TECHNICAL UNIVERSITY OF VALENCIA

DEPARTMENT OF COMPUTER ENGINEERING

Study of segmentation and identification techniques applied to environments with natural illumination and moving objects

Thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatics

Juan Alfonso Rosell Ortega

Ph.D. advisors:
Dr. Gabriela Andreu García
Dr. Alberto Pérez Jiménez

Valencia, April 2011

Abstract

The developments introduced in this thesis belong to the area of computer vision and are aimed to provide with ideas to solve the problem of automatically segmenting objects in images acquired in environments with activity, i. e., those in which objects are moving and the illumination is not controlled. In order to develop the ideas and evaluate their performance, two different problems, from the point of view of requirements and environment conditions, are given a solution. Both of the problems are considered as challenges by the research community in the area of computer vision.

The first problem to be considered consists on segmenting, and later identifying, the container's codes of trucks. For this challenge, images taken at the entrance of a commercial port were used. In this case, segmentation techniques that allow the extraction of concrete objects fom indidivual images, characters in this case, were implemented. Natural light is not the only challenge in this case, but also the conditions of the containers themselves. In this context, we study different techiques from the literature as LAT, Watershed, Otsu's algorithm, local variation and thresholding algorithm, using them to segment gray-tone images. Based on this study, a solution is proposed combining several different techniques as an approach to successfully extract characters regardless of the environmental conditions.

Joining several segmentation techniques into a single method produces noisy segmentations. The knowledge of the features of the seeked objects helps designing filters which can discriminate the valid objecs from noise, avoiding this way to relay only on a classifier. The proposed system does not need any parameter tuning in order to adapt to light variations and achieves a high level of segmentation and identification of characters, although the system performance depends greatly on the classifier's capacity. Afterwards, experiments with sequences of images are performed in order to improve system's response. Each sequence contains several images of the same truck in consecutive moments of time.

The second problem considered in this thesis, is extracting all the objects which do not belong to an scene, using streams of sequences and by designing background models which can adapt to changes in the scene, specially, light changes.

Based on thecniques proposed in the literature for background subtraction and bearing in mind the memory and computational time constraints imposed by some intelligent systems, in this thesis several techniques are proposed to obtain adaptive background models with requirements specially suited to automatic surveillance systems. The proposed techniques, called BAC (Background Adaptative with Confidence) and FBS (Fuzzy Background Subtracion), use a measure of similarity and a computation of probability based on experimental studies and are able to model the background adapting it to changes in the environment providing at the same time with a confidence measure of the built model. BAC and FSB subtract a frame from the background by assigning each pixel with a possibility of belonging either to foreground or background.

At this point, our developments are evaluted with several video sequences obtained indoors where problems shadows or sudden light changes may appear, and also the Wallflower benchmark, accepted in the literature as a good means to test background modelling technqiues. Results obtained by BAC and FSB are promising when compared to those obtained by other algorithms accepted by the community as representative.

Resumen

La presente tesis está enmarcada en el área de visión por computador y en ella se realizan aportaciones encaminados a resolver el problema de segmentar automáticamente objetos en imágenes de escenas adquiridas en entornos donde se está realizando actividad, es decir, aparece movimiento de los elementos que la componen, y con iluminación variable o no controlada. Para llevar a cabo los desarrollos y poder evaluar prestaciones se ha abordado la resolución de dos problemas distintos desde el punto de vista de requerimientos y condiciones de entorno, ambos considerados como retos por los investigadores en el área de la visión por computador.

En primer lugar se aborda el problema de segmentar, para posteriormente identificar, los códigos de los contenedores de camiones con imágenes tomadas en la entrada de un puerto comercial que se encuentra ubicada a la intemperie (luz natural). En este caso se trata de proponer técnicas de segmentación que permitan extraer objetos concretos, en nuestro caso caracteres en contenedores, procesando imágenes individuales. No sólo supone un reto el trabajar con iluminación natural, sino además el trabajar con elementos deteriorados, con contrastes muy diferentes, etc. Dentro de este contexto, en la tesis se evalúan técnicas presentes en la literatura como LAT, Watershed, algoritmo de Otsu, variación local o umbralizado para segmentar imágenes en niveles de gris. A partir de este estudio, se propone una solución que combina varias de las técnicas anteriores, en un intento de abordar con éxito la extracción de caracteres de contenedores en todas las situaciones ambientales de movimiento e iluminación.

El aunar varias técnicas de segmentación en un único método produjo segmentaciones ruidosas. El conocimiento a priori del tipo de objetos a segmentar nos permitió diseñar filtros con capacidad discriminante entre el ruido y los caracteres, evitando con ello que toda la responsabilidad de esta decisión recayera en el clasificador. El sistema propuesto tiene el valor añadido de que no necesita el ajuste de parámetros, por parte del usuario, para adaptarse a las variaciones de iluminación ambientales y consigue un nivel alto en la segmentación e identificación de caracteres, aunque las prestaciones del sistema dependen en gran medida de la capacidad del clasificador. En un paso posterior, se realizan experiencias con secuencias de imágenes de cada contenedor para refinar la respuesta del sistema.

El segundo problema analizado en la tesis, aborda la temática de extraer todos los objetos que no forman parte del fondo en una escena, utilizando secuencias de imágenes y diseñando modelos de fondo capaces de adaptarse a los cambios en la escena, especialmente a los cambios de iluminación.

A partir de las técnicas propuestas en la literatura para el restado de fondo (background subtraction) y teniendo en cuenta restricciones de memoria y computo impuestas por algunos sistemas inteligentes, en esta tesis, se proponen técnicas para obtener modelos de fondo adaptativos con requisitos propios de los sistemas de vigilancia automática (surveillance system). En concreto, mediante una definición propia de simili-

tud y utilizando un computo de probabilidad basado en un estudio experimental, se proponen dos algoritmos denominados BAC (Background Adaptative with Confidence) y FBS (Fuzzy Background Subtracion) capaces de modelar el fondo con capacidad adaptativa y de proporcionar una medida de confianza. BAC y FBS llevan a cabo el restado de fondo asignando a cada pixel una probabilidad de pertenecer a fondo o de ser parte de un objeto (foreground).

En este punto nuestros desarrollos se evalúan con secuencias de imágenes adquiridas en interiores donde aparecen problemas de sombras, cambios de iluminación, así como con el benchmark Wallflower aceptado en la literatura especializada para testear técnicas de modelado de fondo. Los resultados obtenidos por las técnicas propuestas BAC y FBS se muestran prometedores al ser comparados con los obtenidos por otras técnicas presentes en la literatura.

Resum

La present tesi pertany a l'àmbit de la visió per computador i realitza aportacions amb l 'objectiu de resoldre el problema de segmentar automáticament objectes en imatges adquirides en entorns amb activitat, és a dir, on apareix moviment dels elements que les composen i amb il.luminació variable. Per a poder dur a terme els desenrollatments i poder evaluar les prestacions des de'l punt de vista dels requisits i condicions de l'entorn, ambdós considerats com a reptes pels investigadors en l'área de la visió per computador.

Primerament, s'aborda el problema de segmentar, i posteriorment identificar, les matrícules dels contenidors de camions amb imatges presses a l'entrada d'un port comercial que es trova situada a l'intemperie (llum natural). En est cas, es tracta de proposar tècniques de segmentació que permitisquen extraure objectes concrets, caracters de les matrícules, processant imatges individuals. No és tracta només de superar el repte de treballar amb il.luminació natural, si no, a més, treballar amb imatges deteriorades, amb contrastos molt diferents.... Dins d'est context, a la tesis se evaluen tècniques de la literatura com ara LAT, Watershed, l'algoritme d'Otsu, la tècnica de variació local o l'umbralitzat per a segmentar imatges en nivells de gris. A partir de l'estudi, es proposa una solució que combina varies de les tècniques presentades en un intent d'abordar amb èxit l'extracció de caracters de la matrícula en totes les situacions ambientals possibles.

Unir varies tècniques de segmentació en un sol mètode produeïx segmentacions sorolloses. El coneiximent a priori del tipus d'objects a segmentar ens permiteix dissenyar filtres amb capacitat discriminant entre el sollor i els veritables caracters, evitant que tota la responsabilitat d'esta separació caiga sobre el classificador. El sistema proposat té un valor afegit en tant que no necesita ajustar parámetres externs per part de l'usuari per a adaptar-se a les variacions d'iluminació ambiental i aconsegueix un nivell alt tant en la segmentació com la identificació de caracters, malgrat que les prestacions del sistema depenen en gran mesura de la capacitat del clasificador utilitzat. En un pas posterior, es realitzen experiències amb secuències d'imatges de cada contenidor per a refinar la resposta del sistema.

El segon problema que s'analitza és el d'extraure tots els elements que no formen part del fons d'una escena, utilitzant secuències d'imatges i dissenyant models de fons capaços d'adaptar-se als cambis en l'escena, especialment als canvis d'il.luminació.

Prenint com a base les tècniques proposades en la literatura per al restat de fons (background subtraction) i tenint presents les restriccions de memòria i còmput imposades per alguns sistemes inteligents, en esta tesi es proposen tècniques per a obtindre models de fons adaptatius amb requeriments propis dels sistemes de vigiláncia automática (surveillance systems). Concretament, mitjançant una definició propia de similitut i utilitzant un còmput de probabilitat bassat en estudis experimentals, es proposen dos algorismes anomenats BAC (Background Adaptive with Confidence) i FSB (Fuzzy Background Subtraction) que poden modelar el fons adaptativament i propor-

cionar una mesura de confiança. BAC i FSB resten cada fotograma del fons assignant a cada píxel una probabilitat de pertányer al fons o de ser part d'un objecte (foreground).

En est punt, els nostres desenvolupaments s'evaluen amb secuències adquirides en interiors on apareixen problemes amb sombres, canvis d'il.lumniació, així com també utilizant el benchmark Wallflower acceptat per la comunitat científica per a provar tècniques de modelatje de fons. Els resultats obtesos per les tècniques proposades, BAC y FSB es mostren prometedors si es comparen amb els obtesos per les tècniques presents en la literatura.

Contents

List of Tables

List of Figures

Notation

I - acquired image.

$I(x, y)$ - value of image I in coordinates (x, y).

S - image representing the result of a segmentation algorithm, $S(x, y) = 0$ means that pixel (x, y) was classified as background, $S(x, y) = 1$ means that pixel (x, y) was classified as foreground.

T - threshold.

p, q - generic pixels.

$N_k(p)$ - k- neighbourhood of pixel p.

$N_8(p)$ - neighbours of pixel p at a distance equal to 1.

R - region of interest of an image, represented as a set of connected pixels.

$G(V, E)$ - a graph defined on an image, being V the vertices of the graph, representing each pixel of the image and E the set of edges that connect each vertex to its 8 connected neighbours.

$F(t)$ - image acquired in time t.

$F_{x,y}(t)$ - value of frame F in coordinates (x, y) in time t.

$B(t)$ - background model in time t.

Chapter 1

Introduction

In this chapter we present the framework in which this thesis was developed: the Computer Vision Group of the DISCA department in the Technical University of Valencia and the University Institute of Control Systems and Industrial Computing (better known with the acronym AI2). The goals of the projects which motivated our work in these areas are outlined along with the contributions made.

1.1 Computer vision systems

Computer vision deals with extracting meaningful descriptions of physical objects from images. Tasks such as identifying a signature, locating faces or recognizing objects in a scene are considered to be within the scope of computer vision. Humans can perform these tasks effortlessly, but developing a system to perform them is a very difficult process. Usually, for a given process, the whole task must be divided into easier-to-solve stages, which perform one or more operations in the incoming data.

The design of a specific computer vision system consists on finding the most suitable techniques in order to process the captured images. For each concrete problem, it is necessary to select specific algorithms and techniques. In figure 1.1, the phases in which a computer vision system may be divided are shown.

Each step in the vision process corresponds to:

- Acquisition : acquisition and digitalization of the image.
- Preprocessing : modify the image to improve it in some way; usually with the aim of ease its manipulation.
- Segmentation : divide the image in a set of objects and background.
- Feature extraction : extract descriptors of the regions of interest of the image. Some typical descriptors are shape descriptors, colour, histograms...
- Classification : classify objects attending to the descriptors extracted in previous step.
- Interpretation: give a description of the image, in terms of the application, taking into account the results of previous stages.

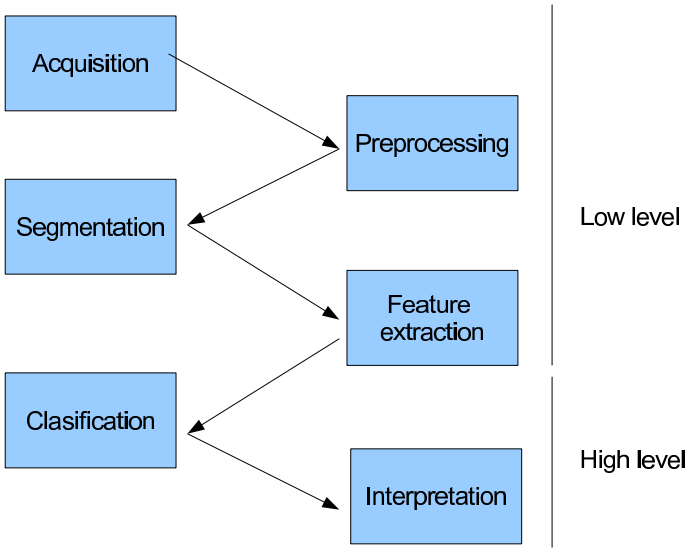


Figure 1.1: The vision process in its different steps.

Each one of these steps depends highly on the concrete information we look for in images. This means, that a processing technique which works fine with a concrete set of images for a certain problem, probably will not perform good for another problem.

Thus, for each different problem, a different solution has to be found. It is a challenge finding the best techniques for each step in the process that can give the best possible result. Of course, not every step has to appear in a real process. Sometimes, situations may be met, in which a certain step is not needed. And it is also true, that these steps are not always so clearly separate. It may occur, for instance, that we have to apply different preprocessing techniques before segmenting, or that one or more of these steps are fused into one.

1.2 Image segmentation

Image segmentation, either in grey tones or in colour images, is defined in the computer vision literature as the process of dividing an image into disjoint regions in such a way, that the union of all these regions results in the original image. This division is achieved by grouping neighbouring pixels according to criteria of proximity and/or similarity. With this process, further processing steps on the image may be centred only in certain regions and not in the complete image. The classical definition of segmentation of an image I defined as a partition of I into components or regions R_i, which verify that $R_i \cap R_j = \emptyset \wedge i \neq j$ and that $\bigcup R_i = I$.

The task of finding the best segmentation method for an specific application is still a difficult challenge. One of the difficulties that any segmentation technique has to face is the illumination of the scene. Due to the fact that shines or shadows affect dramatically the search of valid thresholds to segment the image; it is always desirable to control illumination and the kind of light used to capture images. However, this is not always the case, and situations exist in which images must be taken under light conditions out of control and processed in real time, what really diminishes the amount

of preprocessing that can be performed on the image and also, the control on the capture conditions.

As said, when images are taken under non controlled light conditions, it is very difficult to adjust algorithm's parameters, in order to be able to detect all regions of interest in images. Applying an algorithm with the suitable parameters for some concrete environmental conditions, would run fine for some conditions or some images, but would fail in others, as images will contain shines or shadows which will be, unfortunately in most cases, detected as regions of interest. As a result, a very coarse segmentation may be obtained (*undersegmentation*), or the opposite case, a segmentation with a lot of regions (usually known as *oversegmentation*).

Choosing among several elements always requires a set of criteria to ground the decision on. In the case of choosing among several segmentation techniques, to find the best for a given problem, all available techniques should be evaluated in the same conditions.

Evaluation of the segmentation techniques, consists in determining if the regions detected by the algorithms as regions of interest do really correspond with regions of interest according to what is expected in the application. Usually, a human operator labels which regions are expected to be the result of the algorithm, the so called ground-truth. This ground-truth is then checked with the automatic result; optionally, after applying any filter to remove noise in the output, for instance. The quality of the response may be evaluated in several ways, for instance, testing if the areas segmented by the human operator and those detected by the algorithm overlap in a given percentage, or if centres of the regions are close one to each other. This method of a human operator labelling images in order to test the segmentation algorithms is followed in the experiments discussed later in this work.

1.3 Motivation

Visual information makes up an important amount of all the sensorial information received by a person during a lifetime. This information is processed not only efficiently but also transparently by the human brain.

The ultimate goal of computer vision is to mimic human visual perception. Therefore, in the broadest sense, robustness of a computer vision algorithm is judged against the performance of a human observer performing an equivalent task. In this context, robustness is the ability to extract the visual information of relevance for a specific task, even when this information is carried only by a small subset of the data, and/or is significantly different from an already stored representation.

One of the components of visual information is information about objects which are actually seen. The information associated to each object is huge, estimations of distance, height and width and other measures, along with boundaries of regions and relationships between these regions.

Being able to distinguish some regions from others, is very useful when a system has to recognize some specific objects. Also, the relationships between regions may be useful when the possibility exists that objects are composed of different regions, which are, by any means, connected between them.

Finding regions in images and the relationships between them is performed by segmentation techniques in computer vision. The aim of these techniques is finding the different regions in which an image can be decomposed to ease any further processing on them.

First works of this thesis started the Computer Vision group of AI2 in a project aimed to detect and recognize characters on containers, developed with the support of the grant FEDER-CICYT DPI2003-09173-C02-01. The goal of the project was developing a system that could find containers identification numbers under uncontrolled environmental situations.

In this project, a solution for the problem of segmentation under different light conditions was tested with success. The solution basically used the fusion of information of different algorithms to obtain an accurate result.

Following with a similar situation, the group started in a project within the sixth framework programme priority IST 2.5.3 Embedded systems called SENSE (Smart Embedded Network of Sensing Entities). In this case, the problem to face had some similarities with the previous one, but also notable points of difference. The goal was detecting people inside a scene, in which light was not controlled by the system. Though similar techniques to those developed previously could have been applied, one of the constraints of the system was that besides locating objects in the scene, they had to be tracked, as well.

1.4 Goals

Developments in this thesis are centred in the study of different segmentation methods applied to images taken in environments in which light cannot be always controlled. As said before, segmentation algorithms try to separate images into independent regions which meet some given properties, in order to ease further processing. Under these variant light conditions, for instance, it is difficult to set parameters for some segmentation algorithms, as it is difficult that they cover all possible situations.

The general goal of this thesis is segmenting images regardless light conditions. Techniques to achieve this goal have been developed for two different real-life applications. In the first one, the problem was recognizing truck container codes with images taken in the entrance of a commercial port. Images represent the container as it approaches the entrance and were taken as isolated images. Similar systems may be found in [brad01], [barroso97] and[hegt98] and previous related work with this issue may be found in [salva01], [salva02] and [atienza05].In this project, developed with the support of the grant FEDER-CICYT DPI2003-09173-C02-01, our goals were:

- To implement and to test image segmentation algorithms from current literature
- To test best configuration of the implemented algorithms according to problem constraints
- To develop new techniques to detect regions of interest regardless of the light conditions
- To propose methods to determine which regions of interest are valid from the point of view of the application

The second project is project 033279 within the sixth framework programme priority IST 2.5.3 Embedded systems SENSE. A surveillance project that consists in segmenting and tracking people and luggage in an airport in video sequences. In this case, illumination differences inside the same scene due to the light coming from windows,

doors and so on were in some cases important. Similar surveillance systems are described and discussed in [haritaoglu00], [vsam99], [wren97] and [hu04]. Our goals for this project were:

- Implement and test video analysis algorithms from literature
- Propose techniques to create background models within the constraints of the SENSE application
- Define a measure of the quality of a background model
- Develop sets of features which can be used to classify objects into three classes of interest

1.5 Contributions

Bearing in mind the general goal of this work, and the challenges which motivated these developments, several contributions were made either directly related to the problem of object segmentation or to add new techniques to the existing corpus. A brief introduction to our contributions is listed below:

- **Merge different algorithms to segment images**. In [rosell06] we discuss the use merging of the segmentation results of several algorithms for the same image as a way to improve segmentation performance.
- **Apply filters to remove false positives**. In [rosell06a], we propose methods to discard regions of interest obtained by using segmentation methods which do not represent valid elements from the point of view of the application. Extending this idea, in [rosell06b] we explore the use of time-shifted images representing the same object to obtain better segmentation results; the integration of the information obtained in each image to conclude a final segmentation is presented.
- **Background modelling**. Having a background model that describes accurately a scene is crucial when background subtraction techniques are used. In [rosell08a] the BAC algorithm is introduced, this algorithm creates or restores a background model based on the behaviour of pixels in successive frames and, at the same time, performs a segmentation of objects in the scene, with the novelty that it yields a confidence value for the obtained background. BAC is extended to use colour [rosell09] and to support multiple model descriptions per pixel (MBAC [rosell10]). The difficulty to find a method to determine segmentation threshold motivated the development in [rosell10b] of the FSB algorithm, which eludes the use of fixed or probabilistic thresholds usually found in the traditional background subtraction.
- **Background model's confidence and corruption detection**. In [rosell08a], [rosell09], [rosell10] and [rosell10b] we discuss algorithms that attach the background models they build with a measure of the confidence of the model. With this measure, the algorithms estimate how close to the reality the model is. The algorithms discussed also detect model's corruption by means of different measures, this detection permits them recomputing the model in case of failure.

- **Person, group and luggage recognition**. Research was done in the recognition of different objects in scenarios. A set of different techniques aimed to classify objects in three groups are discussed in [rosell08] and [atienza08].

1.6 Organization

The thesis has been organized into 7 chapters and 2 appendices. First chapter corresponds to this introduction. In chapter 2, a review on different segmentation methods and preprocessing techniques is discussed and some algorithms are introduced in detail. Chapter 3 shows the state of the art in segmentation techniques for video streams; as these are slightly different from those used to segment stand alone images, it seemed better to write a chapter for each set of techniques. In chapter 4, 5 two different problems are solved using similar approaches, several segmentation techniques and filters for the output. First problem, detecting and recognizing characters on truck containers is addressed in chapter 4. In chapter 5, the problem is detecting and tracking people and luggage in an airport. Chapter 6 presents the contributions made if the field of background modelling algorithms and a comparison with current state of the art. In chapter 7, conclusions are discussed and future works are proposed. In appendix A a brief introduction to the top-hat operator may be found; appendix B explains in more detail the classifiers used during the development of the solutions for both problems.

Chapter 2

Image segmentation

This chapter is devoted to introduce the concept of image preprocessing and segmentation, and discuss well-known preprocessing techniques, such as Sobel or top-hat operator, and segmentation techniques, such as Otsu's algorithm, Local Adaptive Thresholding (LAT), Watershed, global thresholding or local variation algorithm. These techniques will be the basis for further developments introduced in this thesis.

2.1 Introduction

Image segmentation is the process of dividing an image into different, non overlapping regions with the aim of ease the processing of the image. Several algorithms have been developed to segment images, though techniques may be grouped into *thresholding techniques*, which are those that calculate a threshold, either local or global, to segment the image and *growing regions algorithms*, which create the regions by joining pixels between them if some criteria is met.

In many applications of image processing, the grey tones of pixels belonging to searched objects are substantially different from the grey tones of the pixels belonging to background. Thresholding becomes a simple but effective tool to separate objects from the background. Examples of thresholding applications are *document image analysis*, where the goal is to extract printed characters, logos, graphical content, or musical scores. *Map processing*, where lines, legends, and characters are to be found. *Scene processing*, where a target is to be detected and *quality inspection of materials*, where defective parts must be delineated. Other applications can be listed as follows: *cell images* and *knowledge representation*, *ultrasonic images*, *thermal images*, *x-ray computed tomography*, *CAT endoscopic images*, *laser scanning*, *extraction of edge field*, *image segmentation in general*, *spatio-temporal segmentation of video images*, ...

Various factors, such as non stationary and correlated noise, ambient illumination, distribution of grey tones within the object and its background, inadequate contrast, and object size not commensurate with the scene complicate the thresholding operation.

Though thresholding algorithms assume no preprocessing, enhancing images by applying preprocessing techniques on them is always worth the effort. This way, interesting features in images may be enhanced or differences between background and foreground pixels may be increased. Also reducing noise in the image may be helpful when thresholding it. There are a lot of preprocessing techniques which can help achieving better results in the segmentation step by removing noise, enlarging contrast

and so on.

Working under uncontrolled light conditions adds more difficulties to the segmentation process. As different light configurations may be found in the images to be processed, the task involves not only finding the most suitable algorithm for the image but also, finding which parameters, if any, have to be tuned in order to achieve best results.

Most segmentation algorithms proposed in the literature base their work on suppositions about light distribution on an image. Most of them expect that light is smoothly distributed on the captured surface and thus, different regions are easily distinguished by trying to find differences in the colour or grey levels of pixels. This is not always the case if the image has been taken outdoors. In this case, sunlight, clouds and other natural events may interfere with the light arriving to the scene and induce shadows and artefacts in the captured image. In this circumstances, the supposition of smooth light distribution may not hold for the entire image.

The lack of objective measures to assess the performance of various thresholding algorithms, and the difficulty of extensive testing in a task-oriented environment, are other major handicaps. It is true, that thresholding algorithms which perform well in a certain problem will not behave that good in another, for instance, algorithms that apply well for document images are not necessarily the good ones for medical images, and vice versa, given the different nature of document and medical images.

We focus our attention on algorithms aimed to segment regions in any kind of images without previous constraints. In this case, the problem is finding areas in the images which may be of interest for further processing. The major difficulty affecting this kind of images is that only information found in the image may be used to segment it; so it is crucial that algorithms are well tuned, in order to achieve good results. In the following sections, five different algorithms for image segmentation are introduced which are well known in the literature. They, and variants of them, are widely used to segment images depending only on the relationships of each pixel with their neighbours.

In next section, notation and definitions are discussed. Some preprocessing techniques are introduced in 2.3. Finally, in section 2.4 some segmentation algorithms are explained.

2.2 Notation and definitions

The input data for these segmentation algorithms will be images denoted by I. There are several mathematical definitions that can formally describe images, for our purposes, we will define an image as a function of pairs of coordinates (x, y) which define a position in the plane associated with the scaled scene represented in the image. In the case of a grey tone image, this definition can be written as,

$$I : \{\mathbf{N} \times \mathbf{N} \rightarrow \{0...255\}\} \tag{2.1}$$

Colour and multispectral images are arrays of grey tone images and may be defined as,

$$I : \{\mathbf{N} \times \mathbf{N} \rightarrow \{0...255\}^c\} \tag{2.2}$$

being c the dimensions of the image I.

Coordinates of a pixel (x, y) define a pixel. These coordinates may be summarized as p for brevity, and so will done in following sections whenever pixel coordinates are meaningless.

Previous definitions however, do not reveal the neighbouring properties of the pixels of the image. We define a graph $G = (V, E)$, being V a set of *nodes* and E a set of *arches* connecting nodes. G is associated to the image I, in such a way that nodes in V correspond to pixels in I and E is the set of arches connecting neighbouring pixels in I.

With these definitions, we can define the set of 8-*connected neighbours*, denoted by N_8, of a pixel p as,

$$N_8(p) = \{q \in V/(p, q) \in E\} \tag{2.3}$$

Abusing notation, $I(x, y)$ will refer both to the element in I with coordinates (x, y) and to the node in G with the same coordinates.

The segmentation process aims to divide the image into regions according to any given criterion. The output of a segmentation process is always expected to be a classification of pixels, and extensively, of the regions to which these pixels belong to in the input image, either as foreground or as background. Pixels classified as foreground are expected to be of interest for further steps of the vision system; background pixels, on the other side, are discarded.

The output of a segmentation algorithm will always be a binary image S of the same dimensions as I defined as,

$$S(I) : \{\mathbf{N} \times \mathbf{N} \rightarrow 0, 1\} \tag{2.4}$$

Where $S(x, y) = 1$ if the pixel $I(x, y)$ has been classified as foregound and $S(x, y) = 0$ if the pixel has been classified as background. The regions are then sets of pixels or subgraphs of image S.

As pointed out before, in the segmentation of an image we are not just interested on individual pixels, but on the groups they form, which may be, or not, relevant areas for further processing. We define a *region* R as a set of connected pixels of S, which are classified as foreground. These regions are known as *blobs*.

Connectivity in a graph is defined as follows: two nodes in a graph G are connected if each pair of nodes of V that belong to the path, are joined by an arch of E. Two different pixels belong to the same region if there is a path that connects them.

By path from p to $q \in S$ it is understood a set of nodes $\delta(p, q) = \{z_0, z_1...z_k\} \in S$ such that each point $z_i \in N_8(z_{i-1}), 0 \leq i \leq k$, being $z_0 = p$ and $z_k = q$.

The following equation defines a region R as a set of connected pixels,

$$R = \{p_0, p_1...p_n : S(p_i) = 1 \wedge \ \delta(p_j, p_i) \neq \emptyset, \forall\ 0 \leq j, i \leq n \wedge i \neq j\} \tag{2.5}$$

The intersection of regions will be empty,

$$R_i \cap R_j = \emptyset \iff i \neq j \wedge R_i, R_j \subset S(I) \tag{2.6}$$

Let's suppose we have two regions R_i and R_j which share a point p and $i \neq j$. By the definition of region, we would then have a path from p to every point $q \in R_i$. On the other side, there would be a path from p to any point $v \in R_j$. This means, that we can connect any point $q \in R_i$ to any point $v \in R_j$ with paths $\delta(q, v) = \delta(q, p) \ \cup \ \delta(p, v)$,

which is the definition of region given before, so both regions R_i and R_j are the same and thus $i = j$.

Also, the union of all regions R_i will cover, partially or completely, the original image,

$$\bigcup R_i \subseteq S(I), \forall i \tag{2.7}$$

2.3 Preprocessing techniques

Preprocessing is a common name for operations with images at the lowest level of abstraction; it is really a set of techniques whose aim is improving in some sense the input image to make it easier to process by the segmentation step.

There are four categories of image preprocessing methods according to the size of the pixel neighbourhood that is used for the calculation of a new pixel intensity,

- pixel brightness transformations
- geometric transformations
- preprocessing methods that use a local neighbourhood of the processed pixel
- image restoration that requires knowledge about the entire image

Though the list of techniques is very large and a complete description is out of the scope of this thesis, there are some which are worth to be mentioned because they are used later through this work.

2.3.1 Sobel operator.

The Sobel operator [gonzalez93], is an edge detection algorithm. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each pixel in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations.

In simple terms, the operator calculates the gradient of the image intensity at each point, giving the direction of the largest possible increase from light to dark and the rate of change in that direction. The result shows how "abruptly" or "smoothly" the image intensity changes at that pixel, and therefore how likely it is that part of the image represents an edge, as well as how that edge is likely to be oriented. In practice, the magnitude (likelihood of an edge) calculation is more reliable and easier to interpret than the direction calculation.

Mathematically, the gradient of a two-variable function is at each point a 2D vector with the components given by the derivatives in the horizontal and vertical directions. At each image point, the gradient vector points in the direction of largest possible intensity increase, and the length of the gradient vector corresponds to the rate of change in that direction. This implies that the result of the Sobel operator at an image point which is in a region of constant intensity is a zero vector and at a point on an edge is a vector which points across the edge, from darker to brighter values.

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives, one for horizontal changes, and one for vertical. If we define I as the source image, an estimation of the horizontal (∇_x) and the vertical (∇_y) derivatives in each point is obtained as follows,

$$\nabla_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \oplus \mathbf{I} \quad \text{and} \quad \nabla_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \oplus \mathbf{I} \tag{2.8}$$

where $\oplus$ here denotes the 2-dimensional convolution operation.

At each point in the image, the resulting gradient approximations is given by the vector

$$\nabla(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \nabla_x(x, y) \\ \nabla_y(x, y) \end{bmatrix} \tag{2.9}$$

and its gradient magnitude can be easily obtained as,

$$|\nabla| = \sqrt{{\nabla_x}^2 + {\nabla_y}^2} \tag{2.10}$$

Using this information, we can also calculate the gradient's direction,

$$\alpha(\nabla) = \arctan\left(\frac{\nabla_y}{\nabla_x}\right) \tag{2.11}$$

2.3.2 Top-hat transform.

Top-hat operator ([gonzalez93],[soille99]), is useful for enhancing details in complex backgrounds. This operator transforms a grey-level image into a binary image by using a rectangular structural element, and the morphological operations of *closing* and *opening*. See appendix A for more details about morphological operators and structural elements.

This technique is really two techniques into one, as it may be applied either to enhance clear or dark areas, depending on the application. In the case clear regions have to be enhanced, the original image is subtracted to the result of applying the opening operator to the own image with the structural element. In the other case, if dark regions have to be enhanced, then it is the original image which is subtracted from the closing of the image with the structural operator. Both the operations of opening and closing in the case of grey-level images are defined as,

$$\phi(I(x, y)) = min_{(i,j)\in b}(max_{(i,j)\in b}(I(x + i, y + j) + b(i, j))) \tag{2.12}$$

$$\omega(I(x, y)) = max_{(i,j)\in b}(min_{(i,j)\in b}(I(x + i, y + j) - b(i, j))) \tag{2.13}$$

Bearing in mind previous definitions, top-hat for clear regions, called "white top-hat", may be expressed mathematically as,

$$WTH(I) = I - \omega(I)$$

and for dark regions, called "black top-hat" as,

$$BTH(I) = \phi(I) - I$$

It must be noted, that in the case the wrong top-hat is chosen, the significant areas of the image are wrongly preprocessed. For instance, figure 2.1 shows a sample image preprocessed with both top-hat techniques. Figure 2.1c shows the result of choosing the wrong top-hat operator, the significant areas of the image are lost.

(a) Original image (b) White top-hat (c) Black top-hat

Figure 2.1: Sample execution of top-hat technique. a) An image I representing a truck container. b) Final image ($WTH(I)$) looking for clear areas. c) The same for dark areas ($BTH(I)$).

2.4 Segmentation algorithms

Several approaches to image segmentation are available in the literature. One class of algorithms are the thresholding methods, which seek for a valid threshold, either global or local, to separate foreground regions from background regions. Otsu's method [otsu79], tries to calculate a global threshold for the complete image. A sample of a segmentation algorithm calculating local thresholds is LAT, acronym of local variation algorithm, [kirby79]. Both are discussed later.

Another approach are the growing regions methods; these grow regions using seeds in the image as basis. There are several ways of choosing these seeds, either manually or automatically. In the local variation algorithm [felzen98], authors propose using the graph interpretation of the image and choose as seeds those groups of pixels whose difference in grey tones are zero. Watershed algorithm for instance, [beucher79], seeks for pixels with the lowest (or highest) grey tone and takes these pixels as seeds.

Figure 2.2 shows the taxonomy of segmentation algorithms with the sample algorithms chosen. Growing regions algorithms with supervised seeds were not considered in our developments, due to the fact, that the goal was obtaining a method that did not need human supervision.

In this section we describe the algorithms used to segment static images and we give a formal introduction to each one. These algorithms are: the global thresholding algorithm [gonzalez93], otsu's method [otsu79], LAT [kirby79], the watershed algorithm [beucher79] and local variation algorithm [felzen98].

2.4.1 Global thresholding

Global thresholding [gonzalez93] is the simplest approach to image segmentation. Foreground pixels are assumed to have different colours or grey tones to background values, so a threshold is supposed to be enough to separate both classes. Different approaches may be found in the literature with this idea in the basis, either for colour images as for grey tone images.

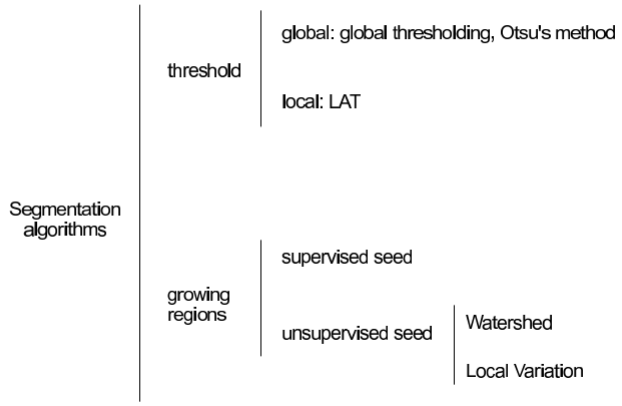
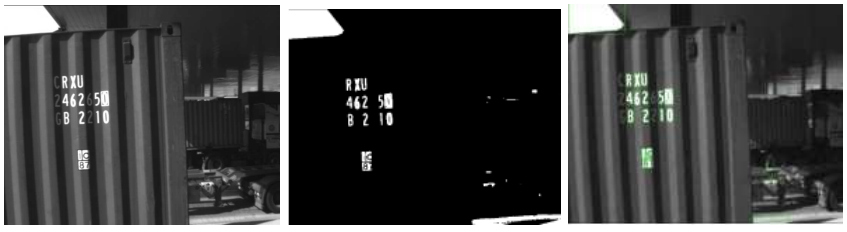


Figure 2.2: The taxonomy of segmentation algorithms used in this work. Classification is performed from the point of view whether thresholds or growing regions techniques are used. In the case of using thresholds, these may be local or global. For each approach, a sample algorithm representative of the approach is named.

The basic version of the algorithm works taking as input an image I and the output is a binary image $S(I)$ which represents the segmentation of I. In a single pass, each pixel in the image is compared with a given threshold T. If the pixel's intensity $I(x, y)$ is higher than or equal to the threshold T, the pixel $S(x, y)$ is set to foreground in the output. If it is under T, then $S(x, y)$ is set to background. In this case, segmented image $S(I)$ is made up of pixels which,

$$\forall (x, y) \in I, S(x, y) = \begin{cases} 1, \ if \ I(x, y) \geq T \\ 0, \ otherwise \end{cases} \tag{2.14}$$

A sample execution of this algorithm for a complete image, may be found in figure 2.3. One of the drawbacks of this algorithm is the election of the parameter T; which can be made by using the Otsu's method described in section 2.4.2.



(a) I (b) $S(I)$ (c) Bounding boxes limiting the connected regions represented over the original image

Figure 2.3: Figure 2.3a represents an image of a truck container. The result of applying global thresholding to this image with $T = 170$ is shown in figure 2.3b. Figure 2.3c shows the connected regions R_i over the original image. This illustrates the need of gathering foreground pixels into regions in the segmentation result.

2.4.2 Otsu's method

Otsu's method ([otsu79]) is used to automatically perform histogram shape-based image thresholding or, the reduction of a grey-tone image to a binary image. The algorithm assumes that the image to be thresholded contains two classes of pixels (e.g. foreground and background) and calculates the optimum threshold separating those two classes so that their within-class variance is minimal. The extension of the original method to multi-level thresholding is referred to as the multi Otsu method [huang09].

In Otsu's method we exhaustively search for the threshold that minimizes the within-class variance, defined as a weighted sum, $\sigma_w^2(T)$, of variances of the two classes,

$$\sigma_w^2(T) = \omega_1(T)\sigma_1^2(T) + \omega_2(T)\sigma_2^2(T) \tag{2.15}$$

Weights ω_i a are the probabilities of the two classes separated by a threshold T and σ_i^2 the variance of class i. Otsu shows that minimizing the within-class variance is the same as maximizing between-class variance,

$$\sigma_b^2(T) = \omega_1(T)\omega_2(T)[\mu_1(T) - \mu_2(T)]^2 \tag{2.16}$$

which is expressed in terms of class probabilities ω_i and class means μ_i which in turn can be updated iteratively. This idea yields an effective technique. The algorithm starts by computing the image histogram and the probabilities of each intensity level. The algorithm steps through all possible thresholds from 1 to the maximum intensity. At the end, the desired threshold T corresponds to the maximum $\sigma_b^2(T)$. The image segmentation $S(I)$ is calculated then using the computed threshold as in equation 2.14.

2.4.3 Local Adaptive Thresholding (LAT)

This algorithm was proposed by Kirby and Rosenfeld in [kirby79]. LAT is an acronym which stands for Local Adaptive Thresholding. For each pixel $p \in I$ a threshold $\mu(p)$ is calculated. If the pixel value is below the threshold it is classified as background, otherwise it is classified as foreground; or vice versa. Authors based their work on the supposition that neighbouring pixels would be similarly illuminated.

The algorithm uses the neighbourhood of pixel p, $N_m(p)$, to calculate statistics to examine the intensity $I(p)$ of the local neighbourhood of each p. This neighbourhood is usually a square window of $m = n \times n$ elements. In most cases, the computed statistic is the mean of the local intensity, computed as,

$$\mu(p) = \frac{1}{O(N_m(p))} * \sum_{q \in N_m(p)} I(q) \tag{2.17}$$

The size of the considered neighbourhood is important, it has to be large enough to cover sufficient foreground and background pixels, otherwise a poor threshold is chosen. On the other hand, choosing regions which are too large can violate the assumption of approximately uniform illumination.

A constant factor c can be used to adjust the comparison of the mean with the pixel intensity value. This factor multiplies the grey level of the pixel when comparing it to the mean grey level of its neighbourhood.

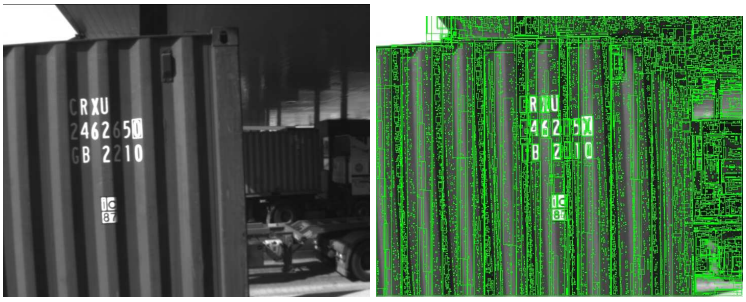
On the other hand, a level term l can be added to eliminate salt and pepper noise.

Input image I is then converted into output image $S(I)$ by applying the following formula,

$$\forall p \in I, S(p) = \begin{cases} 1, \ if \ \mu(p) \geq c \cdot I(p) + l \\ 0, \ otherwise \end{cases} \tag{2.18}$$

Three parameters can be adjusted in this algorithm, on one hand, c which usually takes values around 1; l which takes values close to 0 and the size of the neighbour window n, which, as mentioned before, must be set to be big enough as to cover sufficient foreground and background pixels but small enough as not to violate the assumption of uniform illumination.

A sample execution of this algorithm for a complete image, may be found in figure 2.4.



(a) Original image

(b) Connected regions found after LAT thresholding

Figure 2.4: Sample execution of LAT segmenting an image representing a truck container.

2.4.4 Watershed

In [beucher79] and [beucher91] the application of the watershed transform to image segmentation was proposed. The algorithm takes a grey scale image and considers it as a topographic surface. A process of flooding is simulated on this surface; during this process two or more floods coming from different basins may merge. To avoid this, dams are built on the points where the waters flooding from different basins meet; at the end of the algorithm, only dams are over water level. These dams define the watershed of the image. The way this algorithm works and how it transforms image I into a set of segmented regions $S(I)$, is as follows.

Let l be the current grey tone under examination, being the first one $l = 0$. We initialize the region set as,

$$S(I)^{-1} = \emptyset \tag{2.19}$$

The algorithm floods each time a grey level of the image by increasing l up to $L-1$, being L the total amount of grey levels,

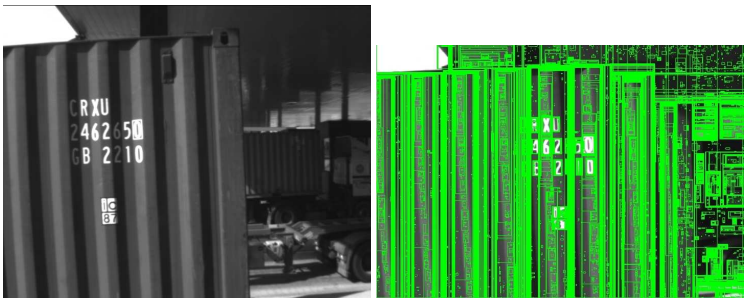
$$\forall (x, y) \in I, \ if I(x, y) \leq l \rightarrow I(x, y) = l \tag{2.20}$$

After this flooding step, the algorithm searches for connected regions R_l. Being $S(I)^l$ the set of all the connected regions found in flooding level l.

In each new iteration, the algorithm checks whether regions in previous level have joined in the current level,

$$if\ \exists\ R_i \in S(I)^{l-1} / \wedge R_j \in S(I)^l\ R_i \subset R_j \rightarrow S(I)^l = S(I)^l \bigcup R_i \tag{2.21}$$

$S(I)^{L-1}$, would be the final segmentation. A sample execution of this algorithm for a complete image, can be found in figure 2.5.



(a) Original image I (b) Different flooding levels Watershed

Figure 2.5: Sample execution of Watershed segmenting an image representing a truck container. Image 2.3b represents the different flooding levels of Watershed algorithm over the original image.

2.4.5 Local variation

This algorithm was introduced by Felzenswalb and Huttenlocher in [felzen98]. Its approach consists in considering a criterion for segmenting images based on intensity differences between neighbouring pixels. Figure 2.6 shows a sample execution of this algorithm on an image representing a truck container.

The main idea is partitioning an image into regions, such that for each pair of regions the variation between them is bigger than the variation inside each region. The measure of the internal variation of a region is a statistic of the colour or intensity differences between neighbouring pixels in the region. The measure of the external variation between two regions R_i and R_j is the minimum colour or intensity difference between two neighbouring pixels $p \in R_i$ and $q \in R_j$ along the border of the two regions.

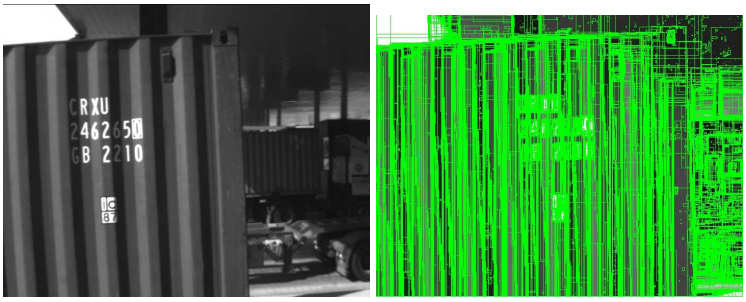
The algorithm uses two parameters, the minimum size of the regions in the final result, and a constant used to smooth the image before processing it.

The algorithm starts by creating a graph that represents the image. This graph follows the structure introduced in section 2.2, with the particularity that arches in the graph are given a weight that corresponds to the difference in intensity of pixels represented by their nodes. The function used to calculate the weight of the arches is defined as follows,

$$w(v_i, v_j) = \begin{cases} |I(v_i) - I(v_j)|,\ if(v_i, v_j) \in E \\ \infty,\ otherwise \end{cases} \tag{2.22}$$

Arches are ordered by non-decreasing weight. To achieve the fastest ordering, authors of [felzen98] recommend in their paper the bucket sort algorithm [cormen90].

First step consists in taking arches between pixels v_i, v_j such that $w(v_i, v_j) = 0$. These pixels determine the seeds used to grow regions. The algorithm then takes an arch at a time and compare the regions it joins. Both regions will be merged if they meet the established criteria. The output of the algorithm is a set of regions in which the image is segmented.



(a) Original image

(b) Connected regions found after local variation thresholding

Figure 2.6: Sample execution of the local variation algorithm segmenting an image representing a truck container.

2.5 Conclusions

Several algorithms exist to segment images depending only on local features of the image, as relationships of pixels with their neighbours, for instance. In this chapter we have introduced well-known algorithms which are extensively used to segment images in a wide range of applications. These techniques were chosen based on previous knowledge of the algorithms and also on the comparison of the literature describing them.

As long as algorithms designed to segment images rely heavily on the light conditions, images taken under non controlled light conditions become difficult to deal with.

