Corso di Laurea Magistrale in Ingegneria Meccanica

Laurea di II Livello

Development of Novel Vision Based Surveillance Algorithm for People Tracking, Suspicious Object Detection, and Loitering Detection

Supervisor: Prof. Andrea Trivella

Co-Supervisor: Prof. Yukinori Kobayashi

Candidate: Matteo D'Angelo

2018

Acknowledgements

I am deeply thankful to my supervisor, Prof. Yukinori Kobayashi, whose support has always been present and concrete since the beginning of this challenging and extraordinary experience. Furthermore, I would like to share the credit of my work with Abhijeet Ravankar, whose help, guidance and cooperation convincingly gave me the opportunity to discover an unexplored world.

Abstract

Remote camera surveillance systems together with image recognition have gained much more importance over the last years. Suspicious behaviors, target tracking, wildfires recognition, traffic controlling and autonomous patrolling on wide areas are just few fields of application. Furthermore, modern activities and organizations seek out services and infrastructure that are cost-effective and simple to manage. In this experimental research, by providing the program with wide used filtering functions, people tracking, suspicious object recognition and loitering detection are deeply investigated in order that a highly performing algorithm is formulated. The groundwork is organized as follows: image fundamentals and mathematical operations on images, algorithms and methods commonly used in video surveillance, the formulation proposed, benchmark analysis and conclusions. We suggested a thorough algorithm able to deal with a broad variety of different scenarios. The proposed Waterfall algorithm increased the reliability by outdoing the challenges in the background modeling process. By taking into account the two background models, abandoned and taken away objects are robustly detected. Additionally, trespassing detection and loitering detection are successfully achieved. Furthermore, time thresholds are investigated together with important and relevant statistical tools, which notably helped us to understand the tendency of the position of the target. Undoubtedly, future work can be arranged in the interest of enhancing the robustness, the overall performances and the accuracy of the main functions of the algorithm.

Contents

List of Figures **vi**

List of Tables **ix**

1 Introduction **1**

1.1 Introduction to People and Object Tracking and Video Surveillance Industry 1

1.2 People and Object Detection 3

1.3 Video Surveillance-integrated Robotic Systems 4

2 Image Fundamentals **6**

2.1 Introduction 6

2.2 Binary Images 8

2.3 Operations on Images 8

2.3.1 Normal Histogram Equalization 12

2.3.2 Contrast Limited Adaptive Histogram Equalization 16

2.3.3 The Relevance of Image Filtering in Real-life Applications . 18

2.4 Operations on Binary Images 18

2.4.1 Segmentation 19

2.4.2 Thresholding 19

2.4.3 Morphological Transformations 21

3 The OpenCV Library and Embedded Systems 26

3.1 Introduction 26

3.2 OpenCV with Python 27

3.3 Realtime Computer Vision with OpenCV 27

3.3.1 The GPU OpenCV Module 28

3.4 Embedded Systems 30

3.4.1 Applications 31

3.4.2 Characteristics 32

3.4.3 Raspberry Pi 32

4 Algorithms and Methods for Video Surveillance 34

4.1 Introduction 34

4.2 Background Modelling 35

4.2.1 Two Simple Methods for Background Modelling 35

4.3 Other Approaches to Background Modelling 37

4.3.1 Pixel-based Methods 39

4.3.2 Running Gaussian Average 39

4.3.3 Modified Moving Average 40

4.3.4 Gaussian Mixture Model 42

4.3.5 Adaptative Gaussian Mixture Model 45

4.3.6 A Self-organizing Background Subtraction Method 46

4.3.7 Region-based Methods 47

4.3.8 Kernel Density Estimator 48

4.3.9 Hybrid Methods 49

4.4 Challenges in Background Modelling 50

4.5 Moving Object and Pedestrian Detection 51

4.5.1 The Waterfall Algorithm 53

4.6 Multiple Moving Object and Pedestrian Detection 57
4.7 Unattended or Abandoned Object Detection 58
4.8 Stolen or Taken Away Object Detection 67
4.9 Trespassing Detection 70
4.10 Loitering Detection 72
4.11 Methods for People and Object Tracking 79
4.11.1 Template Matching in OpenCV 81

5 Benchmark Analysis **82**
5.1 Introduction 82
5.2 Python Benchmarking 83
5.3 Results 85
5.3.1 Optimization 88
5.4 Real-time Application 90

6 Conclusions **92**

References **93**

List of Figures

1.1 Scheme of a video surveillance system with a static camera 3

2.1 Most popular formats for image resolution 7

2.2 Three different ways of image filtering 9

2.3 Kernel matrix used for common blurring and the one used for Gaussian blurring. 10

2.4 Kernel convolution. The central element is replaced by the result of the expression. 11

2.5 Normal blurring with the different sizes of the Kernel matrix 11

2.6 Gaussian blurring with the different sizes of the Kernel matrix . . . 12

2.7 Median filter with the different sizes of the Kernel matrix 12

2.8 An example of histogram 13

2.9 Low-contrast picture and its normal histogram equalized counterpart 14

2.10 Low- and high-contrast comparison with histograms 15

2.11 CLAHE limitation on pixel values 16

2.12 Low-contrast picture and its contrast limited adaptive histogram equalized (CLAHE) counterpart 17

2.13 The application of different thresholds on the same gray-scale image 20

2.14 Balanced Histogram Threshold. 21

2.15 Erosion image compared to its reference image 22

2.16 Dilation image compared to its reference image 23

2.17 Two examples of opening and closing 24

3.1 Block diagram of a general-purpose embedded system. 30

3.2 Raspberry Pi 2 board 33

4.1 Background images obtained with two basic background modelling methods 38

4.2 Comparison among different learning rates α at the same frame with Running Gaussian average method 41

4.3 Kernel Density Estimator and its relative histogram 49

4.4 Background Subtraction with a fast-update background model . . . 52

4.5 Background subtraction with a fast-update background model after the process of erosion and dilation 53

4.6 Waterfall algorithm flowchart 54

4.7 Proximity check criterion 55

4.8 Drawing two bounding box around a foreground object 56

4.9 Result of the Waterfall algorithm 57

4.10 Drawing a bounding box around a foreground object 58

4.11 Flowchart of object and pedestrian detection 59

4.12 Particular of the flowchart of multiple object/pedestrian detection . . 60

4.13 Drawing two bounding box around two foreground object 61

4.14 Dynamically updating background models 62

4.15 Unattended or abandoned object detection process 64

4.16 Cropped unattended object 65

4.17 Flowchart of abandoned or unattended object detection 66

4.18 Stolen object detection process 69

4.19 The process of trespassing detection 71

4.20 Flowchart of trespassing detection 71

4.21 Scheme of the two angles $\theta_{horizontal}$ and $\theta_{vertical}$ considered for loitering detection 74

4.22 Scheme of the two variance thresholds 77

4.23 Different phases of the loitering detection process, with $\theta_{horizontal}$ and $\theta_{vertical}$ displayed in the upper left corner of the frame 77

4.24 Flowchart of loitering detection 78

4.25 Template matching comparison 80

5.1 Implementation from the Terminal of Unix utility time 83

5.2 The processing time of the three main functions which compose the proposed algorithm 86

5.3 The comparison of processing time of the core of the algorithm . . . 88

5.4 The processing time of the three main stages of Background modelling 89

5.5 The comparison of processing time of the main stage of Background modelling 90

List of Tables

4.1 Background modelling methods 49

5.1 Computer specifications 84

5.2 Processing times of the subprocesses composing the background modeling phase 87

5.3 Processing times of the subprocesses composing the image treatment phase 88

5.4 Processing times of the three main functions which compose the proposed algorithm 91

Chapter 1

Introduction

1.1 Introduction to People and Object Tracking and Video Surveillance Industry

Video surveillance systems have become a solid reality in everyday life. The origins of the first video surveillance systems are traced back to 1940s, with the invention of closed circuit television (CCTV). Due to military activities, its main early uses was intended to obtain different types of information from aircrafts and from satellites. However, with the advent of cheaper video cameras and, more generally, optical sensors, security systems became widely used items for new building constructions, powerplants, shops, apartments, subway stations and airports monitoring. Furthermore, modern activities and organizations seek out services and infrastructure that are cost-effective and simple to manage. Likewise, the request and the importance of keeping high level of security in running businesses such as production plants have become a priority, resulting in an increasing trend toward surveillance to:

- Enhance overall safety and security;
- Reduce theft both internally and externally;
- Discourage vandalism and other minor crimes;
- Improve business operations and productivity;
- Improve customer experiences and grow loyalty.

Consequently, as a matter of fact, video surveillance has become a fast-growing business since its invention. The wide variety of possibilities has increased significantly over the years and their applications are keeping on emerging [1].
In the early years, pedestrians tracking and object recognition have become another relevant field of research and application also due to the recent and current global trend of terrorist attacks. Therefore, it has become necessary to develop and search for new and more accurate ways of monitoring wide areas, especially public and crowded spaces, both for civil purposes and for safety reasons. These systems are engineered in order to make the accomplishment of a safer and more precise surveillance possible. Certainly, making such kind of engineering devices available for any kind of application would make the prevention of terrorist attacks more straightforward. However, the application mentioned above is not exclusively intended for protection and public safety. Indeed, more and more areas of application have been developed through the years. Traffic controlling, illigal parking detection, riot controlling, monitoring of endangered species and wild areas, home nursing, augmented reality, video editing and transportation flow analysis are just few examples that may be mentioned along with others [2]. On the other hand, the recurring decrease in labor force, particularly in developed countries, and the effective reliability of operators while monitoring, are other factors that have to be taken into account. In fact, the majority of safety cameras are connected to video recorders which gather thousands of hours of videotape, the majority of which will never be looked at. Others are connected to closed circuit television monitors where human operators may be able to extract some frames of the event displayed or may be fallacious in doing so. In these cases, but not only, tedious and monotonous works can be easily replaced by automatic surveillance systems, resulting in savings resources and resulting in the increase of the task performances and accuracy. Even if this could be accomplished in high-velocity videotapes analysis, where the amount of time saved would be significant.
Some people say that human privacy could be deeply affected by such invasive video surveillance systems, and this is unquestionably true. However, a tradeoff analysis between privacy and security issues is then necessary even if it is a matter of fact that most of the people would consider a small loss of privacy a welcome price to pay to achieve an higher level of security [3].
Advanced autonomous surveillance systems requires the application of many functions such as visual surveillance of objects motion and their behaviors, visual and

automatic recognition of human body language, segmentation and real-time tracking, silhouette recognition and identification and, above all of them, image processing operations. In fact, in order to achieve good results with any type of image and frame quality, image operations are strongly needed. For instance, real-time tracking is used to detect one, or more, target object (e.g., suspicious person or object) in consecutive frames of the video. The position is then calculated using reference systems depending on the type of video surveillance system and their relative position to the observed target.

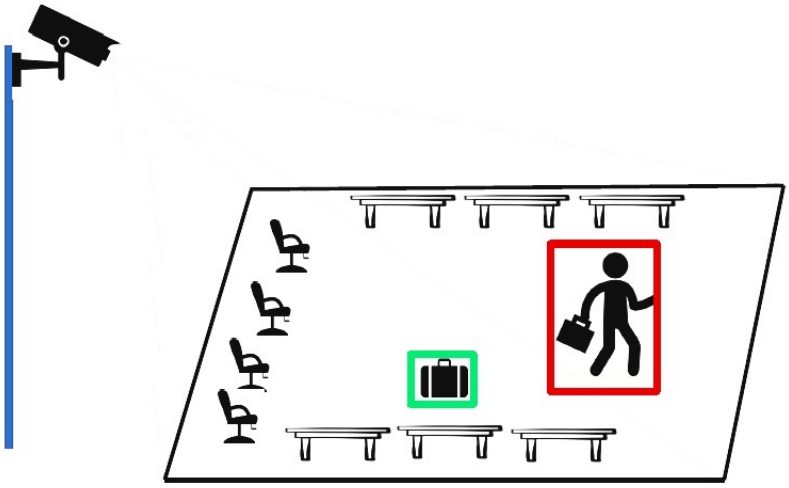


Fig. 1.1 A simplified scheme of a video surveillance system with a static camera.

1.2 People and Object Detection

People and object detection is a particular branch of the field of video processing in digital image handling systems. People and object detection is the process of locating a moving object over time using a camera [4]. Therefore, it is totally understandable that the frame rate[1] of the video plays an important role, in particular for those application where objects are moving fast relatively to the frame rate. Another issue can be related to the data contained in the video file that may affect the effectiveness and the performance of the whole process. Furthermore, another situation that increases the complexity of the problem is related to orentation changes of the

[1] fps.

tracked object over time [4]. With the regard to the last scenario, difficulties arise when the object (or subject) to be tracked are pedestrians rather than cars or constant-shape objects. First, humans change shape significantly as they move. Second, they vary in shape, size, and the colors they wear may cover an impressive range: from very garish colors to opaque colors. Third, it is likely that tracking a person in a very crowded space may result in the loss of the path. Moreover, the continous change in the lighting conditions, how an object reflects lights and produces shadows, moving shadows from vehicles and clouds are other common factors that have to be considered. All these problems may seem easy to manage, if considered separately, but the algorithms that have been developed so far have a limited range of application, as deeply explained in the further Sections. Therefore, the handling of all these issues together may be much more difficult than it seems. For instance, because of the need to operate continously in real time, some algorithms are quite simple, whereas others are much more complex because they have to guarantee an higher accuracy. In any case, it is quite uncertain that a human operator could always get correct anwers, given the visual data that the computer receives [3].

1.3 Video Surveillance-integrated Robotic Systems

All the features previously discussed can be integrated by the advanced use of intelligent robotic systems. Nowadays the importance of such kind of devices is remarkably increasing due to the necessity to control more autonomous systems. These systems can be designed to deal with a large amount of different scenarios making it possible to increment the efficiency and the reliability of specific applications.
In our case, we test our algorithm through different scenarios where both people and objects are moving inside the videosurveilled area. The aim is to detect suspicious behaviors and objects to state whether they may be risky and unsafe or not. Once the video surveillance system declares the existence of an imminent danger, another scenario, which will need further studies, can be considered: a robot can be sent to check the area in order to have a deeper understanding of the data collected in the previous phase; consequently, stronger results could be retrieved. For instance, if the system is applied to a public square or a subway platform and the warning of an abandoned object is triggered, a UGV[2] robot can be maneuvered in the direction of

[2] An Unmanned Ground Vehicle is a vehicle which operates without an onboard human presence.

the suspicious object. Because of the proximity of the robot to the target, more and different data can be collected. The use of x-ray and thermal cameras, for example, may help to detect the presence of explosive or hazardous materials which make the presence of a human operator inconvinient, threatening and critical. Furthermore, the UGV could be equipped with a mechanical limb to gather other different measures such as the weight of the target (an abandoned suitcase, a box, a piece of luggage).

Chapter 2

Image Fundamentals

2.1 Introduction

In computer vision or image recognition and processing systems an image is considered as an array of pixels, minuscule cells which contain information. Thus, an image, digitally speaking, is a simple numeric representation. Depending on the quality of the sensor used to acquire the image, a pixel can usually carry 8 bits of useful information or even 9 bits. For instance, older television cameras typically gave a gray-scale resolution that was accurate only to about one part in 50, corresponding to about 6 bits of useful information per pixel. On the contrary, modern cameras commonly give less noise and may allow, as mentioned above, 8 or 9 bits of information per pixel. Nonetheless, in some applications, carrying that big amount of information may not be needed, in particular when the result will not be visible to the human eye [3]. Likewise, depending on the precision or on the field of the application, carrying colors along with the image itself may be useless. Many applications, for example, are only intended to gather information about the precise shape of the objects inspected rather than their colors. That is the case of mechanical components inspected on a workbench or through an automatic system with the use of a conveyor belt. On the other hand, if we are designing an automatic fruit picker, it will be crucial to check colors. A green apple has the same shape of a red apple, but it may be necessary to categorize them into different apple qualities. Thus, the potential of color in helping with many aspects of inspection, surveillance, control, and a wide variety of other applications including medicine

above all the others, is outstanding. Next, it is useful to consider not the potential but the processing aspect of color. In many cases, good color discrimination is required to separate two kinds of objects from each other, as demonstrated by the example above. Generally, this means not to use one or another specific color channel but subtracting two, or combining three in such a way as spot the differences. The problem arises because the amount of processes and suboperations in different cases may vary in an enormous range. For instance, if each pixel is treated identically, the process load will be very light. On the other hand, if we want to combine data from the color channels and carry out different operations on the same image at the same time, this may be far from doable. In contrast with gray-scale images, where the bits representing the gray-scale are all of the same type and, thus, can be processed as single entities, color signals are inhomogeneous [3].
The definition of a digital image is therefore related to image processing. An image, as cited above, represents the amount of pixels through which the image itself is digitally represented. For instance, an image with a 1280x1024 definition has 1280 pixels on the horizontal edge and 1024 pixels on the vertical edge. A lower quality image can be represented by a 320x240 rectangular pixel-made shape. Aspect Ratio (often AR) is the division between the two edges of the images: the width and the height both measured with the same unit. The most famous standard Aspect Ratios are the 16:9 International Standard for HD Televisions, the 4:3 International Standard for the Computer monitors and it used to be the standard for the 35 mm films in the silent era.

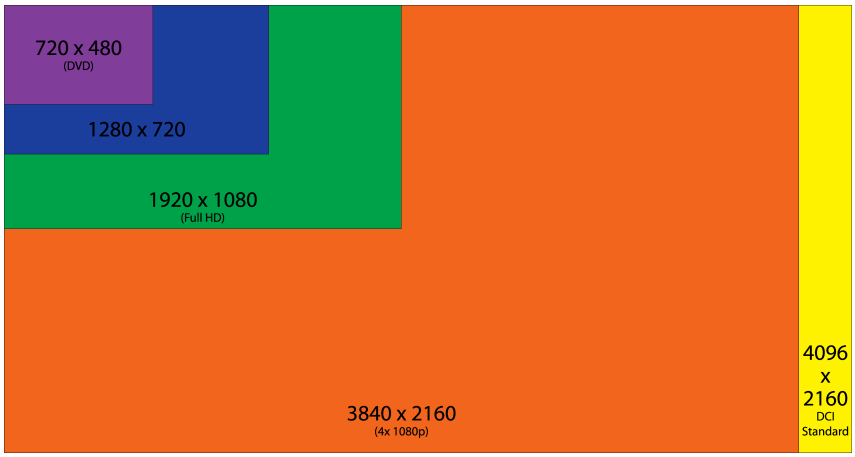


Fig. 2.1 Most popular formats for image resolution with the constant AR of 16:9.

Considering Fig. 2.2 we may ask whether it would be better to replace Fig. 2.2b with Fig. 2.2a, using the three channels for the three main colors (Red, Green and Blue). Generally, two aspects of color are fundamental for machine vision: the intrinsic value of color and the additional storage and processing penalty it may bring. It is tempting to say that the second aspect is of no such great importance if we consider the cheapness of modern calculator which have both high storage and high speed. On the other hand, high-resolution images can be gathered at huge data rates and it could take many years to analyze exaclty all the data stored so far. As a consequence, if the process of adding colors to a gray-scale image substantially brings additional storage and processing penalties, its use will have to be justified [3].

2.2 Binary Images

Among all the types of digital images, binary images, or bi-level images, play a relevant and significant role. Even though they contain much less information than their gray-scale counterpart, they are used to shape detection. Binary images are composed of pixels which can only acquire two different values. Generally, the most common colors used for binary images are black and white because of their ease to be distinguished, even though any colors can be used [5]. In fact, binary images only have one channel (instead of the three channels of RGB images, Red, Green, Blue) where it is possible to disinguish the brightness intensities that goes from 0 (black) to 255 (white). These 256 values comes from 2^8. Binary images often appear in digital image processing as masks or as the result of operations such as segmentation, dilation, erosion and so on. In Fig. 2.2 the binary image is obtained setting the threshold to 100. This means that all the pixels below the threshold are set to the brightest value, thus 255.

2.3 Operations on Images

In image processing there is an immense variety of operations that can be applied on digital images. Starting from very basic operation, maybe the simplest one is that of selecting all the pixel of the image and setting them to a specified and constant

(a)

(b)

(c)

Fig. 2.2 Three different ways of image filtering. (a) Shows the original image, (b) Shows the gray-scale image, (c) Shows the binary image.

value. Practically, this operation can be achieved by implementing a two-dimensional for-loop since the image can be expressed as a two-dimensional array of the local pixel intensity value. Another very simple operation is to copy an image to another, resulting in two identical images at the end. As far as we can see, there are many easy operations of this type like shifting the image left, right, up, down and so on. Most practical situations demand more sophisticated operations. Among them, thresholding is often applied to gray-scale images to convert them to a binary images, as widely explained in Section 2.4. In the case of more sophisticated operation such as Gaussian blurring, the definition of the so-called Kernel matrix is fundamental. The Kernel matrix is used for the operations mentioned in the following sections and it depends on the operation that has to be achieved [6]. Here are some examples:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.1}$$

Fig. 2.3 Kernel matrix used for common blurring and the one used for Gaussian blurring.

Typically, any kind of image processing operation is accomplished by doing a convolution between a Kernel matrix and an image. For instance, if we have two $3x3$ matrixes, convolution means inverting the Kernel matrix and making an element by element multiplication summing up the results at the end. The obtained result will become the central element of the resulting processed image. Clearly, this is an iterative process, and the image would be composed by a combination of all the values of the Kernel matrix adopted. In the figure below, we can see the process of replacing the central element of the resulting element by element multiplication, where the literal matrix represents the Kernel matrix:

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2,2] =$$
$$(i*1)+(h*2)+(g*3)+(f*4)+(e*5)+(d*6)+(c*7)+(b*8)+(a*9). \tag{2.2}$$

Fig. 2.4 Kernel convolution. The central element is replaced by the result of the expression.

Beside Fig. 2.3 that shows two different Kernel matrices for two different operation, another filter that can be used is the median filter. Median filter is very broadly used in digital image processing because, under certain conditions, it preserves edges while removing noise. Its main idea is to scroll through each pixel value and replace each of them with the median value, which can be considered as the middle value of a general dataset, a data sample, a probability distribution [7]. Specifically, the dataset is represented by the neighboring pixels in image processing and the median value is considered among the pixels which are under the Kernel area. The pattern that neighbor pixels follow is typically called "window" and it is obvious that in one dimensional arrays the simplest window is just the elements preceding and following the median pixel. In two or higher-dimensional datasets, a bigger complexity can be added to the window such as box or cross patterns.

(a) (b) (c)

Fig. 2.5 Normal blurring with the different sizes of the Kernel matrix: (a) 3x3 Kernel, (b) 11x11 Kernel, (c) 25x25 Kernel.

(a) (b) (c)

Fig. 2.6 Gaussian blurring with the different sizes of the Kernel matrix: (a) 3x3 Kernel, (b) 11x11 Kernel, (c) 25x25 Kernel.

(a) (b) (c)

Fig. 2.7 Median filter with the different sizes of the Kernel matrix: (a) 3x3 Kernel, (b) 11x11 Kernel, (c) 25x25 Kernel.

