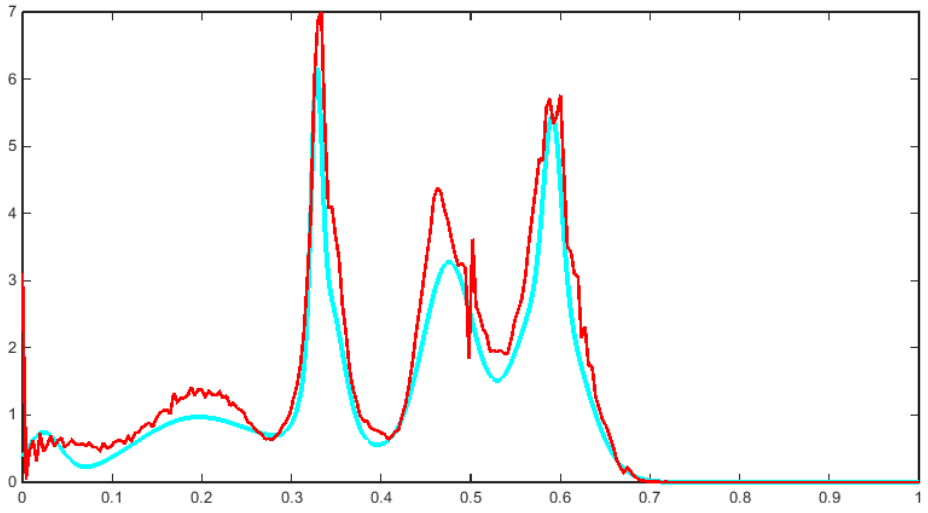


How many modes must we choose?

4.1.2.7 Case of use – Soccer field segmentation

- **Proposed solution:**
  1. Randomly select a subset of data → It will allow decreasing the computational cost.
  2. Approximate the histogram by a parametric model → Mixture of N Gaussians using E-M algorithm.
     - The number of Gaussians is automatically chosen.

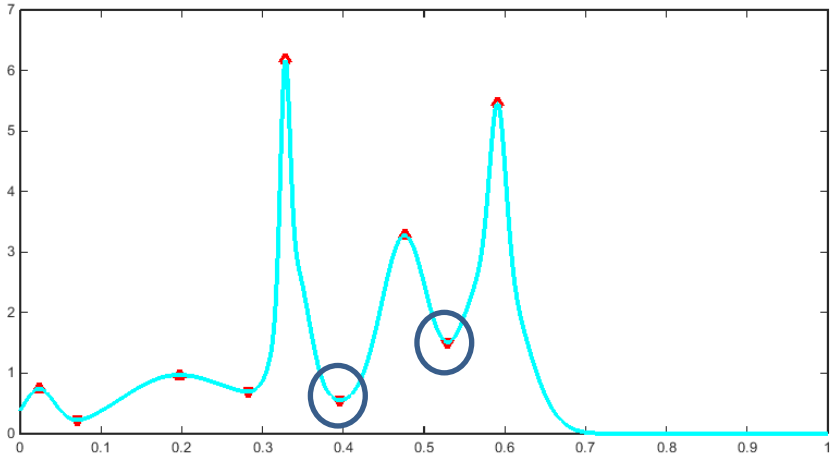




4.1.2.7 Case of use – Soccer field segmentation

- **Proposed solution:**

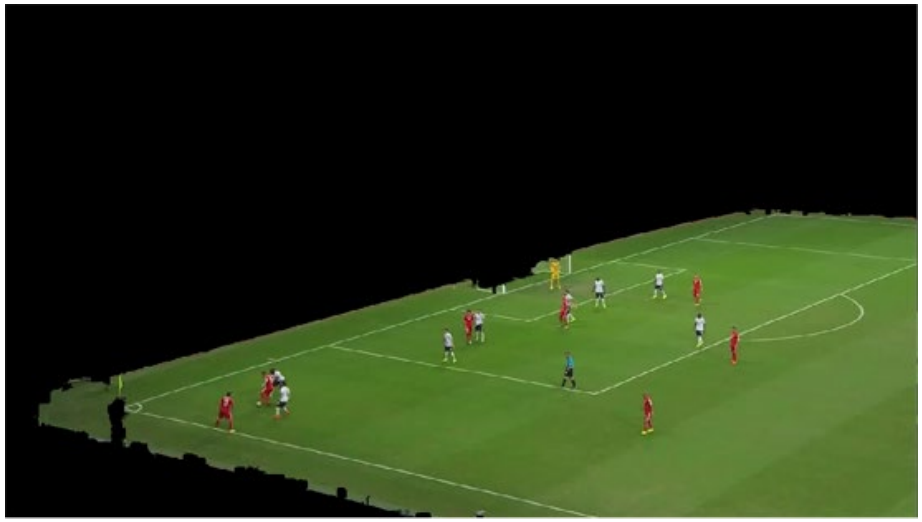
  3. Find local maxima and minima.



- Modes corresponding to green are those satisfying:
  - $G > R$ and $G > B \rightarrow g > \frac{1}{3}$
- Only these modes are selected.

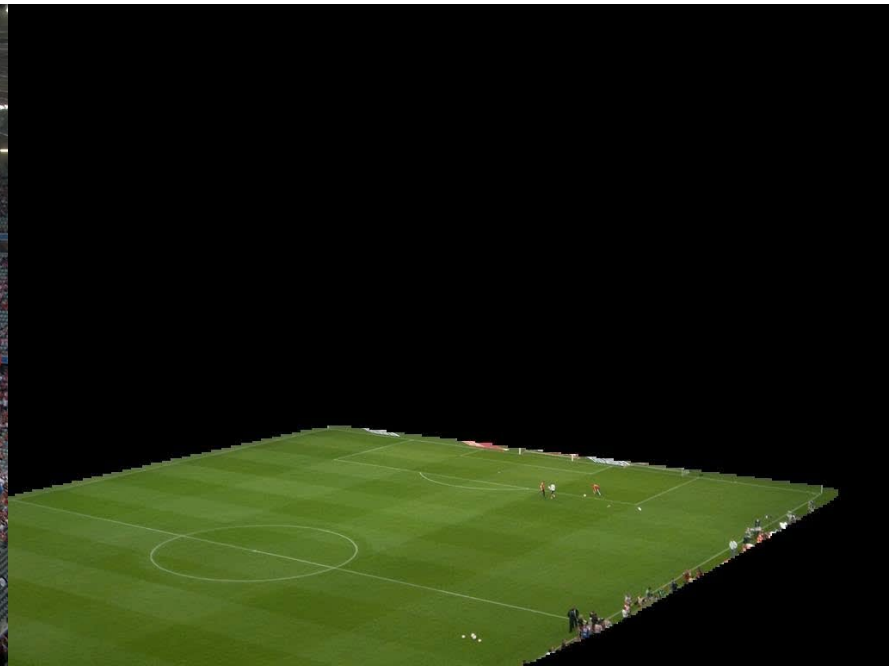
4.1.2.7 Case of use – Soccer field segmentation

- **Proposed solution:**
  4. Post-processing:
     - Holes are filled.
     - Only the largest compact segment is selected.



4.1.2.7 Case of use – Soccer field segmentation

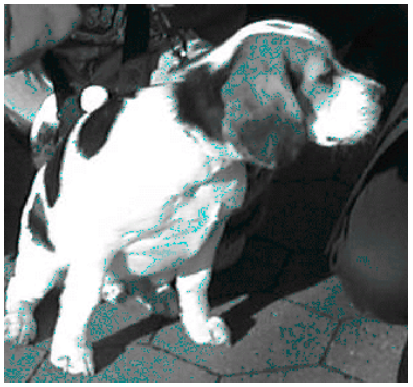
- **Results:**

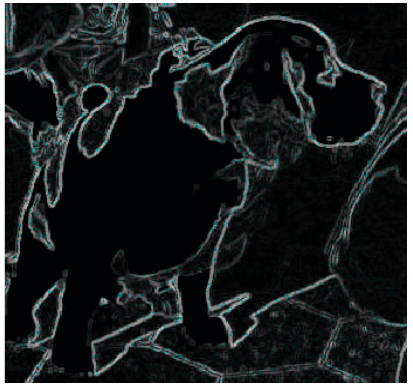


4.1.3 Edge-based segmentation

- Since objects are fully represented by their edges, the images can be segmented into objects by finding the edges of such objects.

- The main stages in edge-based segmentation are:





1. Compute the edge image (Sobel, Prewitt, Canny, etc.).

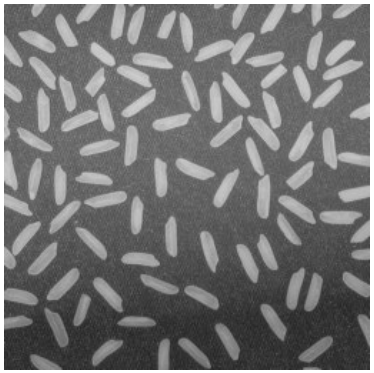
2. Process the edge image to locate the contour of objects of interest.

3. Fill inside the object whose contour we have identified.

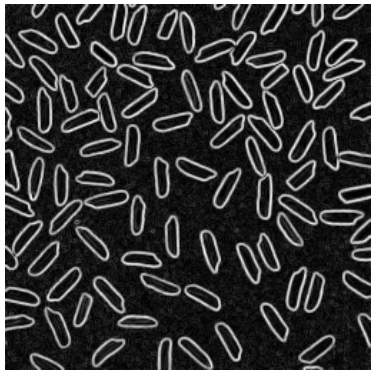
4.1.3 Edge-based segmentation

- **Algorithm: Laplacian-based segmentation**
  - This is a simple example to show how to segment an object when its boundary is perfectly closed (rare case).

1. Obtain the edge image (that is, the gradients of the image, G).
   - In this example we have used the module of the edge vectors obtained with the Sobel operator.



I



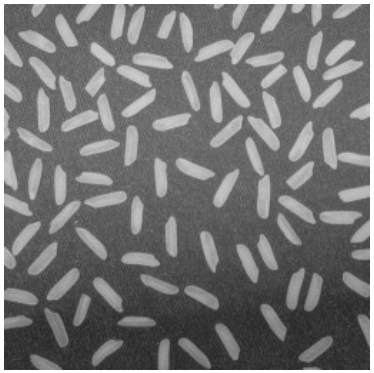
G

$$G_h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \quad G_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I$$

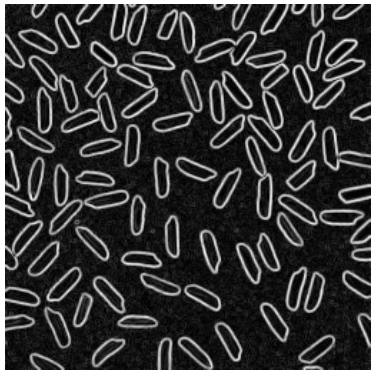
$$G = \sqrt{G_h^2 + G_v^2}$$

4.1.3 Edge-based segmentation

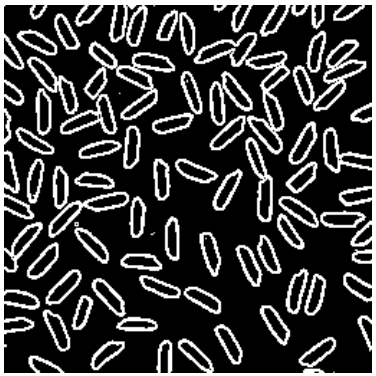
- **Algorithm: Laplacian-based segmentation**
  2. Threshold G to have a binary image, G_{th}, showing the most prominent edges in the image.
     - The threshold used is typically very low → It allows removing noisy gradients but preserves most edges.



I



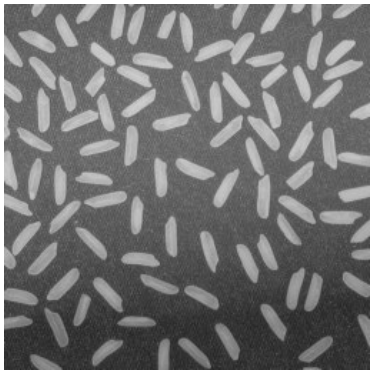
G



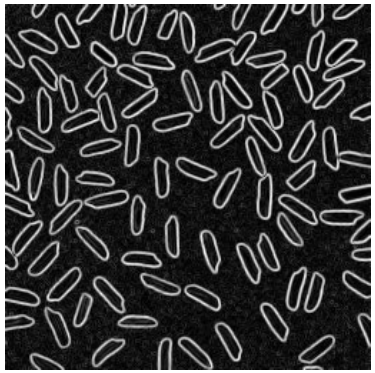
G_{th}

4.1.3 Edge-based segmentation

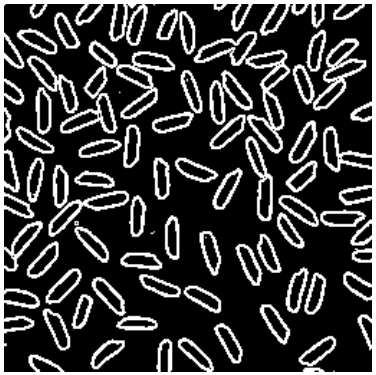
- **Algorithm: Laplacian-based segmentation**
  3. Compute a Laplacian image, ΔI, from I.
     - Any discrete or continuous Laplacian operator can be used.
     - The Laplacian is a differential operator given by the divergence of the gradient of a function:
       - Positive values when the gradient is growing.
       - Negative values when the gradient is decreasing.
       - Nulls if the gradient is constant.



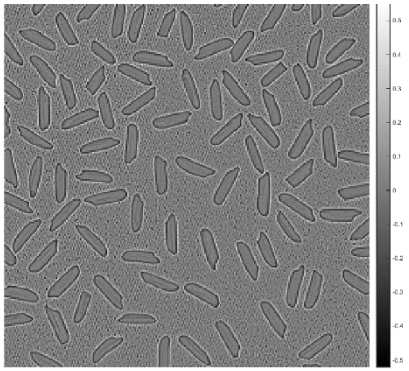
I



G



G_{th}

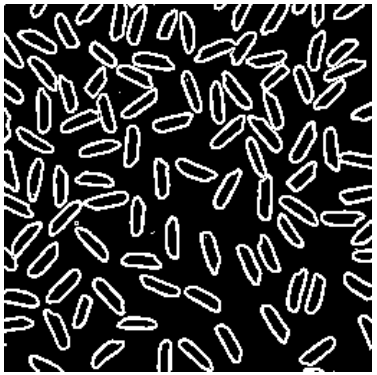


ΔI

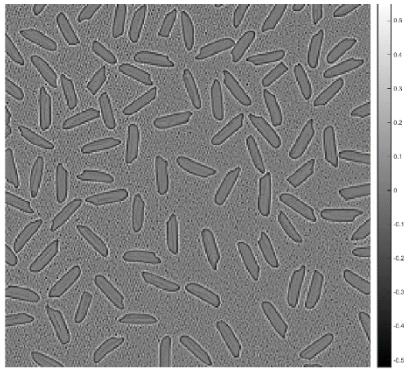
4.1.3 Edge-based segmentation

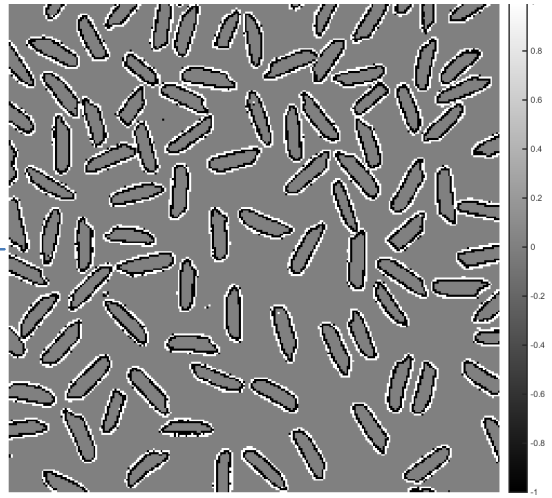
- **Algorithm: Laplacian-based segmentation**
  4. Compute the image $g = G_{th} \cdot \mathrm{sgn}(\Delta I)$.

G_{th}



ΔI

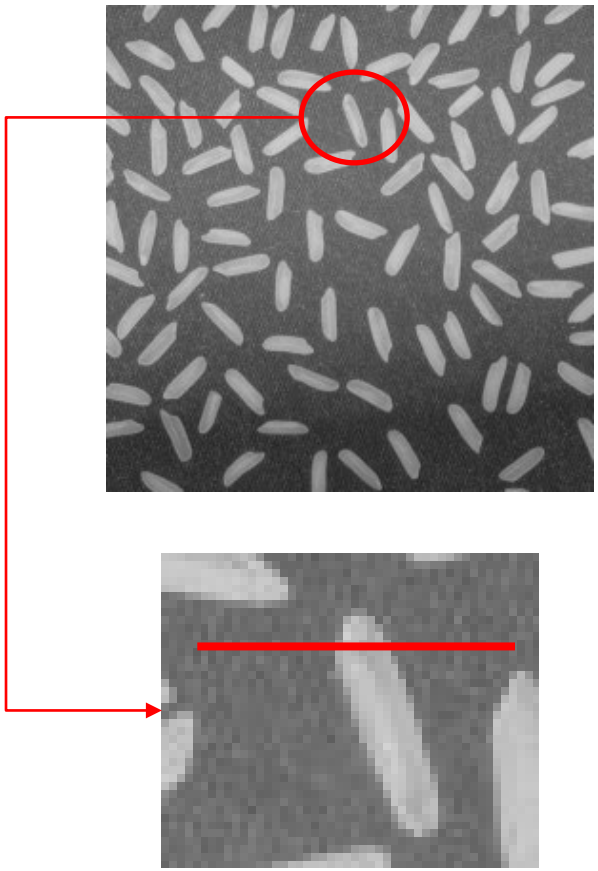


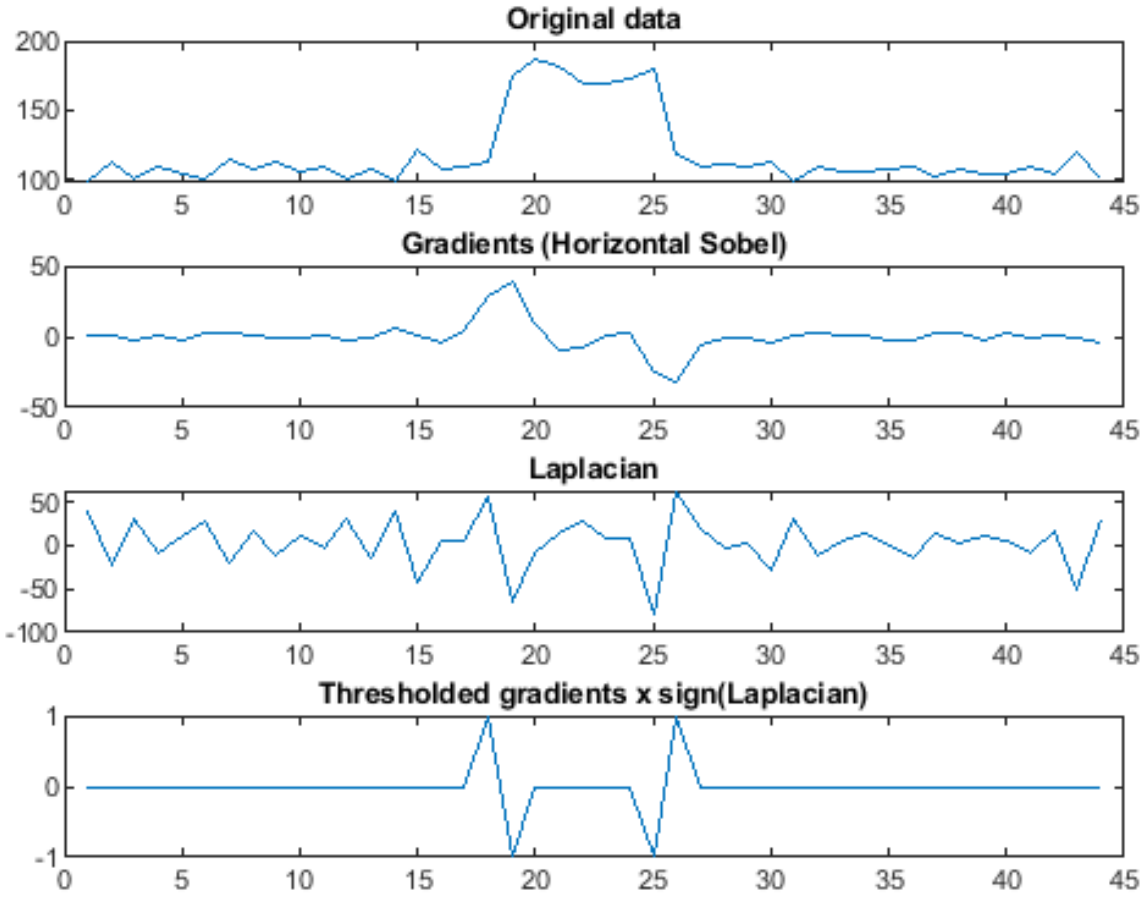


- This image has only 3 possible values:
  - 0 at non-edge pixels.
  - -1 at edge-pixels on the bright side of the edges.
  - 1 at the edge-pixels on the dark side of the edges.

4.1.3 Edge-based segmentation

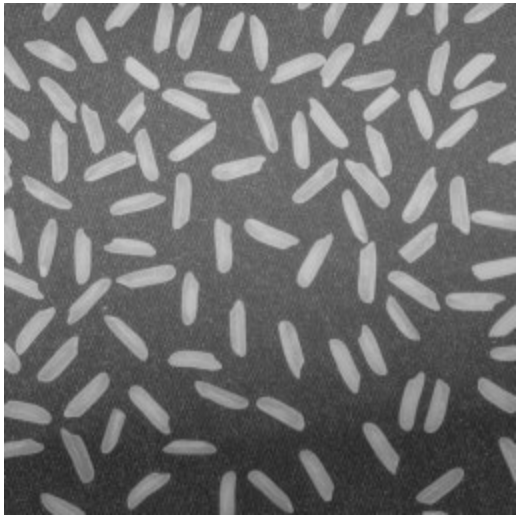
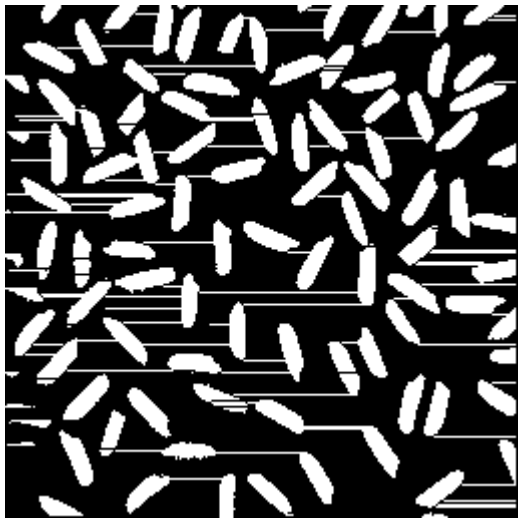
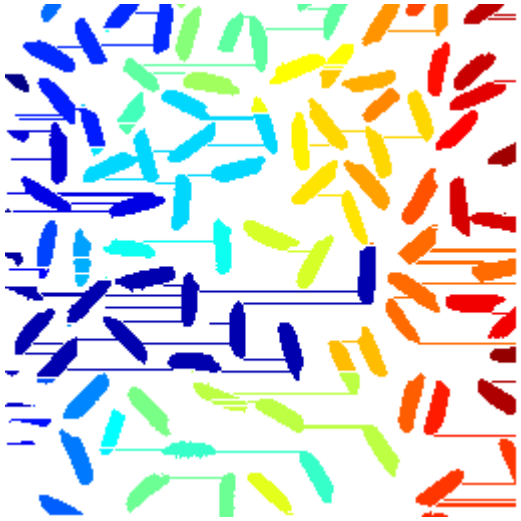
- **Algorithm: Laplacian-based segmentation**





4.1.3 Edge-based segmentation

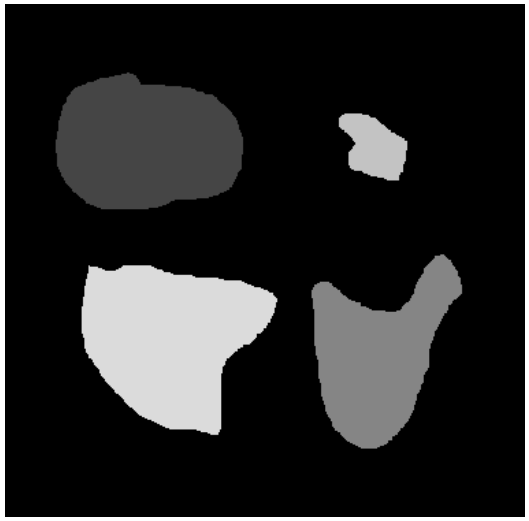
- **Algorithm: Laplacian-based segmentation**
  5. The image g is traversed from left to right:
     - Adjacent pixels with values 1 and -1 → Entering an object.
     - Adjacent pixels with values -1 and 1 → Leaving an object.

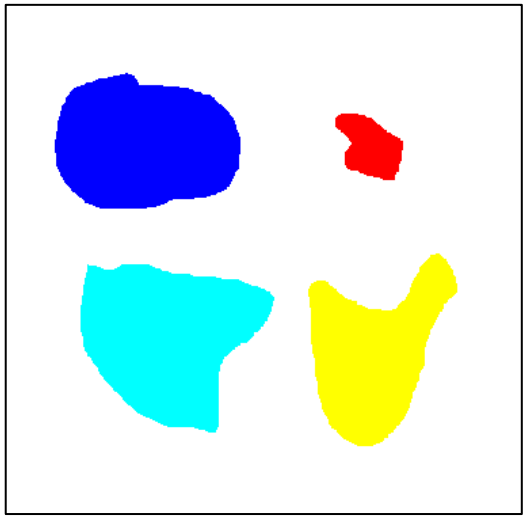




- The errors are due to the noise in the original image → It results in erroneous sign transitions.

4.1.3 Edge-based segmentation

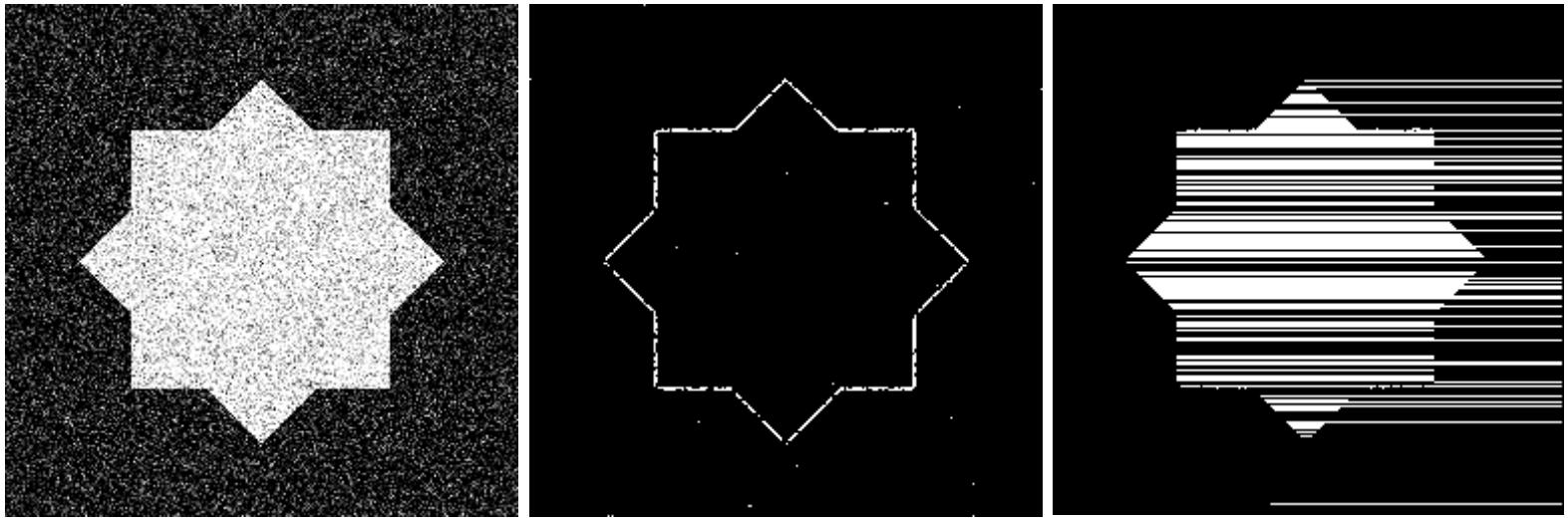
- **Algorithm: Laplacian-based segmentation**
  - Result in an image with better defined edges.





4.1.3 Edge-based segmentation

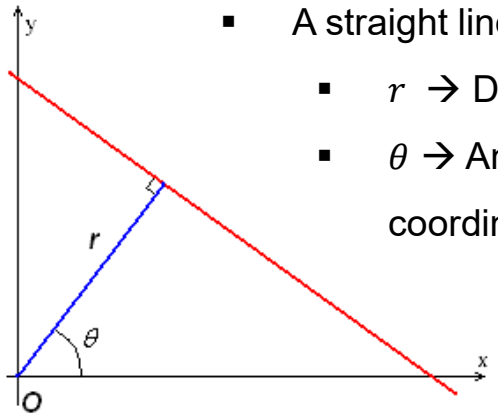
- **Algorithm: Laplacian-based segmentation**
  - Drawbacks:
    - Edge detection rarely provides perfect and closed boundaries.
    - There are spurious detected edges → Post-processing?
    - There are gaps where there should be edges → Post-processing?



4.1.3.1 The Hough transform

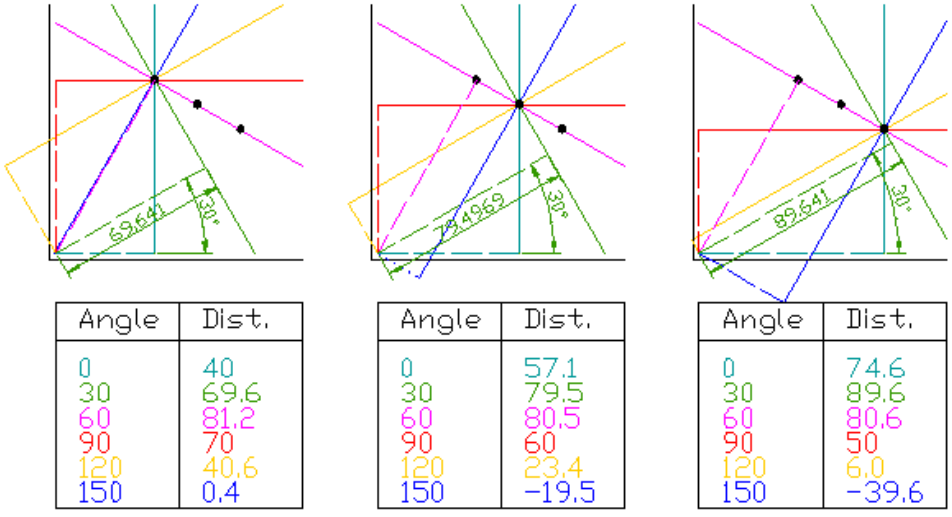
- It is very useful if we have some idea of what the edges should look like (e.g., lines, circles, etc.).
- It works successfully if the objects are simple (their boundaries can be modeled with few parameters).
- It usually requires post processing stages.

- <u>The simplest case is detecting straight lines:</u>
  - A straight line can be represented (polar notation) as $r = x\cos\theta + y\sin\theta$
    - r → Distance from the origin to the closest point on the line.
    - θ → Angle between the x axis and the line connecting the origin of coordinates with that closest point.



4.1.3.1 The Hough transform

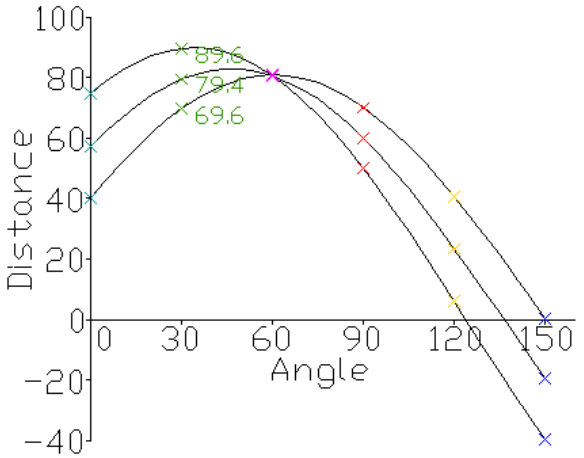
- Given a simple point in the plane, the set of all straight lines going through that point corresponds to a **sinusoidal curve in the** (r, θ) **plane**, which is unique for that point.
- A set of two or more points belonging to the same line will produce sinusoids that cross at the (r, θ) for that line → A line can be detected by looking of concurrent curves.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

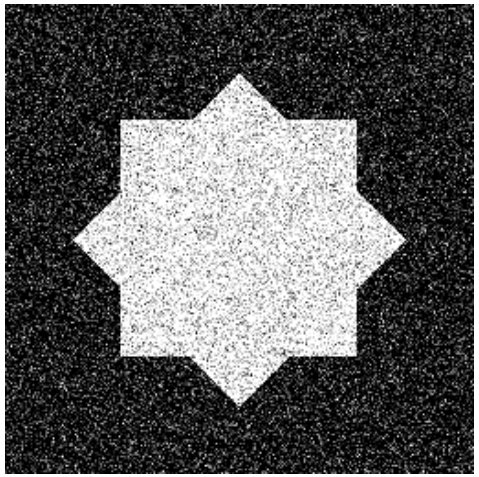
Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6



4.1.3.1 The Hough transform

- **Example:**
  - Let us suppose that we want to segment the following image, I.
  - We know that the object to segment is the intersection of a square and a diamond → It is bounded by 8 straight lines.



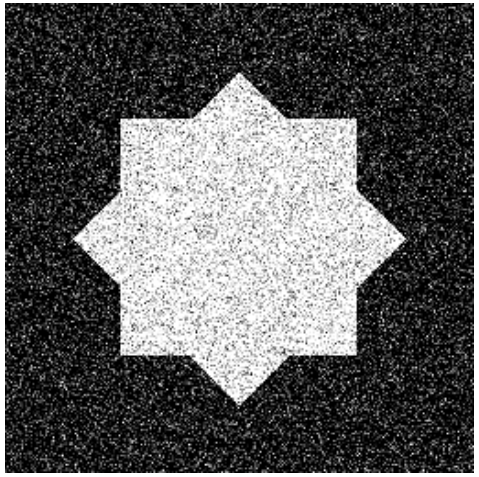
I

4.1.3.1 The Hough transform

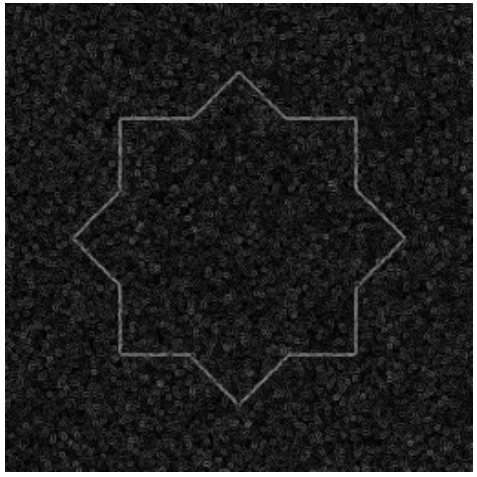
- **Example:**

1. Obtain the edge image, G.

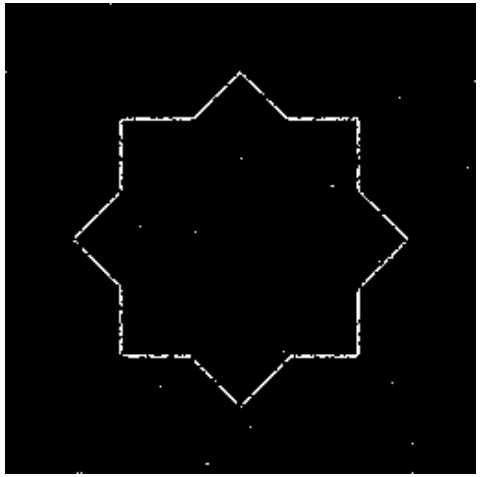
2. Threshold G to have a binary image, G_{th}.