It must be noted, that no specific mechanism exist, which may be used to test the performance of the segmentation algorithms. In chapter 4 a systematic method for testing the performance of segmentation algorithms is proposed.

Chapter 3

Background modelling and motion segmentation

This chapter introduces the most representative techniques applied to background modelling and motion segmentation available in the literature. Special attention is paid to background subtraction and background modelling methods which are used in the developments discussed in chapter 5.

3.1 Introduction

Visual analysis of human motion [wang03] is currently one of the most active research topics in computer vision. This strong interest is driven by a wide spectrum of promising applications in many areas such as *virtual reality*, *smart surveillance* and *perceptual interface*, just to mention the most representative.

Visual analysis concerns the detection, tracking and recognition of objects in general, and particularly, people; and the understanding of human behaviour in the case of image streams involving humans. Visual analysis of a scene starts from a segmentation of the scene in order to classify pixels as foreground or background; then, other steps may be taken depending on the application, such as motion analysis, object detection, object classification, human tracking, action recognition and semantic decision. Figure 3.1 shows the general framework of human motion analysis proposed in [wang03].

Human motion analysis has been investigated under several large research projects worldwide. For example, DARPA (Defence Advanced Research Projects Agency) funded a multi-institution project on Video Surveillance and Monitoring (VSAM) introduced in [vsam99], whose purpose was to develop an automatic video understanding technology that enabled a single human operator to monitor activities over complex areas such as battlefields and civilian scenes.

In another project, the real-time visual surveillance system W4, an acronym for Who? Where? What? When? is introduced in [haritaoglu00]. It employed a combination of shape analysis and tracking, and constructed the models of people's appearances to make itself capable of detecting and tracking multiple people as well as monitoring their activity. Other examples of similar systems may be found in [hu04] and [wren97].

Visual analysis algorithms do usually assume a fixed camera without motion, though some extensions to moving cameras are available in the literature [marcenaro00]. In general, they take as input an image, or frame, from a video stream. These frames are

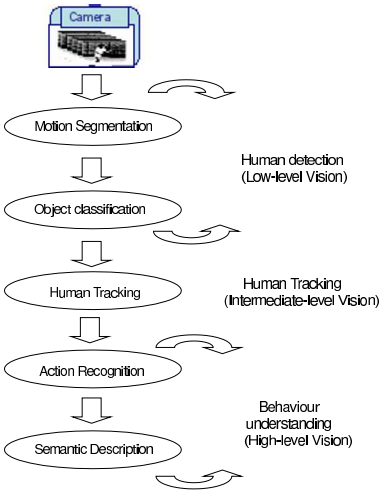


Figure 3.1: Stages in the process of human motion analysis.

time shifted representations of what has happened in the scene under analysis; so a temporal relationship between consecutive frames can be established.

Illumination changes produced by weather, indoor illumination, shadow and repetitive motion from clutter, make motion segmentation difficult to process quickly and reliable. Thus, methods must be implemented to distinguish carefully which regions must be processed and which not.

Currently, most segmentation methods use either temporal or/and spatial information of the images. Several general approaches used in motion segmentation are outlined in the following paragraphs: *temporal differencing*, *background subtraction*, *statistical methods* and *optical flow*. The final election depends greatly on the application and the constraints to be met; for instance, for real-time applications it is totally discouraged the use of optical-flow techniques which are very time-consuming.

We centre our interest in segmenting objects and distinguish them from the background of the scene. Following section introduces notation and definitions used further in the chapter. Background modelling algorithms, which are crucial if background subtraction is used, are explained in section 3.3. Section 3.4 discusses motion segmentation techniques. Finally, some techniques used to remove shadows are described in section 3.5.

3.2 Notation and definitions

Visual analysis algorithms take as input a video stream which may be described as a set of images or frames, $F(0), F(1), ..., F(i)$, which represent the activity in a given scene, being i the order of the i-th frame in time. The time difference between two consecutive frames is known and constant with a value of $\frac{1}{fps}$ seconds, where fps stands for *frames per second*.

Segmenting each frame $F(i)$ and stablishing relationships between consecutive frames is known to be a difficult problem. Recalling the definitions introduced in section 2.2, regions R_j, detected in the segmentation of each $F(i)$, provide a focus of attention for later processes, such as tracking or activity analysis, because only those pixels belonging to these regions need to be considered.

In order to segment frames, different approaches are available as will be further explained in following sections. Some techniques, introduced in section 3.4.2, use a model of the scenario, called background model, to perform their computations.

A general definition of background model B in grey tones as a single image is given by equation 2.1. In some cases, however, each pixel may be described with more than one model. In general, a model can be seen as a set of overlaid images as,

$$B = \{B^0, B^1, ..., B^m\} \tag{3.1}$$

where each B^i corresponds to the definition given above and the superindex i indicates which description it refers to. $B^m_{x,y}$ will refer to the m-model of pixel in location (x, y). It is straightforward to see that this definition includes the previous with just setting $m = 1$.

The background model can be updated with a given period T. Being $B(i)$ the background model updated in time i.

Each pixel in the input frame must be compared to at least, one background model for it. This means that the following condition must always hold,

$$\forall x, y, \in F_{x,y}(i) \rightarrow \exists\, B_{x,y}(i)^m, i \geq 0\ , m \geq 1 \tag{3.2}$$

The goal of segmenting scenes is finding objects, shadows and background of the scene. Some important and precise definitions about what it is considered to be an object, a shadow or background of a scene may be found in [shoushtarian05] and [xu05] and are reproduced below.

- *Moving object*. A set of connected points in the input image which in comparison with the static camera, are currently characterised by non-null motion and a different visual appearance from the background

- *Static object*. A set of connected points in the input image which, in comparison with the static camera, are currently characterised by null motion but show a different visual appearance from the background

- *Ghost*. A set of connected points in the input image detected as an object but not corresponding to any real object

- *Shadow*. An area of the background on which light has been reduced by locating an object which avoids light to arrive to it. In figure 3.2, some sample blobs with and without shadows, along with the original grey-tone image may be found. It may be remarked the effect of shadows in the segmentation of objects; joining objects or enlarging objects dimensions. In [xu05] shadows are distinguished in two different types:

  - *Cast-shadows*. Area of the background projected by the object in the direction of the light rays producing inaccurate silhouettes
  - *Self-shadows*. Parts of the object which are not illuminated. They are usually part of the silhouette and thus, pointless from the point of view of a shadow removal scheme should not remove them

- *Highlight*. Areas of exceptional lightness in the input image.

- *Background*. All pixels in the input image which belong neither to moving objects nor ghosts. The former represents the current set of pixels which are not considered for further analysis. On the other hand, the background model describes what the background of the scene is expected to be.

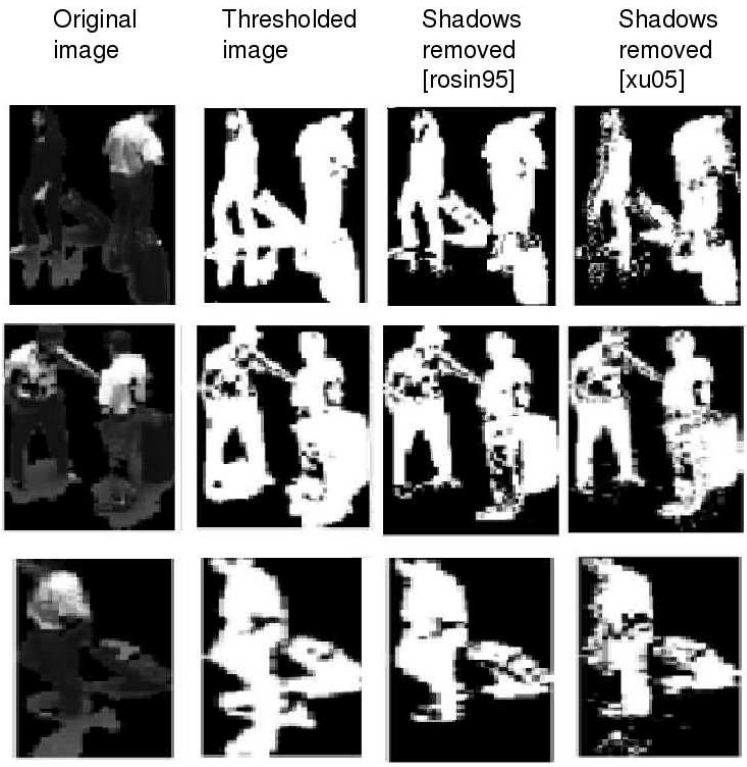


Figure 3.2: Sample blobs with and without shadows, along with the original images. Each row corresponds to a different sample. First column shows the original image, second column to the thresholded figure, including cast shadows. The third column represents the result of applying the shadow removal algorithm introduced in [rosin95] and the fourth column, the result of applying the technique discussed in [xu05]. Both techniques are introduced in section 3.5

3.3 Background modelling

The construction and updating of background models is indispensable to visual surveillance. In the literature, techniques either for 2-D or 3-D background modelling can be found. Due to their simplicity, 2-D techniques have more applications. In fact, current work with 3-D background modelling techniques is limited due to the difficulty of 3-D reconstructions in outdoor scenes.

In any case, different scenarios may appear when building background models, depending on whether the camera stays in a place or moves. In this chapter, we will consider only techniques in which the camera is stationary.

According to the discussion made in [toyama99], an ideal background maintenance

system would be able to avoid the following problems:

- *Moved objects*: A background object can be moved. These objects should not be considered part of the foreground forever after.
- *Time of day*: Gradual illumination changes alter the appearance of the background.
- *Light switch*: Sudden changes in illumination and other scene parameters alter the appearance of the background.
- *Waving trees*: Backgrounds can vacillate, requiring models which can represent disjoint sets of pixel values.
- *Camouflage*: A foreground object's pixel characteristics may be subsumed by the modelled background.
- *Bootstrapping*: A training period absent of foreground objects is not available in some environments.
- *Foreground aperture*: When a homogeneously coloured object moves, change in the interior pixels cannot be detected. Thus, the entire object may not appear as foreground.
- *Sleeping person*: A foreground object that becomes motionless cannot be distinguished from a background object that moves and then becomes motionless.
- *Waking person*: When an object initially in the background moves, both it and the newly revealed parts of the background appear to change.
- *Shadows*: Foreground objects often cast shadows which appear different from the modelled background.

In their work, authors of [toyama99] propose a very useful benchmark which challenges background algorithms with the mentioned situations. This benchmark is widely accepted by the community as a valid test and we will also use it when comparing our proposals for background modelling with algorithms found in the literature in chapter 5.

In [piccardi04] and [benezeth08] a review of background modelling techniques and a comparison among them may be found. Both papers discuss similar results regarding performance and agree that no method outperforms the rest in all aspects.

The background subtraction technique used, should be carefully chosen according to the scene where action will take place. Moreover, authors of [benezeth08] point out that, when dealing with videos under the fundamental background subtraction assumption, i.e., fixed camera and static and noise-free background, the basic background subtraction will perform reasonably good, and no better results may be expected from other, more complicated, methods.

The techniques proposed in the literature show a fundamental difference in the process followed to obtain the initial background model $B(0)$, and also in the criteria followed to decide when and how creating a new model for a given pixel.

According to the reviewed literature, authors propose in their papers techniques for background modelling with the capability of adapting the model to changes in the scene background. Moreover, each pixel in the background may be represented by more than one model.

When background subtraction techniques are used, background update in time i depends on $B(i-1)$, the previous model obtained in time $i-1$, and on the difference computed between the current frame $F(i)$ and model $B(i-1)$. Usually, this update operation of a model is given by the following expression:

$$B(i) = \alpha \cdot B(i-1) + (1-\alpha) \cdot F(i) \tag{3.3}$$

being α an update factor in the range $[0, 1]$, which controls the speed at which new information is included in the model.

Tables 3.1 and 3.2 summarize information of the most popular techniques for background modelling found in the literature. In table 3.1, columns show at which level the background model is computed, pixel or region level; if multi-modal support is considered in the original algorithm, the segmentation method and if the algorithm may work with grey tone images, colour images or may deal with both. Table 3.2 shows the spatial complexity of each algorithm following the big O notation considering an image of size $n \times m$, the capability of the algorithms to detect or react against model corruption and a reference paper for each algorithm.

Algorithm.	Granularity	Multimodal	Segmentation method	Color or grey tones
Adjacent frame difference	pixel	No	frame differencing	both
Running Gaussian average	pixel	No	background subtraction	both
Mixture of Gaussians	pixel	Yes	background subtraction	both
Kernel density estimation	pixel	Yes	background subtraction	both
Sequential kernel density approx.	pixel	Yes	background subtraction	both
Temporal median filter	pixel	No	background subtraction	both
Eigenbackgrounds	region	No	background subtraction	both
Cooccurrence of image variations	region	No	background subtraction	both
Textures	pixel	Yes	background subtraction	grey tone
Wallflower	pixel	Yes	background subtraction	grey tone
Edges histogram	region	No	background subtraction	both
Salient motion	pixel	No	optical flow	both

Table 3.1: Features of the most popular techniques for background modelling found in the literature depending on which the background model granularity is, their multi-modal support capability, whether they support grey tone images, colour images or both, and the motion segmentation method employed.

Algorithm.	Spatial complexity	Model corruption detection	References
Adjacent frame difference	$O(1)$	-	-
Running Gaussian average	$O(1)$	No	[wren97]
Mixture of Gaussians	$O(Knm)$	No	[grimson99]
Kernel density estimation	$O(Knm)$	No	[elgammal00]
Sequential kernel density approx.	$O(Knm)$	No	[han04], [piccardi04a]
Temporal median filter	$O(Knm)$	No	[yang92]
Eigenbackgrounds	$O(Knm)$	No	[oliver00]
Cooccurrence of image variations	$O(Knm/N^2)$	No	[seki03]
Textures	$O(Knm/2^P)$	No	[heikkila06] , [heikkila04]
Wallflower	$O(nmV)$	Yes	[toyama99]
Edges histogram	$O(mn)$	No	[mason01]
Salient motion	$O(mn)$	-	[wixson00]

Table 3.2: Other important features of background modelling techniques, the spatial complexity of each algorithm following the big O notation considering an image of size $n \times m$, the capability of the algorithms to detect or react against model corruption and a reference paper for each algorithm.

The spatial complexity of the Eigenbackgrounds depends on K the number of im-

ages chosen to compute the background model. In the case of cooccurrence of image variations, the memory complexity is $O(nmK/N^2)$ where K is the number of variations in the training model and nm their dimensions. At classification time, the eigenbackground method requires a memory complexity per pixel $O(M)$, with M the number of the best eigenvectors. However, at training time the method requires allocation of all the K training images, with an $O(Knm)$ complexity. LBP complexity depends on 2^P the size of each histogram, being P the number of equally spaced pixels used to compute each LBP feature. Wallflower complexity only takes into account the model's complexity, as no model weights are considered in this algorithm. Its complexity is then $O(n \times m \times V)$, where V is the sum of the number of coefficients and the number of past values used to predict future values with the Wiener filter. Wallflower considers a fixed number of models.

The problem of model corruption is discussed in very few papers; in [toyama99] it is proposed to maintain a database with valid background models which may be chosen for each situation. The background may become corrupt for several different reasons, being sudden changes of light the most usual. It is not an easy problem to solve as it is not trivial determining when a model becomes corrupt and, when it does, recovering it is very difficult even if models are stored in a database, as all stored models have to be compared with current scenario and the best match is then chosen to keep the system up and running.

In the following subsections, each of the techniques referenced in previous tables, is briefly introduced.

3.3.1 Adjacent frame differencing

$$J(i) = |F(i) - F(i-1)| \tag{3.4}$$

Regions of interest are detected by using a single threshold T ,

$$S_{x,y}(i) = \begin{cases} 1, & J_{x,y}(i) \geq T \\ 0, & J_{x,y}(i) < T \end{cases} \tag{3.5}$$

This method is very sensitive to small light changes, generates a noisy segmentation and does not find complete objects.

3.3.2 Running Gaussian average

Authors of [wren97] propose modelling the background independently at each location (x, y). In this technique, each background pixel $B_{x,y}(i)$ is independently modelled, trying to estimate its average value on the last n frames. In order to avoid fitting a gaussian distribution by using the last n values of each previous frame, each time a new frame $F(i)$ is acquired in time i a cumulative average $B_{x,y}(i)$ is computed as stated in equation 5.2, that is, .

$$B_{x,y}(i) = \alpha B_{x,y}(i-1) + (1-\alpha) F_{x,y}(i) \tag{3.6}$$

The learning factor α used in this equation is an empirical weight often chosen as a trade-off between stability and quick update. The advantage of this method is its simplicity and low memory requirements.

Pixels in $F(i)$ can then be classified as a foreground pixel if the following inequality holds,

$$|F(i) - \mu(i)| > k\sigma(i) \tag{3.7}$$

where k is a parameter set by user and $\sigma(i)$ the computed standard deviation per pixel is computed as,

$$\sigma^2(t) = \alpha(F(i) - B(i-1))^2 + \sigma^2(i-1)(1-\alpha) \tag{3.8}$$

Pixels for which the previous inequality does not hold are classified as background.

The model was initially proposed for grey tones images, but the extension to RGB coordinates or other colour spaces is quite straightforward.

The main disadvantage of this method is that it cannot handle properly scenarios with clutter in the background and that the value of the α parameter is arbitrarily chosen.

3.3.3 Mixture of Gaussians (MoG)

While previous models will adapt different background objects appearing over time, sometimes the changes in the background are not permanent and appear at a rate faster than that of the background update. A typical example is an outdoor scene with waving trees partially covering a building, for instance. The same pixel in location (x, y) will show values from the tree branches, tree leaves and the building. In [grimson99], authors raised the case for a multi-valued background model able to cope with multiple background objects.

These are the guiding factors in the choice of model and update procedure in this algorithm. The recent history of each pixel, $F_{x,y}(1), ..., F_{x,y}(i)$, is modelled by a mixture of K Gaussian distributions. The probability of observing the current pixel value is,

$$P(F_{x,y}(i)) = \sum_{j=1}^{K} \omega_j(i) \cdot \eta(B_j(i), F_{x,y}(i), \Sigma(j)(i)) \tag{3.9}$$

where K is the number of distributions, $\omega_j(i)$ is an estimate of the weight (what portion of the data is accounted for by this Gaussian) of the j-th Gaussian in the mixture at time t, $B_j(i)$ is the mean value of the j-th Gaussian in the mixture at time i, $\Sigma_j(i)$ is the covariance matrix of the j-th Gaussian in the mixture at time i, and η is the Gaussian probability density function,

$$\eta(B_j(i), X_i, \Sigma_j(i)) = \frac{1}{2\pi|\Sigma_j(i)^{\frac{1}{2}}|} e^{-\frac{1}{2}(X_i - B_j(i))^T \Sigma_j(i)^{-1}(X_i - B_j(i))} \tag{3.10}$$

K is determined by the available memory and computational power. Also, for computational reasons, the covariance matrix is assumed to be of the form,

$$\Sigma_j(i) = \sigma^2 \cdot F \tag{3.11}$$

being σ the variance of the i-th model. If the pixel process could be considered a stationary process, a standard method for maximizing the likelihood of the observed data is expectation maximization (EM). Unfortunately, each pixel process varies over time as the state of the world changes, so they use an approximate method which essentially treats each new observation as a sample set of size 1 and uses standard learning rules to integrate the new data.

Because there is a mixture model for every pixel in the image, implementing an exact EM algorithm on a window of recent data would be costly. Instead, authors implement an on-line K-means approximation. Every new pixel value, X_t , is checked against the existing K Gaussian distributions, until a match is found. A match is defined as a pixel value within 2.5 standard deviations of a distribution, see equation 3.12

$$\frac{(x_t - B_j, (i))}{\sigma_j(i)} > 2.5 \tag{3.12}$$

This threshold can be perturbed with little effect on performance. This is effectively a per pixel/per distribution threshold. This is extremely useful when different regions have different lighting, because objects which appear in shaded regions do not generally exhibit as much noise as objects in lighted regions. A uniform threshold often results in objects disappearing when they enter shaded regions.

If none of the K distributions match the current pixel value, the least probable distribution is replaced with a distribution with the current value as its mean value, an initially high variance, and low prior weight. The prior weights of the K distributions at time i, $\omega_k(i)$ are adjusted as follows,

$$\omega_k(i) = \alpha \cdot \omega_k(i) + (1 - \alpha) \cdot M_k(i) \tag{3.13}$$

where α is the learning rate and $M_k(i)$ is 1 for the matched models and 0 otherwise.

The Gaussians are ordered according to the value ω/σ. This value increases both as a distribution gains more evidence and as the variance decreases. After re-estimating the parameters of the mixture, it is sufficient to sort from the matched distribution towards the most probable background distribution, because only the matched models relative value will have changed. This ordering of the model is effectively an ordered, open-ended list, where the most likely background distributions remain on top and the less probable transient background distributions gravitate towards the bottom and are eventually replaced by new distributions.

Then the first B distributions are chosen as the background model, where

$$B = argmin_b(\sum_{k=1}^{b} \omega_k > C) \tag{3.14}$$

being C a measure of the minimum portion of the data that should be accounted for by the background. This takes the best distributions until a certain portion, C, of the recent data has been accounted for. If a small value for C is chosen, the background model is usually unimodal. If this is the case, using only the most probable distribution will save processing.

If C is higher, a multi-modal distribution caused by a repetitive background motion (e.g. leaves on a tree, a flag in the wind, etc.) could result in more than one colour being included in the background model. This results in a transparency effect which allows the background to accept two or more separate colours.

This model is able to handle clutter in the background by associating several models to each pixel. It fails when sudden changes in the background appear, because the model takes a long time to stabilize again. Several other methods are built taking the work of [grimson99] as a basis, for instance, [zeng08], [lee05], [varcheie08], [zivkovic04] and [elbaf09].

3.3.4 Kernel density estimation (KDE)

An approximation of the background probability density function, can be given by the histogram of the most recent values classified as background values. However, as the number of samples is necessarily limited, such an approximation suffers from significant drawbacks: the histogram, as a step function, might provide poor modelling of the true, unknown probability density function, with the tails of the true probability density function often missing.

In order to address such issues, in [elgammal00] have proposed to model the background distribution by a non-parametric model based on Kernel Density Estimation (KDE) on the buffer of the last K background values. KDE guarantees a smoothed, continuous version of the histogram.

In [elgammal00], the background probability density function is given as a sum of Gaussian kernels centred in the most recent K background values, $I_{x,y}(i))$,

$$P(x_i) = \frac{1}{K} \cdot \sum_{q=1}^{K} \eta(I_{x,y}(i) - I_{x,y}(q), \Sigma_q) \tag{3.15}$$

where η is a Gaussian probability density function, $F_{x,y}(i)$ are the last K observed values of pixel x and Σ_q is the covariance matrix of the q-th Gaussian in the mixture at time i. Likewise equation 3.9, this model seems to be dealing with a sum of Gaussians. However, differences are substantial: in equation 3.9, each Gaussian describes a main mode of the probability density function and is updated over time; here, instead, each Gaussian describes just one sample data, with n in the order of 100, and $\Sigma(i)$ is the same for all kernels. If background values are not known, unclassified sample data can be used in their place; the initial inaccuracy will be recovered along model updates. Based on 3.15 classification of $FI_{x,y}(i)$, as foreground can be straightforwardly stated if the following condition holds,

$$P(I_x(i)) < T \tag{3.16}$$

being T a user selected value.

Model update is obtained by simply updating the buffer of the background values in fifo order by selective update; in this way, pollution of the model by foreground values is prevented. However, complete model estimation also requires the estimation of Σ_t, which is assumed diagonal for simplicity. In [elgammal00], the variance is estimated in the time domain by analysing the set of differences between two consecutive values.

The methods introduced so far model independently single pixel locations. However, it is intuitive that neighbouring locations will exhibit spatial correlation in the modelling and classification of values. To exploit this property, various morphological operations have been used for refining the binary map of the classified foreground pixels. In [elgammal00], instead, this same issue is addressed at the model level, by suggesting to evaluate $P(I_{x,y})$ also in the models from neighbouring pixels and use the maximum value found in the comparison against T.

3.3.5 Sequential kernel density approximation

Mean-shift vector techniques have been employed for various pattern recognition problems such as image segmentation and tracking. The mean-shift vector is an effective gradient-ascent technique able to detect the main modes of the time probability density function directly from the sample data with a minimum set of assumptions, unlike

the approach in [grimson99], the number of modes is unrestricted. However, it has a very high computational cost since it is an iterative technique and it requires a study of convergence over the whole data space. As such, it is not immediately applicable to modelling background probability density functions at the pixel level.

There have been recent approaches trying to solve this problem. In [piccardi04a], authors propose some computational optimisations promising to mitigate the computational drawback. Moreover, in [han04], the mean-shift vector is used only for an off-line model initialisation. In this step, the initial set of Gaussian modes of the background probability density function is detected from an initial sample set. The real-time model update is instead provided by simple heuristics coping with mode adaptation, creation, and merging. In their paper, authors of [han04], compared the probability density function obtained with their method against that of a KDE approach over a 500-frame test video, finding a low mean integrated squared error in the order of 10^{-4}; this justifies the name of *Sequential Kernel Density approximation* (SKDA) that the authors gave to their method.

This method can effectively model a multimodal distribution without the need of assuming the number of modes a priori, but at the cost of a very high computational cost.

3.3.6 Temporal median filter

Various authors argued that other forms of temporal average perform better than the one introduced in the previous section. For instance, in [cucchira01] a method using the median of the last n frames to calculate the background model $B(i)$, even if these n frames are subsampled from the original frame rate by a factor of 10. A disadvantage of this method is that its computation requires a buffer with the nK most recent values; moreover, the median filter does not accommodate for a rigorous statistical description and does not provide a deviation measure for adapting the subtraction threshold.

Authors of [yang92] proposed an algorithm for constructing the background primal sketch by taking the median value of the pixel colour over a series of images. The median value, as well as a threshold value determined using a histogram procedure based on the least median squares method, was used to create the difference image. This algorithm could handle some of the inconsistencies due to lighting changes, etc.

3.3.7 Eigenbackgrounds

The approach proposed in [oliver00] is based on an eigenvalue decomposition, applied to the whole image instead of blocks. Such an extended spatial domain can extensively explore spatial correlation and avoid the tiling effect of block partitioning.

The method can be divided into two different phases, the learning phase, in which the initial model is learned and the classification phase, in which pixels are classified as background or foreground.

Learning phase:

- samples of K images are acquired, each image with p pixels; the mean image, μ_b, and the covariance matrix C_b are computed.
- the covariance matrix is diagonalized via an eigenvalue decomposition. By applying PCA to the eigenvalues of the covariance matrix, only M eigenvectors, eigenbackgrounds as called in the paper, are kept. These values are stored in an eigenvector matrix, θ_{M_b}, of size $M \cdot p$.

- the mean value of the eigenbackground images μ_b is computed.

Classification phase:

- Every time a new frame, $F(i)$, is available, it is projected onto the eigenspace as $F'(i) = \theta_{M_b}(F(i) - \mu_b)$
- By computing the euclidean distance between the input image, $F(i)$ and the projected image, $F'(i)$, moving objects may be detected,

$$D_i = |F(i) - B(i)| > T \tag{3.17}$$

  being $B(i) = \theta_{M_b}(F(i) - \mu_b)$ and T a given threshold.

- F' is then back projected onto the image space as $F'' = \theta_{M_b} \cdot F' + \mu_b$. Since the eigenspace is a good model for the static parts of the scene, but not for the small moving objects, F'' will not contain any such objects;

In [oliver00], however, it is not explicitly specified what images should be part of the initial sample.

3.3.8 Coocurrence of image variations

Authors of [seki03] try to go beyond the idea of mere chronological averages by exploiting spatial cooccurrence of image variations. Their main statement is that neighbouring blocks of pixels belonging to the background should experience similar variations over time. Although this assumption proves true for blocks belonging to a same background object (such as an area with tree leaves), it will evidently not hold for blocks at the border of distinct background objects.

The method in [seki03] works on blocks of $N \cdot N$ pixels instead of on simple pixels. This technique consists in two phases. A learning phase and a classification phase. In the first phase, for each block, a certain number of samples is acquired over time. The temporal average is computed and the differences between the samples and the average are called the image variations. In the classification stage, each block is classified as background or foreground.

Instead of working at pixel resolution, the method in [seki03] works on blocks of $N \cdot N$ pixels treated as an N^2-component vector. This trades off resolution with better speed and stability. Two phases may be distinguished: the learning and the classification phase.

- *Learning phase*:
  - for each block, a certain number of time samples is acquired; the temporal average is first computed and the differences between the samples and the average are called the *image variations*
  - In block u, covariance matrix C_u is computed from the image patterns $i_{u,t}$ of the background image sequence and the average pattern $\hat{i_u}$,

$$S_u = \sum_{t=1}^{\tau} ((i_{u,t} - \hat{i_u})(i_{u,t} - \hat{i_u}))' \tag{3.18}$$

    being τ the learning time.

  - the covariance matrix C_u is decomposed into K eigenvectors.
  - Image patterns $i_{u,t}$ of a background image sequence in block u are transformed into points in the eigen space and memorized with observation time,

$$z_{u,t} = E_u^T \cdot (i_{u,t} - \hat{i_u}) \tag{3.19}$$

- *Classification phase*:
  - new input patterns for each u are transformed into points z_u in each eigen space E_u
  - the L-nearest neighbours to z_w in the eigenspace, $z_{u,t}$, are found and z_u expressed as their linear interpolation
  - the same interpolation coefficients are applied to the values of the current block, b, which have occurred at the same time of the $z_{u,i}$; this provides an estimate, z'_b, for its current eigen image variation z_b
  - Background subtraction is performed by calculating background-likelihood in each block. The rationale of the approach is that z_b and z'_b should be close if b is a background block.

In [seki03], the probability of background-likelihood when judging only based on the input pattern in the focused block and the probability of background-likelihood when judging based on the patterns estimated from some neighbouring blocks, are combined to dynamically narrow the range of background image variations in a focused block.

Authors of the paper do not specify whether the learning phase should be repeated over time to guarantee model update. As this model is based on variations, it is likely to show a natural robustness to limited changes in the overall illumination level. However, a certain update rate would be needed to cope with more extended illumination changes.

3.3.9 Local Binary Patterns

Authors of [heikkila06] and [heikkila04] introduce a novel method in which textures are used to perform background modelling and object detection. In this method, each pixel is modelled as a group of adaptive local binary patterns (LBP) histograms, calculated over a circular region around the pixel to model. This method permits modelling each pixel with several models if needed as the algorithm incorporates naturally the extension to multimodels.

LBP is a grey-scale invariant texture primitive statistic. The operator labels the pixels of an image region by thresholding the neighbourhood of each pixel with the centre value and considering the result as a binary number (binary pattern). The definition of the texture is as follows,

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c) \cdot 2^p \tag{3.20}$$

being $s(x)$ defined as,

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.21}$$

where g_c corresponds to the grey value of the centre pixel (x_c, y_c) of a local neighbourhood and g_p to the grey values of P equally spaced pixels on a circle of radius R. See figure 3.3 for an illustration of the LBP operator. The value of neighbours which do not fall exactly on pixels are estimated by bilinear interpolation.

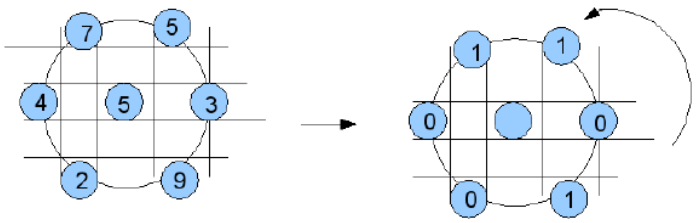


Figure 3.3: Calculation of the binary pattern of a pixel. On the left, the region around the pixel. On the right, the result of binarization. According to the arrow direction, the resulting binary pattern would be in this case 101100

The processing of each frame is performed in two phases: background modelling and foreground detection. Each one of them is briefly discussed in the following paragraphs.

Each pixel of the background is modelled identically, which allows a high speed parallel implementation if needed. In the following, the process of modelling the background is explained for just one pixel, but the procedure is identical for each pixel in the model.

For each pixel, a number of LBP histograms $\{m_0, m_1, ..., m_{K-1}\}$ are maintained, being K a user settable parameter. Each model histogram has a weight between 0 and 1 in such a way, that the weights of the K models of a pixel sum up to 1. The weight of the k-th model histogram is denoted by w_k. Let us denote the LBP histogram of the given pixel computed from the new video frame by h.

Each LBP histogram h is computed using equation 3.20. The computation is performed by considering a region of pixels around the centre and comparing them to the same amount of pixels in a region around each neighbour. The binary patterns are shifted to the left and cumulated. Finally the normalized histogram is computed by diving all components by the maximum value.

At the first stage of processing, h is compared to the current K model histograms using a proximity measure. Authors use the intersection between histograms (as in equation 3.31) because they claim that it has an intuitive motivation in that it calculates the common part of two histograms.

Its advantage is that it explicitly neglects features which only occur in one of the histograms. Another advantage, is that the complexity is very low as it requires very simple operations only. The complexity is linear for the number of histogram bins.

A threshold for the proximity measure, T_p, a user-settable parameter, is used to threshold this distance measure. If the proximity is below this threshold, then the algorithm would replace the lowest-weighted histogram of the model with h. This new histogram is given a low weight, in the experiments performed by the authors, the value was 0.01. In the paper, authors state that a match with the background is treated in a different way, however, there is no explanation about what is considered as a match. Though authors state in their paper that the best match is the model histogram with the

highest proximity value (i.e. the lowest distance). This best matching model is adapted with the data obtained from the image as follows,

$$m_k = \alpha_b \cdot h + (1 - \alpha_b) \cdot m_k \tag{3.22}$$

where $\alpha_b \in [0, 1]$ is a user-settable learning rate. The weights of all models per pixel is updated with the following expression,

$$w_k = \alpha_w \cdot M_k + (1 - \alpha_w) \cdot w_k \tag{3.23}$$

being $M_k = 1$ for the best matching histogram and 0 for the rest, and α_w is a user-settable learning rate which verifies that $\alpha_w \in [0, 1]$.

As a last stage of the updating procedure, the model histograms are sorted in decreasing order according to their weights, and the first B histograms are selected as the background histograms,

$$w_0 + w_1 + ... + w_{B-1} > T_B \tag{3.24}$$

where $T_B \in [0, 1]$ is a user-settable threshold.

Finally, foreground pixels are detected by testing which pixels in current frame did not match any histogram of the background model.

3.3.10 Wallflower

The approach introduced in [toyama99] is the only one which combines three different processing layers; pixel, region and frame layer, in order to determine the regions of interest of a frame. Instead of considering a model for the scene, each pixel is assigned a Wiener filter. The algorithm is designed to work in grey tones.

In the first stage, the pixel level, a Wiener filter is used to compute predictions about future values of a pixel. In the case predictions fails, the pixel is labelled as foreground. Adaptation is achieved by recomputing the filter parameters for each pixel.

The prediction for each pixel (x, y), denoted by $\hat{F}_{x,y}(i)$, is computed as,

$$\hat{F}_{x,y}(i) = \sum_{k=0}^{P} a_k * F_{x,y}(i - k) \tag{3.25}$$

being a_k the filter coefficients, P the total amount of coefficients used for predictions and $F_{x,y}(i)$ are previous values of the pixel. The coefficients of the filter are computed from the sample covariance values of $F_{x,y}(i)$. Details of these computations can be found in [makhoul75]. For each prediction, $\hat{I}_t(x, y)$, a expected squared error is computed as,

$$E[e_t^2] = E[F_{x,y}(i)^2] + \sum_{k=0}^{P} a_k * E[F_{x,y}(i)F_{x,y}(i - k)] \tag{3.26}$$

If the condition,

$$|F_{x,y}(i) - \hat{F}_{x,y}(i)| >= 4 \cdot \sqrt{(E[e_t^2])} \tag{3.27}$$

the pixel is considered foreground. The linear prediction works well for periodically changing pixels and produces large values of $E[e_t^2]$ for random changes.

In order to reduce possible corruptions of the history values of a pixel, a history of predicted values is also stored. For each new frame, two predictions are computed, one based on the actual history and another one based on predictions. A pixel is considered as background if any of both values is within the tolerance stated by inequality 3.27.

Model adaptation is achieved by computing the filter coefficients a_k for each frame. The new coefficients are kept if their corresponding expected error is less than 1.1 times the previous error.

The region level recovers foreground pixels whose prediction failed. This is achieved by considering the intersection of moving regions in frame $F(i-1)$ and frame $F(i)$.

For each frame $F(i)$, the difference with previous frame is computed and thresholded in a similar way to section 3.3.1. With the result of the difference, $J(i)$, the previous difference $J(i-1)$ and the previous foreground image $S(i-1)$, the candidate regions for growing are computed as,

$$K = J(i-1) \wedge J(i) \wedge S(i-1) \tag{3.28}$$

For each four connected regions R_i discovered in K a histogram H is computed for all grey tones s of image R_i as,

$$H_i(s) = \frac{Card(\{p : p \in R_i \wedge (F_p(i-1) = s)\})}{\{Card(\{R_i\})} \tag{3.29}$$

The histogram is backprojected in frame $F(i)$. For each R_i the intersection $R_i \wedge S_{t-1}$ is computed and each point p in the intersection contributes to grow $S(i)$ as,

$$S_{x,y}(i) = \begin{cases} 1, \ H_i(I_p) > \epsilon \\ 0, \ otherwise \end{cases} \tag{3.30}$$

In a third stage, the frame region considers information at frame level to decide when a change in the overall background has occurred and the model must be recomputed for all pixels.

3.3.11 Edges histograms

Edge histograms are used in [mason01] to detect foreground objects in images. Authors propose dividing a colour image into different cells and associate an edge histogram to each cell. The background model is a set of colour and edges histograms.

Index of the edge histograms are computed using the edge orientation; the resolution of the histogram is 10° per bin. When the bin index is determined the bin is incremented with the edge magnitude.

For each new frame, histograms for cells are computed in the same way and compared to the background model.

Histogram comparison techniques are used in the paper to detect the foreground objects. For two histograms h_a and h_b, the two comparison techniques are histogram intersection 3.31 and chi-squared measure 3.32.

$$f(h_a, h_b) = \frac{\Sigma_i min(h_a(i), h_b(i))}{\Sigma_i h_a(i)} \tag{3.31}$$

$$\chi^2(h_a, h_b) = 2 \cdot \Sigma_i \frac{(h_a(i) - h_b(i))^2}{h_a(i) + h_b(i)} \tag{3.32}$$

being i the bin index in each histogram.

Authors of [mason01] state that their method is tolerant to camera noise and illumination changes, however, they do not specify how the model may be updated in order to deal with light evolution, for instance.

3.3.12 Salient motion

In [wixson00], authors state that motion detection can play an important role in many vision tasks. Yet image motion can arise from uninteresting events as well as interesting ones. In their paper, they define salient motion as motion that is likely to result from a typical surveillance target, for instance, a person or vehicle travelling with a sense of direction through a scene as opposed to other distracting motions, for instance the oscillation of vegetation in the wind.

The proposed algorithm for detecting this salient motion is based on intermediate-stage vision integration of optical flow.

The salience field is a consistent measure of the distance travelled in the x- and y-direction by a pixel. This measure takes into account that flow field is rarely perfectly computed for every pixel and also that objects may temporarily pass behind small occlusions.

This salience field is computed separately for the x- and y- components. In general, it is computed as,

$$S'_j = \begin{cases} 0,\ j = 0 \\ \Delta_j + warp(S_{j-1},\ {}^{j-1}_{j}E),\ otherwise \end{cases} \tag{3.33}$$

where Δ_j is the contribution to the cumulative flow of the frame-to-frame flow from frame $j-1$ to frame j, $warp(I, F)$ is a function that computes the warp image I by applying to it the flow field F and finally, ${}^{j-1}_{j}E$ is the extended flow field.

The extended flow field is computed to achieve robustness to errors in computed flow and temporal gaps created when a moving object temporarily passes behind small occlusions. It is derived from the flow filed by checking for each pixel p in the flow field, whether there exists an scalar multiple s of the original vector ${}^{j-1}_{j}F(p)$ that extends the vector so that it connects to a location with large salience.

After the salience field is computed, the maximum cumulative flow field B in the x and y directions are computed as,

$$B_{j,x}(p) = \begin{cases} S'_{j,x}(p),\ if\ sign(S'_{j,x}(p)) == sign(m_x) \wedge |S'_{j,x}(p)| > |m_x| \\ mx,\ otherwise \end{cases} \tag{3.34}$$

being m_x the value of the x-component of the maximum cumulative flow vector at location p in frame I_{j-1},

$$m_x = B_{j-1,x}(p + {}^{j-1}_{j}\ E(p)) \tag{3.35}$$

The component $B_{j,y}$ is updated anlogously with an equivalent definition of m_y.

Image segmentation is performed as follows,

$$S_{j.x}(p) = \begin{cases} 0,\ if|B_{j,x}(p)| > (k_s \wedge \frac{|S'_{j,x}(p) - B_{j,x}(p)|}{|B_{j,x}(p)|}) > k_r \\ S'_{j,x}(p),\ otherwise \end{cases} \tag{3.36}$$

In the case that $S_{j.x}(p)$ is equal to 0, then $B_{j,x}(p)$ is naturally also set to 0. Authors of [wixson00] state that typically, the minimum salience k_s is set to 8 to ensure that

some minimal salience has a chance to accumulate before it can be reset to 0. The fractional change k_r is typically set to 0.1, indicating that if the cumulative flow drops to 90% of the largest value previously observed, a direction change is occurring. The precise setting, however, is not important, since in general pixels on vegetation will exhibit direction reversals that represent large percentage changes relative to their maximum value.

Empirical results are presented that illustrate the applicability of the proposed methods to real-world video. Unlike many motion detection schemes, no knowledge about expected object size or shape is necessary for rejecting the distracting motion.

3.4 Motion segmentation algorithms

There are three general techniques used when it comes to motion segmentation: temporal differencing, background segmentation, and optical flow. In the following subsections, each basic technique is introduced along with refinements proposed in different works to improve their performance.

3.4.1 Temporal differencing

The approach of temporal difference makes use of pixel-wise difference between several consecutive frames in an image sequence to extract moving regions. Temporal differencing is very adaptive to dynamic environments, but generally does a poor job extracting the entire relevant feature pixels, e.g., possibly generating holes inside moving entities and also missing static objects, which may be of interest in some applications.

As an example of this method, authors of [vsam99] detected moving targets in real video streams using temporal differencing. After the absolute difference between the current and the previous frame was obtained, a threshold function was used to determine change. By using a connected component analysis, the extracted moving sections were clustered into motion regions. These regions were classified into predefined categories according to image-based properties for later tracking.

Generally, given $F(i)$ the frame captured in time i, the temporal differencing is calculated as,

$$J(i) = |F(i) - F(i-1)| \tag{3.37}$$

and the binary image representing the segmentation is easily computed as,

$$\forall(x,y) \in F(i), S(x,y) = \begin{cases} 1, \ if \ J_{x,y}(i) \geq \mu_{x,y} \\ 0, \ otherwise \end{cases} \tag{3.38}$$

Where μ is a threshold which can, in turn, adapt itself to changes in the scene or keep constant.

In [vsam99], this threshold varies according to changes in the scenario. In this paper, authors propose using three consecutive images to calculate the motion in the scene, being $J^1(i) = |F(i) - F(i-1)|$ and $J^2(i) = |F(i) - F(i-2)|$, the expression used is as follows,

$$\forall(x,y) \in I_t, S(x,y) = \begin{cases} 1, \ if \ J^1(i) \geq \mu_{x,y}(i) \wedge J^2(i) \geq \mu_{x,y}(i) \\ 0, \ otherwise \end{cases} \tag{3.39}$$

The threshold is computed in a pixel per pixel basis in each time i, according to the following expression,

$$\mu_{x,y}(i) = \alpha \cdot \mu_{x,y}(i-1) + (1-\alpha) \cdot 5 \cdot (F_{x,y}(i) - F_{x,y}(i-1)) \tag{3.40}$$

where α is a learning rate which ranges in $[0, 1]$. In image 3.4, an example of frame differencing may be seen.

Authors of [vsam99] justify their equations by considering each non moving pixel position as a time series, $B_{x,y}(i)$ is analogous to a local temporal average of intensity values and $\mu_{x,y}(i)$ is analogous to 5 times the local temporal standard deviation of intensity, both computed using an infinite impulse response (IIR) filter.

(a) $F(i)$ (b) $F(i+1)$ (c) Frame differencing result S

Figure 3.4: Example of frame differencing. Figure 3.4a and 3.4b are subtracted and the result is shown on Figure 3.4c, note that only pixels which are very close to the border of the objects are marked as foreground.

This technique has the drawback that cannot find all foreground pixels, but only those which have changed from one frame to the next. Of course, static objects are not detected and are considered as background. On the other side, they are easy to implement, do not need the maintenance of a background model as following methods and do easily detect the silhouette of moving objects.

3.4.2 Background subtraction

Background subtraction is a particularly popular method for motion segmentation, especially under those situations with a relatively static background. It attempts to detect moving regions in an image $F(i)$ by differencing between current image and a reference background image B_{t-1} in a pixel-by-pixel fashion. This technique relies on an accurate background used in the subtraction operation; by means of this subtraction, relevant areas are found and further processing may follow in them.

However, this technique is extremely sensitive to changes of dynamic scenes due to lighting and extraneous events. The numerous approaches to background subtraction differ in the type of background model used and the procedure used to update it. The simplest background model is the temporary averaged image, a background approximation that is similar to the current static scene.

The subtraction performed thus, depend on the exact type of background computed. In general, background subtraction methods start with a background model $B(0)$ in time $i = 0$. It is a representation of the background of the scene. The subtraction is performed as,

$$\Delta(i) = |F(i) - B(i-1)| \tag{3.41}$$

and the segmentation is given by,

$$\forall(x,y) \in F(i), S_{x,y} = \begin{cases} 1, \ if \ \Delta_{x,y}(i) \geq \mu \\ 0, \ otherwise \end{cases} \tag{3.42}$$

being μ the segmentation threshold. Figure 3.5 shows the typical result of subtracting the background from an scene.

(a) Background model $B(i-1)$ (b) Incoming frame $F(i)$ (c) Result of background subtraction S

Figure 3.5: Example of background subtraction, on the left, the background model; on the centre, the incoming frame; on the right, the result of performing the subtraction of both images and thresholding the result.

Model B may be adapted over time to cope with light or scenarios changes (an object being set or removed) or not. In the case it is not, these changes may corrupt the background.

In the case the model is adapted over time, pixels in the model are updated in order to cope with light changes. The simplest way of doing this, is by using the following equation, which computes an average of the input,

$$B_{x,y}(i) = \alpha \cdot B_{x,y}(i-1) + (1-\alpha) \cdot F_{x,y}(i) \tag{3.43}$$

Where α is a learning parameter which ranges in $[0,1]$. Both α and the model's adaptation period are parameters which must be tuned together. The model's adaptation period is a parameter that controls when the background model is adapted to cope with light changes; and it may be static or dynamic for each individual pixel or for the entire image.

On the other side, α can also be static or dynamic and even defined individually for each pixel or be fixed for the entire image. In [porikli03] an interesting work on how to adapt this value is discussed. In their work, the authors propose a method, in which each pixel (x,y) in the incoming image is given a value of $\alpha_{x,y}$ and an update period $T_{x,y}$, providing a method for illumination compensation.

Different subtraction methods are used depending on how the model is built, besides the basic 3.41. For instance, in [grimson99], see section 3.3.3, each pixel is compared to the variance of several Gaussian distributions to perform the background subtraction. Other approaches based on [grimson99], perform similar subtraction with slight differences as for instance, in [elbaf09], where foreground detection is performed by means of fuzzy gaussian distributions.