2.3.1 Normal Histogram Equalization

Histogram Equalization, among the other practical operation in image processing, needs to have a specific explanation due to its importance and relevance.
First of all, what is an histogram? It is a graph which gives a broad overall idea about the pixel values distributed in the considered image. Generally, the x-axis represents the pixel values ranging from 0 to 255, whereas the the y-axis shows the occurences of a specific pixel value. By analyzing an image histogram, we can get a fair amount information about brightness, exposure, contrast, etc. Nowadays, with the use of digital cameras and other image sensors, all the image processing tools provides features on histograms.
An example is displayed in Fig. 2.8.
Normal Histogram Equalization procedure is largely applied to those images which lack of contrast and exposure. The operation exploits the image's histogram to adjust those parameters mentioned above, generally enhancing the overall constrast and remarkably re-setting to good values the exposure of the image. Through these adjustments, the pixel value intensities, represented by the histogram, can be better

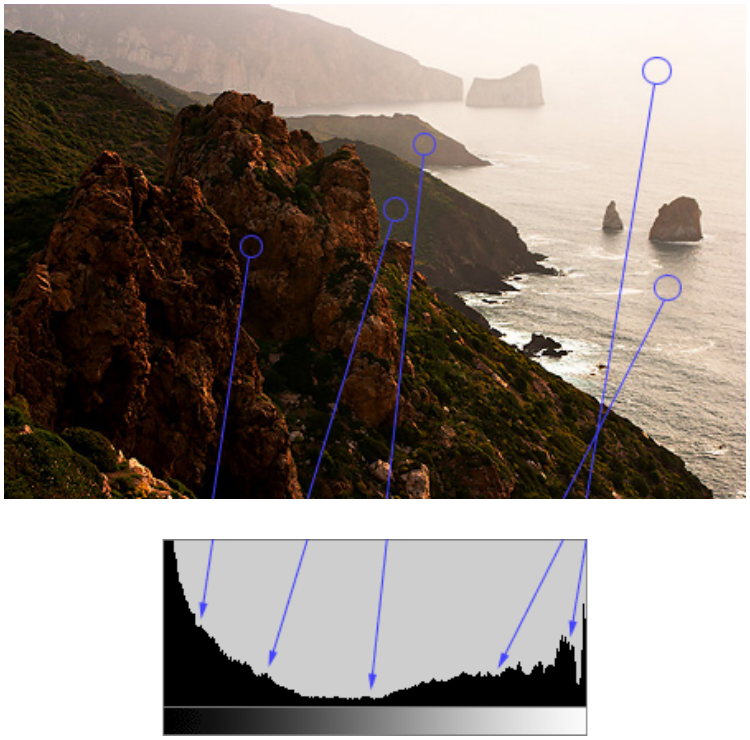


Fig. 2.8 An example of histogram (source: cambridgeincolour.com).

distributed on the histogram itself, as shown in Fig. 2.9. This allows for areas of lower local contrast to gain a higher contrast.

This method is really useful when both background and foreground share the same intensity values; hence, they are either dark or bright. For instance, Normal Histogram Equalization can achieve great results in displaying bones and tissue structures in x-ray images. Different and various results can be also achieved in the field of photography, for those pictures with high or low exposure. Furthermore, thermal and satellite images can be optimized.

One of the major advantage of Normal Histogram Equalization is that the mathematical implementation is invertible. As a consequence, given the equalized histogram of a sample picture, the original histogram can be recovered. Additionally, the implementation is not computationally intensive. Nonetheless, Normal Histogram Equalization method may often produce unrealistic effects in photographs. For example, it can also yield to inconvenient effects when applied to images with low color depth. For example, if applied to 8-bit image displayed in gray-scale, it will further reduce color depth of the image. Histogram Equalization will work the best

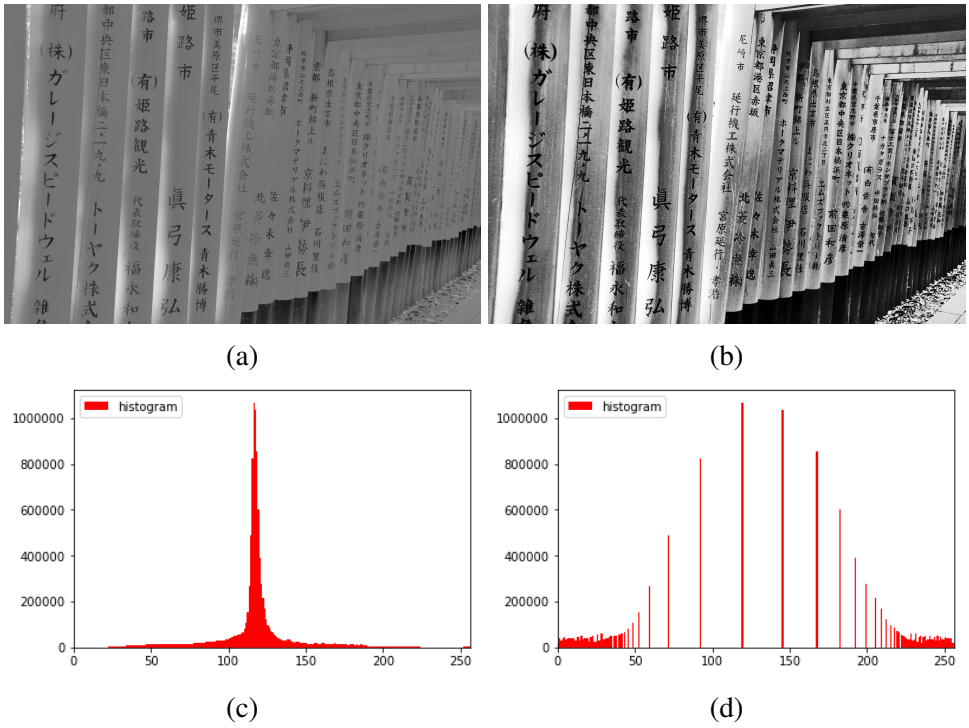


Fig. 2.9 Low-contrast picture and its normal histogram equalized counterpart. (a) Shows a gray-scale low-contrast image, (b) Shows the same picture after applying the Normal Histogram Equalization, (c) Represents the histogram of the low-contrast image. Again, we can see how the pixel intensity values are remarkably distributed in a very limited part of the graph, (d) Represents the obtained histogram. Here, it is easy to notice how the pixel intensity distribution has sharply been distributed in a wider part of the graph.

when applied to images with much higher color depth than the give palette [1] size, like continuous data or 16-bit gray-scale images [8].

Let us consider a gray-scale image G and let us consider n_i the number of recurrences of the pixel intensity i. Hence, the probability of finding a pixel $p_G(i)$ with the intensity value of i is:

$$p_G(i) = \frac{n_i}{n} \tag{2.3}$$

where i is the number of total gray-scale levels in the image, which is typically 256. n is the total number of pixel in the image and $p_G(i)$ is the value of pixel intensity i's histogram, normalized to $[0,1]$.

Let us also define the cumulative distribution function cdf_G. Especially in these kinds of application, the CDF should be named cumulative histogram since it is

[1] In computer graphics and image processing, a palette is a distinct set of colors.

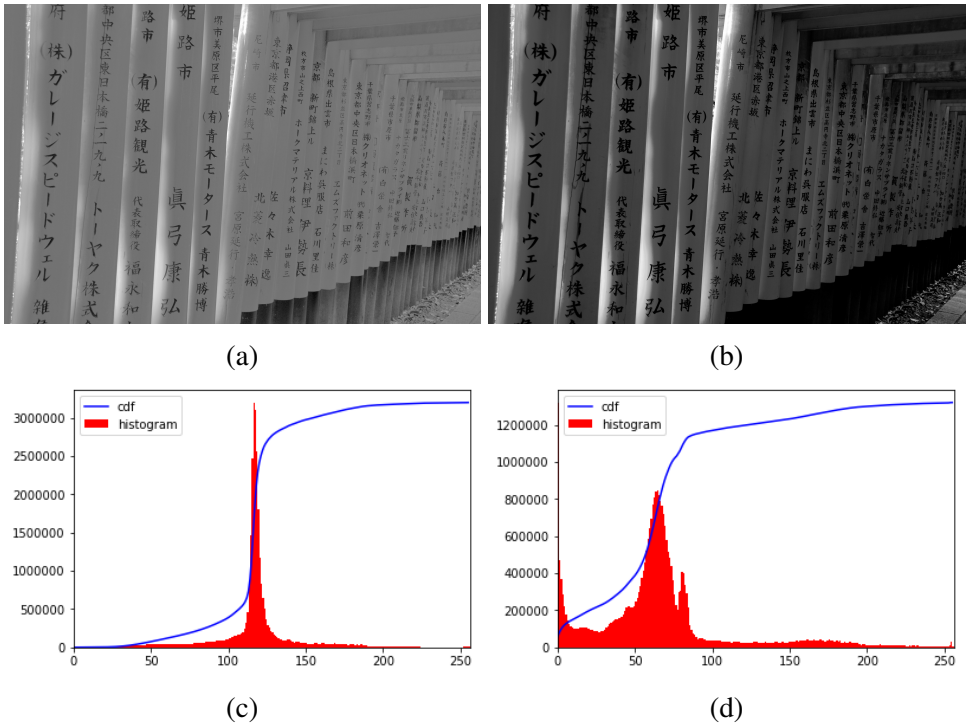


Fig. 2.10 Low- and high-contrast comparison. (a) Shows a gray-scale low-contrast image, (b) Shows a gray-scale high-contrast image, (c) Represents the histogram and its cumulative distribution function of the low-contrast image. We can clearly see how the pixel intensity values are remarkably distributed in a very limited part of the graph, (d) Represents the histogram and its cumulative distribution function of the high-contrast image. Here, the pixel intensity values are sharply distributed in a wider part of the graph.

derived by dividing values in the cumulative histogram by the overall amount of pixels.

$$cdf_G(i) = \sum_{k=0}^{i} p_G(k) \tag{2.4}$$

After it, our aim is to create a transformation of the form $H = T(G)$ to produce a new image H. The new image H should have a flat histogram. Consequently, this image will have a linearized cdf_H function like the one shown in Equation 2.5:

$$cdf_H(i) = i \cdot C \tag{2.5}$$

where C is only a constant. Then:

$$cdf_H(H') = cdf_H(T(k)) = cdf_G(k) \tag{2.6}$$

where k is again the number of total gray-scale levels in the image, ranging in $[0,255]$. It is important to point out that the transformation T maps the level of the new image H in a normalized range $[0,1]$, since we used the normalized histogram of the previous image G.
Because of the mathematical invertibility of the Histogram Equalization, we can map the values back into their original range $[0,255]$ by implementing Equation 2.7:

$$H' = H \cdot (\max\{G\} - \min\{G\}) + \min\{G\} \tag{2.7}$$

2.3.2 Contrast Limited Adaptive Histogram Equalization

The Constrast Limited Adaptive Histogram Equalization (CLAHE) is a particular case of Adaptive Histogram Equalization (AHE). AHE differs from the Normal Histogram Equalization since it uses several different histograms deduced from distinct sections of the same image. Hence, AHE is suitable for improving the local contrast and enhancing the definitions of edges in each of the specific region of the image. To achieve this objective, AHE applies the so-called *tile*.

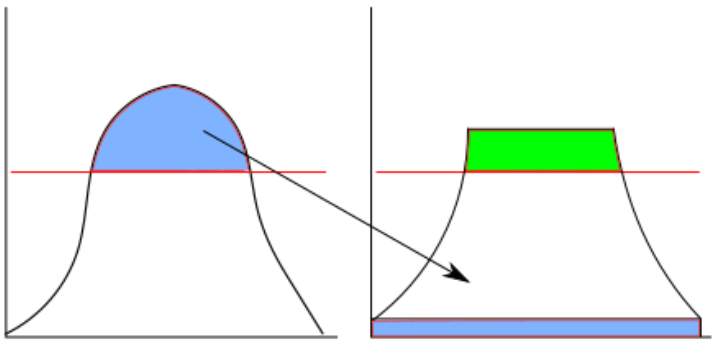


Fig. 2.11 CLAHE limitation on pixel values.

Tiles are small blocks belonging to the image that would be equalized as usual. However, AHE presents a big drawback to be considered. It sometimes overestimates noise and defects, but not only, in regions where pixel values are considerably homogeneous. This is where CLAHE plays its important role. Indeed, to avoid the enhancing of noise in certain sectors, CLAHE puts a limit on the contrast, as shown in Fig. 2.11. Then, if one of the calculated histogram is above the set limit, those pixels are clipped and distributed homogeneously on the very base of the general histogram before re-applying the equalization again. The redistribution procedure can be repeated recursively until the excess is negligible.
An example is shown in Fig. 2.12.

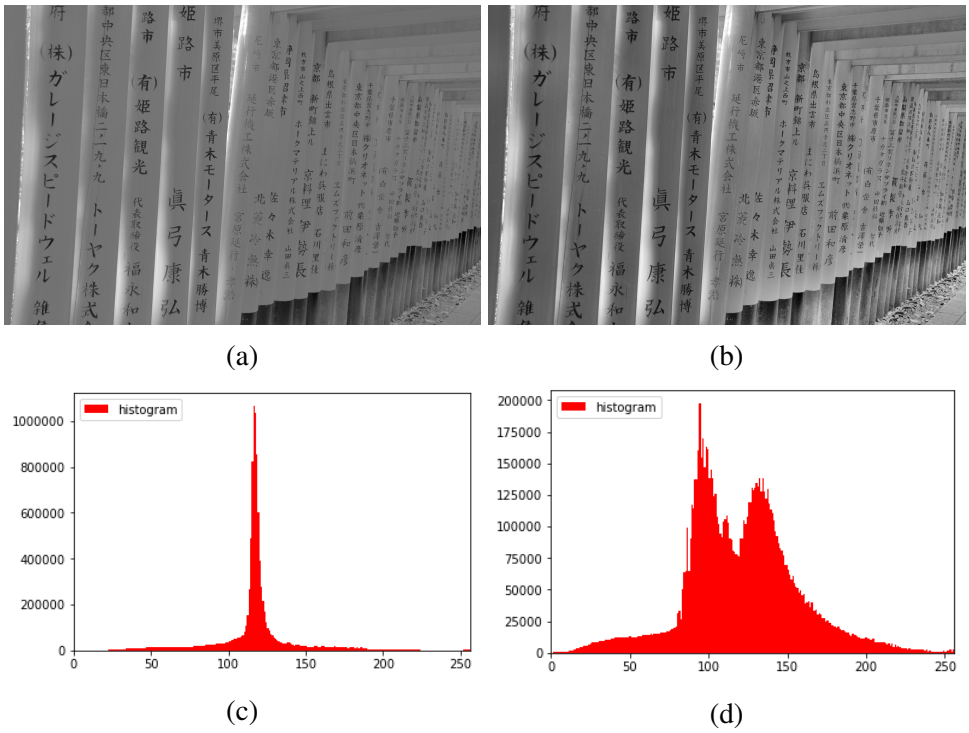


Fig. 2.12 Low-contrast picture and its constrast limited adaptative histogram equalized (CLAHE) counterpart. (a) Shows a gray-scale low-contrast image, (b) Shows the same picture after applying the Contrast Limited Adaptive Histogram Equalization, (c) Represents the histogram of the low-contrast image, (d) Represents the obtained histogram. Compared to the Normal Histogram Equalization the pixel values are unsymmetrically distributed due to the inner nature of the method, explained in the Subsection.

2.3.3 The Relevance of Image Filtering in Real-life Applications

So far we have analyzed a wide variety of operations on both color and gray-scale images. Their major aim is to remove background noises, spots and defects. However, in some real-life applications, such as scrutinizing of high-precision mechanical components on a production line, those filters may remove fine details such as lines and small holes, considered as fundamental features. Additionally, it also has been found that an erroneus filtering may lead to corner clipping and so on. Eventually, filtering may moderately shift edges, which, in some high-precision applications, as the one cited above, are vitally important.
Such imprecisions are quite disturbing and mititgate against the variegated and indiscriminate use of filters [3]. If applied in situations where precise measurements have to be made on images, particular care must be considered in order not to dramatically change the nature of the image. Even though it is feasible to make reasonable corrections to the modified data, it is obviously harder to accomplish them after the application of filtering processes rather than before. For this reason, often an alternative way is taken: the edge, shape, and object detection algorithms integrate partially or totally defect-abolishment processes as an integral part of their functions [3].
Summing the principles explained in this Subsection, we can state that one of the preeminent scope of algorithms should be robustness towards defects, noises and other errors that may affect measurements and the overall validity of the algorithm itself.

2.4 Operations on Binary Images

The operations which can be made on binary images are reflected on important mathematical properties that go under the name of Mathematical Morphology. This field of study has encountered a rapid development over the last twenty years. It has reached a considerable depth resulting in a consolidated basis for the current ongoing studies in the branch of the image processing. Mathematical Morphology is highly important because it provides a firm support for the whole study of shapes and object tracking. In digital image processing, binary images are typically used for straight-forward operations due to their ease to be handled. The most common operations that use mathematical morphology are:

1. Subtracting;
2. Morphological Transformation:
   (a) Dilating;
   (b) Erosion;
   (c) Opening;
   (d) Closing;
3. Thresholding;
4. Segmentation.

Even though thresholding is only the simplest way of image segmentation, still it can be considered as a different operation. Within the image, for example, the binary image is captured in order to make the spotting of differences between two consecutive frames easier. Then, it is possible to work on them using filters and other functions as the ones mentioned right above.

2.4.1 Segmentation

Image segmentation is based on the process whereby a group of pixels (also know as super-pixels) are gather together. The target is to create different regions limited by different boundaries of the same image that share similar characteristics such as color, intensity and texture. This eventually leds to a simpler understaing of the image through its demarcation. Segmentation, together with the Edge detection technique is widely and overall used for medical scopes and face recognition.

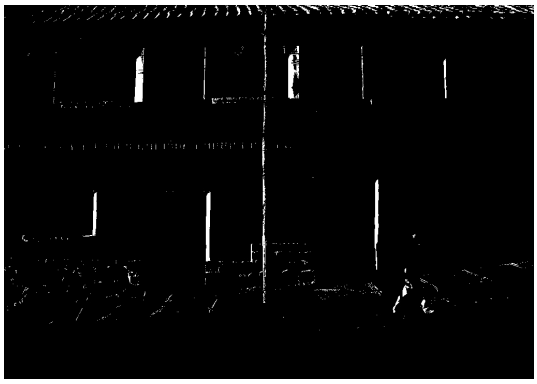
2.4.2 Thresholding

Thresholding is the simplest method of image segmentation and it is often applied to gray-scale images to convert them into binary images. The actual goal of this method is to higlight the desidered objects with the respect of the given background. With a correct choice of the threshold value, which is a severe difficulty, the pixels below the chosen value are commonly set to the brightest pixel value (255), whereas all the pixels above remains unchanged. As a consequence, clearer and more defined shapes can be obtained over a black background. Obviously, the pixel values can be reverted anytime with simple operations, resulting in black-shaped objects over a

(a)

(b)

(c)



(d)

Fig. 2.13 The application of different thresholds on the same gray-scale image: (a) Threshold equal to 10, (b) Threshold equal to 35, (c) Threshold equal to 100, (d) Threshold equal to 200.

bright white background. Again, this process is mainly intended to detect objects on a given background and it can be implemented through different and various algorithms and methods, each of them having a particular efficacy and related weaknesses. In many industrial applications especially, taking control of the light and the general environment conditions is pretty easy. In addition, the objects shapes and variability is quite limited. Hence, Thresholding to be applied in these scenarios is not problematic. The weaknesses arise when our application has to deal with dynamic environment conditions, where the light, brightness and other factors interact significantly with the objects.
Among the various algorithms and thresholding methods, the most frequent is the histogram analysis of the intensity levels involved in the digital image, also called Balanced Histogram Thresholding Method (BHT) [9]. This automatic method is

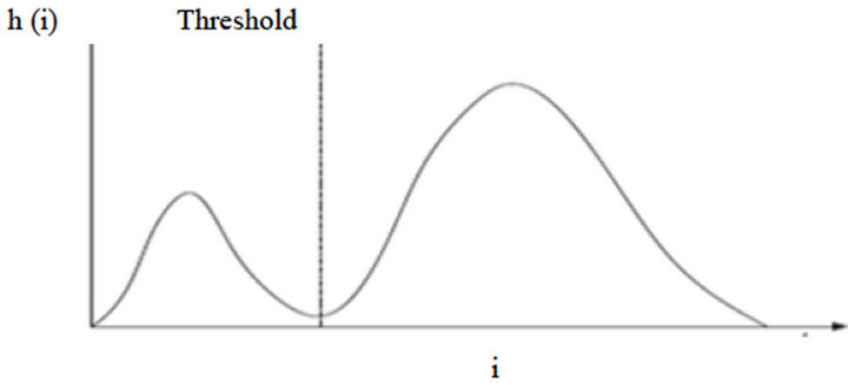
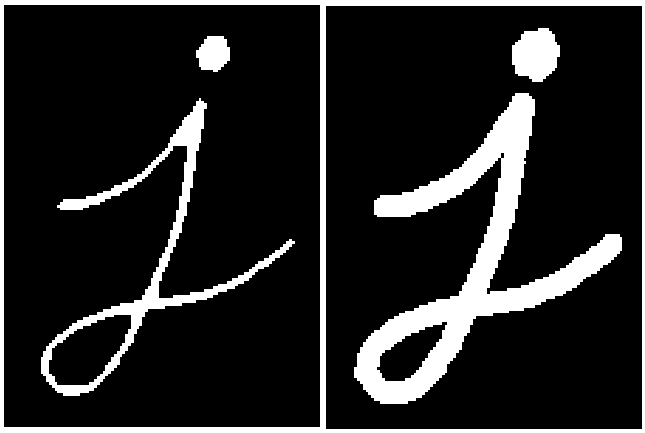


Fig. 2.14 Balanced Histogram Threshold.

based on the assumption that the image is separated in two groups, each of them with different light conditions: the background and the foreground. Assuming the background to be at higher intensity levels, therefore brighter, the method tries to find the optimum intensity level that splits the two groups up. Of course, this method has several critical points that more complicated algorithms try to solve, among them the Adaptative Thresholding.

2.4.3 Morphological Transformations

Among the morphological transformations, we can distinguish four main methods that are conventionally, but not necessarily, applied to binary images. As mentioned above, any morphological transformation needs the input image frame and the Kernel matrix which decides the final goal of the operation to apply to the image.



(a) (b)

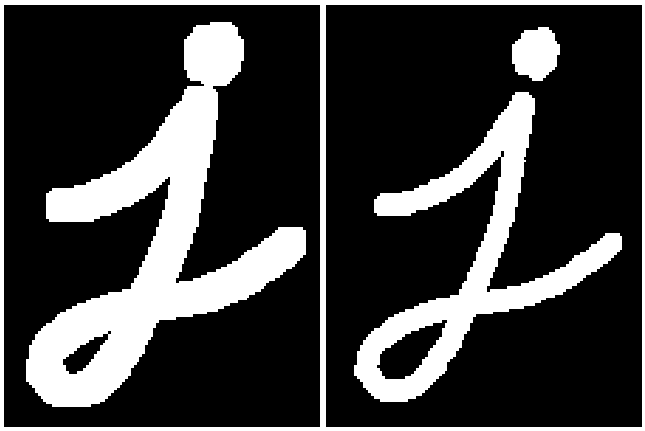
Fig. 2.15 Erosion morphological transformation: (a) Erosion effects, (b) Reference image.

Erosion

As soon as we see the word "erosion", our common sense directly goes to the idea of morphological erosion, like soil erosion. As the weather agents erode the soil, the image erosion erodes the outer boundaries of the foreground object. The Kernel matrix is applied on each pixel of the original binary image and, if all the pixels included in the kernel is 1, the original pixel remains stable at the value 1 otherwise it is eroded, thus turned into zero.
As we can notice from the original picture on the right and from the eroded one on the left, the pixels near the outer boundaries will be discarded. Therefore, the thickness of the foreground object will decrease depending on the Kernel matrix dimension. This method is useful for removing background noises such as "pepper" noise, especially when these points lie very far from a high-density white pixels (in this case) region.