I



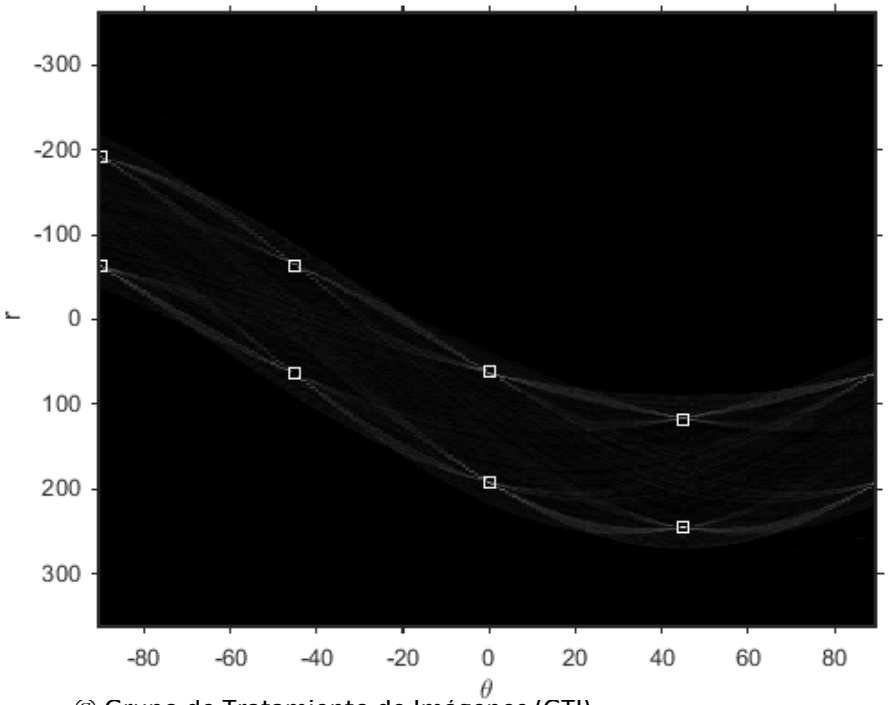
G

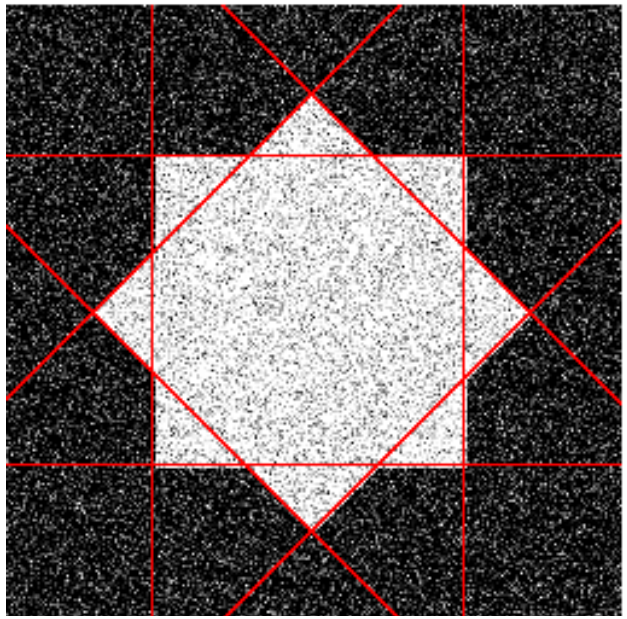


G_{th}

4.1.3.1 The Hough transform

- **Example:**
  3. Compute the Hough transform, H (one Gaussian per edge point in G_{th}).
  4. Select the maximum 8 values.
     - These maxima will correspond to the 8 straight lines determining the boundaries of the object to segment.



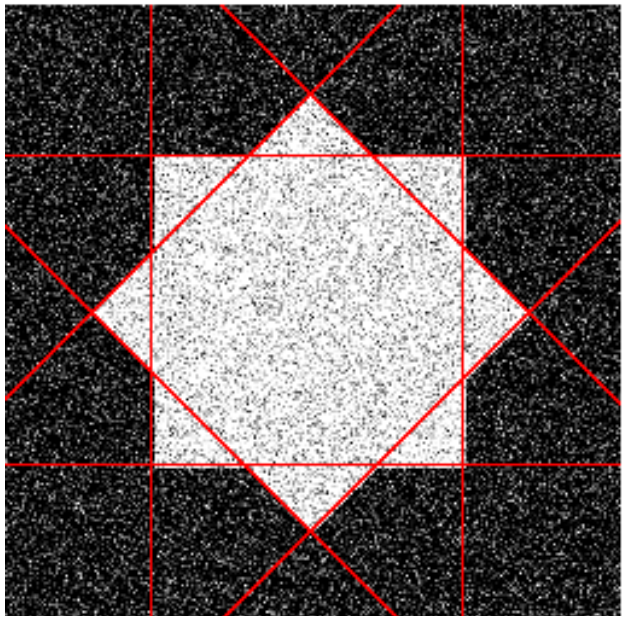


4.1.3.1 The Hough transform

- **Example:**

  5. Create a binary image from the detected lines:

     a. For each line, compute its cut points with the image boundaries.



4.1.3.1 The Hough transform

- **Example:**

  5. Create a binary image from the detected lines:
     a. For each line, compute its cut points with the image boundaries.
     b. Determine what pixels belong to the straight lines between the computed cut points (e.g., using Bresenham algorithm).

Bresenham algorithm

Given the coordinates of two points (x_0, y_0) and (x_1, y_1), it allows to find all the intermediate points required for drawing line on an image of pixels.

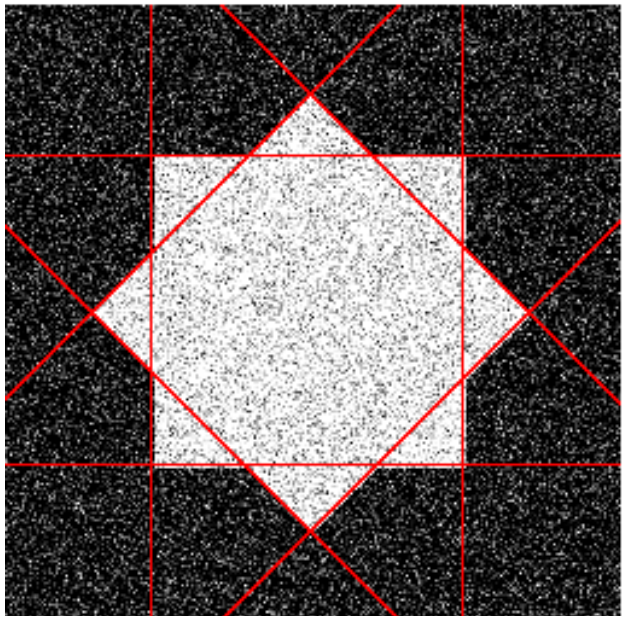
4.1.3.1 The Hough transform

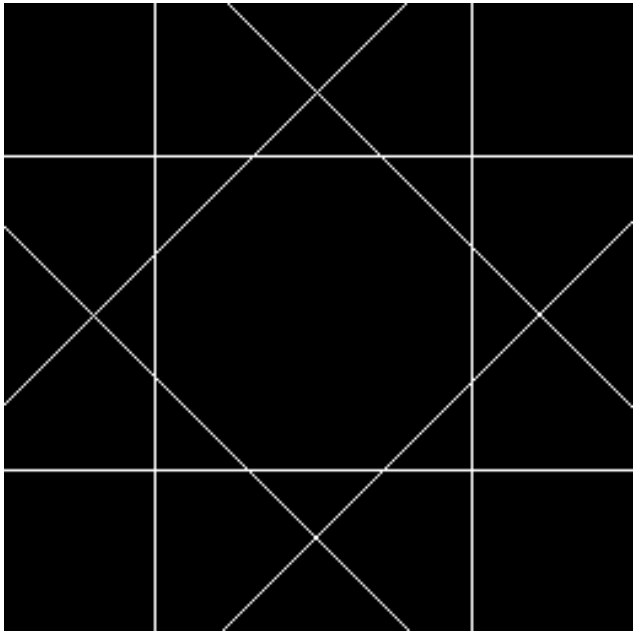
- **Example:**

  5. Create a binary image from the detected lines:

     a. For each line, compute its cut points with the image boundaries.

     b. Determine what pixels belong to the straight lines between the computed cut points (e.g., using Bresenham algorithm).

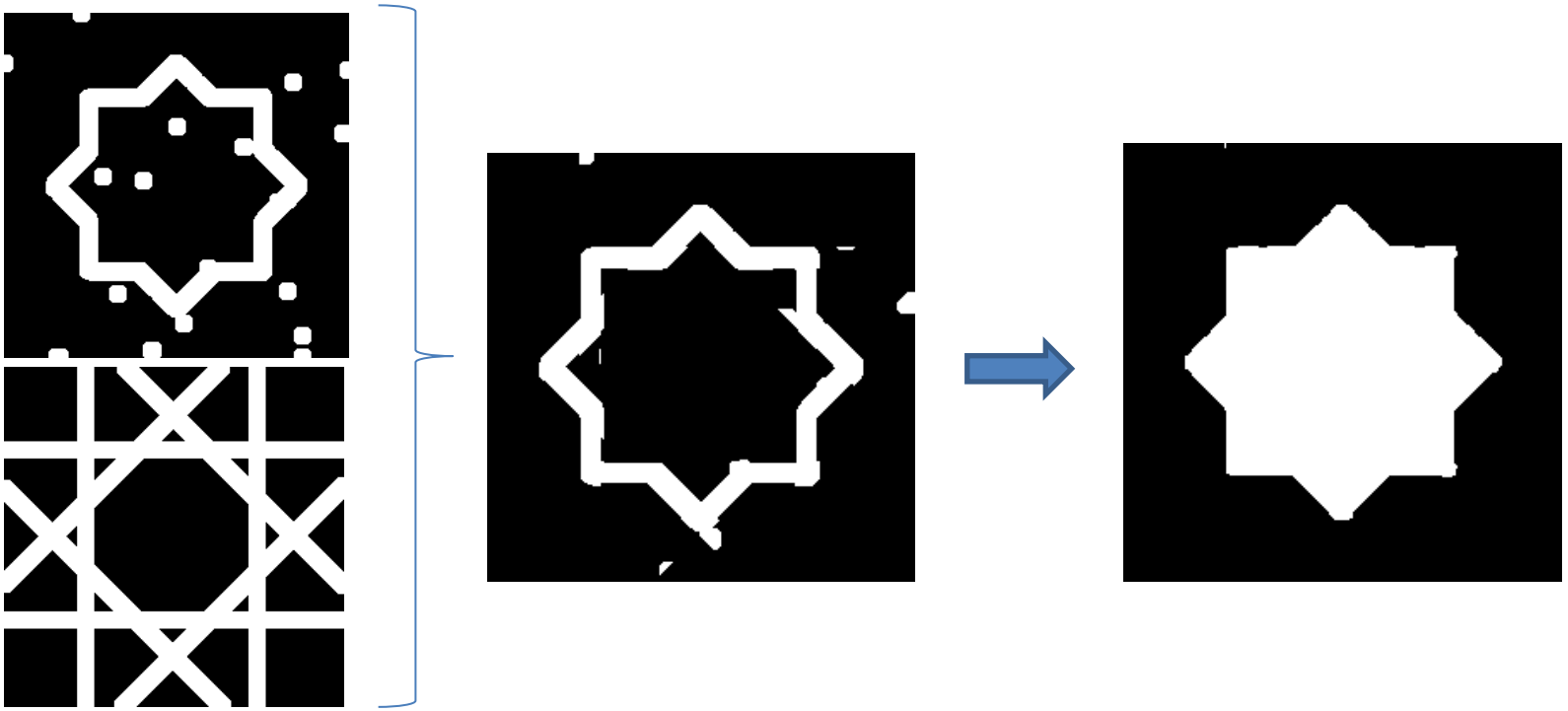




4.1.3.1 The Hough transform

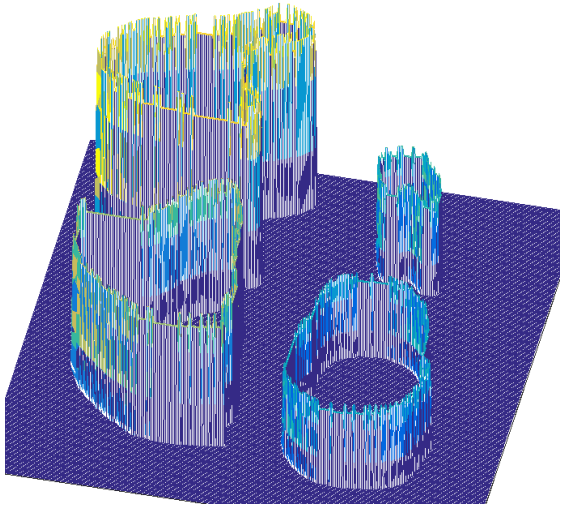
- **Example:**

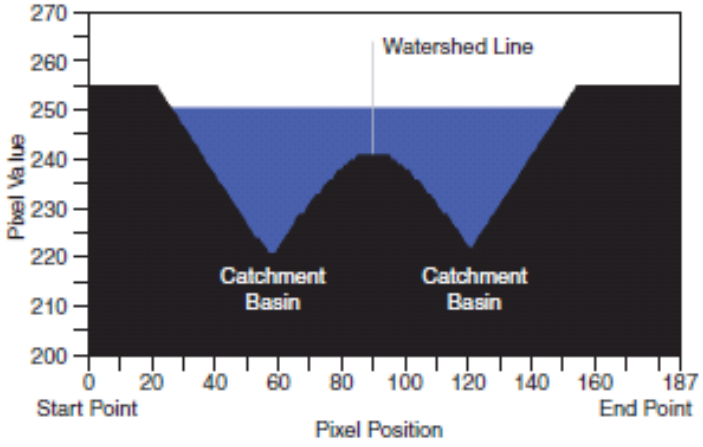
6. Post processing:
   a. Dilate both G_{th} and the binary image obtained in point 5.
   b. Compute the intersection between such images.
   c. Erode and fill the result.



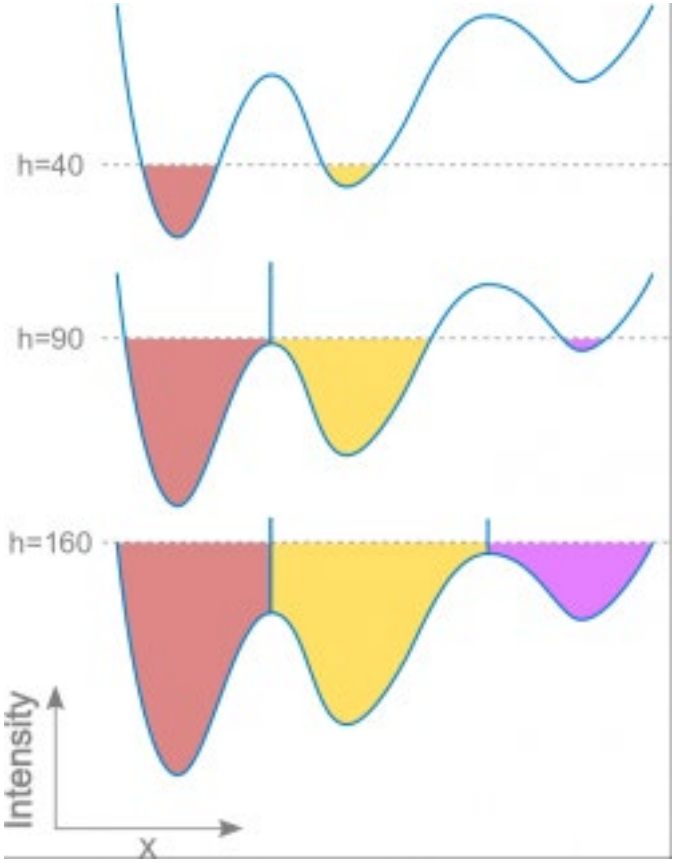
4.1.3.2 Watershed segmentation

- Any grey scale image can be considered as a topographic surface.
  - The gradients are like dams separating the objects.
- Watershed algorithm: Flood this topographic surface from its minima and prevent the merging of the waters coming from different sources.
  - The result is an image partitioned into two sets: **catchment basins** and **watershed lines**.



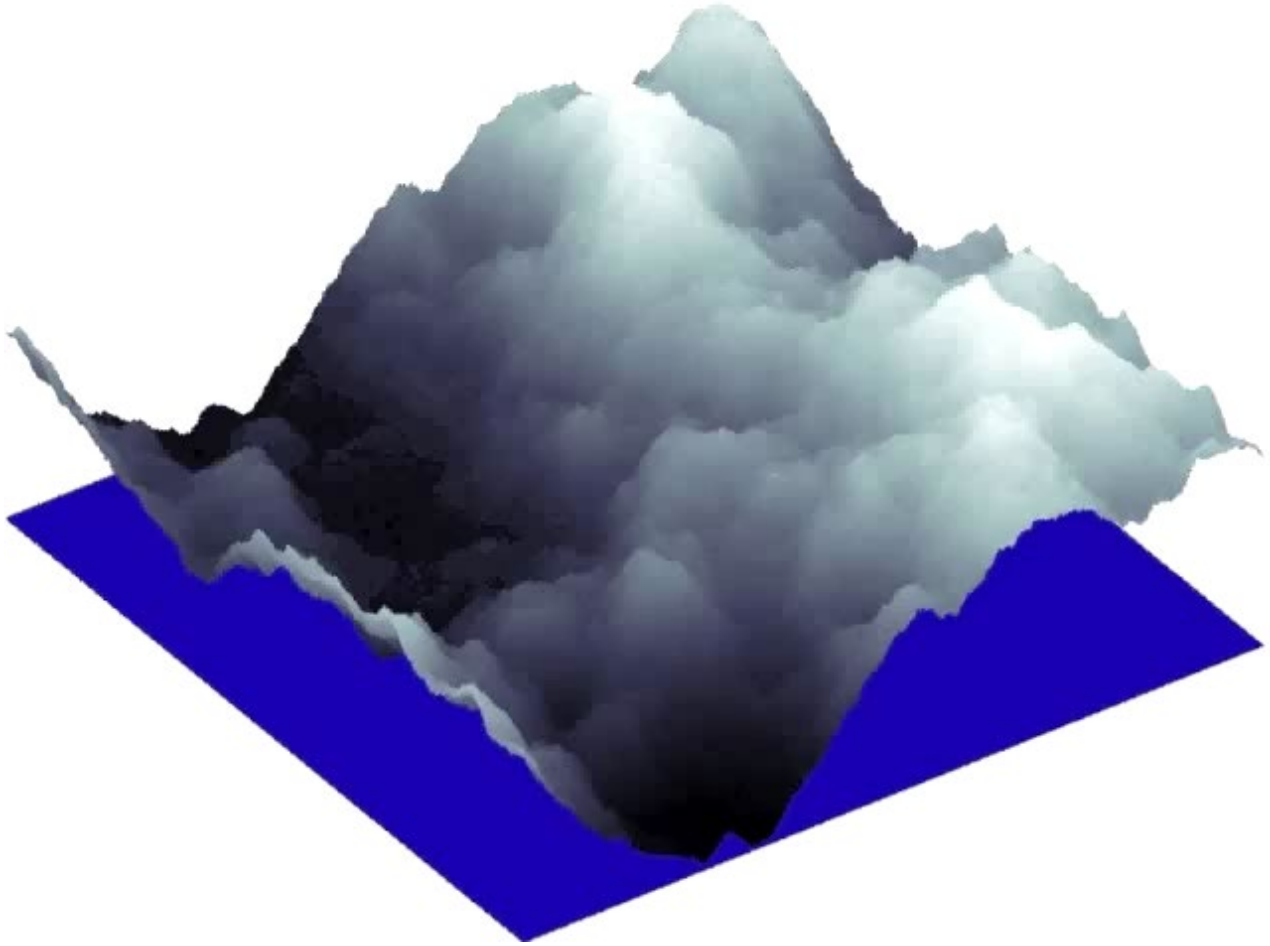


4.1.3.2 Watershed segmentation



https://imagej.net/plugins/classic-watershed

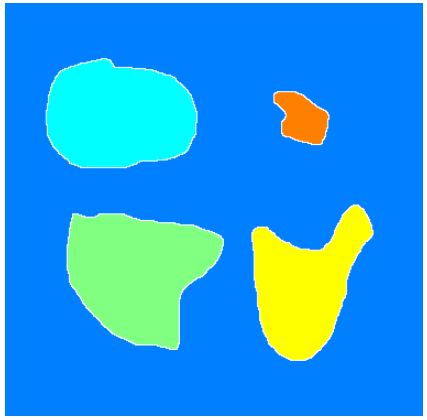
4.1.3.2 Watershed segmentation



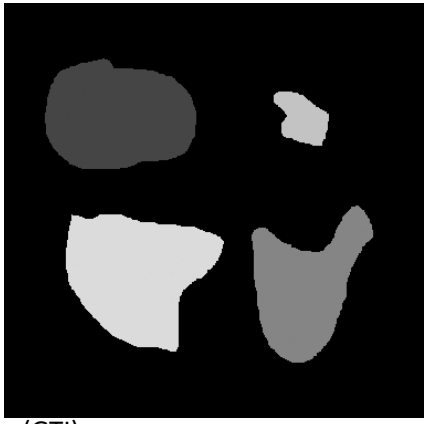
https://www.youtube.com/watch?v=ILBiDWWiVeU

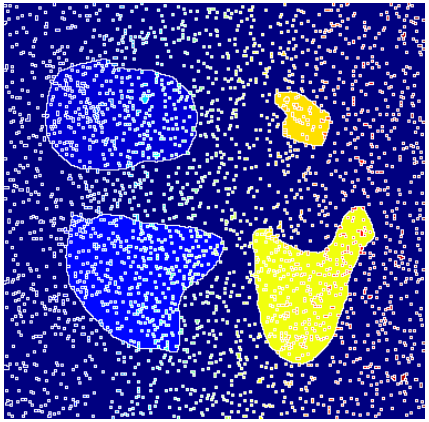
4.1.3.2 Watershed segmentation

- This transform provides high-quality segmentations in simple images.



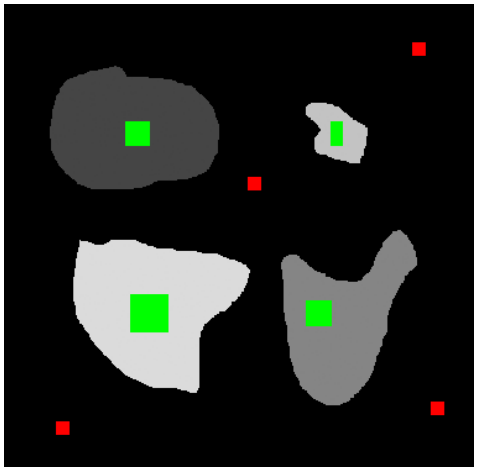
- However, in most images, it produces an important **over-segmentation** due to noise or local irregularities in the gradients.

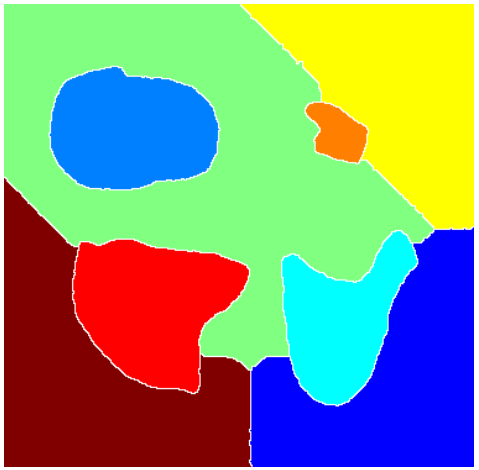




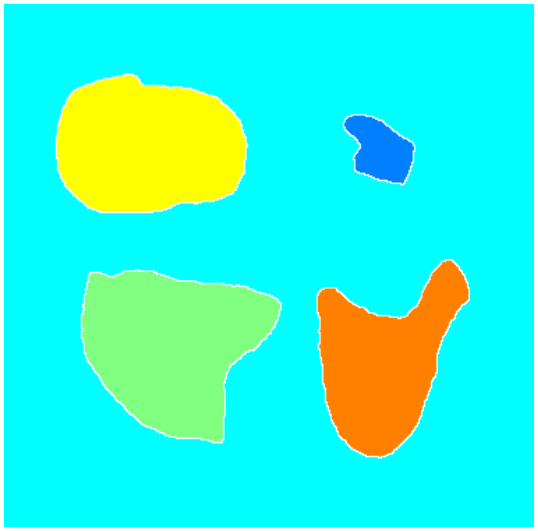
4.1.3.2 Watershed segmentation

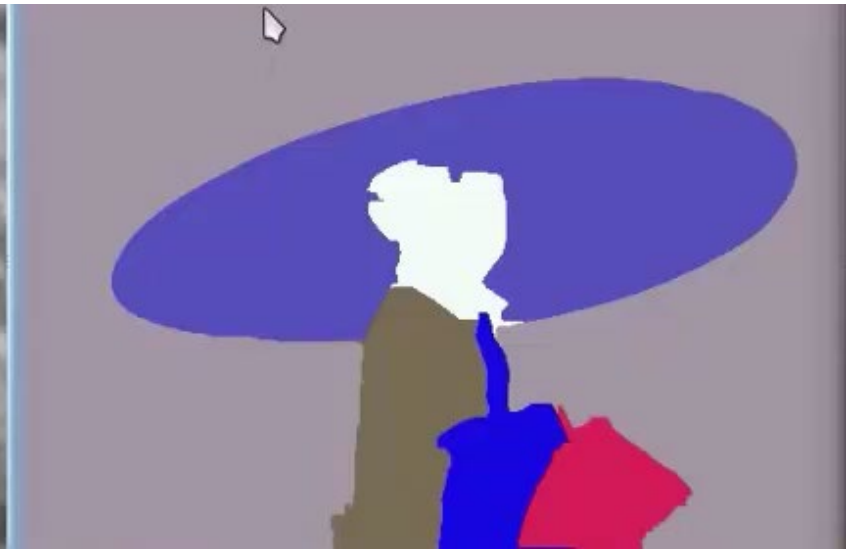
- Over-segmentation can be avoided if the seeds used in the transformation are a set of previously defined markers.
- Marker-controlled watershed segmentation follows this basic procedure:
  1. Compute the segmentation function → In our case, the image of gradients.
     - It must be an image whose dark regions are the objects to segment.
  2. Compute de foreground markers → Pixels within each of the objects.
  3. Compute the background markers → Pixels that are not part of any object.
  4. Modify the segmentation function → Set marker locations as its minima.
  5. Compute the watershed transform of the modified segmentation function.





4.1.3.2 Watershed segmentation





4.1.3.3 Active contours (snakes)

- An active contour (typically called snake) is a curve defined in an image that can change its location and shape until it best satisfies predefined conditions.
- The initial curve is an inaccurate approximation of the boundary of the object to segment → They are useful if the approximate shape of the boundary is known.
- The curve evolves looking for minimizing an energy function. This evolution is influenced by:
  - Image constraints (e.g., edges) that pull it toward the object contours.
  - Internal forces that resist deformation.
  - Restrictions imposed by the user (e.g., deformation speed).

4.1.3.3 Active contours (snakes)

- A snake is typically modeled as a parametrized curve $C(s) = \big(x(s), y(s)\big)$, where $s \in [0,1]$.
  - $C(0)$ gives the coordinate par corresponding to the starting point → $\big(x(0), y(0)\big)$.
  - $C(1)$ gives the coordinate par corresponding to the end point → $\big(x(1), y(1)\big)$.
  - $C(s)$, with $0 < s < 1$ gives the remaining coordinates of the curve.

- The energy function to be minimized is the following:

$$E = \int_0^1 E\big(C(s)\big)ds = \int_0^1 \Big(E_{int}\big(C(s)\big) + E_{image}\big(C(s)\big) + E_{con}\big(C(s)\big)\Big)\, ds$$

4.1.3.3 Active contours (snakes)

$$E = \int_0^1 E\big(C(s)\big)ds = \int_0^1 \Big(E_{int}\big(C(s)\big) + E_{image}\big(C(s)\big) + E_{con}\big(C(s)\big)\Big)\,ds$$

- E_{int}:
  - It imposes smoothness constraints. Depends, exclusively, on the curve shape.

$$E_{int}\big(C(s)\big) = \frac{\alpha(s)|C_s(s)|^2 + \beta(s)|C_{ss}(s)|^2}{2}$$

  - $C_s(s)$ → First order derivative of the curve → It makes the curve act like a membrane (elasticity).
  - $C_{ss}(s)$ → Second order derivative of the curve → It makes the curve act like a thin-plate (rigidity).
  - $\alpha(s) \in [0,1]$ → Weight for controlling the importance of the elasticity.
  - $\beta(s) \in [0,1]$ → Weight for controlling the importance of the rigidity.

4.1.3.3 Active contours (snakes)

$$E = \int_0^1 E(C(s))ds = \int_0^1 \left(E_{int}(C(s)) + E_{image}(C(s)) + E_{con}(C(s))\right) ds$$

- E_{image}:
  - It allows the curve to be attracted by the image features.

$$E_{image}(C(s)) = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term}$$

  - $E_{line} = I(i,j)$ → Energy contribution that depends on the intensity values covered by the curve.
    - If $w_{line} > 0$ → The curve will be attracted to the darkest values.
    - If $w_{line} < 0$ → The curve will be attracted to the lightest values.
  - $E_{edge} = -|\nabla I(i,j)|^2$ → Energy contribution that depends on the image gradients. It is higher in the pixels with lowest gradient module (minimizing this energy, the curve evolves to object boundaries).
  - E_{term} → Energy contribution that depends on the curvature of the image content.
    - It is high for those pixels in which the curvature is low.
    - So, minimizing this energy, the curve evolves to terminations of line segments and corners.

4.1.3.3 Active contours (snakes)

$$E = \int_0^1 E\big(C(s)\big)ds = \int_0^1 \Big(E_{int}\big(C(s)\big) + E_{image}\big(C(s)\big) + E_{con}\big(C(s)\big)\Big)\,ds$$

- E_{con}:
  - It considers any additional constraint, such as:
    - Penalizing the creation of loops in the curve.
    - Penalizing moving too far away from the initial position.
    - Penalizing moving into an undesired image region.
    - Etc.
  - These constraints may come from:
    - High level interpretation.
    - User interaction.
    - Etc.
  - In many applications this term is not used (it is set to zero).