In the algorithm introduced in [heikkila06], section 3.3.9, the subtraction consists of an histogram comparison by means of histogram intersection.

In the work introduced in [toyama99], see section 3.3.10, the value of each pixel is predicted by means of a Wiener filter. The background subtraction is verified by computing the difference between the current value of the pixel and its prediction. This difference is compared with an error computed taking into account the coefficients of the filter. An study discussed in [haritaoglu00], presents a statistical model by representing each pixel with three values; is minimum and maximum intensity values, and the maximum intensity difference between consecutive frames observed during the training period. The model parameters were updated periodically.

3.4.3 Optical flow

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene.

Optical flow methods can be used to detect independently moving objects even in the presence of camera motion. However, most flow computation methods are computationally complex and very sensitive to noise, and cannot be applied to video streams in real-time without specialized hardware.

In [rowley97], authors also focused on the segmentation of optical flow fields of articulated objects. Its major contributions were to add kinematic motion constraints to each pixel, and to combine motion segmentation with estimation in EM (Expectation Maximization) computation.

3.4.4 Other methods

In addition to the basic methods described above, there are some other approaches to motion segmentation. Authors of [friedman97] implemented a mixture of Gaussian classification model for each pixel. This model attempted to explicitly classify the pixel values into three separate predetermined distributions corresponding to background, foreground and shadow. Meanwhile it could also update the mixture component automatically for each class according to the likelihood of membership. Hence, slow-moving objects were handled perfectly, meanwhile shadows were eliminated much more effectively.

Authors of [stringa00] also proposed a novel morphological algorithm for scene change detection. This method allowed obtaining a stationary system even under varying environmental conditions. From the practical point of view, the statistical methods are a far better choice due to their adaptability in more unconstrained applications.

3.5 Shadow removal techniques

As mentioned in section 3.2, shadows may become a problem when detected objects must be separated accurately. Removing shadows is important in order to improve object disambiguation and classification.

Usually algorithms aim to remove those shadows which are prone to affect the blob's shape, the so-called cast-shadows. Self-shadows or shadows that form part of the background are not relevant as they do not affect the tracking process.

Research on shadow detection focuses on two main uses:

- disambiguation for object recognition
- recovery of the underlying surface detail

The first problem is the most important when it comes to surveillance applications, because it is necessary to disambiguate which pixels belong to the blob representing the real object and which do not.

In [prati01] an extensive comparison among different shadow removal techniques is performed, see figure 3.6. The different algorithms are divided into deterministic approaches and statistical approaches. In the former, an on/off decision process is used to decide whether a pixel belongs to background or to foreground; in the later, a probabilistic function is used to describe the class membership.

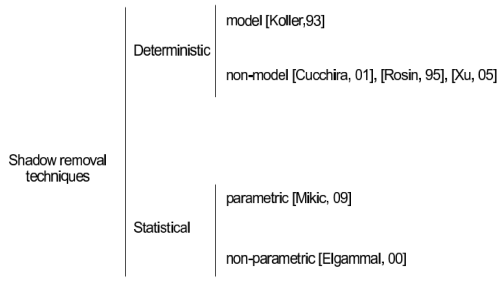


Figure 3.6: A taxonomy of shadow removal techniques as found in [prati01]

In the statistical-based methods the parameter selection is an issue. The work reported in [mikic00] is an example of the parametric approach, whereas [elgammal00] and [horpraset99] are examples of the non-parametric approach.

Within the deterministic class, another sub-classification can be based on whether the on/off decision can be supported by model based knowledge as in [koller93] or not, as in [cucchira01], [rosin95] or [xu05].

Choosing a model based approach achieves undoubtedly the best results, but is, most of the times, too complex and time consuming compared to the non-model based. Moreover, the number and the complexity of the models increases rapidly if the aim is to deal with complex and cluttered environments with different lighting conditions, object classes and perspective views. Due to their better time response, only the deterministic non-model based approaches are considered in this work.

3.5.1 Non-model deterministic methods

For instance, the algorithm discussed in [rosin95] would be classified in the deterministic approach set. Authors state that shadows can be interpreted in images and the effect they have on the pixels in the scene is as a semi-transparent region in which the scene reflectance undergoes a local attenuation.

Provided that the imaging sensor is stationary, it is feasible to identify those regions within shadows by analysing their photometric properties. The photometric gain computed as,

$$\forall(x,y) \in F(i), gain(x,y) = \frac{F_{x,y}(i)}{B_{x,y}(i-1)} \tag{3.44}$$

shows that, firstly, shadow pixels will have a photometric gain with respect to the background image B_{t-1}, smaller than unity. Secondly, this gain will be reasonably constant over all the shadow region, except at the edges, where the effects of a finite size illumination source will tend to reduce the attenuation, i.e. the penumbra.

In this case, shadows are modelled as a constant contrast change between the background model image and the current image, and are detected by expanding the region to locate areas of constant photometric gain in the image. Heuristic rules are then used to cue possible shadow regions.

The algorithm proposed in [xu05] belongs also to the deterministic approaches, in this case RGB coordinates are used, instead of grey tones. According to this approach, a shadow is an area that is not, or is only partially, irradiated or illuminated because of the interception of radiation by an opaque object between the area and the source of radiation.

Authors state that, assuming that the irradiation consists only of white light, the chromaticity of a shadowed region should be the same as when it is directly illuminated.

This also applies to lightened areas in the image. Based on a similar assumption, a normalised chromatic colour space, $r = R/(R+G+B)$, $g = G/(R+G+B)$, $b = B/(R+G+B)$ for instance, is immune to shadows, but the lightness information is unfortunately lost. Keeping lightness information is important in order to avoid some simple errors such as confusing a white car with a grey road.

Based on the fact that both brightness and chromaticity are very important, a good distortion measure between foreground and background pixels should account for the discrepancies in both their brightness and chromaticity components, as in [horpraset99].

Brightness distortion (BD) can be defined as a scalar value that brings the expected background close to the observed chromaticity line, being $\overrightarrow{p_{i,j}}$ and $\overrightarrow{q_{i,j}}$ vectors representing the RGB values of a pixel as expressed in [horpraset99],

$$BD = argmin_{\alpha}(\overrightarrow{p_{i,j}} - \alpha \cdot \overrightarrow{q_{i,j}}) \tag{3.45}$$

BD can be easily computed as,

$$BD = \frac{\overrightarrow{p_{i,j}} \cdot \overrightarrow{q_{i,j}}}{\overrightarrow{q_{i,j}}^2} \tag{3.46}$$

Similarly, *colour distortion* (CD) can be defined as the orthogonal distance between the expected colour $\overrightarrow{q_{i,j}}$ and the observed chromaticity line $\overrightarrow{p_{i,j}}$,

$$CD = ||\overrightarrow{p_{i,j}} - \alpha \cdot \overrightarrow{q_{i,j}}|| \tag{3.47}$$

being $CD_{i,j}$ the colour distortion between foreground and background for a pixel in coordinates (i, j) and $BD_{i,j}$ is the bright distortion for the same pixel.

A set of thresholds can be defined to assist the classification into foreground, highlighted, or shadowed pixel. Authors of [xu05] propose using a decision tree in order to classify pixels according to the computed CD and BD.

Experiments performed with these two algorithms may be found in appendix B.

3.6 Conclusions

Different techniques proposed in the literature and aimed to extract regions of interest in sequences of frames have been introduced in this chapter. Deciding which method

should be used to model the background and detect regions of interest, depends greatly on the problem to solve and the constraints under which the system should work.

Despite a big amount of algorithms exist in the literature, they all miss the capability of working in conditions in which they cannot be provided with a background model, or in which their working conditions cannot be set up at start by a human operator. Also, some of the algorithms introduced in this chapter require a big amount of memory for their operation, and that makes them difficult to run in special devices, such as smart cameras.

Authors of [toyama99], propose a very useful benchmark which challenges background algorithms with different situations aimed to test specific features that background modelling algorithms should posses. This benchmark is widely accepted by the community as a valid test for new algorithms.

Background subtraction techniques are the most popular in the literature. They have proved to achieve a good performance in scenarios with small changes in the background. The basic technique consists on obtaining a model of the scenario, several methods may be found in the literature to build this background model. Adaptive background models or with a multi-modal support are the most suitable to be used in scenarios in which background motion may appear, still achieving a good performance.

Light conditions also have an influence on the performance of these algorithms, as most of them rely on thresholds to distinguish which pixels belong to foreground and which belong to background. Algorithms may be sort into two groups, regarding the threshold used. One group uses fixed thresholds and another group uses thresholds which depend on statistical computations.

Chapter 4

Character identification in containers

This chapter introduces the developments performed with segmentation techniques, in order to design a system that identifies and recognizes the identification codes of containers. Selection of the most suitable segmentation scheme together with the experiments that support this selection are shown. In the approach discussed in this chapter [rosell06], [rosell06a], [rosell06b]), we propose using several segmentation algorithms to achieve a robust segmentation under uncontrolled light conditions and object filtering in order to reduce oversegmentation. Moreover, we extend the processing of one image and describe an algorithm that identifies the container's code by using a sequence of images.

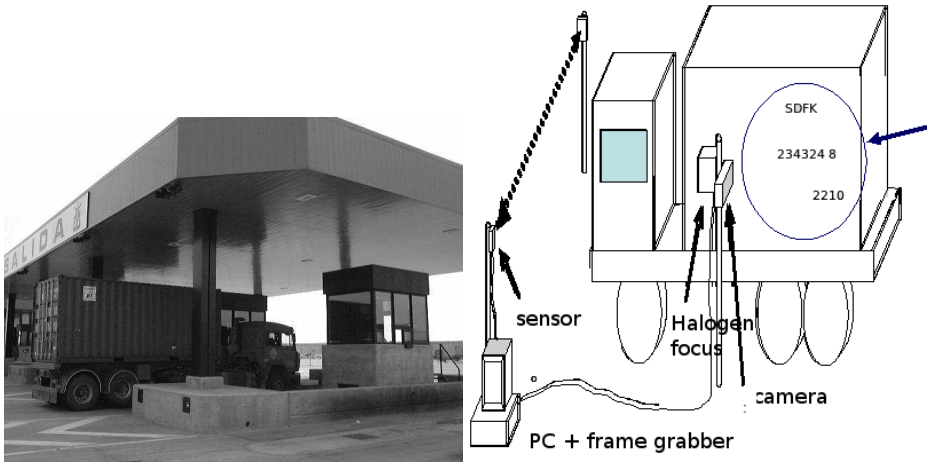
4.1 Introduction

In this chapter, an application of segmentation techniques to a real-life problem is introduced. In chapter 2, image segmentation and some segmentation techniques were discussed. Recalling what was introduced then, the segmentation of an image is the process of separating an image in regions according to some predefined criteria.

Our goal is locating and recognizing the identification code of containers in the entrance of a trading port. Controlling the entrance of truck containers inside the port is crucial in order to control the stuff entering the docks. Currently in most trading ports, gates are controlled by human inspection and manual registration. This process can be automated by means of computer vision and pattern recognition techniques.

Such a system can be built by developing different techniques, such as image preprocessing, image segmentation, feature extraction and pattern classification. The process is complex, because it has to deal with outdoor scenes, days with different climatology, changes in light conditions (day, night) and dirty or damaged containers.

Images were acquired by means of a system installed in the gates of the port of Valencia, the schema of the system can be seen in figure 4.1. The system works with a sensor that detects an incoming truck in the admission gate and triggers a signal that starts the process of taking pictures.



(a) Port's entrance (b) Acquisition system

Figure 4.1: a) Port's gate. b) Acquisition system scheme.

4.1.1 Related work

There are several papers in the literature that deal with locating and recognising vehicles' plates, and some of them do solve this problem with a high degree of productivity. For instance, methods discussed in [brad01],[barroso97] and [hegt98] work efficiently with cars' plates. However, these algorithms expect characters to be arranged in the plate in conditions which are not posible to expect in truck containers. Also, most of these algorithms are focused on the arrangements for a given country, reducing the variability the algorithm has to face, and making the plates processing easier.

In the case of truck containers, no theoretical uniform location of the code may be expected nor the colour of the sought objects is known in advanced. From now on, we will call *symbols* either the letters or the numbers in a code for the sake of simplicity.

ISO 6346 is an international standard managed by the International Container Bureau (BIC) for coding, identification and marking of intermodal containers (shipping containers) used within intermodal freight transport as part of containerization. It establishes:

- an identification system with: an owner code, commonly known as BIC code, an equipment category identifier, a serial number and a check digit
- a size and type code
- a country code
- operational marks

Codes are formed by several letters and numbers arranged either in columns or in rows. Moreover, the colour of the codes may vary, being possible that not all symbols in the same code are drawn using the same colour. Symbols on a container are not constrained to appear in a concrete area of the container as, for instance, cars' plates are, though they usually appear close to the upper corners of the container if arranged in rows, or close to the backmost edge if arranged in columns.

The maximum amount of symbols in the code is limited to 18, though usually, there are among 13 and 15. Some samples (in grey tones) may be seen in figure 4.2. It is very difficult to export solutions for cars' plates to this problem, a different solution is obviously needed.

Figure 4.2: Some sample images representing truck containers. As explained in the text, note the wide variability of possible situations, that the designed algorithm has to face.

A first approach to the process of code detection in truck containers is presented in a previous work [salva01] and the overall process is discussed also in [salva02]. In these works, authors use the top-hat morphological operator (see section 2.3.2 for more details) to preprocess images before segmenting them with a multi-thresholding algorithm. Though this method had good results, we tried to improve their performance by using the methods we propose later in this chapter.

On the other hand, in [atienza05], authors aim to use the optical flow (see section 3.4.3) applied to a sequence of images representing the same container, in order to shrink the area where the container code could be found and speed up the segmentation process; however, the method is very time consuming and an effort should be done in order to optimize it.

4.1.2 Goals and constraints

Our goal is finding a suitable successful segmentation algorithm for the process of code detection mentioned previously. To achieve this goal, several segmentation algorithms found in the literature are tested. The constraints that should be met by the selected technique are:

- It must detect all characters in the code, or as much as possible

- It must find characters independently of their colour (white characters on a dark background and vice versa)
- It must run without human intervention, as the gates are supposed to be automatically driven
- It must be independent of image light conditions
- It must create the minimal list of found objects; as segmentation algorithms will always find objects which are not relevant (see figure 4.3), the optimum will be the one which creates a list that only contains objects which correspond to characters in the code

In the following study several algorithms are tested in order to find out which algorithm could fit better the constraints of the process, and, if it was the case, which could be used together, producing better results than any other algorithm on its own.

One of the problems that a system working outdoors has to face is the environmental conditions. These conditions translate,from the point of view of vision systems, into sudden and uncontrolled changes of light that imply changes on the parameters of the segmentation algorithms.

A usual constraint that these systems must sometimes meet is the limited amount of time for image processing they may have, as most of them are supposed to give a real-time response. Though it is not a critical issue in most cases, it is always desirable to produce an answer in a short period, if possible. In the system developed in this chapter, for instance, it is not necessary an immediate answer but trucks cannot wait infinitely for an answer.

Taking into account these constraints, we propose using a set of standard segmentation algorithms ranging their parameters in intervals as wide as possible; thus covering different environmental situations. This solution could be criticized because a lot of wrong regions of interest, denoted as false positives from now on, will be generated in the output. Being this true, we propose the use of specific filters to remove from the process output all objects which do not meet the expected constraints. We achieve this way a general schema that may be applied in any application in which segmentation must be performed.

Section 4.2 outlines the schema followed. The preprocessing algorithms used are described in section 4.3. Section 4.4, discusses the implementation and experiments made with some well-known segmentation algorithms. Section 4.5 introduces the techniques used to reduce the false positives generated in the segmentation process, together with the experiments performed with these techniques. Finally, section 4.6 explores the use of sequences of images to improve the fault tolerance of the process.

4.2 Proposed schema

The schema proposed in this chapter is illustrated in figure 4.4. For an image I we define the result of applying a set of segmentation techniques Σ to I with a set of parameters K, as the set $S(I, \Sigma, K)$.

The connected regions $R_i \in S(I, \Sigma, K)$, shown in figure 4.3, are filtered according to a set of constraints Φ. Those which meet these constraints form the set $\Upsilon(S(I, \Sigma, K), \Phi)$ which will be called the final segmentation of I. Finally, once the tone of symbols is detected, the set $\Gamma(I)$ will contain the truck container code.

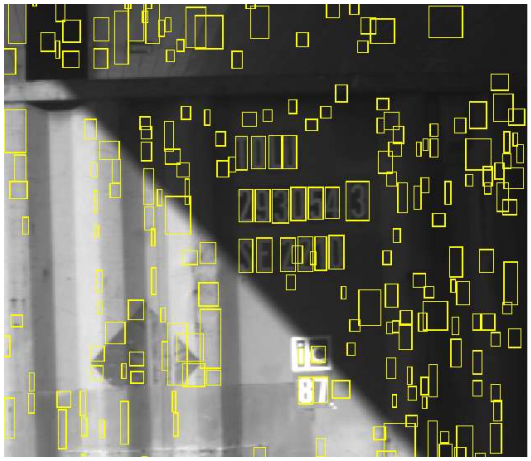


Figure 4.3: Connected regions R_i surrounded by the bounding boxes that enclose them and drawn on the original image. Note that there are more regions than symbols in the code. These regions are the result of applying a segmentation algorithm to one image representing a truck container.

4.3 Preprocessing

As introduced in section 2.3, the aim of preprocessing images is improving in some sense the input image to make it easier to be processed by the following stages.

In the case of truck containers, it was taken into account that symbols may be drawn on the container in different colours. One solution for this problem, could be using a growing-region algorithm to segment the image in different regions according to their colour and then, classify regions. However, an easier approach is taking profit that, if images are converted into grey tones, no colours have to be searched for. By applying the top-hat technique the obtained grey-tone image will contain only dark or light regions. This reduces the complexity of the problem.

Despite the complexity of the problem is smaller, still two different symbol tones have to be distinguished: light and dark symbol tones. As no further information may be obtained with this transform, light and dark preprocessing must be done and the two images are treated in parallel.

If we consider as the input of the process an image I in grey tones, the process starts by applying the top-hat transform to I in order to enhance dark and light regions. This produces two output images called I_{dark} and I_{light}, as seen in figure 4.4. Figure 4.5 shows an example with truck containers.

4.4 Testing different segmentation approaches

Choosing the segmentation technique that better fits the problem's constraints is not an easy task. Depending on the aim of the application and the constraints it has to meet, different algorithms will suit it better than others. In our case, five algorithms with different approaches to segmentation were tested [rosell06]. These algorithms, already introduced in section 2.4, are *Otsu's method*, *LAT*, *thresholding method*, *Watershed* and an adaptation of the *local variation* algorithm.

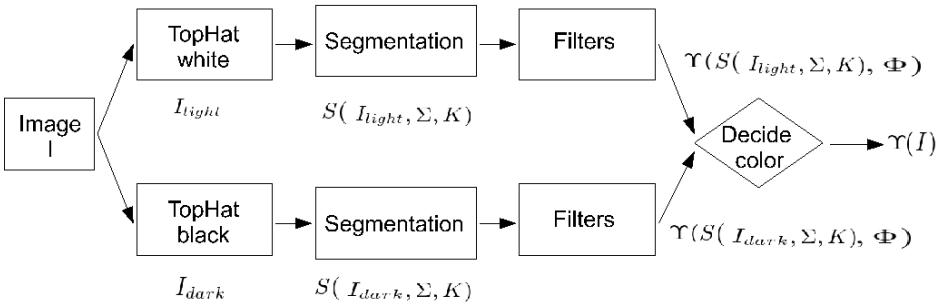


Figure 4.4: Different phases in the proposed processing schema. After preprocessing I, the segmentation stage provides the system with a set of regions of interest. These regions are filtered according to problem dependent criteria. Finally, a decision step tests which is the correct set of regions.

It must be remarked that experiments were performed with the chosen segmentation techniques in order to test their performance in the specific problem we wanted to solve; not to compare their general performance. The most important factor to be measured was the number of found symbols from the code.

4.4.1 A new version for local variation algorithm

The original local variation algorithm, introduced in 2.4.5, is a growing region algorithm which works either with colour or with grey tone images. It requires several parameters which are difficult to set in the general case, due to the high variability of the images; for instance, the expected number of regions. In order to avoid using the final number of regions because not all containers have the same amount of symbols in the code, an adaptation of the algorithm was designed.

In the adapted version, it was decided that two different regions should join into one if the mean intensity of both regions is similar. Another difference is that no minimum size of region was used to force regions to merge. Regions were merged until the process reached a situation in which no more regions could be merged. The only parameter was the percentage of similarity (k) that allowed two regions to be merged into one.

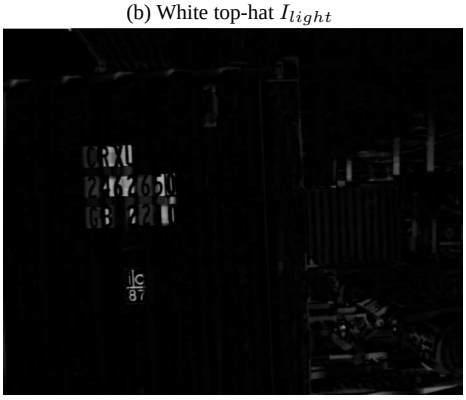
4.4.2 Data set

In order to be able to extract conclusions about the behaviour of the algorithms and techniques applied, a previous work was done. We manually labelled all the characters in the images to be used in the experiments. This process was done by drawing the bounding box for each symbol as accurate as possible. Its coordinates along with the class of the symbol were stored in a file indexed with the name of the image. A sample handcrafted bounding box is shown in figure 4.6. Then, the process of checking results was done automatically with the help of these files.

The experiments were done as close to real conditions as possible, thus, algorithms were not provided with information about light conditions in the images or about the colour of symbols. This way, it could be tested how they would behave under real con-

(a) Original image I

(b) White top-hat I_{light}



(c) Black top-hat I_{dark}

Figure 4.5: Sample execution of top-hat technique. a) An image I representing a truck container. b) Final image ($WTH(I)$) looking for light areas. c) The same for dark areas ($BTH(I)$).

Figure 4.6: Bounding boxes labelled manually. Image is zoomed in in order to show only the container code area.

ditions. For this reason, images used in the tests represented real containers acquired in the entrance gate of a port. We tried to join in this data set as much variety as possible. It was assured that the test set contained images taken during day, night, during cloudy days and combining dark or light symbols on the containers. A total amount of 309 images were used in the tests. In average, each image represented a container with 15 symbols, resulting in about 4635 symbols to recognize. The quality of the symbols, that is, how easy to see they were for a human eye is difficult to quantify. However, we estimate that about a 10% of them was difficult or very difficult to recognize at a first glance.

Preliminary experiments took us to test only four of the five chosen algorithms. These experiments consisted on testing, for a reduced set of images, the performance on detecting symbols of each algorithm and the computational cost.

The performance was measured quantitatively checking how many true positives were obtained, that is, how many symbols were properly segmented. At this stage, no action was expected to take with false positives, which are the regions of the image which contain no interesting objects but are labelled by the segmentation algorithm as regions of interest.

After these experiments, the Otsu's algorithm was discarded because it proved to be faster executing the thresholding algorithm with different thresholds than computing the threshold by means of the Otsu's algorithm and more important, some images need to be thresholded several times in order to find all symbols. Only LAT, watershed, thresholding and the adaptation we made of the local variation algorithm were used in the following experiments.

4.4.3 Parameter set

At this stage, it is necessary to know which are the best fitting parameters for each algorithm. Because they have to adapt to different light conditions, bearing in mind the temporal constraint.

In the preliminary experiments, it was concluded that for LAT algorithm, a value of c ranging in $[0.9; 1.6]$ with an step of 0.03 and $l = 0$, for a total amount of 23 iterations, would yield good results. For the thresholding algorithm, a value of T ranging in $[20; 220]$ with a step of 5 was considered to be enough, for a total amount of 40 iterations. For the local variation algorithm, the likelihood percentage k ranging in

$[70\%; 85\%]$ with an step of 5 was used, yielding a total amount of 4 iterations of the algorithm. Watershed algorithm has no parameters to adjust.

Figure 4.7 illustrates the execution of the thresholding technique on a sample image. The effect of shadows on the container makes it difficult to find a single threshold that successfully detects all characters. By using different thresholds, regions may be discovered and joined together in a single solution.

4.4.4 Evaluation criterion

The response of the segmentation algorithms, modelled as the sets $S(I_{dark}, \Sigma, K)$ and $S(I_{light}, \Sigma, K)$ is obtained. The elements R_i of the sets, correspond to regions with homogeneous grey tones. Some of these regions will be relevant for the container's code seek, but others will not. As shown in figure 4.3, the number of objects found in the segmentation, is always greater than the number of symbols in the code. For the sake of brevity, in next paragraphs we will denote $S(I_{dark}, \Sigma, K)$ by $S(I_{dark})$ and $S(I_{light}, \Sigma, K)$ by $S(I_{light})$.

The good or bad performance of each algorithm was evaluated depending on the total amount of symbols of the code, that could be found in the list of found objects; independently of the total amount of found objects. This is so, because those irrelevant objects of the list could be removed from the final set by means of the filters introduced later in this chapter.

Only a lax geometric filter, based on height and width, was applied with the aim of reducing the number of irrelevant objects in the solution, without the danger of removing valid objects. At this stage, applying other processes to the output would mean masking out the segmentation algorithms' performance, hiding errors or adding errors of other procedures.

A method was designed to compare automatic segmentation to human's results. The algorithm was considered to be successful if the bounding box it calculated for an object and the bounding box drawn by a human operator did overlap in a given percentage. This percentage was set, after some manual tune, to 65%.

In order to proceed with comparisons, for each algorithm the total amount of successfully segmented symbols (the true positives), the number of missing symbols (the false negatives), the total amount of objects found and total amount of time spent in the process was obtained.

4.4.5 Experiments

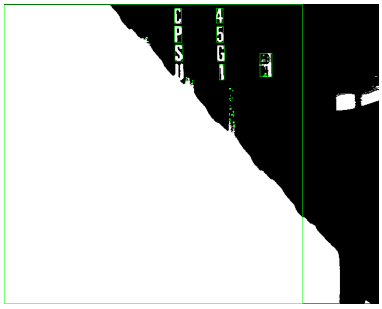
First results of applying the segmentation algorithms to the 309 images in the database are found in table 4.1. In this table, under the column *"missed"*, the total amount of objects that belong to the container's code but the algorithms were not capable to find, is shown. First row shows the number of images from which successfully, the container code symbols were obtained.

According to data in the column corresponding to LAT, it may be seen, that this is the algorithm which achieves best results. It is the one with more images between 0 and 1 lost symbols. Watershed is the second in performance.

On the other side, the adaptation of local variation has an erratic behaviour, maybe due to the fact that only the grey tone average was used to decide whether regions should be merged or not, due to its poor performance.

Values shown in table 4.1 are plotted in figure 4.8. The plot shows the number of images that each algorithm can find according to the number of missing symbols.

(a) Original image



(b) T= 50

(c) T= 170

(d) T= 220

Figure 4.7: Different segmentations of image a) using the thresholding algorithm with different values of T. Note that depending on the threshold used, some characters could be lost or detected.

For instance, if 2 mistakes are affordable, LAT could be able to find up to 287 images correctly out of the initial 309, which represents a high percentage.

Missed	LAT	Watershed	Thresolding	Local variation
0	189	173	143	88
1	68	68	78	84
2	30	32	32	49
3	12	12	20	24
4	4	10	11	19
5	2	4	2	11
6 or more	4	10	23	34
Total	309	309	309	309

Table 4.1: Performance of the segmentation algorithms. Amount of images depending on the number of missed characters. Each column shows results for an algorithm; in each row, the characters missed per image.

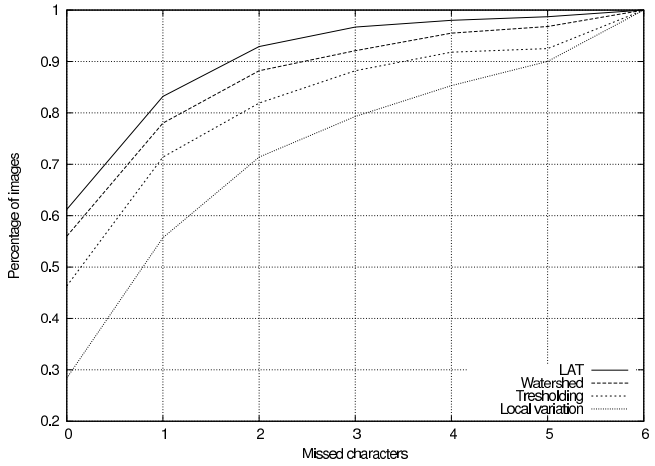


Figure 4.8: Cumulated plot of images depending on the number of missed characters. For instance, in the case 2 missed characters could be tolerated, LAT could be able to find around 93% of the codes correctly.

In table 4.2, the average execution time of each algorithm is shown (measured in seconds). LAT, as it may be seen, is the quickest algorithm. Global thresholding has a good execution time, though results are poorer than those of Watershed or LAT.

Mean time \ Alg.	LAT	Watershed	Thresolding	Local variation
seconds	1.31	7.11	1.78	26.54

Table 4.2: Average execution time of the implemented algorithms. Algorithms were run in a Pentium 4 at 3 Ghz.

After these results, the possibility of combining different algorithms was considered. It is quite likely to be a good solution if we think that, in theory, one algorithm could correct the errors of another or just find objects that another algorithm, whichever the reasons, cannot segment properly. The process of combining the algorithm must be understood as parallelizing their executions and merging their final results.

New experiments were performed joining the results of LAT and Wahtershed, because they are the two with best performance. Also, LAT and Thresholding algorithm were joined, despite the second one has not very good results but it is fast and that could mean best results than with LAT alone with a reasonable execution time. Also a test merging the three algorithms was performed. Though in previous paragraph it was mentioned that these algorithms could be parallelized in order to overlap their execution, in the experiments algorithms were run in sequence.

In table 4.3 we compare results of the LAT-Thresholding algorithm, the LAT-Watershed algorithm, and the algorithm LAT-Watershed-Thresholding (summarized as LWT in the table). It may be seen that the union of LAT and Watershed outperforms any other combination of algorithms when 0 or 1 missed symbols are allowed, achieving the same results as the union of LAT, Watershed and Thresholding. In the case 2 symbols may be missed, then the union of LAT and Thresholding performs identically to LAT and Watershed.

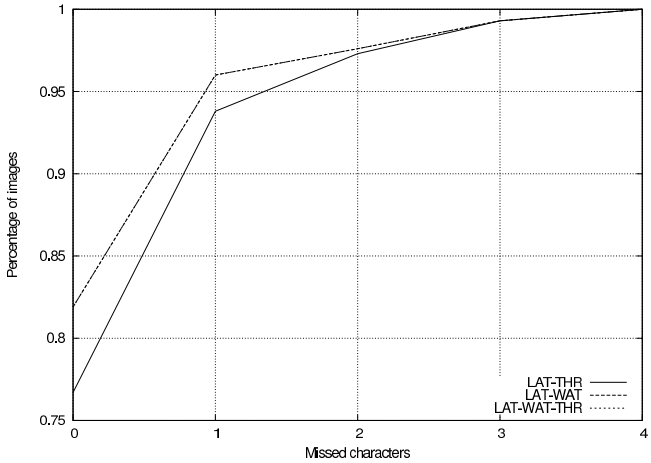


Figure 4.9: Cumulated plot of images depending on the number of missed characters for the improved techniques. For instance, in the case 2 missed characters could be tolerated, LAT-Thresholding technique could be able to find around 97% of the codes correctly. Note that plots of LAT-Watershed and LAT-Watershed-Thresldoing collapse in the same line.

Missed symbols	LAT-Thr.	LWT	LAT-Wat.
0	237	253	253
1	53	44	44
2	11	5	5
3	6	5	5
4	0	0	0
5	2	2	2
6 or more	0	0	0
Total 309	309	309	309

Table 4.3: Performance of the joined algorithms. Amount of images depending on the number of missed symbols. Each column shows results for an algorithm; in each row, the characters missed per image.

In the table 4.4, execution time of the joined algorithms may be seen; the union of

LAT and thresholding algorithm is quite faster than the other two algorithms. In figure 4.9, a plot comparing the different unions of algorithms is shown.

Mean time	LAT-Thr.	Lat-Wat-Thr.	Lat-Wat
seconds	2.67	10.18	8.19

Table 4.4: Average execution time of the improved versions.

Though in next section, techniques aimed to reduce the amount of false positives are introduced, the most false positives generated, the most effort will have to be done to remove them. Table 4.5 shows the average amount of false positives found per image depending on the algorithm used and the symbols under seek. As it may be seen, all algorithms produce a big amount of objects, due to the method of joining together segmentations with different parameters. Specially, LAT and Watershed algorithms multiply by a factor of 10 the amount of objects found by Thresholding or local variation algorithms.

	LAT		Watershed		Thresholding	
	Light	*Dark*	*Light*	*Dark*	*Light*	*Dark*
Objects found	73806.9	68032.5	41609	33510	1843.2	2195.3
Average false positives	73792.3	68017.9	451594.4	33495.4	1828.6	2180.7

Table 4.5: Average amount of false positives found per image depending on the algorithm used.

In the case algorithms are merged, the total amount of detected elements corresponds to the sum of the elements found by each algorithm on its own.

4.5 Filters proposed for noise reduction

As seen in previous section, segmentation algorithms produce always a big amount of false positives. These false positives could be, in part, removed by constraining the ranges of their segmentation algorithms parameters, at the expense of probably covering a narrower spectrum of light conditions, which is not our aim.

We propose instead, to remove objects which do not meet certain features as, for instance, an expected geometry or a minimum contrast with background. These constraints, or restrictions, can be defined by taking into account the environment of the system. By applying them to the output set of symbols of the segmentation algorithms, the number of false positives can be dramatically reduced.

These restrictions will be called filters in the remaining sections. One of the restrictions that objects must meet, for instance, is that their dimensions should range within certain limits, or that their contrast should be greater than a given value.

Also a classifier could be used in order to filter out all those objects that cannot be classified into some predefined classes, and are thus considered as noise. In the case of container codes, objects could be classified as either letter or noise. Any object classified as noise would be rejected and could not be part of the final result.

We define mathematically filters as functions that operate on sets, by selecting objects which do meet given properties. Being $S(I, \Sigma, K)$ a set result of a segmentation algorithm, the function f defined as,

$$f : S(I, \Sigma, K) \rightarrow \{0, 1\} \tag{4.1}$$

associates a value 1 to object o if it meets an specific constraint, or 0 otherwise. With the help of this function, a set can be built, containing the objects which verify $f(o) = 0$ as,

$$\Delta = \{o \in S(I, \Sigma, K) : f(o) = 0\} \tag{4.2}$$

it is straightforward to see that $\Delta \subseteq S(I, \Sigma, K)$ and that filtering the set $S(I, \Sigma, K)$ is just subtracting the objects in Δ from $S(I, \Sigma, K)$ as follows,

$$\Upsilon(I) = \{S(I, \Sigma, K) - \bigcup \Delta_i\} \tag{4.3}$$

After applying the segmentation techniques, two solution sets are available, $S(I_{dark})$ and $S(I_{light})$; applying filters to both sets, one of the sets should be left ideally empty and the other one, should contain the code of the container; the final result will be denoted by $\Upsilon(I_{dark})$ or $\Upsilon(I_{light})$, depending on the image considered in the process.

In the following subsections the designed filters are introduced.

4.5.1 Size filter

This filter, as may be deduced by its name, will set the minimum and maximum dimensions of objects considered as valid. It is the first filter to be executed as it is the filter which more objects remove and it is one of the fastest. For each object $o \in S(I, \Sigma, K)$, bounded by a rectangle R_o,

$$f_{shape}(o) = \begin{cases} 1 & \text{if } height(R_o) \in [h_{min}, h_{max}] \\ & \wedge\ width(R_o) \in [w_{min}, w_{max}] \\ 0 & \text{any another case} \end{cases} \tag{4.4}$$

Being the functions $height(R_o)$ and $width(R_o)$ functions whose results are the height and width, respectively, of the rectangle R_o. On the other side, h_{min}, h_{max}, w_{min}, w_{max} represent the maximum and minimum dimensions in height and width of a valid object.

The set associated with this filter is formally defined as,

$$\Delta_{shape} = \{o \in S(I, \Sigma, K) : f_{shape}(o) = 0\} \tag{4.5}$$

4.5.2 Contrast filter

This filter removes all objects whose contrast is very low. Regions which do not show enough variability are not considered for further processing because it is quite unlikely that they contain a character of the contained code. If the variance of the grey tones of the pixels of object $o \in S(I, \Sigma, K)$ is defined as $\mu(o)$. The filter may be mathematically defined as:

$$f_{contrast}(o) = \begin{cases} 1 & \text{if } \mu(o) >= \mu_{min} \\ 0 & \text{any another case} \end{cases} \tag{4.6}$$

Being μ_{min} a constant which determines the minimum variance that an object must show, in order to be considered as valid.

And the set $\Delta_{contrast}$ defined as:

$$\Delta_{contrast} = \{o \in S(I, \Sigma, K) : f_{contrast}(o) = 0\} \tag{4.7}$$

4.5.3 Classification

The use of a classifier may also help when filtering undesired symbols in the output of segmentation algorithms. By training a classifier with a set of valid symbols and a set of non-valid symbols, false positive symbols may be removed from the output reducing the load in further processes.

A big amount of different classifiers exist, but all of them may be modelled in this case as a function $class(o)$ that gives the class to which the object o belongs, being $o \in S(I, \Sigma, K)$. Along with the class, each object is given a value of confidence that measures how confident the classification is. This confidence value is used in other filters.

As explained in appendix B, this filter was implemented with a k-nn classifier trained with a corpus of 654 images, with an approximate total amount of 9810 symbols.

A function $f_{noise}(o)$ that determines if an object is classified as noise or not is defined as follows,

$$f_{noise}(o) = \begin{cases} 0 & \text{if class(o) = } \textit{Noise} \\ 1 & \text{in another case} \end{cases} \tag{4.8}$$

and the set Δ_{noise},

$$\Delta_{noise} = \{o \in S(I, \Sigma, K) : f_{noise}(o) = 0\} \tag{4.9}$$

4.5.4 Confidence filter

An indicator of the goodness of the solution provided is counting how many elements have a classification confidence over a given threshold ($conf_{min}$). The bigger the number of objects with a high confidence degree, the bigger the likelihood that the result set is valid. The classification confidence is calculated at classification time as explained in appendix B.

The confidence assigned by the classifier to each classification is also borne in mind, this way, only objects with a high confidence are considered in the final decision. Being $confidence(o)$ the confidence of object $o \in S(I, \Sigma, K)$, the filter is defined as,

$$\Delta_{confidence} = \{o \in S(I, \Sigma, K) : confidence(o) < \epsilon\} \tag{4.10}$$

where ϵ is the minimum confidence considered valid.

4.5.5 Fusion filter

Bearing in mind that segmentation algorithms are applied several times on the same image, just ranging their parameters within a given set; it is quite likely that several objects are found with similar coordinates. It is quite likely also that objects found with similar coordinates are really the same object. This filter eliminates objects which are replicated.

Given two bounding boxes R_p y R_q enclosing objects p and q, such that $p, q \in S(I, \Sigma, K)$; the overlap of R_p and R_q, denoted by $R_p \odot R_q$, is defined as the percentage of surface that R_p overlaps on R_q. We define the fusion filter function as,

$$f_{fusion}(p) = \begin{cases} 0 & \text{if } \exists\, q \in S(I,\Sigma,K) : \frac{R_p \bigcap R_q}{R_p} \geq 0.65 \\ & \wedge\, p \neq q \\ 1 & \text{any another case} \end{cases} \tag{4.11}$$

and the set Δ_{fusion} as,

$$\Delta_{fusion} = \{o \in S(I,\Sigma,K) : f_{fusion}(o) = 0\} \tag{4.12}$$

4.5.6 Position filter

Characters are arranged either by columns or by rows on the container and thus, the location of objects on the container and its relative position from to other objects is quite important. Also, characters are always arranged into 2 or 3 columns or rows containing similar amount of objects.

This filter detects the position of objects on the image and tries to determine whether objects are arranged in rows or columns. With this information, it should be easy to remove all objects which are not in the correct row or column just by inspecting the coordinates of its centre, for instance.

The algorithm works as follows, it seeks the two columns with more elements, col_1 and col_2 being c the total amount of elements of both columns. Then, it it seeks for the 2 rows with more elements row_0 y row_1, being f the total amount of elements in both rows. If $c > f$ verifies, then elements are arranged in rows; If $c < f$, in columns. Otherwise nothing can be decided. Two functions are defined, $col(o)$ denotes the column to which object o belongs to and $row(o)$ denotes on which object o is located.

Once the arrangement of objects is detected, the algorithm removes all objects which do not lay in the valid columns or rows. Depending on the case, one of the following two functions would be used,

$$f_{column}(o) = \begin{cases} 1 & \text{if } col(o) \in [col_0, col_1] \\ 0 & \text{in another case} \end{cases} \tag{4.13}$$

$$f_{row}(o) = \begin{cases} 1 & \text{if } row(o) \in [row_0, row_1] \\ 0 & \text{any another case} \end{cases} \tag{4.14}$$

and the set Δ is built depending if the arrangement is done in columns or rows as,

$$\Delta_{rows} = \{o \in S(I,\Sigma,K) : f_{rows}(o) = 0\} \tag{4.15}$$

$$\Delta_{columns} = \{o \in S(I,\Sigma,K) : f_{columns}(o) = 0\} \tag{4.16}$$

4.5.7 Decision of the tone of the characters

If we recall both expressions for $\gamma(I_{white})$ and $\gamma(I_{black})$, with the filters designed before we have:

$$\begin{aligned} \gamma(I_{white}) \quad = \quad & ((((((S(I_{white}) - \Delta_{shape}) \\ & - \Delta_{contrast}) - \Delta_{classifier}) - \Delta_{confidence}) \\ & - \Delta_{fusion}) \end{aligned} \tag{4.17}$$

$$\begin{aligned}\gamma(I_{black}) &= (((((S(I_{black}) - \Delta_{shape}) \\ &\quad - \Delta_{contrast}) - \Delta_{classifier}) - \Delta_{confidence}) \\ &\quad - \Delta_{fusion})\end{aligned} \tag{4.18}$$

A decision can be taken in order to decide which is the valid set. If the constraint 4.19 is met, the valid set will be the one containing the light symbols. If, otherwise, the constraint 4.20 is met, the valid set is the one containing the dark symbols and if 4.21 nothing can be decided.

$$|\gamma(I_{white})| > |\gamma(I_{black}| \tag{4.19}$$

$$|\gamma(I_{white})| < |\gamma(I_{black})| \tag{4.20}$$

$$|\gamma(I_{black}| = |\gamma(I_{black}| \tag{4.21}$$

being $|\gamma(I_{white})|$ the cardinality of set $\gamma(I_{white})$.

It must remarked that, though it may seem that the order in which filters are applied does not matter, it really does. Some filters may be applied in any order as, for instance, size and contrast filters; this is due to the fact, that they just consider geometric or statistical properties of an object at a time. They are used first, also because they remove a big percentage of irrelevant objects, and because they execute very fast, compared to others.

Classification also takes one object at a time, but it is much more time consuming that the previous. This is one of the reasons to execute it in the last stages of filtering, to avoid losing time.

On the other hand, the fusion filter is executed at the end of the process because it takes the area shared by several objects. If it was applied in the first stages, we would be at risk of removing valid objects of the contained code, as long as these valid objects could be fused with others which would be later removed by any of the other filters (geometric, contrast...). Also, the execution time would be affected by the number of comparisons.

4.5.8 Code extraction

The codes of containers are extracted by appending all symbols which successfully went through all the filters. The class of each remaining symbol is obtained in the classification step, see appendix B, and corresponds to the character it represents.

4.5.9 Filtering segmentation results

As with segmentation algorithms alone, experiments were performed to test the behaviour of these filters.

The position filter was not used because preliminary experiments did show, that it was very sensitive to noise. Even just a few objects appearing in any part of the container, did mislead this filter taking it to remove most of (if not all) valid objects of the code. Thus, it was not considered for further processing.

Values for filters' parameters were set manually and tuned with a reduced set of images. For the contrast filter a value $\mu_{min} = 200$ was chosen; being low enough as to

avoid losing symbols due to shadows, but keeping the number of noisy symbols low. For size, constraints are: width in the range $[20, 50]$ and height in the range $[6, 30]$. We established a confidence level of 80%. Table 4.6 shows how the amount of objects decreases as filters are applied.

Algorithm	*Tone*		*Initial amount*	*Size filter*	*Contrast filter*	*Noise filter*	*Fusion filter*
LAT	Dark	*Total*	73806.9	978.5	222.7	205.2	14.5
		Left		0.013	0.22	0.92	0.07
	Light	*Total*	68032.5	825.5	216.4	158.7	16.4
		Left		0.012	0.25	0.73	0.10
Watershed	Dark	*Total*	41609	1236.7	868.7	822.7	14
		Left		0.03	0.70	0.94	0.01
	Light	*Total*	33510	1173.9	902.5	795	16.5
		Left		0.03	0.76	0.88	0.02
LAT	Dark	*Total*	1843.2	218.1	207.1	191	13.2
		Left		0.12	0.95	0.92	0.07
	Light	*Total*	2195.3	189.3	183.1	160.2	13.5
		Left		0.09	0.97	0.87	0.08

Table 4.6: Average computation of the amount of remaining symbols after each filter is applied, depending on the segmentation algorithm used. In row *total*, the total amount of symbols left after the filter is applied to the input. The row *Left* shows the percentage amount of objects left by the filter with respect to previous step.

4.5.10 Code extraction results

Experiments aimed to analyse the performance of the method extracting codes, were made with 309 images. Results are shown in figure 4.10 and table 4.7. First row in the table shows the amount of images in each segmentation which were successfully recognized (i.e., with zero errors). Columns show the result of the complete process for each algorithm, ordered according to the number of symbols not recognized successfully.

Missed characters	LAT	Wat.	Thr.
0	25	13	22
1	55	25	39
2	47	37	46
3	55	44	40
4	28	49	39
5	26	38	29
6	14	27	19
more than 6	59	76	75

Table 4.7: Performance of the recognition process, using stand-alone segmentation algorithms and filters.

In these experiments, results are measuring not only the performance of the segmentation algorithms, but also the performance of the filters implemented. One of the most sensitive steps in the process is the classifier used. After reviewing which symbols had been missed, it was decided to improve the classifier by adding more instances of the different expected symbols.

The incorrectly classified symbols of the first experiments were used to improve it as explained in appendix B. Results of this new classifier applied to a new set of 312 images is shown in figure 4.11 and table 4.8. As it may be seen, improving the classifier improves significantly results.

As proposed in previous section, performance could be improved by using several segmentation algorithms. Filters are applied to each segmentation algorithm's results

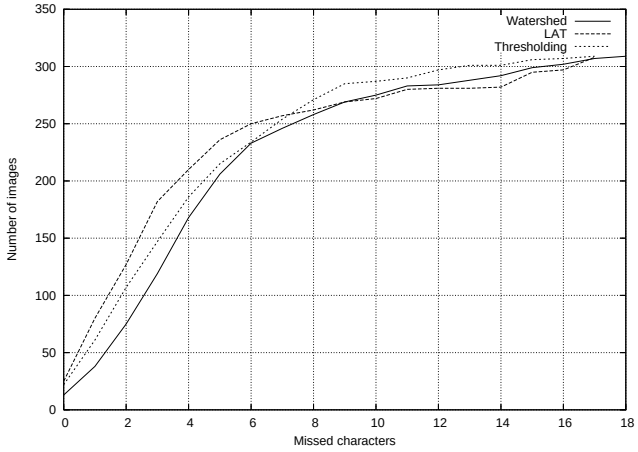


Figure 4.10: Cumulated plot of images according to the number of lost symbols , using stand-alone segmentation algorithms and filters.