Dilation

Dilation is a method diametrically opposed to erosion. In this case, the Kernel matrix still works on every pixel of the image, but if just one pixel included in the Kernel is 1, the original group of pixels remain inhaltered. Therefore, as we can see from the example below the dilation method increases the density of foreground pixels. Specifically, the process of dilation is able to eliminate "salt" noise as a background noise. Normally, if we need to get rid of background noise, both erosion and dilation



(a) (b)

Fig. 2.16 Dilation morphological transformation: (a) Dilation effects, (b) Reference image.

are applied. This is because erosion removes background noise, but it also shrinks our object and sometimes the shrinkage may be too evident.
As we will see, along with opening and closing, also erosion and dilation are strongly related. They help to join separated parts of the same object to become an unique part of it.

Cancellation Effects

Considering the application of both erosion and dilation, we may ask whether the first one cancels the second one and viceversa. As we mentioned above, if an erosion has been made, the pepper noise will has been removed. If we would like to have those kind of noise back to our image, we could only cancel the operation because of the irreversibility of the operation. The same happens with dilation. Therefore, if we call S the set of pixels, we can write:

$$erode(dilate(S)) \neq S \tag{2.8}$$

$$dilate(erode(S)) \neq S \tag{2.9}$$

More generally, we can state:

$$erode(dilate(S)) \supseteq S \tag{2.10}$$

$$dilate(erode(S)) \subseteq S \tag{2.11}$$

Furthermore, we can also write the dilation and erosion operations formally as:

$$A \oplus B \tag{2.12}$$

$$A \ominus B \tag{2.13}$$

where respectively A is the image and B is the mask or the Kernel matrix applied on the image.

Opening and Closing

Opening and closing are complementary operations compared to erosion and dilation. Basically, opening removes noise from the background image and often opens gaps as shown on Fig. 2.17a, whereas closing removes small holes in the Foreground image and often has the effect of closing gaps between near shapes as shown on Fig. 2.17b. In the Subsection 2.4.3, we were interested in the possibility of an erosion cancelling a dilation and vice versa. Therefore, we have defined these two operations that express a sort of "degree of cancellation". Opening and closing can be formally

(a) (b)

Fig. 2.17 Two examples of morphological transformation: (a) Opening, (b) Closing. They are both referred to shape of the examples above.

defined by the following equations:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.14}$$

$$A \circ B = (A \ominus B) \oplus B \tag{2.15}$$

As we can see, these operations are very important especially for practical applications such as defection tasks. For instance, if we subtract the obtained binary image to the original image, we can be able to spot and locate the defects we cancelled through the operations cited above. Closing and opening have the interesting property to be idempotent. This means that repeating or iterating the same operation more times has no effects, whereas we know that erosion and dilation can be iterated a number of times. The propery of being idempotent can be written as follow:

$$(A \bullet B) \bullet B = A \bullet B \tag{2.16}$$

$$(A \circ B) \circ B = A \circ B \tag{2.17}$$

From a practical point of view, this property is to be expected since any hole or dot that has been filled or removed cannot be further filled or removed without recreating it first. Equation 2.18 shows that it is pointless to apply dilation and opening within the same mask:

$$(A \oplus B) \circ B = (A \oplus B) \tag{2.18}$$

Finally, closing an image will increase the areas of the objects, while opening will make their areas smaller [3].

Chapter 3

The OpenCV Library and Embedded Systems

3.1 Introduction

OpenCV is a library of programming functions concerning real-time computer vision. It was originally developed by Intel in 1999 by Gary Bradsky and the first release came out in 2000. Later on, OpenCV library active development was supported by Willow Garage, a technology incubator devoted to developing hardware and open source software for personal robotics applications. The company was probably best known for its open source software ROS (Robot Operating System), which has been rapidly and widely becoming a common, standard tool among robotics researchers and industry, since its initial release in 2010 [10]. The OpenCV library is cross-platform and free for use, which means that the computer software can be implemented on multiple computing platforms. In other words, the software is platform-indipendent [11]. For instance, a cross-platform application can run on Microsoft Windows, Linux and macOS on either the PowerPC or x86-based Apple Macintosh systems. Currently OpenCV supports a wide variety of programming languages like C++ and Python [12]. OpenCV's major application areas are:

- Face recognition systems;
- Gesture recognition;
- Image segmentation;
- Motion tracking;

- Augmented reality;
- Object identification;
- Mobile robotics.

3.2 OpenCV with Python

OpenCV-Python is the Python API (Application Programming Interface) of OpenCV. It combines the best qualities of OpenCV C++ API and Python language. Python is a general purpose programming language which became very popular in a very short period of time due to its simplicity and code readability. For this reason, Python philosophy can be correctly summarized by the so-called "The Zen of Python" [13], which includes aphorisms such as:

- Beautiful is better than ugly;
- Explicit is better than implicit;
- Simple is better than complex;
- Complex is better than complicated;
- Readability counts.

Nevertheless, Python is slower compared to other languages like C or C++,. However, one of the most important feature of Python is that it can be easily extended with C and C++. This gives the programmer two main advantages:

1. The code is as fast as the original C or C++ code;
2. Python coding is very straight-forward.

3.3 Realtime Computer Vision with OpenCV

As already mentioned in Section 1.1 and Section 1.2 with a special reference to video surveillance systems and pedestrian tracking, machine vision is a very rapidly growing field. Its objective is to understand images, what is happening in front of a camera through a deep knowledge of the core of the image itself. The process

of understanding is then applied to multiple fields such as computer and robotic systems in order to provide people a more informative image. Furthermore, movie production, photography, augmented reality and video surveillance systems are wide areas of its application. OpenCV provides, among other tools, means needed to solve computer-vision problems. It contains a huge amount of low-level image-processing functions and high-level algorithms such as face detection, pedestrian detection, feature matching and tracking [14].

3.3.1 The GPU OpenCV Module

In the early years of processors developing, the simplest way to increase a device performance was to wait for new incoming semiconductor technology to improve, resulting in an increase in the clock speed of the device[1]. This could and still can be achieved employing a single and very fast processor fabricated with the latest material technologies, for example with optical processing elements [3]. As a consequence, when the clock speed increased, all applications were getting faster with no need of modifying them in their inner parts. As transistor got denser and more sophisticated, they have more current leakages resulting in a lower energy efficiency. Therefore, one of the nowadays priority has become to improve the energy efficiency of these systems. One of the proposed solution is to have more transistors per area, thus denser, as suggested above. In this way, two primary ways of optimizing them to good use are:

- Parallelization: this method stands on the statement that is better and faster to create more identical processing units instead of making a single one processing unit faster and more powerful. Parallel processing involves the use of N processors working in parallel that give the possibility of increase the processing sped by a factor N. Thus, to achieve a given processing speed, it is only necessary to increase the number of parallel processors properly;

- Specialization: this other method suggests to build domain-specific hardware accelerators that can perform a particular kind of function more efficiently. Therefore, each distinct kind of function has its own accelerator and just there that process is fully optimized.

[1] The clock speed or clock rate is the speed at which a microprocessor executes instructions. It is generally expressed in Giga Hertz (GHz)

This duality is called heterogeneous parallel computing [14]. High-level computer vision tasks are often composed of a wide variety of peculiar subtasks that can be faster processed on special-purpose hardwares rather than on the main processor. That is the case of the so-called GPU, also known as "Graphics Processing Unit". It is a processor accelerator that can run graphics-related subtasks with more efficiency than a normal CPU. Every laptop or home computer, together with smartphones and tablets, is equipped nowadays with this sort of device.
The first GPUs were fixed-function pipelines specialized for accelerated drawing of shapes on a computer display [14]. Currently, since GPUs are partially programmable, programmers are able to write special or parts of programs that are run by the GPU on purpose making graphics accelerators possible to compute tasks for which they were not originally intended for. Due to this extension of the graphics accelerators, GPU became an useful tool for image processing and computer vision operations.
GPUs are still not quite as flexible as the CPUs. However, since they perform parallelization much more efficiently, an increasing number of non-graphics functions and applications are actually being rewritten using the GPUs code languages, which is different form the more standard CPUs code languages. Nonetheless, a lot of tasks are not parallelized so easily and straight-forward, because they contains a lot of processes where the results of later stages are mostly dependant on the results of earlier stages, resulting in a low efficiency and low velocity of the processes, if made on GPUs rather than on CPUs [14]. Since many high-level tasks are composed by both serial and parallel subtasks, the entire task, case by case, could be accelerated by running some of its steps on the GPU and some others on the CPU. However, this introduces two source of inefficiency:

- Synchronization: when one subtask depends on the result of another subtask;
- The back and forth movement of the data between the CPU and the GPU memories.

Finally, regarding the GPU implementation method in OpenCV, this was added in 2010 resulting in a dramatic acceleration performance. In fact, the GPU module covers a significant part of the library's functionality and new developments are soon to be proposed [14].

3.4 Embedded Systems

An embedded system is a computer system with a specific dedicated function. It is included within a larger and generally more complex mechanical or electrical system, often with real-time computing constraints. Nowadays, embedded systems control many devices in use. As a matter of fact, ninety-eight percent of all microprocessors are manufactured as components of embedded systems. Embedded systems find applications where general-purpose computers would be too costly. Nonetheless, microcontrollers may be programmed to fulfill the same role as a large number of separate components even if their complexity is higher than a traditional system, but most of it is included within the microcontroller itself [15, 16]. The main properties of typically embedded systems when compared with the general-purpose counterparts are:

- Low power consumption;
- Small size;
- Limited operating ranges;
- Low cost.

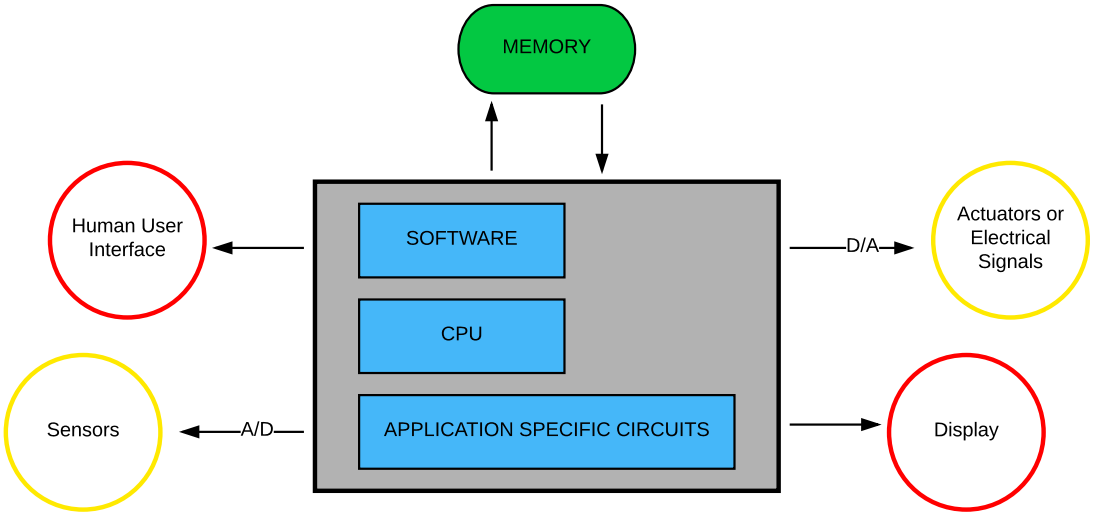


Fig. 3.1 Block diagram of a general-purpose embedded system.

As a consequence, embedded computers come with limited processing resources resulting in a much higher difficulty in programming them and in interacting with them. Modern embedded systems are often based on microcontrollers (CPUs in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip. They are designed to perform specific tasks where the relationship of input and output is defined). However, ordinary microprocessors (it has only the CPU inside. These microprocessors do not have RAM, ROM, and other peripheral on the chip. A system designer has to add them externally to make them functional) are also common, especially in more complex systems. In both case, the unit used may range from a general purpose type to a specialized one in a certain class of computations. Sometimes, even custom designed units are necessary for the application to be as handy as possible. Since embedded systems are dedicated to run very specific tasks, system designers can move widely to achieve a good reliability and high performances reducing the size of the unit and its cost. A single microcontroller chip is generally installed on low-complexity systems, whereas multiple units are taken into consideration for high-complexity applications where peripherals and networks may be mounted inside a large enclosure [16].

3.4.1 Applications

Embedded systems range from portable devices such as digital watches, MP3 players, videogame consoles, digital cameras, GPS receivers and printers to larger stationary installations like traffic lights, factory controllers, video surveillance systems and climate applications. Transportation systems from planes to automobiles increasingly use embedded systems. For instance, brand-new airplanes use inertial guidance systems and GPS receivers. In addition, brushless DC motors, induction motors and DC motors use electronic motor controllers. Automobiles use embedded systems to maximize their efficiency and reduce pollution. Other systems that can benefit from the use of embedded systems are: Anti-Braking System (ABS), Electronic Stability Control (ESC/ESP), Traction Control (TCS) and automatic four-wheel drive.
Embedded systems are also found to be very useful and versatile in safety, medical and life critical applications, as these systems can be easily isolated from being hacked resulting in an higher reliability. Moreover, for safety systems such as fire detection systems, embedded systems can be designed to handle high temperature and continue to operate. In security applications such as video surveillance systems,

the embedded systems can be self-sufficient and they can be able to deal with malfunctioning electrical and communication systems [16].

3.4.2 Characteristics

As cited in Section 3.4, embedded systems are designed to accomplish some specific tasks. Among all the features, the real-time performance constraint is one of the most relevant. Due to safety and usability reasons, real-time performance has to be reached. On the contrary, other applications may have low or no performance requirements resulting in simpler and cheaper systems.
Embedded systems need program instructions to run to accomplish their tasks. The program instructions are called firmware or sometimes embedded softwares. Firmware are executable software that is stored within a read-only memory or on flashmemory chips. They run with limited computer hardware resources such as little memory, small or non-existent keyboard or screen [15, 16].

Processors

Embedded processors can be divided into two main categories. Ordinary microprocessors have only the CPU inside, as mentioned in Section 3.4. Therefore, there is the need of integrating the system with peripherals (i.e. USB and multi media cards such as SD cards) and memory. As a consequence, microprocessors require more support circuitry than microcontrollers. Microcontrollers is very similar to a microprocessor, except that it is designed specifically for use in embedded systems. In fact, they include on-chip peripherals including small amount of RAM and/or ROM memory, resulting in a lower power consumption, smaller size and cost. Since the software depends on the end application, many different basic CPU architectures are used.

3.4.3 Raspberry Pi

Embedded systems can easily be implemented with already made boards such as Raspberry Pi and Arduino, among the most famous and used. These are single-board computers with a single microprocessor or multiple microprocessors, memory, inputs and outputs (I/O). Regarding Raspberry Pi, all models feature a central processing unit (CPU) and an on-chip graphics processing unit (GPU). Secure Digital (SD) cards are used to store the operating system and to program memory. Most boards

have between one and four USB slots, HDMI and other various video output and a 3.5 mm phono jack for audio.

Fig. 3.2 The Raspberry Pi 2 board

The Raspberry Pi Foundation provides Raspbian, a Debian-based Linux distribution for download. Other third party operating systems available via the official website include Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core, RISC OS and specialised media center distributions [17]. It promotes Python and Scratch (Scratch is a visual programming tool allowing the user to create animations and games) as the main programming language, with support for many other languages such as Python Games, Sonic Pi (write code to make music) [18].
In our specific case, the board can be installed on a UGV robot in order to extend the range of action inside the controlled area to make it possible to achieve more on-ground operations after the video analysis. The robot, as already briefly explained in Section 1.3, could be considered as an additional tool to make this specific application even more integrated. Of course, in this case, an object recognition algorithm with dynamic cameras should be taken into account so that a dynamic point of view can be exploited.

Chapter 4

Algorithms and Methods for Video Surveillance

4.1 Introduction

As already introduced in Chapter 1, security surveillance systems have become a solid part of everyday's life and a solid business opportunity, resulting in an increasing number of new methods and suggested solutions. Generally, because video surveillance systems do not provide the possibility of analize in a complete and satisfactorily way all the contents they record, they require the presence of security staff that stands by the monitor. The staff has the task of detecting what is happening in front of the camera(s) or periodically searches for suspicious behaviors and similar or dangerous acts [19]. Therefore, it is desiderable but also challenging to develop surveillance systems that are capable of autonomously detect noticeable and evidently suspicious events. For instance, a useful application could be the detection of unattended bags or pieces of luggage in an airport waiting room for boarding. Of course, an automatic danger alert has to be taken into consideration in order to not have a strict requirement of the security staff presence. This could lead to a faster and more efficient integrated security system.
Motion detection and object tracking are two fundamental task that will be deeply investigated in the further sections. In this Chapter, we define different scenarios that will be analized:

- Moving object/pedestrian detection: a moving object or a walking pedestrian is detected in the region of interest;

- Multiple objects/pedestrians detection: multiple objects or persons detected inside the surveilled area;
- Unattended or abandoned object detection: an object that is left in the surveilled area and it is unattended by its owner;
- Stolen or taken away object detection: an object such as a bag that is left by someone and then picked up by someone else.
- Trespassing: a person or an object that trespasses a sensible or highly surveilled region of interest of the surveilled environment;
- Loitering detection: a person who suspiciously and repetitivily walks in the videosurveilled area.

4.2 Background Modelling

Foreground object detection is the first step in computer vision applications. Received a video data from any source, the aim is to detect and track a single object, multiple objects, a person or more persons in real-time as a foreground object. Additionally, event analysis can be implemented. The foreground object(s) dynamically moves over a background in the camera field and, therefore, can be spotted through different ways. Among them, an efficient way to obtain the above mentioned foreground objects is the background modelling that features different methods, developed over the last decades and still under investigation. Background modelling methods generally share the same scheme: the first frame captured by the camera is memorized to build the so-called background model or background image. As we will see in Section 4.4, the background image is kept updated on a regular basis so as to adapt to the different luminance conditions and geometry settings [20]. Then, it is compared with the current frame captured to detect both moving objects in the camera frame and static objects previously not included in the first frame. In this way, for example, the detection of unattended bags in a subway station can be achieved along with persons detection.

4.2.1 Two Simple Methods for Background Modelling

The two simplest methods for foreground object and motion detection to be detected are the background difference and the frame difference. They both share the scheme

described in Section 4.2. However, as we will notice going through the Subsection, the simplest background difference method just needs to memorize two frames each iteration, whereas the frame difference method needs at least three frames to be memorized.

Background Difference

The background difference, also know as background subtraction, is a widely used method which involves the subtraction of a fixed background frame with an updated current frame. The fixed background frame is generally the first frame of the video that can be considered as the "reference frame" which every following frames point to. The background image or background model may be updated regularly as it should not contain moving objects or shinings due to changing in the light conditions (as explained in Section 4.4) in the region of interest. The background frame is then compared with the second one, the current frame, that keeps on being updated in loop. By converting these frames into binary images and subtracting them as shown in Equation 4.1, the presence of new objects and bodies in the recorded zone can be achieved.

$$P_t(s) = \begin{cases} 1, & \text{if } d(C_{s,t}, B_s) > T \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

In Equation 4.1, $P_t(s)$ represents the value of the pixel s at time t and it is often called the motion mask. d is the subtraction between the current $C_{s,t}$ frame at time t at pixel s and the background model B_s at pixel s. It is important to note that the distance d between the two frames operates, in this case, on gray-scale images as illustrated by Equation 4.2. Finally, T is the set threshold.

$$d = |C_{s,t} - B_s| \tag{4.2}$$

The weakness of this method relies on the fundamental necessity of the use of static cameras. Therefore, the application of mobile surveillance systems through mobile robots would need to be analyzed separately.

Frame Difference

The frame difference involves three subtraction operations. Three frames have to be memorized and updated after every iterations. Naming the frame at the initial recording time "previous", the following frame "current" and the next one "next", two subtraction are initially implemented between the current and the previous frame and between the next and the previous frame again. Their results are then subtracted to get the relative motion of the target object. The weakness of this method relies on the way the movement is spotted. In fact, the background is not memorized. Therefore, if there is no relative movements between two consecutive frames, no difference will be noticed. For instance, if an object is put beneath a waiting room couch on purpose, it will be noticed only in one frame difference. Thus, the suspicious object would be treated as a background element on the consecutive iteration. This could be fallacious for many daily situations. A clear example is shown in Fig. 4.1. In Fig. 4.1a, we can barely see the contours of the moving object. Due to the slight relative movement between two consecutive frames, the contours detected and then showed are almost indiscernible. On the other hand, in Fig. 4.1b, the shape and the nature of the object is more evident, even though we can easily spot some shadow errors.

4.3 Other Approaches to Background Modelling

Other background modelling approaches can be broadly grouped into:

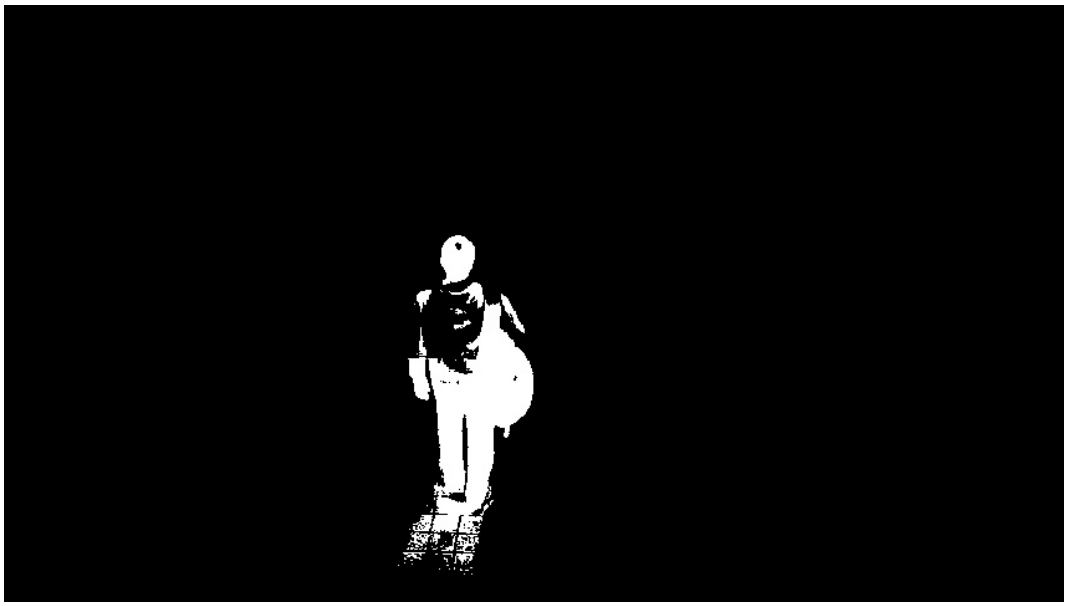
- Pixel-based;
- Region-based;
- Hybrid methods.

or:

- Parametric methods;
- Nonparametric methods.

With the regard of the latter classification, the main difference is: parametric methods have a fixed number of parameters, whereas the number of parameters grows with training data in nonparametric methods. The former ones have the advantage of often being faster to use, but the drawback of making stronger assumptions about the

(a)



(b)

Fig. 4.1 Background images obtained with two basic background modelling methods. (a) Shows a binary background image obtained through the frame difference method. As we can see, the contours are almost imperceptible, (b) Shows the same binary background image, but obtained with the background difference method. Here, the contours are remarkably more visible.

nature of the data distribution. Nonparametric methods, instead, are more versatile, but often computationally complicated and intricate for large datset [21].
From now on we will use the first classification to group the methods together.

4.3.1 Pixel-based Methods

Pixel-based methods can effectively obtain good results when detecting detailed shapes of foreground objects. However, they are commonly affected by defects such as noise, illumination changes and modifications in the background.