4.1.3.3 Active contours (snakes)

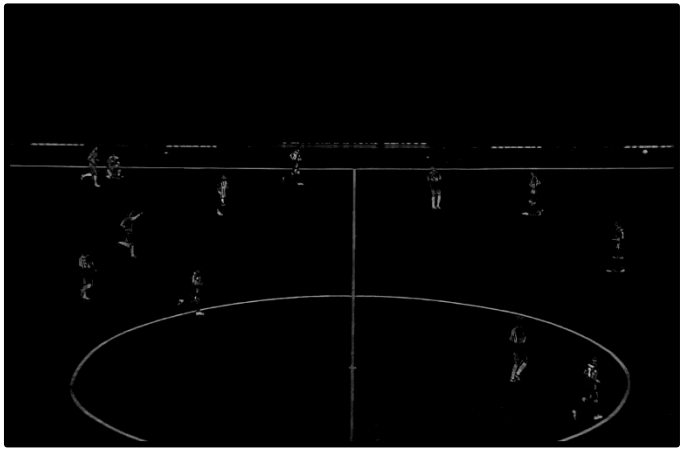
https://www.youtube.com/watch?v=ceIddPk78yA

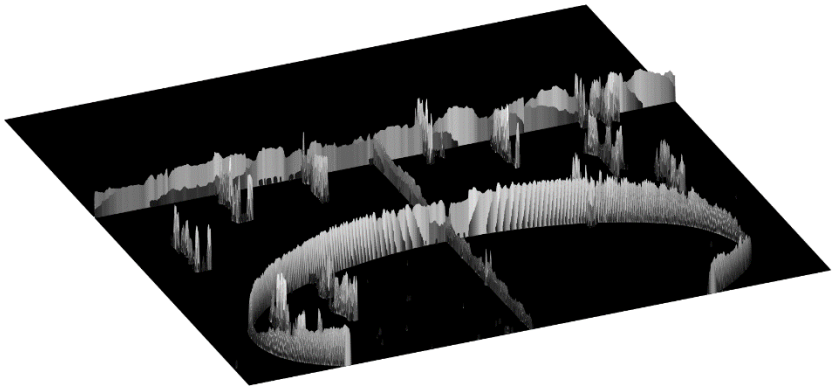
4.1.3.4 Case of use – TSLAB

- **TSLAB: Tool for Semiautomatic LABeling.**

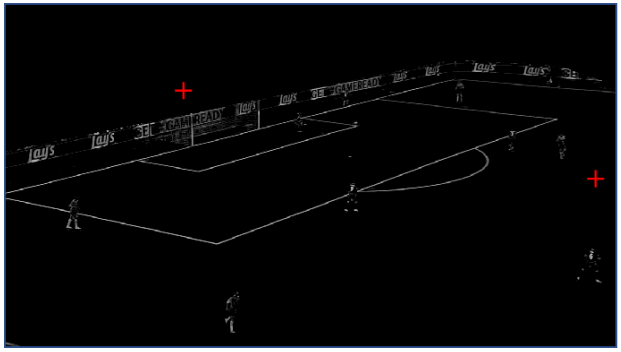
https://www.gti.ssr.upm.es/data/TSLAB.html

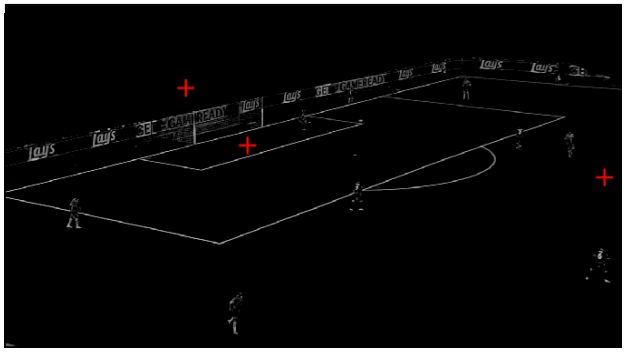
4.1.3.4 Case of use – Line mark detection

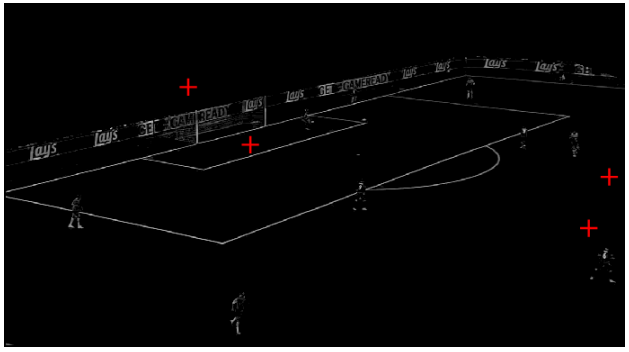




4.1.3.4 Case of use – Line mark detection



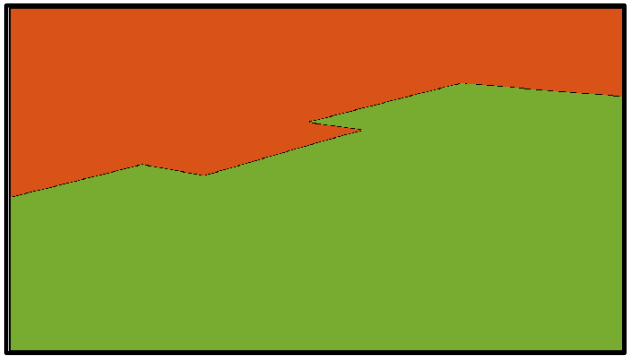


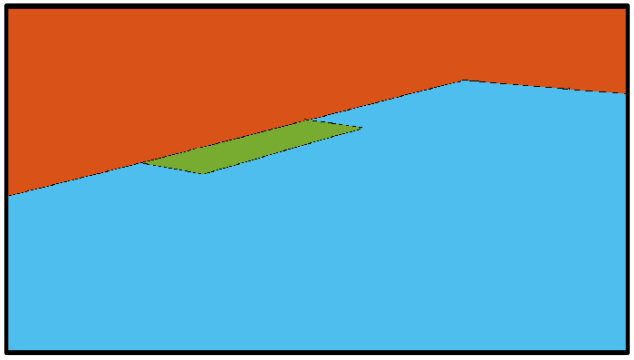


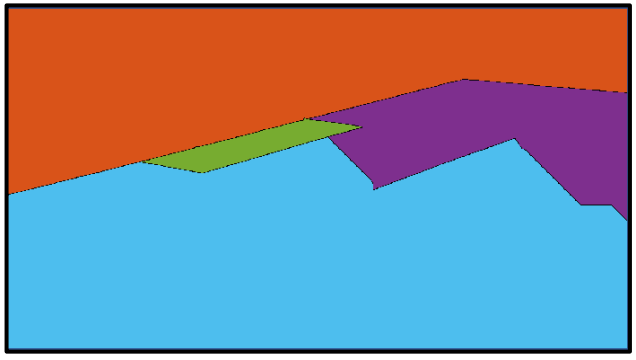
2 seeds

3 seeds

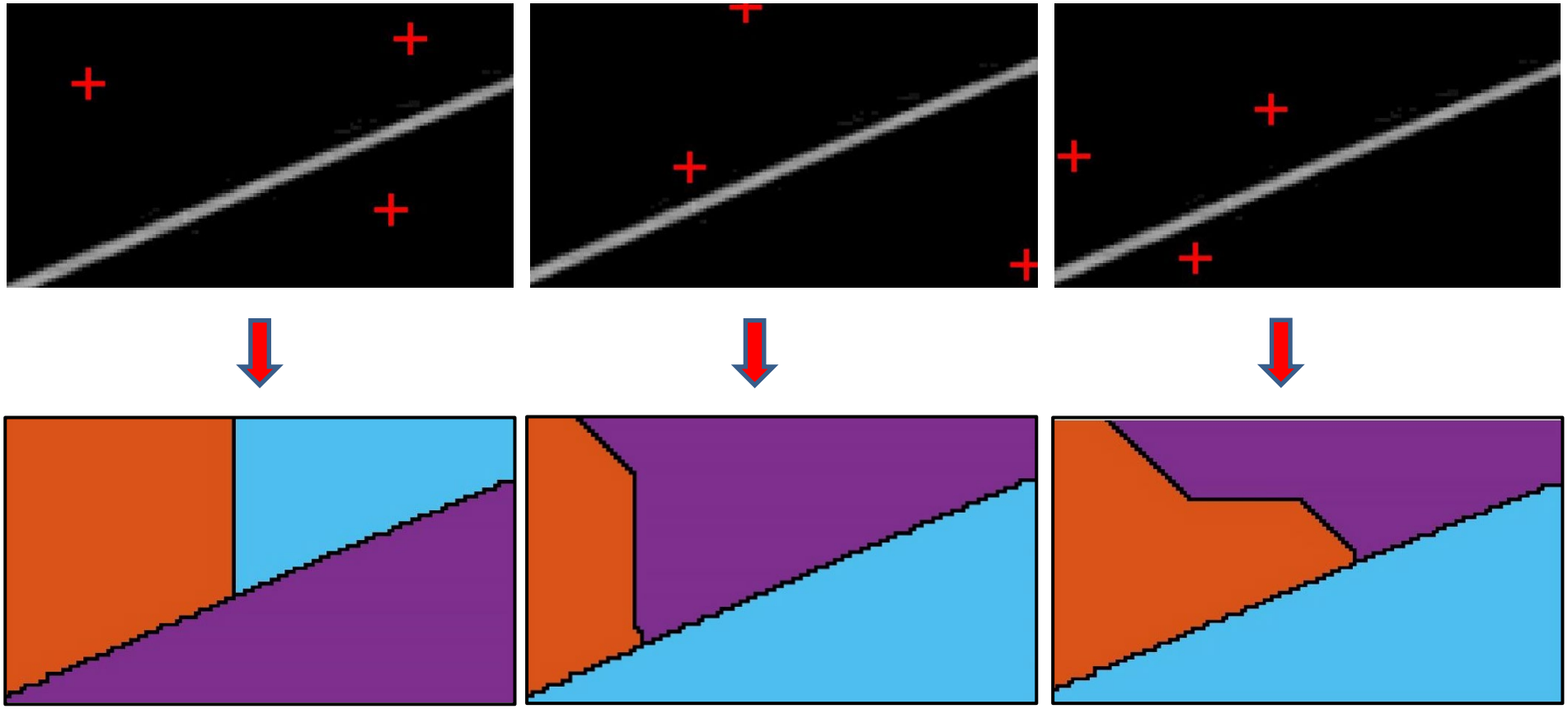
4 seeds



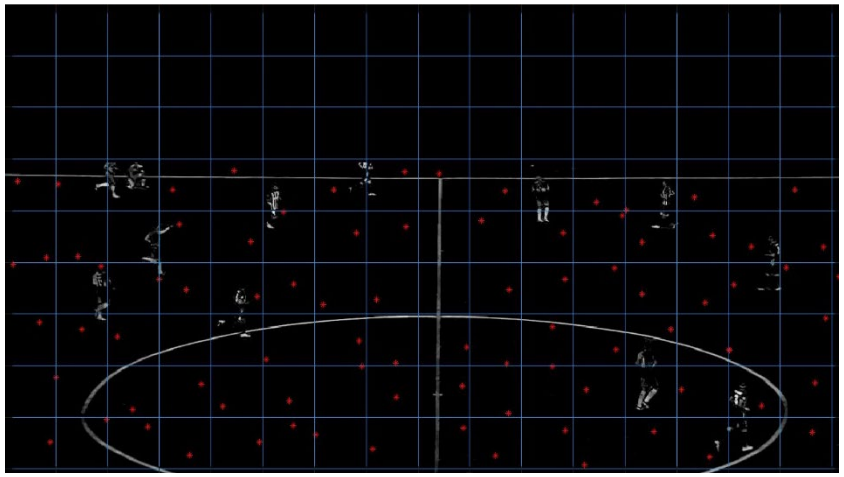




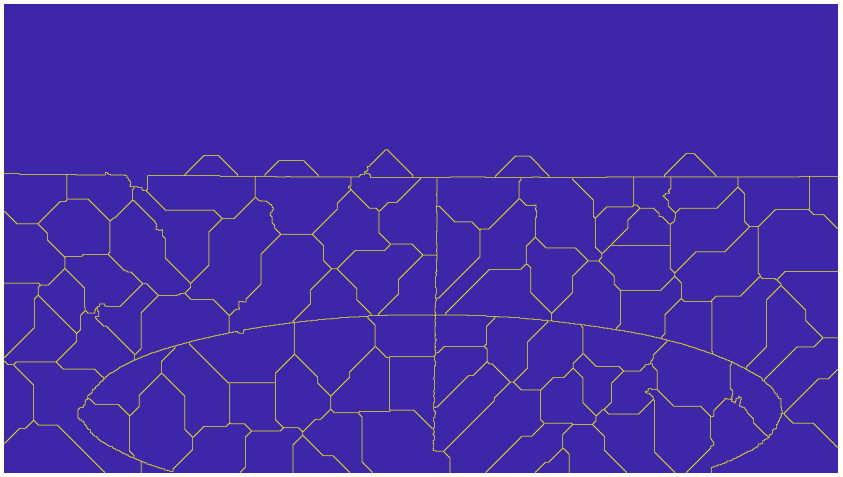
4.1.3.4 Case of use – Line mark detection



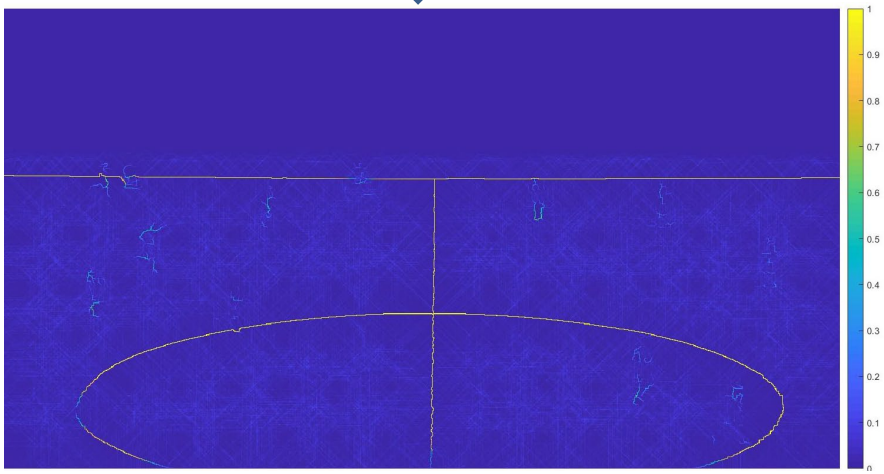
4.1.3.4 Case of use – Line mark detection

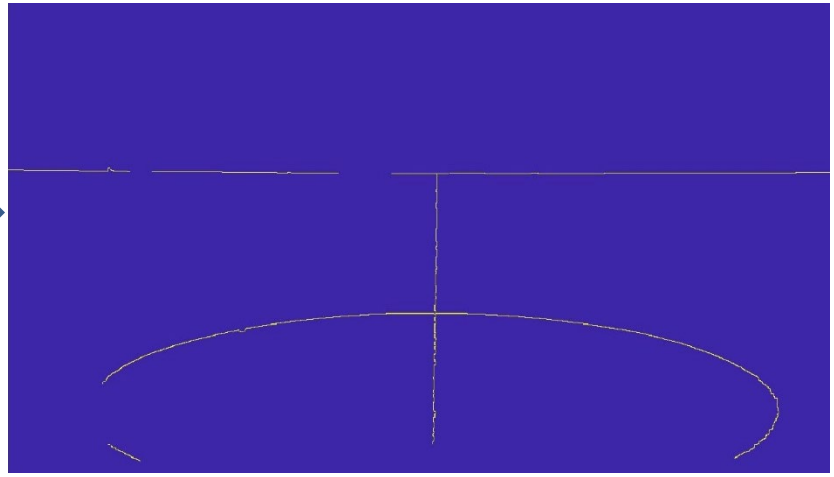


1 iter.



100 iter.





4.2 Foreground segmentation

4.2.1 Introduction

4.2.2 Difference-based detection

4.2.3 Temporal median filtering

4.2.4 Single Gaussian modeling

4.2.5 Gaussian Mixture Models

4.2.6 KDE-based modeling

4.2.7 Evaluation

4.2.8 Shadow detection

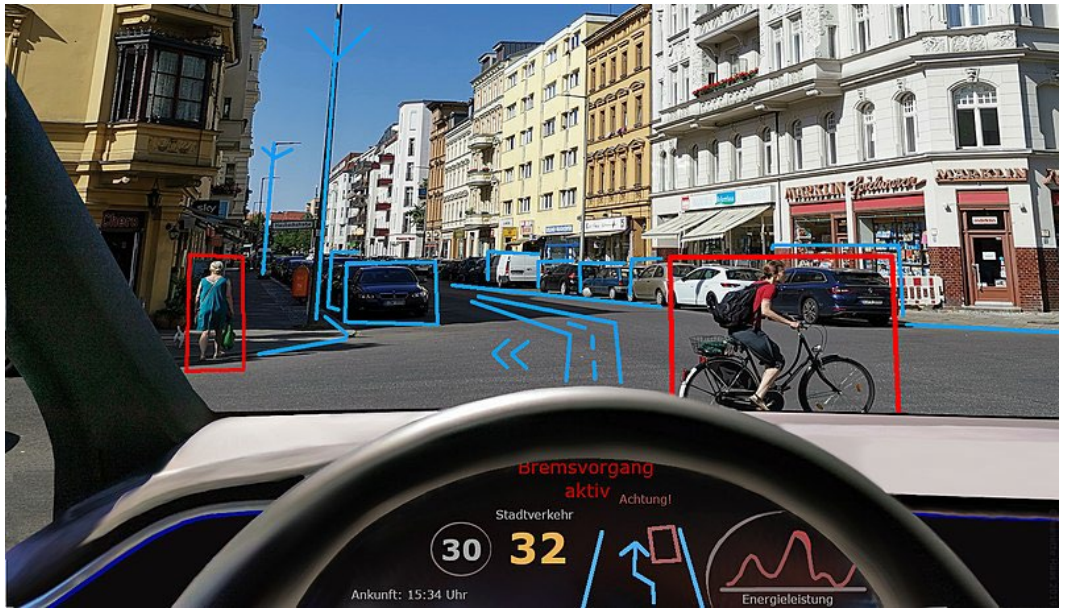
4.2.9 Stationary foreground detection

4.2.1 Introduction

- Detecting moving objects is a key task in many computer vision applications:
  - Video surveillance.

4.2.1 Introduction

- Detecting moving objects is a key task in many computer vision applications:
  - Video surveillance.
  - Augmented reality.
  - Photo effects.
  - Human-machine interaction.
  - Sport analysis.
  - Etc.



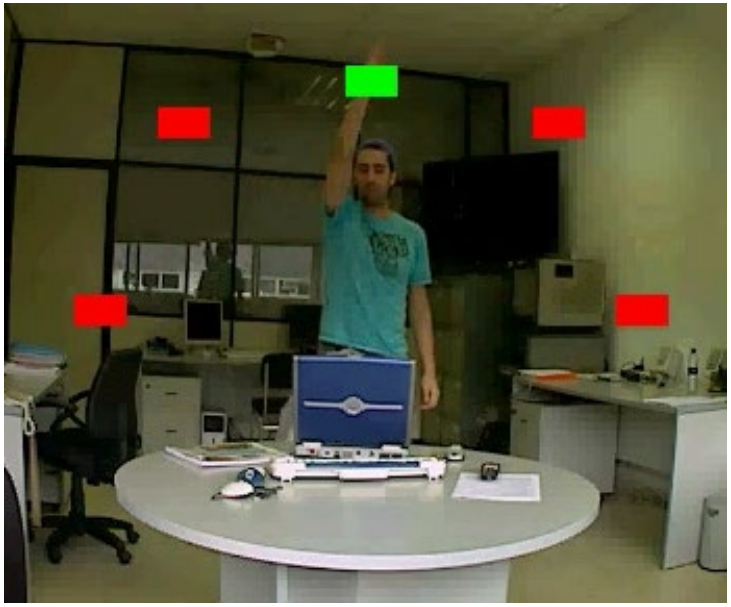
https://en.wikipedia.org/wiki/File:Autonomous-driving-Barcelona.jpg

4.2.1 Introduction

- Detecting moving objects is a key task in many computer vision applications:
  - Video surveillance.
  - Augmented reality.
  - Photo effects.
  - Human-machine interaction.
  - Sport analysis.
  - Etc.

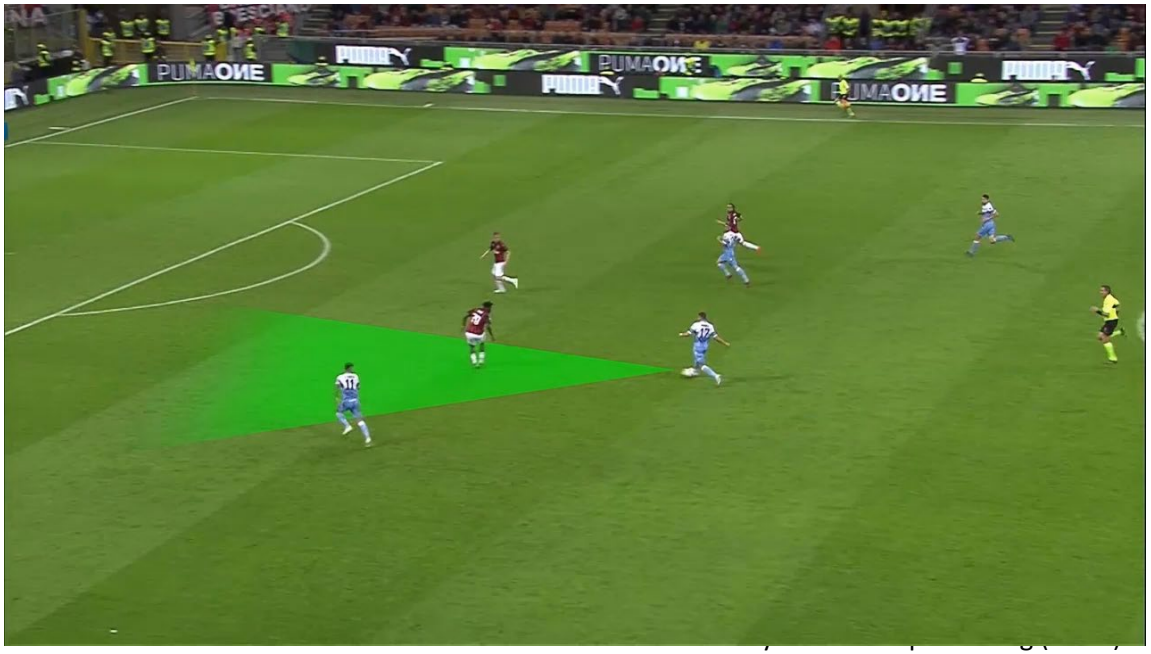
4.2.1 Introduction

- Detecting moving objects is a key task in many computer vision applications:
  - Video surveillance.
  - Augmented reality.
  - Photo effects.
  - Human-machine interaction.
  - Sport analysis.
  - Etc.



4.2.1 Introduction

- Detecting moving objects is a key task in many computer vision applications:
  - Video surveillance.
  - Augmented reality.
  - Photo effects.
  - Human-machine interaction.
  - Sport analysis.
  - Etc.



4.2.1 Introduction

- The basic operation in moving object detection strategies is the separation of the moving objects (called "**foreground**") from the rest of the scene (called "**background**").

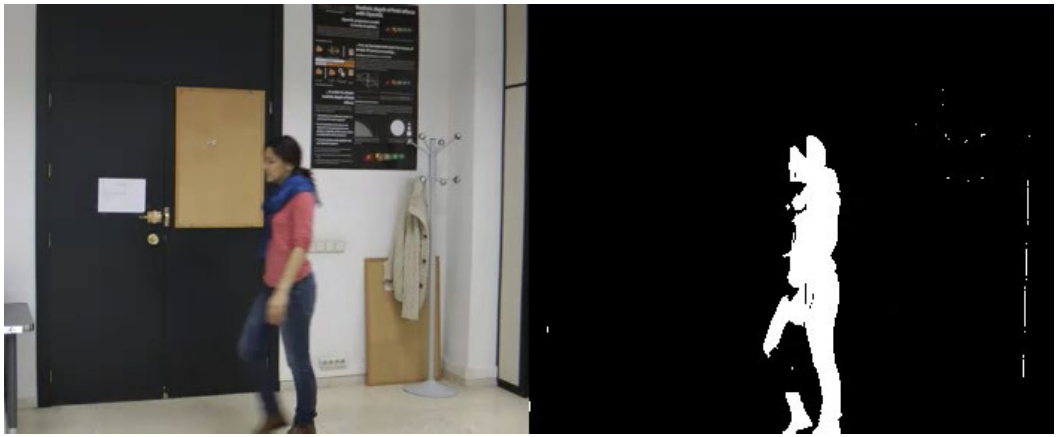
Original Foreground Background

4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:

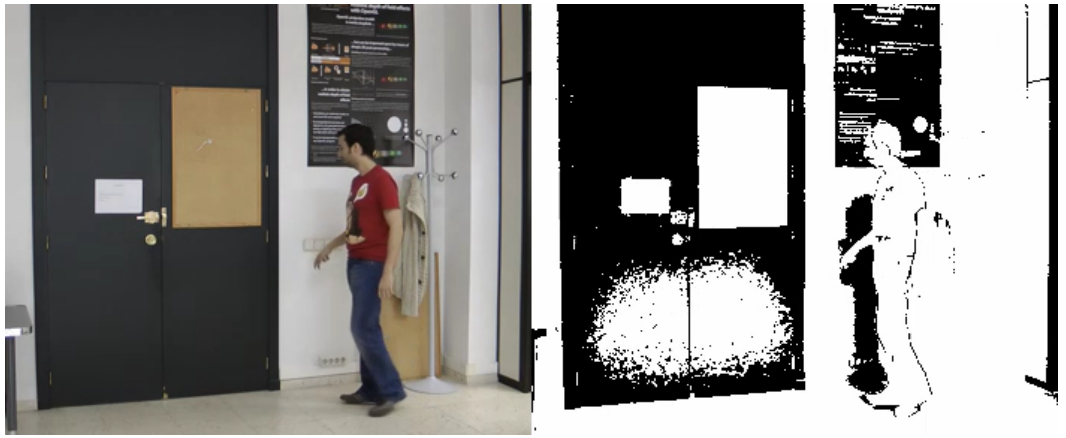
4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:
    - Image noise.



4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:
    - Image noise.
    - Illumination changes.

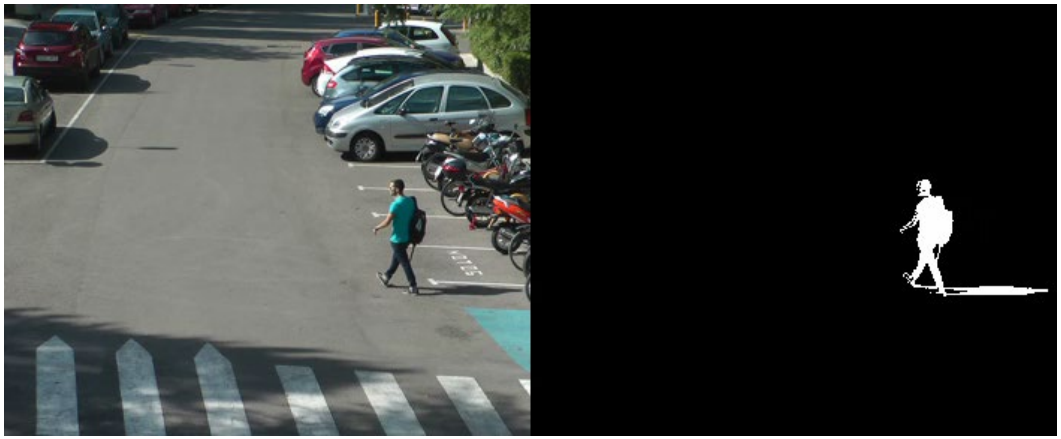


4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:
    - Image noise.
    - Illumination changes.
    - Background changes.

4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:
    - Image noise.
    - Illumination changes.
    - Background changes.
    - Shadows.
    - Etc.



4.2.1 Introduction

- **Background subtraction:**
  - It is the typical process used to separate background and foreground.
  - The most basic techniques are based on the hypothesis that the background does not vary significantly.
  - However, this rarely occurs:
    - Image noise.
    - Illumination changes.
    - Background changes.
    - Shadows.
    - Etc.

More complex techniques, capable of modeling the background variations, are required.

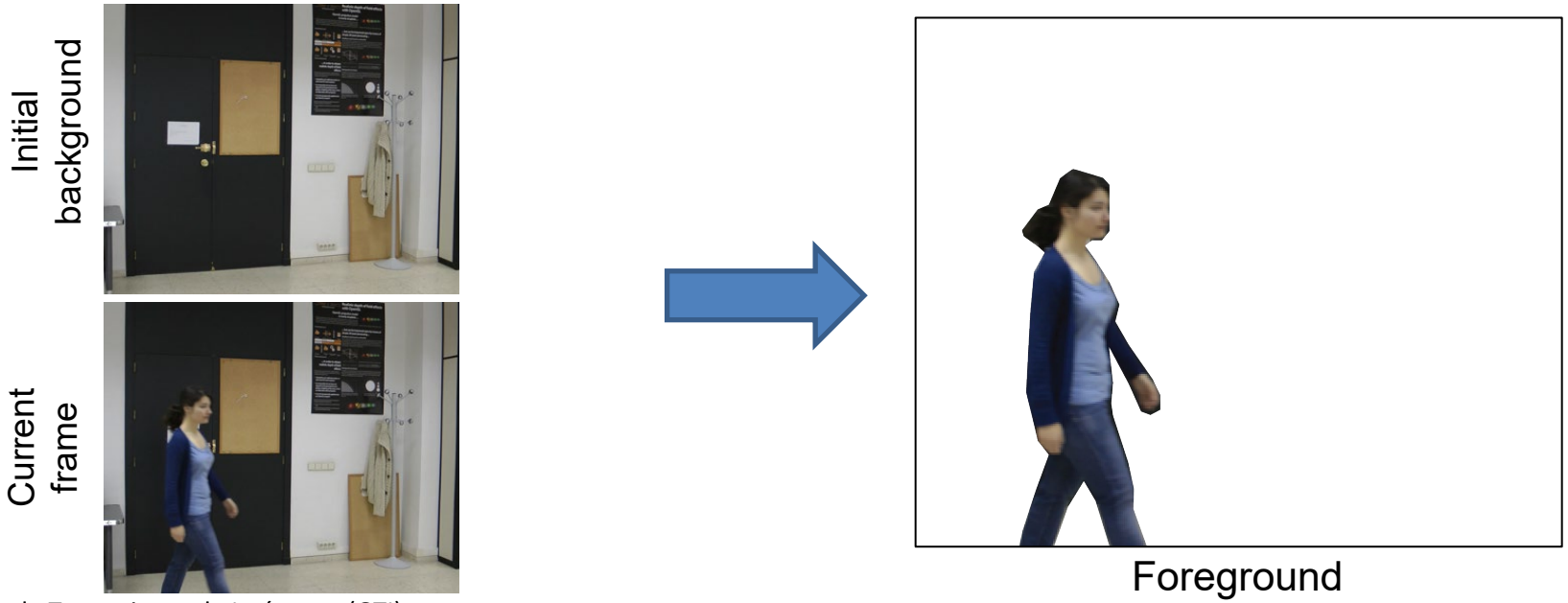
4.2.1 Introduction

- **Stages in background subtraction:**
  1. Background initialization:
     - An initial background model is constructed from one or more frames (typically, frames without foreground objects are required).
     - This initial model can be designed by multiple ways: Heuristic, statistical, fuzzy logic, etc.

Initial background

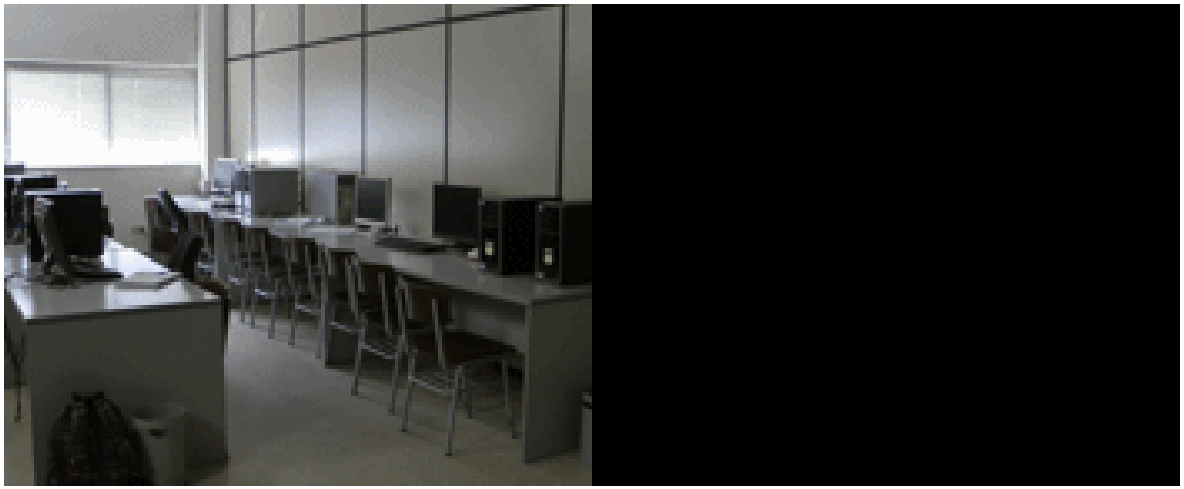
4.2.1 Introduction

- **Stages in background subtraction:**
  2. Foreground detection:
     - In the next frames, a comparison between the current frame and the background model is carried out.
       - The simplest comparison is the difference between both images.
     - The result of the comparison is the foreground of the scene.



4.2.1 Introduction

- **Stages in background subtraction:**
  3. Background maintenance:
     - The background model is updated along time to adapt the changes in the background: Illumination changes, removed objects, etc.
     - This stage is not mandatory.



4.2.2 Difference-based detection

- It is the most basic (intuitive) detection.
- Compute the difference between consecutive frames.

$$\mathbf{D}_n(i,j) = |\mathbf{I}_n(i,j) - \mathbf{I}_{n-1}(i,j)|$$

  - (i,j) → Pixel coordinates (row and column, respectively).
  - $\mathbf{I}_n$ → Current frame, at time n.
  - $\mathbf{I}_{n-1}$ → Reference (previous) frame, at time $n-1$.
  - $\mathbf{D}_n$ → Current difference frame.

- Compare the differences with a threshold value Th:

$$\boldsymbol{\varphi}_n(i,j) = \begin{cases} 1 & \text{if } \mathbf{D}_n(i,j) > Th \\ 0 & \text{if } \mathbf{D}_n(i,j) \leq Th \end{cases}$$

  - $\boldsymbol{\varphi}_n$ → Current foreground mask.

4.2.2 Difference-based detection

- **Results in a simple sequence:**

Original sequence

Differences

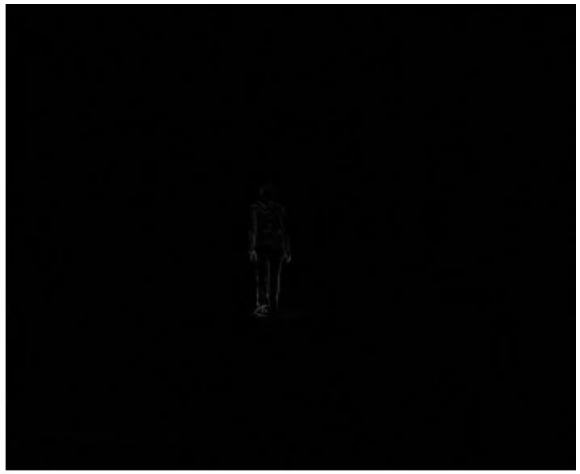
Foreground (low threshold)

Foreground (high threshold)

4.2.2 Difference-based detection

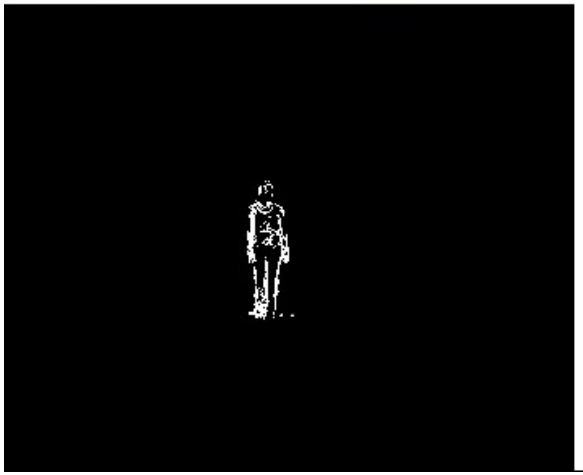
- **Results in a sequence with objects moving along the camera axis:**

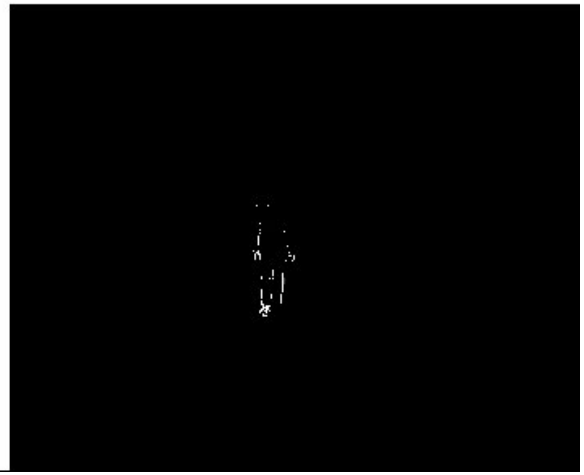
Original sequence



Differences

Foreground (low threshold)





Foreground (high threshold)

4.2.2 Difference-based detection

- An alternative is not to update the background model → Use always the same image (e.g., the first image in the sequence).

$$\mathbf{D}_n(i,j) = |\mathbf{I}_n(i,j) - \mathbf{I}_1(i,j)|$$

$$\boldsymbol{\varphi}_n(i,j) = \begin{cases} 1 & \text{if } \mathbf{D}_n(i,j) > Th \\ 0 & \text{if } \mathbf{D}_n(i,j) \leq Th \end{cases}$$

  - Advantages:
    - The detections will be more compact (less misdetections).
  - Disadvantages:
    - A change in the background (e.g., an illumination change) will result in persistent false detections.