Missed characters	LAT	Wat.	Thr.
0	48	13	41
1	62	35	69
2	62	35	52
3	41	36	32
4	17	33	23
5	14	28	28
6	16	36	21
more than 6	52	96	46

Table 4.8: Performance of the recognition process with the improved classifier.

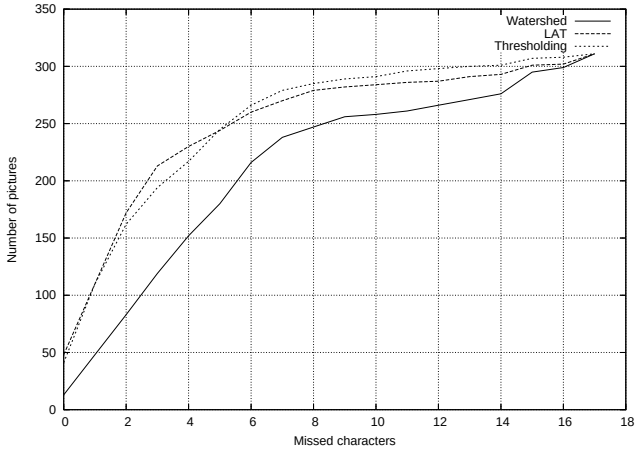


Figure 4.11: Cumulated plot of images according to the number of lost objects using the algorithms and the improved classifier.

separately and then, results are joined into one only list. Results of these experiments can be seen in table 4.9 and figure 4.12.

Missed characters	LAT+Thr.	LAT+Wat.	LAT+Thr.+Wat.
0	102	91	163
1	87	90	72
2	49	43	19
3	23	30	6
4	10	8	7
5	11	10	2
6	11	8	1
more than 6	18	31	41

Table 4.9: Performance of the merged algorithms. Number of images correctly detected depending on the amount of missed symbols.

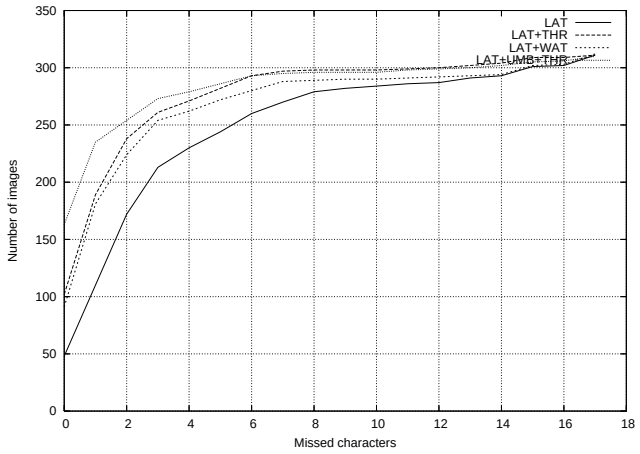


Figure 4.12: Results of merged algorithms.

Execution time of the process with segmentation techniques together with filters, is shown in table 4.10. The process becomes very time consuming when it is Watershed the chosen segmentation technique. On the other side, LAT or the thresholding algorithm used together improve results with a low penalization in time execution. From result plots may be seen that, for instance, in case 2 symbols could be missed, using LAT plus the thresholding technique nearly $76,28\%$ of images could be successfully recognized. A sample application of this schema to an image representing a container may be seen in figure 4.13.

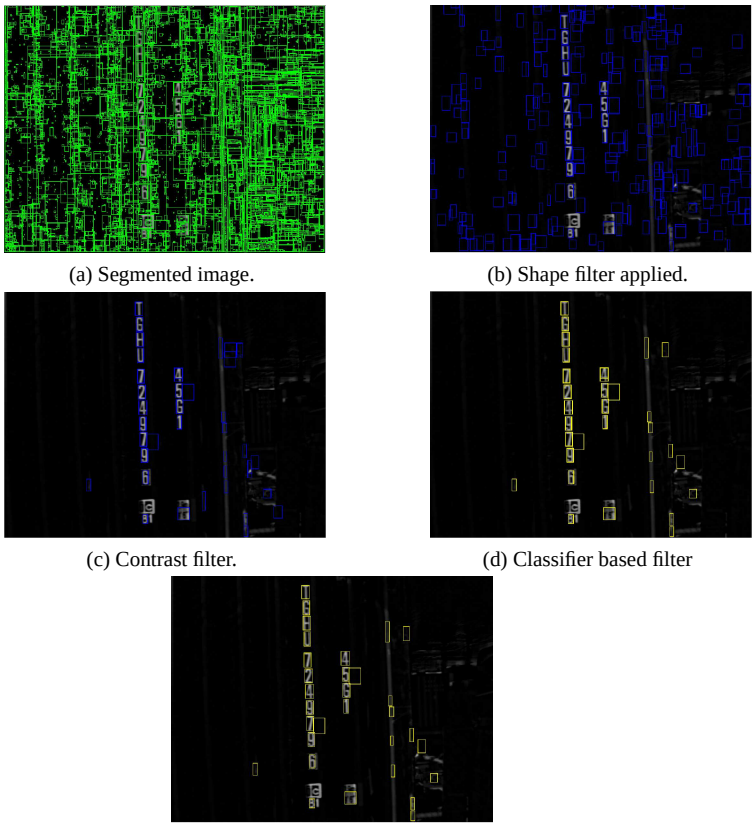
Mean time \ Alg.	LAT	Water.	Thres.	LAT-THR	LAT-WAT	LAT-WAT-THR
seconds	2.99	14.81	1.53	4.36	16.56	18.33

Table 4.10: Mean execution time. From left to right, the three implemented algorithms and the results of joining them.

Finally, in table 4.11, the percentage of successfully character tones detection in images, see section 4.5.7, is shown.

Mean time \ Lg.	LAT	Water.	Threes.	LAT + Threes.	LAT + WAT	LAT + Threes. + WAT
Colour detected	0.93	0.85	0.94	0.96	0.93	0.95

Table 4.11: Percentages of successfully detected colours using the segmentation algorithms together with filters. Columns determine the algorithm used, and rows the percentage of successfully detected container codes.



(a) Segmented image.

(b) Shape filter applied.

(c) Contrast filter.

(d) Classifier based filter

(e) Fusion filter

Figure 4.13: Sample evolution of the process applied to an image searching for light characters. a) The result of segmenting the image with LAT. b) After applying the shape filt er. c) After applying the contrast filter. d) After removing noisy regions. e) After fusing regions.

4.6 Use of sequences of images

Results obtained so far may be considered satisfactory, from the point of view that just one image was used to obtain the truck code. Analysing results, it may be seen that in some occasions, symbols of the container code are lost. Three causes are detected as the most important:

- Shadows which become confused with symbols or do totally or partially cover them.
- Classification errors, for instance, classifying noise as a valid symbol.
- Confusions in the classification, for instance, a symbol "B" confused with an "8".

It is possible taking advantage that several pictures per container are available. Each one of these images was taken as the truck was approaching the port's gate. The time span between two consecutive images may be enough to change the light conditions on the container, revealing symbols that could be affected previously by shadows (maybe shadows of the own texture of the container or casted by external factors), which were not visible in previous images. Some sample sequences may be seen in figure 4.14.

For the first problem, shadows, we trust that, by taking several pictures of the container, we will be able to find the characters of the code in most of the cases. If eventually a character is shadowed in an image, it is likely that it was found in other images, so, by checking all them together, we can have it in the final solution. The second issue, is easier to achieve, noisy objects classified as characters will appear only in a small amount of images, thus, if we set the constraint of appearing in a minimum number of images, noisy objects will be removed from the final solution.

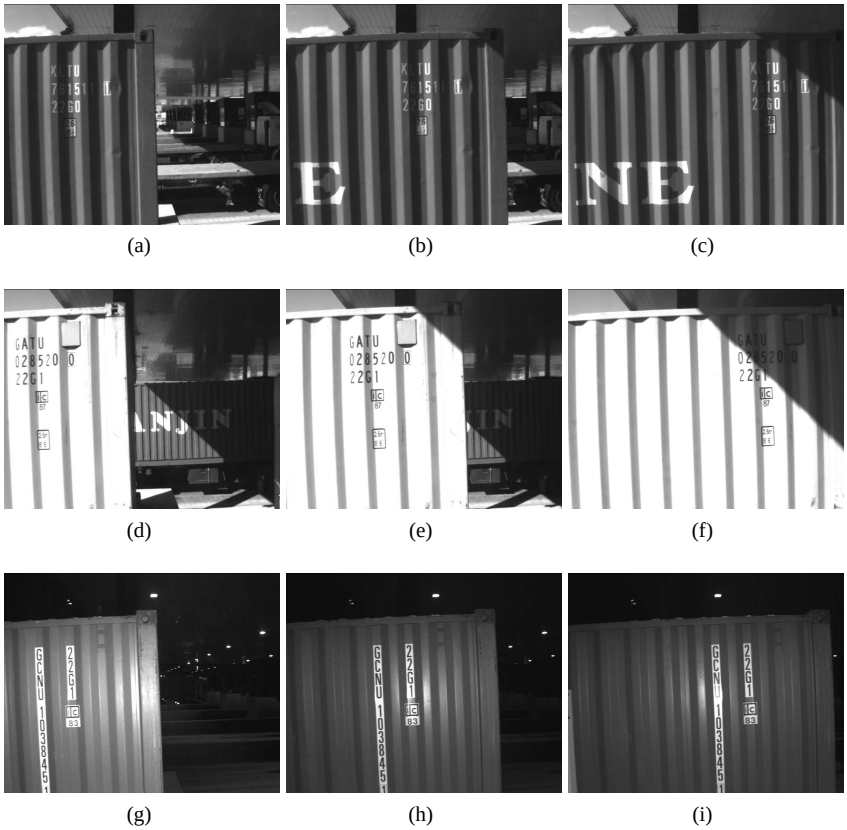
The last problem has to do with characters which are misclassified due to, for instance, a bad segmentation of the image or a classification error. As in the first case, if such a problem appears, we can still know which is the correct one by keeping only characters which appeared in a minimum number of images of the sequence.

In this section we explain how we process a sequence of images to extract the container's code from it. We first try to group objects (which are the result of applying the previous algorithm to each image in the sequence) in each image into clusters of close objects. This way, we can use properties of clusters such as the centre of mass of the cluster to faster overlap objects of a reference image with the objects in the following images.

4.6.1 Clustering objects

The algorithm used to process images is divided into two steps; first, the cluster identification in each image and second, the processing of the sequence itself. Both steps are described thoroughly in the following sections.

Code symbols in the container appear following a defined structure. Though we do not know before taking the picture what kind of lay-out the characters will have and either the amount of valid objects in the image, it is true that after the processing, we can apply some knowledge on the resulting objects in order to get rid of those which are not likely to be in the code, but have gone trough up to this stage of the processing. For instance, symbols belonging to the code appear close one to each other; we can use this feature to group objects in clusters and use only the valid cluster, i.e. the cluster with the code, to perform further calculations.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 4.14: Some sequences of truck containers, taken in different day-times. In each row, a different sequence is represented. Brights and shadows may be appreciated in images, on different parts of the container.

Given an image I, we obtain the set $\gamma(I) = \{r_0, r_1..r_n\}$, the set of objects found in the segmentation of image I that do meet some constraints (as mentioned in the previous section). We calculate the clusters of $\gamma(I)$, denoted by $\{\kappa_k\}$, as sets containing objects $r_i \in \gamma(I)$ whose euclidean distance must be less or equal to a given ϵ. Algorithm 1 illustrates this process.

Algorithm 1 Algorithm for sequence processing

1: $\forall\, r_i \in \gamma(I) : \kappa_i^0 \leftarrow \{r_i\}\ , t \leftarrow 0$
2: **repeat**
3: $\quad t \leftarrow t + 1$
4: $\quad \forall i\ \kappa_i^t \leftarrow \kappa_i^{t-1}$
5: $\quad \forall\ \kappa_i^t, \kappa_j^t \wedge i \neq j\ , if\ \exists\ r_p \in \kappa_i^t \wedge r_m \in \kappa_j^t : distance(r_m, r_p) < \epsilon \Rightarrow \kappa^t \leftarrow \kappa_i^t \cup \kappa_j^t\ \wedge \kappa_j^t \leftarrow \emptyset$
6: **until** $\forall\ i,\ \kappa_i^t = \kappa_i^{t-1}$
7: $\kappa \leftarrow \kappa_i : i = argmax_j(|\kappa_j|)$

After running this algorithm, it is assumed that the cluster κ with the biggest amount of objects will contain the truck container code. We calculate the centre of mass of this cluster. This centre of mass will be useful in the following algorithm, when we try to match objects in one image with objects in another.

The centre of mass of a cluster κ is computed as the average coordinates in the x and y axes of the centres of each of the objects $r_p \in \kappa_i$. By using this value, the translation between the objects of different clusters can be done easily by computing the translation between centres of masses. This is proposed so because clusters will probably have a similar lay-out on the image.

4.6.2 Processing a sequence

A sequence of images Ψ is a set of images ordered in time $\Psi = \{I_0, I_1, ...I_{n-1}\}$. The result of segmenting the sequence Ψ is a set denoteb by $\gamma_s(\Psi)$.

Each object p in the solution set $\gamma_s(\Psi)$ has an associated counter that tells how many times it appears in different images, accessed as $p.counter$. An auxiliary function $overlap(p, q)$ is used, this function returns 1 in case objects p and q overlap and 0 otherwise. Function $symbol(q)$ returns the symbol associated to this object by the classifier. The function $centreOfMass(\gamma)$ computes the centre of mass of the objects belonging to the set γ. Each object p in the solution set $\gamma_s(\Psi)$ which was not found at least in half of the images of the sequence is discarded. The process is formalized by algorithm 2.

4.6.3 Experiments

We used 51 real images to perform our experiments, corresponding to 17 containers. These images represent truck containers and have a size of 720×574 pixels in grey levels. They were acquired under real conditions in the admission gate of the port of Valencia; in several days under different light conditions. Digits and characters can be clear or dark and they appear in both plain and non-plain surfaces. We selected randomly a set from a large amount of pictures and assured all variability was represented in this set of pictures (sunny or cloudy days, daytime or night-time, damaged containers...).

Algorithm 2 Algorithm for sequence processing

1: Given a set of images $\Psi = I_0, I_1, I_2, ...I_{n-1}$, we take a reference image I_0.
2: $\gamma_s(\Psi) \leftarrow \gamma(I_0)$
3: $c^0_{x,y} \leftarrow CentreOfMass(\gamma(I_0))$
4: $\forall\, p \in \gamma_s(\Psi)\ p.counter \leftarrow 1$
5: **for all** $I_i \in \Psi, i > 0$ **do**
6: $c^i_{x,y} = CentreOfMass(\gamma(I_i))$
7: Compute T translation between point $c^0_{x,y}$ and $c^i_{x,y}$
8: Apply T to $\gamma(I_i)$
9: $\forall\ p \in \gamma_s(\Psi), p.counter \leftarrow p.counter + 1 \iff \exists\ q \in \gamma(I_i) : overlap(p,q) = 1 \wedge symbol(p) = symbol(q)$
10: $\gamma \leftarrow \{q \in \gamma(I_i) : \nexists\, p \in \gamma_s(\Psi) : overlap(p,q)\}$
11: $\forall\, p \in \gamma'\ : p.counter \leftarrow 1$
12: $\gamma_s(\Psi) = \gamma_s(\Psi) \bigcup \gamma'$
13: **end for**
14: $\gamma_s(\Psi) = \gamma_s(\Psi) - \{p \in \gamma_s(\Psi) : p.counter \leq \frac{n}{2}\}$

In table 4.12, results are shown for some sequences. Under the column objects, we show the number of valid objects in the container, that is, the number of characters in the container's code. Under columns image0 to image2, we show results for each image on its own, first number stands for the number of valid objects found in this image, and the second number stands for the number of noisy objects that could not be removed; the addition of both numbers gives the amount of objects found in the picture. In case a sequence had less than 3 images, we filled the corresponding cells with dashes. The last column, gives the same information after applying the sequence algorithm.

Sequence	*Objects*	Img0		Img1		Img2		Process	
		H	*N*	*H*	*N*	*H*	*N*	*H*	*N*
1	*15*	15	6	15	3	11	0	15	0
2	*17*	14	5	12	3	11	6	15	1
3	*15*	12	22	12	10	11	5	12	0
4	*15*	14	2	14	2	4	0	11	1
5	*15*	15	5	15	10	9	6	13	3
6	*15*	10	10	13	5	14	4	11	3
7	*15*	13	16	10	9	0	5	5	0
8	*15*	13	3	15	0	14	1	12	0
9	*15*	14	6	14	6	11	4	13	2
10	*15*	15	6	15	4	11	3	15	1
11	*16*	13	4	15	4	12	2	11	2
12	*15*	10	13	11	3	11	0	11	0
13	*17*	12	10	14	10	11	7	12	4
14	*15*	15	16	15	4	-	-	15	0
15	*15*	14	13	14	12	13	9	14	2
16	*17*	16	1	16	9	-	-	15	1
17	*15*	15	18	15	13	-	-	15	2

Table 4.12: Comparison of results looking symbols in just one image or merging the results of the complete sequence.

In each row we show results for each image of a sequence and the result for the sequence by applying the sequence algorithm. For instance, first row corresponds to first sequence. For first image of this sequence, we obtain 15 correct objects and 6 noisy objects, that is, 30% of the objects are noisy objects; for the second image we

obtain again 15 correct objects and 3 noisy objects; on the other side, by applying the sequence algorithm we obtain 15 objects that correspond exactly to the container's code. In sequence 8 however, the sequence algorithm loses 3 characters of the solution, and in any of the images of the sequence on their own we would have more hits, but, on the other side, the result of the algorithm has no noisy object and it is difficult to choose which picture in any sequence would have the best results.

Though using the sequence algorithm may be seen as adding a time penalty to the execution, it may bee seen that, it is impossible to find all objects just with one image. In addition, the process itself can be implemented in such a way, that time penalty does not influence the final result. It could, for instance, be divided into two different processes, one taking pictures, segmenting them and locating objects on them; and the other one, taking these objects and comparing them with the previous following the sequence algorithm presented now. These two processes follow the producer-consumer paradigm and that can be easily parallelized.

4.7 Conclusions

In this chapter, a process developed to segment and detect container codes in images is proposed ([rosell06], [rosell06a], [rosell06b]. In order to detect the symbols of the code with the highest precision, meeting real-time constraints, a system using the top-hat operator, several segmentation techniques and filters is designed and tested.

In the experiments it is shown that parallelizing several segmentation techniques has the benefit of reducing the number of lost objects due to the limitations of the techniques, enhancing results.

After performing experiments, three well-known algorithms were chosen to segment images, merging afterwards their outputs.

The problem of a large amount of false positives is solved by using filters which remove all the symbols in the output which do not meet problem-specific constraints.

Image processing is performed in two different processing branches, one for light symbols and another one for dark symbols. By filtering the output of each processing branch, we can determine which the correct tone was and obtain the container's code.

Finally, fault tolerance is improved by performing this individual process on several images representing the same truck container in different time snaps. An algorithm is proposed which processes a sequence of images individually, and then gathers all this information to extract the container's code.

Chapter 5

Low level vision developments for SENSE project [1]

This chapter introduces the low-level video processing algorithms developed for an intelligent node that is part of a distributed intelligent sensory network for surveillance purposes (within the SENSE project). Details of the software architecture developed for this node are given, together with the low-level video processing algorithms used, and the results obtained after their implementation, as well.

The low-level software includes acquisition, segmentation, tracking and classification of detected objects into three main categories [rosell08]: *person*, *group of people* and *luggage*. The experiments performed with the aim of finding the best set of features for classifying the objects are discussed in this chapter. The unit has to communicate the classification results and the main features obtained using XML streaming to upper levels, as well as the processed frames, using a JPEG stream. All these functionalities are currently running in the built prototypes [benet10].

5.1 Introduction

Visual surveillance is an active research topic in computer vision and some different surveillance systems have been proposed in recent years:[wren97], [haritaoglu00], [vsam99]. The aim is to develop intelligent systems that give support to humans involved in surveillance, by calling their attention when abnormal situations are detected. In order to achieve this goal, a real time analysis of input video is needed. In general, the processing framework of visual surveillance in dynamic scenes includes the following stages: modelling of environments, detection of motion, classification of moving objects, tracking, understanding and description of behaviours, human identification, and fusion of data from multiple cameras. Recent developments and general strategies for all these stages are reviewed in [hu04].

Distributed smart cameras have received increasing focus in the research community over the past years [fuentes03], [nguyen03], [hengstler07]. The notion of cameras combined with embedded computation power and interconnected through wireless communication links opens up a new realm of intelligent vision-enabled applications [foresti05].

[1] Sixth framework programme priority IST 2.5.3 Embedded systems. Project 033279.

Real-time image processing and distributed reasoning made by distributed smart cameras can not only enhance existing applications but also instigate new applications [mckenna00], [sacchi01]. Potential application areas range from home monitoring and smart environments to security and surveillance in public or corporate buildings. Critical issues influencing the success of smart camera deployments for such applications include reliable and robust operation with as little maintenance as possible.

In this line, the SENSE [sense04] project undertakes the task of developing a distributed intelligent network of sensory units that aid to describe their environment in a cooperative way.

5.1.1 Goals and constraints

The goal of the process introduced in this chapter is to propose vision techniques aimed to detect, track and give a first rough classification of people and objects standing or moving in images acquired in areas of an airport. Specifically, the objectives are:

- To study vision techniques capable of detecting objects which do not belong to background and producing the most accurate segmentation possible.
- To study the most suitable features in order to classify objects automatically into one of the classes *person, group of people* or *luggage* in images obtained in the surveillance context.

The main challenges to cope with are,

- Cameras are stationary.
- The background is dynamic, it is affected by light changes but also by objects that may be left intentionally in the background and must be, after a reasonable period of time, considered as background as well.
- System must response to stimuli in real-time.

5.1.2 SENSE project

This section describes the video node in this SENSE architecture, together with the results obtained with the algorithms developed for the video modality. Figure 5.1, shows an schema of the general framework of video modality.

SENSE intends to overcome problems with current centralized networks. SENSE uses a completely decentralized approach. The system consists of a number of identical, autonomous acting entities, or nodes, mounted at fixed locations. Each entity has sensors with static sensing parameters (i.e. intelligent camera and microphones modules), gathers information from its surroundings, and interprets it. Consequently, each entity can be seen as a standalone system. However, entities can share their knowledge with neighbouring nodes, acquiring information indirectly from other sensors. By fusing it with its own information, a global view is created autonomously.

Each node within the overall SENSE system will be able to process its own sensory data and communicate with other local nodes to build a shared understanding of objects and events how they are related across nodes and modalities, and how they are related to the environment. Key to this distributed intelligence is the concept of a node interacting with its neighbours. For example, as figure 5.2 shows, if a person walks in front of a sensor in a given direction, then that person may also walk in front of a neighbour

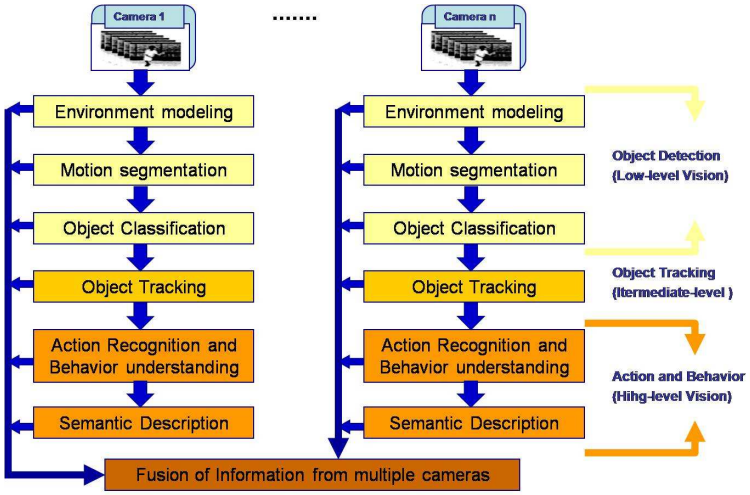


Figure 5.1: General framework of video modality.

sensor a short time later, in a direction influenced by the positioning of the node sensors relative to each other. Frequent repetition of this pattern will result in the two sensors detecting this correlation, and using it both to increase the dependability of their own observations, and to establish common views which finally should help to learn about for instance usual paths over sensor boundaries. The topology of the network is thus developed over time, and reflects the degree to which nodes can correlate their observations and thus help each other to draw conclusions about their environment, rather than a designer-defined notion of neighbourhood.

Figure 5.2: Two different cameras whose fields of view overlap. Note that the person in the field of view of camera 1 is also seen by camera 2.

The components of a SENSE node are organized modularly in terms of their functionality [benet10]. These components are:

- **RDU (Reasoning and Decision-making Unit)**: This module correlates information from all sensorial components in the node as well as information received from other SENSE nodes working in the neighbourhoods. This module is the responsible of triggering alarms and selecting which information is offered to the rest of SENSE nodes. It decides the working mode (e.g. indoors, parking area, etc.) and propagates this modality to the sensorial components.

- **EVS (Embedded Video Subsystem)**: It acquires and processes video images extracting features in a modal way. The features extracted are communicated to the RDU to enrich the perceived state of SENSE node surroundings. This component offers an interface with higher level software to control the modality, QoS and other special features.

- **EAS (Embedded Audio Subsystem)**: It acquires and processes multiple audio signals extracting features in a modal way. As the previous module, it sends its computed features to the RDU and offers an interface to higher level software.

- **WCS (Wireless Communication Subsystem)**: This module controls the wireless communications flow in a neighbourhood of SENSE nodes.

- **CIM (Communications and Interface Module)**: This module is the hardware and software communication interface that provides the way to share information and control among modules in a SENSE node.

Following sections outline the proposed system and the experiments performed with it. Section 5.2 explains the algorithm used to segment images and section 5.3 describes the creation and update of the background model, section 5.4 shows the segmentation schema followed, the following section discusses how the output of the segmentation is filtered to remove as much noise as possible. The process followed to classify objects is explained in section 5.6. The tracking system is discussed in section 5.7 and finally, the experiments performed with the classification features, which are the central contribution of this chapter, are discussed in section 5.9.

5.2 Proposed processing for video modality

The system proposed, in the SENSE context, will process images representing different scenes, such as halls, corridors, different views of check-in desks and other facilities. The main two issues to control are the motion of people and the location of unattended luggage. In figure 5.3, some sample frames showing normal situations in an airport are shown.

The process of detecting objects and providing them with a first rough classification is called, in this application, *video modality*. This video modality passes information to upper levels and is not focused to directly solve the problem of video surveillance.

The three classes of objects which this system deals with are *person*, *group of people* and *luggage*. Being class *person* a single person with or without luggage and *group of people* is a group of 2 or more people with or without luggage.

An outline of the different steps involved in the scene processing is represented in figure 5.4. Images will be captured by a camera and processed to detect differences with the background, in order to locate objects in the image.

The technique chosen to segment objects in frames is background subtraction (see chapter 3), which is a popular technique in video surveillance.

In order to remove false positives, regions detected in the segmentation step are filtered by size. Afterwards, the remaining regions are classified and tracked.

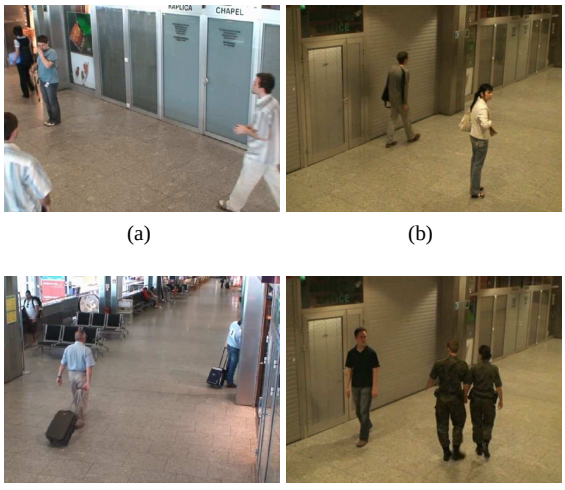


Figure 5.3: Sample images of typical recordings from the airport aim of the surveillance.

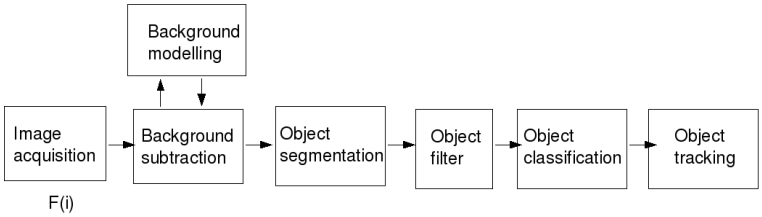
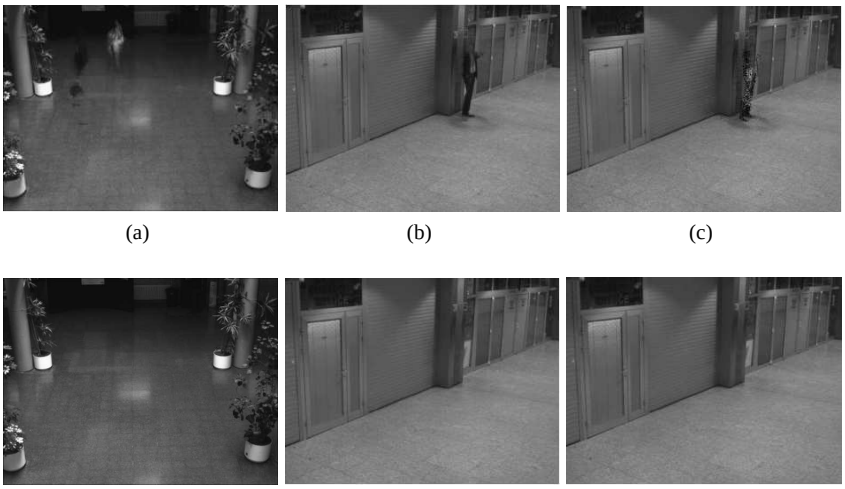


Figure 5.4: The different phases of the proposed processing schema. After preprocessing $F(i)$, the segmentation stage provides the system with a set of regions of interest. These regions are filtered according to problem dependent criteria and then classified and tracked.

5.3 Background modelling

The system acquires a sequence of frames $F(0), F(1), ..., F(i)$ ordered in time with a given rate of frames per second. At start time or after a restart the system computes a background model denoted by $B(i)$, using the median of the last m frames, and the system starts normal operation from frame $m + 3$.



(a) (b) (c)

(d) (e) (f)

Figure 5.5: First row shows background models built with, from left to right, mean (figure 5.5a), median (figure 5.5b) and mode (figure 5.5c) of a set of consecutive frames with activity. Note that some areas of the model are blurred due to this activity. Bottom row shows models which were built with frames with low or null activity, from left to right, mean (figure 5.5d), median (figure 5.5e) and mode (figure 5.5f); these models are more accurate than those shown in the row above. In all cases, 30 consecutive frames were used to compute the models.

There are several methods proposed in the literature to construct a background model as explained in chapter 3. In the first developments of the system, the mean, mode and median of several images were used to compute background models of the scenes. These models are characterized because they are unimodal, i.e., only one model per pixel is considered. Usually, for indoor scenes it is the best method, as authors of [benezeth08] point out.

In our system, we model an initial background $B(k)$ taking a set of frames at start-up, $F(0), F(1), ..., F(k-1)$ and computing,

$$B(k) = mean(F(0), F(1), ..., F(k-1)) \tag{5.1}$$

The background model is updated over time in a frame by frame basis, though for indoor scenarios the update can be delayed several frames. The expression used to update the model is,

$$B_{x,y}(i) = \alpha \cdot B_{x,y}(i-1) + (1-\alpha)F_{x,y}(i) \ , \forall x, y \tag{5.2}$$

being α an update factor in the range $[0, 1]$, which controls the speed at which new information is included in the model.

One of the problems of unimodal methods, is that they cannot determine whether a suitable background model is built or not. If moving objects are present in the frames used to construct the background, blurred areas may appear as a mixture of the values of the objects and the values of the background will be done. Figure 5.5 illustrates this problem with some sample background models.

The quality of the background model determines dramatically how good or bad the segmentations will be; thus, it is crucial being able to provide the system with a good background model.

Current techniques rely on building a first model which will not ever become corrupt, that is, the model gives up working correctly. Authors of [toyama99] propose maintaining a database of background models and choose the most suitable for the situation. However this may take to maintain an extremely huge database with different models, though in their paper, authors consider only two different models per scenario.

This lack of control over the quality of the model is not addressed currently in the related literature. In next chapter, section 6.2 introduces the concept of model's quality as defined in our algorithms.

5.4 Segmentation

Once the background is modelled, objects moving in the scene may be found by using background subtraction. Each incoming frame $F(i)$ is subtracted from the background model $B(i-1)$. Instead of using the classical background subtraction,

$$d_{x,y}(i) = |F_{x,y}(i) - B_{x,y}(i-1)| \tag{5.3}$$

we favour detecting pixels in dark areas by using,

$$\bar{d}_{x,y}(i) = \frac{|F_{x,y}(i) - B_{x,y}(i-1)|}{B_{x,y}(i-1)} \tag{5.4}$$

Image $S_{x,y}(i)$ is the thresholded version of $\bar{d}_{x,y}(i)$ and can be defined as,

$$S_{x,y}(i) = \begin{cases} 1, & \bar{d}_{x,y}(i) > T \\ 0, & \bar{d}_{x,y}(i) \leq T \end{cases} \tag{5.5}$$

The binary image $S_{x,y}(i)$ contains the pixels which are different from the background model. Sets of connected pixels in $S_{x,y}(i)$ are considered as a global entity called blob from now on, each blob represents a segmented object.

5.5 Filters

After segmenting the incoming frame and grouping pixels into blobs, blobs are filtered in order to reduce the overhead of following steps, as shown in figure 5.4. A size filter is applied to blobs discarding those which could represent noise. This is done by removing all blobs whose area is under a given threshold or whose dimensions are small. For each blob the following two features are computed,

- BlobArea : number of pixels of the blob.
- Bounding box (Bbox): minimal rectangle that encloses the blob.

Every object, whose dimensions are considered as valid by this filter, is classified in the next step as belonging to the three classes considered in the system: *person*, *group of people* and *luggage*.

5.6 Feature selection

The extraction of meaningful features is an important issue for any image processing algorithm. In order to choose a suitable feature set, we took in mind two important considerations:

- The feature set should permit a quick computation and classification.
- Appearence, properties and differences between the three considered classes: *person*, *group of people* and *luggage*.

After reviewing the literature and performing some informal laboratory tests, we finally decided to work with two feature sets, *geometric features* and *foreground pixel density*.

Both sets of features, geometric and foreground pixel density, attempt to extract the essence of shapes and blob's silhouettes. Unfortunately, shadows can connect separate blobs, deform values associated with shapes and confuse classifiers. A previous work was performed to determine whether removing shadows was useful in order to obtain a higher classification rate even with the evident time penalty that would suppose.

After analysing the experimental results of the feature sets aforementioned, a new feature, *number of heads*, was proposed in order to improve performance.

The following subsections discuss the features outlined in the previous paragraphs.

5.6.1 Geometric features

Geometric features are mentioned in the literature discussing surveillance systems [vsam99] and those which seemed to be the most efficient, such as dispersedness, aspect ratio, and others were chosen. The constraint of real-time response can only be met if the most efficient in classification success rate and computational cost are chosen. The selected features were the following:

- Dispersedness: this feature is computed for each blob as,

$$\frac{BlobPerimeter^2}{BoundingBoxArea} \tag{5.6}$$

- Inverse dispersedness: The inverse of the previous one. It is computed as,

$$\frac{1}{4\pi}\frac{BoundingBoxArea}{BlobPerimeter^2} \tag{5.7}$$

  in order to normalize it in the range $[0, 1]$.

- Extent: the proportion of the pixels in the bounding box that are also in the region, computed as,

$$\frac{BlobArea}{BoundingBoxArea} \tag{5.8}$$

- Solidity: the proportion of pixels in the convex hull that are also in the region; computed as,

$$\frac{BlobArea}{ConvexArea} \tag{5.9}$$

Geometric features reduce the complexity of a blob's silhouette into a simple result that may be the same for very different silhouettes, which can be a disadvantage when classifying different objects whose silhouettes are similar or produce similar results.

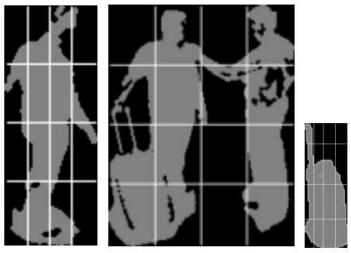


Figure 5.6: Samples of blobs representing objects of classes *person*, *groups of people*, and *luggage* and their bounding boxes with a 4×4 grid with a total amount of 16 cells.

5.6.2 Foreground pixel density

Following ideas found in [gepperth05], a classifier based on the average density of foreground pictures in areas of the blobs was developed.

Paying attention to figures; either a person, groups of people, or luggage show a different pattern of occupancy. It can be seen that, for instance, blobs representing people exhibit a pattern of low foreground pixels density in the areas of the bounding box close to its borders; while luggage usually has a greater density of foreground pixels in nearly all the bounding box except in the top areas. This leads us to think that the density of foreground pixels measured in a regional basis could be a good feature set for classifying the blobs.

We compute these features by dividing each bounding box containing a blob into a grid of $n \times m$ cells. For each cell C_i, the amount of foreground pixels (P_i) and background pixels (B_i) is calculated and the result of the division is stored in a vector $C = \{C_0, ..., C_i, ...C_{n \times m-1}\}$ where,

$$C_i = \frac{P_i}{P_i + B_i} \tag{5.10}$$

As each bounding box is always divided into the same number of cells, this method already incorporates the requirement of scale invariance. This way, we have a set of values pointing out where the majority of foreground pixels is likely to be. For instance, people are expected to have their maximum densities in the area of the body. On the other hand, luggage is expected to have its foreground pixels more uniformly distributed. For groups, foreground pixels are expected to be spread all over the bounding box, but as for single people, they should be more concentrated in the mid-area of the blob, where the bodies should be found. An example can be seen in figure 5.6.

The features described so far can be used to train a classifier in order to distinguish which class a blob belongs to, according to its silhouette. This works reasonably well when blobs are very different. In our case, however, experiments will show that classes *person* and *group of people* have silhouettes which are difficult to model using the previous features and, depending on how people inside the group is arranged and seen by the camera, the silhouette of a person and of a group may be quite similar.

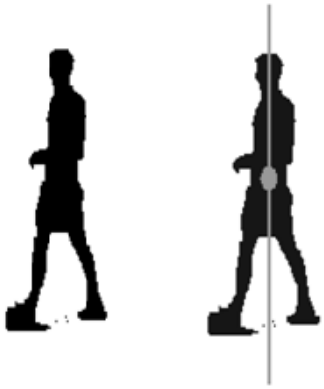


Figure 5.7: An illustration of the symmetry axis of a person approximated by the axis that crosses through the maxima of the blob.

5.6.3 Number of heads

Considering the fact that some of the sought objects are people, either alone or in group, the idea of detecting people is also considered, following approaches already discussed in [cheng00] a [haritaoglu00]. In both works, vertical projections are used to detect pedestrians and people standing in public areas. Other approaches can be found in the literature, as the work discussed in [viola03] to detect pedestrians, but the aim of simplicity induced us to try new methods.

In the case of a single person, figure 5.7 shows that an approximate vertical symmetry axis can be computed as the axis that crosses the body through its maximum. Moreover, this axis can be related with the position of the head, if the person is standing. Following this approach, we use the superior part of the blob together with the vertical projection to detect the vertical symmetry axes because, when people walks, they create empty spaces between their legs that may mislead the search of maxima.

For a given blob, the algorithm computes the vertical projection and the maxima and minima of the superior silhouette. It ensures that each selected maximum is followed by a minimum and vice versa. Figure 5.8 shows the superior silhouette of a given blob and its maxima and minima. It can be seen that the superior silhouette is computed only in the top third of the blob, which, approximately, corresponds to the height where shoulders are expected to be. In order to avoid errors such as local maxima or minima because of noise, minima are not allowed to appear in a higher position that any maximum and vice versa. In addition, each axis is associated with a value,

$$porVert_i = \frac{hX(i)}{BlobHeight} \tag{5.11}$$

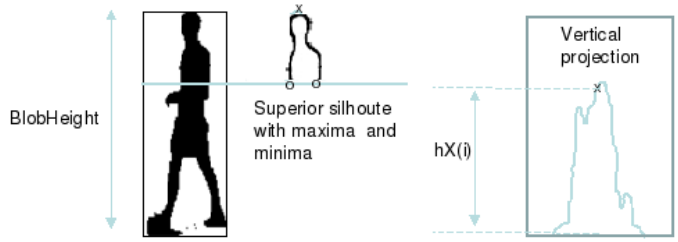


Figure 5.8: Vertical projection and superior part of the silhouette, showing some local maxima and minima. On the right, the vertical projection of the blob. Points labelled with an x are the same, the difference in the shape of each figure comes from the fact that the one in the middle of the figure is the superior silhouette and the figure on the right, the vertical projection of the blob.

being $hX(i)$ the value of the vertical projection in the coordinate of the axis and $BlobHeight$ the height of the blob.

The value $porVert_i$ measures the relative amount of blob located under each maximum, this way, maxima result of raised hands are removed if the value is conveniently thresholded, as shown in the figure 5.9. It can be seen in the figure that the raised fist of the person is detected as a maximum and neglected later due to the its low associated $hX(i)$. Any maximum m which verifies that $porVert_m > 0.5$, is considered a valid maximum.

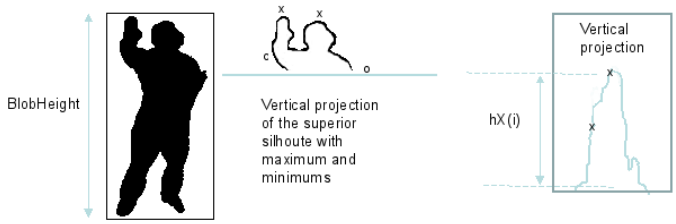


Figure 5.9: For a person with a raised hand the algorithm detects two possible maxima in the superior silhouette labelled with the symbol "x" in the figure, but when the vertical projection is computed and the coefficients $portVert_i$ are computed for each possible maxima, the maximum located in the hand is discarded and only one maximum is considered as valid.

As mentioned previously, the position of each valid maxima can be considered as the approximate location of a head, thus, it is easy to infer how many heads are visible in the blob by counting how many valid maxima are discovered by the algorithm.

5.7 Object tracking

After motion detection, the next step is to track objects, that is being able to know which objects of previous scene have moved and where to. Tracking over time typically involves matching objects in consecutive frames using features such as points, lines or blobs. In our system, each segmented blob is considered an object to track.

We have implemented a tracking system based on the bounding box region. For each blob x obtained in the motion segmentation process, its bounding box R_t^x is calculated. We assume that the movement of the object in two successive frames is such that the bounding boxes surrounding it in two successive frames will overlap each other. Tracking is performed by checking which bounding boxes in the current frame overlap with those in the previous one. This way, we can detect easily and with a good confidence degree which blob has moved and where to, without any further test. This method has been found to be effective in other approaches and does not require the prediction of the blob's position [fuentes03].

The function used to detect whether two different regions R_1 and R_2 overlap can be defined as,

$$O(R_1, R_2) = \begin{cases} 1, \ if \ R_1 \bigcap R_2 \neq \emptyset \\ 0, \ otherwise \end{cases} \tag{5.12}$$

where $R_1 \bigcap R_2$ represents the bit-wise logical and operation.
The following situations are considered:

- Blob stayed in the scene: when a bounding box R_i^x in the current scene overlaps only one bounding box of previous scene R_{i-1}^y, then it is assumed that R_i^x and R_{i-1}^y contain the same object. Figure 5.10 illustrates this case, expressed mathematically as,

$$\exists R_{i-1}^y \ , O(R_i^x, R_{i-1}^y) = 1 \wedge O(R_i^z, R_{i-1}^y) = 0 \ , \forall z \neq y \rightarrow \\ R_i^x \ stayed \ in \ the \ scene \tag{5.13}$$

Figure 5.10: Object stays in the scene, though it moves, from frame to frame there is still the possibility of finding a temporal overlapping between consecutive frames.

- New blob in the scene: When a Bbox R_i^x in the current scene does not overlap with any other R_{i-1}^y in the previous scene, R_i^x is supposed to be new blob. Figure 5.11 illustrates this case. Mathematically, this case can be expressed as follows,

$$\forall R_{i-1}^y, \ O(R_i^x, R_{i-1}^y) = 0 \rightarrow R_i^x \ is \ a \ new \ blob \tag{5.14}$$

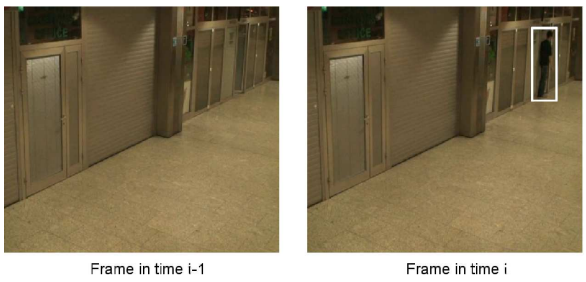


Figure 5.11: An object enters the scene. Note that the frame in time $i-1$ shows an empty scene, in the following frame, there is a person that entered the scene.

- A blob leaves the scene: When a blob of the previous scene R_{i-1}^{y} does not overlap any other R_i^x in the current scene, it is supposed that R_{i-1}^{y} left the scene. This case is the opposite to the previous one and figure 5.12 shows an example. Mathematically, this case can be expressed as,

$$\forall R_i^x,\ O(R_{i-1}^{y}, R_i^x) = 0 \rightarrow R_{i-1}^{y}\ left\ the\ scene \tag{5.15}$$

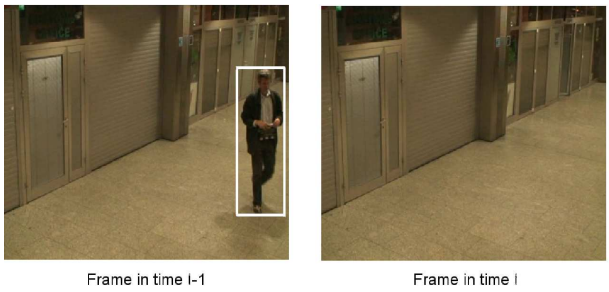


Figure 5.12: An object leaves the scene. Note that in the frame in time $i-1$ there is a person in the scene and in the following frame it is not there any more.

- Two or more blobs join: When two or more different bounding boxes $R_{t-1}^{y_0}$, $R_{t-1}^{y_1}$, ... , $R_{t-1}^{y_n},\ n \geq 1$ segmented in frame $F(t-1)$ overlapped a single blob R_t^x segmented in the frame $F(t)$, we say that they have joined. Joints happen when people are picking a suitcase up or people are crossing or joining. Figure 5.13 shows an example with two people that move close one to each other and finally, collapse into one only object.

$$\exists \{R_{i-1}^{y_j}\}_{j=0}^{n} \ /\ \prod_{j=0}^{n} O(R_i^x, R_{i-1}^{y_j}) \neq 0 \rightarrow \{R_{i-1}^{y_j}\}_{j=0}^{n}\ joined\ in\ R_i^x \tag{5.16}$$

Figure 5.13: Two people which are moving close one to each other in frame $F(i-1)$ finally collapse into one in frame $F(i)$.