4.3.2 Running Gaussian Average

Wren *et al.* in [22] proposed to model the background at each pixel location with a Gaussian probability density function (PDF). Often, the background subtraction problem may result in a PDF-thresholding problem [23]. Therefore, to avoid fitting the function from the beginning at every instant for each new frame and to deal with a low updating rate due to the dynamic changes of the background, an easier function is implemented:

$$\mu_{t,s} = \alpha I_{t,s} + (1-\alpha)\mu_{t-1,s} \tag{4.3}$$

where $I_{t,s}$ is the s pixel's value at time t and $\mu_{t,s}$ is the average calculated at pixel s at time t. α is an empirical weight constant that ranges from 0 to 1. α is generally chosen taking the updating velocity and the image stability into account. For instance, if α is equal to 1, the average at $t = 2$ would not take the average at $t = 1$ into consideration. Therefore, as α becomes closer to unit, the update velocity increases, but, on the other hand, the image stability decreases (refer to Figure 4.2). Although not explicitly stated in [22] as concerned in [20], the other parameter of the Gaussian probability density function, that is the standard deviation $\sigma_{t,s}$, can be calculated similarly.
RGA can be effectively exploited in the OpenCV module in Python by using the function $cv2.accumulateWeighted()$. This function sums up every pixel values. It requires three inputs: the first one is the incoming current frame detected (in this case I_t), the second one is the background reference frame and the third one is the empirical weight constant α.
Since for each pixel only two parameters $(\sigma_{t,s}, \mu_{t,s})$ have to be memorized, the Gaussian average features the good advantage of a low memory requirement. However,

it may happen to have strict constraint on real-time applications. In this case, the computational load has to be considered. Indeed, the two parameteres $(\sigma_{t,s}, \mu_{t,s})$ related to each pixel have to be updated with a rate that is less than the frame rate. Consequently, the computational load decreases but the system may show a slower response to the changing in the background. Some extensions can be made to this method with multiple-channel images (R, G, B) rather than using only gray-scale images, as stated in [23].

4.3.3 Modified Moving Average

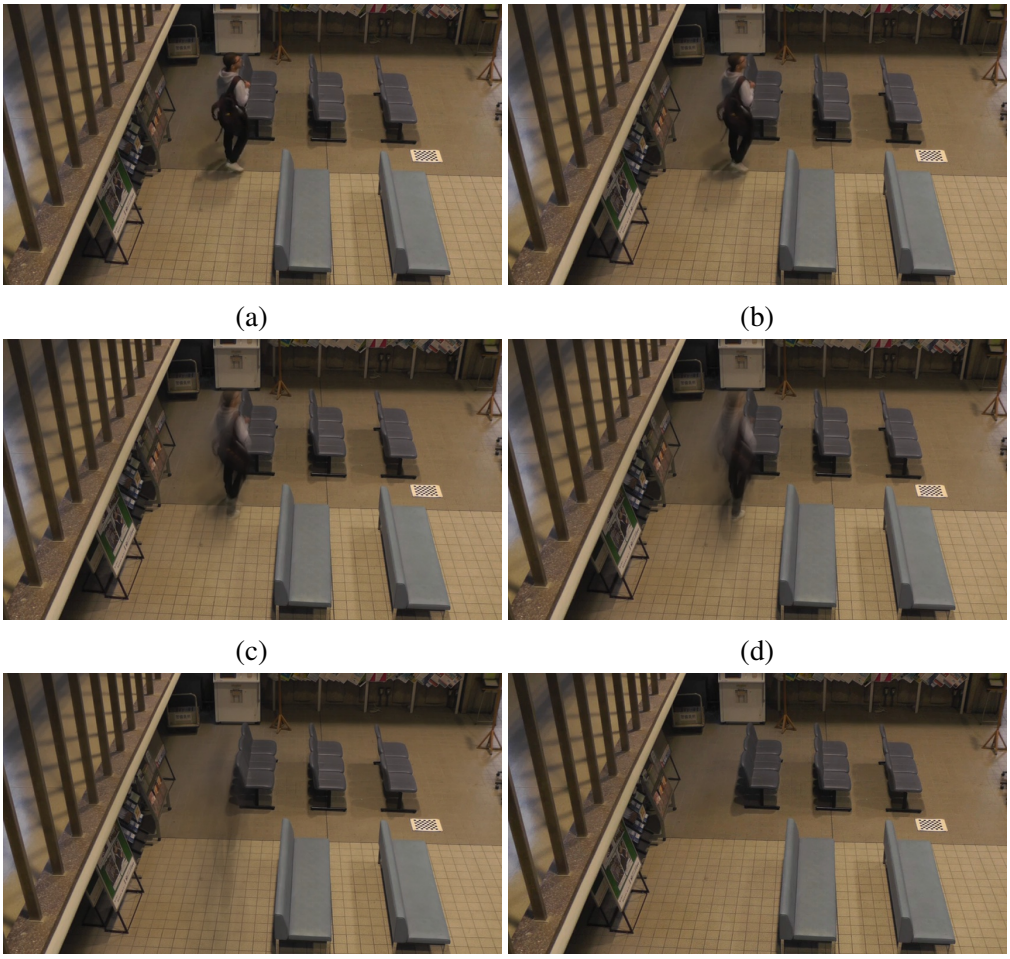
A similar approach to the Running Gaussian average is the one proposed by Kamaraj *et al.* in [2]. The so-called modified moving average (MMA) is used to compute the average of a first set of frames for the initial background model generation. For each pixel (x,y) the corresponding value of the current background model $B_t(x,y)$ is given by Equation 4.4:

$$B(x,y) = B_{t-1}(x,y) + \frac{1}{n}(I_t(x,y) - B_{t-1}(x,y)) \tag{4.4}$$

where n is the video frame number in the video sequence, $B_{t-1}(x,y)$ is the previous background model, $I_t(x,y)$ is the incoming current video frame [2]. The model that will be used for the further analysis, such as detection of stolen objects or pedestrian tracking, is the calculated average over a fixed number of K frames. After the previous part, that we can consider as a training part, the obtained background model $B(x,y)$ is compared to the new incoming frames $F(x,y)$ detected by the static camera. Here, a simple background difference operation is applied and, if the pixel difference is greater than a set threshold T, then it becomes a pixel of a moving object in the region of interest, otherwise is set to zero as a background pixel. The following operation is represented by Equation 4.5:

$$D_t(x,y) = \begin{cases} 1, & \text{if } |F_t(x,y) - B_{t-1}(x,y)| > T \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

where $D_t(x,y)$ is the binary image of the subtraction operation. Since T represents a fixed value for the threshold, it is only suitable and reliable for simple background



(a) (b)

(c) (d)

(e) (f)

Fig. 4.2 Comparison among different learning rates α at the same frame (100) with Running Gaussian average method. (a) α equal to 0.8, (b) α equal to 0.5, (c) α equal to 0.2, (d) α equal to 0.1, (e) α equal to 0.01, (f) α equal to 0.001. The higher is the learning rate, the faster the background model is updated. Oftentimes, α is mostly dependent on how the original scene is affected by gradual changes in illumination, sudden changes in luminance, if the scene is set up to an outdoor environment, and so on.

environments. They should not be affected by any kind of noise, luminance changes, shinings and so on. Hence, a dynamic threshold is proposed in [2], where the possible defects of a background enviroment can affect the threshold itself making it possible to have a clearer background model and an easier moving object detection. The dynamic threshold thus reflects the overall changes in the region of interest detected by the static camera. It takes a small value if there is a small change in the image illumination, whereas it increases accordingly with the presence of noises and illumination changes. This dynamic threshold is proposed:

$$\Delta T = \lambda \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} |F(x,y) - B(x,y)| \tag{4.6}$$

adding ΔT to Equation 4.5, we obtain:

$$D_t(x,y) = \begin{cases} 1, & \text{if } |F_t(x,y) - B_{t-1}(x,y)| > T + \Delta T \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

where *nxm* is the dimension of each image. λ is called by the authors the inhibitory coefficient. Its values vary according to the background environment, but a reference value exists and it is set to 2.

4.3.4 Gaussian Mixture Model

In order to avoid the influence of animated textures caused by waves on the water, moving branches of trees shaken by the wind, Stauffer and Grimson in [24] proposed the Gaussian mixture model (GMM) which models every pixels with a mixture of K Gaussian functions [25]. As stated in [20], the model proposed by Stauffer and Grimson could be defined not only as a background modelling method, but also as an image model as it provides a description for both foreground and background pixels. The probability of observing the input pixel value x at time t being a background pixel is represented by the following equation:

$$P(x_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(x_t, \mu_{i,t}, \sum_{i,t}) \tag{4.8}$$

where $\eta(x_t, \mu_{i,t}), \sum_{i,t}$ is the $i-th$ Gaussian probability distribution function (PDF), $\omega_{i,t}$ is its weight at time t, $\mu_{i,t}$ is the mean value of the $i-th$ Gaussian in the mixture at time t[1], K is the number of Gaussian function and $\sum_{i,t}$ is the covariance matrix of the $i-th$ Gaussian function at time t. Due to computational load reasons, in practical cases, as per [24], the covariance matrix can be assumed as diagonal and K is set to be between 3 and 5.
At each iteration, a new pixel x_t is checked against the K Gaussian distributions, until a match is found. If the pixel value is within 2.5 standard deviation of the distribution, the match is then defined as per Equation 4.9 and the considered pixel is labeled as in background.

$$(x_t - \mu_{i,t}) < 2.5 \cdot \sigma_{i,t} \tag{4.9}$$

In order to recognize a pixel as a foreground or background pixel, a criterion is therefore required. Stauffer and Grimson used the following way: first, all the K distributions are orded based on the ratio between their peak amplitude ω_i and their standard deviation σ_i. The assumption is that the higher and more close-fitting is the distribution function, the more is probable that the pixel belongs to the background. Then, only the R most reliable pixels are chosen as part of the background following Equation 4.10:

$$R = argmin_r(\sum_{k=1}^{r} \omega_k > \tau) \tag{4.10}$$

where τ is an assigned empiric threshold. If none of the K distribution functions matches the current pixel value, the least probable distribution function is replaced with a distribution function centered in x_t with its mean value μ which remains the same, an high variance σ^2 and low weight ω.
In [24], these parameters are only updated for matched components as follows:

$$\omega_{i,t} = (1-\alpha)\omega_{i,t-1} + \alpha \tag{4.11}$$

[1] In a single Gaussian distribution function μ is the peak value.

where α is a user-defined learning rate. The new matched mean value is updated as per Equation 4.12

$$\mu_{i,t} = (1-\rho)\mu_{i,t-1} + \rho \cdot x_{i,t} \tag{4.12}$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(x_t - \mu_t)^T(x_t - \mu_t) \tag{4.13}$$

where ρ is a learning value defined by the following equation:

$$\rho = \alpha\eta(x_t|\mu_k, \sigma_k) \tag{4.14}$$

The Gaussian mixture model has diverse advantages:

- GMM does not have to store a set of parameters in the ingoing process resulting in saving memory space. It only uses the covariance and the peak value μ to measure and work on the pixel leading to the possibility of having different thresholds for each pixel;
- From the previous point, we inherit the absence of a unique global threshold;
- GMM can deal with moving and dynamic backgrounds caused by waving trees, gradual illumination changes.

However, the Gaussian mixture model also has some drawbacks:

- The number of Gaussian probability distribution function K must be predetermined before inizializing the process. As mentioned in the Subsection 4.3.4 and in [24], K is oftentimes set to 3, 4 or 5;
- GMM with only few Gaussian probability distribution functions cannot accurately model a sudden change in the background. It is stricyly related to the user-defined learning rate α. For small learning rate, it is hard to detect sudden changes in the background, whereas for a big learning rate it is very likely that slowly moving objects are absorbed faster into the background;
- A series of frames have to be run without any moving objects or pedestrians in order to train the GMM background model;
- The previous step requires a moderate amount of memory.

4.3.5 Adaptative Gaussian Mixture Model

Since the Gaussian mixture model uses a fixed number of components, Zivkovic proposed an improved algorithm as per [26]. The aim of this work is to adaptively adjust the parameters and the number of components of its predecessor, the GMM. The proposed algorithm can be applied to the given dataset by choosing automatically the number of components for each pixel [25]. Given a new dataset x_t at time t, the algorithm updates constantly the weight $\omega_{i,x,t}$, the mean value $\mu_{i,x,t}$ and the standard deviation $\sigma_{i,x,t}$ as follows:

$$\omega_{i,x,t} = \omega_{i,x,t} + \alpha(o_{i,x,t} - \omega_{i,x,t}) \tag{4.15}$$

$$\mu_{i,x,t} = \mu_{i,x,t} + o_{i,x,t}(\alpha/\omega_{i,x,t})\delta_{i,x,t} \tag{4.16}$$

$$\sigma^2_{i,x,t} = \sigma^2_{i,x,t} + o_{i,x,t}(\alpha/\omega_{i,x,t})(\delta^T_{i,x,t}\delta_{i,x,t} - \sigma^2_{i,x,t}) \tag{4.17}$$

where $\delta_{i,x,t} = x_t - \mu_{i,x,t}$ and α is equal to $1/P$. P is a user-defined time adaption period; thus α defines an exponentially decaying plot which is able to limit the influence of the previous data. It is important to notice that the new parameters are related to the above cited new dataset x_t, as we can see from the subscript x. For a new sample dominion $o_{i,x,t}$ is set to 1 for the close components with largest weight $\omega_{i,x,t}$ and the other $o_{i,x,t}$ is set to 0. For instance, a sample is close to a component if the Mahalanobis distance (Equation 4.18) is less than three.

$$d_m = |C_{s,t} - \mu_{s,t}| \sum_{s,t}^{-1} |C_{s,t} - \mu_{s,t}|^T \tag{4.18}$$

If the components are sorted to have a descending weight $\omega_{i,x,t}$ the Equation 4.10 changes as follows:

$$R = argmin_r(\sum_{k=1}^{r} \omega_k > (1 - c_f)) \tag{4.19}$$

where c_f is a measure of the maximum portion of the data that can belong to foreground objects without influencing the background model. For instance, if an unknown objects come into the region of interest of the camera, it will be treated

as a foreground object. However, if the same object remains stationary inside the same region of interest, the weight ω of the new cluster will increase constantly because the old background is occluded. Therefore, if the objects remains there for enough time, its ω will become larger than c_f and it will be classified as a part of the background [25]. It is reasonable to think that the more c_f increases and comes closer to the unit, the faster is to consider a new incoming object as part of the background environment.

4.3.6 A Self-organizing Background Subtraction Method

In [27], Maddalena and Petrosino proposed a background subtraction model called SOBS based on self-organization of the pixels belonging to the scene by an artificial neural networks [25]. For each pixel, the SOBS method builds a map which consists of a *nxn* vector of weights. Thus, the model of every pixel can be represented as:

$$C = (c_1, c_2, ...c_n) \tag{4.20}$$

which dimensions are n^2. The entire set of weight vectors acts as a background standard. Calling p_t a new incoming pixel which belongs to the $t-th$ frame I_t, if a best matching weight element c_m can be found in the current pixel mode (represented by Equation 4.20), p_t is considered within the background standard. A second case arises when p_t is in the shadow; therefore, it does not belong to the proper object that has to be detected. In this case p_t is still considered within the background model, but it should be not used to update the corresponding vectors. The last case is represented by p_t to be a foreground pixel. When does it occur to have a best matching weight element c_m for the incoming pixel p_t? If the distance from p_t is the smallest among all the weight vectors and the above cited distance is not greater than a fixed threshold [25]. The last situation is expressed by the following Equation:

$$d(c_m, p_t) = min_{i=1,...,n^2} d(c_i, p_t) \leq T \tag{4.21}$$

where n^2 is the size of the current pixel model C. The threshold T can distinguish a background pixel from a foreground pixel by:

$$T = \begin{cases} \varepsilon_1, & \text{if } 0 \leq t \leq K \\ \varepsilon_2, & \text{if } t > K \end{cases} \tag{4.22}$$

In order to obtain a more accurate background model, ε_1 should be greater than ε_2 because high values for ε_1 make it possible to obtain a background model which includes pixel intensity variations, within the first K sequence frames. To sum up, SOBS algorithm initializes each neuron of pixels using the first frame detected by the video source. Then, it accomplishes self-organization of neurons via the weighted average method. Furthermore, the current pixel model will be diffused over the neighboring models C_i after it is updated. If the best match c_m is not found, p_t will be classified as a foreground pixel and the background model will not be updated [25].

The advantages of the SOBS method are a correct adaptation to dynamic backgrounds and to gradual luminosity changes. Moreover, the algorithm achieves a robust detection for different types of videos captured by static cameras. On the contrary, its main disadvantage is the computational time. For each pixel, SOBS builds a neuronal map which dimensions are n^2. Each incoming pixel p_t has to be compared with the weight vector and Equation 4.21 has to be solved, which is time consuming.

4.3.7 Region-based Methods

Differing from pixel-based methods, region-based methods, as the name suggests, exploit relations among the pixels belonging to a certain region. In this way, region-based methods successfully segment the images and detect the foreground objects. In contrast to pixel-based methods, region-based methods can effectively reduce the effect of errors, noises, illumination changes and so on. Nonetheless, the obtained shapes of foreground objects detected are less precise and effective compared to the shapes which can be obtained with a pixel-based method.

4.3.8 Kernel Density Estimator

As stated in [20], an approximation of the background probability distribution function can be given by the histogram of the recent values considered as background values. An example is shown in Fig. 4.3. There, the six Kernel are the red dashed function on the right which represents the histogram data on the left, whereas the Kernel density estimator is the blue solid curve. This kind of approximation may suffer from substantial disadvantages. Due to the step function nature, the histogram may provide a rough modelling of the the probability distribution function whose tails may be fallacious or affected by some errors. In order that such kinds of errors are avoided, Elgammal *et al.* in [28] proposed the introduction of the Kernel density estimation (KDE) on N recent sample of intensity values of the pixel represented by the vector $x_1, x_2, ..., x_N$ to model the background distribution [25]. The probability of the intensity of each pixel x_t at time t is represented by the formula:

$$P(x_t) = \frac{1}{N} \sum_{i=1}^{N} \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi\sigma_j^2)}} e^{-\frac{1}{2}\frac{(x_{t,j}\ x_{t,i})^2}{\sigma_t^2}} \tag{4.23}$$

where N is the number of recent samples considered, d is the number of channels (generally 3 for RGB images). σ represents the Kernel function bandwidth [2] for each color channel and it is estimated by the Equation 4.24:

$$\sigma = \frac{MAD}{0.68\ \sqrt{2}} \tag{4.24}$$

where m is the median absolute deviation (MAD), a measure of statistical dispersion. The median absolute deviation is calculated over the sample for consecutive values of the pixel [28]. If $P(x_t) > T$, then the pixel x_t will be straightforwardly classified as a background pixel. The threshold T is a global value over all images. Furthermore, the algorithm presented in [28] distinguishes between two ways to generate the background model: a long term model and a short term model to achieve better update decisions and make the model fastly adapt to changes [25].
One of the main advantage of the KDE method is that it can easily deal with

[2] The bandwidth of the Kernel is a theoretically or experimentally estimated value which exhibits a strong influence on the resulting estimate.

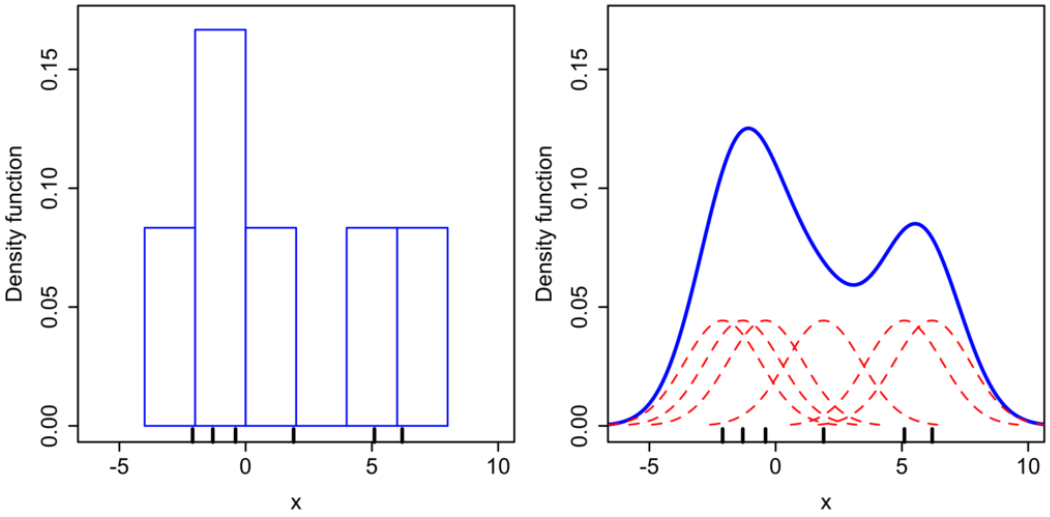


Fig. 4.3 A Kernel density function and the histogram constructed with the same data (source: boundless.com).

multimodal and changing backgrounds by quickly including newly observed values in the pixel model. On the other hand, the main drawback of the KDE method relies on the fact that it has to keep N frames in memory. Consequently, when N is large, the computational load and the elapsed time increase.

4.3.9 Hybrid Methods

Hybrid methods are a combination of both region-based methods and pixel-based methods. They can achieve better background modelling and also deal with sudden changes of illumination levels and dynamic backgrounds, as stated in [29]. Although

Background modelling methods		
Approaches	Pixel-based	Region-based
Parametric	RGA	
	MMA	
	GMM	
	AGMM	
Nonparametric	SOBS	KDE

Table 4.1 Background modelling methods

hybrid methods can efficiently retrieve foreground objects from the background, their computational cost and complexity is considerably high [25].

4.4 Challenges in Background Modelling

All the background modelling algorithms that we have deeply analyzed so far, but not only them, have to deal with different challenges deriving from the quality and the nature of the dataset which the system uses. We refer to the most common challenges clearly defined by Toyama *et al.* in [30]. As stated in the latter work, the difficult part of background modelling is not the modelling itself, but the maintenance of a background model. Indeed, an ideal background maintenance is able to avoid the following problems:

- Dynamic background objects: a background is composed by a broad variety of objects. Sometimes, they can be moved. Hence, the moved objects should not be considered as foreground elements. For instance, moving tree branches and swinging traffic lights;
- Gradual illumination changes: over the day the illumination level varies widely. A good background maintenance should take this issue into account and it should adapt to the gradual changes of the appearance of the environment;
- Sudden light switches: they generally occur in indoor environments when, for instance, a light is turned on or off. Due to fast moving clouds, they may also occur in outdoor environments and strongly influence the appearance of the background;
- Camouflage: some objects may poorly differ from the real appearance of the background, making a correct classification ardous. This is really important in surveillance system applications;
- Video noise: there is always some noise in a video signal. Background maintainance has to deal with such signals. The noise may come from a sensor noise or bad compression rate by the video source.
- Shadows: foreground objects often cast shadows which appear different from the modeled background.

In our work, we try to deeply investigate and further solve these challenges in order to obtain a robust pedestrian and object detection algorithm.

4.5 Moving Object and Pedestrian Detection

As already cited in Section 4.1, moving object and pedestrian detection is the simplest application of video surveillance systems. One or multiple static cameras take a certain dataset defined by an input video, elaborate it, and eventually provide an output. The output can be represented by the processed video showed on a screen where the targets are bounded by rectangular blobs which move following the targets. Generally, most of the surveillance video starts with a certain number of empty frames during which foreground objects are not present in the region of interest. Due to this, a background training session can be easily achieved in order to have a solid and consistent background model in which noise, shadows, illumination changes can be either partially or totally polished. Based on this, we can further analyze the moving object detection. Here, we propose again Equation 4.3, with a different α learning rate:

$$BG_t(x,y) = (1-\alpha_{fast})BG_{t-1} + \alpha_{fast}C_t(x,y) \tag{4.25}$$

where $BG_t(x,y)$ is the background frame at time t at a certain pixel location (x,y), BG_{t-1} is the previous background at time $t-1$, $C_t(x,y)$ is the current frame at time t, and α_{fast} is the learning rate. The reason why the weight constant is named "alpha fast" is because the background model is updated quite frequently compared to the other background model which will be used for further analysis, such as abandoned object detection (Section 4.7).