4.2.2 Difference-based detection

Original sequence

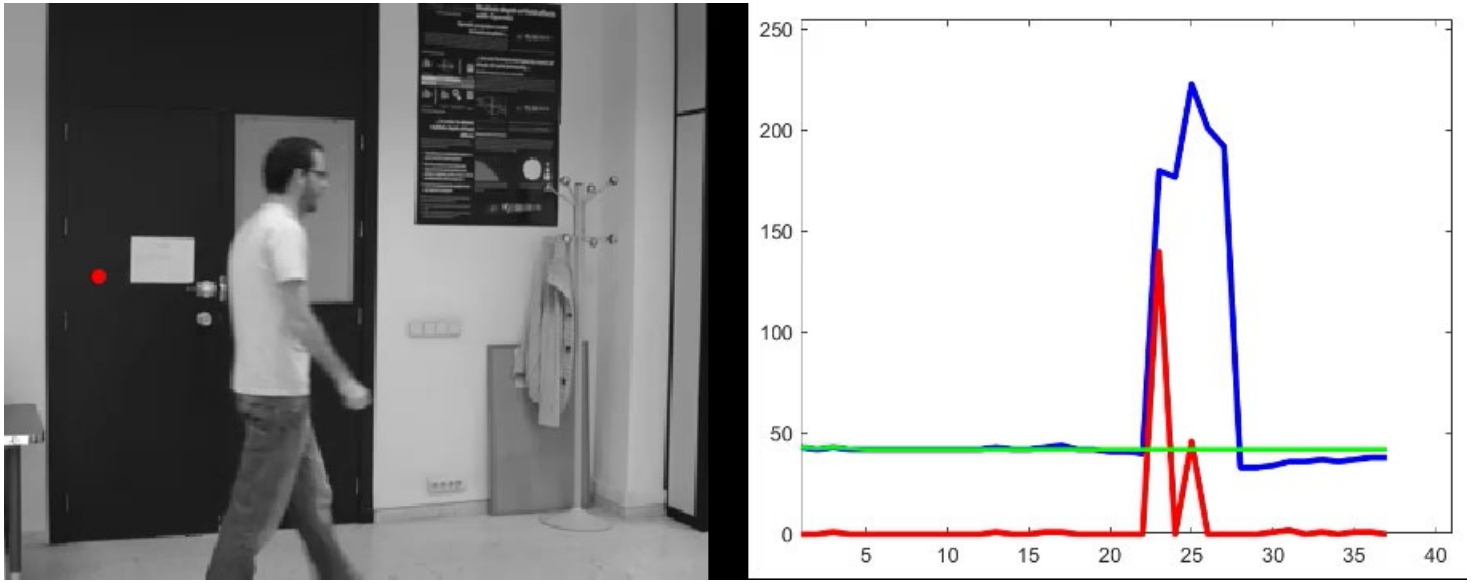
Foreground (differences between consecutive frames)

Foreground (differences with the first frame)

4.2.3 Temporal median filtering

- On the one hand, we want to update the background model to consider the changes in the background.
  - In this way, the persistent false detections will be avoided.
- On the other hand, we do not want to compare temporarily-close frames.
  - We want to detect compact foreground objects.
- Possible solution → **Compute the background model as the median of the pixel values in a set of previous frames**:
    - The median provides a good representation of a "typical" value in a dataset.
  - Pixel values corresponding to foreground objects will not affect to the median:
    - The detections will be compact.
  - By updating the background model, the persistent false detections will be avoided.

4.2.3 Temporal median filtering



- Blue: Luminance.
- Red: Differences between consecutive frames.
- Green: Median of the 30 previous frames.

4.2.3 Temporal median filtering

- At each instant n and for each pixel position (i,j), compute the median of the values at such position in the set of N previous frames.

$$\mathbf{M}_n(i,j) = \text{median}\{\mathbf{I}_{n-N}(i,j), \cdots, \mathbf{I}_{n-1}(i,j)\}$$

  - **This median is the background model**.

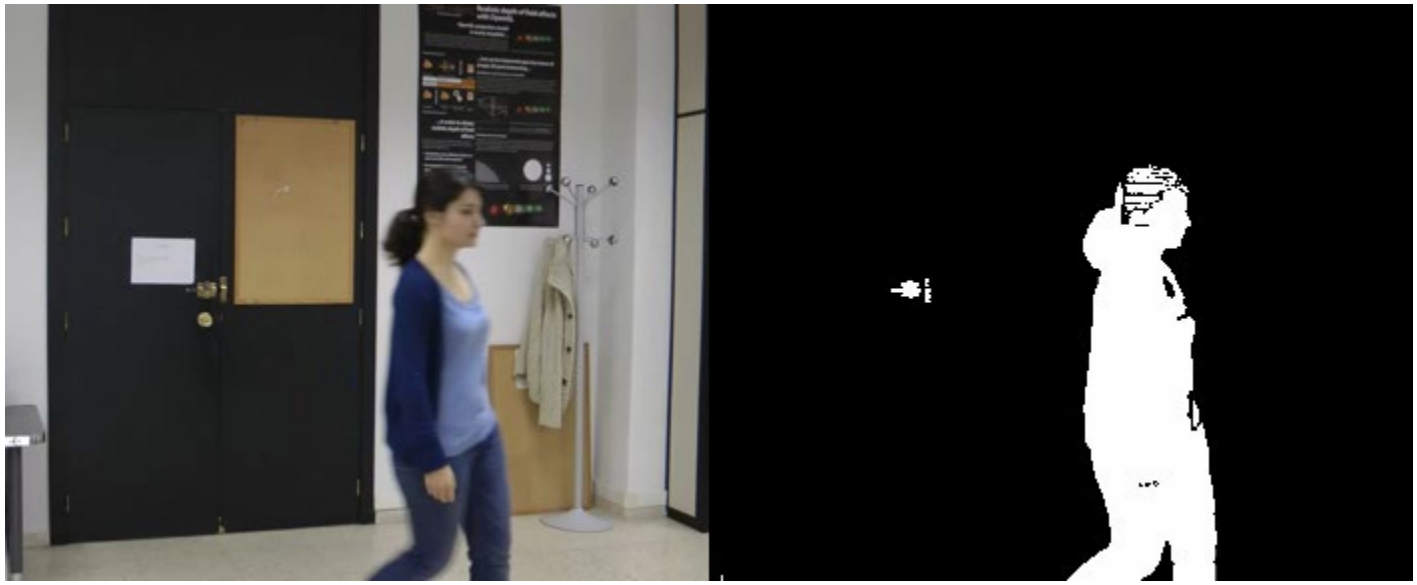
- Compute the absolute difference between the current image and the background model:

$$\mathbf{D}_n(i,j) = |\mathbf{I}_n(i,j) - \mathbf{M}_n(i,j)|$$

- Compare the differences with a threshold value Th:

$$\boldsymbol{\varphi}_n(i,j) = \begin{cases} 1 & \text{if } \mathbf{D}_n(i,j) > Th \\ 0 & \text{if } \mathbf{D}_n(i,j) \leq Th \end{cases}$$

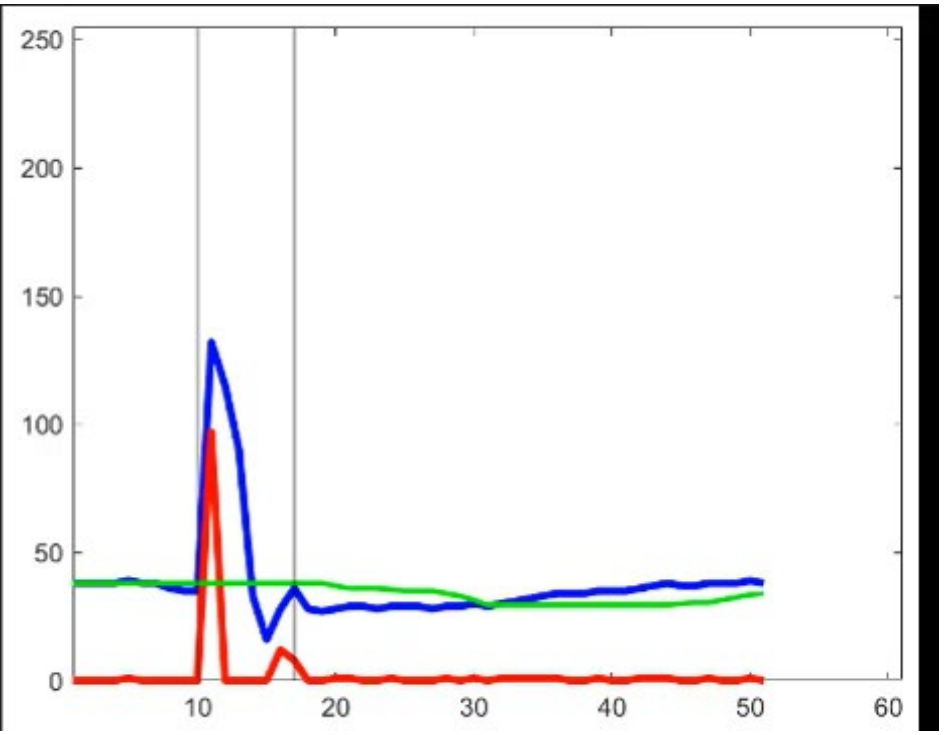
4.2.3 Temporal median filtering



- The segmented objects are more compact → Less misdetections.
- Most camera noise is avoided → Less false detections.

What is the reason of the holes (misdetections)?

4.2.3 Temporal median filtering

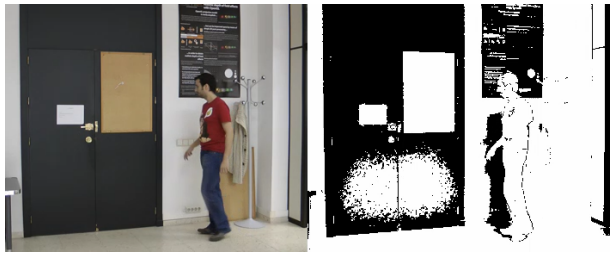


What is the reason of the holes (misdetections)?

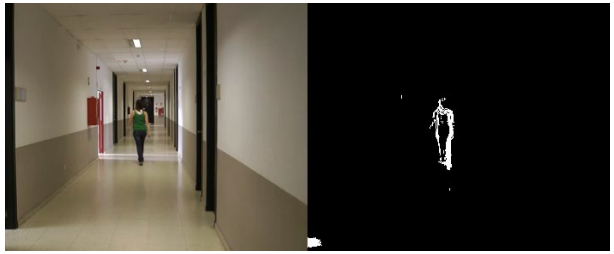
How can they be solved?

4.2.3 Temporal median filtering

- What are the problems?
  1. It is required to store many reference images → Memory cost.
  2. The computation of the median is complex → Computational cost.
  3. The adequate length of the buffer is crucial for a correct performance:
     - Large buffers lead to false detections (false positives) when the background changes.



     - Short buffers result in misdetections (false negatives) when the foreground objects move slow or along the optical axis of the camera.

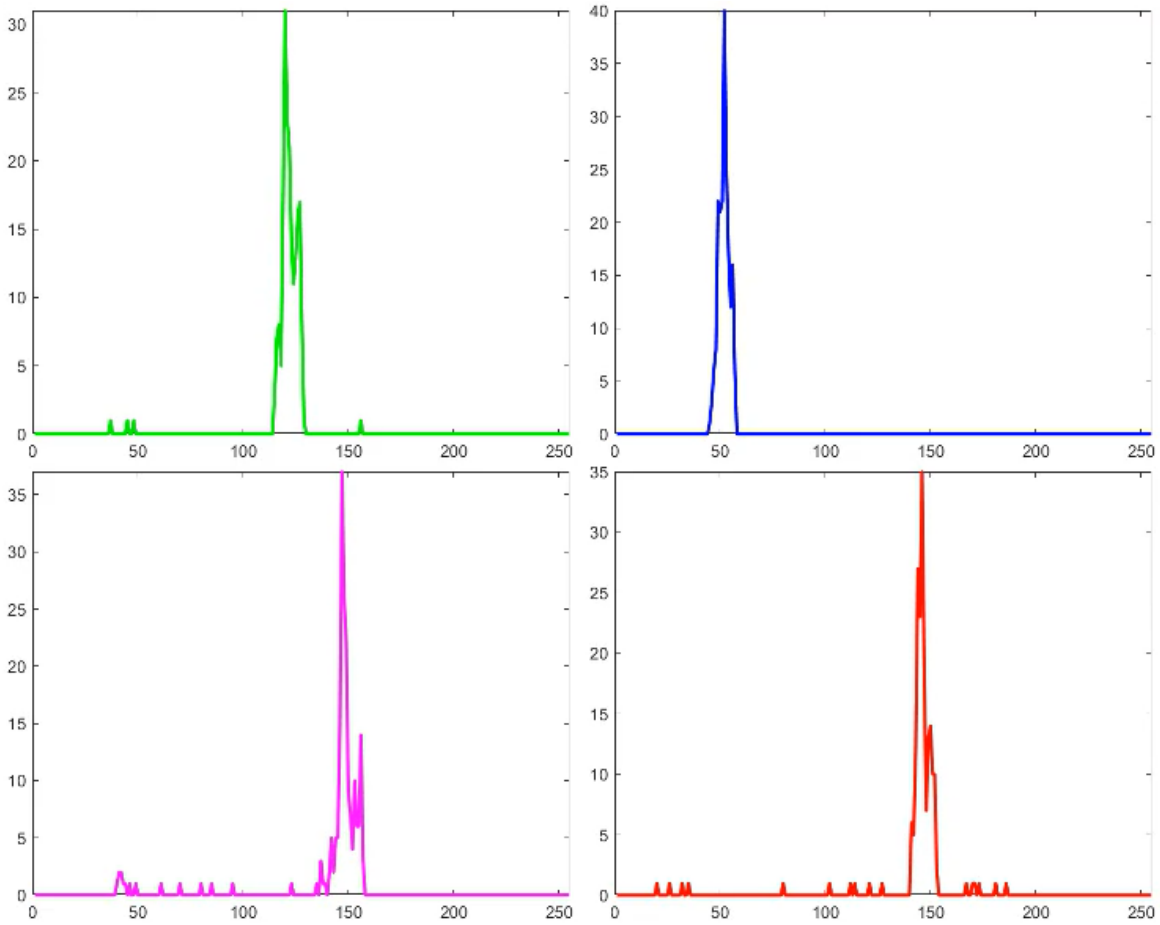


4.2.3 Temporal median filtering

- What are the problems?
  - Therefore, not only selecting an adequate threshold is crucial, but also selecting an adequate buffer size.
    - Worse usability.
  - The memory and computational costs are high.

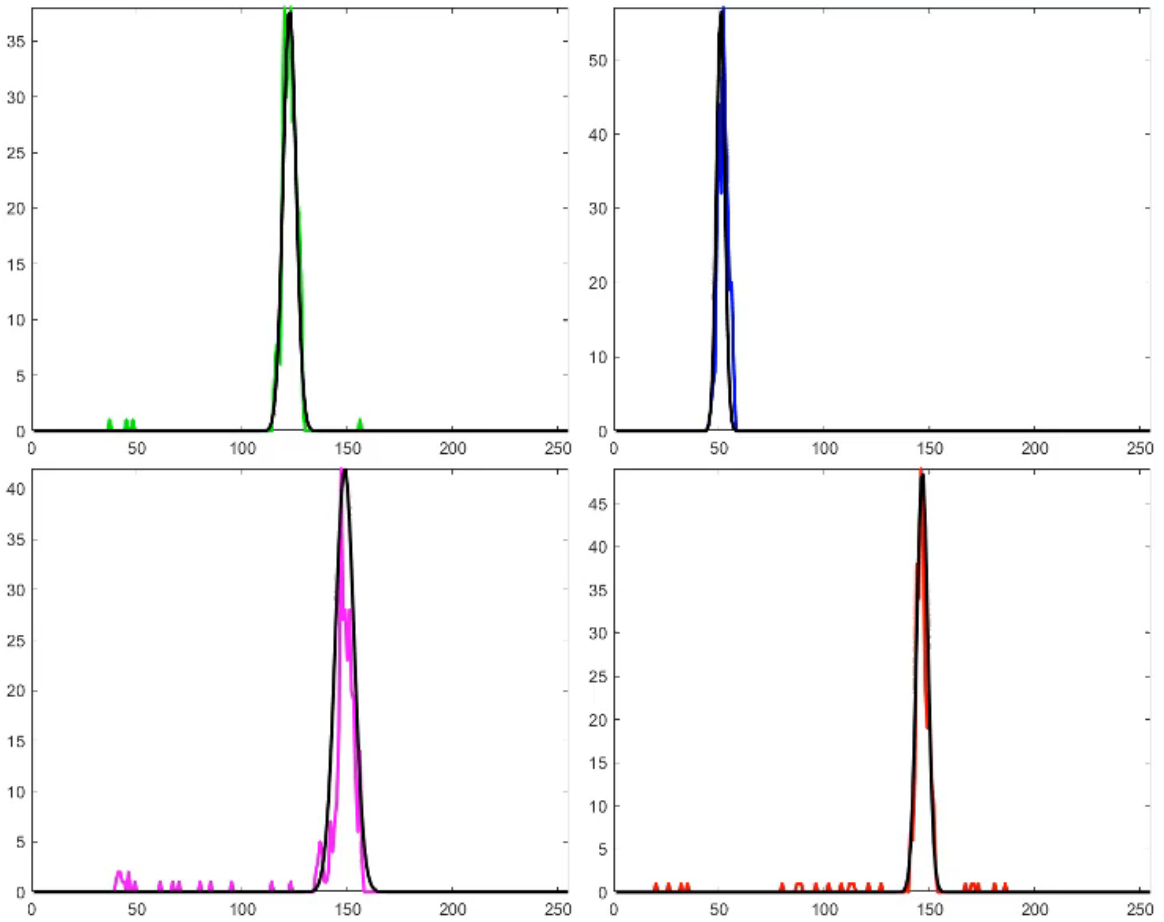
4.2.4 Single Gaussian Modeling

- Let us analyze the histogram of the luminance values of some pixels throughout a video sequence:



4.2.4 Single Gaussian Modeling

- Let us analyze the histogram of the luminance values of some pixels throughout a video sequence:



4.2.4 Single Gaussian Modeling

- **Algorithm:**
  1. Initialization of the background model.
  - For each new image:
    2. Update of the background model.
    3. Foreground detection → Segmentation.

- **Definitions and notation:**
  - x_n → Luminance value of the pixel at coordinates (i, j), at instant n.
  - It is assumed that the background can be modeled with Gaussian probability density functions (pdfs). Then, the probability of x_n of being part of the background is: $\mathrm{p}(x_n) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(\mu_n - x_n)^2}{2(\sigma_n)^2}\right)$
    - μ_n → Mean of the Gaussian pdf at time n.
    - σ_n → Standard deviation of the Gaussian pdf at time n.

4.2.4 Single Gaussian Modeling

1. **Initialization:**
   - Option 1: Using a training period.
     - The mean and variance of the Gaussians are estimated from all the pixel values in the N_0 first frames of the sequence.

$$\mu_{N_0} = \frac{1}{N_0}\sum_{n=1}^{N_0} x_n$$

$$\sigma_{N_0}^2 = \frac{1}{N_0}\sum_{n=1}^{N_0} \left(\mu_{N_0} - x_n\right)^2$$

4.2.4 Single Gaussian Modeling

1. **Initialization:**
   - Option 2: Without a training period.
     - The means of the Gaussians are set as the pixel values in the first frame of the sequence.

$$\mu_1 = x_1$$

     - The variances of the Gaussians are manually set by the user.

$$\sigma_1^2 = \sigma_0^2$$

4.2.4 Single Gaussian Modeling

2. Update of the background model:

- For each new frame, the mean and variance values are updated as follows:

$$\mu_n = \mu_{n-1}(1-\alpha) + \alpha x_n$$

$$\sigma_n^2 = \sigma_{n-1}^2(1-\alpha) + \alpha(\mu_n - x_n)^2$$

- α → Learning rate.
  - The update speed of the Gaussians is proportional to the value of this parameter.
  - Typical values → $\alpha \in (10^{-4}, 10^{-2})$

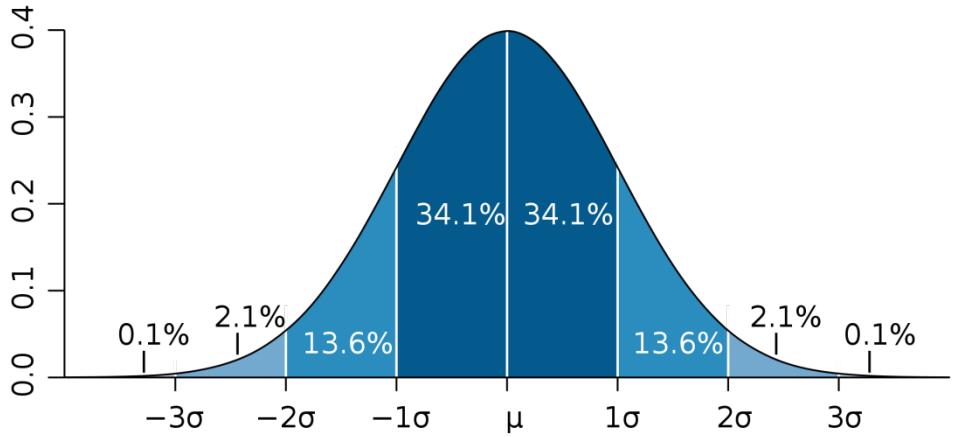
4.2.4 Single Gaussian Modeling

3. Foreground segmentation:

- A pixel is classified as foreground if:

$$|x_n - \mu_n| \geq k\sigma_n$$

- If $k = 3$ → We are classifying as background approximately the 99% of the pixels covered by the Gaussian.



4.2.4 Single Gaussian Modeling

3. Foreground segmentation:

- If we use multidimensional data (e.g., RGB or YCrCb), the condition is related to the Mahalanobis distance:

$$D_{mah,n,D} = \sqrt{\sum_{d=1}^{D} \frac{(x_n(d) - \mu_n(d))^2}{\sigma_n^2(d)}} \geq k$$

- Where D is the number of the components representing the data (e.g., $D = 3$ for RGB color).
- The value assigned to k to consider the 99% of the probability of the Gaussian depends on D:
  - If $D = 1 \rightarrow k = 3$
  - If $D = 2 \rightarrow k = 3.44$
  - If $D = 3 \rightarrow k = 3.76$

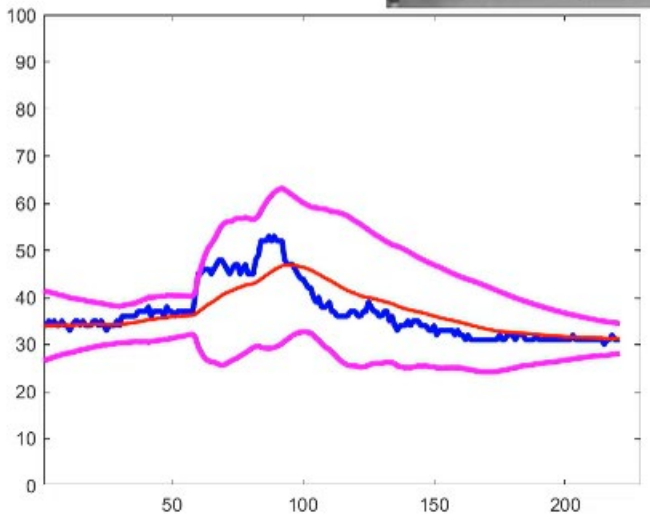
4.2.4 Single Gaussian Modeling

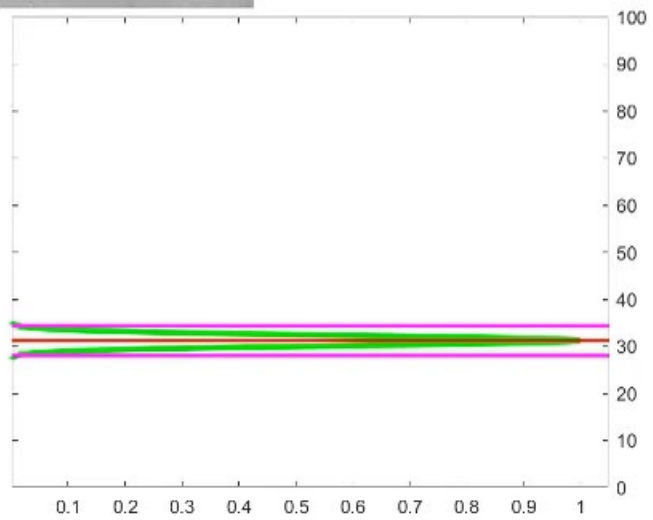
- **Update example: Illumination change**

$\alpha = 10^{-2}$

σ_1 =2.55 (1%)

μ_1 =34





- Blue: Luminance.
- Green: $N(\mu_n, \sigma_n)$
- Red: μ_n.
- Purple: Limits $\frac{|x_n - \mu_n|}{\sigma_n} \geq 3$

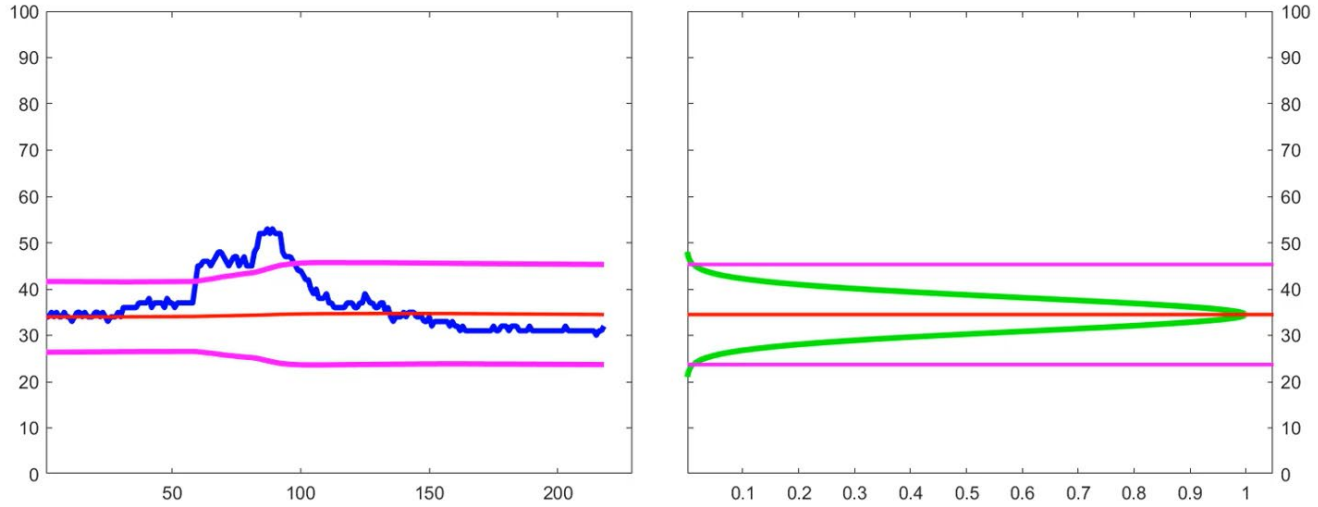
4.2.4 Single Gaussian Modeling

- **Update example: Illumination change**

$\alpha = 10^{-3}$

σ_1 =2.55 (1%)

μ_1 =34



- Blue: Luminance.
- Green: $N(\mu_n, \sigma_n)$
- Red: μ_n.
- Purple: Limits $\frac{|x_n - \mu_n|}{\sigma_n} \geq 3$

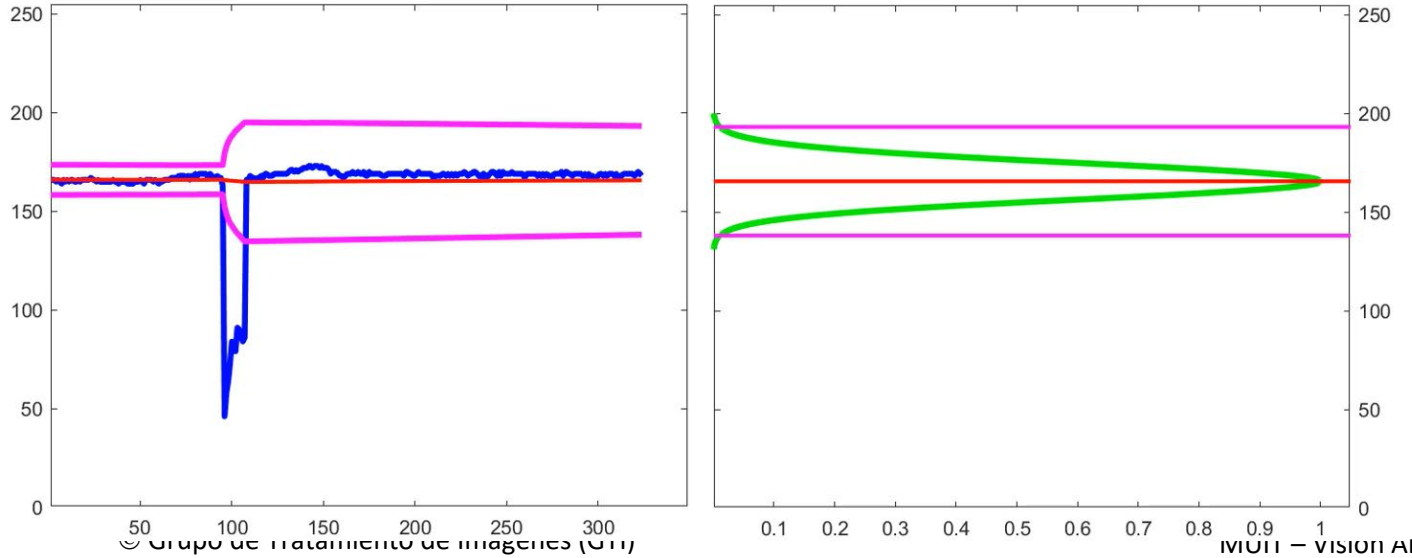
4.2.4 Single Gaussian Modeling

- **Update example: Short duration change**

$\alpha = 10^{-3}$

σ_1 =2.55 (1%)

μ_1 =161



- Blue: Luminance.
- Green: $N(\mu_n, \sigma_n)$
- Red: μ_n.
- Purple: Limits $\frac{|x_n - \mu_n|}{\sigma_n} \geq 3$

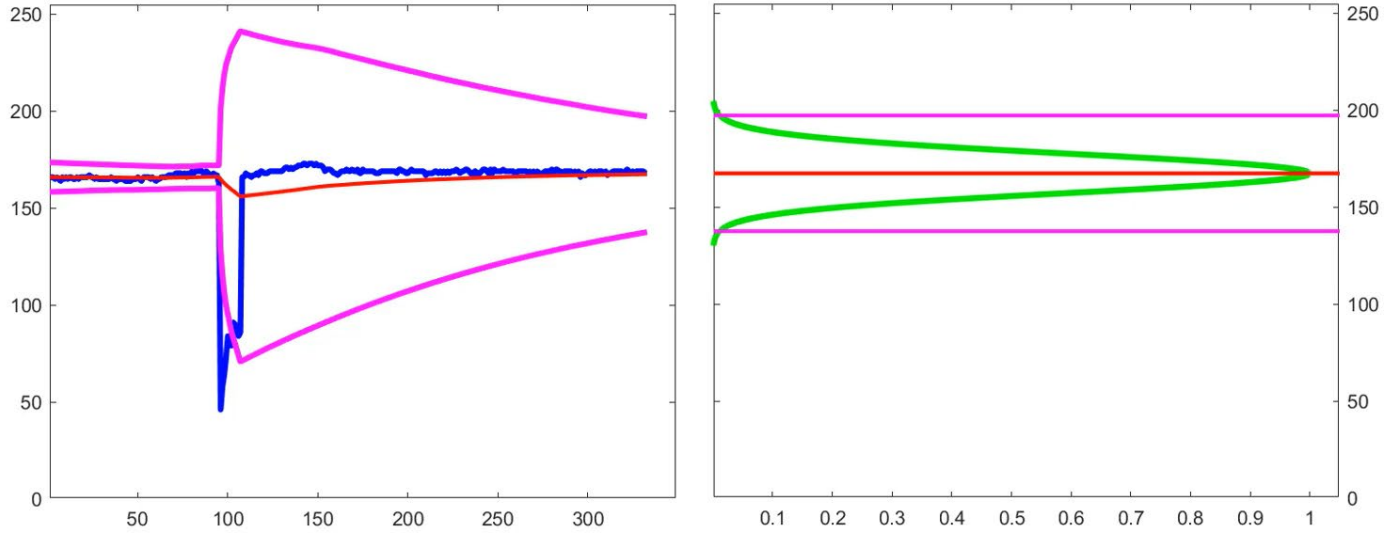
4.2.4 Single Gaussian Modeling

- **Update example: Short duration change**

$\alpha = 10^{-2}$

σ_1 =2.55 (1%)

μ_1 =161



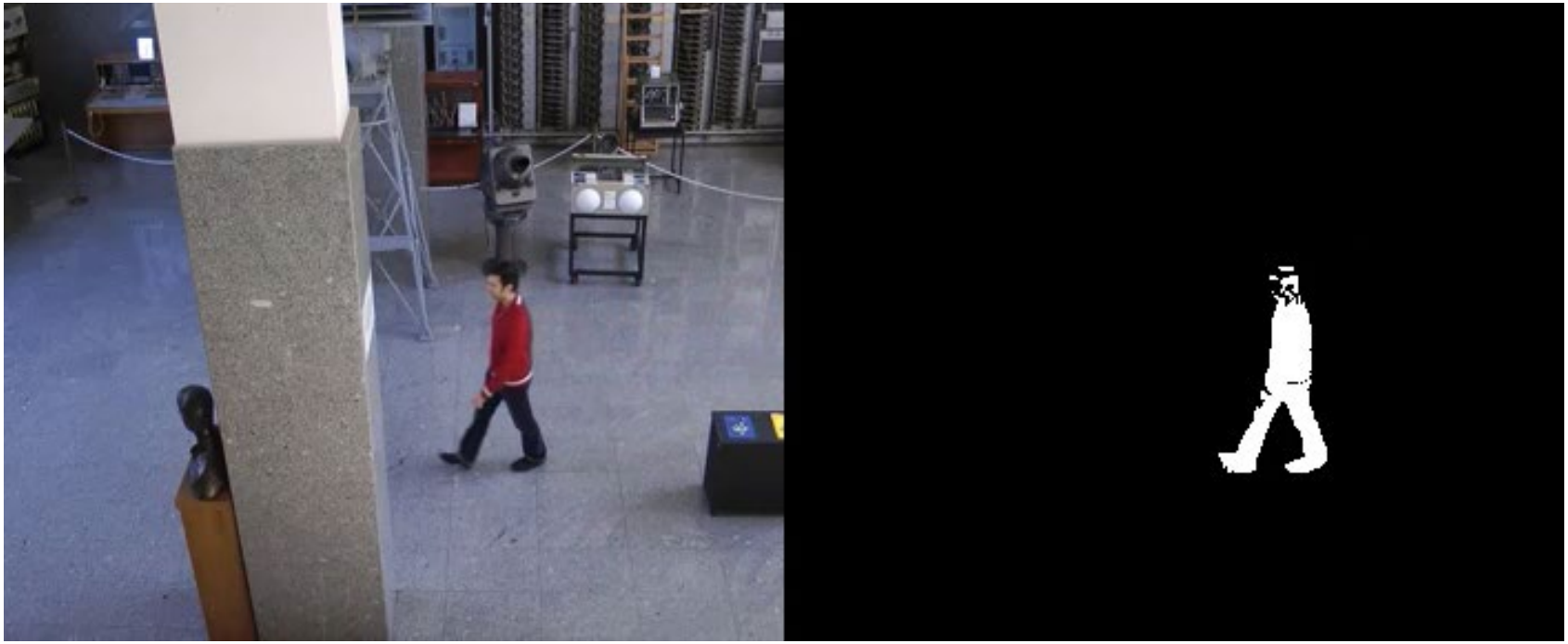
- Blue: Luminance.
- Green: $N(\mu_n, \sigma_n)$
- Red: μ_n.
- Purple: Limits $\frac{|x_n - \mu_n|}{\sigma_n} \geq 3$

4.2.4 Single Gaussian Modeling

- **Results: Simple sequence**

4.2.4 Single Gaussian Modeling

- **Results: Simple sequence**



4.2.4 Single Gaussian Modeling

- **Strengths:**
  - Low memory requirements:
    - It depends on few reference data → Two parameters for each image pixel:
      - Mean and variance.
  - Low computational cost:
    - Mean update → Two products and a sum.
    - Variance update → Two products and a sum.
  - The Mahalanobis distance as thresholding criteria.

