Final Report of the work done on shadow removal

An adaptive approach to Shadow Removal for the detection of vehicles in Video Surveillance Systems and an 8-connectivity labelling algorithm

Riccardo Carlesso

February 2, 2003

Contents

Abstract

This 6-month collaboration with ARCES has given birth to the implementation of two main works: an adaptive shadow removal algorithm and an 8-connettivity labelling algorithm optimized in order to remove 'small' blobs[1].

In all video surveillance tasks a good blob-detection module is highly desirable for a good elaboration. The detection module must cope with many problems that arise for the 'overconfidence' to be able to get back the whole scene information from its 2D model (exploiting time redundancy). Among the problems of detection, the shadow-removal is yet an unresolved issue; many techniques have been proposed to remove shadows but they mostly rely on manually tuning parameters which are critic to the task. Many efforts have been made to produce a real adaptive algorithm but although good proposes have been made, they often just work in isolated cases. A novel algorithm is provided that can automatically tune the most critical parameters in the shadow-removal algorithm (providing some requirements are met) based on a heuristics coming out from the direction of the strongest light source. Experimental proofs are shown along with times estimates. Conclusions are taken along with some possible future work.

[1]This paper provides documentation about the former only, the latter being described in another paper.

Chapter 1

Introduction

Video Surveillance is a particular task of Computer Vision that involves comprehension of what is being observed (typically, a video-sequence), with the ultimate goal to provide *automatic information* about it. In [5], a three-generation evolution is explained and motivated in video surveillance topic.

This wide research area has undergone a strong evolution since 60's until today. In the early years, just analog cameras and close-circuit systems (CCS) where available; they allowed to have a central control room where an operator could monitor many different places of the building. These *first generation video surveillance systems (1GSS)* had the only ability to time-multiplex N analog cues into one monitor.

As the years went by, though, digital elaboration became possible; moreover, it was proven that the operator's attention span is limited and after a couple of hours in front of many cameras the miss rate gets high: computers must aid operators in order to pre-filter the video sequences and 'wake' him just when something happens, whether by playing sounds when some motion is detected or by selecting the most interesting camera from that point of view.

A typical block scheme of this includes 3 modules, as can be seen from:

- a *detection module*, which computes the input images (coming from an analog or digital source, i.e. a camera or an AVI file) and outputs a binary image selecting the only pixels thought as *moving* (or *interesting*, it depends on the semantics). The output of this module mainly consists in noisy groups of *BLOBs* (white uniform spots of moving points in a black background). **Def.** *A **BLOB** (*Binary Large OBject*) is a connected component of a bitonal image (true = 255 = white = foreground; false = 0 = black = background).* Single blobs cannot be considered yet as atomic moving objects in the scene: a real object

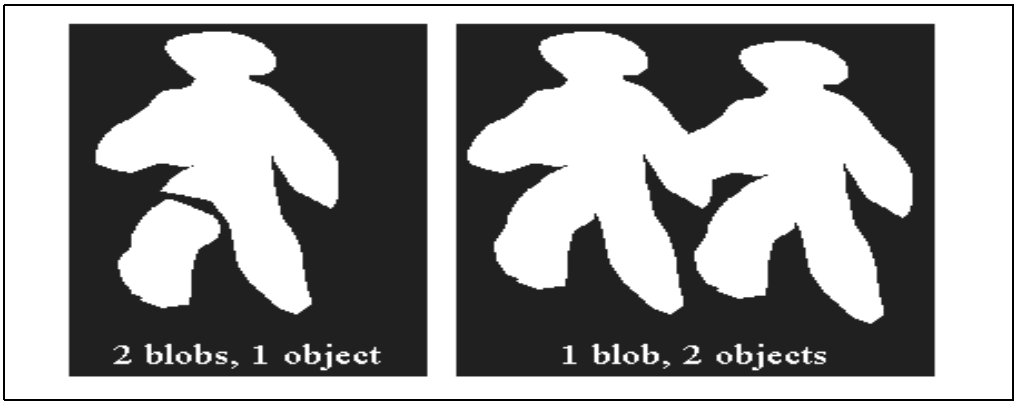

Figure 1.1: The difference between Blob and Object: the former is just a connected spot, the latter depends on the scene's semantics.