Image Processing Once we retrieve the background model, we can obtain the difference of the background and the current frame by the background subtraction technique, obtaining a mere and simple pixel difference. However, before applying the background subtraction, the current color frame and the background are converted into a gray-scale image. The resulting gray-scale current frame is then filtered using the Median blurring filter (Fig. 2.7) with a Kernel size of $5x5$. Then, the background subtraction can be accomplished and the obtained pixel difference is filtered again to

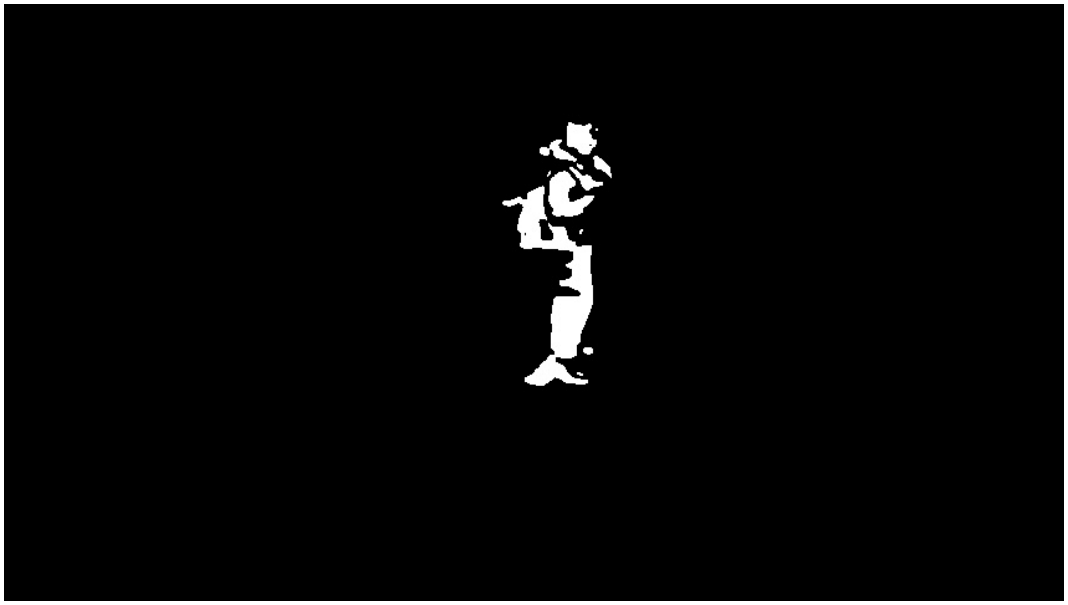


Fig. 4.4 Binary result of the background subtraction between the fast-update background model and the current colored frame.

achieve smoother shapes for the foreground object analysis. Finally, the result of the subtraction is thresholded and the resulting image, shown in Fig. 4.4, is obtained.
A further step inside the image processing phase is made: the obtained result of thresholding is morphologically treated iterating once the erosion transformaton and five times the dilation. Because of the high-segmented shape previously obtained, we want to fill the small areas shown in Fig. 4.4 as much as possibly provided by those processes (Subsection 2.4.3) in order to merge all the components of the silhoutte together. The result is displayed in Fig. 4.5. Additionally, antoher final step may be considered. Due to low contrast and low exposure of our specific dataset, the obtained binary image is still partially imprecise and does not fully represent the real silhouette of the object or pedestrian that has to be extracted from the image source. As a consequence, to make it as smooth as possible, more studies could be made. For instance, histogram equalization may be applied to make the object extraction more reliable and efficient. Nevertheless, we consider this obtained difference precise enough for the purpose of object and pedestrian detection.

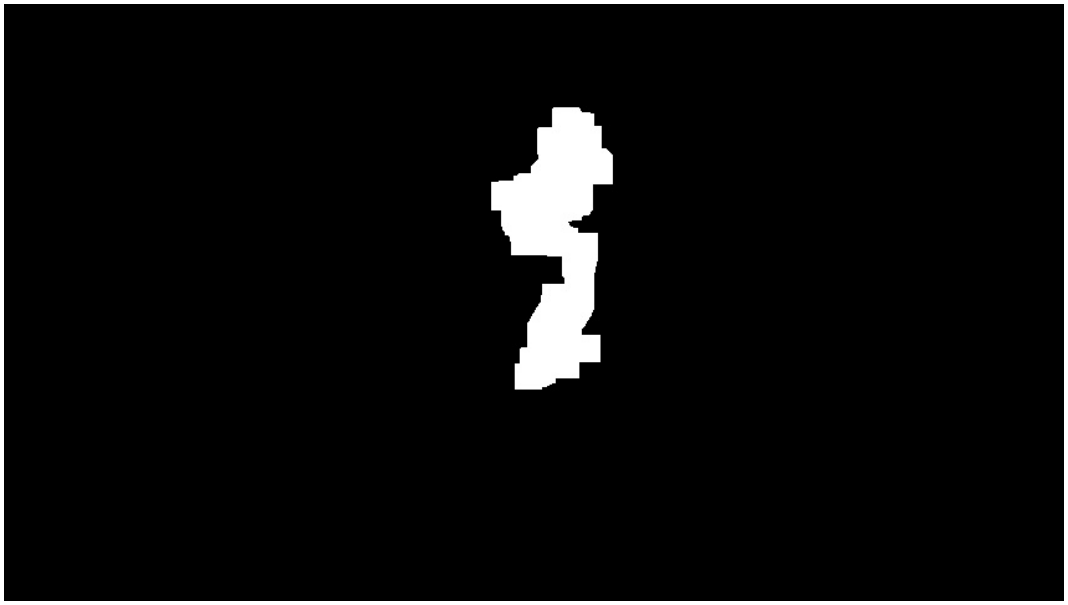


Fig. 4.5 Background subtraction with a fast-update background model after the process of erosion and dilation.

4.5.1 The Waterfall Algorithm

After the image processing phase, object detection can be achieved. Implementing the OpenCV function $cv2.findContours()$ allows us to find all the contours in the obtained binary image. It follows that if the length of the Python multiple-dimension list is zero, no contours are found. Consequently, there are no coordinates to show. Otherwise the algorithm works as follow:

Process 1 When the length of contour list is greater than zero, we first calculate the area of each contour with the OpenCV function $cv2.contourArea()$. Our aim is to apply a lower threshold which discards all the contours with too small areas, in order to get rid of those contours which could be defined by defects or noise errors. Successively, the OpenCV function $cv2.boundingRect()$ is applied and the coordinates of the bounding boxes are obtained. Indeed, it creates a list of coordinates for each contour detected. More precisely, the function gives four outputs:

- The x coordinate of the upper left corner;
- The y coordinate of the upper left corner;
- The width of the rectangle;

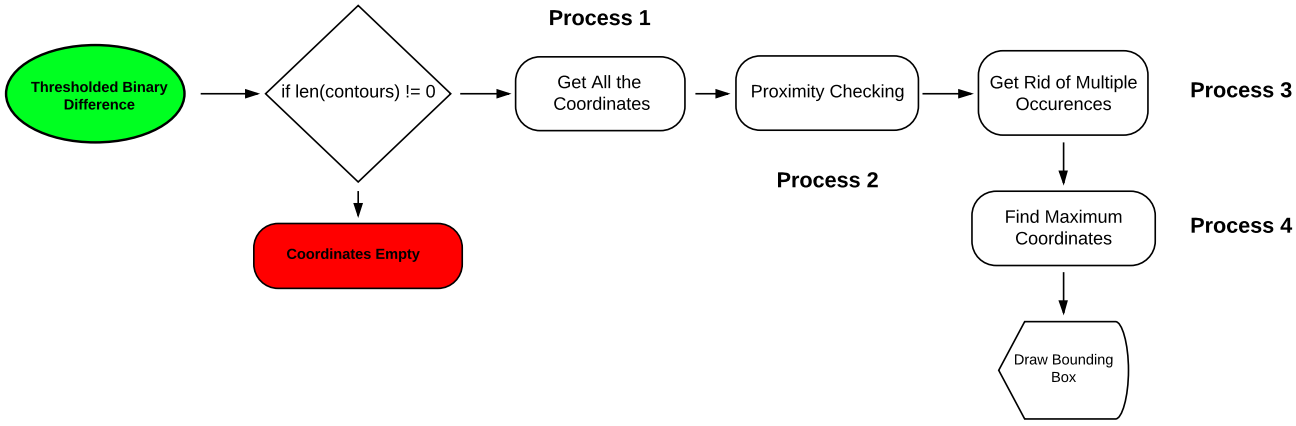


Fig. 4.6 Waterfall algorithm flowchart.

- The height of the rectangle.

Process 2 Given the coordinates to draw the bounding rectangle around the lineation of the foreground object, we made another step to make it possible for a more precise detection to happen. Taking the list of all the rectangles obtained in the previous step into account, we build a list of sets (displayed in Python by the mean of curly brackets, as per Equation 4.27) checking all the N rectangles against each other within a set threshold, named *delta* in Fig. 4.7. Therefore, rectangle 1 will be compared to rectangle 2 to N. In the same way, rectangle 2 will be compared to rectangle 3 to N, an so on. Afterwards, the resulting list of sets will contain the indeces of those rectangles which passed the proximity check, as the example in Equation 4.27. What happens to the ones that do not pass the proximity test? We simply consider them as standing-alone shapes. Indeed, it is possible that a certain shape is too distant from the others to be listed as a standing-alone rectangle. That is the case of multiple object detection.
In this specific example, following the criterion we created, rectangle 1 intersects with rectangle 2. At the same time, rectangle 2 is considered in proximity of rectangle 3.

$$[\{1,2,3,4,5,6,7,8,\ldots,N\}] \tag{4.26}$$

$$[\{1,2\},\{2,3\},\{7\},\{4,5\},\{3,6\},\{8\},\ldots] \tag{4.27}$$

Process 3 manipulates the list of sets of nearby rectangles in order to get rid of those shapes with more than one occurence.

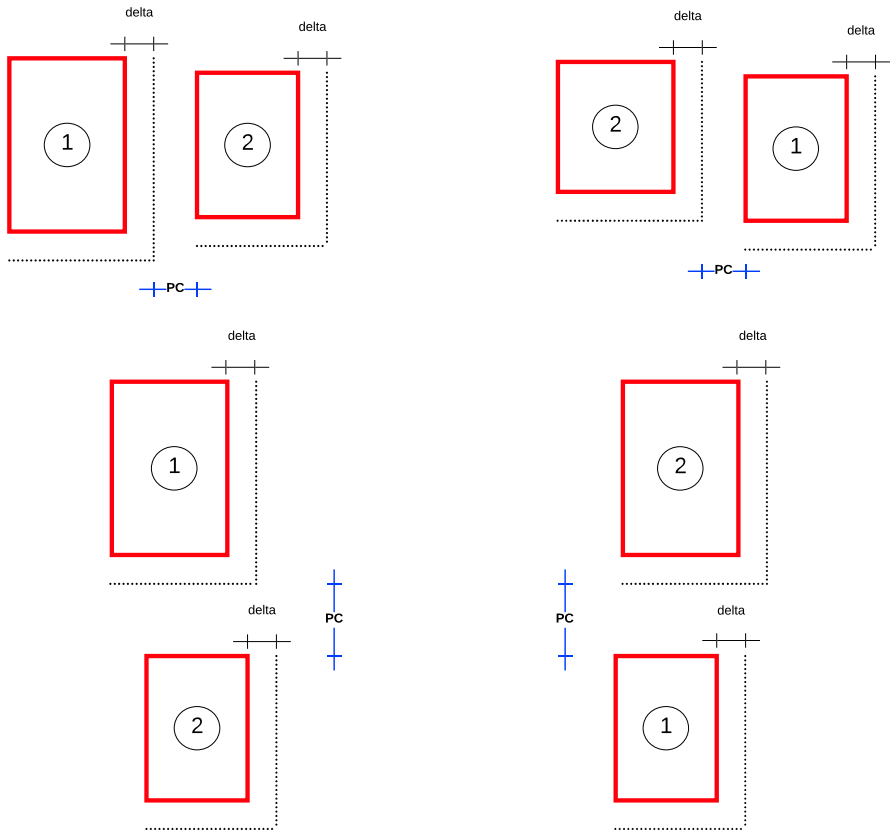


Fig. 4.7 Proximity check criterion.

As we can see in Fig. 4.7, we considered four cases, which cover all the possible scenarios:

1. The distance between the most right side of the first rectangle, *delta* included, and the most left side of the second rectangle;
2. The distance between the most right side of the second rectangle, *delta* included, and the most left side of the first rectangle;
3. The distance between the most bottom side of the first rectangle, *delta* included, and the most upper side of the second rectangle;

(a) (b)

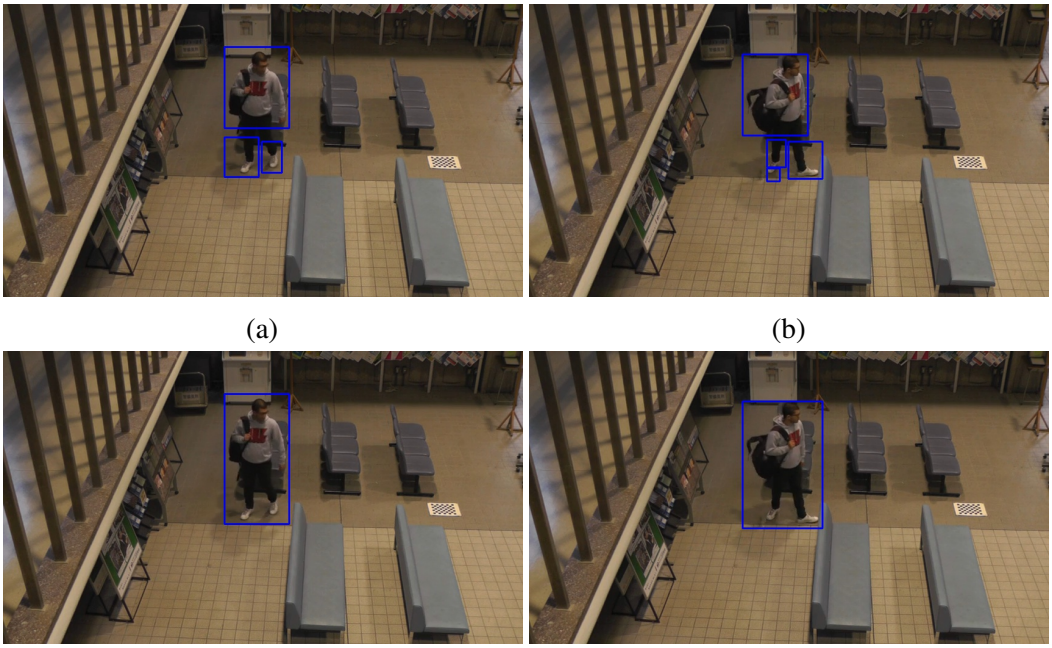
Fig. 4.8 The binarized foreground object and its colored counterpart, with a visible rectangle error. (a) Displays the binarized foreground object through thresholding. We can see some considerable different contours resulting in different areas detected, (b) Shows two bounding boxes around the single foreground object, but composed by different contours.

4. The distance between the most bottom side of the second rectangle, *delta* included, and the most upper side of the first rectangle;

Process 3 The obtained list of sets cannot be printed as it is, otherwise there would be several overwritten blobs. Phase 3 takes Equation 4.27 and exploits the Numpy Array function $.isdisjoint()$ to get rid of multiple recurrences. Being only applicable to sets, this function perfectly fits with the inner nature of them. It returns $True$ if two sets does not have elements in common, whereas it returns $False$ if the sets share some elements. Based on this, we can achieve the following result with the regard of the list of sets represented by Equation 4.27:

$$[\{1,2,3,6\}, \{7\}, \{4,5\}, \{8\}, ...] \tag{4.28}$$

Process 4 The last step provides the four coordinates needed to draw the active blobs around the detected shapes. The new list of arranged sets is scrolled and the maximum coordinates, together with the width and the height, are found. This passage is fundamental because it oftentimes avoids to have sides or part of the bounding boxes crossing the actual shape of the detected object or pedestrian. However, due to the discrete quality of the datased used and the *delta* used in Process 2, it rarely happens that the same shape is bounded by more than one rectangle, as we still have some frames where the morphological transformations have not been efficient enough to merge different segments together, as shown in Fig. 4.8a. Insted, in Fig.



(a) (b)

(c) (d)

Fig. 4.9 Result of the proposed algorithm to merge the sorrounding blobs. (a), (b) Display frames 228 and 241 without the algorithm applied, (c), and (d) Show the same frames with the algorithm applied.

4.9, it is displayed the comparison between the same frames without the Waterfall algorithm and with it.
The final result is displayed in Fig. 4.10. A simplified flowchart of the explained algorithm is shown in Fig. 4.11.

4.6 Multiple Moving Object and Pedestrian Detection

Multiple object and pedestrian detection exploits the same algorithm of a single object and pedestrian detection, explained in Section 4.5. As already stated, the only difference concerns how the rectangles are grouped together. In the previous case, since only one object or pedestrian had to be detected, all the rectangles were considered as nearby by the Waterfall algorithm. On the other hand, there will be two or more groups of rectangles in this case, as shown in Fig. 4.13b. The flowchart shown in Fig. 4.12 is exactly identical to the one in Fig. 4.11, except for the lower

(a)

(b)

Fig. 4.10 The binarized foreground object and its colored counterpart. (a) Shows the foreground object after being thresholded, and after the process of erosion and dilation which gave the image this particular shape, moving on the black background model obtained with RGA method, (b) Shows the drawn bounding box aroun the foreground object in the current color image at frame 147.

part where the multiple bounding boxes are drawn on the current color frame by the $cv2.rectangle()$ function depending on the number of the sets contained in the list.

4.7 Unattended or Abandoned Object Detection

Unattended or abandoned object detection expects a sharp change in the consideration of the background model. In fact, no matter how many training frames are provided to the background model, also strongly depending on the learning rate α, the pixels belonging to temporary unattended object may be mistaken as part of the background

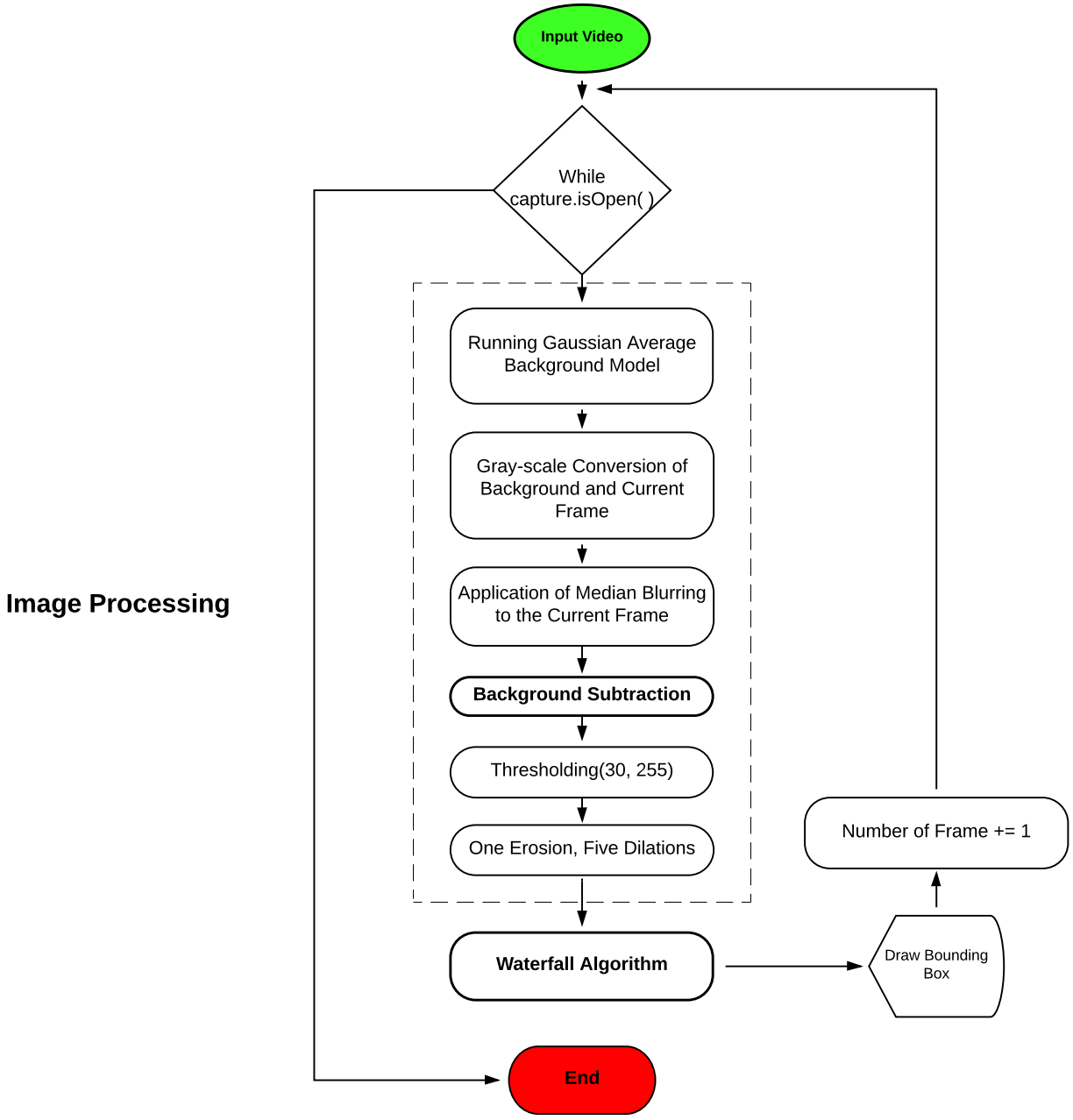


Fig. 4.11 Flowchart of the simple object and pedestrian detection.