- Two or more blobs split: When a single Bbox R_{i-1}^{y} in frame $F(t-1)$ overlapped with two or more different blobs $R_i^{x_0}$, $R_i^{x_1}$ ... $R_i^{x_n}$, $n \geq 1$ in current scene, we say that they have split. Splits happen when someone leaves a suitcase or a group of people unravels. Figure 5.14 illustrates this case with two objects. This situation can be expressed as,

$$\exists\{R_i^{x_j}\}_{j=0}^{n} \,/\, \prod_{j=0}^{n} O(R_i^{x_j}, R_{i-1}^{y}) \neq \emptyset \rightarrow \{R_i^{x_j}\}_{j=0}^{n} \; splited \; from \; R_{i-1}^{y} \tag{5.17}$$

Figure 5.14: Two people which were considered as one move apart and reveal themselves as two different objects.

The joint/split of blobs corresponds to situations such as people crossing, people interchanging material or people leaving or picking up something, for instance.

A Kalman filter [kalman60] is also used in order to perform object disambiguation when objects cross in the scene. The prediction of new position is compared to object's position in the case of two or more blobs splitting.

Once blobs are located in the scene, they are given a unique label or keep the one they had previously. This label is used by higher levels to identify them in each camera. Blobs are called from now then objects. In the next step, for each object a membership probability for each class are calculated in order to ease the final classification done by higher processing levels.

Figure 5.15: Some of the blobs included in the dataset used in the experiments. There are samples of the three considered classes, *person*, *group of people* and *luggage* represented in the figure. These blobs were manually labelled after automatic segmentation.

5.8 Blob dataset

The database used in the experiments was designed to contain images of the objects which were expected to be found in the airport. The data consist of blobs extracted from several videos of different lengths recorded in the hall of our university in different days with different light conditions. Each blob is associated with a bitmap crop of its bounding box from the real video in RGB space and in grey levels. Together with each blob, we kept the coordinates of its bounding box in the image and the frame in which it was captured, so it was easy locating in the sequence each image. Each object was labelled manually by us as belonging to one of the classes; *person*, *group of people*, and *luggage*. Blobs were classified according to the following rules:

- Blobs are classified according to the object it represents, as *person*, *group of people*, and *luggage*.
- Every blob representing a person with or without luggage is labelled as *person*.
- Only objects with at least $2/3$ of their upper part of the figure were included in the database.
- A blob representing more than two people is considered to be a *group of people*, provided that at least $2/3$ of the people are visible.

We kept a database of 2369 images extracted from the sequences, corresponding to 1371 images of class *single people*, 367 of class *group of people* and 631 images of class *luggage*.

In order to apply both algorithms, we kept a background model either in RGB and grey tone. The background model was updated, for each coordinate, following the running Gaussian average approach discussed in section 3.6 in chapter 3. RGB and grey tone distances were computed using euclidean distances and a single threshold to decide how to classify pixels as foreground or background.

	Person	*Group of people*	*Luggage*	*Total*
Blob dataset	1371	367	631	2369
Training data. 80%	1097	294	504	1895
Test data. 20%	274	73	127	474

Table 5.1: Distribution of objects per class and approximate size of the training and test sets used in the experiments.

Figure 5.15 shows some sample blobs from the database. Table 5.1 shows the distribution of blobs per class in the dataset, it can be seen that the size of class person is larger than the others, this reflects the proportion of the objects of different classes in the sequences.

5.8.1 Evaluation of features with blob dataset

In order to evaluate which of the features proposed in section 5.6 is the best that can be used to classify objects, a set of experiments was performed. Several experiments were made considering only the objects in the blob database, the aim of these experiments was:

- test the suitability of the dataset.
- check that the chosen features were, at least, consistent.
- find out the best configuration for the number of features to be used.

The evaluation criteria for the features are:

- the success in the classification of objects and the interclass confusion.
- the time needed to compute each set of features.
- the time needed to classify objects.

The experiments were performed with the features described in section 5.6 using a k-NN classifier in all cases, therefore, the most suitable value of k is sought for. Experiments may be divided into off- and on-line, as some were performed with the database and others with real sequences. With the geometric features, the experiments were aimed to test the validity of the considered features to classify blobs. Experiments with foreground pixel density features are twofold. A first goal was obtaining a measure of how many cells would be necessary to classify correctly the blobs in the database. On the other hand, once the amount of cells was fixed, it was also necessary to obtain the optimal amount of neighbours for the k-NN classifier used. In both cases, the experiments were performed with the database of blobs and also using the shadow removal algorithms. They were performed by building a training set using the 80% of the total amount of samples randomly chosen and a test set with the remaining 20% of samples. For each value of k, the experiment was performed 100, recomputing each time the training and test set in order to achieve statistical independency.

Grid density features were extended in order to allow different granularity in specific parts of the blob. In this case, the experiments performed were also aimed to find the optimal configuration of cells with the blobs in the database, without considering shadow removal algorithms.

Finally, experiments were performed with the head detection algorithm described in section 5.6.3 together with the features inversed dispersedness and extent. Results of this combination are better than the previous in time consumption and memory requierements.

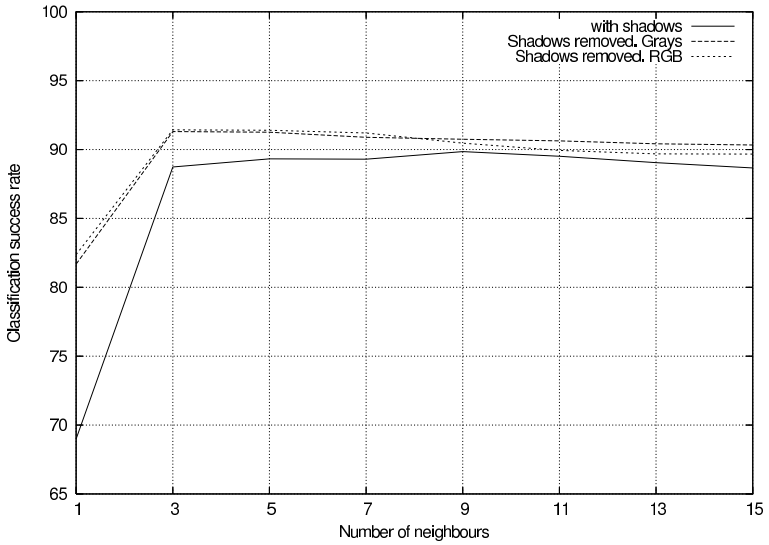


Figure 5.16: Classification rate of the blob dataset with and without shadows using geometric features for different values of k.

5.8.2 Experiments with geometric features

Each blob is represented with a vector of 3 features: dispersedness, extent and solidity of the blob. As said before, classification is performed by using a k-NN classifier. The training set was built choosing randomly samples of the database up to a size equal to 80% of the total amount of samples, the remaining 20% was kept for testing. For each value of k, each experiment was performed 100, recomputing each time the training and test set in order to achieve statistical independency.

In figure 5.16 we show the global results of object classification for different values of k and for each feature set. Each value in the plot corresponds to the average of 100 experiments performed with a given value of k. Results are computed using geometric features with shadows left and with shadows removed. It can be seen for geometric features that for $k = 1$ the classification rate is 68%, for higher values of k, the classification rate rises and stays between 87% and 90%.

	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$	$k=13$	$k=15$
Person	36.25%	48.39%	48.39%	50.73%	51.69%	54.51%	54.51%	54.51%
Luggagge	0.22%	0.09%	0.02%	0.17%	0.12%	0.08%	0.09%	0.08%

Table 5.2: Comparison of the confusion rates between class *groups of people* and classes *person* and *luggage* classes using geometrical features.

From figure 5.16 can be also seen removing shadows, either using grey tone images or RGB images, has not a deep impact in classification performance, about 5% in the

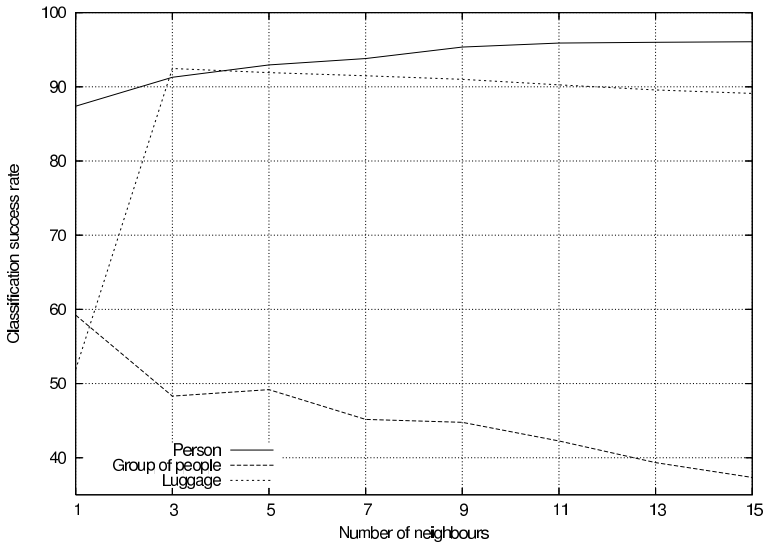


Figure 5.17: Classification success rates with k-NN using geometric features, each line correspond to results for one class. In this case, shadows were not considered. For classes *person* and *luggage* the chosen features perform quite well but, on the other side, It can be seen that for *group of people* class shows low values.

best case using the proposed geometric features. It is considered that the effort devoted to remove shadows is not worth, considering the time penalty introduced by the shadow removal algorithms.

Though global results seem promising, it also necessary to test what the behaviour for each single class is. It can be seen in figure 5.17 that classes *person* and *luggage* have a good classification success rate for both feature sets, but the *group of people* class shows low values.

Table 5.2 shows the confusion between classes *person* and *luggage* with class *group of people* for different values of k. It can be seen that class *person* has a big rate of confusion with *group of people*. This could justify the low success rate shown for class *group of people* in figure 5.17.

5.8.3 Experiments with foreground pixel density features

Some experiments were performed aimed to establish the optimal number of cells in which a blob should be divided before performing experiments with this feature. This experiments were conducted by dividing the bounding box of a blob into an identical number of rows and columns, different sets of 2×2, 4×4, .., 10×10 cells and classifying objects using a k-NN classifier, as in previous section, experiments for each grid configuration and value of k are represented in the plot with the average of 100 repetitions of the experiment.

In figure 5.18 results for different values of k and different sizes of square grids are shown. Though the 7×7 yields the best results, we considered that the 4×4 grid was the one that best fitted our needs, because it is the smallest grid from which classification rate starts stabilizing having the same behaviour for all the values of k used in the experiments.

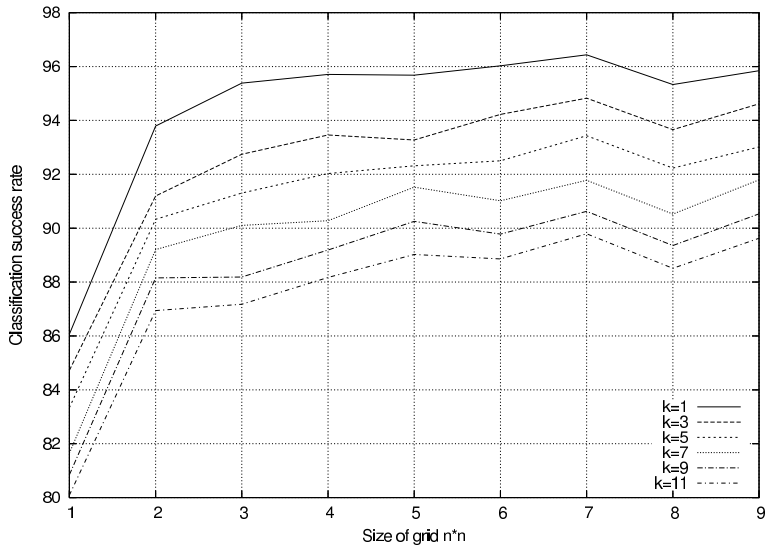


Figure 5.18: Results of classification experiments with k-NN, with different values of k and different cell configurations. The value n is the number of cells in which the blob was divided, the same vertically and horizontally. Thus the grid considered range from 1×1, 2×2 up to 9×9 cells. As expected, the best values are obtained for $k = 1$ with a difference of only 6% between the best values and the worst, corresponding to $k = 6$.

In figure 5.19, we show results for foreground pixel density with shadows and with shadows removed, using a grid size of 4×4 with different values of k. As it happened with geometrical features, in this case the low increase in classification rate does not justify the time penalty introduced by the shadow removal algorithms.

Comparing figure 5.19 to figure 5.16, it can be noted that foreground pixel density behaves completely different to geometric features and show a good success rate ($95\% - 92\%$) for values of k between 1 and 5 and then decreases as k grows.

Another issue that must be compared, is how the features perform for each of the considered classes as done in previous section. Figure 5.20 shows the performance of foreground pixel density features for each individual class. Class *person* and *luggage* have a good classification success rate for both feature sets, while class *group of people* behaves better than with geometric features.

Table 5.3 shows the confusion between classes *person* and *luggage* with class *group of people* for different values of k. It can be seen that class *person* has a big rate of confusion with *group of people*.

Summarizing the information in tables 5.3 and 5.2, we can conclude that the highest confusion occurs between classes *person* and *group of people* and increases as k grows. Confusion between *group of people* and *luggage* is very low. In general, results point out, in any case, that both feature sets are valid for classifying classes *person* and *luggage* with a good degree of accuracy. With geometric features we obtain a high level of stability and with foreground pixel density a slightly higher percentage of success.

It may be seen in figures 5.16 and 5.19, that by removing shadows we increase the classification success rate; although the gain does not justify the computational effort required. For instance, for a movie in which the complete segmentation-tracking loop

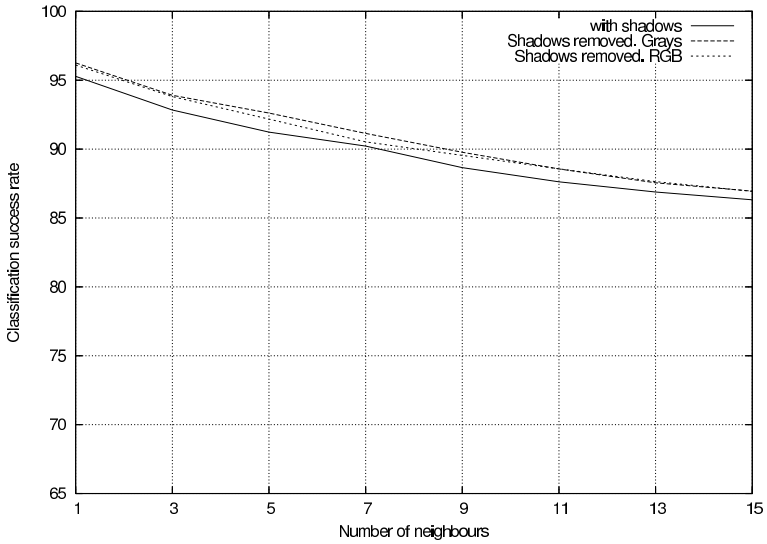


Figure 5.19: Classification rate using k-NN of blobs with, and without shadows, using the matrix of foreground pixels density. Grid size was 4×4. It can be seen that using the shadow removal algorithms does not improve significantly the performance of the features.

	$k=1$	$k=3$	$k=5$	$k=7$	$k=9$	$k=11$	$k=13$	$k=15$
Person	0.48%	0.89%	11.06%	13.79%	15.76%	16.33%	15.93%	16.11%
Luggage	0.02%	0.02%	0.01%	0.04%	0.14%	0.27%	0.56%	0.89%

Table 5.3: Comparison of the confusion rates between class *groups of people* and classes *person* and *luggage* classes using foreground pixel grid density features.

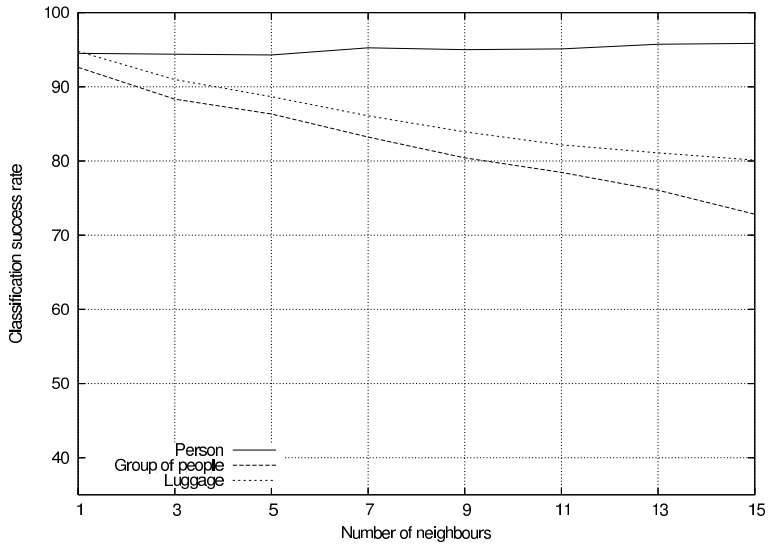


Figure 5.20: Classification success rates with k-NN using foreground pixels density features and grid size 4×4. Each line corresponds to results for one class. Results are obtained with k ranging in $[1, 15]$. It can be seen the the amount of objects of class *person* correctly classified is about 95%, in this case, the class with the worst classification rate is *group of people*.

takes up to 0.91 seconds, and there are an average of 1.8 objects per frame, the shadow removal algorithm in grey levels takes up to 14.03 seconds on average per frame; and the shadow removal algorithm in RGB space takes up to 30.10 seconds on average per frame. From these results, removing shadows in both cases is important if we want to achieve optimal accuracy, but, as plots and time results show, maybe the gain will not be worth the effort.

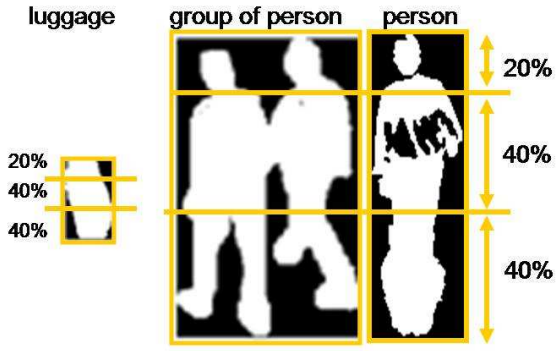


Figure 5.21: Sample division of the silhouette of a person, a group of people and a suitcase in the three unequal regions, head, body and legs.

5.8.4 Foreground density with different granularity

A step forward with foreground density features is giving more importance to some parts of the object over others. This is due to the fact that some parts of the blob may be more significant when distinguishing it from blobs of other classes. This justifies that a bigger amount of cells can be used to represent a concrete area of the blob, having this way a more accurate representation of this area. For instance, the central part of the blob of a person will be quite similar to the central part of the blob of a suitcase, however, the lowest parts will have difference occupancy patterns.

Having different granularities can be achieved by dividing each object in three unequal regions, which we will call from now on, *head*, *body* and *legs*, being the head region the top of the object, legs region will correspond to the bottom of the object and body region will be the rest. Figure 5.21 shows an example of the silhouette of blobs representing a person, a group of people and a suitcase divided into three different regions.

Figure 5.22 shows some blobs with different granularity of cells for regions *head*, *body* and *legs*. Stripped lines separate the three different regions and straight lines delimit cells in regions. It is clear that head region may be more useful for distinguishing instances, for example, of classes *person* and *luggage*.

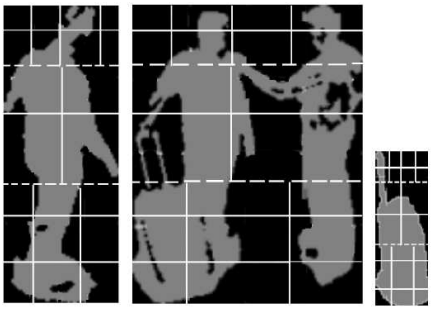


Figure 5.22: Samples of person, group of people, and luggage with different divisions in the 3 regions defined as head (granularity 4×2), body (granularity 2×2) and legs (granularity 3×3).

With this schema, we divide each region into a grid of $n_r \times m_r$ cells, where r stands for the region (head, body or legs), being the grid size of each region independent of the rest. We then perform the same calculations for each cell as in expression 5.10.

Off-line experiments were made in order to test the validity of having more granularity in some parts of the blob over others. In separated tests, we decided that the top 20% blob would correspond to the head region, the bottom 40% blob would correspond to legs, and the rest to the body. Then we made experiments varying the values of n_{head}, m_{head}, n_{body}, m_{body}, n_{legs} and m_{legs} in the range $1..20$, these values represent the number of cells in which each region is divided into.

As in previous experiments, a k-NN classifier was trained using 80% of the database and the test set was composed of the other 20%. The experiments were repeated 100 times to ensure statistical independence of the selected samples. In table 5.4 we show confusion tables for different values of n_r, m_r, and $k = 5$, these results shown are the best obtained with these experiments. Results point out that not always a bigger granularity in any of the regions yields better results than having an equal distribution

of cells.

Features	93 *features*			101 *features*			27 *features*		
Cells	$3 \times 4, 7 \times 8, 5 \times 5$			$5 \times 2, 8 \times 7, 5 \times 7$			$3 \times 1, 4 \times 3, 4 \times 3$		
Class	*Person*	*Group*	*Luggage*	*Person*	*Group*	*Luggage*	*Person*	*Group*	*Luggage*
Person	98.1%	4.5%	0%	98.5%	3.3%	0%	98.2%	7.3%	1.0%
Group	1.9%	95.5%	0%	1.5%	95.2%	0%	1.8%	92.7%	0%
Luggage	0%	0%	100%	0%	1.5%	100%	0%	0%	99.0%

Table 5.4: Confusion tables for different division granularities with $k = 5$. The diagonal of each table represents the rate of successful classifications. The number of features that the different granularities handle is specified by the row *Features*, row *Cells* specifies how they are organized. For instance, the first classifier computes 93 different granularities.

The main issue we took into account when selecting the best arrangement, was the performance with class *group of people*. As we learned in the previous work, objects belonging to this class are easily confused with instances of class *person*, and it is easy to see that, depending on how people are arranged inside the group (for instance, with occlusions), a group may be confused with a person. The arrangement of cells ($n_{head} = 3, m_{head} = 4, n_{body} = 7, m_{body} = 8, n_{legs} = 5, m_{legs} = 5$) was the chosen one, with a total amount of 93 features. Results shown in table 5.4 improves slightly the performance with the class *group of people* over the performance of the classifier with the arrangement $3 \times 1, 4 \times 3, 4 \times 3$.

Confusion between *luggage* and *group of people* is nearly reduced to zero with this new schema and confusion of classes *person* and *group of people* is also very low; improving results obtained without forcing different granularities in different areas of the object.

The performance of the foreground pixel density features to separate objects of classes *person* and *group of people* has the disadvantage of using a big number of features, which can be expensive to compute if a big number of blobs are in the scene.

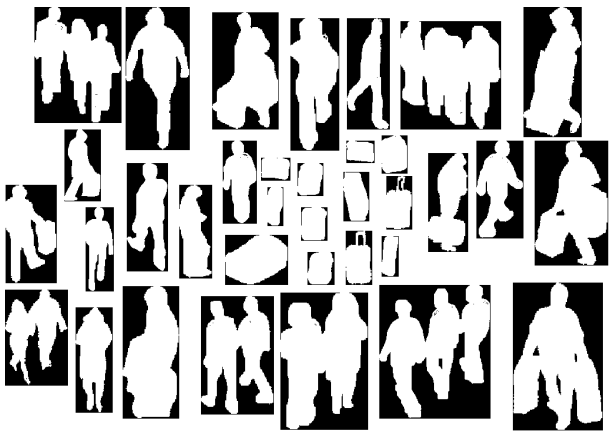


Figure 5.23: Some of the blobs included in the new database used in the experiments. There are samples of the three considered classes, *person*, *group of people* and *luggage* represented in the figure. These blobs were manually segmented before inserting them in the database.

5.8.5 Experiments with head detection algorithm

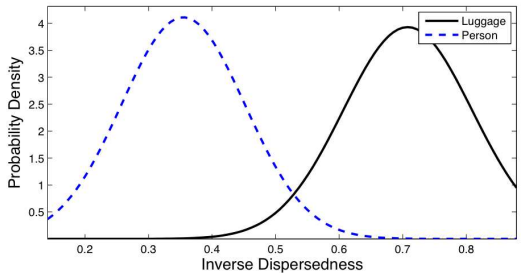


Figure 5.24: Statistical distributions of feature inverse dispersedness for classes *luggage* and *person* for blobs in the database.

Either with geometric features or with foreground density features, confusion between classes *person* and *group of people* is reduced but at the expense of computing a big amount of features, with its associated time penalty. Previous experiments show that using shadow removal algorithms does not improve significantly the classification rate of the features considered.

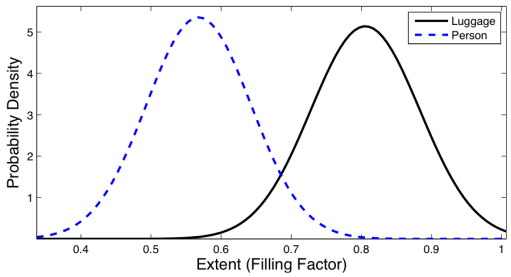


Figure 5.25: Statistical distributions of feature extent for classes *luggage* and *person* for blobs in the database..

A new database was built segmenting by hand the objects belonging to the considered classes. Figure 5.23 shows some sample objects of this database. The aim of this database was providing us with more accurate blobs on which new features could be tested. This database contained 631 blobs corresponding to class *person*, 101 blobs corresponding to class *group of people* and 991 blobs of class *luggage* for a total amount of 1723 objects. Table 5.5 shows the distribution of objects according to the represented classes.

An alternative approach to classification of the three classes at a time, is taking advantage of individual and exclusive features that may help classifying one type of object. For instance, classes *group of people* and *person* are easily distinguished by the number of heads of the blobs.

The features inverse dispersedness and extent were computed all the blobs in the new dataset, figures 5.24 shows the probability density function of objects depending on their values of inverse dispersedness. For instance, if a blob yields an extent value of 0.6 it is quite more probable that the blob belongs to class *person* than to class

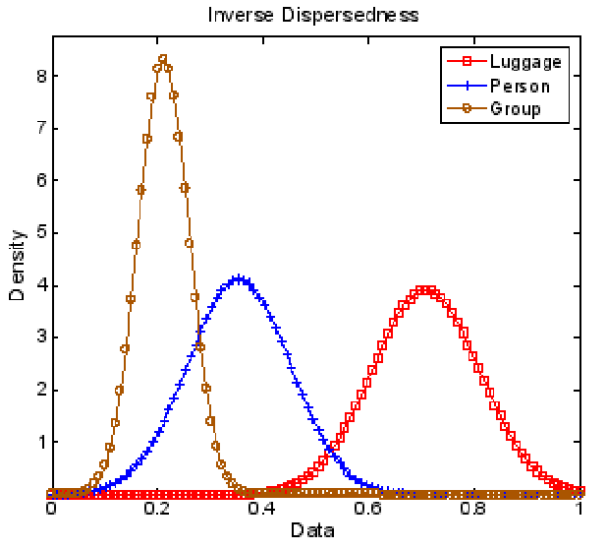


Figure 5.26: Statistical distributions of feature inverse dispersedness for classes *group of people*, *luggage* and *person* for blobs in the database.

luggage. On the other side, figure 5.25 shows the probability density function of objects depending on their values of extent. Figures 5.26 and 5.27 show the probability density of objects for all classes, it can be seen that for both features, class *group of people* overlaps areas belonging either to class *person* or class *luggage*.

	Person	*Group of people*	*Luggage*	*Total*
Blob dataset	631	101	991	1723
Training data. 80%	504	80	792	1378
Test data. 20%	127	21	199	345

Table 5.5: Distribution of objects per class and approximate size of the training and test sets used in the experiments.

The information obtained from the four plots discourages the use of these features to distinguish the three classes, but induce to think that a classifier can be built to separate classes *person* and *luggage* using the features extent and inverse dispersedness. Using such a classifier has the advantage that only contains two features, being very fast to compute. Moreover, the feature vectors of inverse dispersedness and extent were reduced using the multiedit-condensing [hart68], [devijver82] technique. The final database contained 11 blobs of class *person* and 7 blobs of class *luggage*. Blobs of class *group of people* were not considered any longer in this database.

On the other hand, in order to separate classes *person* and *group of people*, the head detection algorithm proposed in section 5.6.3 is used. The algorithm described in section 5.6.3 for counting the number of heads in a blob was tested with a set of experiments. The goal of these experiments was knowing how accurate the algorithm is and in which cases it could be expected to fail. Before the experiments, all the blobs in the database corresponding to the class *group of people* were labelled with the number of heads that they contained. Then the results obtained with the experiment were checked against the real amount of heads.

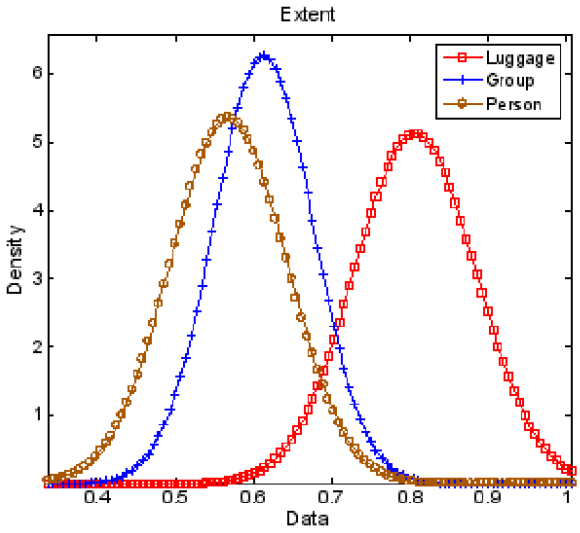


Figure 5.27: Statistical distributions of feature extent for classes *group of people*, *luggage* and *person* for blobs in the database..

The amount of heads expected to be detected in the groups ranged between 2 and 4, having 60 blobs of 2 members, 30 blobs of 3 members and 11 blobs of 4 elements. The error rate of the head detection algorithm is shown in table 5.6. It can be seen that for blobs of 2 members the success is complete. As the number of members increase, the accuracy decreases. As figure 5.28 shows, occlusions and excessive separation of head heights inside the blobs are the main sources of inaccuracy. In both cases, the algorithm detects one head less than the real number.The gap between the bottom of the bounding box and the legs of the person whose head is missed, is responsible of the failure, producing a number of maxima lower than expected. This effect occurs mainly in big groups (groups containing more than 6 people) making it difficult the accurate computation of the number of heads with the information of just one camera.

(a) (b)

Figure 5.28: Some blobs representing groups, it is remarkable that separation of head height inside the blob (blob on the left), and excessive overlapping (blob on the right), avoid the algorithm to detect exactly the amount of people in the blob. In both cases, however, the error was just a person left.

Number of heads found	*Group of 2 people*	*Group of 3 people*	*Group of 4 people*
Total amount	60	30	11
1 head	0%	0%	0%
2 head	100%	40%	9.1%
3 head	0%	60%	63.6%
4 head	0%	0%	27.3%

Table 5.6: Percentage of successfully detection of heads for blobs representing groups of people, depending on the amount of heads expected. Groups in the database vary from 2 to 4 members.

They were performed by building a training set using the 80% of the total amount of samples randomly chosen and a test set with the remaining 20% of samples. For each value of k, the experiment was performed 100, recomputing each time the training and test set in order to achieve statistical independence.

Then, with the aim of classifying objects using the features number of heads, extent and inverse dispersedness a classification algorithm is designed joining the head detecting algorithm to the reduced training set of geometric features, see algorithm 3. Performance of this algorithm with the objects of the database can be found in table 5.7. This performance is measured by dividing the database into two disjoint sets, the training set built with the 80% of the total amount of samples randomly chosen and a test set with the remaining 20% of samples. For each value of k, the experiment was performed 100, recomputing each time the training and test set in order to achieve statistical independence. Results in this case are more satisfactory either for the database as for real-videos shown in table 5.8.

Algorithm 3 A two step decision tree combining the feature number of heads in the root to distinguish objects of class *group of people* from the rest and, at a second level, a k-NN classifier trained with features extent and inverse dispersedness to classify objects of class *person* and *luggage*.

1: **if** headNumber(blob) >= 2 **then**
2: $blobClass \leftarrow GROUP$
3: **else**
4: $blobClass \leftarrow k$-NN(blob.inverseDispersedness(), blob.extent())
5: **end if**

k-NN				k-NN + number of heads			
	Person	*Group*	*Luggage*		*Person*	*Group*	*Luggage*
Person	89.5%	59.2%	16.3%	*Person*	97.4%	4.0%	6.3%
Group	2.5%	22.9%	0.01%	*Group*	0%	96.0%	0%
Luggage	7.9%	17.9%	83.5%	*Luggage*	2.6%	0%	93.7%

Table 5.7: Columns under k-NN classifier were classified using only a k-NN trained with the features extent and inverse dispersedness and $k = 1$. On the right, the results of classifying the new database using also the feature number of heads as described in the algorithm 3. It is quite evident the improvement in the classification of the three classes, specially regarding class *group of people*, if the k-NN classifier together with the number of heads feature is used.

Performance of algorithm 3 with the objects of the database shown in 5.23 can be found in table 5.7. After visual analysis of the results we must emphasize that most

of the errors found in the classification phase are caused by poor segmentations that distort the shape of blobs.

5.9 Experiments with sequences

Also experiments with sequences were performed using the algorithm 3. Working data are videos recorded in different days and hours in an european airport. These videos show different areas of the airport taken from one or two vision angles, partially covering the same area and do usually have between 900 and 8000 frames. An effort was done to extract background models for each scene in order to process it simulating real-time surveillance. In section 5.3, more details about how these models were created may be found. The sequences have been chosen trying to capture different real life operation situations. Figure 5.29 shows one of the frames processed in these experiments.

Figure 5.29: Image obtained from a client application showing information provided by low level video processing services. Each detected object has a label with the object identifier number and the assigned membership probability. Each BoundingBox is drawn with a colour that represents the class (green for class *luggage*, red for *group* and blue for *person*) and the detected heads have been marked with a cross.

Experiments are evaluated at the end of the process, that is, only if an object is successfully segmented and classified, it is considered to be a success. In any other case, it is accounted as a failure.

Also, measuring the validity or not of the thresholds used in the segmentation is very difficult due to the variety of scenarios that may be faced by the system. These thresholds should be set manually for each individual situation.

Table 5.8 shows the classification results obtained with the algorithm 3. Experiments were performed using two different values for k in the k-NN classifier. Results seem to be slightly better for class *luggage* if $k = 1$ is used.

Taking into account the great variety of scenes and working conditions under which have been made these tests the results of the low-level classification can be considered as reasonably good.

$K=1$				$K=3$			
	Person	*Group*	*Luggage*		*Person*	*Group*	*Luggage*
Person	86.8%	15.2%	6.8%	*Person*	86.8%	15.5%	8.2%
Group	13.0%	84.5%	0.0%	*Group*	13.0%	84.5%	0.0%
Luggage	0.2%	0.0%	93.2%	*Luggage*	0.2%	0.0%	91.8%

Table 5.8: Results of experiments for test videos using the decision tree described in algorithm 3. Results are given relative to the total amount of objects correctly segmented and expected to be found of each class.

5.10 Conclusions

A surveillance system was developed in this chapter based on frame differencing and background subtraction with a per-frame classification of objects. The output is filtered to remove noisy blobs. Blobs in the scene are tracked over frames by the application. In this case the classification process is more difficult than the one discussed in chapter 4, due to the fact that objects move and may vary their shape, making it very difficult to train a classifier based directly on shapes. The main contribution of the chapter is the features used to classify the blobs [rosell08], [benet10].

In order to classify objects in the scene, different feature sets were tested: geometric features, foreground pixel density and a head detection algorithm. Experiments were performed in order to quantify if reducing shadows in blobs could improve classification results, but results made us conclude that using shadow removal algorithms does not improve significantly the classification rate of the features considered.

Geometric features showed a poor global performance due to the fact that the *group of people* class is often misclassified as *person*, although *luggage* and *person* classes are classified satisfactorily. Foreground pixel density had a better performance with less interclass confusion, although performance decreases when k increases, leading to a poor performance in both *groups of people* and *luggage* classes. Geometric features seem to be more stable than the foreground pixel density.

New experiments were considered in order to improve results for foreground pixel density feature. In these experiments, different granularities were assigned to three different parts of the blob. In this case, contrary to the what was expected, results point out that not always a bigger granularity in any of the regions yields better results than having a equal distribution of cells. In fact, the distributions with best results had the disadvantage that their performance separating objects of classes *person* and *group of people* used a big number of features, which can be expensive to compute if a big number of blobs are in the scene.

Finally, a solution based on joining a head detection algorithm together with a k-NN classifier trained with the features extent and inverse dispersedness performs better than the rest. In this case, the features were sought to find the most notable differences between each pair of classes. For instance, the head detection algorithm provides the system with a clean and reliable method to separate classes *person* and *group of people*. On the other side, inverse dispersedness and extent discriminate well objects of classes *person* and *luggage*. By using the algorithm 3 the rate of object recognition in, both off-line and on-line experiments, is quite high. Objects of class *person* and *luggage* from the database obtain, respectively, a classification rate of 97.4% and 93.7%, for objects of class *group of people* the classification rate is about 96.0%. In on-line experiments rates are also equally promising in the tested sequences, with a minimum rate of 84.5% of successfully classified objects of class *group of people*.

Chapter 6

Background modelling and object detection

In this chapter we develop two algorithms designed to model scenarios and detect objects, with the aim of being able to give a good performance in demanding scenarios. BAC algorithm [rosell08a] creates or restores a background model based on the behaviour of pixels in successive frames and, at the same time, performs a segmentation of objects in the scene. BAC has the novelty that yields a confidence value for the obtained background. BAC is extended with two versions developed with the aim of permitting colour processing [rosell09] or multi-modal support (MBAC [rosell10]). Also, a novel background subtraction method is discussed, called fuzzy background subtraction. This later approach eludes the use of fixed or probabilistic thresholds usually found in the traditional background subtraction methods and use of per pixel foreground and background probabilities which can be useful for further processes. Finally, we propose a method that by means of BAC computes automatically the input parameters for a background modelling algorithm [rosell10b]. The performance of the proposed algorithms are evaluated and compared to well-known techniques.

6.1 Introduction

In chapter 3, section 3.3 offers a thorough description of different background modelling algorithms available in the literature. Although the methods mentioned there obtain good results in the tested scenarios, in general, all of them expect working in scenarios with low or null activity to build their first background model. One of aspects which we miss in these approaches is that there is no measure of when a suitable background is achieved.

Most of the methods proposed in the literature for background subtraction operate mainly in two steps. The first step consists in subtracting an input frame $F(i)$ from a background model $B(i-1)$ and, the following step consists in obtaining $B(i)$, the background model in time i. Usually, both steps expect several parameters to be tuned. For instance, the subtraction method needs a criterion to decide when two pixels are considered equal or not. This criterion is usually a comparison between the difference and a given threshold. In the case of the update step, decisions about how often should the model be updated or how often should previous background values be forgotten are also parameters to be taken into account.

Besides the different approaches to background modelling, another issue related to this technique is the detection of corrupt models, that is, models which are not useful any more for surveillance purposes. A corrupt model yields false segmentations and it is important being able to detect and correct them. Few papers in the literature address this issue, as far as authors of this chapter are concerned. In the literature it is generally assumed that changes in the background will occur smoothly and abrupt changes are not considered. In [toyama99], authors propose maintaining a database of models. In the case the background model is considered to be corrupt, a search in the database should be enough to find the most suitable model.

6.1.1 Goals and constraints

In this chapter we propose techniques aimed to answering the questions mentioned above:

- develop techniques that permit obtain a good background model no matter if there is activity in the frames used to build it and without previous assumptions about the scene
- design a measure of quality of the background models. This measure would indicate how confident or reliable a background model and could used to determine when a background model is useless. A background model which does not yield a proper segmentation of the input frame.

The main constraints that the algorithms have to face is,

- they must work with minimal human intervention, that is, as autonomously as possible and with a minimal set of parameters
- they should work with minimal resource needs
- they should build a background model without human intervention even in scenes with high activity
- they should detect automatically if the background model is not valid any longer and reconstruct it, avoiding the storage of background models as in [toyama99] as it would be incompatible with the memory constraint

In BAC we study the effect of changing the classical measure of distance for a similarity function, discussed further in section 6.2. Based on this algorithm, two extensions are proposed in sections 6.3 and 6.4. The former is an extension of BAC to allow the use of several models per pixel and the latter an extension of BAC using RGB coordinates to describe each pixel with just one model.

The second, section 6.5 introduces a background subtraction algorithm that computes a global threshold on the fly based on the intuitive idea that similar images should yield, in average, low distances when subtracted. For both algorithms, a measure of confidence, at pixel and model level, is defined and a way to detect corrupt models is also described.

Finally, in section 6.6 the performance of the proposed algorithms are evaluated and compared to well-known techniques. The algorithms introduced in this paper are combined with the classifier discussed in chapter 5. It is also explored the possibility of using BAC as an algorithm to compute properties of the scene and an initial background model that can be further used by other algorithms.

6.2 Background adaptive with confidence (BAC)

In this section, we propose an algorithm that does not use only statistical properties of pixels, but also their behaviour, to build the model. As stated in the introduction, the aim is reconstructing or creating a background model from the scratch, with no previous assumption about the scene activity. Similarity with the background and motion criteria are used to determine how the model must be updated.

Our algorithm considers a sequence $F(0), ..., F(n)$ of consecutive grey scale frames, in which any pixel $p \in F(j)$ must belong either to foreground or to background and builds a background model $B(i)$. In the first frame $F(0)$ it is impossible to classify pixels as background or foreground, as no further information is given. To decide which pixels may be used to update the background model and which not, a new similarity and motion criteria is defined in section 6.2.1.

The quality of the background model is computed according to a pixel-wise value, that indicates in the range $[0, 1]$ how stable the background model in that location is.

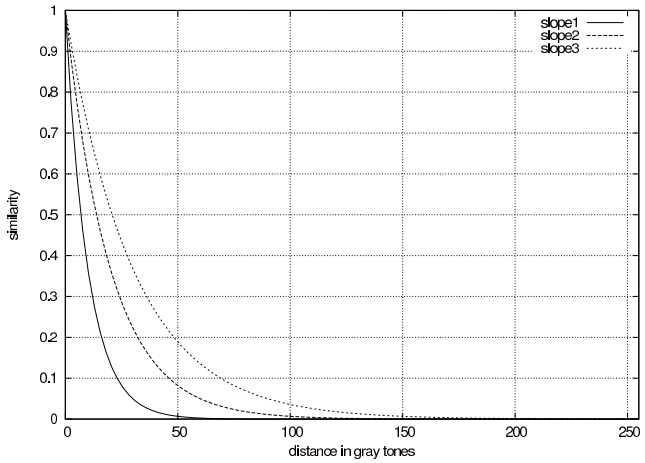


Figure 6.1: Plot of function similarity with different slopes, slope1 corresponds to $\kappa = 10$, slope2 corresponds to $\kappa = 20$ and slope3 to $\kappa = 30$.

6.2.1 Similarity criteria between two grey pixels

Similarity between two pixels is usually tested by comparing the difference of their grey levels with a threshold. We propose to translate into a function the intuitive idea behind "very similar" or "similar" by using a continuous function defined as, b

$$S(p, q) = e^{-\frac{|p-q|}{\kappa}} : \Re - > [0, 1] \tag{6.1}$$

being p and q grey levels of two pixels, κ is a constant determined experimentally. This way, a difference degree and not an absolute difference value is calculated for pixels similarity, having values of $S(p, q)$ close to 1 for similar pixels and close to 0 for very different pixels.

Figure 6.1 shows the plot of $S(p, q)$ for distances in grey tones in the range $[0, 255]$ for different values of κ. It can be seen that increasing the value of κ, increases the amount of distances values which yield similar values for function $S(p, q)$. In our

case, we find it better to use a restrictive value for κ for two reasons, first, because small regions induced by camera noise can easily be removed a posteriori from the segmentation result and second, because it is better having a noisy input that confusing very distant grey tones.

6.2.2 Motion and similarity with the background

By using equation 6.1, it can be measured the similarity of each pixel with the background. Similarity between a pixel $p_{x,y} \in F(i)$ with a background pixel $b_{x,y}$ is then given by $S(p_{x,y}, b_{x,y})$, being $b_{x,y} \in B(i)$ the pixel in the background model.

Also, motion can be computed using equation 6.1. Motion of a pixel can be defined in terms of its dissimilarity with previous values of the pixel, that is, the more similar a pixel is with its previous values in previous frames, the less motion it has. According to the equation 6.1 motion of pixel $p_{x,y} \in F(i)$ with respect to $r_{x,y} \in F(i-1)$ could be defined as,

$$1 - S(p_{x,y}, r_{x,y}) \tag{6.2}$$

This equation is equivalent to frame difference as in equation 3.4.1. In order to reduce the effect of noise in the computation of motion, $M(p_{x,y})$ can be filtered by computing the motion of a pixel with respect to two previous values and averaging the result. Being $p_{x,y} \in F(i)$ a pixel in the current frame, $r_{x,y} \in F(i-1)$ and $q_{x,y} \in F(i-2)$; we define the motion of $p_{x,y}$ as,

$$M(p) = \frac{(1 - S(p_{x,y}, q_{x,y})) + (1 - S(p_{x,y}, r_{x,y}))}{2} \tag{6.3}$$

This way, motion in the scene is detected by considering similarities of three consecutive frames.

6.2.3 Segmentation process

Using the similarity criterion discussed in section 6.2.1 and the motion criterion discussed in section 6.2.2, we segment a frame and separate the pixels into two sets, based on their computed probabilities of belonging to foreground or background.

We define then the probability that any pixel $p \in F(i)$ belongs to foreground as,

$$P_{fore}(p) = max(M(p), 1 - S(p, b)) \tag{6.4}$$

because pixels will belong to foreground if either their motion value is high or their difference with the background is high. This way, we can include in the foreground set all pixels which, even being similar to the background but show significant motion and vice versa. On the other side, the following expression,

$$P_{back}(p) = max(1 - M(p), S(p, b)) \tag{6.5}$$

defines the probability that a pixel $p \in F(i)$ belongs to background if both its motion value is low and its similarity to current background is high, as stated in the constraints described before. It must be noted that the relationship,

$$P_{back} + P_{fore} = 1 \tag{6.6}$$

does not necessary verify, because the values of $M(p)$ and $S(p,b)$ are not related one to each other. [1]

The background algorithm with confidence (BAC) starts by taking a frame $F(i)$ to be the initial background model $B(i)$ (the model in time i), and sets,

$$\forall b_{x,y} \in B(i), c_{x,y}(i) = 0,\ \sigma_{x,y}(i) = 0 \tag{6.7}$$

being $c_{x,y}(i)$ the confidence value of pixel $b_{x,y}$ and $\sigma_{x,y}(i)$ the filtered probability in time i.