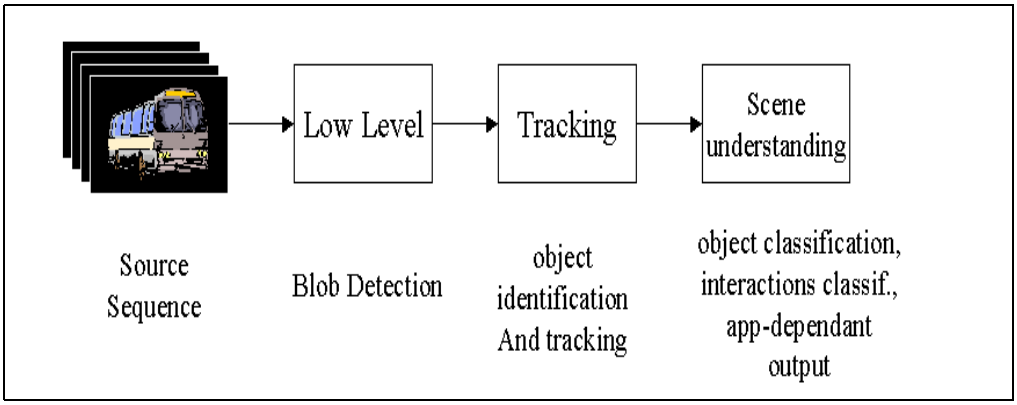

Figure 1.2: A typical block scheme of a Video Surveillance System.

could consist, in the operator's opinion, of more than one blob (*split object*), or viceversa a single blob could consist of more *merged objects* (fig. 1.1).

- a *tracking module*, which has to correctly associate the $\widetilde{n}_t$ blobs in each frame to n_t objects *and* to correctly associate object in time, setting correspondences between the n_{t-1} objects of the past frame and the n_t ones in the current frame. This two numbers do not have necessarily to be equal: some objects could (1) be born, (2) die, (3) merge or (4) split. Many decisions are to be made in this section, after which the history of an object is gained (i.e., a *consistent temporal information*): point of birth/death, trajectory, speed. This task is very tricky and

much knowledge is often applied in order to cope with the difficulties arising in this part.

- a *classification/decision module*. This module aims to provide semantics to the scene, starting from the single objects, passing through the interactions between objects, and finally coming to a synopsis that depends on the application. The most obvious way to grant semantics to an object is to assign it to an *a priori*-known class (i.e. {vehicle, human, group of humans}, or even {car, motorcycle, van, truck}...), based on some features or perhaps their behavior detected in tracking phase (a man that zigzags slowly in a parking lot, a case left in a railway station, ...). This the most customizable module, and the most application-dependant. For generality purposes, our group has tried not to implement this one in order to achieve a *generic* tracking system, whose output is the only tracking module output.

This simple scheme is agreed by a vast community working on video surveillance ; most researchers implement the first module according to a restricted number of well-known algorithms; the implementation of the second and third module, instead, is very different from case to case. The third module largely consists of a classifier (often neural-based), a decision module and a part of data-logging. It has to be an help to the operator, as it has to be easy for the operator to validate the decisions took from the program. The third module has to be a front-end for the operator, as it is the ultimate output.

It's important to point out that nowadays, as networks have become a natural extension of most applications, a strong video surveillance aim is to save *bandwidth* doing hard elaboration tasks locally, then conveying the output in a server; the band saving goes in two directions:

time the line isn't busy at every time (just when something interesting happens);

content high level semantics is much cheaper than raw video: a few words ($\approx 100B$) can 'replace' 10 seconds of full video ($\approx 100MB$, uncompressed).