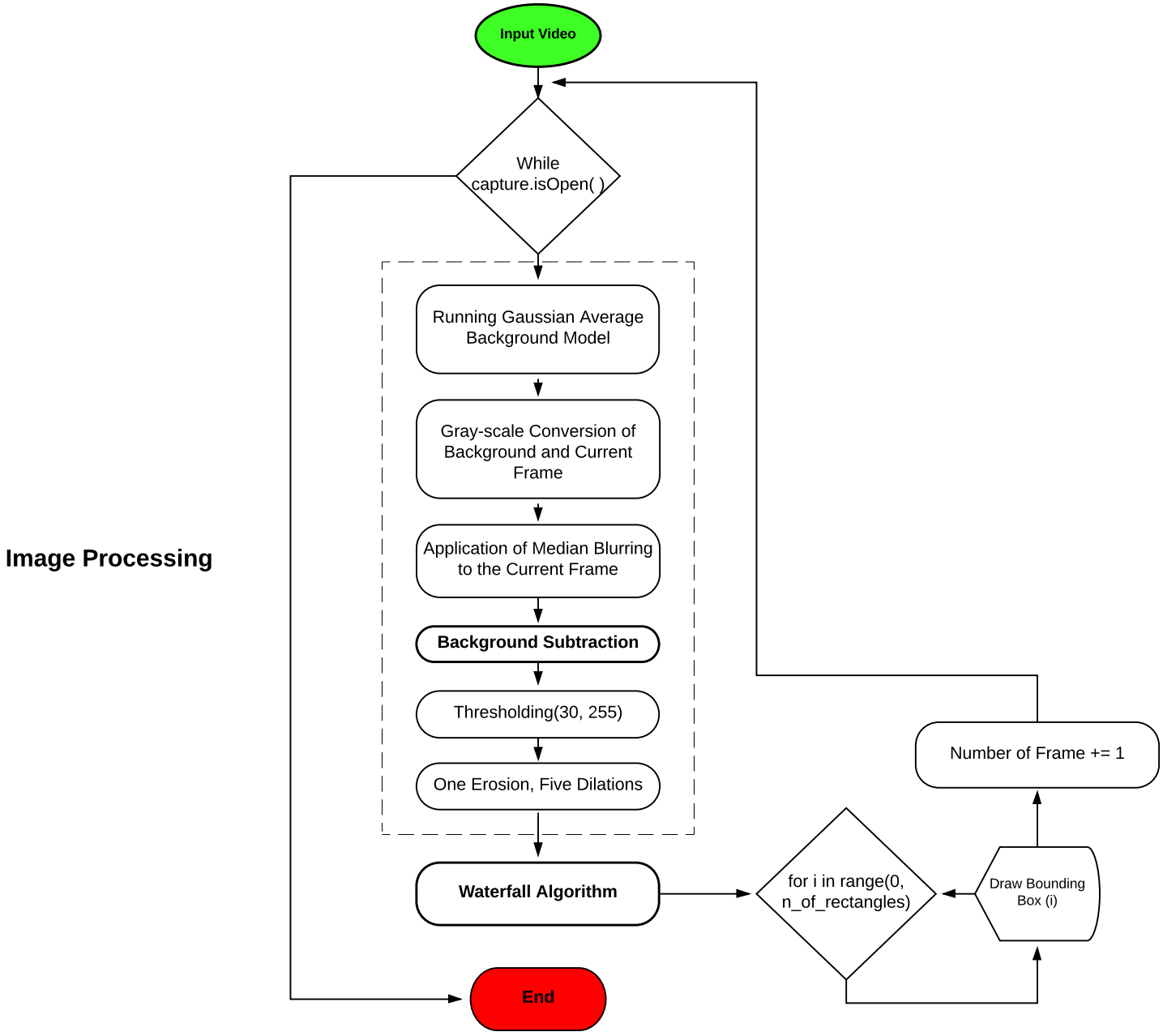
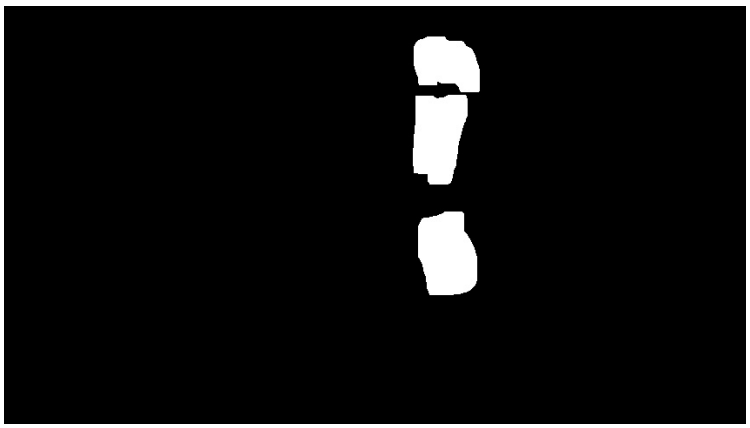


Fig. 4.12 Particular of the flowchart of the multiple object/pedestrian detection.



(a)

(b)

Fig. 4.13 The binarized foreground object and its colored counterpart, with two rectangles. (a) Shows two foreground objects. Now, the objects are clearly multiple, (b) Shows the couple of rectangular blobs around the two object in the current color image at frame 377.

or as a part of moving object [31]. Therefore, it is our belief that it is strongly necessary not to be limited to a single background model, but to build at least two of them.

Process 1 In our specific case, we consider two background models having different learning rates α_{fast} and α_{slow} as per Equations 4.29, 4.30. One has a "fast" weight/learning rate, so that the background model is fastly updated to the changing environment or to the moving of a pedestrian or an object. The second one has a "slow" weight/learning rate which allows to have a slowly updated background model. A wide comparison among different learning rate is clearly shown in Fig. 4.14.

(a) (b)

Fig. 4.14 Two dynamically updating background. (a) Displays a "fast-updating" background model with a learning rate of 0.003 at frame 517, (b) Shows a "slow-updating" background model with a learning rate of 0.0001 at frame 517.

The aim of providing the algorithm of two background models is not only related to the necessity of getting rid over time of defects in the reference frames, but also to provide a solid instrument for the analysis of unattended object detection and taken away object recognition.

$$BG_{t,fast}(x,y) = (1-\alpha_{fast})BG_{t-1,fast} + \alpha_{fast}C_t(x,y) \tag{4.29}$$

$$BG_{t,slow}(x,y) = (1-\alpha_{slow})BG_{t-1,slow} + \alpha_{slow}C_t(x,y) \tag{4.30}$$

Having the two background models available, we can easily state whether an object belongs to the original scene or not. Indeed, if the foreground mask changes rapidly, it will not affect the background model, whereas if an object is brought into the scene and left in a static position for more than a certain amount of time, it will merge gradually into the background and then considered as a new part of the original scene. How can we achieve this? As mentioned several times, the background model cannot remain static forever. In that case, for instance, considering our region of interest, if a couch is added to the waiting room, it will be always labeled as a moving object and then bounded by an active blob in our algorithm. On the contrary, if the background model(s) is currently updated, the couch will be described by the algorithm as part of the scene after a set number of frames, depending on the learning rate α. A clearer example is shown in Fig. 4.14a, where the backpack is gradually appearing into the background image. As soon as its shape is fully defined, it will be a part of the original scene. On the other hand, in Fig. 4.14b, the backpack

has not appeared yet because of the low learning rate α_{slow}.
Applying two background models has the advantage of being able to be customized by the user. As a consequence, the update time interval could be chosen depending on the circumstances and on the environment needs.

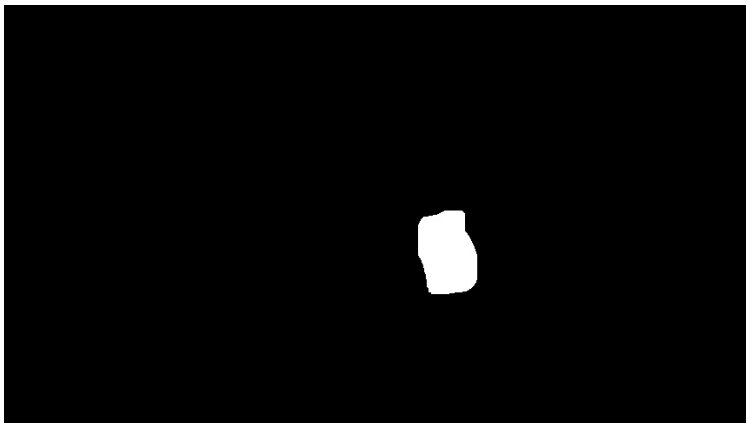
Process 2 To detect unattended or abandoned object is necessary to implement another background subtraction but, this time, between the two background models. In this way, when an object is brought and left static into the scene (Fig. 4.15a), it will slowly disappear in the difference between $BG_{fast}(x,y)$ (the "fast-updating" background model) and the current frame $C(x,y)$. At the same time, it will slowly appear in the difference between $BG_{fast}(x,y)$ and $BG_{slow}(x,y)$ (Fig. 4.15c), since it has not appeared yet in the difference between $BG_{slow}(x,y)$ (the "slow-updating" background model) and the current frame $C(x,y)$.

$$BG_{difference,unattended}(x,y) = BG_{fast}(x,y) - BG_{slow}(x,y) \qquad (4.31)$$

Process 3 As shown in Equation 4.32, the more the difference between the two background models increases with time, the easier is that $|BG_{fast}(x,y) - BG_{slow}(x,y)|$ goes beyond the set threshold and then the object is considered as unattended (as shown in Fig. 4.15b). It is absolutely necessary to clarify that T in Equation 4.32 is not a time threshold, but a pixel threshold. It represents the pixel threshold operation for the two background models difference. In other words, if the $cv2.findContours()$ function finds contours in the $BG_{difference,unattended}(x,y)$ source, it means that $|BG_{fast}(x,y) - BG_{slow}(x,y)|$ has exceeded the pixel trheshold T, set to 30 in this specific case. Additionally, once $|BG_{fast}(x,y) - BG_{slow}(x,y)|$ has exceeded the pixel threshold, if the $cv2.findContours()$ function finds contours in the image source for more than a determined amount of frames, then the object is considered abandoned. Finally, it is important to remember that the time threshold beyond which the static object is described as a part of the original scene is only depending on the user, which is able to define, according to our developed algorithm, the learning rates of the background models.

(a)

(b)



(c)

Fig. 4.15 The process of unattended or abandoned object detection. (a) A backpack is left unattended in the region of interest at frame 736, (b) After a set amount of frames (specifically 360 frames, which represents the set time threshold in the proposed algorithm) the backpack is labeled as part of the original scenario, (c) The binary difference between the two dynamically updating background models.

(a) (b)

Fig. 4.16 The object labeled as unattended or abandoned is cropped: (a) With color information, (b) In gray-scale.

$$BG_{difference,unattended}(x,y) = \begin{cases} 1, & \text{if } |BG_{fast}(x,y) - BG_{slow}(x,y)| > T \\ 0, & \text{otherwise} \end{cases} \tag{4.32}$$

Process 4 What if a pedestrian stands without moving for a long time? the proposed algorithm cannot distinguish precisely a pedestrian from a mere object; thus, the algorithm may confuse the pedestrian with an object, and viceversa. Due to this, the area threshold, which is represented by two simple boundaries, displayed as δ_{lower} and δ_{upper} in the flowchart shown in Fig. 4.17, has to be set according to the application. More precisely, the area threshold is considered in order not to take into account neither too small objects (such as an unpolished defect) nor too big objects (such as a pedestrian, indeed). This leads to the fact that some actual big (or really small) objects may not be considered in the detection. As a consequence, a deeper understanding of the application scenario has to be sorted out. To give a clear example, if the video surveillance system is applied for illegally parked vehicle detection, it is clear that in such environment the biggest object that can be detected is a vehicle and not a pedestrian; therefore, the threshold should only have a lower boundary to get rid of waiting pedestrian to cross the street, etc.

Process 5 In addition, a time threshold is considered. In fact, if the $cv2.findContours()$ function finds contours in the $BG_{difference,unattended}(x,y)$ for

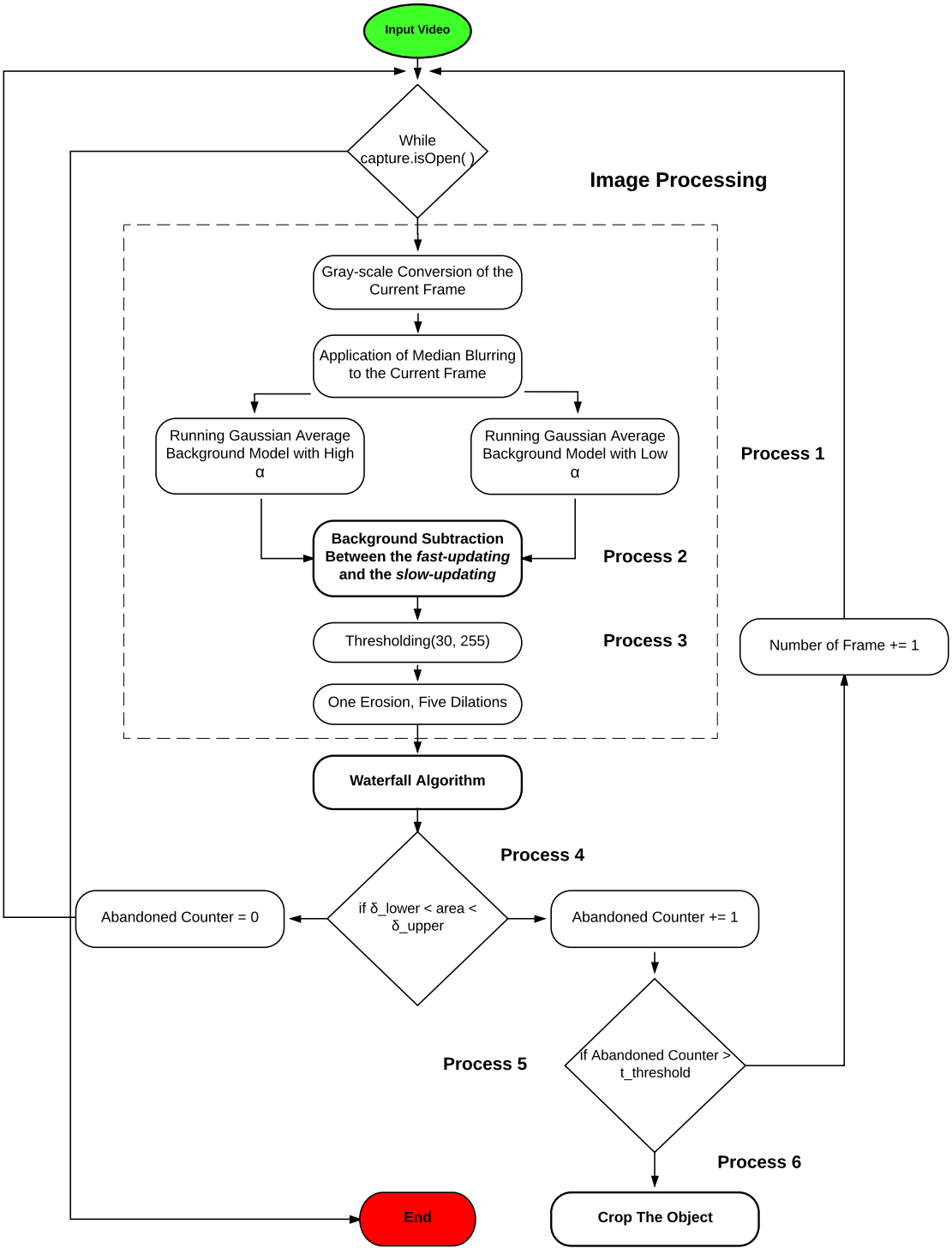


Fig. 4.17 Flowchart of abandoned or unattended object detection.

few frames, it does not mean that an actual unattended object has been detected. This process has to be elaborated upon a prolonged time. For instance, if the video surveillance system is applied in an airport waiting room, where it is likely to find passengers asleep or motionless, the time threshold to detect an unattended object may be bigger compared to applying the same system in a subway platform environment where it is quite rare to find a motionless person for more than a dozen of seconds.

Process 6 If the time threshold is reached, around the unattended object a bounding box is drawn and the object shape is cropped. We considered cropping the image both with color information and in gray-scale the more suitable way to gather further information in a successive phase.

4.8 Stolen or Taken Away Object Detection

Stolen or taken away object detection is a really complex and critical issue in the majority of video surveillance systems. As in Section 4.7, two background models are implemented in order to have a dynamically changing background which deals with background challenges (Section 4.4), and moving objects inside the videosurveilled area.
In this specific scenario, we assume that the involved object is already present inside the area; hence, the criterion which states whether the object has been stolen/taken away or not, will be exaclty the same we applied to the abandoned object detection. Indeed, as proposed again in Equation 4.33, once the object is taken away from the area in which it was previously static, the difference between the two background models, $|BG_{fast}(x,y) - BG_{slow}(x,y)|$, will gradually shows its absence.

$$BG_{difference,unattended}(x,y) = BG_{fast}(x,y) - BG_{slow}(x,y) \qquad (4.33)$$

The object is then considered as stolen or taken away after a certain time threshold, again taken into account. In addition, an area threshold can be smartly considered as done in the previous Section. Consequently, not every static and standing-alone shapes will be labeled as stolen after the set time threshold. In fact, it is clear that it is high likely that a bounding box which would encircle a pedestrian is endowed

```
input : Stolen Counter, Fast-updating background model, slow-updating
        background model, Current Color Frame
output : Stolen Counter

Time Threshold;

while Videocapture.isOpen() do
    get Rectangle coordinates;
    apply the Waterfall algorithm;
    for each (x, y, w, h) in coordinates do
        area = w x h;
        if δ_lower < area < δ_upper then
            Stolen Counter += 1
        end
        else
            Stolen Counter = 0
        end
    end
    if Stolen Counter is Time Threshold then
        print('Stolen or Taken Away Object Detected')
    end
end
```

Algorithm 1: Stolen Object Detection Pseudocode

with a remarkable larger area than a bounding box that sorrounds an object. A clear example is displayed in Fig. 4.18.

To make the algorithm more robust and reliable, the use of color channels can be implemented. Indeed, with color information, we can state more accurately if an object is stolen or taken away from a background scenario. Given a static object and its location inside the area, by exploiting color information, we can calculate the values of each pixel making it possible to know mathematically the color trend within the image. Consequently, when the object is taken away and the color value trends change in a certain region of the considered area, we can state, depending on the resolution of the image, approximately where the object has been stolen. Of course, even this method has a key drawback: the color differences between the object and the background scenario have to be quite noticeable mathematically speaking. If the object is, for instance, a dark gray backpack and it is abandoned on a dark asphalt

road, it may be quite intricate and problematic to detect whether the object was really stolen or not only considering the color difference in the trend.

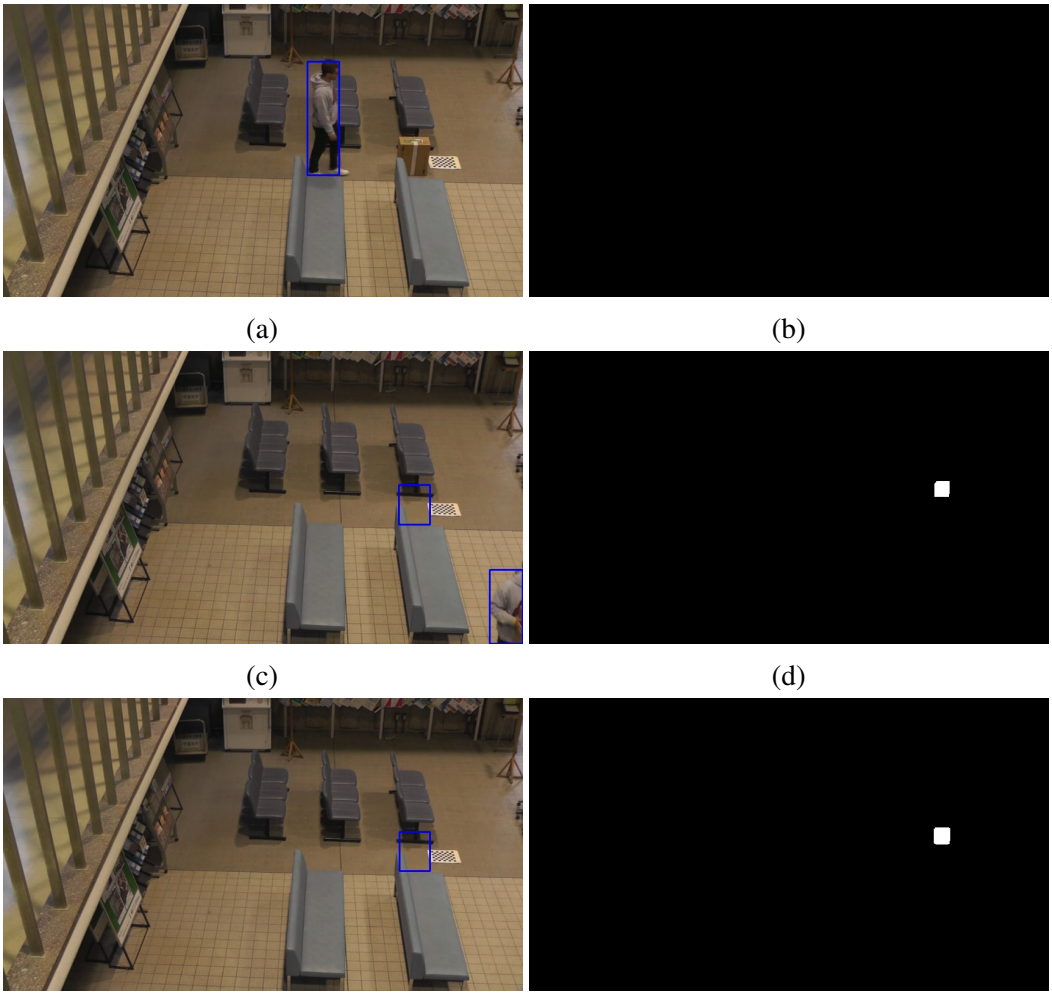


Fig. 4.18 The process of stolen or taken away object detection. (a) The original scene, (b) The binary difference between the two dynamically updating background models at the same frame (427) of the original scene, (c) The carboard box is stolen and the "thief" is going away from the videosurveilled area, (d) The absence of the stolen object is starting to appear in the difference between the two backgrounds. This process will deeply depend on the two learning rates α_{fast} and α_{slow}, and their difference. For instance, if α_{fast} is dramatically bigger than α_{slow}, the absence of a taken away object will be noticed almost immediately, (e) The object is stolen and it is pointed out in the current color image at frame 1470, (f) The absence of the stolen object is fully defined in the difference between the two updating backgrounds.

4.9 Trespassing Detection

Trespassing detection is a fundamental feature in every video surveillance systems. The applied system, in fact, can be used in several high-sensibility areas such as military zones, airports, customs, production plants and so on. In the cited scenarios, handling an automatic system which is able to autonomously warn in the case of a trespassing object or pedestrian, is vitally significant. In this way, security staff does not have to stand in front of the monitors for prolonged periods of time because a warning alarm can be easily implemented. It can be achieved either visually or acoustically.
In the proposed algorithm, trespassing detection basically consists in tracking one or multiple objects inside the videosurveilled area considering the centroids of each active rectangles. If any of those centroid crosses a specific region, whose actual counterpart may be represented by a fenced area or a barricade, the warning alert immediately pops up.
In Fig. 4.20, a single iteration on the videoset is displayed. The core part is composed by Process 1, Process 2 and Process 3.

Process 1 Right after moving object detection, the single or multiple centroids are saved in each frame. This is straighforwardly accomplished considering the output of the function $cv2.boundingRect()$.

Process 2 Then, it is checked if the previously saved centroid(s) crosses the specified area. In our specific case the area considered corresponds to the 40% of the entire width of the video frame. This can be seen in Fig. 4.19.