The value $c_{x,y}(i)$ measures how many times a background pixel $B_{x,y}$ has been continuously classified as background. This value is enclosed into the range $[0,1]$ after its update by computing $\frac{c_{x,y}(i)}{c_{x,y}(i)+1}$. On the other side, the value of $\sigma_{x,y}(i)$ measures how good the background classification of that pixel has been over the time. Values of $\sigma_{x,y}(i)$ are a measure over time of the confidence of the pixel $b_{x,y} \in B(i)$ and how close to previous values of the background model is. This value will be reduced in the case a pixel $p_{x,y} \in F(i)$ is continuously classified as foreground. As soon as this changes, and the pixel is again classified as background, the value of $\sigma_{x,y}(i)$ will increase again. The filtered probability is defined as a function of the confidence $c_{x,y}$ and the probability P_{back},

$$\forall b_{x,y} \in B(i) : \sigma_{x,y}(i) = \frac{\sigma_{x,y}(i-1) \cdot c_{x,y}(i) + P_{back}(p_{x,y})}{c_{x,y}(i)+1} \tag{6.8}$$

This filtered probability is used, together with other parameters, to avoid that an object captured in the first model stays forever in the model if it moved.

Figure 6.2 shows the evolution of $\sigma_{x,y}$ for a pixel which is continuously classified as background. Its background probability is modelled with a normal distribution.

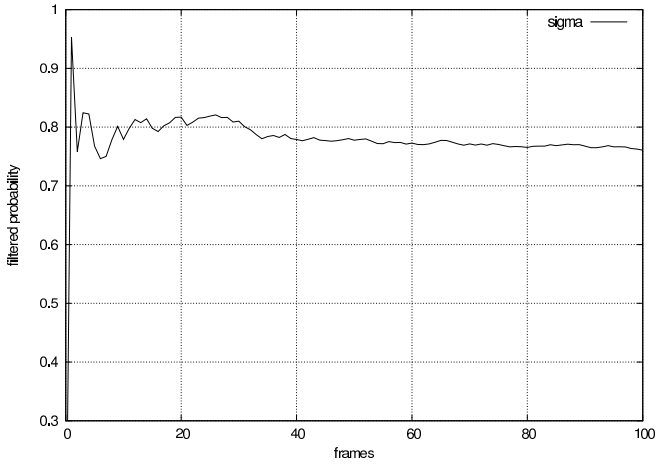


Figure 6.2: Simulated evolution of the filtered probability of a pixel (see equation 6.8) in the case it is always classified as background and its background probability follows a normal distribution.

Next two frames, $F(i+1)$ and $F(i+2)$, are ignored and used only to detect motion in frame $F(i+3)$. For all the next incoming frames $F(i)$, motion and similarities with $B(i-1)$ are sought for, to segment frame $F(i)$.

[1] Equation 6.6 would verify if the definition for $P_{back} = min(1 - M(q), S(q,b))$

Once $F(i)$ is segmented, the model $B(i-1)$ must be updated to obtain $B(i)$, using pixels in $F(i)$. Not all pixels $b_{x,y} \in B(i-1)$ are updated in the same way, it depends on $P_{back}(p_{x,y})$, $c_{x,y}(i-1)$ and $\sigma_{x,y}(i)$.

After background subtraction pixels of frame $F(i)$ are separated into four sets; the foreground set $fSet$, the background set $bSet$, $dSet$ for doubtful pixels and finally $cSet$ for pixels in $B(i)$ whose grey level will be replaced by the grey level of pixels in $F(i)$. Formally, these sets are defined as follows,

$$fSet = \{p_{x,y} \in F(i) : P_{fore}(p_{x,y}) > \tau, being\ 1 > \tau \geq P_{back}(p_{x,y})\} \tag{6.9}$$

$$bSet = \{p_{x,y} \in F(i) : p_{x,y} \notin fSet\} \tag{6.10}$$

$$dSet = \{p_{x,y} \in fSet : \sigma_{x,y}(i) < 0.8 \wedge \frac{c_{x,y}(i-1)}{c_{x,y}(i-1)+1} \geq 0.75\} \tag{6.11}$$

$$cSet = \{p_{x,y} \in fSet : \sigma_{x,y}(i) < 0.8 \wedge \frac{c_{x,y}(i-1)}{c_{x,y}(i-1)+1} < 0.75\} \tag{6.12}$$

Pixels will be classified as foreground in the $fSet$ only if their foreground probability is high. A restrictive probability thresholding τ can be used in equation 6.9 in order to consider as foreground only those pixels $p_{x,y}$ with very high values of $P_{fore}(p_{x,y})$. This way, it is possible reducing some of the shadows of the objects.

Doubtful pixels, $dSet$, are those whose filtered background probability is less than 0.8 (not high enough as to be decisive), but whose confidence is still over the third quartile (0.75), so high that makes doubt. Recalling that the algorithm starts with zero knowledge about the scene, special care must be taken with such behaviour, in order to quickly change pixels which do not describe the background properly. On the other side, pixels in the cSet represent pixels, whose confidence does not arrive to the third quartile, and will be replaced by values from the current frame. Elements of these two sets are in fact extracted from the $fSet$.

Values defining the previous sets should be chosen to be very restrictive, this way, pixels which may yield low background similarity are quickly replaced. In section 6.6, an study on different values of τ is performed.

The regions of interest of frame $F(i)$ are then defined by fSet. We define the following set for convenience,

$$\Lambda = bSet \bigcup dSet \tag{6.13}$$

6.2.4 Model update

We update background pixels in a different way, depending on their observed behaviour. Pixels which belong to the cSet are directly changed by values in $F(i)$. Pixels in the dSet will be forced to update strongly than those in the bSet in order to be more similar to the expected background. Being $p_{x,y} \in F(i)$ a pixel in current frame, the model $B(i)$ is updated as follows,

$$\forall b_{x,y} \in cSet : b_{x,y}(i) = p_{x,y}(i) \tag{6.14}$$

$$\forall b_{x,y} \in \Lambda : b_{x,y}(i) = b_{x,y}(i-1) + (1-\alpha) \cdot p_{x,y}(i) \tag{6.15}$$

Confidence of pixels is also updated distinguishing the set to which each pixel belongs to. In this case, pixels which describe the background increase their confidence and for the others, the confidence is decreased. On the other side, pixels which are copied from the frame, take a confidence equal to zero. The confidence of pixels in $B(i)$ is updated according to the following expressions,

$$\forall p_{x,y} \in bSet : c_{x,y}(i) = c_{x,y}(i-1) + 1 \tag{6.16}$$

$$\forall p_{x,y} \in dSet : c_{x,y}(i) = c_{x,y}(i-1) - 1 \tag{6.17}$$

$$\forall p_{x,y} \in cSet : c_{x,y}(i) = 0 \tag{6.18}$$

As this operation is performed in a frame by frame basis, and pixels are classified after segmentation, any pixel whose confidence is reduced by a temporal occlusion by a foreground pixel will recover its previous confidence as soon as the occlusion finishes.

The value $c_{x,y}$ is never lower than 0 because, should the confidence of a pixel be under 0.75, the pixel would be classified in the $cSet$, see equations 6.12 and 6.18, and thus, its confidence will be set to $c_{x,y} = 0$.

This definition of confidence, which is further used to compute the adaptation coefficient of the model, allows a pixel to change very quick when its confidence is low and become more static as time elapses. In figure 6.3 the simulated evolution of the confidence of a pixel, constantly classified as background, is shown. The plot shows that initially the value is small but quickly saturates.

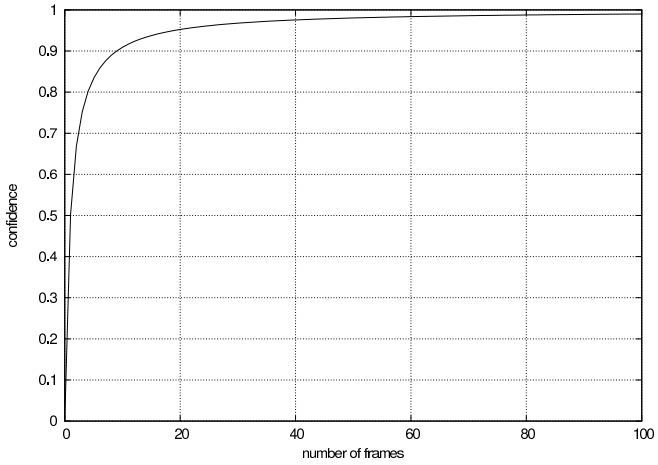


Figure 6.3: Simulated evolution of $\frac{c_{x,y}(i)}{c_{x,y}(i)+1}$ for a pixel which is continuously classified as background.

A difference with respect to other algorithms is that we propose using a different adaptation coefficient $\alpha_{x,y}(i)$ for each pixel depending on $c_{x,y}(i)$ they show. This way, we expect pixels which strongly described the scenario to update smoothly. On the other side, pixels whose confidence diminishes, recalling this means their background probability is descending, take a lower $\alpha_{x,y}(i)$.

The adaptation coefficient for pixel $p_{x,y}$ is computed taking into account the confidence of the pixel in time i according to the following equations,

$$\forall p_{x,y} \in \Lambda : \alpha_{x,y}(i) = 0.98 \cdot \frac{c_{x,y}(i)}{c_{x,y}(i)+1} \tag{6.19}$$

$$\forall p_{x,y} \ni \Lambda : \alpha_{x,y}(i) = 0 \tag{6.20}$$

This coefficient $\alpha_{x,y}$, takes values in the range $[0, 0.98)$. Being 0.98 the value which corresponds to pixels with a high confidence and 0 the value which corresponds to pixels which have been changed. We used the value 0.98 as the maximum value for the adaptation confidence in order to force all pixels to cope with light conditions, except those which received a new value.

6.2.5 Segmentation confidence

As said before, together with its grey level value, each pixel provides a confidence value which may be used to weight the quality of the segmentation. We define the segmentation confidence of the model $B(i)$ as,

$$sc = \frac{1}{M \cdot N} \cdot \sum \frac{c_{x,y}(i)}{c_{x,y}(i)+1}, \forall b \in B(i) \tag{6.21}$$

being $M * N$ the number of pixels of the model. The segmentation confidence sc can be calculated for a target of frame $F(i)$, by particularizing this expression considering only the pixels segmented for this target.

The segmentation provided by model pixel $b_{x,y} \in B(i)$ will depend of its confidence as said before. We can calculate the segmentation confidence (sc) for a region of interest R_i with an area a of frame $F(i)$ segmented using model $B(i)$ as:

$$sc(R_i) = \frac{1}{a} \sum \frac{c_{x,y}}{c_{x,y}+1}, \forall p_{x,y} \in R_i \tag{6.22}$$

The value $sc(R_i)$ measures only the quality of the pixels segmented as foreground, not the segmentation itself. This means that it does not measure if pixels are correctly segmented but the confidence that we can have on the pixels that are labelled as foreground. There is no direct relationship between the amount of segmented pixels of an object and the confidence of their segmentation.

Finally, in order to test when the background model is stable the mean square quadratic difference ($msqd$) of the difference between two consecutive models is calculated. An stop condition for the algorithm could depend on the verification of the following condition,

$$msqd(B(i), B(i-1)) < 10^{-3} \wedge sc > 0.995 \tag{6.23}$$

6.2.6 Corrupt model

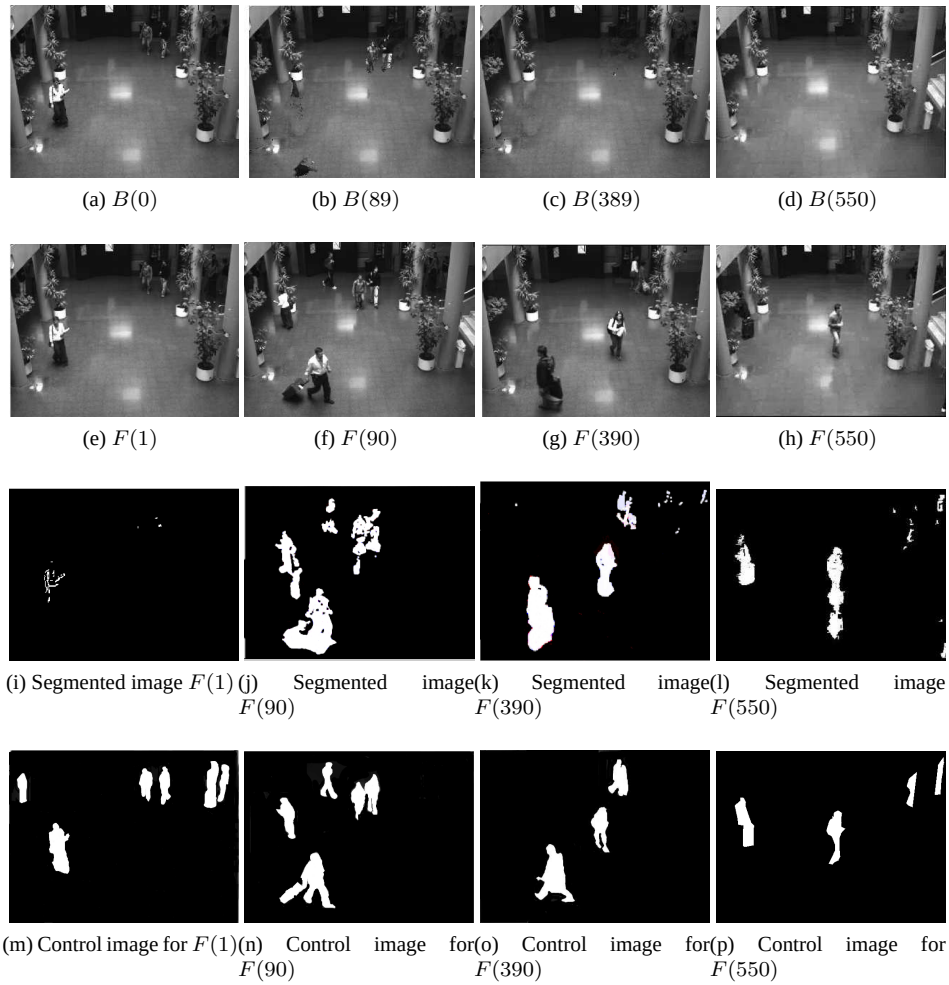
Once pixels are classified as foreground or background, a simple criterion to detect corrupt models is used.

The hypothesis is made, that scenarios contain more background than foreground pixels. The number of foreground pixels P is computed at this stage and compared

with the total amount of pixels $N \cdot M$ in the scene. Following the hypothesis, it should verify that $P < N \cdot M$. In fact, a real number $\mu < 1$ can be found that $P = \mu \cdot N \cdot M$.

The value μ can be used to detect corrupt models, in the case $\frac{P}{N \cdot M} > \mu$ then the model can be considered as corrupt. Pixel motion is implicitly considered, as long as foreground pixels are defined in terms of their discrepancy with background or with motion in their location. The value μ should be set experimentally depending on background clutter.

In the case the process is restarted in time i, model values are set $\forall b \in B, B_b^1(0) = F_b(i)$, $c_b = 0)$. With this initialization, we assume that most pixels which were considered as background are still background, but with different values.



(a) $B(0)$ (b) $B(89)$ (c) $B(389)$ (d) $B(550)$

(e) $F(1)$ (f) $F(90)$ (g) $F(390)$ (h) $F(550)$

(i) Segmented image $F(1)$ (j) Segmented image $F(90)$ (k) Segmented image $F(390)$ (l) Segmented image $F(550)$

(m) Control image for $F(1)$ (n) Control image for $F(90)$ (o) Control image for $F(390)$ (p) Control image for $F(550)$

Figure 6.4: Sequence of background model reconstruction with BAC. From top to bottom, in columns, background model, incoming frame, automatic and hand segmentation for frames in t = 1, 90, 390 and 550. Output of automatic segmentation is filtered with a size filter to remove noise.

6.2.7 Results obtained with a particular sequence

A representative situation aim of our developments is evaluated in this section, see figure 6.4.

Results of the experiments are analysed, considering the amount of true positives (TP) obtained in the segmentation process for each object in some frames, see table 6.1. And the goodness of the background model built by the BAC algorithm, considering the amount of true negatives (TN) is shown in figure 6.5. Also, the evolution of model's confidence is considered. The evolution of the model's construction and how object behaviour influences it, is discussed through concrete situations in which objects enter or leave the scene or stay still.

In order to evaluate quantitatively the evolution of BAC, we first segmented manually 22 frames randomly selected of a sequence that starts in $F(0)$ with several people in a scene, simulating a surveillance system, in that moment $B(0)$ is created with targets and $sc = 0$, see figure 6.4.

The objects present in the scene are segmented using the model obtained with BAC and with the mean. The segmentation using the background model obtained with the mean was performed subtracting the background and thresholding it with a unique threshold ($T = 15$).

In table 6.1, segmentation results for frames $F(90)$, $F(390)$ and $F(550)$ obtained with BAC and mean are compared; sc of pixels found in each target's segmentation with BAC is shown under column sc The original frames, together with segmentation result and the background model built by BAC, may be found in figure 6.4.

Target	*Frame* 90			*Frame* 390			*Frame* 550		
	mean	*BAC*	sc	*mean*	*BAC*	sc	*mean*	*BAC*	sc
1st target	0.74	0.69	0.95	0.88	0.96	0.99	0.75	0.71	0.71
2nd target	0.70	0.90	0.98	0.71	0.89	0.99	0.74	0.83	0.82
3rd target	0.49	0.37	0.99	0.69	0.51	0.99	-	-	-
4th target	0.49	0.47	0.92	-	-	-	-	-	-

Table 6.1: Rate of TP found for each target using different methods to obtain the background model, BAC and mean. In control frame 90 a total amount of four objects were found, in 390 only three and in frame 550, two objects were found. Targets are not the same in frames.

In $F(90)$, a low rate of TP is obtained for both algorithms, due to the fact that they are far in the field of view of the camera are segmented more poorly (target 3); something similar happens with target 4, which is a group of two people moving still in the same area they occupied at the beginning of the movie.

In figure 6.4a and 6.4b $B(0)$ and $B(89)$ are shown, it may be seen that in $B(89)$, background model has achieved $sc = 0.982$ and some targets have been removed. Improvement over time is evident as $B(389)$ contains no target. This improvement manifests in $F(390)$ and $F(550$ with a better segmentation.

Evolution of model's confidence obtained for each input frame $F(i)$, TP and TN, of BAC and mean can be seen in figure 6.5 for the manually segmented frames. Objects standing still for long periods of time influence negatively the value of TP for both algorithms. Figure 6.5 shows that BAC obtains higher values of TP than mean. In $F(201)$ several objects leave the scene and others start coming in and in $F(680)$ some objects stand still; this explains that some foreground pixels are not found. On the other side, TN, easily reach a high level as area of quiet targets is small compared to the image. It can be seen that BAC keeps an stable value for TN during its execution,

background modelling using the mean loses accuracy due to the objects that stood still at the beginning of the process.

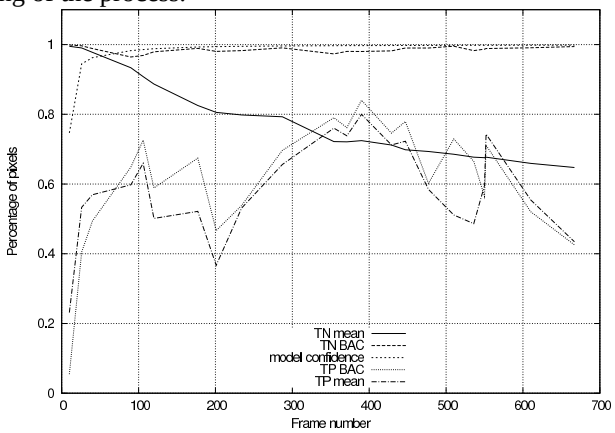


Figure 6.5: Evolution of confidence, TP and TN for the discussed video. Spots correspond to control frames.

This experiment shows that BAC obtains background models comparable to those obtained by using any statistical technique; with the added benefit of permitting segmentation from the very beginning of the process.

6.3 BAC with colour coordinates

In this section we extend the algorithm presented in previous section to work with colour coordinates, by adding the use of RGB coordinates. By doing this, better results for foreground and background are expected.

Choosing a colour coordinates model depends on several factors. On one side, usually, cameras can produce images in RGB or YUV coordinates and though mathematical methods exist that can convert coordinates of one system to another, this conversion may be time consuming and have a severe impact on the system performance. Other systems exists, such as $CIEL^*a^*b^*$, for instance, which could be claimed to have better properties than RGB.

We chose the RGB system because it is a widely used system and it is easy to find cameras which reproduce images using this system. RGB has the drawback that it is not perceptually uniform because it was designed from the perspective of devices and not from a human perspective.

6.3.1 Extension to RGB

We extend first the similarity criteria introduced in section 6.2.1 using RGB coordinates. The similarity between two pixels, p and q is now given by an extended version of equation 6.1. The only difference is that the distance between the pixels is extended to use the three RGB coordinates where the colour distance is computed as the Euclidean distance of two colour vectors and κ is a constant determined experimentally. In this case, equation 6.1 is modified and the similarity function is given by,

$$S(p,q) = e^{-\frac{dist}{\kappa}} \tag{6.24}$$

$$dist = \sqrt{(p_R - q_R)^2 + (p_G - q_G)^2 + (p_B - q_B)^2)} \tag{6.25}$$

As in previous section, motion can be computed using equation 6.24. Motion of a pixel can be defined as its similarity with previous values of the pixel. Being $q_{x,y} \in F(i)$ a pixel in the current frame with three colour components, $p_{x,y} \in F(i-1)$ and $r_{x,y} \in F(i-2)$; we define the motion of $q_{x,y}$ as,

$$M(q) = \frac{(1 - S(p_{x,y}, q_{x,y})) + (1 - S(r_{x,y}, q_{x,y}))}{2} \tag{6.26}$$

The remaining processes are kept the same, as they all involve only computations with probabilities or are easily extended to the three RGB coordinates. A comparison of this process with the original BAC is made by using the Wallflower benchmark in section 6.6.

6.4 MBAC

The BAC algorithm is designed to maintain one model per pixel and it encounters difficulties when it has to deal with clutter in the background or outdoor scenes in which there is a significant amount of motion in the background, for instance, scenarios with waving trees. The use of just one model to describe the background limits the capabilities of the algorithm for such scenes.

In order to cope with this problem, we add the possibility of describing the background with more than one model. Each model will contain the pixel's colour description and a confidence value used to discard models. This confidence value measures how the model matched the pixel in the last frames, and will be close to one if the model matches closely the pixel's value, and close to zero if it does not.

The extension to multiple models per pixel is quite straightforward. Recalling equation 3.1, comparing the value of $p_{x,y}$ with the v models that describe it in the model can be reduced to find the model which yields the maximum similarity with $F_{x,y}$ as computed in equation 6.1.

Following the approach introduced by [grimson99], after background segmentation and model adaptation, the models for each pixel are then ordered in descending order according to their confidence. Then, the first K models whose confidence sums equal or more than a threshold μ are kept and the rest are removed. We employed a value K to indicate the number of models considered per pixel.

Figure 6.6 shows the distribution of pixels which took different colour values due to motion in the background. BAC is unable to handle this situation and this justifies its extension in order to handle several models per pixel. The distribution is represented in grey tones, black pixels did never change their initial model; white pixels indicate a heavy rate of model changes.

Figure 6.6 illustrates the suitability of using a different amount of models per pixel instead of setting a unique value for the entire image and that maybe this information could be used to decide which pixels need more models, or less, than initially expected.

6.4.1 Segmentation process with MBAC

The algorithm MBAC starts by taking a frame $F(i)$ to be the initial background model $B(i)$ (the model in time i), and sets,

Figure 6.6: On the left, image showing the areas in which background models were changed in order to adapt to changes in the background. The frame corresponds to an outdoor sequence of the Wallflower benchmark. On the right, one frame of the original sequence.

$$\forall b \in B, \forall\ 1 \leq m \leq K : c_b^1(0) = 0.01 \tag{6.27}$$

being $c_b^1(0)$ the confidence value of the m-th model of pixel b in time $i = 0$. This confidence value measures how good the model describes the pixel. As pixel b may have more than one mode, a parameter K, limits the maximum number of models for pixel b.

Next two frames, $F(i+1)$ and $F(i+2)$, are ignored and used only to detect motion in frame $F(i+3)$. For all the following frames $F(j),\ j \geq i+3$, motion and similarities with $B(i-1)$ are sought for. We define the probability that any pixel q belongs to background as,

$$pBack(q) = max(1 - M(q), max(S(q, B_b^m))) \tag{6.28}$$

analogously, the probability of belonging to the foreground is computed as,

$$pFore(q) = max(M(q), 1 - max(S(q, B_b^m))) \tag{6.29}$$

being $S(q, B_b^m)$ the similarity between pixel q and the m-th background model of pixel $b \in B(i-1)$ and $M(q)$ the motion of a pixel as defined in equation 6.3 .

It must be noted that the relationship,

$$pFore(q) + pBack(q) = 1 \tag{6.30}$$

does not necessary verify, according to the definitions of both probabilities.

It is also straightforward to see, that equation 6.28 describes mathematically the intuitive idea that pixels similar to background or which are reasonably stationary have a bigger probability of belonging to background.

On the other side, equation 6.29 states that pixels very different to background or very different to previous values will have a bigger probability of belonging to foreground.

The segmentation separates pixels in two different sets; the foreground set (fSet) and the background set (bSet), defined as,

$$fSet = \{p \in F(i) : pFore(p) > \tau,\ being\ 1 > \tau \geq pBack(p)\} \tag{6.31}$$

$$bSet = \{p \in F(i) : p \notin fSet\} \tag{6.32}$$

By changing the value of τ, we restrict the criteria to keep only pixels with high foreground probability in order to remove possible shadows. Anything which is not considered to be foreground, is classified as background.

A notable difference with BAC, is that in this case no $cSet$ nor $dSet$ sets are built after segmentation. This is due to the fact, a more flexible schema using the confidences of models was used to decide whether one of the K possible descriptions of a pixel in the background model should be kept or removed.

6.4.2 Corrupt model

Once pixels are classified as foreground or background, a simple criterion to detect corrupt models is used.

The hypothesis is made, that scenarios contain more background than foreground pixels. The number of foreground pixels P is computed at this stage and compared with the total amount of pixels $N \cdot M$ in the scene. Following the hypothesis, it should verify that $P < N \cdot M$. In fact, a real number $\mu < 1$ can be found that $P = \mu \cdot N \cdot M$.

The value μ can be used to detect corrupt models, in the case $\frac{P}{N \cdot M} > \mu$ then the model can be considered as corrupt. Pixel motion is implicitly considered, as long as foreground pixels are defined in terms of their discrepancy with background or with motion in their location. The value μ should be set experimentally depending on background clutter.

In the case the process is restarted in time i, model values are set $\forall b \in B, B_b^1(0) = F_b(i)$, $c_b^1 = max(c_b^m)$. With this initialization, we assume that most pixels which were considered as background are still background, but with different values.

6.4.3 Model update

In the case the model is not restarted ($\frac{P}{N \cdot M} \leq \mu$), it is updated with information of frame $F(i)$. For every pixel $b_{x,y} \in bSet$, the confidences and colour information are updated. In order to cope with light changes, the model m which matched the background is updated as,

$$B_{x,y}^m(i) = \alpha \cdot B_{x,y}^m(i-1) + (1-\alpha) \cdot F_{x,y}(i) \tag{6.33}$$

and its confidence is updated according to the following expression,

$$c_{x,y}^m(i) = \alpha \cdot c_{x,y}^m(i-1) + (1-\alpha) \cdot pBback(p_{x,y}) \tag{6.34}$$

Note that this computation of confidence is similar to equations 6.16, 6.17 and 6.18 used for BAC, but simplified in order to reduce computation time when several models are considered. Also, provided that several models are considered per pixel and only one is adapted per frame, the adaptation factor can be always the same.

Any other model l describing pixel b which did not match the background model is updated according to equation 6.35,

$$c_{x,y}^l(i) = \alpha \cdot c_{x,y}^l(i-1),\ \forall l \neq m \tag{6.35}$$

where α is a learning rate factor in the range $[0, 1]$. The difference with equations 6.16, 6.17 and 6.18 and the general process followed in BAC is that it is easier to control the quality of each model for each pixel if they are pondered the same way.

For every pixel $p_{x,y} \in fSet$, the K background models describing it are ordered in descending order according to their confidences. The sum of the confidences of pixel $p_{x,y}$ is given by,

$$s_{x,y} = \Sigma c^m_{x,y}(i), 1 \leq m \leq K \tag{6.36}$$

In the case $s_{x,y} < \gamma$, a value set by user, a new model v will be added or the worst model will be replaced by the current value of pixel $p_{x,y} \in F(i)$. For the new model v, the algorithm sets $B^v_{x,y}(i) = p_{x,y}$ and $c^v_{x,y}(i) = 0.01$.

The parameter γ controls the speed at which new models are included in pixels' descriptions. If γ is very close to 1, the algorithm will try to maintain always the most accurate possible description of each pixel. In the case γ is close to 0, it will show a bigger resistance to change the model.

Thus, γ should be kept big for scenarios with a relatively quiet background and small for scenarios with background clutter.

The confidence computed for each pixel is an indicator of how good its models describe the background, in this case, MBAC computes the confidence of a region R_i as,

$$sc = \frac{\sum_{x,y} max(c^m_{x,y}(i))}{|R_i|} \quad \forall 1 \leq m \leq K \tag{6.37}$$

being $|R_i|$ the total amount of pixels in the region R_i.

Thus, an overall estimation of the background's model quality in time i is given by,

$$Q(i) = \frac{\sum_{\forall b, \forall m} max(c^m_{x,y}(i))}{N \cdot M} \tag{6.38}$$

recalling that $N \cdot M$ is the total amount of pixels in the background. Values of $Q(i)$ close to 1 indicate a model which describes the scene accurately, on the other side, values of $Q(i)$ close to 0 indicate the opposite.

Experimental results using the Wallflower benchmark are discussed in section 6.6.

6.5 Fuzzy background subtraction (FBS)

BAC and MBAC compute the segmentation of frame $F(i)$ by considering its similarities with background model $B(i-1)$ and the motion in each pixel. This computation has some drawbacks. First, a fixed threshold must be set in equations 6.1 and 6.9.

Also, the MBAC extension can produce wrong results if computations are not accurate. For instance, one potential problem occurs when a pixel is oscillating in different values. It is possible that very close values are considered to be different and thus, the pixel could have several models very similar among them.

A different line in our research is opened when the uncertainty in threshold selection is faced. In this section we introduce an study of how this uncertainty in the threshold selection can effectively be faced using fuzzy-like techniques. The background subtraction method introduced in this section is based on the common sense

fact that background pixels should yield small distances when subtracted from background. Thus, a per-frame threshold can be computed if a method is obtained to determine when a distance is sufficiently small.

In section 6.5.1 we discuss the empirical results on which we based the design of our method for threshold computation. Two membership functions are built per frame according to the distances of pixels to the background model.

Based on the threshold computation algorithm described in section 6.5.2 we propose an algorithm for background modelling and object detection, which considers consecutive frames $F(0), F(1), ...F(n)$, expressed in CIEL*a*b* coordinates, representing either outdoor or indoor scenarios. Pixels in the background model are described with several models in order to cope with clutter in the background of the scene.

Besides, we propose a mathematical definition of model corruption based on the values of the membership functions.

The background subtraction of each frame is divided into three different stages following a similar schema as the one introduced in [toyama99]. In the first one, a pixel-wise subtraction as explained in section 6.5.3 is performed. The following stage, discussed in section 6.5.4, recovers foreground pixels which could be erroneously considered as background by taking into account spatial relationships with their neighbours. In a third step, if the background is not corrupted see section 6.5.5, it is updated 6.5.6.

By calling our technique fuzzy, we want to stress the fact that our algorithm is inspired in the results obtained in fuzzy logic. As explained later in this section, instead of generating a threshold based on a priori statistics, the algorithm computes a threshold based on an empirical rule applied to actual data.

6.5.1 Empirical results

We base our approach in the idea that background pixels of frame $F(i)$ should yield small distances when subtracted from an accurate background model $B(i-1)$. In this section, we introduce the empirical results that lead us to build an algorithm based on this simple idea.

We performed two kind of off-line experiments with about 2000 frames of different sequences representing different scenarios, among them, some of the Wallflower benchmark [toyama99] used later in the experiments.

In the first set of experiments, the pixels of two consecutive frames of a sequence are pixel-wise subtracted. The aim of these experiments was testing what is the distribution of pixels according to their distance to a background model and according to distances to the previous frame. No foreground regions were considered.

Figures 6.7, 6.8 show the results for two frames of different sequences. Frames are also subtracted from a background model built using a frame by frame update using the algorithm discussed in section 3.6. Both frames are very similar and small differences are expected. In the left column, from top to bottom, it can be seen frame $F(i-1)$, frame $F(i)$, on the right column, from top to bottom, the plot of pixels of the subtraction result grouped by distance to background model, built in a frame by frame basis as the average of previous background values, and the plot of pixels of $F(i)$ grouped by distance to $F(i-1)$.

Another set of experiments was performed taking an initial background model, updated in a frame by frame basis using the algorithm discussed in section 3.6, and subtracting each frame from it. In this case, the distribution of distances of foreground

and background pixels to the background were considered. In order to know which pixels belong to background and which to foreground, pixels were manually labelled. Figures 6.9, 6.10 and 6.11 show the results for a selected group of frames of different sequences, top left image the current frame $F(i)$, the image bottom left represents the manual segmentation of the frame, on the right column, image at top shows the plot of background pixels according to distance to the background model $B(i-1)$ and on bottom the plot of foreground pixels according to distance to the background model $B(i-1)$.

From the results shown in figures 6.7, 6.8, 6.9, 6.10 and 6.11, it can be seen that there is a similarity in how background curves behave. There is a fast grow of the curves of background pixels for small distances, which becomes smoother as the distance increases. In the case of foreground pixels the behaviour is quite different, having the curve small values for small distances and raising to high values when the distance to background is big. This behaviour can be modelled with two membership functions as those depicted in figure 6.12.

In this figure, d is the distance between two pixels and the Y-axis represents the membership value. Two functions are represented, on the left, the background membership function (MB) and on the right, its inverse, the foreground membership function (MF). The assumption can be made, that up to a distance c', pixels have a background membership value of 1. If the distance takes values greater than c', then the background membership decreases and, inversely, foreground membership increases.

The decrease of membership value can be modelled in several ways, one of the easiest is using a straight line with an slope proportional to the probability decrease. The cross of the straight line with the X axis marks the distance at which membership is zero. In our case, the point c marks the distance at which any pixel will be considered as foreground with a membership value of 1 and background with a value 0.

The condition $MB < MF$ could be considered a valid threshold to classify image pixels according to their distance to background. Unfortunately, it is quite difficult knowing in advance which the shape of the distances of background pixels will be. In the following section, we introduce a method to approximate MB and MF.

6.5.2 Computation of membership functions

In this section we explain how the background membership function MB is built. Provided that we only classify elements into two different classes, background and foreground, the following relationship must meet for any pixel $F_{x,y}(i)$,

$$MB(d_{x,y}) + MF(d_{x,y}) = 1 \tag{6.39}$$

We start by considering the euclidean distance $d_{x,y}(i)$ between pixel $F_{x,y}(i)$ and $B_{x,y}(i-1)$ as,

$$d_{x,y} = |F_{x,y}(i) - B_{x,y}(i-1)|,\ \forall F_{x,y}(i) \tag{6.40}$$

In order to ease the following explanations, we define the normalized cardinality of a distance d as,

$$Card(d) = \frac{Card(\{F_{x,y}(i) : d_{x,y} \leq d\})}{M \cdot N} \tag{6.41}$$

That is, the amount of pixels $F_{x,y}(i)$ which yield a distance to background less than d divided by the total amount of pixels in the image $M \cdot N$.

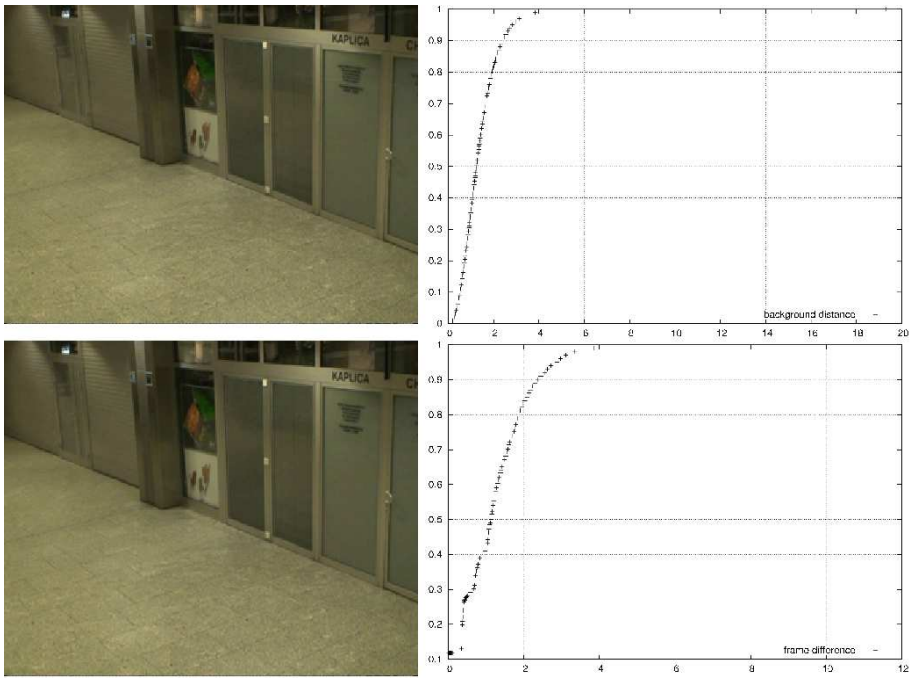


Figure 6.7: From top to bottom, the left column shows frame $F(i-1)$ and frame $F(i)$. And the right column shows a plot of the accumulated percentage of pixels according to their distance to the background model, and a plot of the accumulated percentage of pixels according to their distances to $F(i-1)$. As expected, a big amount of pixels yield small values and the curve reaches the total amount of pixels in the image with small distances. Plots on the right measure distances in CIELAB coordinates.

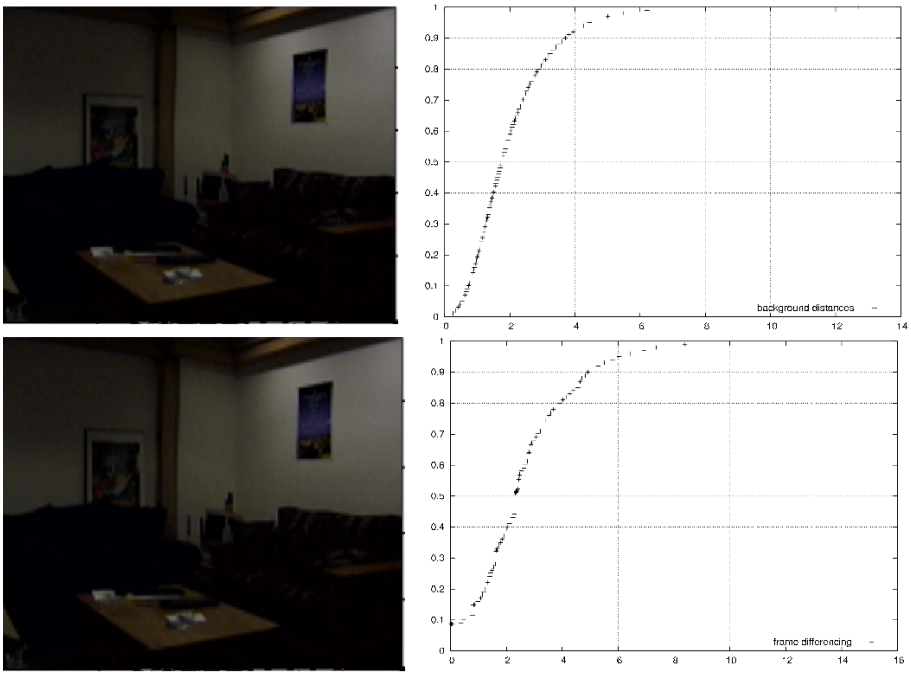


Figure 6.8: From top to bottom, the left column shows frame $F(i-1)$ and frame $F(i)$. And the right column shows a plot of the accumulated percentage of pixels according to their distance to the background model, and a plot of the accumulated percentage of pixels according to their distances to $F(i-1)$. As expected, a big amount of pixels yield small values and the curve reaches the total amount of pixels in the image with small distances. Plots on the right measure distances in CIELAB coordinates.

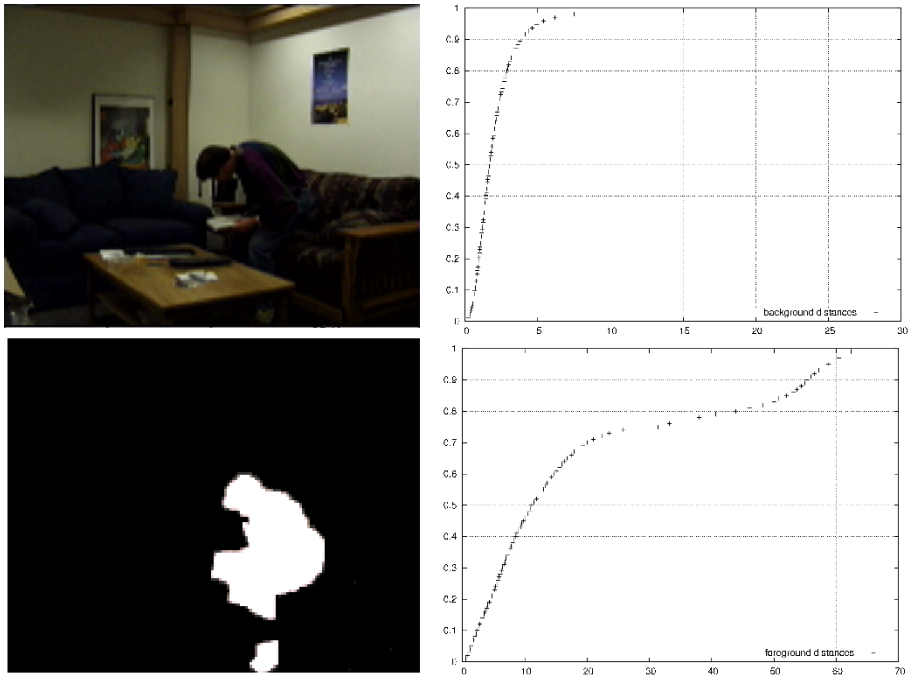


Figure 6.9: The distribution of pixels according to the distance to background model. From top to bottom the left column shows the original frame, the manually segmented image, the column on the right shows the plot of the accumulated percentage of background pixels grouped according to their distance to the background model and finally the plot of accumulated percentage of foreground pixels according to their distance to background model. As expected, background pixels yield small values and the curve reaches the total amount of pixels and the curve for foreground pixels starts raising at further distances. Plots on the right measure distances in CIELAB coordinates.

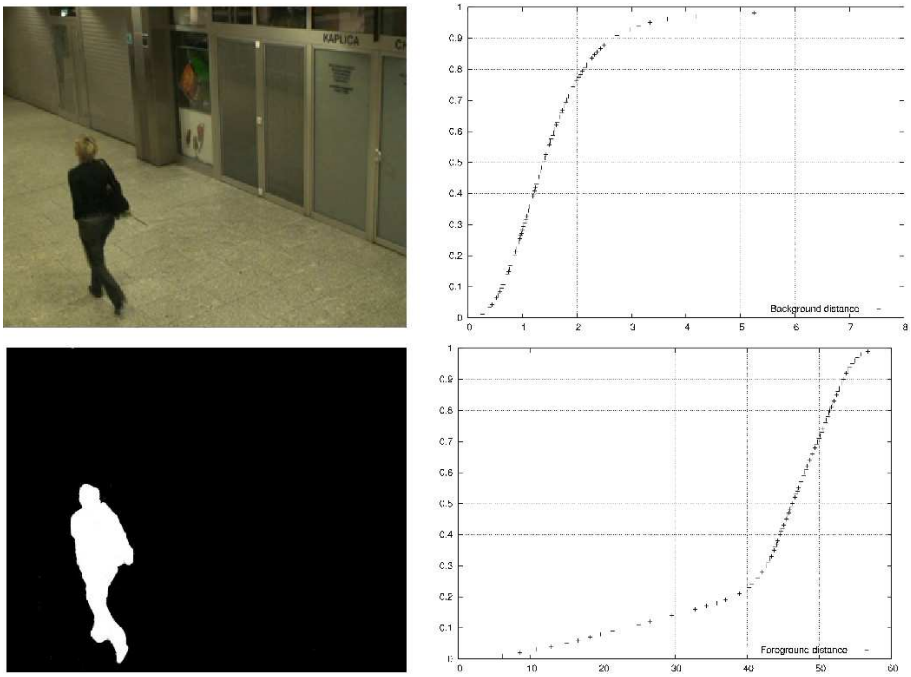


Figure 6.10: The distribution of pixels according to the distance to background model. From top to bottom the left column shows the original frame, the manually segmented image, the column on the right shows the plot of the accumulated percentage of background pixels grouped according to their distance to the background model and finally the plot of accumulated percentage of foreground pixels according to their distance to background model. As expected, background pixels yield small values and the curve reaches the total amount of pixels and the curve for foreground pixels starts raising at further distances. Plots on the right measure distances in CIELAB coordinates.

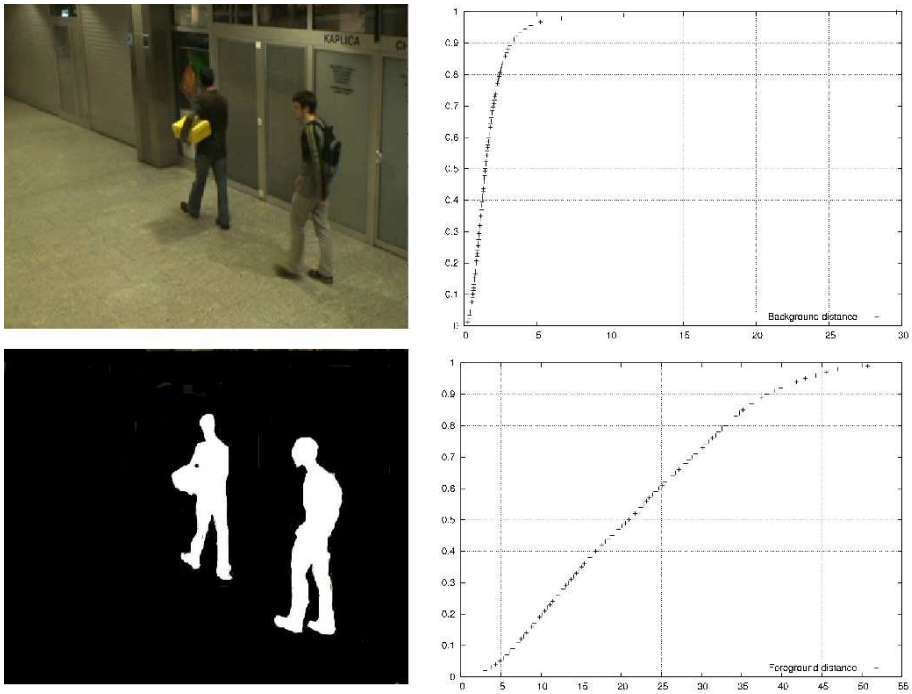


Figure 6.11: The distribution of pixels according to the distance to background model. From top to bottom the left column shows the original frame, the manually segmented image, column on the right shows the plot of the accumulated percentage of background pixels grouped according to their distance to the background model and finally the plot of accumulated percentage of foreground pixels according to their distance to background model. As expected, background pixels yield small values and the curve reaches the total amount of pixels and the curve for foreground pixels starts raising at further distances. Plots on the right measure distances in CIELAB coordinates.