4.2.4 Single Gaussian Modeling

- **Drawbacks:**
  - The selection of an adequate learning rate is critical:
    - High learning rates → Misdetections if the foreground objects move slowly or along the camera optical axis.

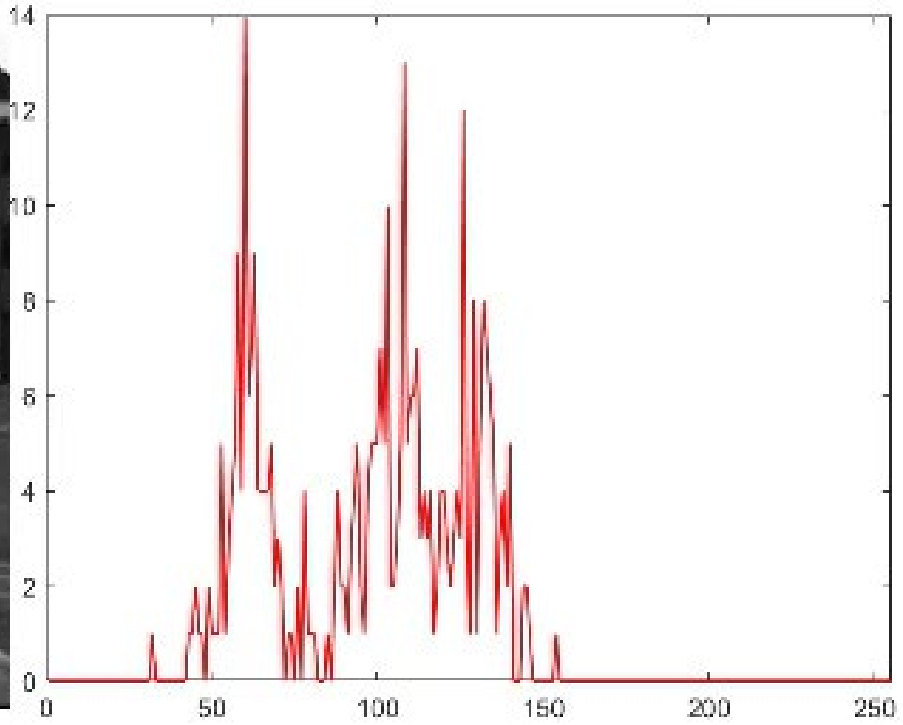
    - Low learning rates → False detections if the background changes suddenly.

4.2.4 Single Gaussian Modeling

- **Additional drawback: Multimodal backgrounds**

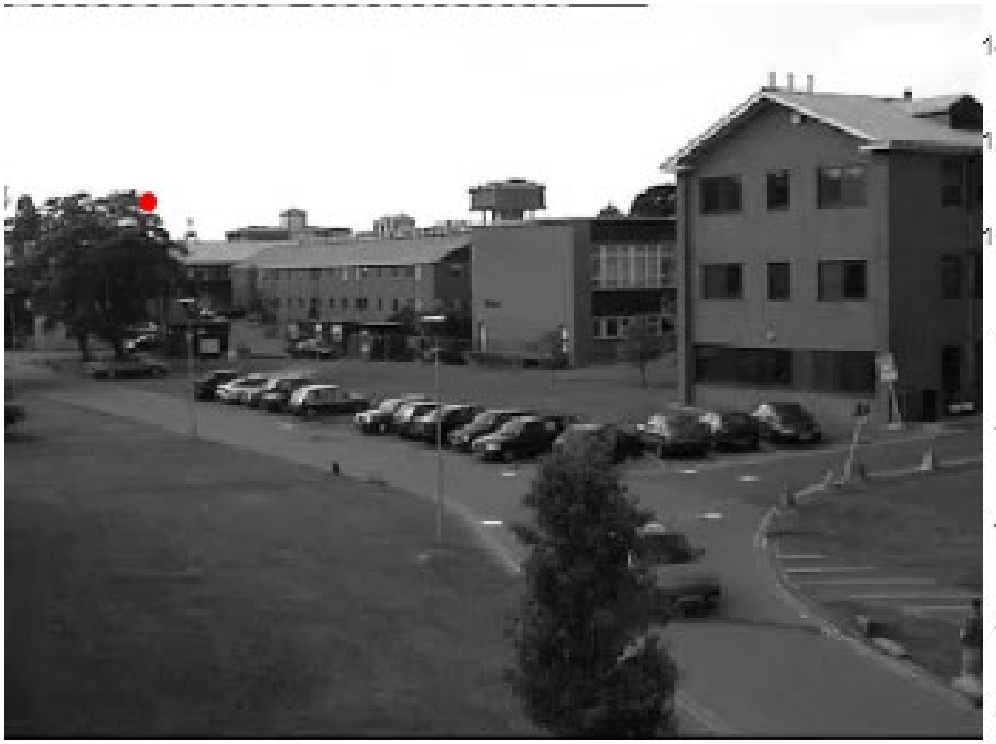
4.2.4 Single Gaussian Modeling

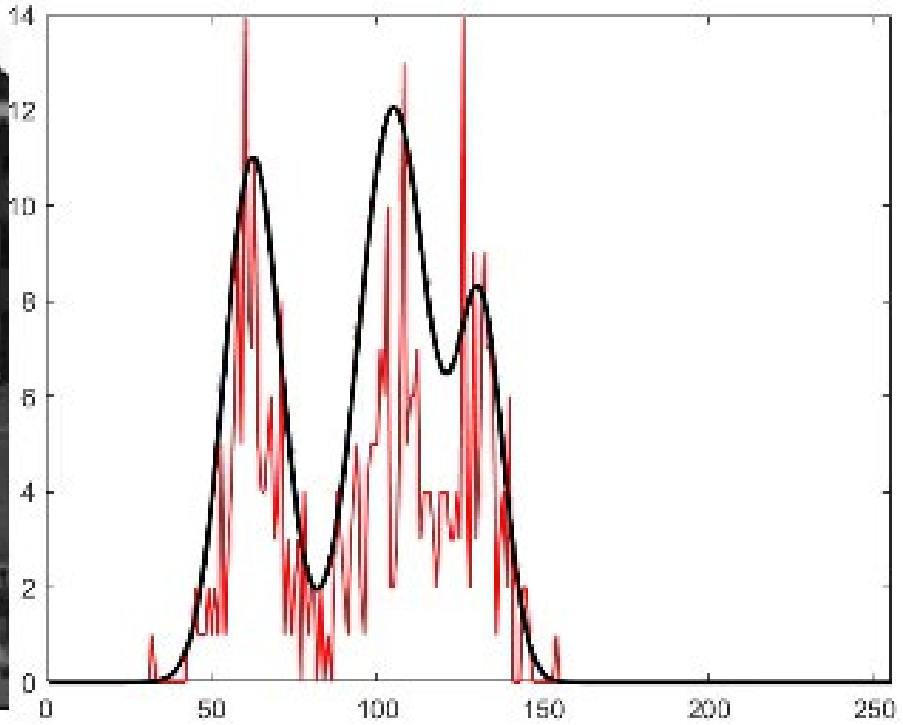
- **Additional drawback: Multimodal backgrounds**



4.2.4 Single Gaussian Modeling

- **Additional drawback: Multimodal backgrounds**





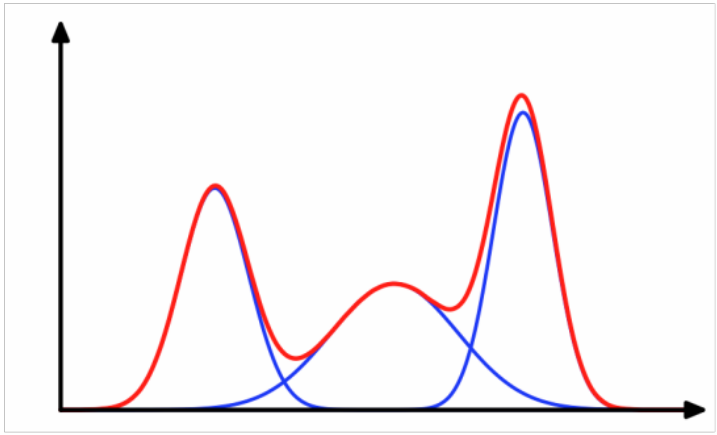
4.2.5 Gaussian Mixture Models (GMMs)

- The variations of the pixel values are modeled by a weighted sum of K Gaussian functions:

$$\mathrm{p}(x_n) = \sum_{k=1}^{K} w_{n,k}\, \eta_{n,k}\left(x_n, \mu_{n,k}, \sigma_{n,k}^2\right)$$

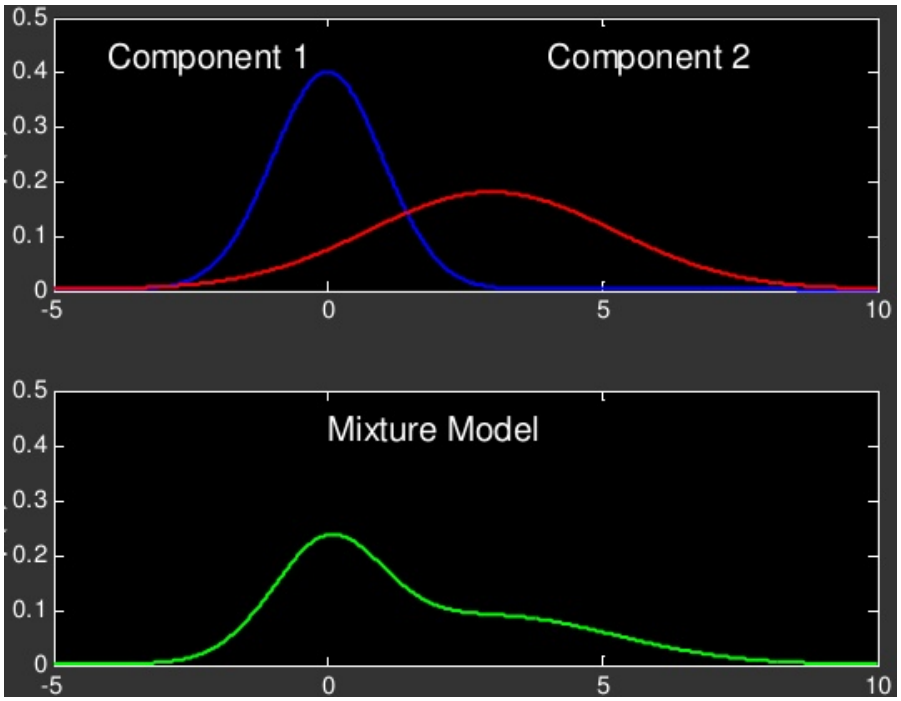
  - x_n → Value of the pixel (e.g., luminance) at coordinates (i, j), at instant n.
  - $w_{n,k}$ → Weight of the k-th Gaussian distribution at instant n.
  - $\eta_{n,k}(\cdot)$ → k-th Gaussian distribution at instant n.

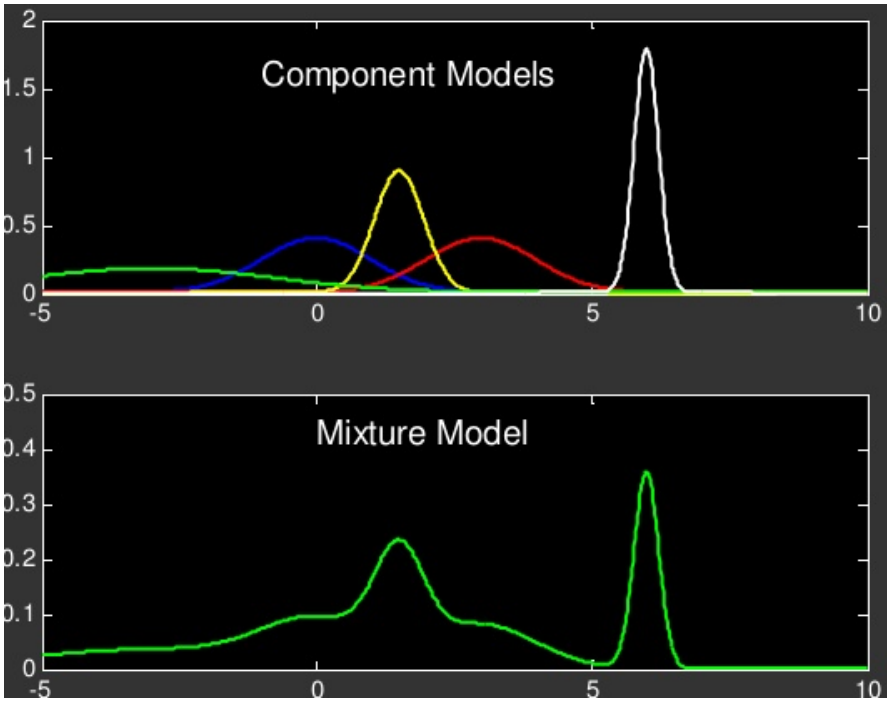
$$\eta_{n,k} = \frac{1}{\sqrt{2\pi}\sigma_{n,k}} \exp\left(-\frac{\left(\mu_{n,k} - x_n\right)^2}{2\left(\sigma_{n,k}\right)^2}\right)$$



4.2.5 Gaussian Mixture Models (GMMs)

- Examples:





4.2.5 Gaussian Mixture Models (GMMs)

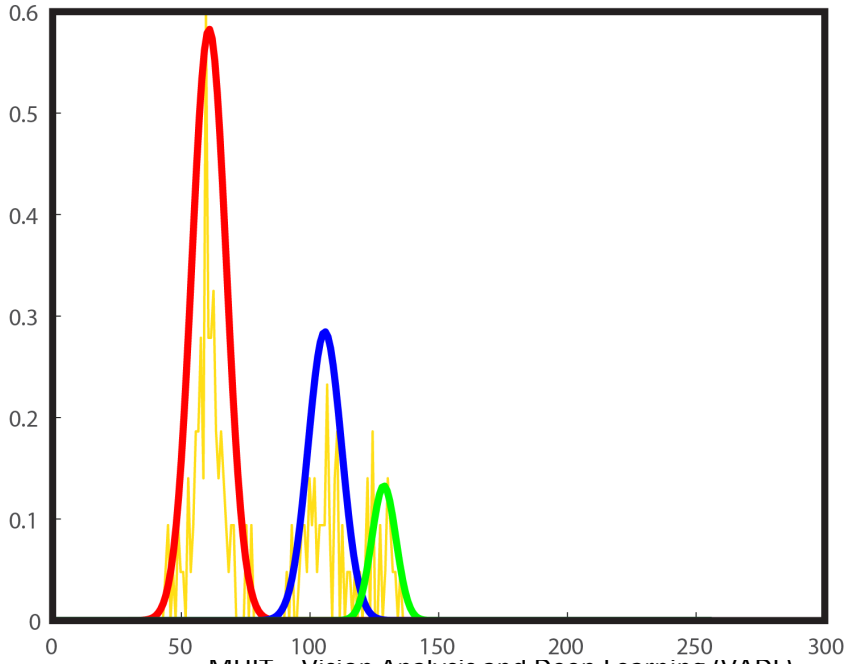
- **Stages:**
  1. Initialization.
  2. Identification:
     - The capability of the Gaussians for representing the current pixel values is analyzed.
  3. Update:
     - The parameters of the Gaussians are updated to deal with background changes.
  4. Classification:
     - Each pixel is classified as part of the background or foreground, depending on how they are represented by the Gaussian mixture.

Loop

4.2.5 Gaussian Mixture Models (GMMs)

1. **Initialization:**

   - Option 1: The mean and variance of the Gaussians are estimated from the pixel values in the N_0 first frames of the sequence.
     - This is typically done using an E-M (Expectation-Maximization) algorithm → Iterative method for finding the mixture model that best represents the data.



4.2.5 Gaussian Mixture Models (GMMs)

1. **Initialization:**

   - Option 2: Only for the first Gaussian in each model, the means are set as the pixel values in the first frame of the sequence and the variances are manually set by the user:

$$\mu_{1,1} = x_1$$

$$\sigma_{1,1}^2 = \sigma_0^2$$

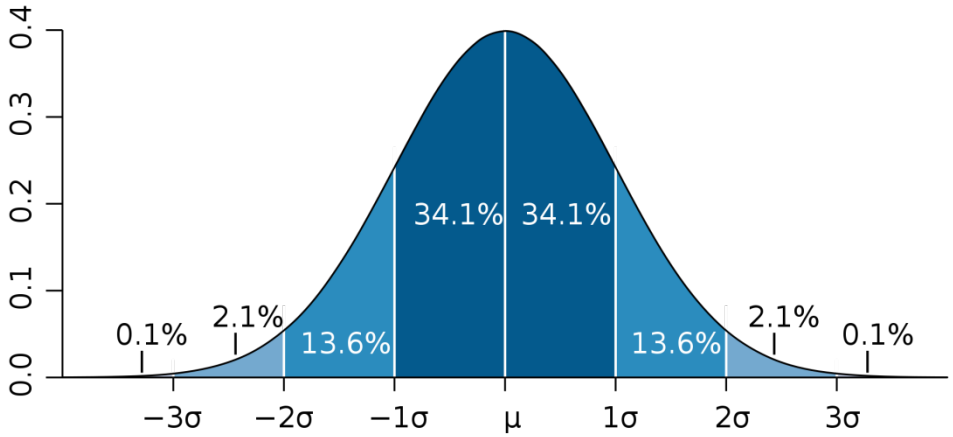
$$w_{1,1} = 1$$

4.2.5 Gaussian Mixture Models (GMMs)

2. Identification:

- At each new frame, every new pixel value is checked against the existing K Gaussian distributions.
  - A sample is represented by a Gaussian if:

$$D_{mah,n,k} = \frac{|x_n - \mu_{n-1,k}|}{\sigma_{n-1,k}} < 3$$



2. Identification:

- If multidimensional data is used (e.g., RGB or YCrCb), the condition is related to the Mahalanobis distance:

$$D_{mah,n,D} = \sqrt{\sum_{d=1}^{D} \frac{(x_n(d) - \mu_{n-1}(d))^2}{\sigma_{n-1}^2(d)}} \geq k$$

- Where D is the number of the components representing the data (e.g., $D = 3$ for RGB color).
- The value assigned to k to consider the 99% of the probability of the Gaussian depends on D:
  - If $D = 1 \rightarrow k = 3$
  - If $D = 2 \rightarrow k = 3.44$
  - If $D = 3 \rightarrow k = 3.76$

4.2.5 Gaussian Mixture Models (GMMs)

2. Identification:

- If the current value is not represented by any of the Gaussian distributions:
  - If there are uninitialized Gaussians (i.e., the previous data have been modeled with less than K Gaussians) → A new Gaussian is created.
  - If there are not uninitialized Gaussians → The least probable distribution (i.e., that with the minimum ratio $r_{n,k} = \frac{w_{n,k}}{\sigma_{n,k}}$) is replaced with a new one.
  - In any case, the new Gaussians are initialized with:
    - Mean → The current pixel value.
    - Variance → Predefined value (typically, a high value).
    - Weight → Predefined value (typically, a low value).
- If a match has been defined, we go to stage 3 (update).

4.2.5 Gaussian Mixture Models (GMMs)

3. Update:

- The mean and the variance of the Gaussian distributions representing the current pixel values are updated as follows:

$$\mu_{n,k} = (1 - \rho_{n,k}) \cdot \mu_{n-1,k} + \rho_{n,k} \cdot x_{n,k}$$

$$\sigma^2_{n,k} = (1 - \rho_{n,k}) \cdot \sigma^2_{n-1,k} + \rho_{n,k}(\mu_{n,k} - x_{n,k})^2$$

- Where $\rho_{n,k}$ is computed from the learning rate α as follows:

$$\rho_{n,k} = \alpha \cdot \exp\left(-\frac{(\mu_{n-1,k} - x_n)^2}{2(\sigma_{n-1,k})^2}\right)$$

→ The update will be slower if the pixel value is far from the mean of the Gaussian.

3. Update:

- The weights of all the Gaussian distributions are updated as follows:

$$w'_{n,k} = (1 - \alpha) \cdot w_{n-1,k} + \alpha \cdot M_{n,k}$$

- Where:
  - $M_{n,k} = 1$ for the models that represent the current pixel values (stage 2).
    - The weight of the Gaussians will increase.
  - $M_{n,k} = 0$ for the remaining models.
    - The weight of the Gaussians will decrease.

- Finally, the weights are normalized:

$$w_{n,k} = \frac{w'_{n,k}}{\sum_{k=1}^{K} w'_{n,k}}$$

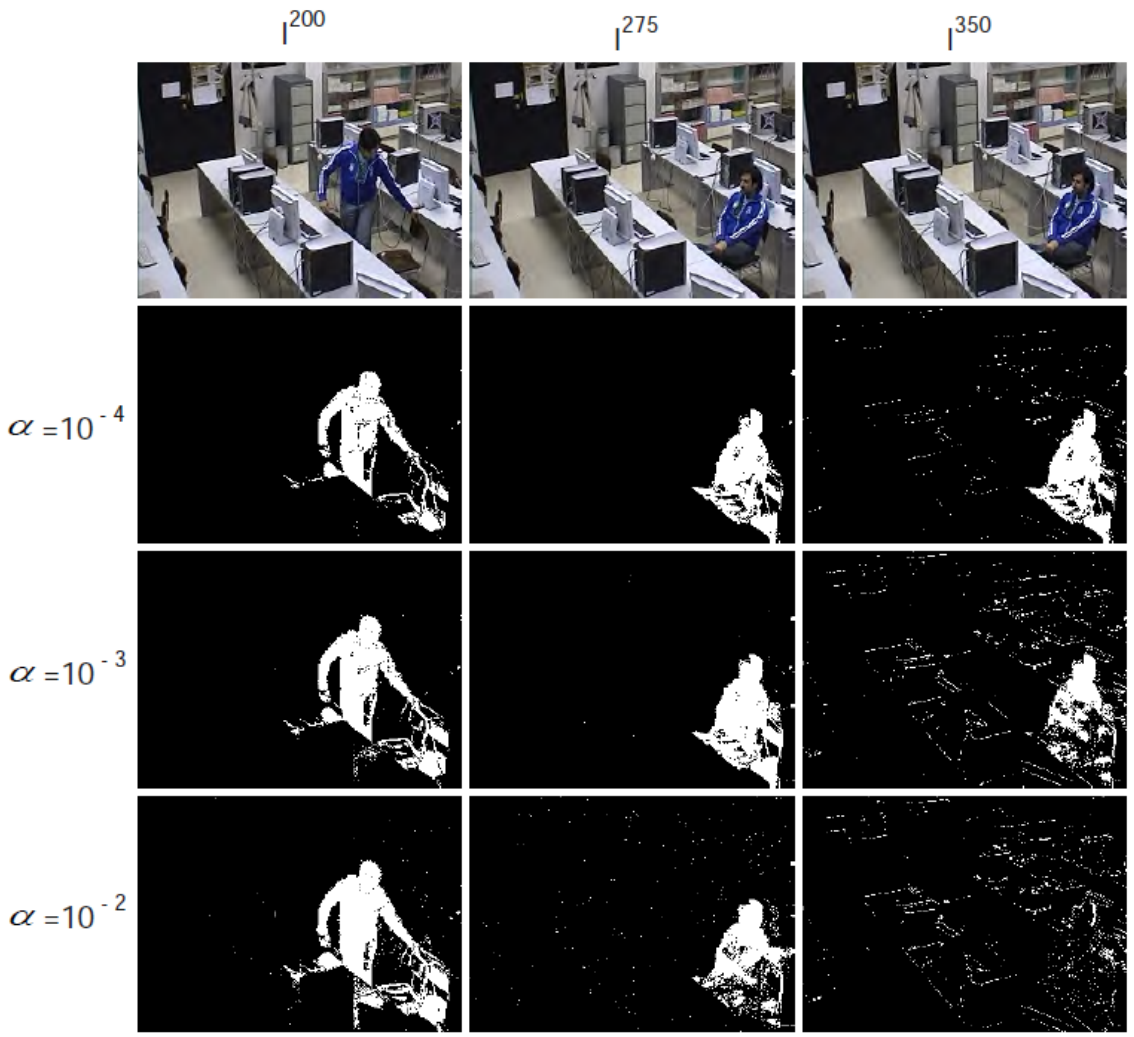
3. **Update:**
   - Again, the selection of an adequate learning rate α is critical:
     - High learning rates → The Gaussian distributions change fast:
       - Misdetections if the foreground objects move slowly or along the camera optical axis.
     - Low learning rates → The Gaussian distributions change slowly.
       - False detections if the background changes suddenly.

4.2.5 Gaussian Mixture Models (GMMs)

3. Update:

Example:

Sequence with a foreground object that becomes temporarily static (a person sits on a chair).

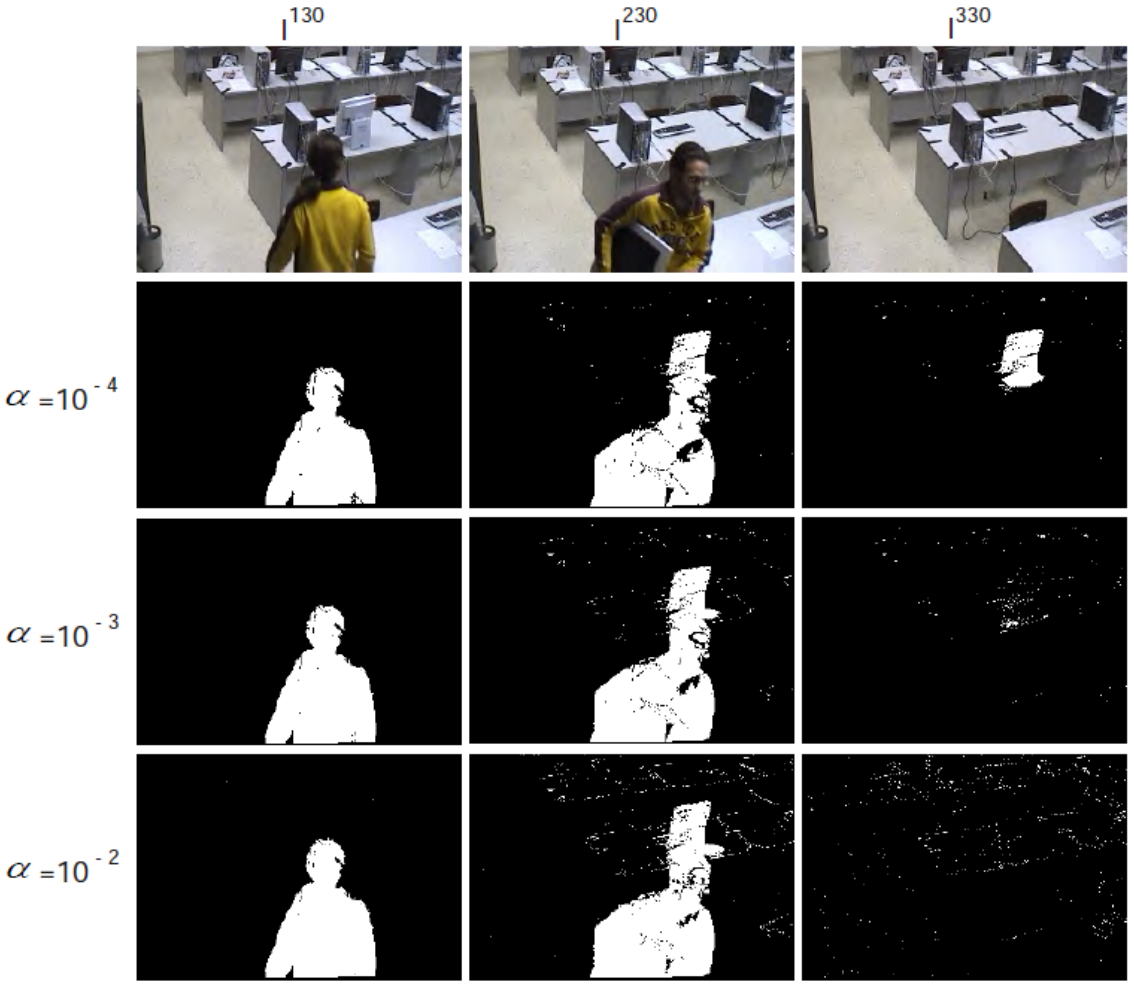


4.2.5 Gaussian Mixture Models (GMMs)

3. Update:

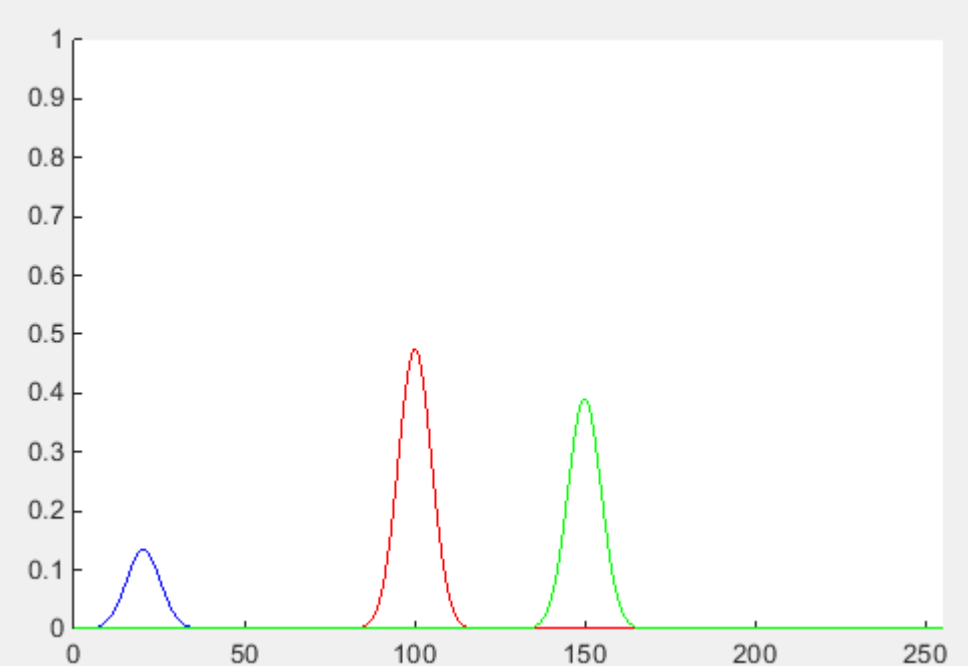
Example:

Sequence with a quick and permanent change in the background (a person is removing a display from a table).



4.2.5 Gaussian Mixture Models (GMMs)

- **Demo (manually introduced data):**



4.2.5 Gaussian Mixture Models (GMMs)

4. Classification:

a) In each Gaussian mixture, the Gaussians are ordered according to the value of the ratio:

$$r_{n,k} = \frac{w_{n,k}}{\sigma_{n,k}}$$

- These values increases both as a distribution gains more evidence (higher weight) and as the variance decreases (lower standard deviation).

- Then, the value of this ratio will be higher for those distributions that best represent the current background.

4. Classification:

b) Only the first B distributions (after they have been ordered) are finally chosen as the background model.

- B is set as the minimum number of distributions whose accumulated weights exceed a preset threshold value:

$$B = \arg\min_{u} \sum_{k=1}^{u} w_{n,k} > Th$$

- Th is the measure of the minimum portion of the data that should be accounted for the background.

4.2.5 Gaussian Mixture Models (GMMs)

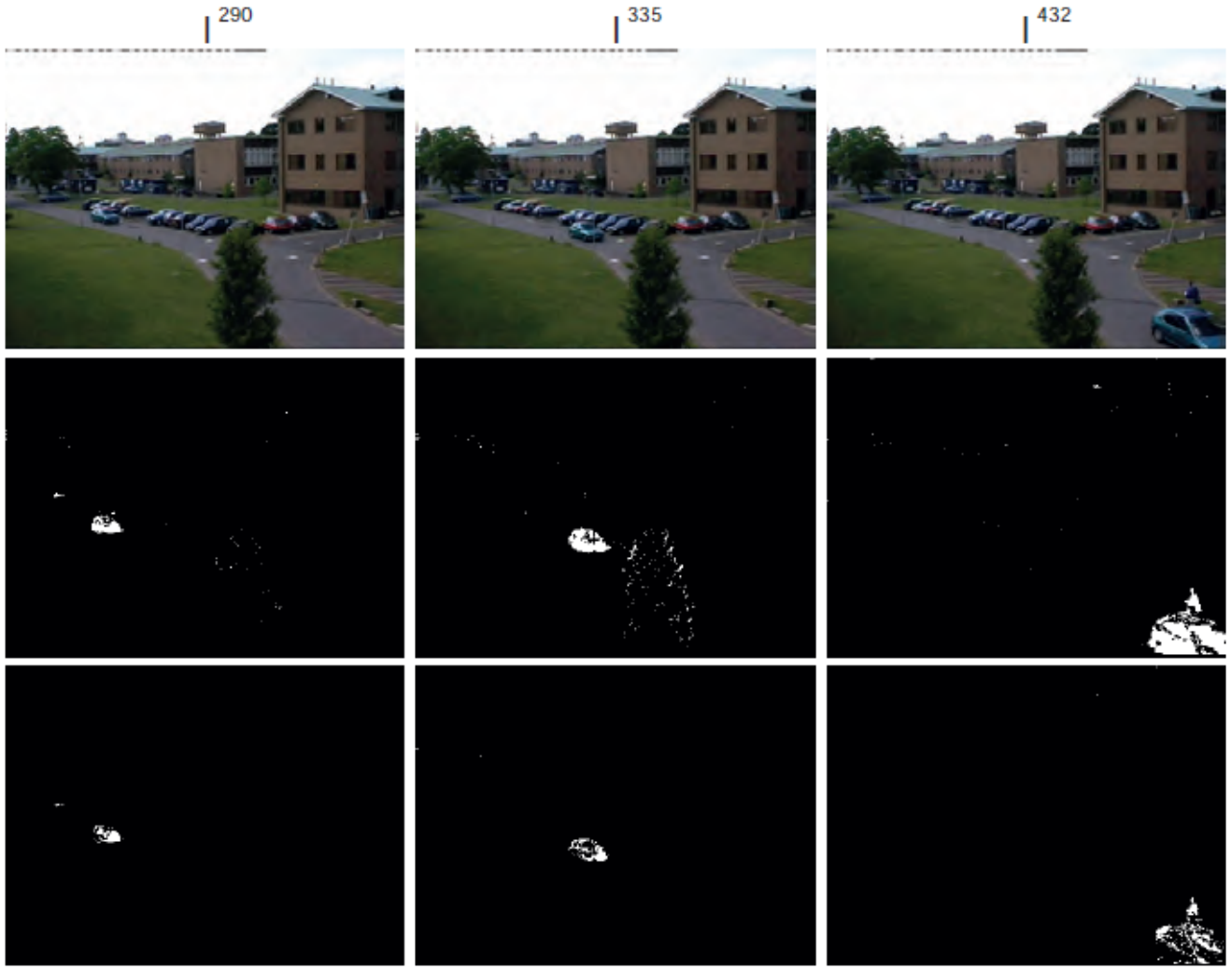
4. Classification:

- If the current value of the pixel matches (i.e., $D_{Mah} < 3$) any of the B selected distributions → The pixel is classified as part of the background.