Process 3 If the previous if-statement is true, an alarm is triggered, as displayed in Fig 4.19b and 4.19c.

$$AlarmTriggered = \begin{cases} True, & \text{if } X\,centroid\,coordinate \geq VideoWidth \cdot 0.6 \\ False, & \text{otherwise} \end{cases} \tag{4.34}$$

Of course, if the high-sensibility area differs from the one we considered, the condition to have the alarm triggered will be slightly different. For instance, if we

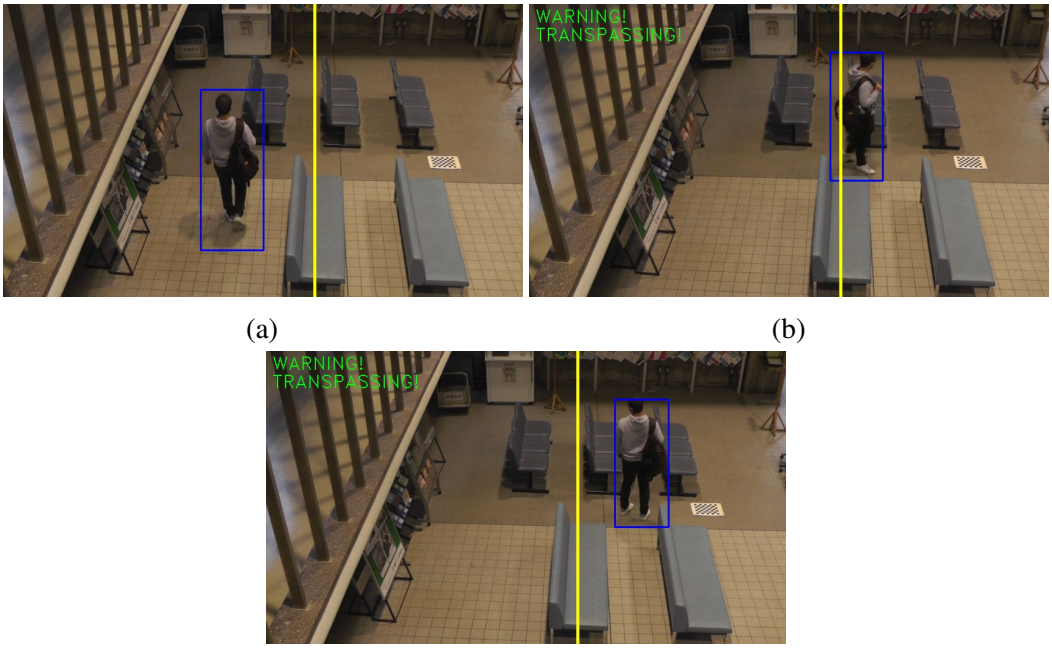


Fig. 4.19 The process of trespassing detection. (a) The single moving object, in this case, a pedestrian, is outiside the controlled area, which is bounded with a yellow line, (b) The centroid of the active blob tracking the pedestrian has just surpassed the boundary. The alarm is triggered and is displayed in the top left corner of the video frame, (c) The centroid is still inside the controlled area. The alaram is still triggered.

consider a rectangular area not situated on the boundaries of the video frame, we will have also to consider both the y coordinate of the centroid and the video height.

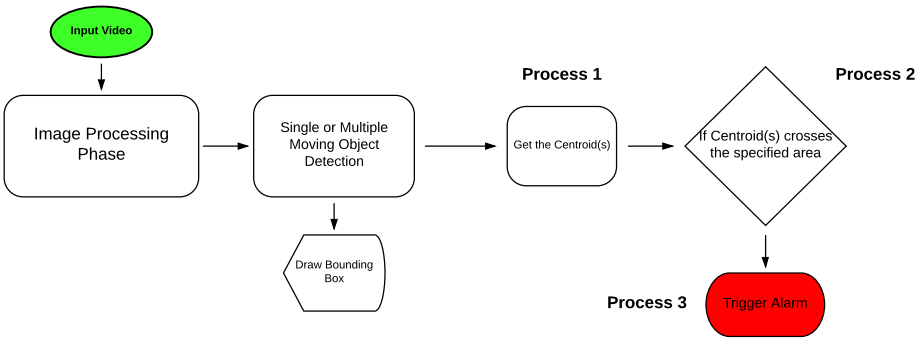


Fig. 4.20 Flowchart of trespassing detection, considering only one iteration inside the while-loop.

4.10 Loitering Detection

Loitering detection is a crucial part of most of the intelligent video surveillance systems. This feature can be used in a wide variety of application, in order to provide a safer environment for people. Indeed, loitering is generally accompanied by potential danger and suspicious behaviors that may lead to armed robberies, terrorist attacks, thefts along with the others. Furthermore, automatic loitering detection in video surveillance systems can help to save human and material resources, by implementing, as in the previous features we discussed, automated alarm alerts.
In the proposed algorithm, only one pedestrian inside the videosurveilled area is tracked. However, in Section 4.11, tracking algorithms and methods for multiple objects and pedestrians are considered. Here, two parameters are believed to be a fundamental starting point of the algorithm: $\theta_{horizontal}$ and $\theta_{vertical}$. They are calculated by the use of simple trigonometry in order to track the changing position of the target. Then, statistical functions are exploited to make it possible to fully examine the correlation between the two direction of movement, horizontal and vertical. Eventually, the parameter of time is studied, too.
In Fig. 4.24 is shown the entire flowchart of the loitering detection function, with a special regard to the core processes, explained below.

Process 1 After that the image is processed and the Waterfall algorithm is applied, Process 1 takes care of updating recurrently a frame counter, named Loiter Counter, every time the list of *coordinates* is not empty which means, therefore, that the pixel difference between the current image C_t and the background model B_t is greater than the set threshold T.

$$P_t = \begin{cases} 1, & \text{if } |C_t - B_t| > T \\ 0, & \text{otherwise} \end{cases} \tag{4.35}$$

Process 2 Process 2 is represented by the call of the function Weighted Coefficient. It calculates a coefficient ω to be considered in the updating process of the angles $\theta_{horizontal}$ and $\theta_{vertical}$, explained in Process 3. Given a time threshold of a fixed number of seconds, the ratio f between the Loiter Counter and the time threshold is obtained. If f is greater than one, the Loiter Counter has overtaken the user-defined time threshold, which automatically means that the tracked pedestrian has been walking inside the videosurveilled area for more than the determined amount of

seconds. However, if this happens, the alarm is not triggered yet because this is not the final criterion used to detect loitering. This process is only exploited in order not to have repetitive dataset to deal with, even after a monotonous-direction walk of the tracked pedestrian. Moreover, the Weighted Coefficient function makes the introduction of the time variable possible, leading to a more constrained problem and to a more precise and reliable detection.
In fact, if f is equal or greater than one, the weight ω linearly increases by a 0.1 factor depending on f.

input : Loiter Counter, fps
output : ω

Time Threshold = Number of seconds x fps;

f = int(Loiter Counter/Time Threshold)

if *Loiter Counter* < *Time Threshold* **then**
  $\omega = 1$
end
; /* f == 0 */
else if *Loiter Counter* $\geq$ *Time Threshold* **then**
  $\omega = 1 + (f \cdot 0.1)$
end
; /* f $\geq$ 1 */

Algorithm 2: Weighted Coefficient Pseudocode

Process 3 Once the weight ω has been calculated, the algorithm proceeds to elaborate the position of the tracked pedestrian through the two angles. As displayed in Fig. 4.21, the reference point of both angles is the feet of the pedestrian. However, as the names suggest, they are attributed to different fixed points, from which the angles are calculated. In addition, we considered $\theta_{horizontal}$ and $\theta_{vertical}$ to have both positive and negative values in order to have a solid dataset for the further statistical calculations. Indeed, if not directly imposed, the arcosine function would automatically give always a positive value resulting in an incomplete and incorrect representation of the real position of the pedestrian. Since both positive and negative values are calculated, the two angles range, without taking ω into account, is $[-90, 90]$.

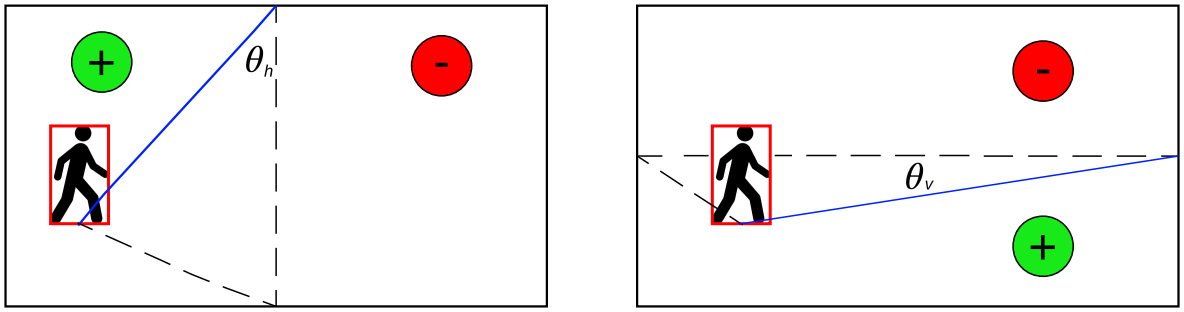


Fig. 4.21 Scheme of the two angles $\theta_{horizontal}$ and $\theta_{vertical}$ considered for loitering detection

Two examples are shown below:

$$\theta = [30, 30, 29, 26, 24, 24, 23, 20, \ldots] \tag{4.36}$$

$$\theta = [30, 30, 29, 26, 24, 24, 23, 20, 18 \cdot 1.1, 18 \cdot 1.1, 17 \cdot 1.1, \ldots] \tag{4.37}$$

In Equation 4.36, the Loiter Counter is less than the time threshold; hence, f is equal to zero. Consequently, ω is one. On the other hand, in Equation 4.37, the weight coefficient ω is multiplied from the nineth frame, when f becomes 1.

Process 4 Process 4 is based on statistical calculations. They give a robust instrument to assert whether the loitering detection is verified or not. In our opinion, the most effective and reliable instrument is the variance [3].
Given $\theta_{horizontal}$ and $\theta_{vertical}$, their variance is calculated as follows:

$$Var(\theta) = \frac{1}{N} \sum_{i=1}^{N} (\theta_i - \overline{\theta})^2 \tag{4.38}$$

[3]The variance measures how far a defined set of values are spread out from their mean value.

where the mean is obtained as follows:

$$\overline{\theta} = \frac{1}{N} \sum_{i=1}^{N} \theta_i \tag{4.39}$$

where N is the total number of angle calculated, then corresponding to the total number of frames elapsed.
It is fundamental to highlight that the variance is not calculated on the list of angles (example in Equation 4.36) which are regularly updated each frame, but on the set retrieved from these lists in order that multiple occurences are avoided, as shown in Equation 4.40.

$$\theta = \{30, 29, 26, 24, 23, 20, \dots\} \tag{4.40}$$

As a consequence, even if the pedestrian remains partially static, the variance will remain stable. This allows to detect loitering also in the scenario of a pedestrian who enters the controlled area, loiters and takes a seat. Indeed, if we only calculated the variance on the lists of angles as per Equation 4.36, 4.37, we would exclusively obtain a constantly decreasing variance, in the scenario previously described.
Additionally, Pearson's correlation is taken into account:

$$PCC_{\theta_h,\theta_v} = \frac{Cov(\theta_h, \theta_v)}{\sqrt{Var(\theta_h) \cdot Var(\theta_v)}} \tag{4.41}$$

where:

$$Cov(\theta_h, \theta_v) = \frac{1}{NK} \sum_{i=1}^{N} \sum_{j=1}^{K} (\theta_{h,i} - \overline{\theta_h})^2 \cdot (\theta_{v,j} - \overline{\theta_v})^2 \tag{4.42}$$

In this specific case, N is equal to K.
Pearson's correlation coefficient measures the linear relationship between two datasets. Similarly to other correlation coefficients, Pearson's correlation coefficient varies between -1 and +1. When it is equal to 0, it implies that there is no correlation between the two dataset, whereas coefficient equal to -1 or +1 implies an exact linear relationship [32]. For instance, given two dataset x and y, if the correla-

tion is equal to +1, it means that x increases as much as y. On the contrary, negative correlations imply that as x increases, y decreases of the same amount. Pearson's correlation coefficient is then useful to check, frame by frame, which direction the loitering pedestrian follows, without the necessity of constant monitoring by the safety staff. This may be advantageous to determine the direction where a criminal escaped, once the case was committed.

Process 5 Process 5 states whether the tracked subject is loitering or not. Three conditions are applied, two of which consider the variance of $\theta_{horizontal}$ and $\theta_{vertical}$.

- The first condition implies a time threshold. If the pedestrian remains inside the videosurveilled area for more than the user-defined amount of seconds, it will labeled as loitering. As already partially specified, the time threshold is controlled depending on the application. It is vitally important to highlight that the threshold here considered is not the same which we looked over in Process 2; In fact, the time threshold treated in Process 2 only marks a window of time beyond which the Weight Coefficient changes, whereas the time threshold in Process 5 examines the amount of time spent by the potential loitering pedestrian inside the controlled area. Obviously, the time threshold in the function Weighted Coefficient must be minor than the one investigated in the main if-statement of Process 5, otherwise the updating process of $\theta_{horizontal}$ and $\theta_{vertical}$ would not take into account scenarios such as a monotonous-direction walking or a long permanence of the target inside the videosurveilled area, which may lead to a decreasing variance;

- The second and the third conditon involve two variances. We measured the angle variances, which could include the Weight Coefficient depending on the set threshold in the Weighted Coefficient function, when a pedestrian crosses the surveilled field once, in each direction indipendently. To set the two boundary conditions, we multiply $Var(\theta_h)$ and $Var(\theta_v)$ for a factor of $2.0 - 2.2$, as displayed in Fig. 4.22.

Why is it so important to avoid a decreasing variance? Since the two main conditions of the algorithm rely on this statistical data, if the pedestrian is labeled for loitering and it suddenly stops its movements, the variance decreases and, after a certain amount of iterations, it goes below the two threshold. Therefore, if the time threshold is not yet reached, the pedestrian will not be labeled as loiterer anymore.

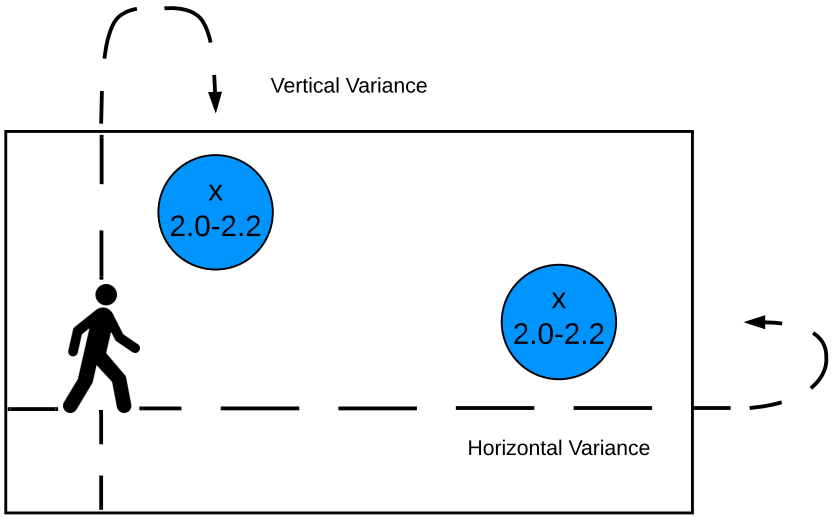
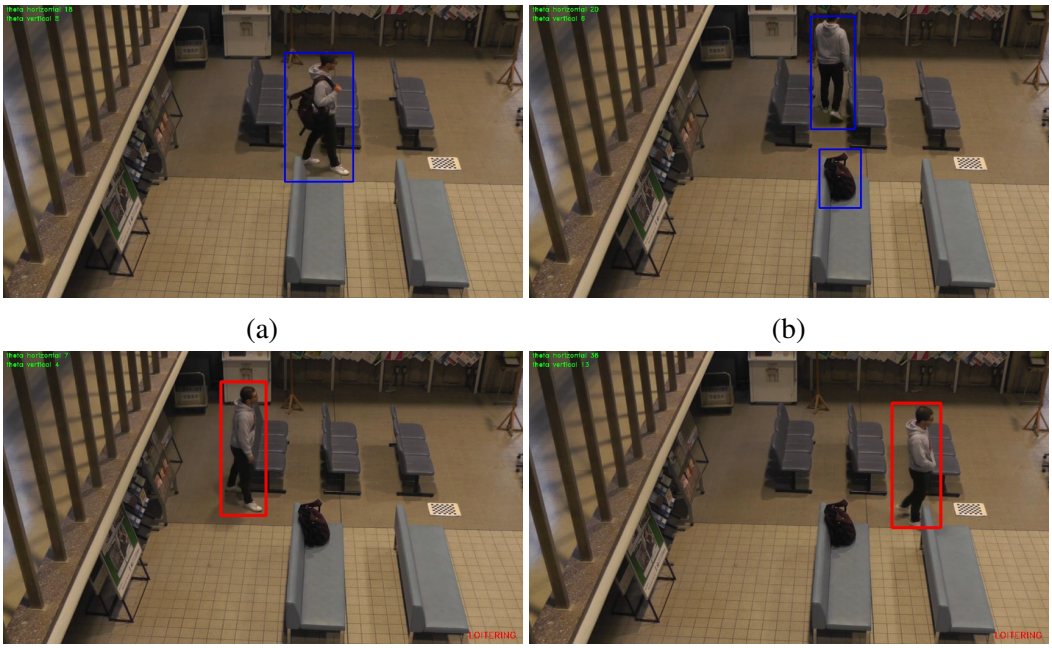


Fig. 4.22 Scheme of the two variance thresholds.



(a) (b)

(c) (d)

Fig. 4.23 Different phases of the loitering detection process, with $\theta_{horizontal}$ and $\theta_{vertical}$ displayed in the upper left corner of the frame. (a), (b) Display frames 136 and 388 when the pedestrian is walking inside the videosurveilled area, but the alarm is not triggered yet because either the variances are not beyond the thresholds or the Loiter Counter is below the time threshold (c), and (d) Show frames 667 and 731 with the alarm triggered. We can notice that the active blob sorrounding the pedestrian is red and the alarm is displayed in the lower right corner of the frame.

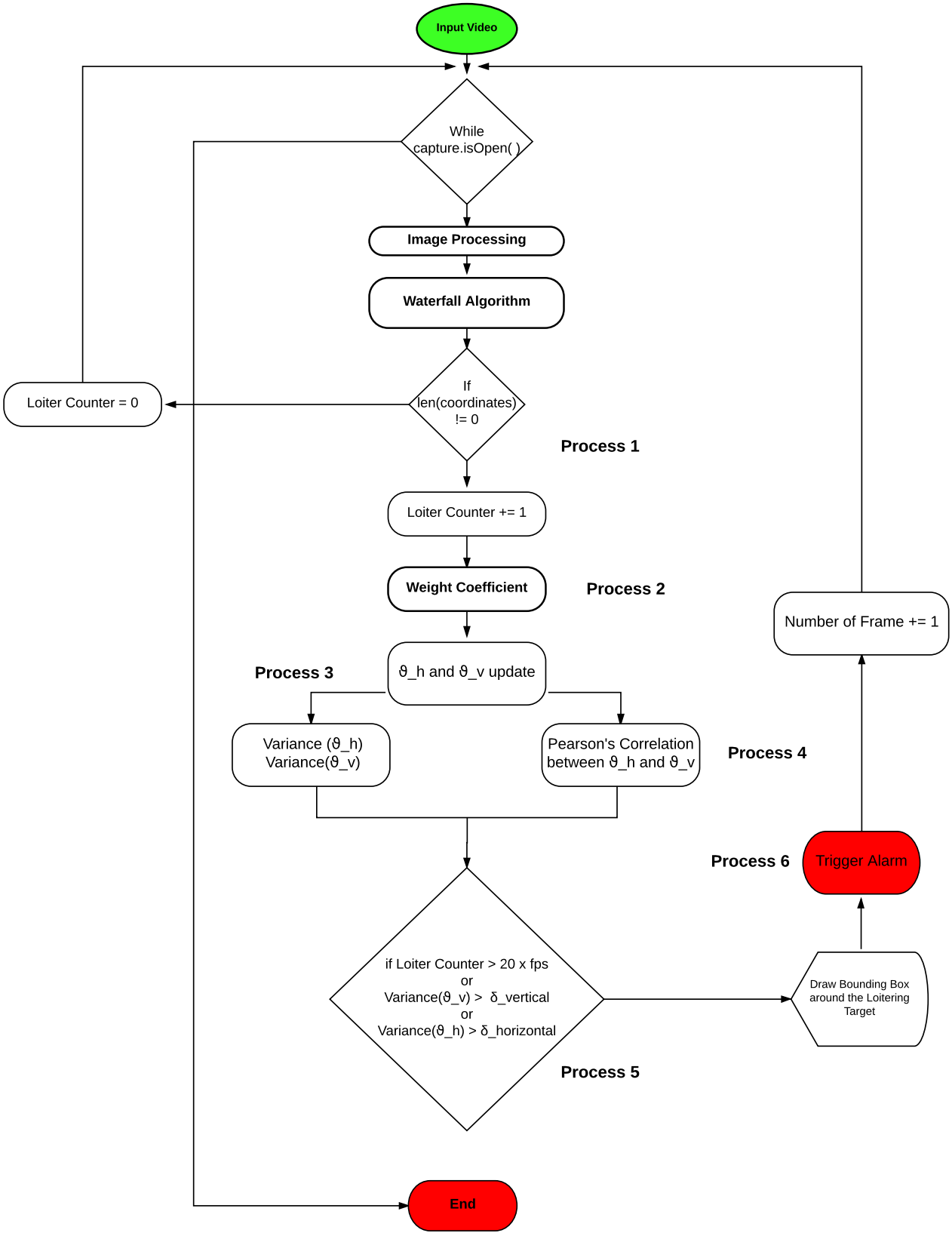


Fig. 4.24 Flowchart of loitering detection.

Process 6 If one of the previous if-statement condition is true, an alarm is triggered. An example of the entire process is shown in Fig. 4.23.

4.11 Methods for People and Object Tracking

As aforecited in Section 1.1, 1.2, the objective of people and object tracking, more commonly referred as video tracking, is to gather information about the target's consecutive positions in successive video frames. To achieve people and object tracking, all the algorithms developed so far analyze the sequential video frames and give as output the relative movement of the target(s) among the frames [4].
Among the most famous techinques, template matching is one of the simplest, most accessible and most used. As the name suggests, template matching is a method for searching and finding the location of an image patch (also referred as template image) in a larger area [33]. This technique can be straightforwardly adopted both with gray-scale images and images which feature colors. With the regard to the last case, it can obviously also exploit color information to retrieve the best matches for the image structure (entire image) to fit on the mask structure (image template). In fact, the cross correlation output will be highest at places where the image structure matches the mask structure the most.
Template matching is generally implemented by comparing a template image, represented by $T_{image}(x_t, y_t)$, where (x_t, y_t) are the coordinates of each pixel in the patch image, with a region of a frame, $S_{image}(x, y)$, through different methods of comparison. Among them, the sum of absolute difference (SAD) is one of the most typical. First, the origin of the template image $T_{image}(x_t, y_t)$ is moved over each pixel of $S_{image}(x, y)$, as graphically shown in Fig. 4.25. Both pixel at (x_t, y_t) location and pixel at (x, y) location have a given level of intensity, respectively named I_t and I. Second, the absolute difference is obtained as follows:

$$AbsoluteDifference_{x_t,y_t,x,y} = |I_t - I| \tag{4.43}$$

Fig. 4.25 Template matching comparison.

Third, the ensemble of the absolute differences looped over the patch image, is summed up as per Equation 4.44.

$$SAD(x,y) = \sum_{i=0}^{N} \sum_{j=0}^{M} AbsoluteDifference(x+i, y+j, i, j) \qquad (4.44)$$

where N is the number of pixel in the x direction and M is the number of pixel in the y direction, both in the template image $T_{image}(x_t, y_t)$. Fourth, we repeat the operation of Equation 4.44 over the source image $S_{image}(x,y)$, as follows:

$$\sum_{x=0}^{K} \sum_{y=0}^{P} SAD(x,y) \qquad (4.45)$$

where K and P denote the rows and the columns of the search image, respectively. The lowest SAD score gives the estimate for the best position of template within the search image.
Another mechanism of comparison of the patch image with the source image is oftentimes referred as linear spatial filtering (LSF). The linear spatial filtering scheme calculates the sum of products between the coefficients in $S_{image}(x,y)$ and $T_{image}(x_t,y_t)$ over the whole area spanned by the template. As opposed to SAD, the position with the highest score is the best position [34].