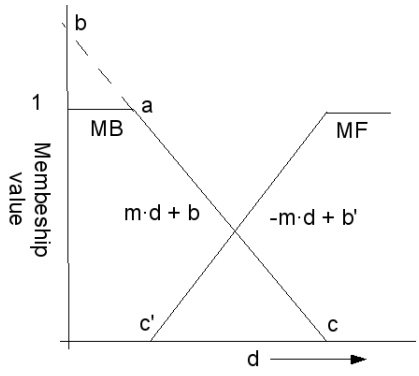


Figure 6.12: Representation of two membership functions, MB and MF. The point labelled with a corresponds to the last value which verifies $MB = 1$ and $MF = 0$. The decrease of function MB is modelled by the line with slope m, crossing X axis at $d = c$. It may be seen that both functions are symmetrical.

Figure 6.12 shows that the membership functions may be approximated by using a straight line between two points with distances d_0 and d_1 parametrized as,

$$m = \frac{MB(d_1) - MB(d_0)}{d_1 - d_0} \tag{6.42}$$

considering d_0 the maximum distance at which it is verified $MB(d_0) = 1$, the parameter b of the straight line can be obtained as,

$$b = 1 - m\ d_0 \tag{6.43}$$

where distances d_0 and d_1 are computed algorithmically.

Observing previous plots in figures 6.7, 6.8, 6.9, 6.10 and 6.11 it can be seen that there is a big amount of pixels which yield very small distances to background. Empirical results show that this amount is around 30% of the total amount of background pixels in the image. Recalling figure 6.12, we can set the point $(1, d_0)$ to be at the distance d_0 that meets the constraint $Card(d_0) < Card_0$, being $Card_0$ a constant value.

Once d_0 is determined, the crossing point of both functions at distance d_1 is computed. In order to do this, again from figures 6.7, 6.8, 6.9, 6.10 and 6.11, it can be noted that the distribution of distances of background pixels tends to raise very quickly. In the absence of noise, this means that the constraint $Card(d_i) < Card(d_{i+1})$ will be met if $d_i < d_{i+1}$. We can approximate the angle of the curve with the X axis between points d_i and d_{i+1} as,

$$\beta = atan(\frac{Card(d_i) - Card(d_{i+1})}{d_{i+1} - d_i}) \tag{6.44}$$

However, the curve stops raising when the distance begins being considerably big and there are few background pixels to be added to the set. It can be appreciated, that the angle of the curve with the X axis takes values close to zero. Usually, at this point the plot for foreground pixels start to take high values. However, real curves have irregularities due to noisy inputs and that sometimes, small angles are obtained for consecutive points of the curve because there is a few increase of pixels between two neighbouring distances.

Bearing all this in mind, we propose testing that the constraint $\beta \leq \beta_0$ is met a number of times before computing the crossing point of MB and MF, being β_0 a constant determined empirically. The distance d_1 for which $\beta \leq \beta_0$ is met for the third time, is the last distance for which $MB(d_i) > MF(d_i), \forall d_i \leq d_1$.

The background membership function can be then written as a function of distance $d_{x,y}$, as follows,

$$MB(d_{x,y}) = \begin{cases} 1,\ if\ 0 \leq d_{x,y} \leq d_0 \\ m\ d_{x,y} + b,\ if\ d_0 < d_{x,y} \leq \frac{-b}{m} \\ 0,\ if\ \frac{-b}{m} < d_{x,y} \end{cases} \tag{6.45}$$

Algorithm 4, formalizes the previous method to compute d_0 and d_1, the threshold that separates background and foreground, is computed. The saturation of the curve is modelled by comparing the angle of the slope between two consecutive points of the curve with a constant angle β_0.

Algorithm 4 Membership function algorithm for computing the threshold d_1 that separates background and foreground pixels following the knowledge acquired with the experiments of section 6.5.1. The ϵ used in this algorithm is an small value set in our experiments to $\epsilon = 0.1$.

$d_{x,y} \leftarrow |F_{x,y}(i) - B_{x,y}(i-1)|$
$d_0 \leftarrow 0$
$\epsilon \leftarrow 0.1$
$\beta_0 \leftarrow 1^\circ$
while $(Card(d_0) < Card_0)$ **do**
  $d_0 \leftarrow d_0 + \epsilon$
end while
$d_i \leftarrow d_0$
$detections \leftarrow 0$
while $(detections < 3)$ **do**
  $d_{i+1} \leftarrow d_i + \epsilon$
  $\beta \leftarrow atan(\frac{Card(d_i) - Card(d_{i+1})}{d_{i+1} - d_i})$
  if $(\beta < \beta_0)$ **then**
    $detections \leftarrow detections + 1$
  end if
  $i \leftarrow i + 1$
end while
$d_1 \leftarrow d_i$

6.5.3 Pixel level processing

Background subtraction methods depend basically on a threshold which acts as a high-pass filter removing from the output all pixels whose difference from the background is smaller than a given value. Instead of relying on a single predefined threshold, we propose computing a background membership function to find out the suitable threshold that separates background and foreground after each background subtraction. This threshold is computed following the algorithm 4 and expression 6.45.

For each frame $F(i)$, the background subtraction is computed as the euclidean distance of the colour coordinates of each pixel $F_{x,y}(i)$ from each $B^v_{x,y}(i)$, being $B^v_{x,y}(i-1)$ the v-th model for pixel in coordinates x, y in time $i-1$. This distance, denoted by $d^v_{x,y}$, is computed as,

$$d^v_{x,y} = |F_{x,y}(i) - B^v_{x,y}(i-1)|, \ \forall F_{x,y}(i) \tag{6.46}$$

The distance finally considered is,

$$d_{x,y} = min(d^1_{x,y}, d^2_{x,y} ... d^v_{x,y}), \forall v \tag{6.47}$$

Thus each $d_{x,y}$ stores the smallest distance between $F_{x,y}(i)$ and the $B^v_{x,y}(i-1)$.

Following the algorithm 4, we compute a threshold d_1 that separates background and foreground pixels. Background pixels are labelled with a 0 and foreground pixels with 1 by means of a binary image S built as follows,

$$\forall F_{x,y}(i), \ S_{x,y}(i) = \begin{cases} 0, \ if \ d_{x,y} < d_1 \\ 1, \ otherwise \end{cases} \tag{6.48}$$

6.5.4 Foreground recovery

The aim of this stage is recovering foreground pixels that may have been lost due to inaccurate segmentation. We followed a similar approach in this stage as authors of [toyama99] and fuse previous segmentations and motion information to locate pixels that may be recovered as foreground. In our research, we used however a colour list for each region instead of building a histogram of grey tones as in [toyama99].

The frame difference between frames $F(i)$ and $F(i-1)$ is computed as,

$$d^1_{x,y}(i) = |F_{x,y}(i) - F_{x,y}(i-1)|,\ \forall F_{x,y}(i) \tag{6.49}$$

With these distances a binary image $S^1(i)$ is obtained by applying equation 6.48 to $d^1_{x,y}(i)$. Then, the intersection of adjacent pairs of differenced images and the previous foreground image is computed as,

$$P = S^1(i) \wedge S^1(i-1) \wedge S(i-1) \tag{6.50}$$

Image P contains the coordinates of the seeds that can be used to recover foreground pixels erroneously classified. Connected regions R_i of P with less than p_{min} pixels are discarded. Colours c_j of each R_i are discovered and stored in a set C_i. Two colours c_p and c_q such that $c_p, c_q \in R_i$ are considered to be the same if their euclidean distance is less than a constant c_{min}.

For each R_i in P, we check the values of each pixel $(x, y) \in R_i$ in frame $F(i)$ and compute a binary image $L_{x,y}(i)$ as follows,

$$L_{x,y}(i) = \begin{cases} 1,\ if\ \exists c \in C_i\ /\ dist(c, F_{x,y}(i)) \leq c_{min} \\ 0,\ otherwise \end{cases} \tag{6.51}$$

being $dist(c, F_{x,y}(i))$ the distance between colour c and colour of pixel $F_{x,y}(i)$. Image $L_{x,y}(i)$ contains all the pixels considered initially as background whose colour is similar enough to neighbouring pixels which were considered foreground, and thus, are labelled as foreground as well.

Binary image $S(i)$ is grown by performing the following operation,

$$S^e(i) = S(i) \vee L(i) \tag{6.52}$$

where $S^e(i)$ is a binary image in which foreground pixels are labelled with 1 and background pixels are labelled with 0.

6.5.5 Model corruption detection

Few methods exist in the literature to detect background model corruption. In [toyama99], finding an unusual percentage of foreground pixels is considered to be enough as to decide that the model is no longer valid.

In our approach, model corruption is detected by testing if the constraint $d_0 > d_{corrupt}$ is met. This means that a small amount of pixels yield small distances to the background values, breaking the assumption that most pixels should be similar to the background model.

In the case of background model corruption, the process restarts and keeps only one model per pixel, as made in section 6.4.2. It sets

$$B_{x,y}(0) = F_{x,y}(i),\ c^1_{x,y} = max(c^v_{x,y}) \tag{6.53}$$

The difference with the usual approach is that the confidence $c_{x,y}$ of the new model is inherited from previous frames. In our opinion, this allows the system to keep the knowledge about the scene dynamics, but does not prevents it from changing the values of the model, if needed. Observing the differences between very different images, the limit $d_{corrupt} = 10$ seems to be safe.

6.5.6 Model update

In this section, the process of background modelling using functions MF and MB is explained. The model is initialized without assumptions about the observed scene and is updated in a frame by frame basis.

The algorithm takes as input a sequence of frames $F(0), F(1), ..., F(n)$ and starts by setting,

$$\forall x, y \ , \ B^1_{x,y}(0) = F_{x,y}(0), \ c^1_{x,y}(0) = 0.01 \tag{6.54}$$

where $B^1_{x,y}(0)$ corresponds to the first model of pixel (x, y) in time 0 and $c^1_{x,y}(0)$ is the confidence value of the first model. This confidence value measures how good the model describes the pixel.

Two parameters are set by user, K the maximum amount of models allowed per pixel, and γ a value in the range $[0, 1]$ that controls the speed at which models of a pixel are replaced or added, the higher the value, the faster models are replaced.

After each background subtraction and foreground recovery, the model is updated depending on information of $S^f(i)$. Model v for pixel $b_{x,y}$ matches the input $p_{x,y} \in F(i)$, denoted by v_{match}, if $S^f_{x,y} = 0$ and $d^{v_{match}}_{x,y} = min(d^v_{x,y}), \ \forall v \neq v_{match}$.

In a similar way as in sections 6.2.4 and 6.4.3, for every pixel that verifies that $S^f_{x,y} = 0$, description and confidence of model v_{match} are updated as follows,

$$B^{v_{match}}_{x,y}(i) = \alpha \cdot B^{v_{match}}_{x,y}(i-1) + (1-\alpha) F_{x,y}(i) \tag{6.55}$$

$$c^{v_{match}}_{x,y}(i) = \alpha \cdot c^{v_{match}}_{x,y}(i-1) + (1-\alpha) \cdot MB(d^{v_{match}}_{x,y}) \tag{6.56}$$

The remaining models $v \neq v_{match}$, for pixel $b_{x,y}$ also update their confidences as,

$$c^v_{x,y}(i) = \alpha \cdot c^v_{x,y}(i-1) \tag{6.57}$$

where α is a learning rate factor in the range $[0, 1]$ and $B^v_{x,y}(i-1)$ is the value in time $i-1$ of v-th model of pixel at x, y.

The models of each pixel are ordered in descending order according to their confidences. If the sum of the confidences verifies that,

$$\gamma > \sum_{v \leq K} c^v_{x,y}(i) \tag{6.58}$$

then a new model will be added or the K-th model be replaced. For the replaced model, the algorithm sets $B^K_{x,y}(i) = F_{x,y}(i)$ and $c^K_{x,y}(i) = 0.01$. Algorithm 5 shows this process algorithmically.

Algorithm 5 The process of background modelling. The function $matchedModel(x, y)$ returns v_{match}, the index of the model for pixel $b_{x,y}$ that matched the input. On the other hand, the function $chooseModelToUpdate(x, y)$ returns the index of the model chosen to be updated for pixel $b_{x,y}$ after ordering them.

$\forall\, p_{x,y}$
if $S^f_{x,y} == 0$ **then**
  $c^v_{x,y}(i) \leftarrow \alpha c^v_{x,y}(i-1),\ \forall v$
else
  $v_{match} \leftarrow matchedModel(x, y)$
  $B^{v_{match}}_{x,y}(i) \leftarrow \alpha B^{v_{match}}_{x,y}(i-1) + (1-\alpha)F_{x,y}(i)$
  $c^{v_{match}}_{x,y}(i) \leftarrow \alpha c^{v_{match}}_{x,y}(i-1) + (1-\alpha)MB(d^{v_{match}}_{x,y})$
  $c^v_{x,y}(i) \leftarrow \alpha c^v_{x,y}(i-1); \forall v \neq v_{match}$
end if
if $\gamma > \sum_{v \leq K} c^v_{x,y}(i)$ **then**
  $v_{last} \leftarrow chooseLastModel(x, y)$
  $B^{v_{last}}_{x,y}(i) \leftarrow F_{x,y}(i)$
  $c^{v_{last}}_{x,y}(i) \leftarrow 0.01$
end if

6.6 Experiments

Several experiments were performed in order to estimate the best set of parameters for the algorithms introduced and compare their results with other well-known algorithms found in the literature. In the experiments, we compared the true negatives TN that is, the amount of pixels classified as background by the algorithm which are truly background and the true positives TP, the amount of pixels classified as foreground by the algorithm which are truly foreground. False positives (FP) and false negatives (FN) can be inferred by subtracting previous values from 1, respectively. This measure served to compare the results of algorithms. Quantitative and qualitative results were obtained by performing experiments using the Wallflower benchmark [toyama99], which seems to be accepted by the scientific community as a common validity test for background modelling algorithms.

We compared the performance of several well-known algorithms proposed in the literature with the techniques introduced in this chapter using the Wallflower benchmark. The algorithms are introduced in section 3.3, a brief summarize follows:

- **Frame Difference (FD)** : Each image is subtracted from the previous image in the sequence. Absolute differences greater than a threshold are marked as foreground.
- **Mean and Threshold (MT)** : Pixel-wise mean values are computed during a training phase, and pixels within a fixed threshold of the mean are considered background.
- **Mixture of Gaussians (MoG)** [grimson99]: A pixel-wise mixture of 3 Gaussians models the background. Each Gaussian is weighted according to the frequency with which it explains the observed background. The most heavily weighted Gaussians that together explain over 50% of past data are considered background. Two different versions of the algorithm were tested, the grey tone and the RGB versions.

- **Textures (LBP)** [heikkila06]: textures in the neighbourhood are modelled by means of LBP histograms. Several histograms may be assigned to each pixel, in order to consider several models.

- **Wallflower (WLL)** [toyama99]: a combination of three different processing layers which operate at different levels, pixel, region and frame.

The algorithms introduced in this chapter are:

- **Background Adaptive with Confidence (BAC)** [rosell08]: motion and background subtraction are considered in order to segment background pixels from foreground pixels. A confidence value is used for each model. Two extensions of this algorithm are also tested BACc, which is an extensionof BAC to colour coordinates and MBAC, a multimodal extension of BAC.

- **Fuzzy background subtraction (FBS)** [rosell10b]: a background membership function is computed taking into account the distances at which pixels of incoming frame lay from the background. This function permits the computation of a segmentation threshold.

Table 6.2 summarizes the differences and similitudes between the compared algorithms. In each row, the most representative features of each algorithm are shown. For instance, frame difference is not adaptive, does not need a training period to start working, does not offer multimodal support, thus in the *multimodal support* column a dash appears in the corresponding cell, and uses a user settable segmentation threshold.

In the case of the mixture of gaussians, the threshold is established by their authors as 2.5 times the variance of the background model. In our opinion, this prevents it from being classified as a manual threshold, however, it is also true that the threshold is not computed directly from data, so it is classified as semi-automatic.

Needing a training period or not, makes a difference if the algorithm is expected to work under any kind of circumstances. For instance, the temporal median filter algorithm needs a set of frames in order to start processing new frames, otherwise, it cannot build a reliable model. The Wallflower algorithm also needs to perform intensive and time consuming computations before starting its normal operation.

On the other side, the other algorithms discussed in this section assume an initial frame as a starting model and update it over time.

Algorithm.	Adaptive	Training period	Multimodal support	threshold computation
Frame difference	N	N	-	Manual
Mean and threshold	Y	Y	N	Manual
Mixture of Gaussians	Y	N	Y	Semi-automatic
LBP	Y	N	Y	Manual
Wallflower	Y	Y	Y	Automatic
BAC	Y	N	N	Manual
BACc	Y	N	N	Manual
MBAC	Y	N	Y	Manual
FBS	Y	N	Y	Automatic

Table 6.2: Summarize of the features of the different compared background modelling techniques.

6.6.1 The Wallflower benchmark

We used the Wallflower benchmark in our experiments with the aim of comparing the performance of the techniques listed in table 6.2. It has the advantage that it seems to be established as a de facto benchmark for background modelling algorithm's comparison.

This benchmark is composed of seven sequences in which different challenging situations are represented, each situation is motivated by a different problem, foreground aperture, sudden changes of illumination, constant motion, background clutter and background adaptation over time.

Authors of the benchmark took one sequence of images to represent each of first seven problems listed in section 3.3. Each test sequence begins with at least 200 background frames for training the algorithms, except for the bootstrap sequence (the first 200 frames of the bootstrap sequence were nevertheless used for training algorithms that require a training phase). Also, each sequence has a control frame, used to test the performance of the background modelling algorithm, this control frame corresponds to a manual segmentation of the scene. The sequences are:

- 1. Bootstrapping: The sequence consists of several minutes of an overhead view of a cafeteria. There is constant motion, and every frame contains people.
- 2. Camouflage: A monitor sits on a desk with rolling interference bars. A person walks into the scene and occludes the monitor.
- 3. Foreground Aperture: A person is asleep at his desk, viewed from the back. He wakes up and slowly begins to move. His shirt is uniformly coloured.
- 4. Light Switch: First, there is a training period in which the camera sees a room with the lights both on and off. Then, during the test sequence, the room starts with the lights off. After a few minutes a person walks in, turns on the light, and moves a chair. We stop the algorithms shortly after the light comes on, leaving the various pixel-level adaptation mechanisms inadequate time to adapt. In addition to the person, the recently moved chair is considered foreground.
- 5. Moved Object: A person walks into a conference room, makes a telephone call, and leaves with the phone and a chair in a different position. The background is evaluated 50 frames after the person leaves the scene, giving those algorithms that adapt time to do so.
- 6. Time of Day: The sequence shows a darkened room that gradually becomes brighter over a period of several minutes. A person walks in and sits on a couch.
- 7. Waving Trees: A person walks in front a swaying tree.

6.6.2 Analysis of BAC and MBAC parameters

In this section, we evaluate the parameters of BAC and MBAC algorithms before comparing their results to other algorithms. With these experiments the impact of parameters on results can be discussed. The parameters studied, together with the reference of the equation where they are employed, are listed in table 6.3. Due to the vast amount of combinations of values, only the most significant experiments and results are discussed in the following paragraphs.

In equation 6.1, the parameter κ determines how strict or loose the comparison with the background will be, in order to determine the most suitable value for this parameter.

equation.	*Expression*	*Parameter*
Eq. 6.1	$S(p,q) = e^{\frac{-\lvert p-q \rvert}{\kappa}}$	κ
Eq. 6.9	$fSet = \{p_{x,y} \in F(i) : P_{fore}(p_{x,y}) > \tau, being\ 1 > \tau \geq P_{back}(p_{x,y})\}$	τ
Eq. 6.36	$\gamma > \sum_{v \leq K} c^{v}_{x,y}(i)$	γ

Table 6.3: Description of parameters influencing BAC and MBAC execution.

Tables 6.4 and 6.5 show the results obtained by varying the values of κ in equation 6.1 and its impact depending on the value of τ used. In general, the lower the value of κ the lower the difference between two pixels should be, in order to be considered similar. This effect is specially significant in sequence *wavingTrees*, in which the percentage of TN improves as the value of κ increases, being better in the case of $\tau = 0.6$. As expected, for $\kappa = 20$ the values of TN are higher than the others; this is explained by the fact that as κ increases, pixels must be more separated to be considered different.

	BAC							
$\tau = 0.4$	$\kappa = 5$		$\kappa = 10$		$\kappa = 15$		$\kappa = 20$	
Seq.	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*
bootstrap	0.87	0.43	0.59	0.92	0.55	0.94	0.50	0.95
camouflage	0.74	0.74	0.77	0.90	0.72	0.90	0.70	0.93
foregroundAperture	0.90	0.60	0.49	0.98	0.24	0.99	0.20	0.99
lightSwitch	0.82	0.15	0.30	0.86	0.48	0.90	0.47	0.91
movedObject	-	0.97	-	1.00	-	1.00	-	1.00
timeOfDay	0.83	0.77	0.42	0.98	0.35	0.98	0.30	0.98
wavingTrees	0.96	0.32	0.86	0.67	0.81	0.73	0.75	0.79

Table 6.4: Rate of TP and TN obtained for the Wallflower benchmark using equation 6.9 to detect foreground regions with $\tau = 0.4$. Dashed results mean that no foreground pixels were labelled in the control image.

	BAC							
$\tau = 0.6$	$\kappa = 5$		$\kappa = 10$		$\kappa = 15$		$\kappa = 20$	
Seq.	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*
bootstrap	0.67	0.82	0.55	0.93	0.48	0.96	0.41	0.97
camouflage	0.40	0.89	0.15	0.95	0.72	0.91	0.70	0.92
foregroundAperture	0.72	0.30	0.53	0.80	0.49	0.90	0.47	0.99
lightSwitch	0.43	0.98	0.33	0.97	0.26	0.98	0.21	0.98
movedObject	-	1.00	-	1.00	-	1.00	-	1.00
timeOfDay	0.70	0.95	0.48	0.97	0.37	0.98	0.31	0.98
wavingTrees	0.91	0.56	0.86	0.68	0.80	0.76	0.74	0.80

Table 6.5: Rate of TP and TN obtained for the Wallflower benchmark using equation 6.9 to detect foreground regions with $\tau = 0.6$. Dashed results mean that no foreground pixels were labelled in the control image.

The parameter τ is intimately linked to the value of κ in equation 6.1. Different slopes for the curve represented by equation 6.1, will yield different values which are then filtered depending on τ. This means that the higher the value of τ the more restrictive the segmentation is. We considered that best parameters are those which offer a good balance between values of TP and TN, giving more importance to the TN values than to the TP values. Using the Wallflower benchmark offered us the possibility of testing the behaviour of the algorithm facing a rich set of situations with different parameters. Thus, the parameter values $\kappa = 5$ and $\tau = 0.6$ seem to be the best option according to the obtained results, values of $TN \geq 0.8$ in the Wallflower sequences which has no clutter in the background nor sudden changes of the scene, in these se-

quences, we obtained a rate $TP \geq 0.6$.

The value of τ is also linked to the value of γ in the case of the MBAC algorithm. In some cases, it may seem that some results do not follow the expected improvement or deterioration of rates, the cause of this effect is the corrupt model detection, that in some cases forces a new model to be build and distorts results.

Tables 6.6, 6.7, 6.8 and 6.9 show the results of varying the the values of τ in equation 6.31 for $\gamma = 0.4$ and $\gamma = 0.6$ and $\kappa = 5$. On the other side, tables 6.10, 6.11, 6.12 and 6.13 show the results for the configuration $\gamma = 0.4$ and $\gamma = 0.6$ and $\kappa = 10$ with different values of τ. Comparing the values represented in tables 6.6, 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13 6.14, 6.15, 6.16 and 6.17, it can be seen that in the case of the MBAC algorithm, the value of κ does not influence largely the obtained results. In some sense, this is not an unexpected result, as long as having several models permits the algorithm to respond to different background configurations.

$\kappa = 5$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.66	0.62	0.59	0.53	0.48	0.38	0.30
camouflage	0.31	0.12	0.73	0.73	0.70	0.70	0.69
foregroundAperture	0.58	0.48	0.47	0.48	0.46	0.47	0.45
lightSwitch	0.49	0.44	0.36	0.27	0.20	0.63	0.56
movedObject	-	-	-	-	-	-	-
timeOfDay	0.64	0.51	0.43	0.32	0.29	0.27	0.25
wavingTrees	0.95	0.94	0.87	0.79	0.69	0.56	0.45

Table 6.6: Rate of TP obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 5$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 5$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.84	0.89	0.91	0.92	0.96	0.97	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.88	0.90	0.92	0.92	0.93	0.92
lightSwitch	0.97	0.97	0.97	0.99	0.99	0.16	0.21
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.93	0.95	0.97	0.97	0.97	0.97	0.97
wavingTrees	0.43	0.49	0.59	0.72	0.79	0.86	0.92

Table 6.7: Rate of TN obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 5$. Dashed results indicate that no foreground pixels were labelled in the control image.

Tables 6.10, 6.11, 6.12 and 6.13 show the results of varying the the values of τ in equation 6.31 for $\gamma = 0.4$ and $\gamma = 0.6$ and $\kappa = 10$. It may be seen, that increasing the value of τ yields a smaller rate of TP, on the other hand, it has the effect of achieving a higher rate of TN. Tables 6.14, 6.15, 6.16 and 6.17 show the results of varying the the values of τ in equation 6.31 for $\gamma = 0.4$ and $\gamma = 0.6$ and $\kappa = 15$. As wish previous combinations of parameters, increasing the value of τ yields a smaller rate of TP, on the other hand, it has the effect of achieving a higher rate of TN. But results seem to be worse, specially for TP, as κ increases.

$\kappa = 5$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.62	0.55	0.50	0.44	0.37	0.31	0.23
camouflage	0.31	0.12	0.74	0.71	0.71	0.69	0.68
foregroundAperture	0.58	0.48	0.47	0.46	0.46	0.47	0.45
lightSwitch	0.49	0.44	0.36	0.27	0.20	0.63	0.56
movedObject	-	-	-	-		-	-
timeOfDay	0.64	0.52	0.43	0.32	0.29	0.29	0.25
wavingTrees	0.95	0.92	0.87	0.77	0.66	0.51	0.37

Table 6.8: Rate of TP obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 5$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 5$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.87	0.93	0.94	0.97	0.98	0.99	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.88	0.90	0.90	0.92	0.93	0.92
lightSwitch	0.97	0.97	0.97	0.97	0.99	0.18	0.23
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.93	0.94	0.97	0.97	0.97	0.97	0.97
wavingTrees	0.45	0.51	0.60	0.73	0.81	0.85	0.93

Table 6.9: Rate of TN obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 5$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 10$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.67	0.69	0.63	0.61	0.54	0.47	0.35
camouflage	0.31	0.11	0.74	0.73	0.71	0.70	0.69
foregroundAperture	0.58	0.60	0.56	0.53	0.48	0.47	0.47
lightSwitch	0.50	0.45	0.37	0.28	0.22	0.17	0.37
movedObject	-	-	-	-	-	-	-
timeOfDay	0.65	0.57	0.47	0.36	0.31	0.29	0.28
wavingTrees	0.96	0.94	0.88	0.79	0.68	0.56	0.44

Table 6.10: Rate of TP obtained applying MBAC for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 10$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 10$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.84	0.83	0.88	0.92	0.95	0.97	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.57	0.74	0.84	0.90	0.92	0.93
lightSwitch	0.97	0.97	0.98	0.98	0.99	0.99	0.18
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.94	0.95	0.96	0.98	0.98	0.98	0.98
wavingTrees	0.44	0.50	0.55	0.67	0.73	0.79	0.86

Table 6.11: Rate of TN obtained applying MBAC for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 10$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 10$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.63	0.62	0.59	0.54	0.48	0.39	0.30
camouflage	0.31	0.13	0.73	0.73	0.70	0.72	0.69
foregroundAperture	0.58	0.49	0.48	0.48	0.47	0.47	0.46
lightSwitch	0.50	0.44	0.36	0.27	0.20	0.63	0.55
movedObject	-	-	-	-	-	-	-
timeOfDay	0.65	0.51	0.43	0.32	0.30	0.28	0.26
wavingTrees	0.96	0.94	0.88	0.79	0.69	0.58	0.46

Table 6.12: Rate of TP obtained applying MBAC to the Wallflower benchmark depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 10$. Dashed results mean that no foreground pixels were labelled in the control image.

$\kappa = 10$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.88	0.89	0.91	0.92	0.96	0.97	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.88	0.90	0.90	0.92	0.93	0.93
lightSwitch	0.97	0.97	0.97	0.97	0.99	0.17	0.22
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.94	0.95	0.96	0.98	0.98	0.98	0.98
wavingTrees	0.45	0.50	0.59	0.73	0.79	0.86	0.92

Table 6.13: Rate of TN obtained applying MBAC to the Wallflower benchmark depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 10$. Dashed results mean that no foreground pixels were labelled in the control image.

$\kappa = 15$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.66	0.62	0.59	0.53	0.48	0.38	0.30
camouflage	0.31	0.12	0.73	0.73	0.70	0.70	0.69
foregroundAperture	0.58	0.48	0.47	0.48	0.46	0.47	0.46
lightSwitch	0.50	0.44	0.36	0.27	0.20	0.63	0.56
movedObject	-	-	-	-	-	-	-
timeOfDay	0.64	0.52	0.43	0.32	0.29	0.27	0.25
wavingTrees	0.95	0.94	0.87	0.79	0.69	0.57	0.45

Table 6.14: Rate of TP obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 15$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 15$	*MBAC*						
$\gamma = 0.4$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.84	0.89	0.91	0.92	0.96	0.97	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.88	0.90	0.90	0.92	0.93	0.92
lightSwitch	0.97	0.97	0.97	0.99	0.99	0.16	0.21
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.93	0.94	0.97	0.97	0.97	0.97	0.97
wavingTrees	0.43	0.49	0.59	0.72	0.79	0.86	0.92

Table 6.15: Rate of TN obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.4$ and $\kappa = 15$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 15$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*	*TP*
bootstrap	0.62	0.55	0.50	0.44	0.37	0.31	0.23
camouflage	0.31	0.12	0.74	0.71	0.71	0.69	0.67
foregroundAperture	0.58	0.48	0.47	0.48	0.46	0.47	0.46
lightSwitch	0.49	0.44	0.36	0.27	0.20	0.63	0.56
movedObject	-	-	-	-	-	-	-
timeOfDay	0.65	0.52	0.43	0.32	0.29	0.29	0.25
wavingTrees	0.95	0.92	0.87	0.77	0.66	0.51	0.37

Table 6.16: Rate of TP obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 15$. Dashed results indicate that no foreground pixels were labelled in the control image.

$\kappa = 15$	*MBAC*						
$\gamma = 0.6$	$\tau = 0.3$	$\tau = 0.4$	$\tau = 0.5$	$\tau = 0.6$	$\tau = 0.7$	$\tau = 0.8$	$\tau = 0.9$
Sequence	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*	*TN*
bootstrap	0.87	0.93	0.94	0.95	0.98	0.99	0.99
camouflage	0.94	0.95	0.86	0.90	0.91	0.93	0.94
foregroundAperture	0.80	0.88	0.90	0.90	0.92	0.93	0.92
lightSwitch	0.97	0.97	0.97	0.99	0.99	0.18	0.23
movedObject	1.00	1.00	1.00	1.00	1.00	1.00	1.00
timeOfDay	0.94	0.94	0.97	0.97	0.97	0.97	0.97
wavingTrees	0.45	0.51	0.60	0.73	0.81	0.85	0.93

Table 6.17: Rate of TN obtained applying the MBAC algorithm for the Wallflower benchmark, depending on the value assigned to the minimum foreground probability with $\gamma = 0.6$ and $\kappa = 15$. Dashed results indicate that no foreground pixels were labelled in the control image.

In general, it can be seen that the MBAC algorithm improves slightly results of BAC, specially in the sequences in which using several models is an advantage, for instance, when there is clutter in the background. If the values $\kappa = 5$, $\tau = 0.6$ and $\gamma = 0.4$ are selected, the TN rate is over 0.70 in all cases in tables 6.7 and 6.6 . On the other hand, it is difficult finding a combination of parameters for MBAC which yield values of TP over 0.8 in all cases. The use of an exponential to detect differences between background and foreground together with multiple values per pixel has the disadvantage that several models can represent tight grey tone values, whose difference is amplified by the exponential. Also, using motion to model the background has an impact in the way models are swaped. This issue produces that the MBAC algorithm does not reach a higher rate of TN in the *wavingTrees* sequence.

6.6.3 Analysis of FBS parameters

The parameter that seems to have the biggest influences in the FBS algorithm is $Card_0$, which measures which pixels will be considered undoubtedly background, as shown in line 5 of algorithm 4. Table 6.18 shows the experiments performed in order to detect the influence of this parameter in the segmentation results. As it can be seen, the bigger the value, the worse the results, although the worsen is sometimes mimicked by the availability of several models per pixel.

However, differences are usually quite small. This may be due to several facts, as the following:

- The cumulative effect of having several models which may eventually be dropped

off and replaced with others.

- The evidence that for some frames, the real value of $Card_0$ could be set to values greater than 0.5.

	FBS											
$Card_0$	0.1		0.2		0.3		0.4		0.5		0.6	
Sequence	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*
bootstrap	0.54	0.97	0.54	0.97	0.55	0.96	0.56	0.96	0.54	0.95	0.55	0.96
camouflage	0.75	0.90	0.74	0.92	0.73	0.92	0.74	0.92	0.28	0.95	0.06	0.95
foregroundAp.	0.59	0.90	0.61	0.91	0.58	0.91	0.60	0.90	0.59	0.91	0.62	0.91
lightSwitch	0.63	0.86	0.68	0.86	0.66	0.85	0.76	0.79	0.80	0.84	0.79	0.85
movedObj.	-	1.00	-	1.00	-	1.00	-	1.00	-	1.00	-	1.00
time.	0.70	0.97	0.72	0.96	0.72	0.96	0.71	0.96	0.69	0.96	0.74	0.97
wavingTr.	0.85	0.88	0.86	0.87	0.85	0.88	0.89	0.86	0.90	0.85	0.37	0.79

Table 6.18: Evaluation of parameter $Card_0$ by computing the rate of TP and TN obtained ranging the value of $Card_0$ in $[0.1, 0.6]$ the FBS algorithm. Note that there is an slight decrease in the percentages of TP, for $Card_0 \geq 0.3$.

Empirical results, as those represented in the figures 6.7, 6.8, 6.9, 6.10 and 6.11, induce to think that a low value is desirable. If line 5 in algorithm 4 is considered with attention, the inequality searches for all the values which are under a given limit, looking for a, presumably, small set. If this set was expected to be larger, maybe it would affect negatively the slope of the curve thus, changing the location of the threshold. This fact, together with the empirical results aforementioned and the results obtained in table 6.18, it was decided finally to set $Card_0 = 0.3$.

6.6.4 The FBS algorithm versus the BAC, MBAC algorithms

For a quantitative comparison of results of these algorithms, we used the TP and TN for each control image of the benchmark.

BAC and BACc were executed taking $\tau = 0.6$ and $\kappa = 5$. For MBAC, a value $K = 3$ and $\tau = 0.8$, $\gamma = 0.4$.

For fuzzy background subtraction, parameter's values are $K = 3$, $p_{min} = 8$ as suggested in [toyama99], $\beta_0 = 1°$, $c_{min} = 7$ and $\gamma = 0.6$. Only in the sequence *camouflage* a value $\gamma = 0.4$ was chosen.

	BAC		BACc		MBAC		FBS	
Sequence	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*	*TP*	*TN*
bootstrap	0.67	0.90	0.68	0.89	0.59	0.91	0.56	0.96
camouflage	0.72	0.76	0.75	0.69	0.73	0.86	0.74	0.92
foreAperture	0.40	0.87	0.43	0.96	0.47	0.90	0.67	0.83
lightSwitch	0.43	0.90	0.54	0.85	0.41	0.36	0.97	0.86
movedObject	-	1.00	-	0.95	-	1.00	-	1.00
timeOfDay	0.70	0.95	0.42	0.74	0.43	0.97	0.72	0.88
wavingTrees	0.91	0.56	0.93	0.57	0.87	0.59	0.86	0.88

Table 6.19: Results of the compared algorithms. Each row in the table corresponds to an algorithm and shows the TP and TN achieved for a given sequence. The numbers of the sequences, correspond to the numbers in the list in section 6.6.1.

In table 6.19, the rates achieved in the Wallflower benchmark of the four algorithms introduced in this section is shown. BAC has a poor performance if there is motion in the background, as it is not able to respond to different values of the background as may be seen in the sequence *wavingTrees*.

BACc seems to improve slightly results considering the TP values in nearly all sequences. However, for sequence *timeOfDay* the similarity function seems to perform poorly in the RGB case yielding a very low rate of TP, lower than that obtained for BAC. This may be due to the fact that the RGB space is no uniformly distributed and, in some cases, some distances yield wrong probability values.

MBAC improves results over BAC and BACc, specially in the case of clutter in the background as in the sequence *wavingTrees*. But using motion to detect regions of interest at the same time that several models are handled does not produce the expected results, as pointed out in section 6.6.2. The use of motion to detect regions of interest, for instance, produces poor results in the *lightSwitch*.

It can be seen that, in all cases, the FBS algorithm outperforms the others if we consider the balance of TP and TN rates. Due to its ability to manage several models per pixel, it is quite obvious its advantage over the BAC or BACc algorithms, which just consider one model per pixel. On the other side, the threshold computation of the FBS algorithm is more accurate than the one computed by the MBAC algorithm.

6.6.5 Our proposal compared to the literature algorithms

Six algorithms, or versions of algorithms, were compared: frame difference, temporal median filter, textures, mixture of gaussians, wallflower and FBS.

For frame difference and the temporal median filter algorithms a segmentation threshold equal to 10 was used.

As an example of the mixture of Gaussians, the Stauffer's algorithm was chosen. It was executed with $K = 5$ and $T = 0.6$.

The LBP algorithm was executed with the suggestions found in [heikkila06], $R = 6$, $P = 2$, $T_B = 0.8$, $T_P = 0.7$ and $\alpha_b = \alpha_w = 0.01$.

Analogously, for the Wallflower algorithm the parameters were set as the authors propose in their paper, 30 coefficients for the Wiener filter and 50 past values.

FBS was executed as discussed in section 6.6.4. For all algorithms, we used $\alpha = 0.99$.

The comparison of algorithms is made from the point of view of the results achieved with the Wallflower benchmark and also, taking into account their time and spatial cost. Table 6.20 shows the global results for each algorithm with the complete benchmark. Qualitative results of the compared algorithms are shown in figure 6.13. In this figure, on the left column, we show the hand segmented versions of the control frames. On the right, the result obtained by fuzzy background subtraction. The fact that no unexpected noises appear on result frames and that objects seem to be segmented quite accurately, makes us think that these results are quite promising.

The LBP algorithm obtains TN rates over 0.8 in all cases but in the *lightSwitch*, because it is not able to manage sudden corruptions of the background model. For this challenge, the TN rate falls to 0.7.

The comparison between the Stauffer's approach using grey tones and RGB coordinates shows that adding colour to the processing improves slightly results. However, the time penalty has to be considered. In both cases, the approach fails when sudden changes in the background occur, as in the *lightSwitch* sequence with $TP = 0.05$ and $TP = 0.07$ respectively. This happens mainly because the algorithm takes a lot of time to react to changes in pixel's values, moreover, if the pixel has several different descriptions modelling it.

Frame difference (FD), as expected, has a good behaviour detecting moving regions, but fails when objects stay still. Also, It usually fails to detect the complete

Algorithm		*boots.*	*camouf.*	*foreApert.*	*lightSw.*	*moved.*	*timeOfDay*	*wav.*
LBP	TP	0.71	0.73	0.79	0.62	-	0.86	0.85
	TN	0.90	0.86	0.66	0.91	1.00	0.97	0.77
MoG	TP	0.44	0.92	0.54	0.75	-	0.46	0.75
	TN	0.91	0.67	0.74	0.05	1.00	0.99	0.86
MoGs RGB	TP	0.44	0.92	0.50	0.73	-	0.41	0.86
	TN	0.95	0.73	0.85	0.07	1.00	0.98	0.90
FD	TP	0.48	0.20	0.25	0.29	–	0.25	0.70
	TN	0.95	0.96	0.94	0.98	1.00	0.98	0.62
MT	TP	0.79	0.97	0.60	0.98	–	0.52	0.91
	TN	0.81	0.28	0.52	0.06	1.00	0.88	0.56
WLL	TP	0.77	0.97	0.99	0.70	–	0.79	0.85
	TN	0.99	0.98	0.90	0.98	1	0.79	0.85
FBS	TP	0.56	0.74	0.67	0.60	-	0.72	0.86
	TN	0.96	0.92	0.83	0.86	1.00	0.88	0.88

Table 6.20: Results of the compared algorithms. Each row in the table corresponds to an algorithm and shows the TP and TN achieved for a given sequence.

area of objects and just detects their edges. Its quantitative results for the Wallflower benchmark can be seen in table 6.20, for all the sequences the value of TN is over 0.90 except in the *wavingTrees* sequence, in which the clutter in the background is quite significant. However, the values of TP do not reach the 80% of the total amount of foreground pixels.

Mean and threshold (MT) algorithm yield very poor results when it has to deal with clutter in the background, due to the fact that it only stores a model of the scene. This problem is evident in the *wavingTrees* or *camouflage* sequences. Sudden background corruption is not easily dealt by these algorithms, as shown in the results obtained in the sequence *lightSwitch* with $TN = 0.06$. In the rest of cases, the TN rate is always under 0.8.

Wallflower algorithm has the best performance in all sequences, achieves the highest rate of TP and TN. It effectively responds to all the challenges proposed by the benchmark. However, as will be discussed later, this efficiency is obtained at the cost of an enormous temporal and spatial cost. It can be seen that the rate of TN is in all cases but one over 0.85, and also the rate of TP is over 0.7, a result which is not obtained by any other algorithm.

Finally, the FBS algorithm also achieves TN rates over 0.83 for all sequences, and handles appropriately the challenge in the *lightSwitch* challenge, achieving a TP similar to that obtained by the LBP algorithm.

In general, the values of TP are over those obtained by the mixture of Gaussians and a little bit worse than those of the textures algorithm or the Wallflower algorithm.

In general, results point out that adaptive, multimodal algorithms handle suitably all situations. Specifically, the Wallflower algorithm is the one with best overall results, followed by the LBP algorithm, FBS algorithm and the Stauffers' approaches.

6.6.6 Temporal analysis of algorithms

In this section we introduce an analysis of the time consumption of each algorithm in order to process a complete sequence of the Wallflower algorithm. Average time responses of each algorithm are shown in table 6.21. In this table, the time invested in segmentation is separated from the time invested in background adaptation.

The segmentation time includes time spent on any operation which involves subtracting, comparing values or extending areas, as in the region stage of the Wallflower algorithm and the fuzzy background subtraction algorithm. On the other hand, the

Algorithm		*boots.*	*camouf.*	*foreApert.*	*lightSw.*	*moved.*	*timeOfDay*	*wav.*
LBP	Segm.	33.6	26.89	21.26	22.28	20.32	31.87	23.16
	Model	5.41	3.97	3.22	3.12	3.00	5.00	3.33
MoG	Segm.	0.08	0.05	0.05	0.05	0.05	0.04	0.05
	Model	5.13	3.71	3.43	3.45	3.44	2.73	3.35
MoG RGB	Segment.	0.08	0.05	0.05	0.05	0.05	0.05	0.06
	Model	5.13	3.72	3.42	3.45	3.44	3.52	4.23
FD	Segm.	3.5E-4	3.4E-4	2.8E-4	2.8E-4	2.9E-4	2.5E-4	4.2E-4
	Model	0	0	0	0	0	0	0
MT	Segm.	5.5E-4	4.7E-4	4.4E-4	4.2E-4	4.1E-4	2.8E-4	5.9E-4
	Model	0.72	0.71	0.78	0.74	0.79	0.79	0.70
BAC	Segment.	0.73	0.48	0.36	0.73	–	0.44	0.51
	Model	0.86	0.96	0.99	0.74	0.99	0.98	0.95
BACc	Segment.	0.67	0.78	0.58	0.46	–	0.53	0.93
	Model	0.85	0.70	0.87	0.97	0.99	0.98	0.59
MBAC	Segment.	0.56	0.49	0.40	0.39	0.50	0.33	0.55
	Model	0.48	1.74	2.19	2.69	2.48	3.02	1.11
WLL	Segment.	0.35	0.34	0.34	0.34	0.34	0.34	0.34
	Model	55.5	55.5	55.5	55.5	55.5	55.5	55.5
FBS	Segment.	0.8	0.2	0.7	0.1	0.1	0.1	0.6
	Model	2.5	2.8	2.8	2.2	2.0	2.0	2.7

Table 6.21: Mean time for frame segmentation and background modeling of each of the studied algorithms.

background adaptation time includes the operations of creating, updating or removing background descriptions.

LBP algorithm and wallflower algorithms have a very high temporal cost if not parallelized; however, experiments were made from the point of view of the constraints stated in section 5.1.1. These were the approaches with better segmentation results, but at the expense of a bigger amount of time per frame in average.

FBS requires more time in average to segment a single frame than Stauffer's approach, however its model update time is lower. BAC and MBAC have also a slow segmentation, influenced probably by the exponential used in the segmentation.

6.6.7 Space complexity of the algorithms

To analyse an algorithm is to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity). Algorithm analysis is an important part of the computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm, as the complexity function for an arbitrarily large input. O notation, ω notation and θ notation are used to this end. Space complexity of algorithms is of crucial importance if the algorithm is designed to run in environments in which tasks will be given a small amount of memory to work with. In this section, the complexity analysis of the algorithms considered in the experiments is performed from the point of view of their space consumption.

In order to compare the space complexity of the algorithms tested in our experiments, we supposed an image of size $n \cdot m$. Table 6.22 shows the comparison of algorithms' space complexity.

BAC and BACc have similar space complexities, so they are all considered in the row corresponding to BAC. In the case of MBAC, its complexity is bigger because it needs to store the models. Frame difference uses no model and of course, no model weights, so its space complexity reduces only to input and output variables. LBP com-

Algorithm	weight	model
LBP	$O(n \cdot m \cdot K)$	$O(n \cdot m \cdot K \cdot 2^P)$
Mix. Gauss.	$O(n \cdot m \cdot K)$	$O(n \cdot m \cdot K)$
Frame difference	$O(n \cdot m)$	$O(n \cdot m)$
Mean and threshold	$O(n \cdot m)$	$O(n \cdot m)$
Wallflower	-	$O(n \cdot m \cdot V)$
BAC	-	$O(n \cdot m)$
MBAC	$O(n \cdot m \cdot K)$	$O(n \cdot m)$
FBS	$O(n \cdot m \cdot K)$	$O(n \cdot m \cdot K)$

Table 6.22: Space complexity for each algorithm considered in the experiments, taking as reference an image of $n \cdot m$ pixels.

plexity depends on 2^P the size of each histogram. Wallflower complexity only takes into account the model's complexity, as no model weights are considered in this algorithm. Its complexity is then $O(n \cdot m \cdot V)$, where V is the sum of the number of coefficients and the number of past values used to predict future values with the Wiener filter. The Wallflower algorithm considers a fixed number of models. Finally, the space complexity of fuzzy subtraction algorithm is $O(n \cdot m \cdot K)$ for weights and $O(n \cdot m \cdot K)$ for the model.

Considering the results obtained in tables 6.20, 6.21 and 6.22, it can be stated that FBS yields results comparable to other approaches with the lowest total time consumption and the lowest space complexity. The BAC algorithm provides with a method to compute a reliable background model but it fails if there's clutter in the background or in the contrast between objects and background is very small.

Figure 6.14 shows the background models considered by the proposed algorithms and the two implemented versions of Stauffer's algorithm when the control frame was about to be segmented. Note that BAC or BACc produce a noisy background model for sequences with clutter in the background such as the waving tree sequence. On the other side, the model used by Stauffer's algorithm for the light switch sequence is wrong as the algorithm cannot adapt itself to sudden changes in the scene.