- Otherwise, the pixel is classified as part of the foreground.

4.2.5 Gaussian Mixture Models (GMMs)

4. Classification:



4.2.5 Gaussian Mixture Models (GMMs)

- **Demo (real data):**

4.2.5 Gaussian Mixture Models (GMMs)

- **Let us remember the result when using the Single Gaussian Modeling:**

4.2.5 Gaussian Mixture Models (GMMs)

- **Results: Outside scenario with multimodal background**

4.2.5 Gaussian Mixture Models (GMMs)

- **Strengths:**
  - The background model adapts to background changes (e.g., illumination changes).
  - It works correctly in scenes with multimodal backgrounds.
  - Its memory cost is not high (nor low): Three parameters per Gaussian (mean, variance and weight).
  - Acceptable computational cost.

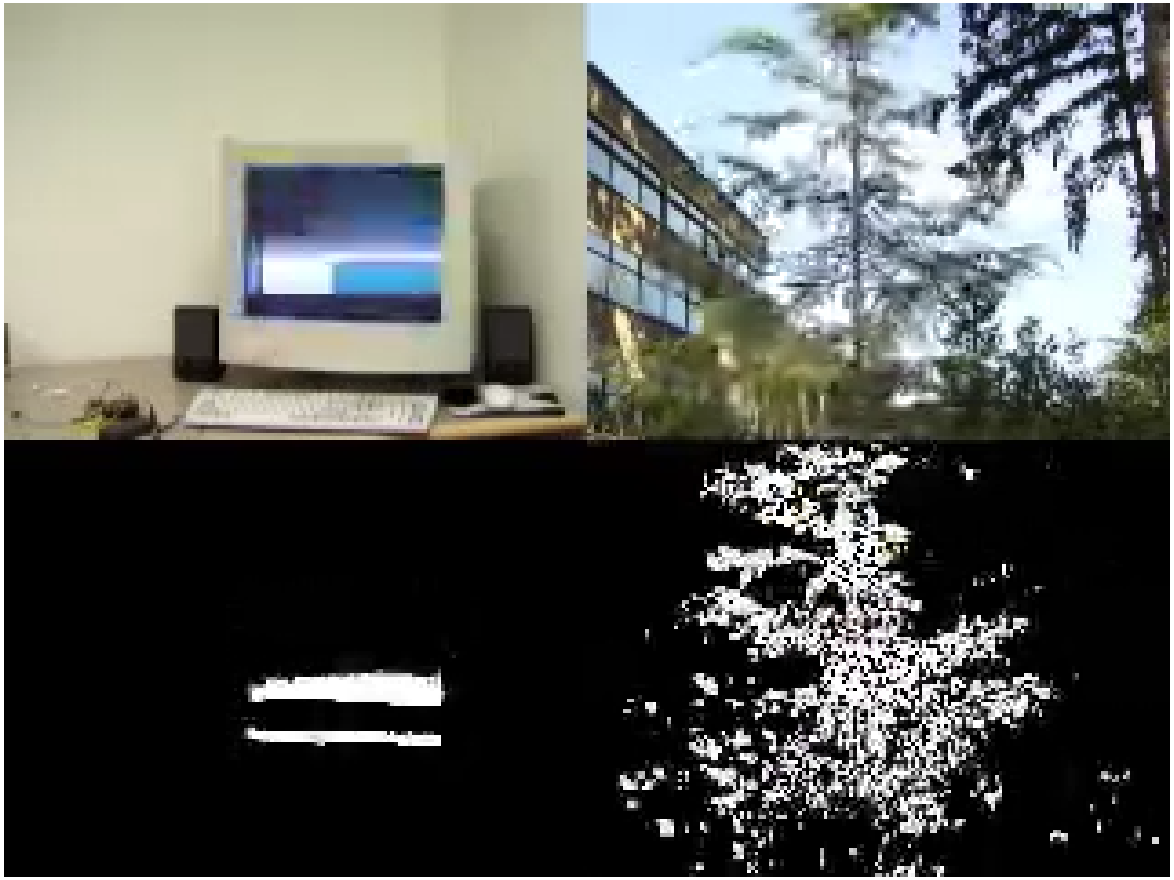
- **Drawbacks:**
  - Many parameters → Worse usability.
    - Number of distributions, N.
    - Learning rate, α.
    - Classification ratio, Th.

    } Critical

    - Others: initial standard deviation, σ_0 and initial weigh, w_0 → Not critical

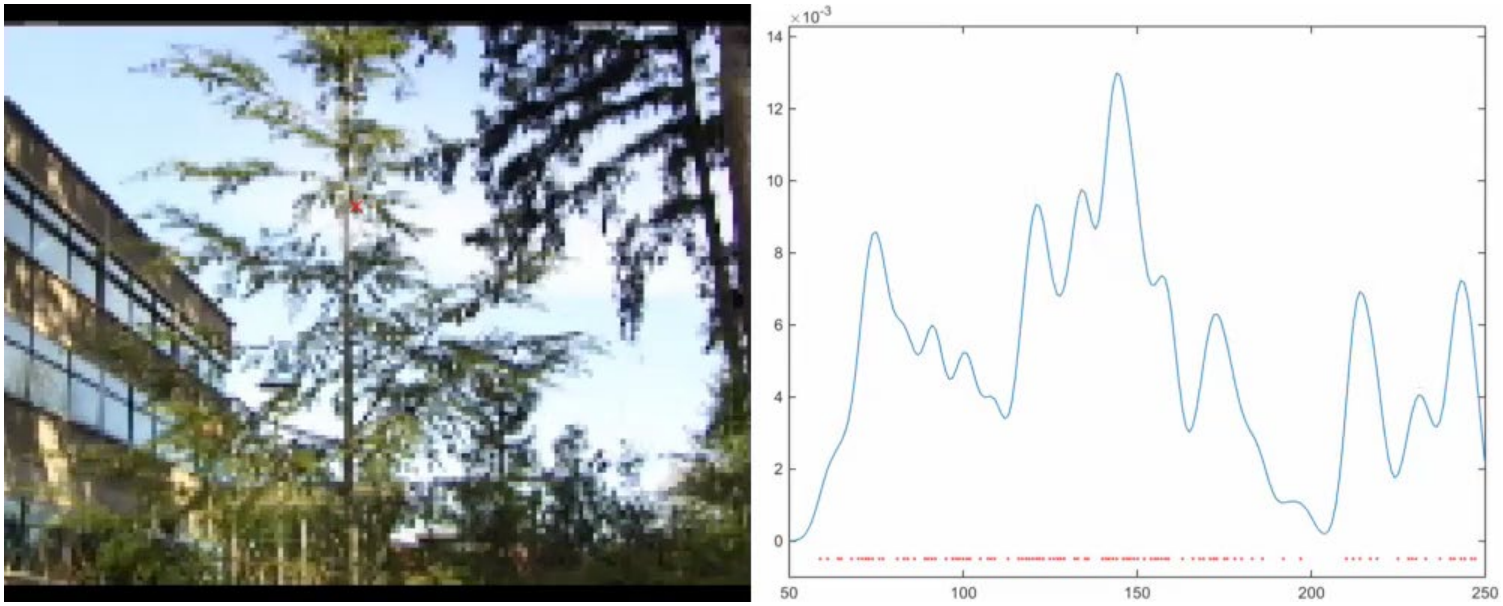
4.2.5 Gaussian Mixture Models (GMMs)

- **Additional drawback: Highly dynamic backgrounds**



4.2.5 Gaussian Mixture Models (GMMs)

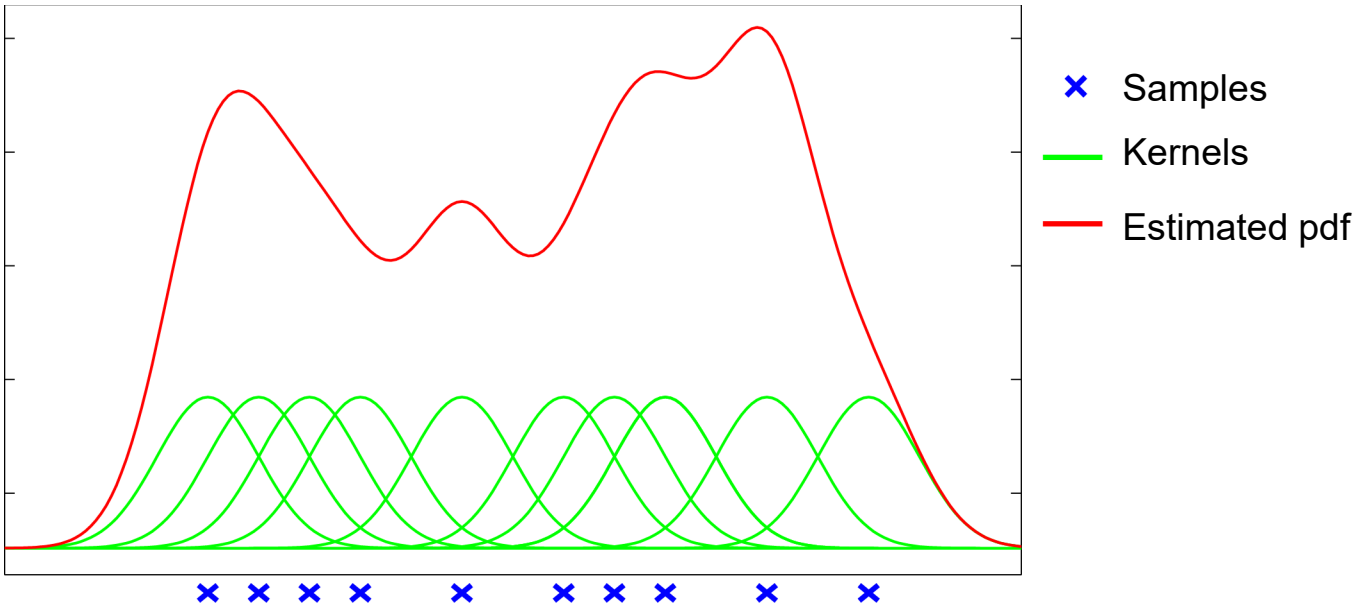
- **Additional drawback: Highly dynamic backgrounds**



  - This pdf can not be correctly modeled if a reduced number of Gaussian distributions is used.

4.2.6 KDE-based modeling

- **Kernel Density Estimation (KDE):**
  - KDE-based methods estimate the density function directly from the data without any assumptions about the underlying distribution.
    - This avoids having to choose a model and having to choose its parameters → Nonparametric modeling.



4.2.6 KDE-based modeling

- **Kernel Density Estimation (KDE):**
  - x_n → Luminance value of the pixel at coordinates (i, j), at instant n.
  - $\{x_k\}_{k=1}^{N}$ → Luminance value of the pixels at coordinates (i, j), in the N previous frames → Reference samples.
  - The probability of x_n of being part of the background is estimated as:

$$\mathrm{p}(x_n) = \frac{1}{N}\sum_{k=1}^{N} K_\sigma(x_n - x_k)$$

  - Where K_σ is a kernel function with a width (scale) σ:

$$K_\sigma(x) = \frac{1}{\sigma}K\left(\frac{x}{\sigma}\right)$$

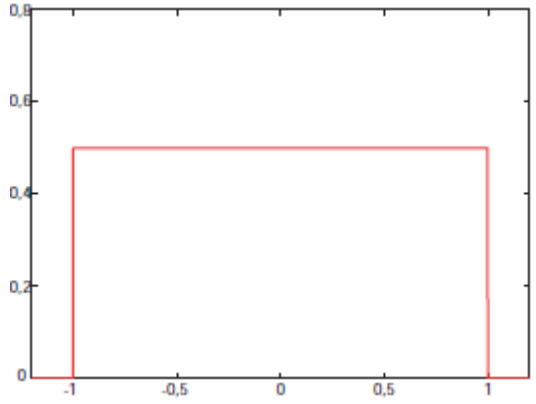
4.2.6 KDE-based modeling

- **Properties of kernels:**
  - At first instance, the kernel k can be any local function.
  - However, to facilitate its use in the estimation of pdfs, some conditions are typically imposed to it:
    - It must integrate 1 → $\int K(x)dx = 1$
    - It must be positive → $K(x) \geq 0$
    - It must be symmetric → $K(x) = K(-x)$
    - Its average must be zero → $\int xK(x)dx = 0$

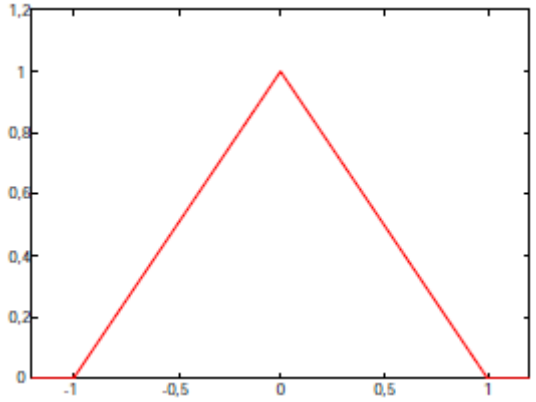
4.2.6 KDE-based modeling

- **Typical kernels:**

Uniform: $K(x) = \begin{cases} \frac{1}{2} & \text{if} \quad |x| \leq 1 \\ 0 & \text{if} \quad |x| > 1 \end{cases}$



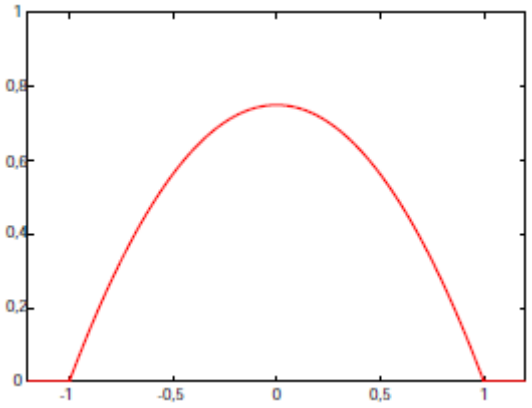
Triangular: $K(x) = \begin{cases} 1 - |x| & \text{if} \quad |x| \leq 1 \\ 0 & \text{if} \quad |x| > 1 \end{cases}$



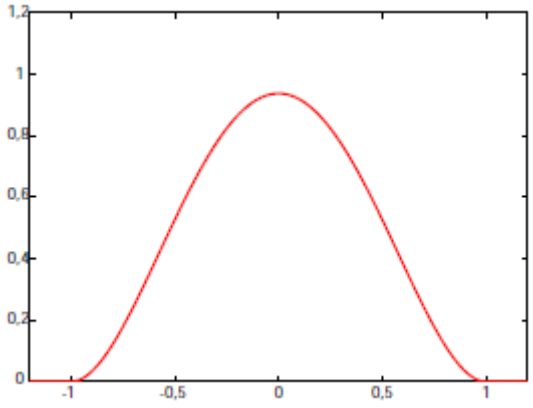
4.2.6 KDE-based modeling

- **Typical kernels:**

Epanechnikov: $K(x) = \begin{cases} \frac{3}{4}(1-x^2) & \text{if} \quad |x| \leq 1 \\ 0 & \text{if} \quad |x| > 1 \end{cases}$



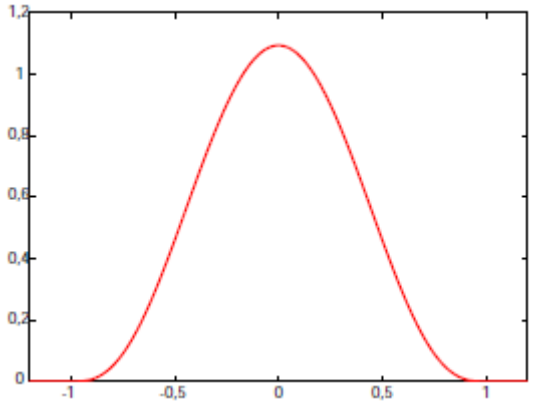
Quartic: $K(x) = \begin{cases} \frac{15}{16}(1-x^2)^2 & \text{if} \quad |x| \leq 1 \\ 0 & \text{if} \quad |x| > 1 \end{cases}$



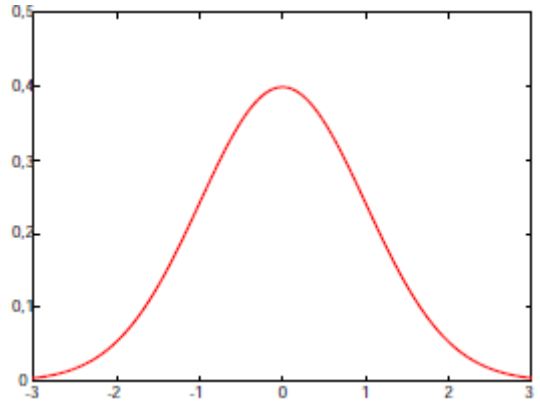
4.2.6 KDE-based modeling

- **Typical kernels:**

Triweight: $K(x) = \begin{cases} \frac{35}{32}(1-x^2)^3 & \text{if} \quad |x| \leq 1 \\ 0 & \text{if} \quad |x| > 1 \end{cases}$



Normal (Gaussian): $K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$



4.2.6 KDE-based modeling

- Typically, the Gaussian kernel is used because its continuity, differentiability, and locality properties.

- KDE using Normal kernel functions is a generalization of the Gaussian mixture model (GMM), where each reference sample is represented by a Gaussian distribution $N(0, \sigma^2)$ by itself.

$$\mathrm{p}(x_n) = \frac{1}{N} \sum_{k=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x_n - x_k)^2}{2\sigma^2} \right)$$

→ Unlike GMM-based methods, these methods **require calculating the values of the Gaussians**.

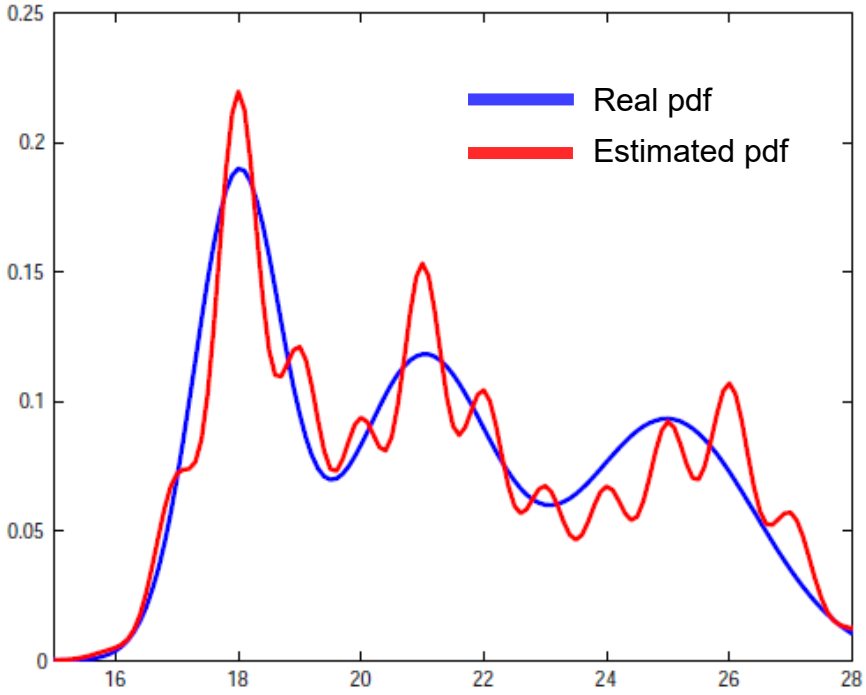
4.2.6 KDE-based modeling

- **Kernel width estimation:**
  - The selection of an adequate width for the kernels is crucial for the results.

  - <u>Narrow width:</u>

    - Adequate if the density of the reference data is high.

    - Inadequate if not → Noisy estimations.



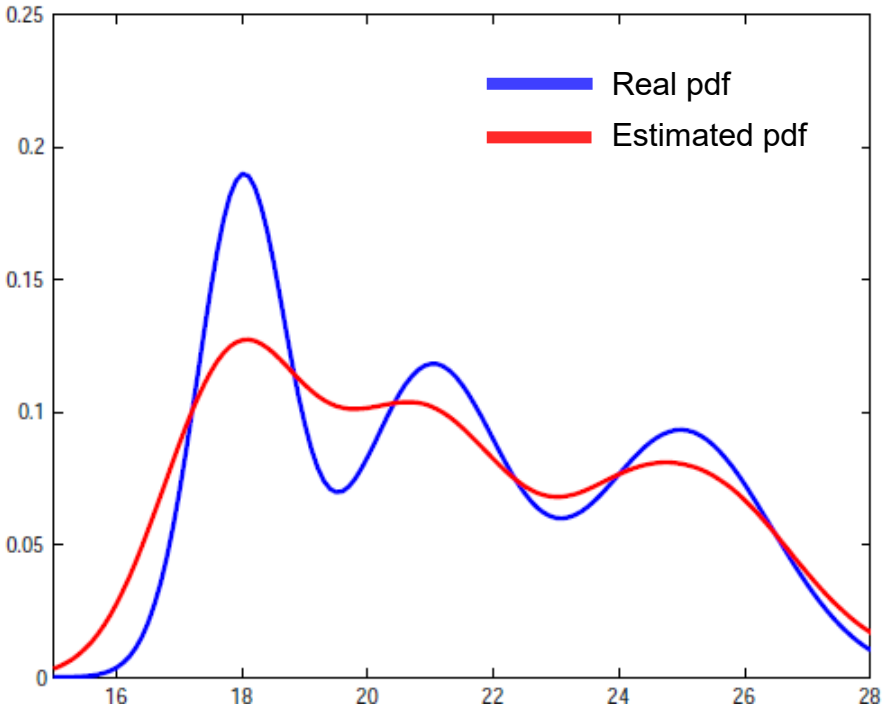
4.2.6 KDE-based modeling

- **Kernel width estimation:**
  - The selection of an adequate width for the kernels is crucial for the results.

  - <u>Broad width:</u>

    - Adequate if the density of the reference data is low.

    - Inadequate if not → Poor representation of the multimodality.



4.2.6 KDE-based modeling

- **Kernel width estimation:**
  - Since the density of the data can change along the sequences, typically, the kernel width is dynamically estimated to adapt such changes.

  - In general, there exist two formulations for estimating the widths of the kernels:
    - Sample-point estimator → A different width is assigned to each kernel.

$$\mathrm{p}(x_n) = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{\sqrt{2\pi}\sigma_k}\exp\left(-\frac{(x_n - x_k)^2}{2\sigma_k^2}\right)$$

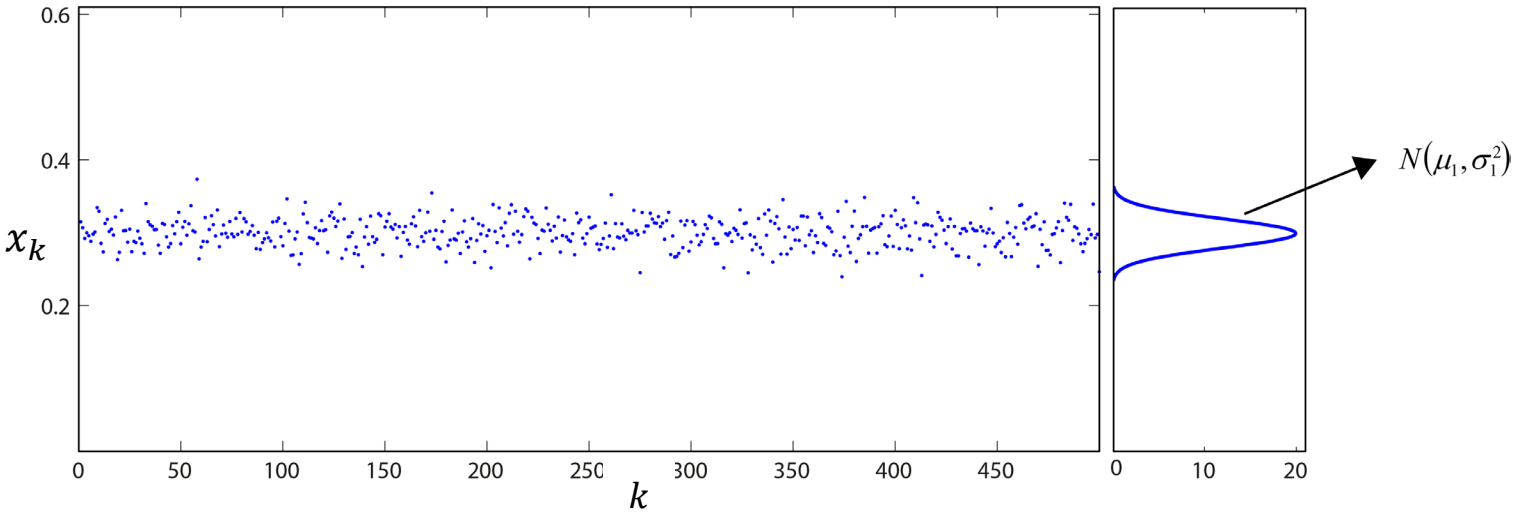
    - Balloon estimator → The same width is assigned to all the kernels.

$$\mathrm{p}(x_n) = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(x_n - x_k)^2}{2\sigma^2}\right)$$

4.2.6 KDE-based modeling

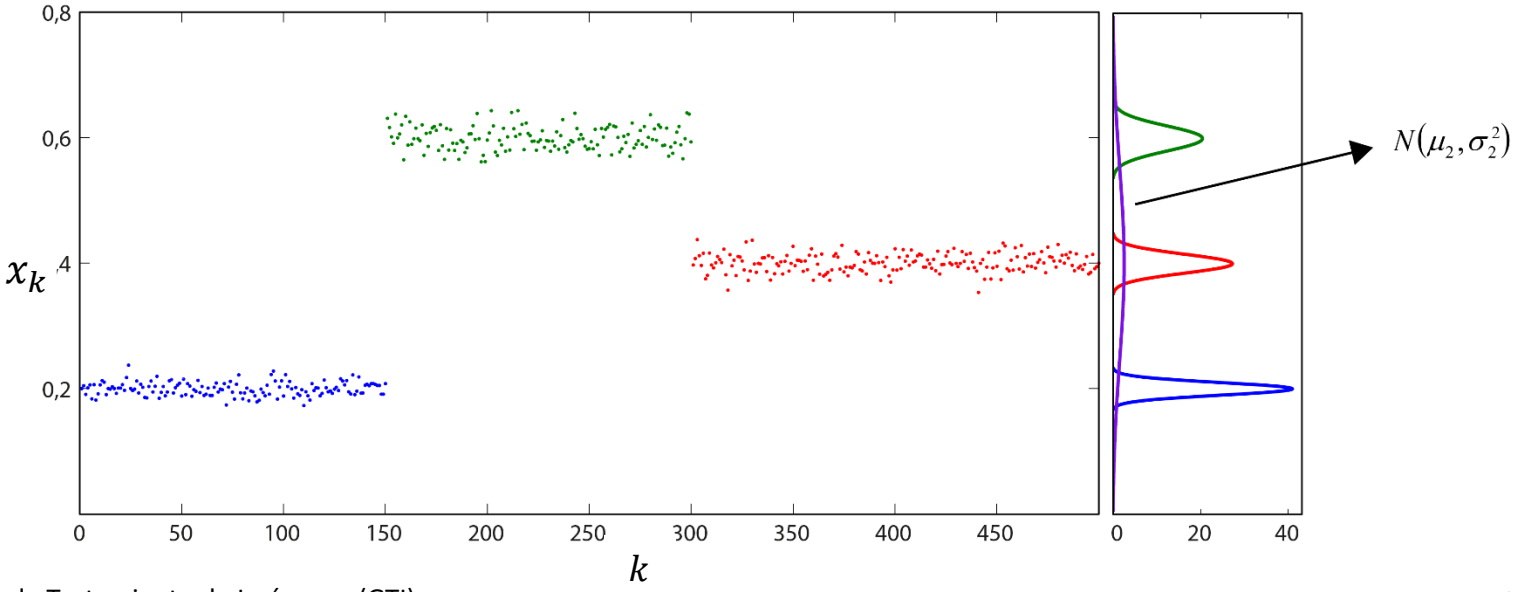
- **Example of kernel width estimation:**

  - In the simplest scenarios, where the values of a pixel belong to one mode that can be represented by a Gaussian distribution, $N(\mu_1, \sigma_1^2)$, the adequate width for the kernels is the standard deviation of such Gaussian.



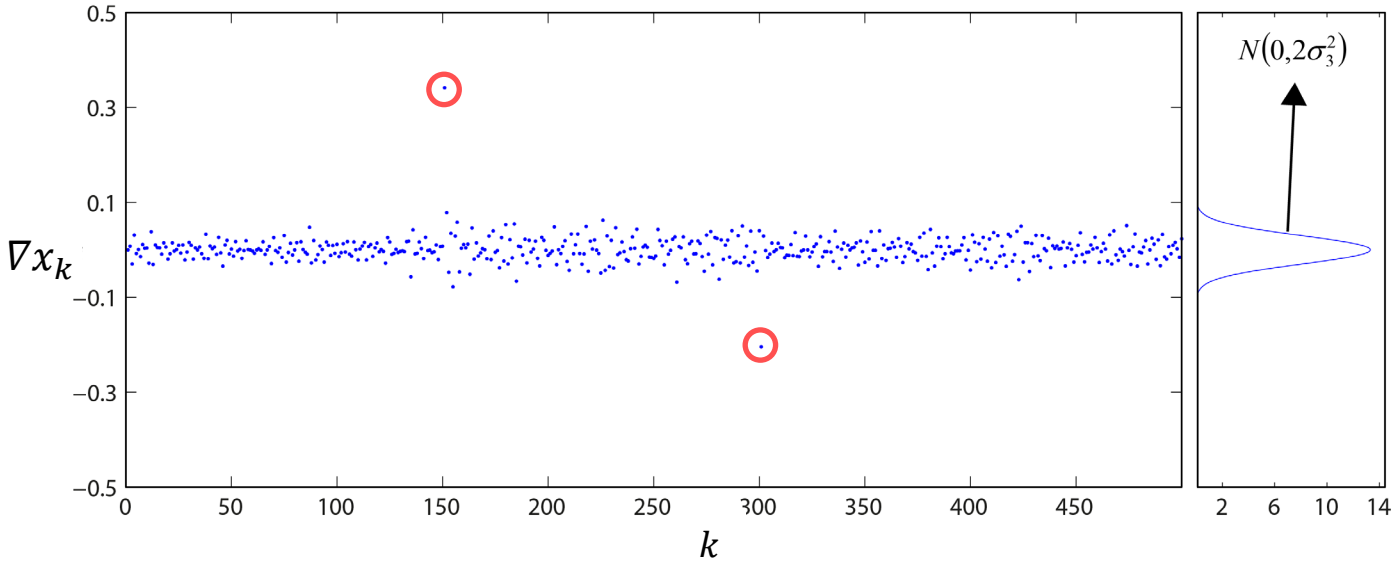
4.2.6 KDE-based modeling

- **Example of kernel width estimation:**
  - In contrast, in more complex scenarios with multimodal background, a Gaussian kernel, $N(\mu_2, \sigma_2^2)$, adapted to all the reference data is not adequate: its width is too large.
  - How can we estimate an adequate width from these data?



4.2.6 KDE-based modeling

- **Example of kernel width estimation:**
  - The width of an adequate Gaussian kernel, $N(\mu_3, \sigma_3^2)$, must fit simultaneously to all the existing modes:
    1. We construct an alternative distribution, $N(0, 2\sigma_3^2)$, from the differences of consecutive reference samples.



4.2.6 KDE-based modeling

- **Example of kernel width estimation:**
  2. To avoid the negative influence of outliers, the width of the kernel adapted to the set of differences is computed from the median, m, of the absolute values of such differences.
     - Since the Gaussian distribution is symmetric, the median of its absolute value is equivalent to the 75th percentile of such distribution:

$$\Pr(N(0,2\sigma_3^2) > m) = 0.25$$

     - From this equivalence it is possible to obtain the kernel width as:

$$\sigma_4 = \sqrt{2}\sigma_3 = \frac{m}{0.68} \longrightarrow \boxed{\sigma_3 = \frac{m}{0.68\sqrt{2}}}$$

4.2.6 KDE-based modeling

- **Results:**

GMM-based modeling

KDE-based modeling

4.2.7 Evaluation

- What dataset is adequate for assessing the quality of the results provided by the studied strategies?
  - It must contain sequences recorded in a wide variety of scenarios:
    - E.g.: Indoor/outdoor.
  - It must contain the typical challenges foreground segmentation:
    - Dynamic background.
    - Shadows and highlights cast by the moving objects.
    - Bootstrapping (moving objects from the beginning of the sequence).
    - Camouflage.
    - Illumination changes.
    - Etc.
  - The sequences must be annotated at pixel level:
    - It is required to provide objective results.