4.11.1 Template Matching in OpenCV

In the OpenCV module, template matching is accomplished by calling the function $cv2.matchTemplate()$. It requires:

- The image where the search is running, S_{image};
- The patch or template image T_{image}. It must not be greater in dimensions than the source image;
- The comparison method the user would like to implement.

$cv2.matchTemplate()$ returns a gray-scale image, where each pixel denotes through its intensity values, thus its brightnees, how much the neighbour of that pixel matches with the template image. In addition, if the input image has X by Y dimensions and the template is of size x by y, the output will have X-x+1, Y-y+1 dimensions. Once the output is obtained, the function $cv2.minMaxLoc()$ finds the exact location of the maximum or the minimum brightness, depending on the method used [33]. Considering that as the top left corner, we consider the two dimensions of the image template x and y as width and height of the rectangle.

Chapter 5

Benchmark Analysis

5.1 Introduction

Benchmarking a computer program is meant to assess the relative performance of the program itself, or of its components in order to find out its bottlenecks, weak points and strength points. Consequently, different architecture processor layouts would be compared.
Since computers have become more advanced and complex over the last decades, a need for a more precise way to analyze different systems arose. Looking through the system's specifications was considered not enough representative. For instance, processing units which run at a higher clock rate does not have automatically more computational power than a lower frequency processor. A benchmark analysis is designed to emulate a defined type of workload on the system, and this is accomplished in two different ways:

- Synthetic benchmark: speicifically creates different tests which have a distinct and precise load on the systems to be tested. Indeed, synthetic benchmark is more suitable for testing individual components, like CPUs, hard disks and networking devices;
- Application benchmark: runs real-world programs on the systems. As a result, it gives a wider understanding of the total performance on the tested system.

As already stated, benchmark analyses are an essential focus on CPU design, giving processor architects the ability to measure and make tradeoffs during the designing approach [35]. For example, if a benchmark analysis is applied to evaluate the

performance of an application, one or several weak points may be straightforwardly highlighted. By a deeper investigation, the key weak points may be pointed out giving indications where to operate fostering velocity and reliability in the system. The most common types of benchmarks are:

- Real program;
- Component benchmark or microbenchmark, where small and specific piece of code are analyzed together with the computer's basic components. It also may be used for gather precise information about the system's hardware parameters, cache size and memory latency [36];
- Synthetic benchmark;
- Database benchmark, intended to measure the volume of information produced and the response times of the database;
- Parallel benchmark, used on machines with multiple cores, processors or systems consisting of multiple machines [35].

5.2 Python Benchmarking

In the Python environment, it is not always the case that every program that has been written needed a deep and accurate performance analysis. However, Python has been endowed over the years with several tools which can accomplish wide and reliable performance analysis.
The quickest and roughest method can be implemented directly from the terminal. It is the UNIX utility time which gives three outputs as shown in Fig. 5.1, based on the machine specification displayed in Tab. 5.1.

```
100
Finished.

real    0m34.731s
user    0m3.754s
sys     0m0.493s
(Robocam) MacBook-Pro-di-Alfredo:matteo_exp_dataset1 Matteo$
```

Fig. 5.1 Implementation from the Terminal of Unix utility time

Computer Specifications	
Model	MacBook Pro Mid 2012
CPU	2.5 GHz Intel Dual-Core i5 (Turbo Boost up to 3.1 GHz)
Cache	3 MB L3
Memory	8 GB RAM - 1600 MHz

Table 5.1 Computer specifications

What is the main difference between the three times? Let us consider them one by one:

- Real is the elapsed real time which the program takes from its very start to end of the call. This time includes also the delays and the time a resource needs to become available. It does not have to be confused with the CPU time, which only covers the time during which the processing unit actually works on tasks;
- User is the actual CPU time used in executing processes, outside the kernel;
- Sys is the amount of CPU time spent in the kernel within the process.

In order to have a better understanding of the difference between the user time and the system time, let us make an example. If we just invoke a program which does several iterations in loop through an array, the time spent will be accumulated in the User CPU time. On the other hand, by calling directly a function from a program the time will be added to the System time. Summing up both the System time and the User time, the total CPU time is obtained. The latter may be greater or less than the Real wall clock time. In the first case, the processing cores may be more than one and then work concurrently. In the second case, the program may spend a certain amount of seconds waiting and not executing any kinds of operation.
With the regard to our specific algorithm, we used a grain timing benchmark analysis by simply importing the *time* module and using the function $time.clock()$. $time.clock()$ returns the current processing time expressed in seconds. It is important to highlight that the output of the considered function is completely subject to the operating system used. Indeed, on Windows operating computers $time.clock()$ returns the total elapsed time (wall clock time) expressed in seconds since the function has been called, whereas on Unix the same function gives as output the current processing time, also expressed in seconds. In every single case, however, the result

is deeply depending on different factors, which cannot be controlled by the user. For robustness of results to be guaranteed, a sequence of timing test should be made. As a result, considering the average of the obtained results, the truthfulness of the process is assured.

5.3 Results

The first analysis is held on the three main parts that compose the proposed algorithm. These parts are:

1. Inside the image processing phase, the background modeling, as described in Section 4.5. Here, the background model is obtained by running the Gaussian average method with the two different learning rates, right after a gray-scale image conversion and the application of a median blurring filter to the current frame;

2. Still inside the image processing phase, the image treatment phase, also described in Section 4.5. This time we work either on the single fast-updating background or on both of them, depending on the application. Thresholding is implemented right after background subtraction is accomplished. Eventually, morphological operations are carried out on the final result.

3. Finally, the Waterfall algorithm is considered. As fully explained in Subsection 4.5.1, given the morphologically treated binary difference, all the coordinates are retrieved, checked and eventually merged to attain one or multiple bounding boxes which sorrounds the active foreground objects.

The results are displayed in Fig. 5.2

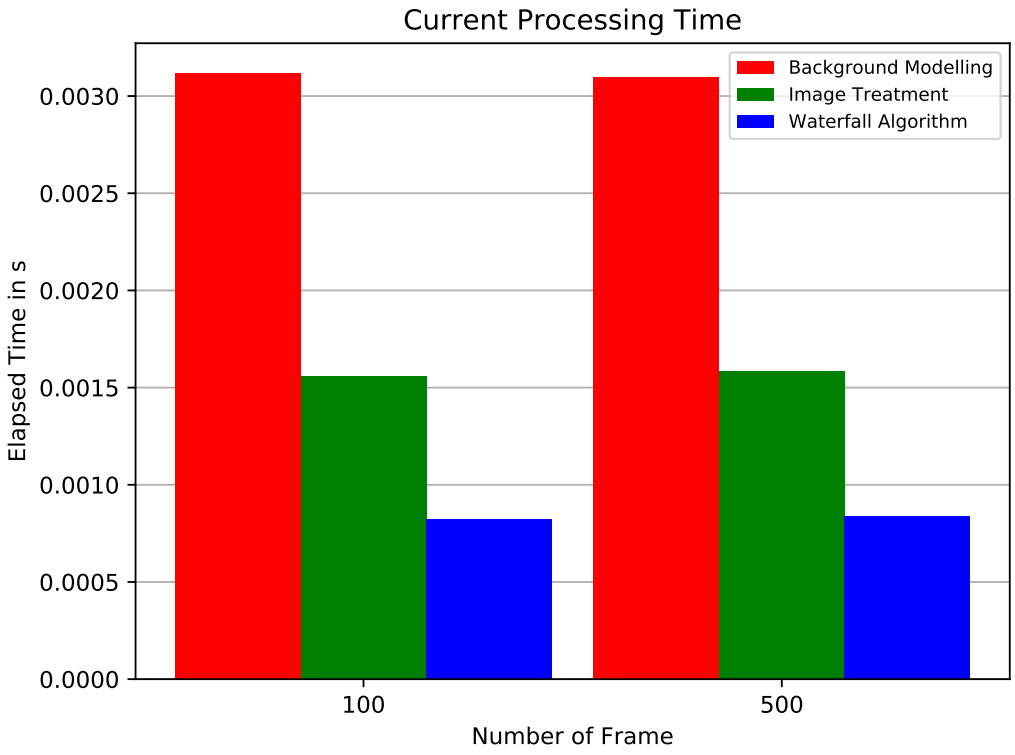


Fig. 5.2 The processing time of the three main functions which compose the proposed algorithm, for the first 100 and 500 frames.

The red bar charts display process number 1, the green ones display process number 2, and the blue ones represent process number 3.
We can distinctly see that the background modeling is sharply the heaviest phase from the computational workload point of view. Indeed, the running Gaussian average (Subsection 4.3.2) is implemented together with matrix conversion to images, median blurring and gray-scale conversion. All of these processes are quite heavy because they deal with large amount of information coming from the input videoset. Due to this, if an optimization has to be conducted, this would surely be the process under the spotlights. Secondly, backgorund subtraction, thresholding and the morphological transformations. They lead to a considerable reduction to almost half of the processing time. Nonetheless, also in this case, the amount of information handled is remarkable. In fact, the program is still working on the image entireness which represents, frame by frame, the dataset. Thirdly, the Waterfall algorithm processing time is calculated. It is noticeable a moderate reduction of the processing time due to the fact that the considered phase works with substantially less amount

of pixel information compared to the previous two phases.
To examine a deeper but still quite simple analysis, background modeling phase and the image treatment are divided into different subprocesses. Afterwards, using the function *time.clock()* makes it possible to calculate the processing time of each subprocess. More precisely, background modeling is separated into four different subprocesses as shown in Tab. 5.2:

- Gray-scale conversion;
- Application of the median blurring filter;
- Creation of the two updating background models by the running Gaussian average.

Simultaneously, image treatment is divided into two main suboperations:

- Binary subtraction and thresholding;
- Morphology operation:
  - Erosion;
  - Dilation.

Processing times in s			
Operation	Processing time at frame 100	Processing time at frame 500	Note
Gray-scale	0.001370	0.001375	
Median Blurring	0.000877	0.000891	
RGA 1	0.000431	0.000449	Fast-updating background model
RGA 2	0.000436	0.000447	Slow-updating background model

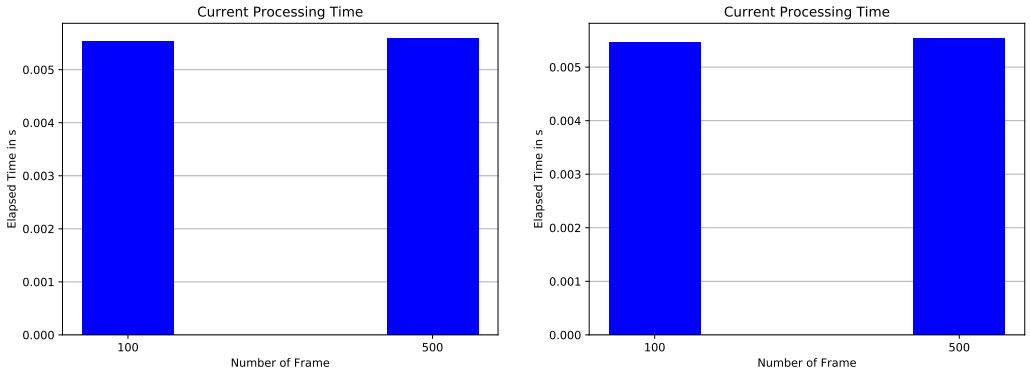
Table 5.2 Processing times of the subprocesses composing the background modeling phase

It is important to mention that the operation of dilation is conducted five times, whereas the counter operation of erosion is only implemented once. Indeed, the processing time of the latter is remarkably smaller, as displayed in Tab. 5.3. Furthermore, the sum of the each operation is attributable to the entire processing time

Processing times in s			
Operation	Processing time at frame 100	Processing time at frame 500	Note
Binary difference	0.000136	0.000129	720x480 frame resolution
Thresholding	0.000276	0.000272	
Erosion	0.000443	0.000449	Iterated once
Dilation	0.000712	0.000734	Iterated five times

Table 5.3 Processing times of the subprocesses composing the image treatment phase

displayed in the tab chart in Fig. 5.2, as expected.
In Fig. 5.3 we can see how, summing up the processing time of the three main functions which build the proposed algorithm, the total processing time nearly corresponds to the sum. An hypothetic minimal difference could be due to some hidden



(a) (b)

Fig. 5.3 The comparison of processing time of the core of the algorithm: (a) Total time calculated by a single iteration of $time.clock()$ function, (b) Sum of the three main functions.

parameters which cannot be directly controlled by the user. Then, as in this specific case, a minimum error is accepted.

5.3.1 Optimization

In this section we will describe a simple process of optimization aimed to decrease the processing time of the heaviest workload phase. The considered phase is the first of the three described in the previous Section. We used exactly the same approach

described in Section 5.2. The Figure below displays the obtained results when we divide the background modeling into three distinct stages, as per Tab. 5.2:

1. Gray-scale conversion of the current frame, given the videoset;
2. Application of a median blurring filter with a Kernel matrix sixe of $5X5$;
3. Implementation of two running Gaussian average modeling which creates two background models.

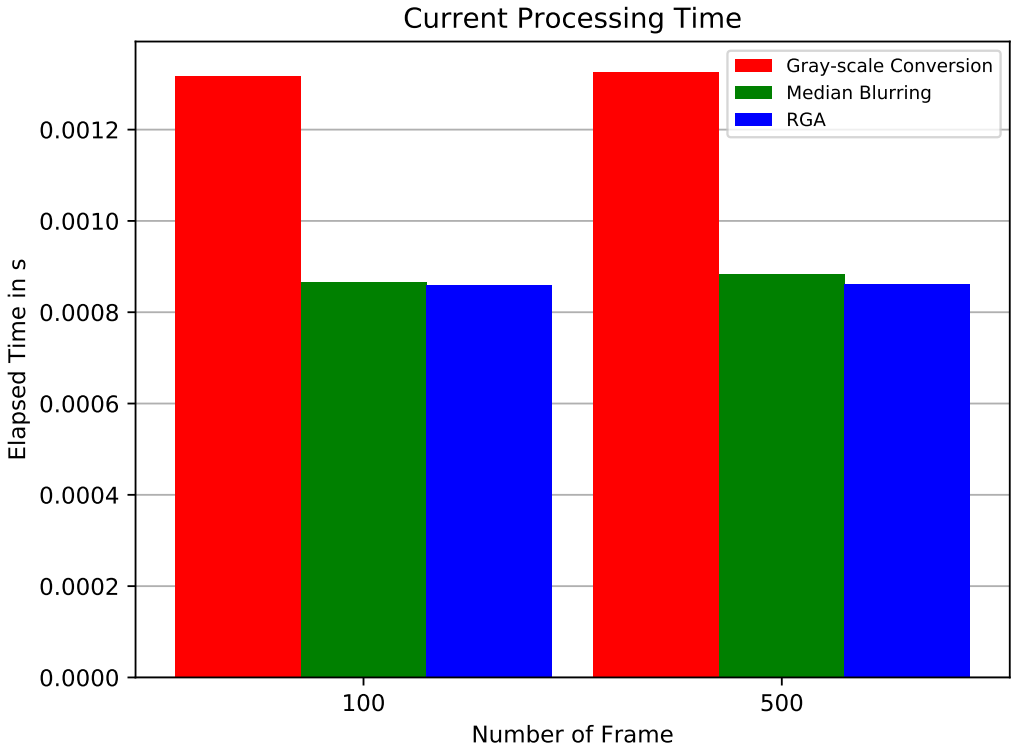
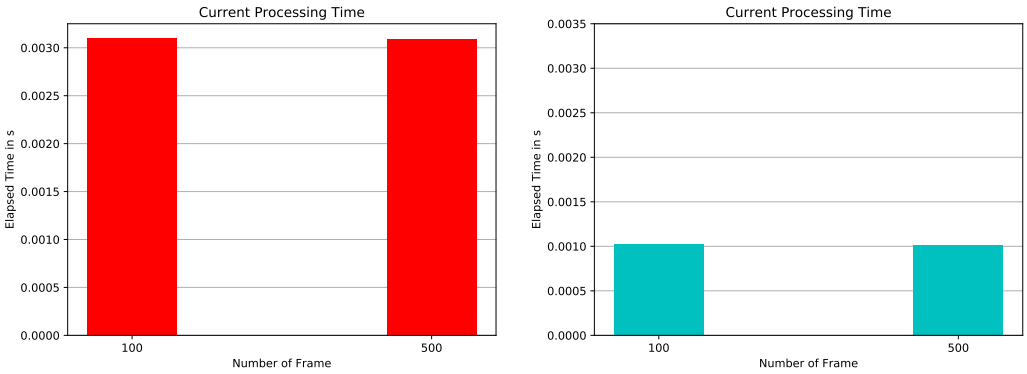


Fig. 5.4 The processing time of the three main stages of Background modelling, for the first 100 and 500 frames.

Again, the first stage is considerably the most substantial one with the regard to the computational process time. The gray-scale conversion considers all the pixel of the image, processes them and assigns a new value to each of them. In order that the processing time is reduced, the videoset is modified and it is turned to a low resolution video 320x240, whereas before was 720x480. The amount of pixel to process now has moderately decreased. Indeed, the new processing time of the three main stages of background modelling are:



(a) (b)

Fig. 5.5 The of processing time of the main stage of Background modelling: (a) With 720x480 frame resolution, (b) With 320x240 frame resolution, then optimized.

Fig. 5.5 displays a decline of the processing time by slighlty more than three times, in average, as a result of the smaller definition of the dataset considered.

5.4 Real-time Application

All the examined measurements were performed on two videoset with different resolutions. However, the fps[1] are both shared and are set to 30[2].
Let us consider the amount of time, calculated in seconds, of a single frame:

$$time_{oneframe} = \frac{1}{30} = 0.0333 \tag{5.1}$$

Let us also consider the sum of the processing time of the core functions of the algorithm, as per Table 5.4. As we can see, the sum of the elapsed time of the three main functions is significantly minor than the amount of time for a single frame to be processed. To be more specific, it approximately represents the 18% of the total available time, $time_{oneframe}$. As a result, considering 30 fps and taking the machine specifications into account, the algorithm could be successfully run within this window of time. Due to this, real-time application is accomplished.

[1] Frames per second.

[2] Both videos were decoded setting fps to 30, but the actual fps conversion is always accompained by errors; indeed, 30 represents the peak frame rate of the video decoder.

Processing times in s		
Operation	Processing time at frame 100	Processing time at frame 500
Background Modelling	0.00315	0.00313
Image Treatment	0.00165	0.00165
Waterfall Algorithm	0.00090	0.00086
Total	0.0057	0.00564

Table 5.4 Processing times of the three main functions which compose the proposed algorithm

Chapter 6

Conclusions

Going through the examined features of image processing, background modelling, people detection and object detection, we proposed a complete algorithm able to deal with a broad variety of different scenarios. The proposed Waterfall algorithm increased the reliability and performance of both single and multiple pedestrian tracking by outdoing the challenges in the background modelling process, shown by a vast amount of previous methods. By taking into account the two background models, abandoned and taken away objects are robustly detected and successfully labeled; consequently, further color and position information are gathered. Additionally, trespassing detection and loitering detection are successfully achieved by comparing the position of the tracked moving object and the high-sensibility area. Furthermore, time thresholds are investigated together with important and relevant statistical tools such as the covariance and the Pearson's Correlation, which significantly helped us to understand the trend of the continuously changing position of the target inside the videosurveilled area. Finally, a benchmark analysis is conducted in order to interpret the computational load of the proposed algorithm. The aim was to discern the effectiveness of a real-time application of the video surveillance algorithm, which showed positive results.

The experiments were performed using several videos personally recorded with a static camera in the main hall of the engineering faculty at the Hokkaido University. Undoubtedly, future work can be arranged in the interest of enhancing the robustness, the overall performances and the accuracy of the main functions of the algorithm.

References

[1] Josh Woodhouse. Enterprise & ip storage used for video surveillance. Technical report, IHS Markit, 2017.

[2] Michael Kamaraj et al. An improved motion detection and tracking of active blob for video surveillance. In *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*, pages 1–7. IEEE, 2013.

[3] E Roy Davies. *Machine vision: theory, algorithms, practicalities*. Elsevier, 2004.

[4] Wikipedia. Video tracking, 2017. Visited on: 15-06-2017.

[5] Wikipedia. Binary image, 2017. Visited on: 12-06-2017.

[6] Wikipedia. Kernel (image processing), 2017. Visited on: 13-06-2017.

[7] Wikipedia. Median filter, 2017. Visited on: 21-06-2017.

[8] Wikipedia. Histogram equalization, 2017. Visited on: 26-07-2017.

[9] Wikipedia. Balanced histrogram thresholding, 2017. Visited on: 13-06-2017.

[10] Willow Garage. About willow garage, 2017. Visited on: 23-06-2017.

[11] PCmag. Encyclopedia: definition of cross-platform, 2017. Visited on: 23-06-2017.

[12] OpenCV. Introduction to opencv-python, 2017. Visited on: 22-06-2017.

[13] Tim Peters. The zen of python. In *Pro Python*, pages 301–302. Springer, 2010.

[14] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6), 2012.

[15] Michael Barr. Embedded systems glossary. *Neutrino Technical Library*, 2007.

[16] Wikipedia. Embedded system, 2017. Visited on: 5-07-2017.

[17] Raspberry Pi Foundation. Raspberry pi downloads, 2017. Visited on: 5-07-2017.

[18] Raspberry Pi Foundation. Raspberry pi documentation, 2017. Visited on: 5-07-2017.

[19] Waritchana Rakumthong, Natpaphat Phetcharaladakun, Wichuda Wealveerakup, and Nawat Kamnoonwatana. Unattended and stolen object detection based on relocating of existing object. In *Student Project Conference (ICT-ISPC), 2014 Third ICT International*. IEEE, 2014.

[20] Massimo Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.

[21] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):780–785, 1997.

[23] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, Hélène Laurent, and Christophe Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[24] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages 246–252. IEEE, 1999.

[25] Yong Xu, Jixiang Dong, Bob Zhang, and Daoyun Xu. Background modeling methods in video analysis: A review and comparative evaluation. *CAAI Transactions on Intelligence Technology*, 1(1):43–60, 2016.

[26] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31. IEEE, 2004.

[27] Lucia Maddalena and Alfredo Petrosino. A self-organizing approach to background subtraction for visual surveillance applications. *IEEE Transactions on Image Processing*, 17(7):1168–1177, 2008.

[28] Ahmed Elgammal, David Harwood, and Larry Davis. Non-parametric model for background subtraction. *Computer Vision—ECCV 2000*, pages 751–767, 2000.

[29] Yosuke Nonaka, Atsushi Shimada, Hajime Nagahara, and Rin-ichiro Taniguchi. Evaluation report of integrated background modeling based on spatio-temporal features. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 9–14. IEEE, 2012.

[30] Kentaro Toyama, John Krumm, Barry Brumitt, and Brian Meyers. Wallflower: Principles and practice of background maintenance. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 255–261. IEEE, 1999.

[31] Weihua Wang and Zhijing Liu. A new approach for real-time detection of abandoned and stolen objects. In *Electrical and Control Engineering (ICECE), 2010 International Conference on*, pages 128–131. IEEE, 2010.

[32] Statsoft. Statistics glossary, 2017. Visited on: 04-08-2017.

[33] OpenCV. Template matching, 2013. Visited on: 29-08-2017.

[34] Wikipedia. Template matching, 2017. Visited on: 29-08-2017.

[35] Wikipedia. Benchmark (computing), 2017. Visited on: 01-08-2017.

[36] Andreas Ehliar and Dake Liu. Benchmarking network processors. In *Swedish System-on-Chip Conference (SSoCC)*, 2004.