6.6.8 Combining BAC and the FBS algorithm

Most of the multi-modal algorithms introduced in this work, usually expect user to provide the algorithm with the values of K, number of models, and γ, model replacement speed. After the experiments performed to compare the performance of the algorithms on their own it can be seen that BAC, when provided with strict values in order to force it creating a reliable background model, can be used to provide with an estimate of both parameters K and γ. This section discusses another set of experiments performed using BAC to compute an initial background model that could be used by the FBS algorithm as the first model.

BAC computes the confidence of a pixel in such a way, that it reaches easily values around 0.9, over a maximum of 1, after 10 or 20 consecutive frames without changes, recall figure 6.3. If we apply this algorithm to the input sequence until the computed background model achieves a certain confidence θ, we obtain an initial model $B_{x,y}(0)$ and for each pixel (x, y), $h(x, y)$,the number of times the model was modified, and $g(x, y)$, the confidence obtained at the end of the process.

It is easy to see that the values of K and γ have an strong relationship with the values of h and g. The bigger the value of $h(x, y)$, the most models pixel $B_{x,y}$ may need, and the higher the confidence $g(x, y)$ obtained, the lowest its need of changing

Figure 6.13: Quantitative results for the Wallflower benchmark. The first row shows the original control frames captured from the sequence. The second row shows the control frames segmented by hand. Remaining rows show the result of each algorithm for each sequence.

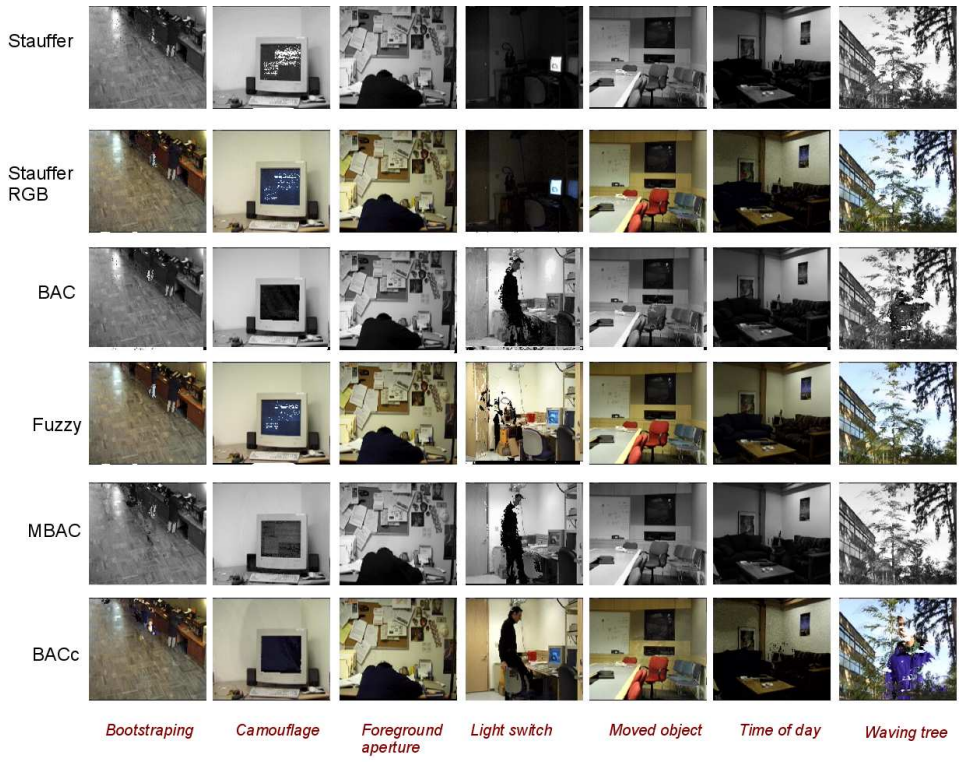


Figure 6.14: Background models computed by the the proposed algorithms together with Stauffer's approach in grey tones and RGB coordinates.

quickly. As models should show flexibility to changes, K may be estimated following the rule:

$$\forall(x,y),\ K(x,y) = \begin{cases} 2, \text{ if } h(x,y) \leq 2,\ \forall(x,y) \\ min(5, h(x,y)), \text{ if } h(x,y) > 2, \end{cases} \tag{6.59}$$

and,

$$\gamma(x,y) = 1 - g(x,y) \tag{6.60}$$

$\gamma(x,y)$ is restricted to the interval $[0.3, 0.6]$, in order to avoid that a pixel never changes or never arrives to an stable state. Model $B_{x,y}(0)$ is set to the model obtained by BAC. When an stability condition is reached, the FBS algorithm continues with the segmentation and background update.

Table 6.23 shows quantitative results, we measured the foreground and background pixels classified successfully, TP and TN. Results improve slightly those obtained by BAC or by the FBS algorithm on their own.

Algorithm		*bootstrap*	*camouflage*	*fore.*	*lightSwitch*	*movingObj.*	*timeOfDay*	*wavingTrees*
Mix. Gauss.	TP	0.44	0.50	0.92	0.73	-	0.41	0.86
	TN	0.95	0.85	0.73	0.07	1.00	0.98	0.90
LBP	TP	0.71	0.62	0.73	0.79	-	0.86	0.85
	TN	0.90	0.91	0.86	0.66	1.00	0.97	0.77
BAC	TP	0.60	0.75	0.56	0.48	-	0.30	0.83
	TN	0.90	0.76	0.87	0.90	1.00	0.98	0.67
FBS	TP	0.47	0.81	0.67	0.65	-	0.82	0.87
	TN	0.96	0.92	0.83	0.86	1.00	0.88	0.88
Wallflower	TP	0.56	0.74	0.67	0.60	-	0.72	0.86
	TN	0.99	0.98	0.90	0.98	1.00	0.99	0.85
BAC + FBS	TP	0.61	0.81	0.69	0.68	-	0.70	0.94
	TN	0.95	0.91	0.93	0.85	0.99	0.98	0.87

Table 6.23: Results (in %) of the compared algorithms when faced to the different situations represented in the Wallflower benchmark.

6.6.9 Combining the algorithms with a classifier

In this section we discuss an experiment performed with the BAC, MBAC and FBS algorithms together with the classifier discussed in chapter 5. The aim of this experiment is showing that the algorithms developed in this chapter can be effectively used with a classifier, for instance in a surveillance application. We used a sequence recorded by ourselves of 456 frames in which we simulate the behaviour of people in a public facility. In order to extract conclusions from results, objects in the frames were manually labelled as belonging to class *person* or *group of people*. There were 350 objects of class *person* and 38 objects of class *group of people* in the sequence.

Results of this experiment can be seen from two different points of view: the segmentation and the classification process. In the first place, segmentation depends exclusively of the background subtraction method chosen.

Table 6.24 compares the total amount of expected objects to be correctly segmented to those really found by BAC, MBAC and FBS. From the segmentation point of view, it can be seen, that MBAC shows a poor performance because it takes longer to produce an stable model, as long as some objects are included into the model until it becomes robust enough. FBS is also affected by the problem of multimodality. However, in MBAC the use of several models per pixel and the segmentation using equation 6.1

produces a less accurate segmentation, and this effect reflects over time in a poorer background model.

	Total	*BAC*	*MBAC*	*FBS*
Objects	388	325	135	239

Table 6.24: Comparison of successfully segmented objects obtained by BAC, MBAC and FBS. Under the column *total*, the total amount of objects expected to be found in the sequence, regardless of their class.

On the other side, classification results for the three algorithms are shown in table 6.25. In general, there is no confusions of classes *person* or *group of people* with class *luggage*. Confusions between classes *person* and *group of people* are due to some configurations of groups in the scenario, which are difficult to deal with. In this case, it is not important however the performance of the classifier, but to see that the algorithms produce an output.

	BAC			MBAC			FBS		
	Person	*Group*	*Luggage*	*Person*	*Group*	*Luggage*	*Person*	*Group*	*Luggage*
Person	282	25	0	102	11	0	202	21	0
Group	9	9	0	14	8	0	11	5	0
Luggage	0	0	0	0	0	0	0	0	0

Table 6.25: Confusion rates for a sequence with objects of classes *person* and *group of people*, there were no objects of class *luggage* in the sequence. Results are obtained by applying BAC, MBAC and FBS algorithms to the sequence and classifying the obtained objects with the classifier discussed in chapter 5.

6.7 Conclusions

In this chapter we develop three algorithms designed to model scenarios and detect objects, with the aim of being able to give a good performance in demanding scenarios. These scenarios are characterized by having always a significant activity level, making it difficult to obtain a clean model with traditional techniques. These algorithms were developed to respond to some issues that arised while building the system (BAC [rosell08a], [rosell09], MBAC [rosell10] and FBS [rosell10b]).

The algorithms were developed with the aim of giving an answer to the problem of system recovery after model corruption and the computation of a model's quality over time.

The addition of colour to BAC did not improve results, yielding in some cases a poor response in some situations. MBAC improves the results of BAC when there is motion in the background because it considers several models per pixel, however, the use of motion to detect degratations in the background model affect seriously its performance in certain situations. The FBS algorithm, which used $CIEL^*a^*b^*$ colour coordinates because of their perceptual uniformity, had a better response than BAC and MBAC.

The algorithms developed in this chapter detect corrupt models easily. Model corruption is an issue not usually covered by other algorthims, in our opinion, if a system is expected to be autonomous, it should consider the posibility that the model may become useless by whichever reason and that it must be eventually restarted. BAC and

MBAC consider that a background model is corrupt if a sudden increase of foreground pixels is detected. In FBS detection is defined in a mathematical way derived from the algorithm's assumptions.

The FBS algorithm proved to be reliable and permits the detection of regions of interest with great accuracy thanks to the frame-by-frame threshold computation. Results obtained by the FBS algorithm are as good as those obtained by Mixture of Gaussians or the LBP approach for the sequences in the Wallflower benchmark. Indeed, this results are obtained with a lower spatial consumption, this way we meet the constraints stated for the developed algorithms.

Traditionally, background modelling algorithms expect users to specify, at least, the amount of models used to describe the each pixel in the model and the speed at which these models will be changed, if needed. This configuration is performed by user relying on previous experiences. We have shown that BAC can be combined with other methods [rosell10b] in order to permit facing the problem of self-configuring units that can work with autonomy. Experiments with the Wallflower benchmark and with our own sequences make us think that our solution is robust when facing different scenarios and challenges.

Chapter 7

Conclusions and future works

7.1 Conclusions

In this thesis, techniques for object segmentation in situations in which light conditions cannot be controlled are developed and discussed. The developed techniques have been successfully applied to two different problems of the real world, the location, segmentation and identification of container codes and the location and tracking of people in a closed environment as an airport.

The proposed schemas try to apply the most suitable segmentation algorithms to the problem to solve, in an effort to cope with as many different environmental situations as possible. In case algorithms need to be supplied with parameters, these are ranged in intervals as wide as needed, whenever possible. This, as seen, has as a consequence, that output will contain very likely the expected objects together with noise. This noise is reduced by means of filters applied to the segmentation output. In certain applications, time performance is a high priority constraint, in this thesis, comparisons about the time consumption of algorithms is done in order to extract conclusions about their suitability to each of the considered applications.

Experimental results were obtained by using real images obtained in the entrance gate of Valencia trading port and, on the other hand real video sequences. In order to evaluate the performance of the algorithms proposed in this thesis or obtained from the literature, we measured with different methods the location of the detected objects. In chapter 4 and 5 we considered the location of the centre of the located object, in chapter 6 a pixel-wise comparison is performed.

In chapter 2 a review on segmentation techniques is done. This techniques are designed in such a way, that they rely heavily on the light conditions and images taken under non controlled light conditions become difficult to deal with.

In chapter 3, it can be seen that background subtraction techniques are the most popular in the literature. They have proved to achieve a good performance in scenarios with small changes in the background. The basic technique consists on obtaining a model of the scenario, several methods may be found in the literature to build this background model. Light conditions also have an influence on the performance of these algorithms, as most of them rely on thresholds to distinguish which pixels belong to foreground and which belong to background. Algorithms may be sort into two groups, regarding the threshold used. One group uses fixed thresholds and another group uses thresholds which depend on statistical computations.

To develop the method of chapter 4 to detect and recognize container's codes, several algorithms were used to segment image and, by the use of filters, the initial response is reduced to a smaller set. This is achieved by using the top-hat operator, a combination of segmentation techniques, classifiers and filters.

Objects expected to be found by segmentation techniques may always be characterized by specific features, which describe them. Simple geometric features such as height or width, but also location in the image or a class assignment given by a classifier are useful to circumscribe objects in a closed set. We use these features, besides from describing objects, also as discriminators. Objects in the output of the segmentation step may be filtered out by checking which expected features do not meet. In the case of characters, specific height and width may be expected and any object not meeting these constrains may be removed from the output. The kind of filters used depend extremely of the application to be developed; though a general schema may be given, the specific needs and particularities of the system will demand one set of filters or another.

The algorithm discussed in chapter 4 finds objects with the highest probability of belonging to the container's code. Sequences of images representing the same container in different moments of time, are used. These sequences are used in an algorithm that tries to reduce the number of errors in the processing of each single image. Common errors are induced by shadows or damaged containers and may be corrected using sequences, as has been proved. This way, a more reliable and fault-tolerant algorithm is obtained.

Different experiments were performed using the single techniques or a combination of them. Finally, experiments were made with 309 images, from the results it may be concluded that is prererable using LAT and the thresholding technique together over the techniques alone, as better results can be obtained.

With our system, a high degree of success can be achieved. The algorithm needs no parameter to be adjusted. However, it does depend on how the classifier was trained as its performance depends directly of the classifier's performance and how it detects noisy objects. The better the classifier is trained, the better the process will work.

Our works within the SENSE project are discussed in chapter 5 and 6. In this case, real video sequences were used in the experiments, we used the Wallflower benchmark together with sequences recorded by ourselves. In chapter 6 we compare the results obtained by our algorithms with other from the literature.

The application discussed in chapter 5, is a system aimed to locate and track people and luggage in the halls of an airport has also been developed. In this case, time constraints are much more restrictive than in previous system; also, in surveillance systems accuracy is an important factor. This implies the necessity of more accuracy when determining whether a given object is noise or not; as no object may be lost under any circumstance.

We proposed a solution that uses background subtraction in order to locate the objects in the scenario and a process of background update within a given period of time. In the discussed system, we built a classifier in order to separate detected objects into three different classes. Several experiments were performed with different features with the goal of achieving the maximal classification confidence in the minimal time. Finally, a solution using a k-NN classifier and a head detection algorithm is the solution chosen.

As a different way of interpreting background subtraction, we have proposed the use of probabilities in chapter 6. During the development of the surveillance system, a lack of literature about recovering or constructing background models have been de-

tected. Two algorithms aimed to perform background subtraction using probabilities. One of them, BAC, performs a pixel-wise computation of similarity with the background. The other one, FSB, performs a global threshold computation for the entire image, using two membership functions computed on the fly.

In chapter 6 we have discussed three algorithms for background modelling: BAC, MBAC or FSB. As a novelty, we developed the concept of background quality in BAC, being this quality a measure of the confidence of the background model. This measure was extended to the other two algorithms. BAC also considers objects' motion when building the background model. MBAC is an extension of BAC in order to support several models per pixel, with the aim of supporting clutter in the background. On the other hand, FSB computes a global threshold by building on the fly two membership functions for background and foreground, eluding the use of fixed or probabilistic thresholds usually found in the traditional background subtraction methods.

BAC, FSB or MBAC detect corrupt models easily. Model corruption is an issue not usually covered by other algorithms, in our opinion, if a system is expected to be autonomous, it should consider the possibility that the model may become useless by whichever reason and that it must be eventually restarted. BAC and MBAC consider that a background model is corrupt if a sudden increase of foreground pixels is detected. In FSB detection is defined in a mathematical way derived from the algorithm's assumptions.

Also research was performed in this thesis about the initialization of the background modelling algorithms. We proved that BAC can be combined with other methods in order to permit facing the problem of self-configuring units that can work with autonomy. Experiments using a combination of BAC + FSB were performed with the Wallflower benchmark and with our own sequences yield promising results about the robustness of this combination when facing different scenarios and challenges.

Summarizing, the conclusions of this thesis are:

- In this thesis we have developed a reliable technique to detect and identify the symbols of truck containers.

- Research has been done in techniques to characterize objects. Specially, different features have been tested for object recognition in the field of surveillance applications, these features were tested on a database of objects manually segmented by ourselves.

- In order to perform experiments, we have recorded and processed real-life sequences.

- In this thesis three novel techniques for background modelling have been proposed. An important issue from our point of view, is the independence of the algorithms to environmental conditions, thus, the proposed techniques are able to restart the background models if errors are encountered. Moreover, the algorithms compute a quality measure of the background model, which is also a novelty in this field.

- The comparison of results obtained with the FSB proposed in this thesis with other works found in the literature, show that our proposal is robust when facing different scenarios and challenges.

7.2 Future works

Further efforts will focus on improving execution times by parallelizing parts of the process of locating and recognizing container's codes. The very nature of the solution proposed clearly points in the direction of parallelizing the execution of the different algorithms and the different processing flows. Moreover, it is possible obtaining a compact and reduced implementation of the system in order to implement the system in low cost, low consumption systems as, for instance, systems-on-chip (SoC).

Better results can also be obtained if the classifier is improved by adding more images to train the system. Specially, improving the description of the class noise by adding more instances would benefit the global results.

Also, following with the truck containers application, designing a fast process that could roughly locate the container's code in the image would be helpful in order to diminish time execution of the algorithm.

The use of sequences can be improved if images are segmented at the same time new images are captured. By using this pipeline structure, the improvements obtained by using sequences can be obtained at a low temporal cost. An open line is also researching new methods to fuse the information obtained from the different images considered in the sequence.

Surveillance system may also be improved by improving the classifier, finding a more robust set of features or a new approach to classify objects. Specially when it is about separating classes *person* and *people*. Experiments with the k-NN classifier show that both classes are not easy to separate and, though the use of symmetry axes improves results, maybe other methods should be explored. A classifier based on other features, different from shape or silhouette is also desirable.

Crowd estimation is also a challenge, either by using directly the results of the classification method introduced in this thesis is an open line or by applying statistical estimations to the gathered data.

Research is still to be done with the background modelling methods using colour constancy [gevers97]. Also, using colour invariance seems to be a way to obtain a better tracking system able to manage with occlusions [nguyen02]. In any case, an open line is still obtaining better background descriptions that yield better segmentations.

Finally, the background modelling algorithms discussed in this thesis have an immediate application to AAL (Ambient Assisted Living) [sunand09]. In this case, the aim of the applications is not locating threats or lost objects but constructing safe environments around assisted peoples and helping them maintain an independent live.

Appendix A

Morphological operators

In this chapter, a quick review to morphological operators is shown. These operators form an algebraic system of operators. They may be combined among them in order to extract the most significant parts of complex images. According to authors in [soille99], this operators allow identifying essential parts of figures in order to reconstruct them optimally from distorted images. Morphological operators can simplify an image, keeping its essential features and removing any information which is not useful.

The basic unit of information in a morphological approximation is the binary image; although the operators may be extended to work analogously with grey-level images. Given A, a binary image, and B an structural element (a binary sub-image), ω a universal binary image (that is, an image which has all its pixels set to 1), the four basic morphological operators are defined as follows:

- intersection : $A \bigcap B = \{p \in A \wedge p \in B\}$
- union : $A \bigcup B = \{p \in A \vee p \in B\}$
- complementation : $A^c = \{p \in \Omega \wedge p \notin A\}$
- traslation : being p a pixel, the traslation of A by p is given by $A_p = \{a + p : a \in A\}$

By combining these morphological operators, other operators may be obtained, as, for instance, the top-hat operator. Two important operations which have to be defined before using this operator are the dilation and erosion operation.

- Dilation: being $A_{b1}, A_{b2}, A_{b3}, ...$, the union traslation of a binary image A by the set of pixels with value equal to 1 of the binary image $B = \{b_1, b_2...\}$; is called dilation of A by B, expressed as :

  $$A \oplus B = \bigcup b_i \in B A_{b_i}$$

  Dilation verifies both the commutative and associative properties. Image B is usually a regular shaped figure known as Structural Element (SE for short).

- Erosion: the dual operation of dilation, defined as :

  $$A \ominus B = \{p : B_p \subseteq A\}$$

Being I a grey-level image with dimensions $M \times N$ and $I(x, y)$ the grey level for pixel located in coordinates (x, y), and the structural element $b(i, j)$ with dimensions $m \times n$. The dilation and erosion operations for grey-level images are defined as :

$$(f \oplus b)(x, y) = max_{i,j}\{f(x-i, y-j)+b(i, j)\}, \forall 0 <= i < m-1, 0 <= j < n-1$$

$$(f \ominus b)(x, y) = min_{i,j}\{f(x+i, y+j)-b(i, j)\}, \forall 0 <= i < m-1, 0 <= j < n-1$$

Appendix B

Object classification

Vision systems do not only perform a task of detecting objects in a scene, but also are expected in most occasions to give a classification of these objects. There are several methods to classify objects, for instance, k-NN classifiers, neural networks, self-organizing feature maps, support vector machines, just to mention some.

Geometrical or statistical classification methods have been used with success in a big amount of real systems. Objects are represented by a set of d numerical values that ease the process of representing them as points in a d dimensional space. Ideally, objects that belong to the same class will be represented by points located in narrow areas in the representation space. The distribution of each of these areas and their organization in the space can be defined by the probability density function associated to the points that compose it. Geometrical methods decide the class of new objects by considering these probability density functions. These functions are, in general, not easy to compute. Usually they can be estimated by means of parametric or non-parametric methods from an initial set of points (the so called training set).

The performance of the classification methods depends on the number of values used to represent objects and also of the number of elements used in the training set. Representing objects with a big amount of values (features from now on) does not assure a better performance. By increasing the amount of features used to represent objects, the complexity of the space increases exponentially. This will force to also increase the number of objects used in the training set, which is not always feasible, with the aim of defining correctly the space areas that belong to each of the classes (non-parametric methods) or being able to estimate the extra parameters that appear (parametric methods). This phenomenon is known in the literature as the *curse of dimensionality* [devijver82]. On the other side, the use of a high number of features will influence the temporal and spatial cost when computing the features for an object and when classifying it, as well.

Another important issue to consider is the possibility that noisy features appear. These features have no real influence in the classification process and confuse classifiers. At the same time, the existence of features with a high degree of correlation that introduce no new information and increase the dimensionality of the space are problems to face.

Summarizing, it is crucial to reduce as much as possible the amount of features used to represent objects; reduce the cost to compute them, speed up the classification process, ease the classification task and improve its precision.

In order to work with the smallest amount of features, there are two steps which are

included in the process of developing a recognition system (see figure B.2):

- *Feature selection*: in this step the goal is obtaining the minimal set of features that maximize the performance of the classifier. As an additional advantage, when reducing the number of features the time invested in obtaining them does also reduce.
- *Feature extraction*: this step is also aimed to reduce the number of features. In this case, the reduction is made by combining the original features in several ways in order to obtain new features that, in a reduced number, obtain the same classification results or even improve them. Feature extraction is aimed to obtain new features from the hidden relationships between the original features.

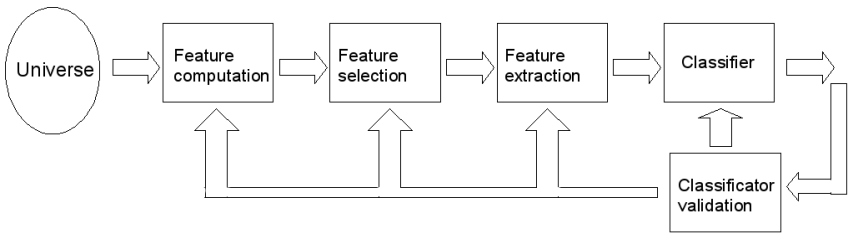


Figure B.1: Stages of a geometrical recognition system.

The k-NN [fukunaga90] is a method for classifying objects based on closest training examples in the feature space. k-NN is a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbours, with the object being assigned to the class most common amongst its k nearest neighbours (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of its nearest neighbour. Nearest neighbour rule in effect compute the decision boundary of classes in an implicit manner.

The neighbours are taken from a set of objects for which the correct classification is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

The k-NN classifier was used in the applications discussed throughout the thesis. In this appendix, the feature sets built for symbols appearing on truck containers along with experimental results.

B.1 Feature sets for characters recognition

Symbols on truck containers are a mixture of characters and numbers. The classifier built for this problem thus, was made collecting images of numbers and characters appearing on images representing truck containers.

B.1.1 Database preprocessing

In [salva02], a k-NN classifier was trained with a corpus of 654 images, with an approximate total amount of 9810 symbols. The image set was acquired in several days

with different lighting conditions. Moreover digits and letters can be light or dark and they appear in both plain and non-plain surfaces. The non-plain surfaces can produce deformations in the code characters and shades. These situations make difficult the recognition task. From each symbol 288 features were obtained, representing the grey tones for each normalized symbol (12×24 pixels). In the following subsections the schema followed to preprocess the database image and obtain the k-NN classifier is outlined.

First, a segmentation process is applied as explained in chapter 4 in order to obtain objects from images. These segmented objects are preprocessed as shown in figure B.2 to obtain the feature vector representing them. For the training set, objects are labelled using the manually segmented symbols in the way described in section 4.4.2. Objects overlapping manual segmented symbols are labelled with the suitable symbol, the others are labelled as noise.

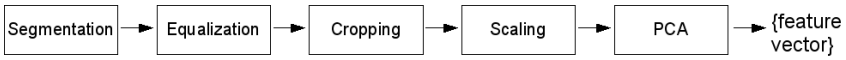


Figure B.2: The stages into which the preprocessing of the database divides.

The feature extraction process is carried out on each of the sub-images obtained by the segmentation process. In order to obtain the characteristics of these objects the following steps are carried out: top-hat (see appendix A), equalization, cropping and scaling.

B.1.2 Equalization

After applying top-hat to the sub-image a grey level image is obtained, it usually has a very poor dynamic range and the background is always darker than the object. Next, by means of an equalization [gonzalez93] the grey levels of this image are spread out and reach white. This technique increases the dynamic range and consequently produces an improvement in the image contrast, see figure B.3. Consequently this process increases distance between the grey values of the object pixels and the background pixels.

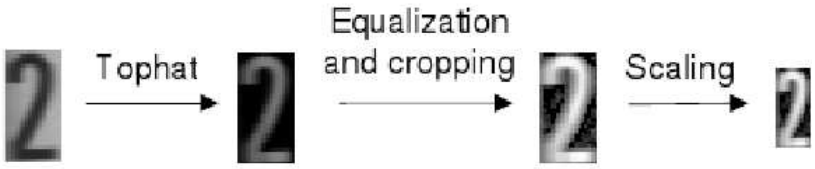


Figure B.3: Steps carried out to extract features for each object present in the segmented sub-image

B.1.3 Cropping

The bounding box obtained in the segmentation step may contain the character and some spots. Thus, if we have two characters of the same class, one with a little noise and other without it, their feature vectors can be quite different after rescaling them.

Therefore, the cropping process has been applied in order to assure that in the considered sub-image the character is touching the edges of the same one.

Figure B.4: Result of cropping in a noisy character

Given the equalized image $E(p,q)$ with pixels $e(i,j)$ for $i = 0...p-1, i = 0...q-1$ the minimum $g1$ and the maximum $g2$ grey tones of $E(p,q)$ are computed as follows,

$$g1 = min(e(i,j) \in E(p,q)) \tag{B.1}$$

$$g2 = max(e(i,j) \in E(p,q)) \tag{B.2}$$

The row $\hat{i}$ is cropped if all of their pixels satisfy the following condition,

$$\forall e(\hat{i},j) \in [g1, m \times (g2 - g1) + g1] \tag{B.3}$$

where $m \in [0,1]$. In a similar way, the column $\hat{j}$ is cropped if all of their pixels fulfil the following condition,

$$\forall e(i,\hat{j}) \in [g1, m \times (g2 - g1) + g1] \tag{B.4}$$

The cropping process begins in the first row, or column, and stops when it finds a row which does not fulfil the condition stated in equation B.3 or, in the case the processing is performed per columns, in a column, in case the condition equation B.4 is not met. Similar steps are carried out beginning by the $p-1$ row, or $q-1$ column, to continue analysing $p-2$ row, the $q-2$ column, etc. The result of this step can be seen in figure B.4.

B.1.4 Scaling

After cropping, the obtained sub-images are scaled or quantified with a grid of 12×24 obtaining 288 grey tones. These 288 features were used to represent the objects.

B.1.5 PCA

After scaling the images and obtaining the features, the PCA technique [fukunaga90] was used in order to reduce the amount of features. This speeds up the classification process as explained previously. Experiments were performed with different amount of features out of the 288 obtained in the previous step. A set of 60 features revealed as the minimum set of features that maintains the original recognition rate.

B.2 Training

In this task 47 classes were considered, 26 for letters, 10 for digits, 10 for control digits (digits surrounded by a square) and 1 for the noise class which contains the most frequent errors as labels, mirrors, tires and some background objects. A total of 7848 samples, the 80% of the total amount of samples, have been used for the training corpus. The test set has 1962 samples, the 20% of the total amount. In order to perform the search we have used the approximated k-NN (ANN) [arya98], which get good results in a very short time. ANN combines the simplicity and performance of classical k-NN with the kd-tree structure [bentley80].

An initial recognition rate of 92.37% was obtained. The corpus can be grown artificially by adding some transformations of the original objects [salva02], such as translations in the 8 directions, rotations and several borders.

After several experiments [salva02], the new corpus was composed of the following sets: the original training corpus (border 0), its translations of 1 and 2 pixels, rotations of 2, 4 and 6 degrees, border 1 with translations of 1 pixel and rotations of 2, 4 and 6 degrees.

This new corpus gets a 93.92% of recognition rate. These results were obtained with $m = 0.0$. The next step is to explore which cropping value gets the best result. Table B.1 shows that for $m = 0.4$ the recognition rate achieves a value of 94.81%, which is approximately 2.5% better than the best one with the original corpus.

m	*Recognition rate (%)*
0.0	93.92
0.25	94.33
0.30	94.54
0.35	94.78
0.40	94.81
0.45	94.26
0.50	93.92
0.55	93.54

Table B.1: Recognition rate for the mixed corpus

In this work, the classifier was improved by means of bootstrapping. With this technique, the classifier is used to classify a sequence of new objects, the objects whose classification is wrong are used to grow the classifier. As explained in chapter 4, a bootstrapping process was performed inserting objects erroneously classified into the classifier. This process led to obtain a performance of 95.69% of classification success.

B.3 k-NN confidence computation

The confidence of the classification is used by the confidence filter explained in section 4.4.2. The confidence value given by a k-NN classifier is computed as an estimation of the maximum likelihood of the *a posteriori* probability of classes by means of the expression:

$$\hat{P}(\omega_i|x) = \frac{k_i}{k} \tag{B.5}$$

where k is the total amount of neighbours of x considered by the classifier and k_i is the number of neighbours that belong to class ω_i. The Bayes error estimation using

the $\hat{P}(\omega_i|x)$ yields an optimistic estimation of error, though quite adjusted in the case the k-NN classifier was designed using the following convergence conditions:

1. $k \to \infty$
2. $k/n \to 0$ when $n \to \infty$

being n the number of elements in the set used to design the k-NN classifier. As long as the number of samples cannot be, usually, arbitrarily large, these constraints will not be met and thus, the error estimation must be computed following a different strategy.

When estimating $\hat{P}(\omega_i|x)$, it is also possible to use the expression:

$$\hat{P}(\omega_i|x) = \frac{\sum_{u \in \theta_{K_i}} \frac{1}{d(u,x)}}{\sum_{v \in \theta_K} \frac{1}{d(v,x)}} \tag{B.6}$$

being θ_K the number of neighbours of x, θ_{K_i} represents the number of neighbours of x that belong to class ω_i and the function $d(.,.)$ corresponds to the distance between two observations [arlandis02]. As it can be seen, expression B.6 takes into account the distance at which neighbours lay. This additional information can be useful in the case the design of the classifier does not meet the convergence constraints.

The confidence measure computed by equation B.6 is used to assign a confidence value to the classified symbols.

Bibliography

[arlandis02] J. Arlandis, J. C. Pérez-Cortés, and J. Cano. Rejection strategies and confidence measures for a k-nn classificer in an ocr task. *IEEE International Conference on Pattern Recognition (ICPR02)*, 2002.

[arya98] S. Arya, D. Mount, N. Netanyahu, R. Silverman, , and A. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 1998.

[atienza05] V. Atienza, A. Rodas, G. Andreu, and A. Pérez. Optical flow-based segmentation of containers for automatic code recognition. *Lecture Notes in Computer Science*, 3686:636–645, 2005.

[atienza08] V. Atienza, J. Rosell, G. Andreu, and J. M. Valiente. People and luggage recognition in airport surveillance under real-time constraints. *IEEE International Conference on Pattern Recognition (ICPR08) Tampa. USA.*, Dec. 2008.

[barroso97] J. Barroso, A. Rafael, E. L. Dagless, and J. Bulas-Cruz. Number plate reading using computer vision. *International Symposium on Industrial Electronics (ISIE97), Guimaraes, Portugal*, 1997.

[benet10] G. Benet, J. Simó, G.Andreu-García, J. Rosell-Ortega, and J. Sánchez. Embedded low-level video processing for surveillance purposes. *Proceedings of the 3rd International Conference on Human System Interaction. Rzeszow, Poland*, May 2010.

[benezeth08] Y. Benezeth, P.M. Jordin, B. Emile, H. Laurent, and C. Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. *IEEE International Conference on Pattern Recognition.(IAPR08)*, 2008.

[bentley80] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected time algorithms for closest point algorithms. *ACM Transactions on Mathematical Software*, pages 563– 580, 1980.

[beucher79] S. Beucher and C-Lantuéjoul. Use of watersheds in contour detection. *CCETT/INSA/IRISA IRISA Report n. 132, Rennes, France*, pages 2.1–2.12, 1979.

[beucher91] S. Beucher. The watershed transformation applied to image segmentation. *Conference on Signal and Image Processing in Microscopy and Microanalysis*, pages 299–314, September 1991.

[brad01] R. Brad. License plate recognition system. *Proceedings of the 3rd International Conference on Information, Communications and Signal Processing, Singapore*, pages 2–6, 2001.

[cheng00] V. Cheng and N. Kehtarnavaz. A smart camera application: Dsp-based people detection and tracking. *Journal of Electronic Imaging*, pages 336 – 346, 2000.

[cormen90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. The MIT Press, McGraw-Hill Book Company, 1990.

[cucchira01] R. Cucchira, C. Grana, M. Piccardi, and A. Prati. Detecting objects, shadows and ghosts in video stream by exploiting colour and motion information. *IEEE International Conference on Image Analysis and Processing (ICIAP01)*, pages 360 – 365, 2001.

[devijver82] P.A. Devijver and J. Kittler. *Pattern recognition, a statistical approach.* Prentice Hall. Englewood Cliffs. London, 1982.

[elbaf09] F. El Baf, T. Bouwmans, and B. Vachon. Fuzzy statistical modeling of dynamic backgrounds for moving object detection in infrared videos. *IEEE Computer Vision and Pattern Recognition Workshops (CVPR09)*, pages 60–65, 2009.

[elgammal00] A. Elgammal, D. Harwood, and L.S. Davis. Non-parametric model for background subtraction. *European Conference on Computer Vision (ECCV00)*, pages 751 – 767, 2000.

[felzen98] P. Felzenszwalb and D. Huttenlocher. Image segmentation using local variation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR98), Santa Barbara, CA*, pages 98 – 104, 1998.

[foresti05] G. Foresti, C. Micheloni, L. Snidaro, P. Remagnino, and T. Ellis. Active video-based surveillance system: the low-level image and video processing techniques needed for implementation. *IEEE Signal Processing Magazine*, 22(2):25–37, 2005.

[friedman97] N. Friedman and S. Russell. Image segmentation in video sequence: a probabilistic approach. *Proc. od the Thirteen Conf. on Uncertainty in Artificial Intelligence*, 1997.

[fuentes03] L. Fuentes and S. Velastin. From tracking to advanced surveillance. *IEEE International Conference on Image Processing (ICIP03)*, 3:121–124, 2003.

[fukunaga90] K. Fukunaga. *Statistical Pattern Recognition*. Academic Press, second edition edition, 1990.

[gepperth05] A. Gepperth, J. Edelbrunner, and T. Beucher. Real-time detection and classification of cars in video sequences. *IEEE Intelligent Vehicles Symposium*, pages 625– 631, 2005.

[gevers97] T. Gevers and A. Smeulders. Color based object recognition. *Pattern Recognition*, 32:453–464, 1997.

[gonzalez93] R. González and R. Woods. *Digital image processing*. Addison-Wesley Publishing Company, 1993.

[grimson99] W.E.L. Grimson and C. Stauffer. Adaptive background mixture for real-time tracking. *IEEE Computer Vision and Pattern Recognition (CVPR99)*, pages 246 – 252, 1999.

[han04] B. Han, D. Comaniciu, and L.S. Davis. Sequential kernel density approximation through mode propagation: applications to background modeling. *Proc. Asian Conference on Computer Vision*, 2004.

[haritaoglu00] I. Haritaoglu, D. Harwood, and L. S. Davis. W4: Real-time surveillance of people and their activities. *IEEE Pattern Analysis and Machine Intelligence*, pages 809 – 830, 2000.

[hart68] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, IT-14:515–516, 1968.

[hegt98] H. A. Hegt, R. J. de la Haye, and N.A. Khan. A high performance license plate recognition system. *IEEE Int. Conference on Systems, Man and Cybernetics*, 5:4357–4362, 1998.

[heikkila04] M. Heikkila., M. Pietikainen, and J.Heikkila. A texture-based method for detecting moving objects. In *British Machine Vision Conference (BMVC04)*, 2004.

[heikkila06] M. Heikkila and M. Pietikainen. A texture-based method for modeling the background and detecting moving objects. *IEEE Pattern Analysis and Machine Intelligence*, 28(4):657–662, April 2006.

[hengstler07] S. Hengstler and H. Aghajan. Application-oriented design of smart camera networks. *First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 9 – 16, 2007.

[horpraset99] T. Horprasert, D. Harwood, and L. S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. *IEEE International Conference on Computer Vision (ICCV99). FRAME-RATE Workshop*, pages 1 – 19, 1999.

[hu04] W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, man, and Cybernetics*, 2004.

[huang09] Deng-Yuan Huang and Chia-Hung Wang. Optimal multi-level thresholding using a two-stage otsu optimization approach. *Pattern Recognition Letters*, 30(3):275–284, 2009.

[kalman60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35 – 45, 1960.

[kirby79] R. L. Kirby and A. Rosenfeld. A note on the use of (gray level, local average gray level) space as an aid in threshold selection. *IEEE Transactions on Systems, Man and Cybernetics SMC-9*, pages 860–864, 1979.

[koller93] D. Koller, K. Daniilidis, and H.H. Nagel. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision (IJCV93)*, 10(3):257–281, June 1993.

[lee05] Dar-Shyang Lee. Effective gaussian mixture learning. *IEEE Pattern Analysis and Machine Intelligence*, pages 827–832, May 2005.

[makhoul75] J. Makhoul. Linear prediction: a tutorial review. *Proceedings of the IEEE*, 63(4):561–580, 1975.

[marcenaro00] L. Marcenaro, F. Oberti, and C. S. Regazzoni. Change detection methods for automatic scene analysis by usingmobile surveillance cameras. *IEEE International Conference on Image Processing (ICIP00)*, 1:244 – 247, 2000.

[mason01] M. Mason and Z. Duric. Using histograms to detect and track objects in color video. *Applied Imaginery Pattern Recognition Workshop*, pages 154 – 159, 2001.

[mckenna00] S. McKenna, S. Jabri, Z. Duric, A. Rosenfeld, , and H. Wechsler. Tracking groups of people. *Computer Vision and Image Understanding*, 80(1):42 – 56, 2000.

[mikic00] I. Mikic, P. C. Cosman, G. T. Kogut, and M. M. Trivedi. Moving shadow and object detection in traffic scenes. *IEEE International Conference on Pattern Recognition (ICPR00)*, 1:13 – 21, 2000.

[nguyen02] H.T Nguyen and A.W.M. Smeulders. Template tracking using color invariant pixel features. *IEEE International Conference on Image Processing. 2002. (ICIP02)*, pages I–569 – I–572, 2002.

[nguyen03] N. Nguyen, S. Venkatesh, G. West, H. Bui, , and A. Perth. Multiple camera coordination in a surveillance system. *Acta Automatica Sinica*, 29(3):408 – 422, 2003.

[oliver00] N.M. Oliver, B. Rosario, and A. P. Pentland. A bayesian computer vision system for modeling human interactions. *IEEE Pattern Analysis and Machine Intelligence*, 22(8):831 – 843, 2000.

[otsu79] N. Otsu. A threshold selection method for gray-level histograms. *IEEE Transactions on Systems, Man and Cibernetics*, 9, 1979.

[piccardi04] M. Piccardi. Background subtraction techniques: a review. *IEEE International Conference on Systems, Man and Cybernetics*, 4:3099–3104, Oct 2004.

[piccardi04a] M. Piccardi and T. Jan. Efficient mean-shift background subtraction. *IEEE International Conference on Image Processing (ICIP04)*, Oct 2004.

[porikli03] F. M. Porikli and O. Tuzel. Human body tracking by adaptive background models and mean-shift analysis. *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, 2003.

[prati01] Andrea Prati and Rita Cucchiara. Analysis and detection of shadows in video streams: A comparative evaluation. *IEEE International Conferece Computer Vision and Pattern Recognition. (CVPR01)*, pages 571–576, 2001.

[rosell06] J. Rosell, A. Pérez, and G. Andreu. Segmentation algorithms for extraction of identifier codes in containers. *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP06) Portugal*, pages 375–380, Feb 2006.

[rosell06a] J. Rosell, G. Andreu, and A. Pérez and. Processing and recognition of characters in container codes. *6th WSEAS Int. Conf. on Signal Procesing, Computacional Geometry and Artificial Vision (ISCGAV06). Crete, Greece*, Aug 2006.

[rosell06b] J. Rosell, A. Pérez, and G. Andreu. Fault-tolerant search of container codes. *6th WSEAS Int. Conf. on Signal, Speech and Image Processing (SSIP06). Lisbon. Portugal*, Sep 2006.

[rosell08] J. Rosell, G. Andreu, A. Rodas, V. Atienza, and J. Valiente. Feature sets for people and luggage recognition in airport surveillance under real-time constraints. *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP08). Madeira, Portugal*, Feb 2008.

[rosell08a] J. Rosell, , G. Andreu, A. Rodas, and V. Atienza. Background modelling in demanding situations with confidence measure. *IEEE International Conference on Pattern Recognition (ICPR08). Tampa, USA. doi = http://dx.doi.org/10.1109/ICPR.2008.4761047*, Dec. 2008.

[rosell09] J. Rosell and G. Andreu. *Pattern Recognition*, chapter Background modelling with associated confidence, pages 15 – 30. In Tech, 2009.

[rosell10] J. Rosell, , G. Andreu, A. Rodas, and V. Atienza. Background modeling with motion criterion and multi-modal support. *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications VISAPP10. Angers, France*, May 2010.

[rosell10b] J. Rosell-Ortega, G. Andreu-García, A. Rodas-Jordà, and V. Atienza-Vanacloig. A combined self-configuring method for object tracking in colour video. *IEEE International Conference on Pattern Recognition ICPR10. Istambul, Turkey. doi= http://dx.doi.org/10.1109/ICPR.2010.1154*, 2010.

[rosin95] P. Rosin and T. Ellis. Image difference threshold strategies and shadow detection. *Proceedings of the British Machine Vision Conference (BMVC95)*, pages 347 – 356, 1995.

[rowley97] H.A. Rowley and J.M. Rehg. Analyzing articulated motion using expectation-maximization. *IEEE Pattern Recognition*, 1997.

[sacchi01] C. Sacchi, G. Gera, L. Marcenaro, and C. Regazzoni. Advanced image-processing tools for counting people in tourist site monitoring applications. *Signal Processing*, 81:1017–1040, May 2001.

[salva01] I. Salvador, G. Andreu, and A. Pérez. Detection of identifier codes in containers. *Simposium Nacional de Reconocimiento de Formas y AnÃ¡lisis de ImÃ¡genes (SNRFAI01). Castellón, Spain*, 1:119–124, May 2001.

[salva02] I. Salvador, G. Andreu, and A. Pérez. Preprocessing and recognition of characters in container codes. *IEEE International Conference on Pattern Recognition ICPR02*, 2002.

[seki03] M. Seki, T. Wada, H. Fujiwara, and K. Sumi. Background subtraction based on coocurrence of image variations. *IEEE Computer Vision and Pattern Recognition (CVPR03)*, 2:65 – 72, 2003.

[sense04] SENSE. Smart embedded network of sensing entities. http://www.sense-ist.org/index.html.

[shoushtarian05] B. Shoushtarian and H. E. Bez. A practical adaptive approach for dynamic background subtraction using an invariant colour model and object tracking. *Pattern Recognition Lettes*, 26(1):5–26, 2005.

[soille99] P. Soille. *Morphological image analysis: Principles and applications*. Springer Verlag, 1999.

[stringa00] E. Stringa. Morphological change detection algorithms for surveillance applications. *British Machine Vision Conference (BMVC00)*, 2000.

[sunand09] H. Sunand V. De Florio, N. Gui, and C. Blondia. Promises and challenges of ambient assisted living systems. *International Conference on Information Technology: New Generations (ITNG09)*, pages 1201 – 1207, April 2009.

[toyama99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. *IEEE International Conference on Computer Vision (ICCV99), Kerkyra, Greece*, pages 255–261, 1999.

[varcheie08] P. D. Z. Varcheie, M. Sills-Lavoie, and G. Bilodeau. An efficient region-based background subtraction technique. *Canadian Conference on Computer and Robot Vision*, pages 71–78, 2008.

[viola03] P. Viola, M. J. Jones, and Daniel Snow. Detecting pedestrians and using patterns of motion of appearence. *IEEE International Conference on Computer Vision (ICCV03)*, pages 734 – 741, 2003.

[vsam99] R. Collins, A. Lipton, and T. Kanade. A system for video surveillance and monitoring. *Proc. American Nuclear Society (ANS) Eighth International Topical Meeting on Robotics and Remote Systems, Pittsburgh*, pages 25–29, 1999.

[wang03] L. Wang, W. Hu, and T. Tan. Recent developments in human motion analysis. *Pattern Recognition*, 3:585–601, 2003.

[wixson00] L. Wixson. Detecting salient motion by accumulating directionally-consistent flow. *IEEE Pattern Analysis and Machine Intelligence*, 8:774 – 780, 2000.

[wren97] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 780 – 785, 1997.

[xu05] Li-Quan Xu, J. L. Landabaso, and M. Pardàs. Shadow removal with blob-based morphological reconstruction for error correction. *Proceedings on Acoustics, Speech, and Signal Processing (ICASSP05)*, 2:729 – 732, 2005.

[yang92] Y. H. Yang and M. D. Levine. The background primal sketch: an approach for tracking moving objects. *Machine Vision and Applications*, pages 17 – 34, 1992.

[zeng08] Jia Zeng, Lei Xie, and Zhi-Qiang Liu. Type-2 fuzzy gaussian mixture models. *Pattern Recognition*, 41(12):3636 – 3643, 2008.

[zivkovic04] Z. Zivkovic. Improved adaptive gaussian mixture model for background subtraction. *IEEE International Conference on Pattern Recognition (ICPR04)*, 2004.