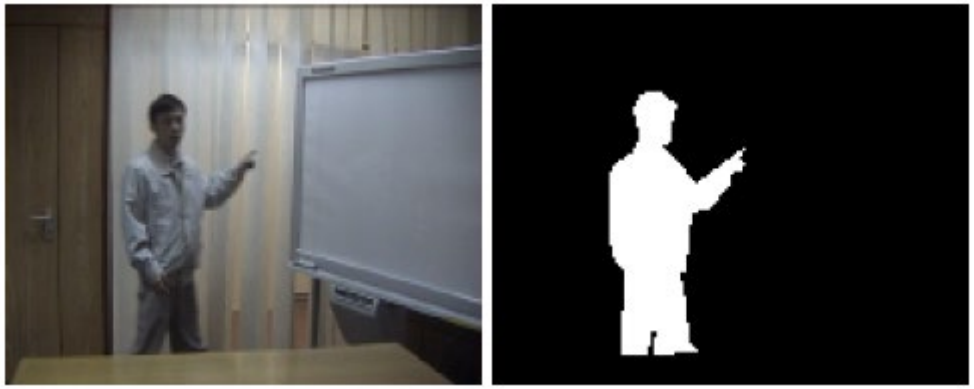
4.2.7 Evaluation

- **Wallflower:**
  - It is one of the early databases for evaluating foreground segmentation strategies.
  - It is composed by 7 sequences that address some typical challenges: camouflage, bootstrapping, object removal and illumination changes.
  - Important drawback → It only provides one ground-truth mask per sequence.

http://research.microsoft.com/en-us/um/people/jckrumm/WallFlower/TestImages.htm

4.2.7 Evaluation

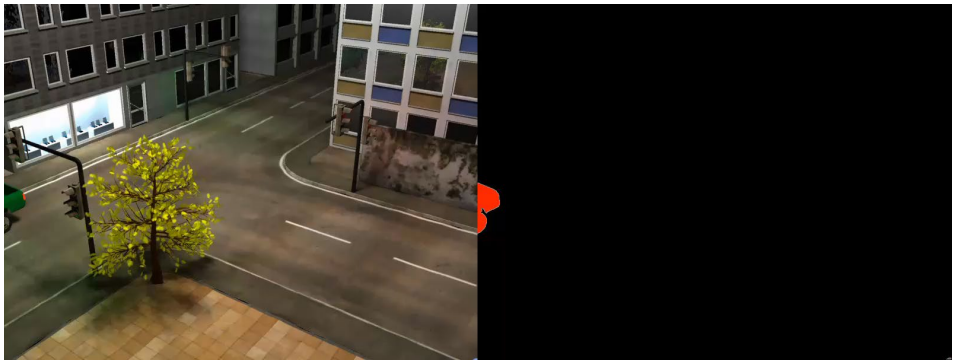
- **STAR:**
  - It is very similar to the Wallflower dataset.
  - It is composed by 9 sequences recorded in different environments: offices, shopping malls, etc.
  - It provides 20 ground-truth masks (randomly selected) per sequence.



http://perception.i2r.a-star.edu.sg/bk_model/bk_index.html

4.2.7 Evaluation

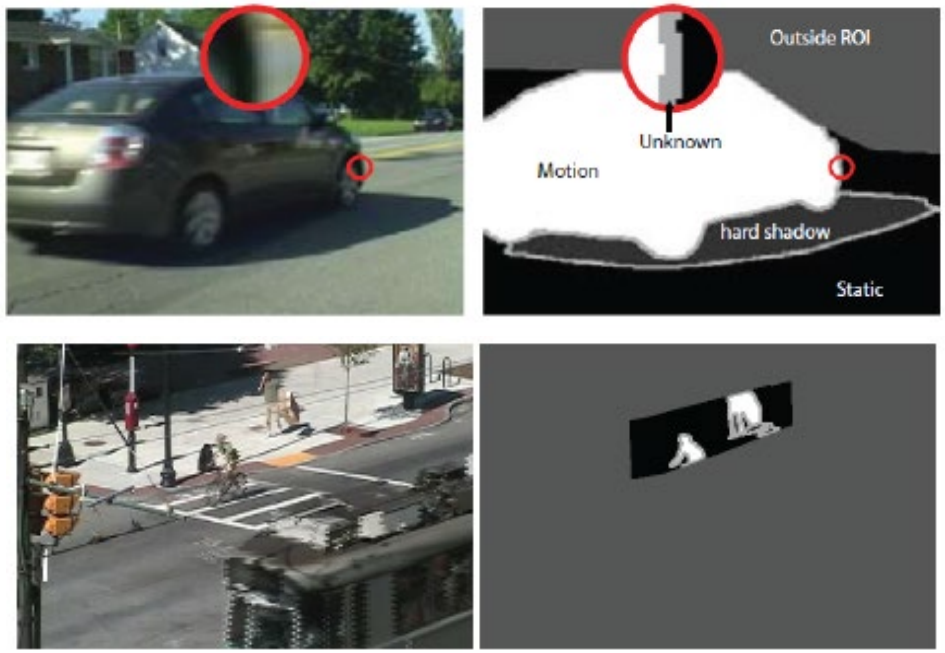
- **SABS:**
  - It is composed by synthetic sequences → The ground-truth is very complete.
  - It allows evaluating illumination changes, dynamic background, shadows, camouflage, bootstrapping and video noise.
  - Drawbacks:
    - Difficulties in the simulation of some realistic circumstances: shadows cast by the foreground objects, dynamic backgrounds, etc.
    - All the sequences show the same scenario.



http://www.vis.uni-stuttgart.de/forschung/informationsvisualisierung-und-visual-analytics/visuelle-analyse-videostroeme/sabs.html

4.2.7 Evaluation

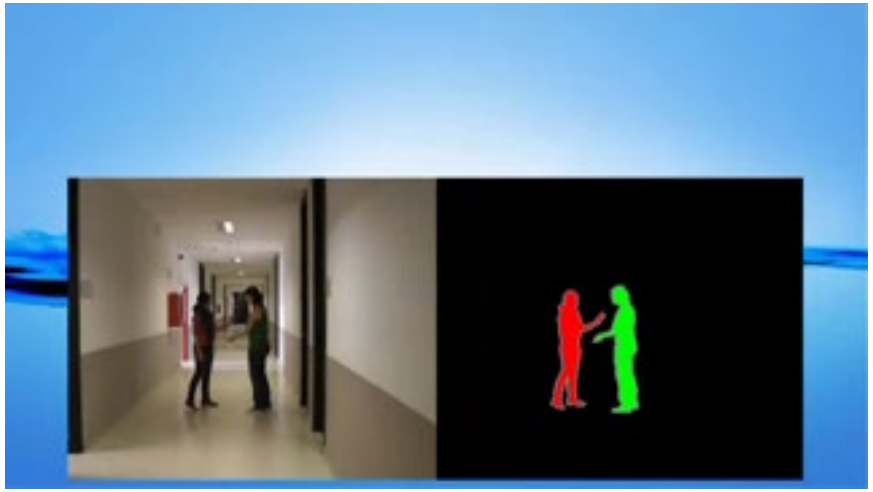
- **ChangeDetection:**
  - It is composed by 31 video sequences classified in 6 categories (each category addresses a different challenge).
  - Most images have associated a ground-truth mask with 5 possible labels per pixel:
    - Background.
    - Foreground object.
    - Unknown area (manually created around the labeled foreground objects).
    - Regions not annotated.
    - Shadows cast by foreground objects.



http://www.changedetection.net/

4.2.7 Evaluation

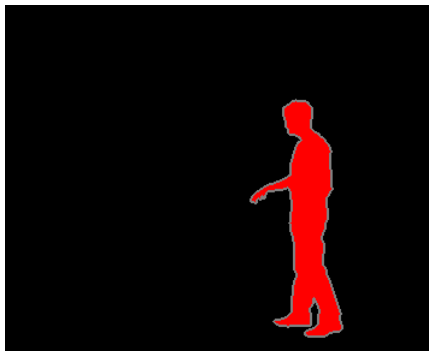
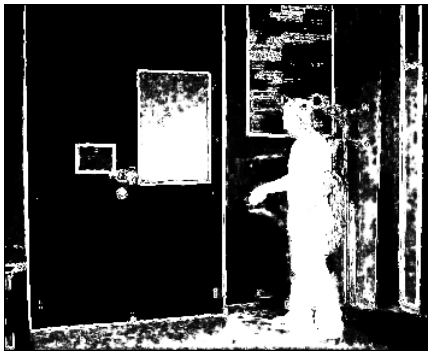
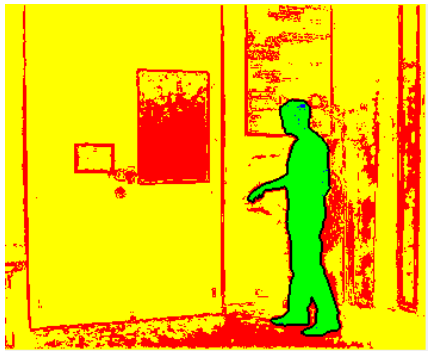
- **LASIESTA:**
  - It is composed by 48 video sequences classified into two sets: indoor and outdoor sequences.
  - Each set comprises many categories addressing different challenges.
  - It is fully annotated at both pixel and object levels.
  - It is very well structured → The challenges are well separated and, as far as possible, they are not repeated.



http://www.gti.ssr.upm.es/data/lasiesta_database.html

4.2.7 Evaluation

- **Notation:**
  - The segmentation results can be classified into 4 types:

- TP (true positives) or correct detections:
  - Foreground pixels classified as foreground.
- FP (false positives) or false detections:
  - Background pixels classified as foreground.
- FN (false negatives) or misdetections:
  - Foreground pixels classified as background.
- TN (true negatives):
  - Background pixels classified as background.

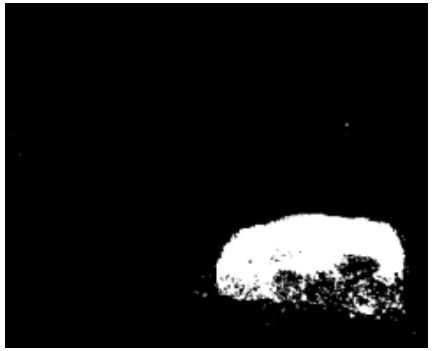
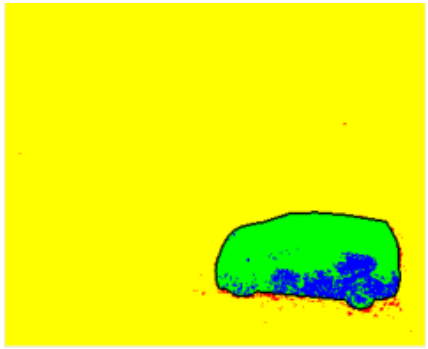
TP + *FN* = total foreground pixels

FP + *TN* = total background pixels

4.2.7 Evaluation

- **Notation:**
  - The segmentation results can be classified into 4 types:



- TP (true positives) or correct detections:
  - Foreground pixels classified as foreground.
- FP (false positives) or false detections:
  - Background pixels classified as foreground.
- FN (false negatives) or misdetections:
  - Foreground pixels classified as background.
- *TN* (true negatives):
  - Background pixels classified as background.

TP + *FN* = total foreground pixels

FP + *TN* = total background pixels

4.2.7 Evaluation

- **Objective measures of quality:**
  - The most used quality measures are those provided by the recall, precision and F-score evaluation parameters.
  - Recall (r):
    - Percentage of pixels correctly classified as foreground from the total of existing foreground pixels.
  - Precision (p):
    - Percentage of pixels correctly classified as foreground from the total amount of pixels classified as foreground.

$$r = 100\frac{TP}{TP + FN}\%$$

$$p = 100\frac{TP}{TP + FP}\%$$

4.2.7 Evaluation

- **Objective measures of quality:**

  - F-score:
    - It is the harmonic mean of the recall and precision percentages.
    - It allows to jointly evaluate both percentages.

$$F = 2\frac{r \cdot p}{r + p}\%$$

$$F = \frac{2TP}{2TP + FP + FN}\%$$

4.2.7 Evaluation

- **Example:**

TP → Green

FN → Red

FP → White

Recall → 71.26 %

Precision → 77.29 %

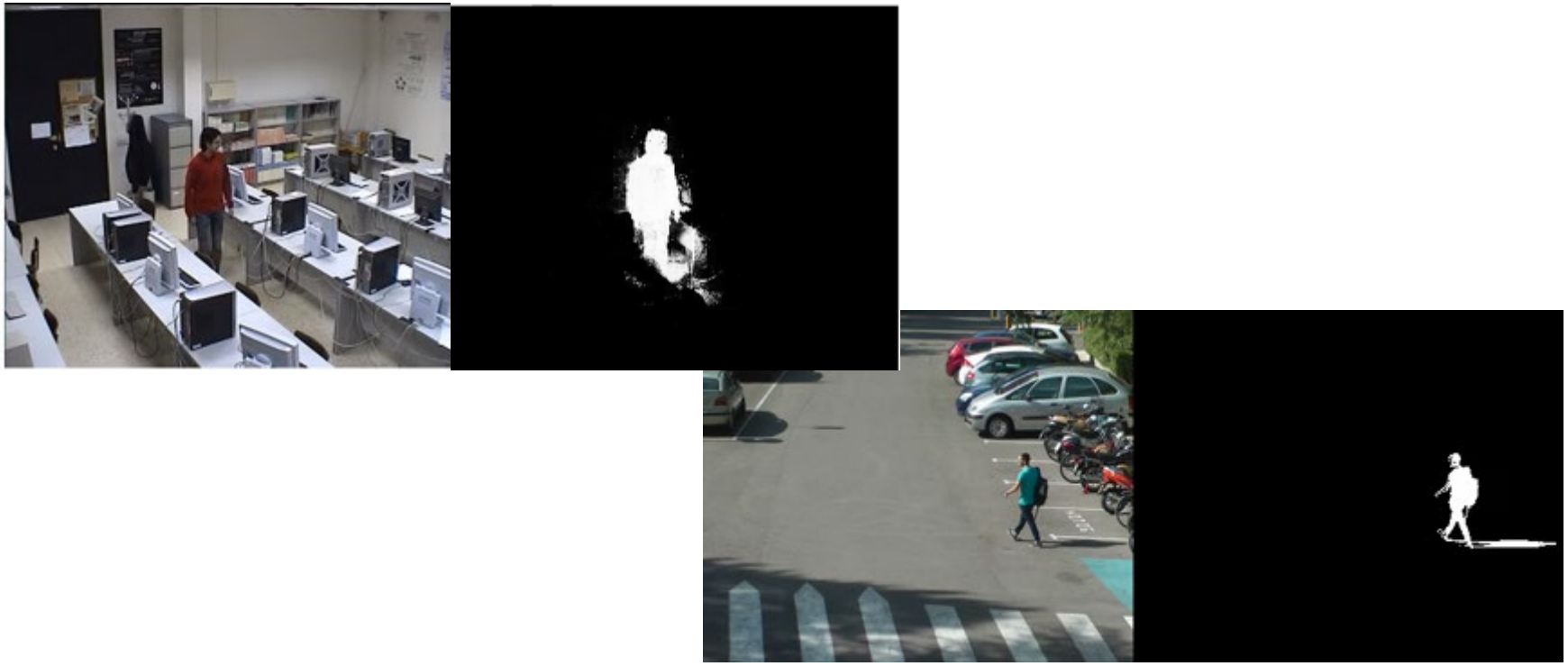
F-score → 74.15 %

4.2.7 Evaluation

- **Objective measures of quality:**
  - Other evaluation parameters (much less used):
    - False negative rate and false discovery rate.
    - Specificity and false positive rate.
    - Negative predictive value.
    - Accuracy and percentage of wrong classifications.
    - Similarity.
    - Mathews correlation coefficient.
    - D-score.
    - Structural similarity and PSNR.

4.2.8 Shadow detection

- Using background modeling strategies, shadows have a high probability to be misclassified as foreground.
- Such error is due to moving objects and shadows share similar motion characteristics.

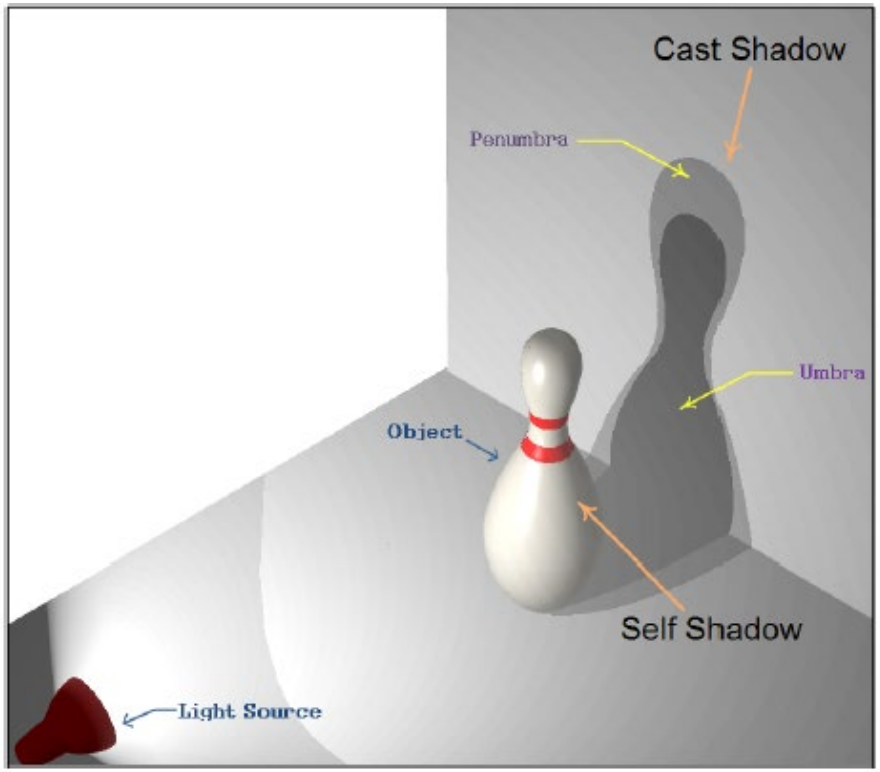


4.2.8 Shadow detection

- **Shadow:** Photometric phenomenon that occurs when an object partially or totally blocks the direct light source.

- Self shadow: It occurs in the portion of an object that is not illuminated by direct light.

- Cast shadow: Area projected on a surface in the direction of direct light.
  - Umbra → Region where the direct light source is totally blocked.
  - Penumbra → Region where the light is partially blocked.

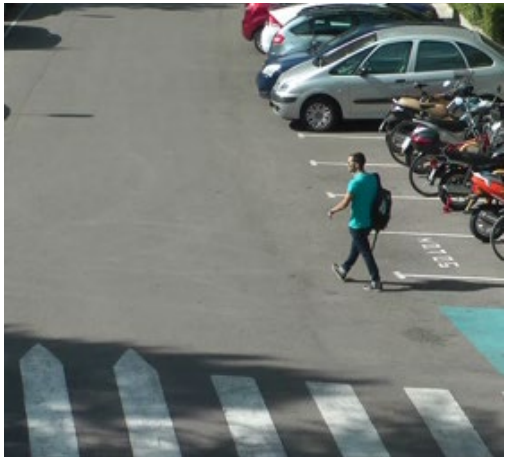


Amato, Ariel, et al. "Moving cast shadows detection methods for video surveillance applications." Wide Area Surveillance: Real-time Motion Detection Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. 23-47.

4.2.8 Shadow detection

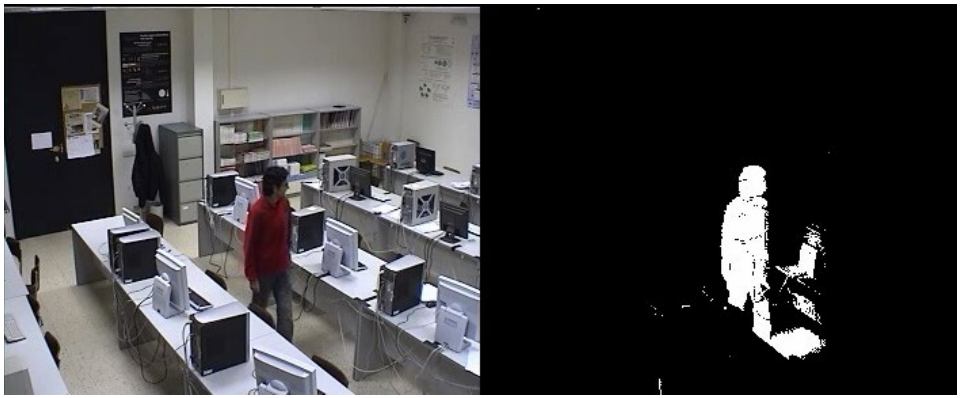
- Static shadows:
  - Due to static objects such as buildings, parked cars, etc.
  - They are not problematic for foreground detection methods → They are modeled as part of the background.
- Dynamic shadows:
  - Due to moving objects.
  - They are harmful for the foreground detection methods → Distort the objects or result in inexistent foreground objects.

https://www.flickr.com/photos/qstreetdc/6593264/

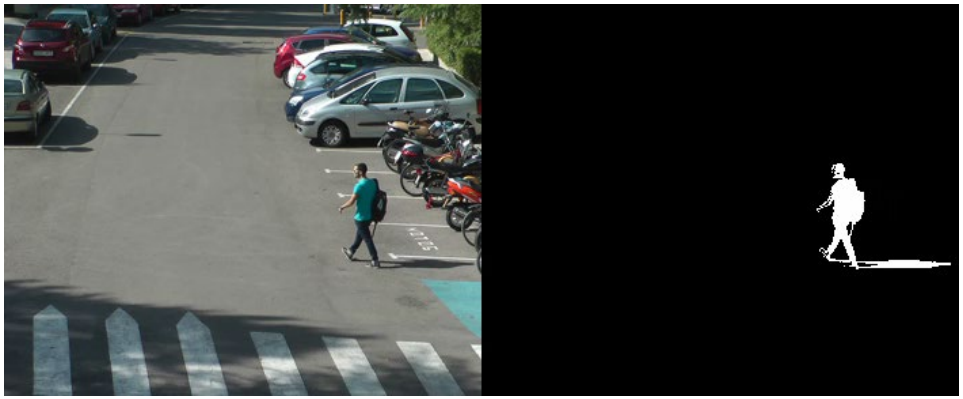


4.2.8 Shadow detection

- Shadows in indoor scenarios:



- Shadows in outdoor scenarios:



4.2.8 Shadow detection

- **Color/spectrum-based methods:**
  - They attempt to describe the color change of a shadowed pixel and find the color feature that is illumination invariant.

- **Texture-based methods:**
  - They consider that the texture of a shadowed area must be the same to the texture of the background (this does not occur with foreground objects).

- **Geometry-based methods:**
  - They are focused on the characteristics of the casted shadow area: direction, size and shape.

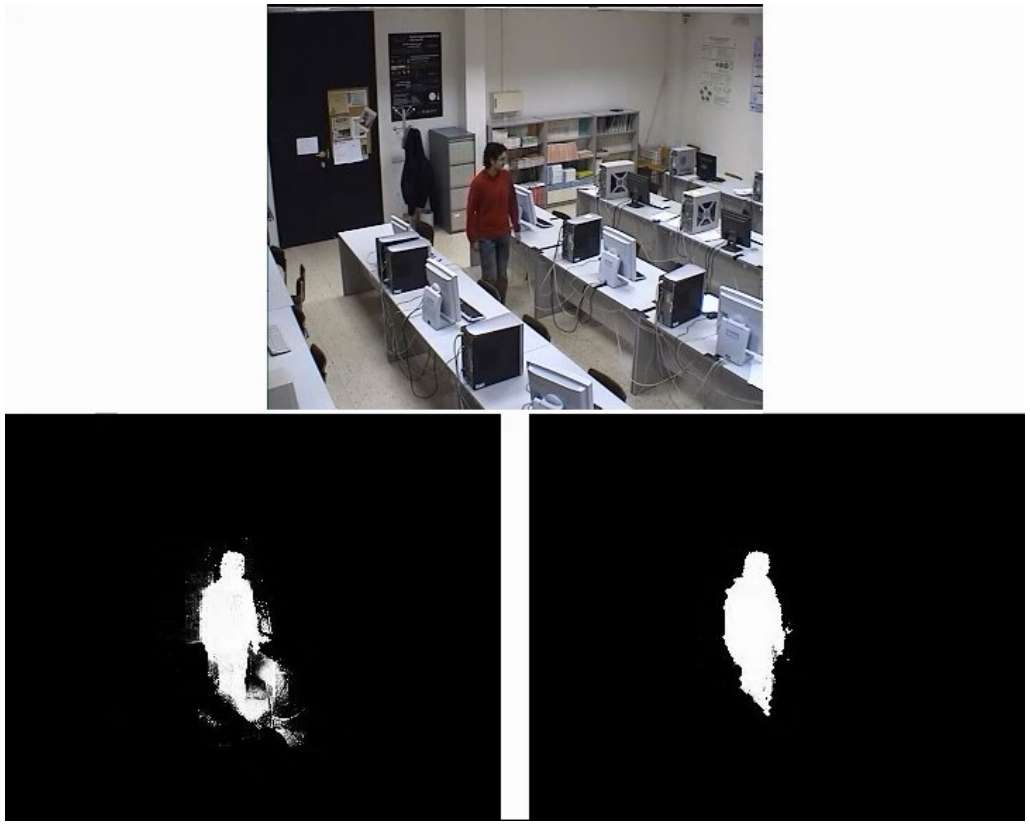
4.2.8 Shadow detection

- **Example (color-based method):**
  - Using a combination of chromaticity and gradients → $(r, g, |\nabla S|)$.

$$r = \frac{R}{R + G + B}$$

$$g = \frac{G}{R + G + B}$$

$$S = R + G + B$$



4.2.8 Shadow detection

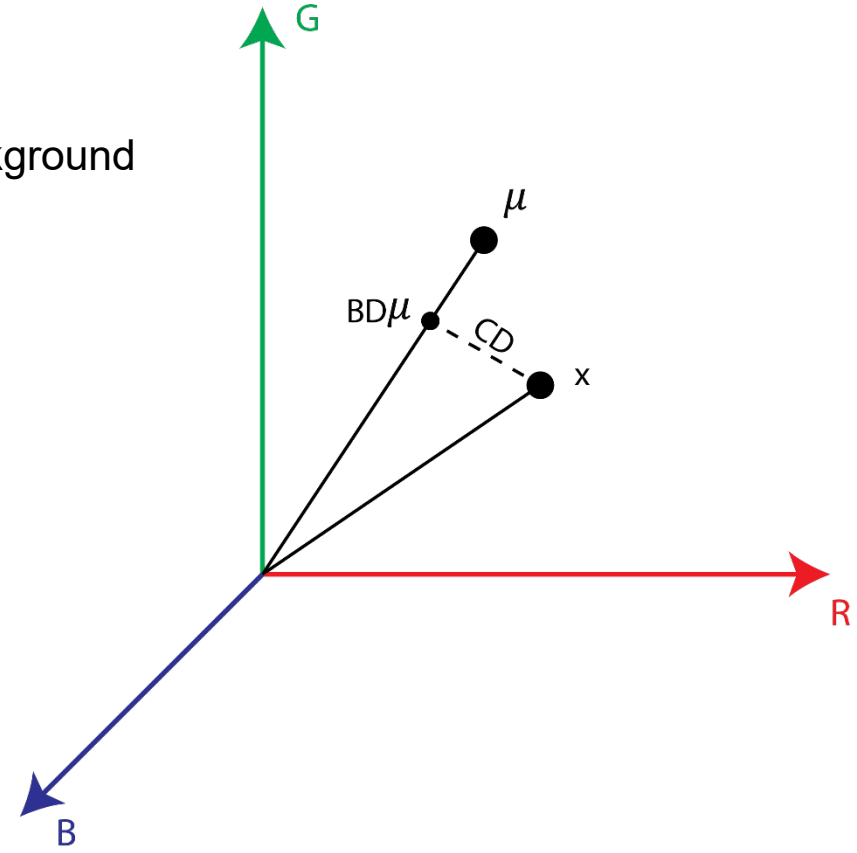
- **Example (color-based method):**
  - Hoprasert: Analysis of the distortion of the brightness (BD) and the chromaticity (CD).

$\mu = (\mu_R, \mu_G, \mu_B)$ → Point estimate of the background

$\mathbf{x} = (x_R, x_G, x_B)$→ Current value

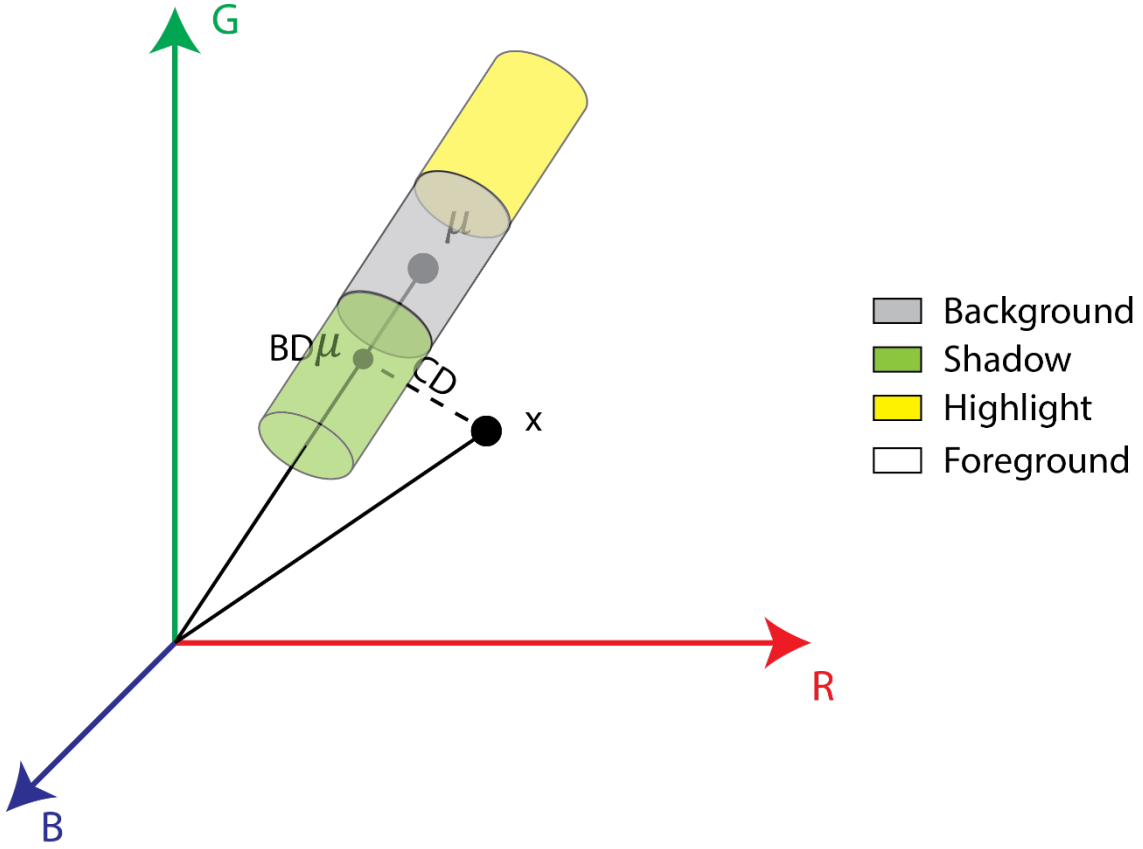
$$BD = \arg\min_{\xi}(\mathbf{x} - \xi\mu)^2$$

$$CD = \|\mathbf{x} - BD\mu\|$$



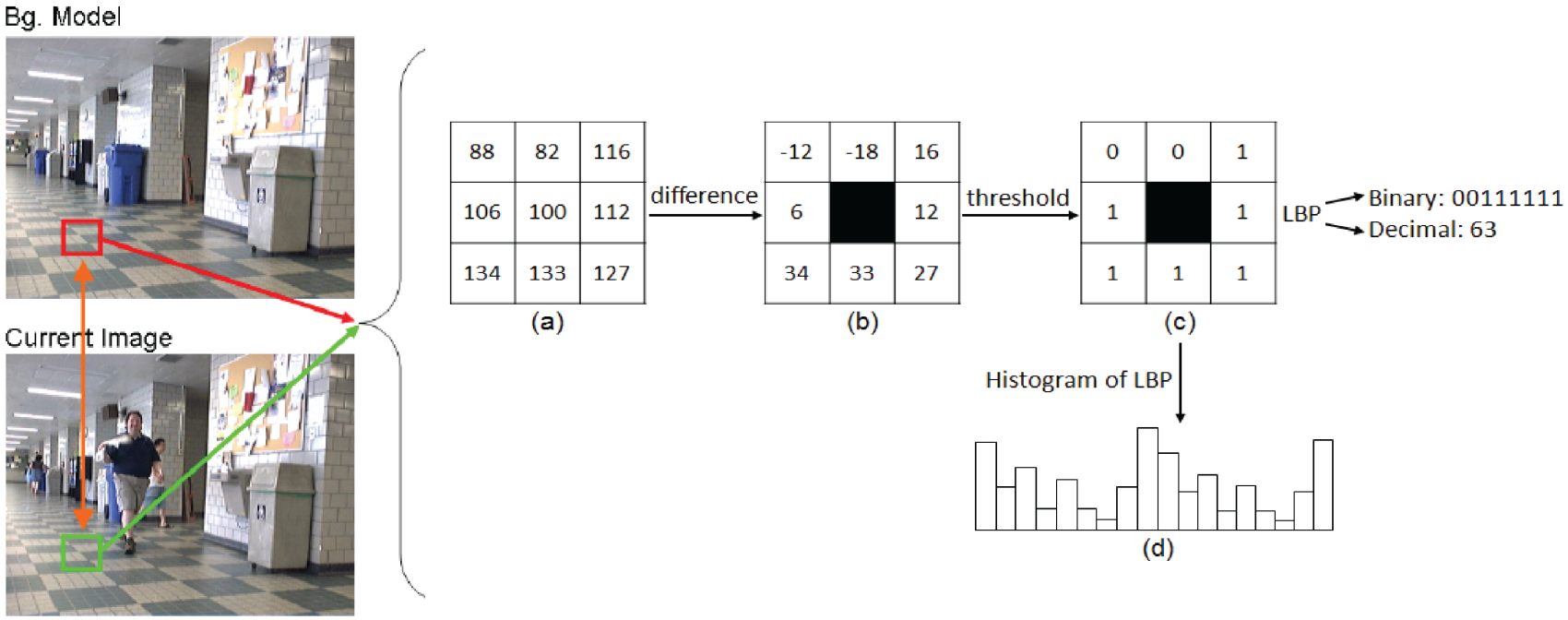
4.2.8 Shadow detection

- **Example (color-based method):**
  - Hoprasert: Analysis of the distortion of the brightness (BD) and the chromaticity (CD).



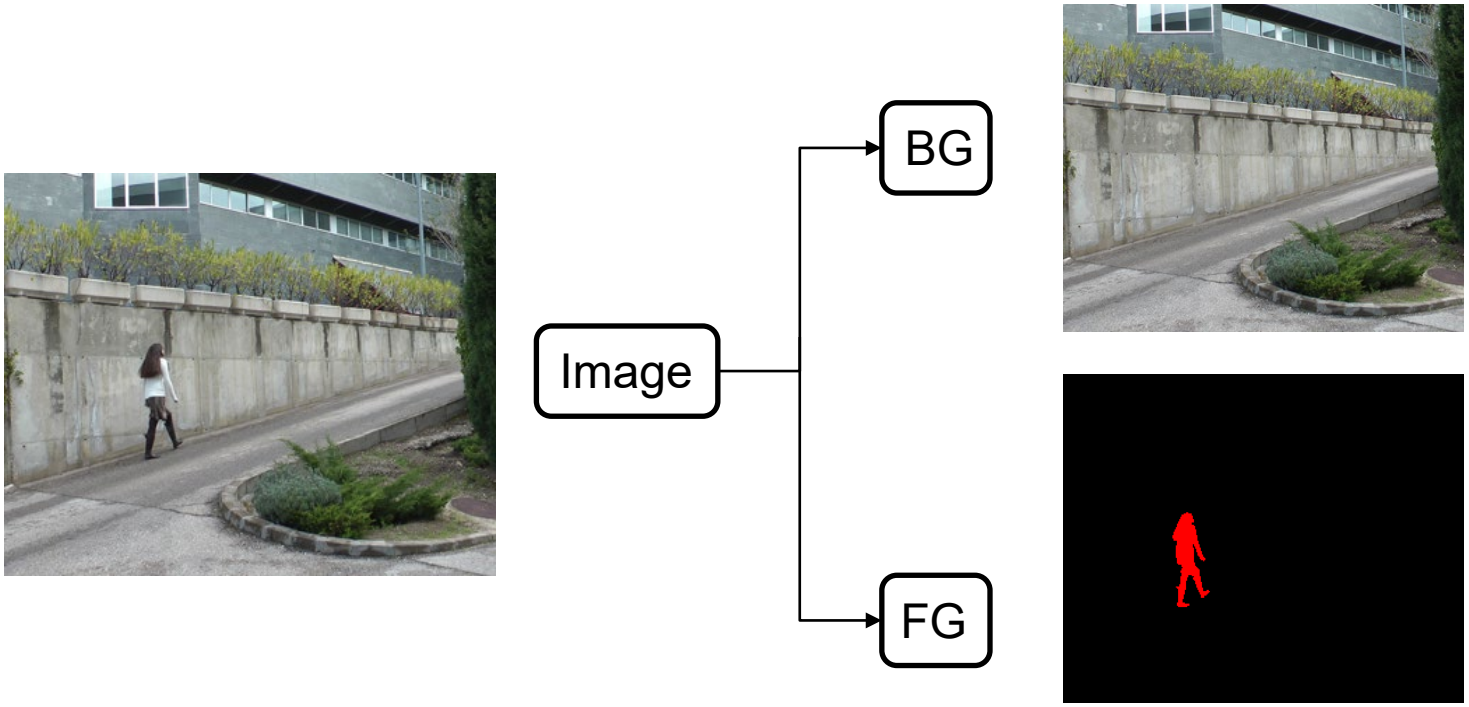
4.2.8 Shadow detection

- **Texture-based methods:**
  - These methods assume a strong correlation between a region affected by a shadow and the same region without shadow.
  - To obtain such correlation they use, for example, local binary patterns (LBP), normalized cross-correlation (NCC), Markov Random Fields (MRF), etc.



4.2.9 Stationary foreground detection

- Foreground detection methods focus on separating the background of the scene (i.e., permanent objects) from the rest of objects (i.e., all the non-permanent objects).
- To do this, the values of the pixels are analyzed through the images looking for representative changes denoting the presence of foreground objects.



4.2.9 Stationary foreground detection

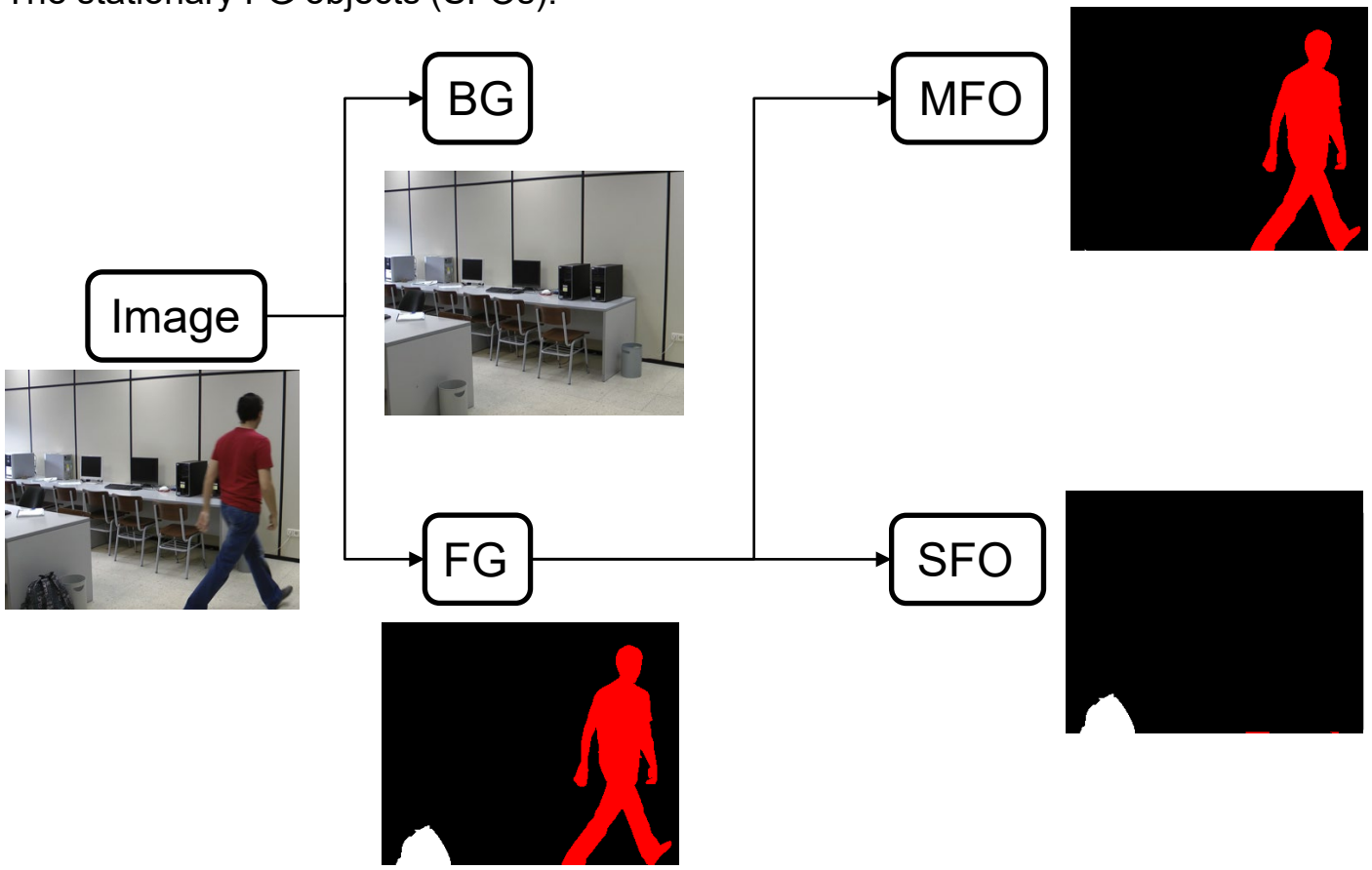
- **Stationary foreground objects (SFOs):**
  - They are non-background objects (i.e., foreground) that are not in motion.
  - Examples:
    - A suitcase that is abandoned by someone.
    - A car that stops in a traffic light.

4.2.9 Stationary foreground detection

- **Stationary foreground objects (SFOs):**
  - All these SFOs are not part of the original background of the scene, so they must be classified as foreground objects.
  - However, since SFOs maintain constant the pixel values through many consecutive frames, typical background subtraction techniques fail in detecting them.

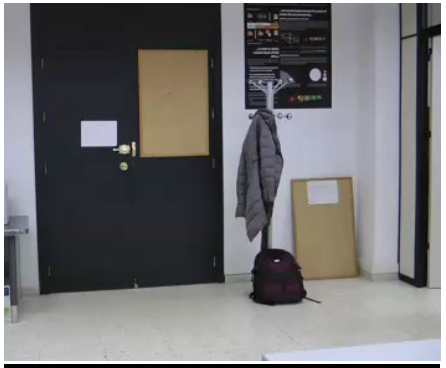
4.2.9 Stationary foreground detection

- The purpose now is to detect both:
  - The FG objects that are in motion → Moving foreground objects (MFOs).
  - The stationary FG objects (SFOs).



4.2.9 Stationary foreground detection

- **Types of SFOs:**
  - Long duration and totally static:
    - E.g.: Abandoned objects.
  - Short duration and totally static:
    - E.g.: A car that stops for a while.
  - Partially static (typically short duration):
    - E.g.: People that stop.

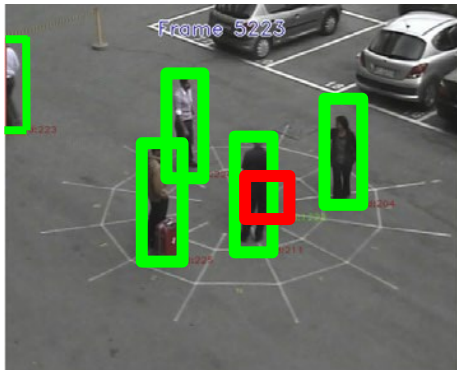


4.2.9 Stationary foreground detection

- **Detection level:**

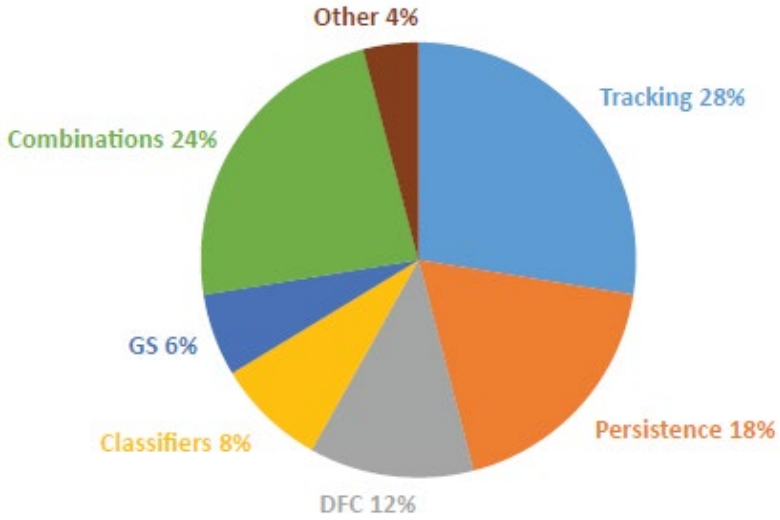
  - Pixel level:
    - They allow detecting partially stationary foreground objects (PSFOs).
    - The results are more difficult to interpret.

  - Object level:
    - Better understanding of the scene.
    - They are unable of detecting PSFOs.



4.2.9 Stationary foreground detection

- **Common approaches for detecting SFOs:**
  - Tracking-based methods.
  - Persistence analysis.
  - Dual Foreground Comparisons (DFC).
  - Classifiers.
  - Gaussian Stability (GS) analysis.
  - Combinations.
  - Other.

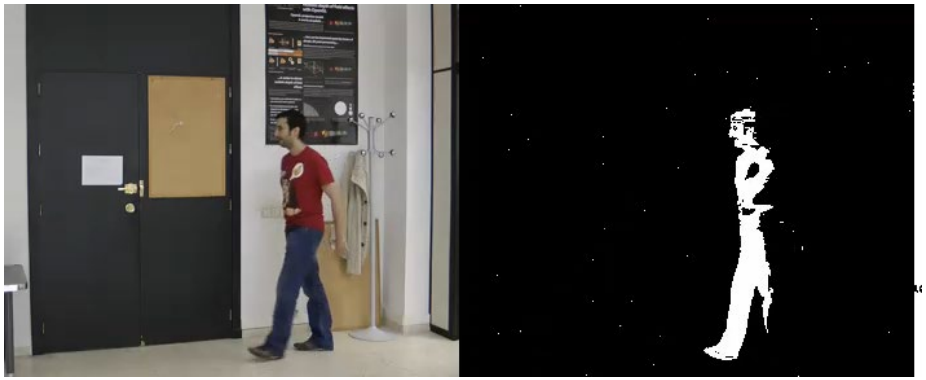


4.2.9 Stationary foreground detection

- **Simple solution → Selective background update:**
  - There exists two main mechanisms for updating the background model: blind update and selective update.
  - <u>Blind update:</u>
    - The model is updated with all the current data (regardless of whether they are foreground or background).

    The changes in the background are absorbed by the modeling.

    The SFOs are also absorbed by the model.

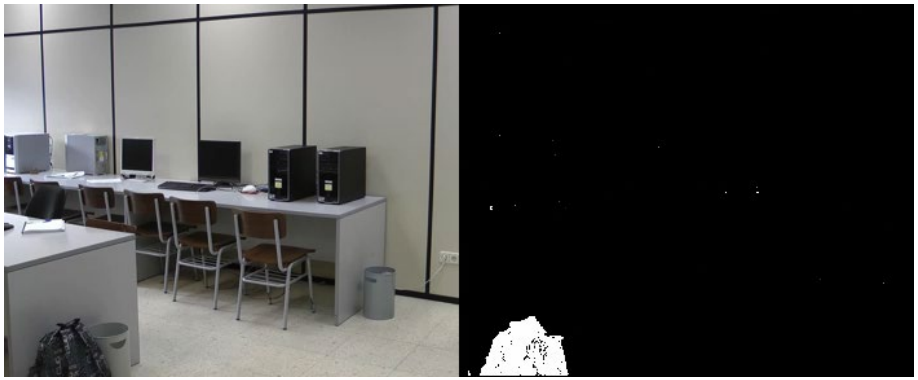


4.2.9 Stationary foreground detection

- **Simple solution → Selective background update:**
  - Selective update:
    - The model is updated with only the data previously classified as background.

✓ The SFOs are not absorbed by the model.

✗ The changes in the background result in persistent false detections.



4.2.9 Stationary foreground detection

- **Persistence analysis:**
  - These methods perform an analysis of the persistence of the pixels that have been previously classified as FG → Pixel-level analysis.
    - That is, a pixel is classified as SFO if it is classified as FG for a predetermined time or along several frames.

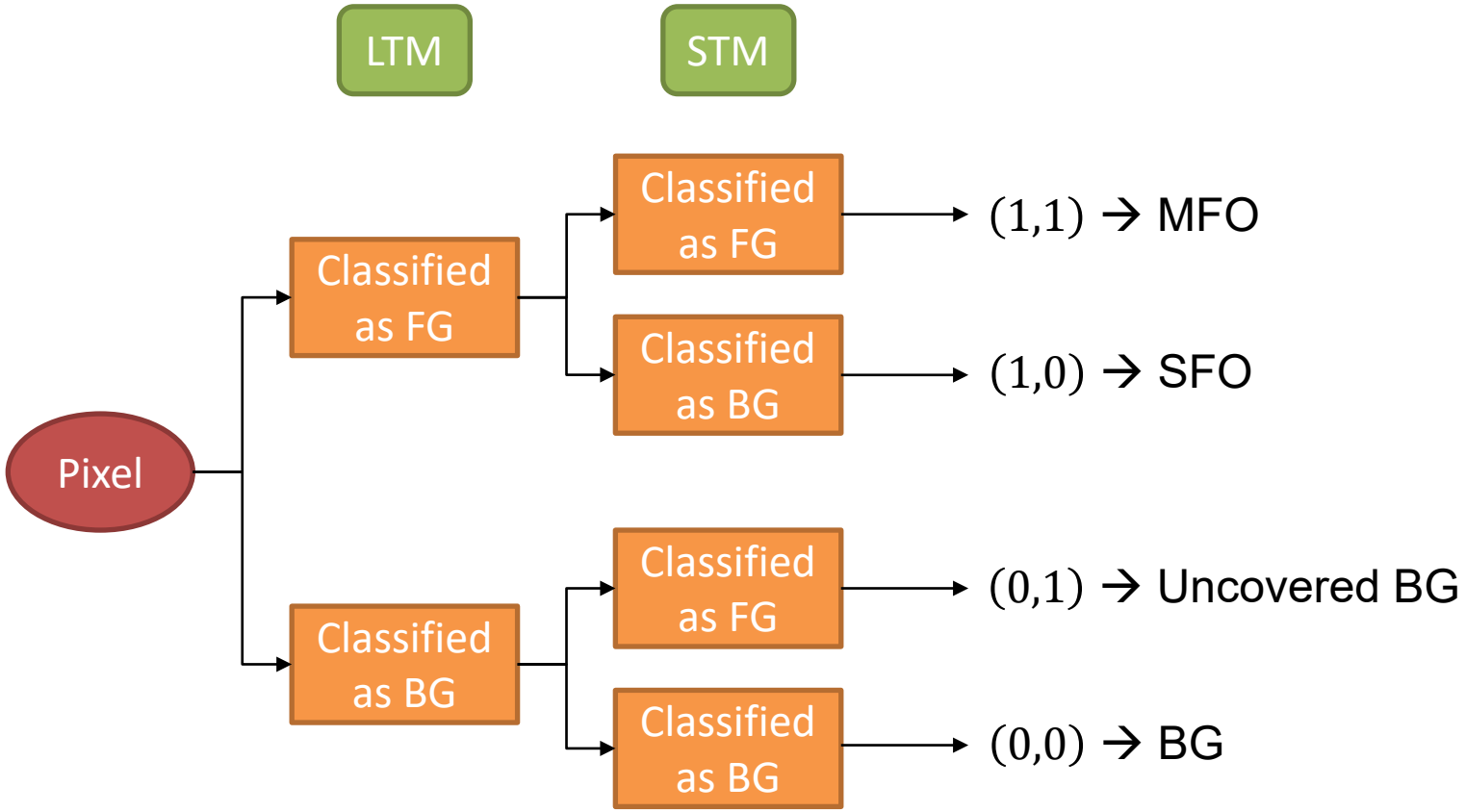
- They are simple and computationally efficient.
- They are not robust in complex scenarios with multimodal background or illumination changes.
- They do not deal with occluded SFOs.
- They are able of detecting PSFOs.

4.2.9 Stationary foreground detection

- **Dual Foreground Comparison (DFC):**
  - They detect the SFOs by comparing two FG masks obtained from two background modeling methods with different learning rates.
  - Model with the highest learning rate:
    - It is commonly named **short-term model (STM)**.
    - It is configured to adapt rapidly to the changes in the scene.
    - It only detects the FG objects that are in motion.
  - Model with the lowest learning rate:
    - It is commonly named **long-term model (LTM)**.
    - It is configured to be resistant against the changes in the scene.
    - It detects both the FG objects that are in motion and the SFOs.

4.2.9 Stationary foreground detection

- **Dual Foreground Comparison (DFC):**

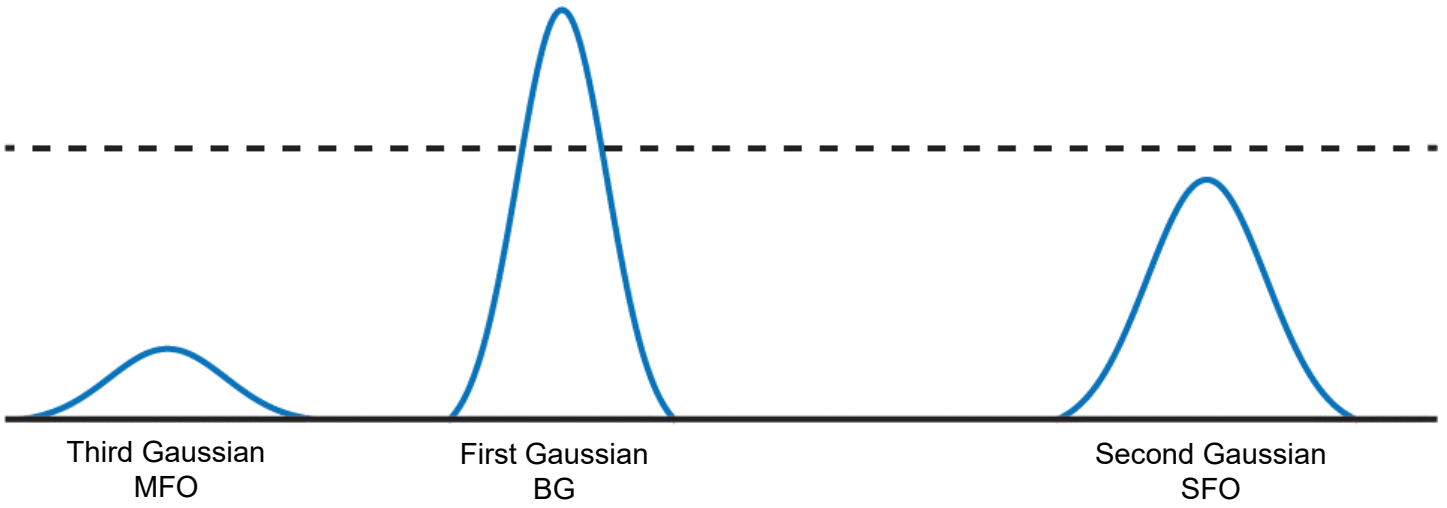


4.2.9 Stationary foreground detection

- **Dual Foreground Comparison (DFC):**
  - Successful results in complex scenarios (e.g., with dynamic BG).
  - Detection of PSFOs.
  - Complex configuration → Too many parameters.
  - Misdetection of long-term SFOs.
  - Misdetection of occluded SFOs.

4.2.9 Stationary foreground detection

- **Gaussian Stability (GS):**
  - Determine that a pixel is part of a SFO by analyzing the stability of the Gaussians in a GMM associated to such pixel:
    - When a moving object appears in a pixel, a new Gaussian is created in its GMM.
    - This Gaussian represents the new value of the pixel.
    - If the object stops moving the new Gaussian will begin to gain importance in the mixture model.



4.2.9 Stationary foreground detection

- **Gaussian Stability (GS):**

  - They are computationally efficient and easy to implement.
  - The depend on few parameters → High usability.
  - They allow dealing with PSFOs.
  - They do not detect occluded SFOs.
  - If the FG object remains static for too long, the Gaussian will become more important than the Gaussian modeling the BG → The long-term SFOs will be misdetected.

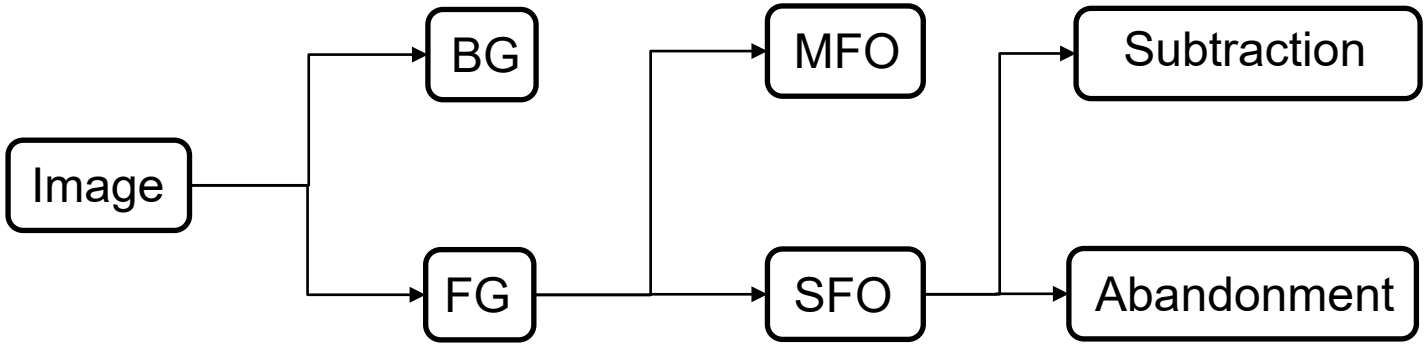
4.2.9 Stationary foreground detection

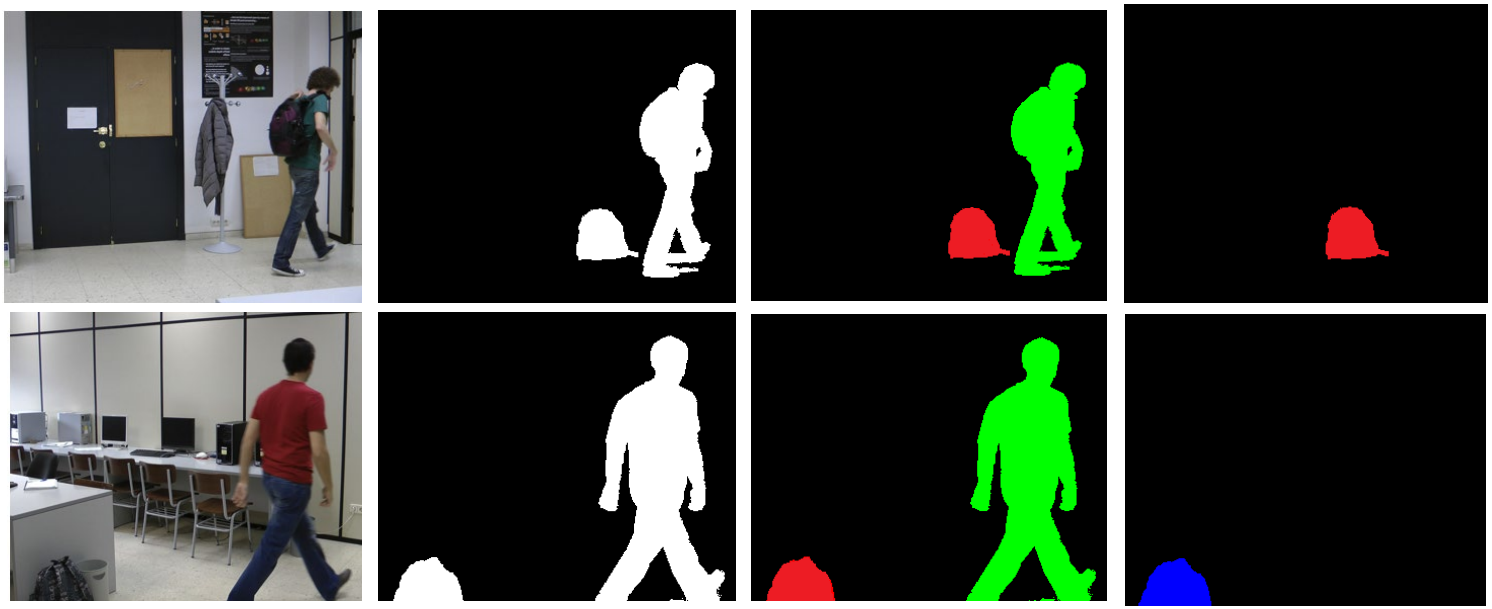
- **Removed objects:**
  - The previously studied techniques establish the presence of a SFO when a persistent change in the BG occurs.
  - Initially, the purpose of these techniques is the detection of objects that have stopped moving (e.g., abandoned objects).
  - However, they also detect a SFO when a BG object is removed:
    - A removed object also result in a persistent BG change.

Mechanisms for discriminating between both situations are required.

4.2.9 Stationary foreground detection

- **Removed objects:**



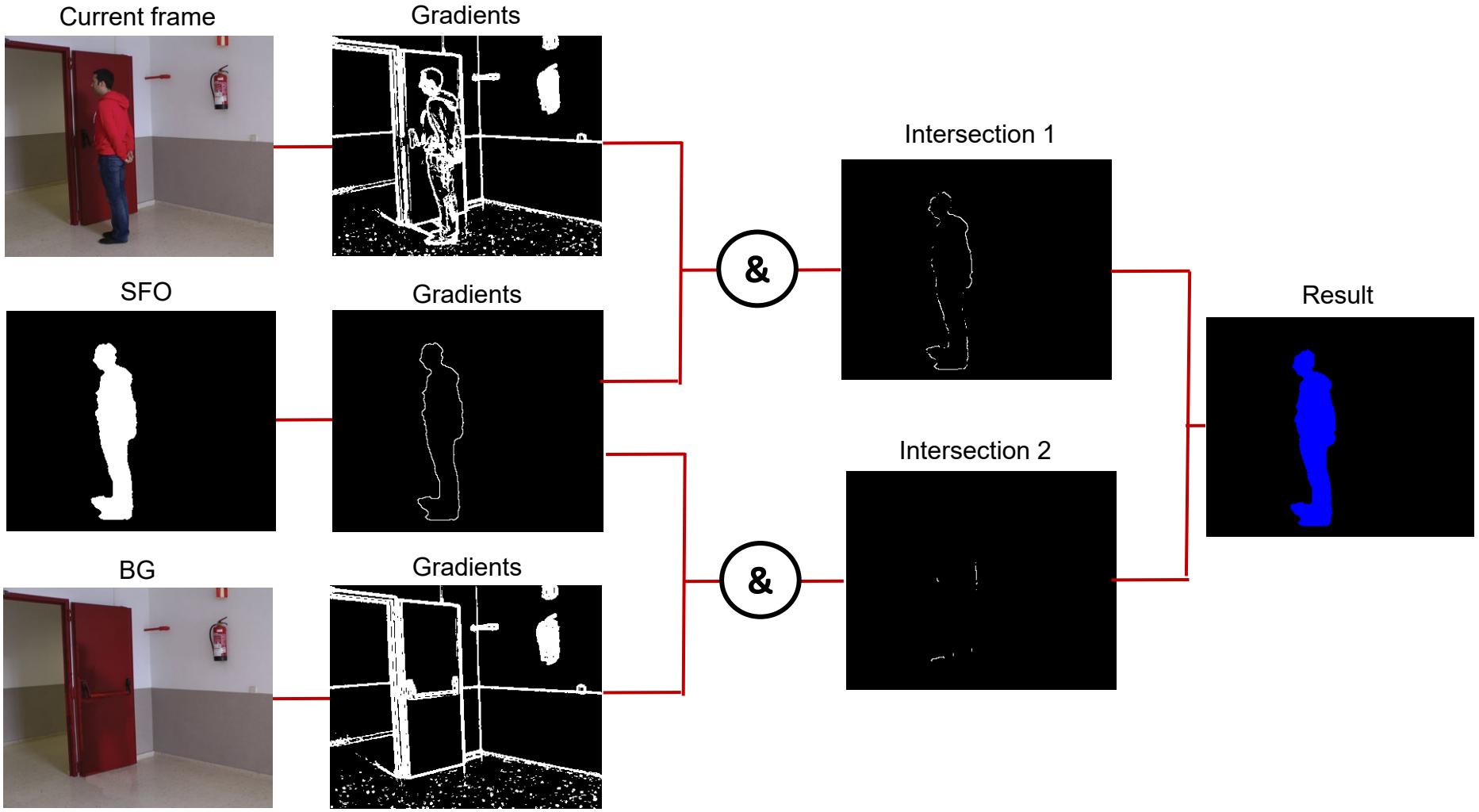


4.2.9 Stationary foreground detection

- **Removed objects:**
  - There are three main groups of strategies for detecting and separating removals from abandonments:
    - Gradient-based methods.
    - Region growing methods.
    - Color-based methods.
  - In any case, it is possible to distinguish among:
    - Methods using a point-estimation of the BG:
      - ✓ They are able of discriminating between abandonments, removed objects and false detections.
      - ✗ The point-estimation of the BG is not always available.
    - Methods not using a point-estimation of the BG:
      - ✓ They avoid using a point-estimation of the BG.
      - ✗ They are unable of discriminating between removed objects and false detections.

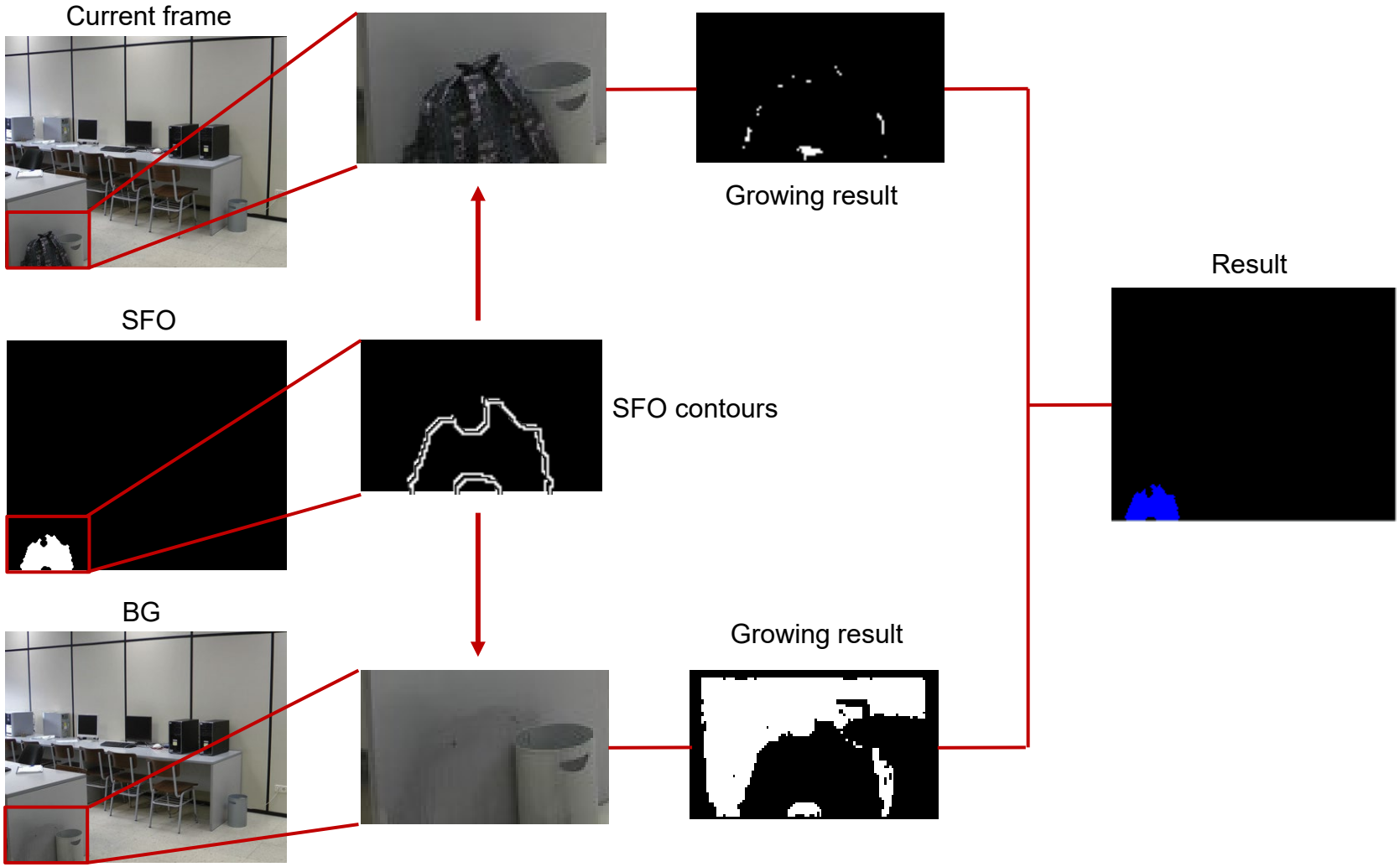
4.2.9 Stationary foreground detection

- **Removed objects – Gradient-based methods:**



4.2.9 Stationary foreground detection

- **Removed objects – Region growing methods:**



4.2.9 Stationary foreground detection

- **Removed objects – Color-based methods:**