It will be important, though, to have the possibility to *mine* sequences back when necessary; much backup data will probably be stored in the peripheral parts of the system.

1.1 A typical 3GSS structure

In this section the video surveillance system projected and implemented by our group will be described, focusing on decisions it relies on and the algorithms used with respect to other possible 3GSS systems.

Module 1: Low Level

The low level module acquires a *true color* (24 bit) video sequence from camera or file and outputs a bitonal (1 bit) video sequence. Choosing which pixels are moving depends on the strategy used for detection: a point is 'detected' as an object if it's *enough* different from a *default* value. When this default is the same point in the previous frame, it's called **frame difference**; if the default is seen as an underlying layer (a fixed background), it's called **background subtraction**. Background can change and adapt to different conditions (light, object removed from the scene, ...).

- **Background subtraction**. A background is modelled and updated every frame making some assumptions. A simple way to do this is to acquire it once for all when the scene is empty of any object.

  A light change can severely interfere with this algorithm, as the whole background can suddenly change and a background pixel can be falsely detected as object pixel.

  The drawback is that the background has to be modelled, and each model has its own flaws (i.e., it fails in some occasions where assumptions made aren't met anymore).

- **Frame difference**. No background is modelled: a simple absolute pixel-wise difference is calculated from frame $F(t)$ and the previous $F(t-1)$. The main disadvantage is that a slow (or stopped) object is not detected this way. A good way to reduce noise is to include in the difference three consecutive frames instead of two (double frame difference).

Our group chose the *background subtraction* algorithm, for it proved more robust. Background models have many flaws but they can be overridden by introducing some knowledge-based criterions (do not update the background when a point is supposed object, do not trust some choices based on a feedback from the tracking module, ...).

Both algorithms suffer from the so called *waving leaves* menace: wind waves tree-leaves so that every pixel in that area is often seen as object

because no background can be stably modelled around it and it tends to change abruptly from scene to scene.

A point is considered object if the pixel-wise difference exceeds a **threshold** (that represents the maximum noise allowed to the 'default' in order to be considered as such).

Module 2: Tracking

The tracking system's purpose is to correctly associate blobs to objects. This is a prerequisite to any scene understanding, so it's possible to choose to implement a generic-purpose tracking module (as our group did). This mid-level layer offers to the decision-making module the strong abstraction of *object* instead of the weak abstraction of *blob*.

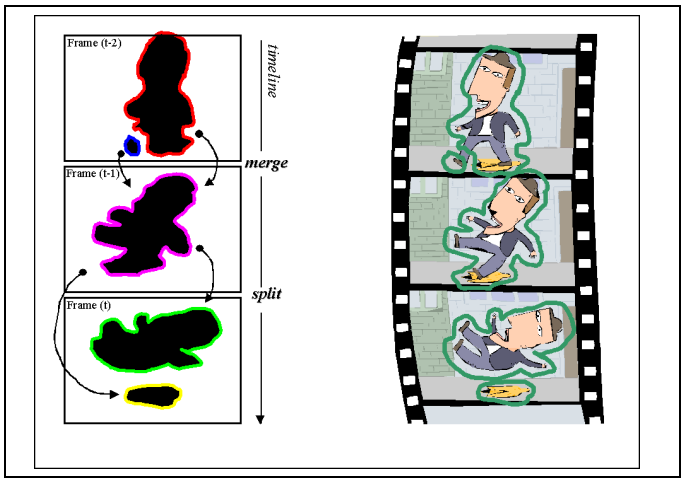

Figure 1.3: The Tracking job: to relate multiple objects during time.

Def. *"An **Object** is an entity that (1) entered into a scene in a* continuous and limited *time window, (2) in a limited and compact zone, and (3) from then on it was always found in a limited and compact zone" (see [4])*

This definition is general and has no semantics applied to it: a group of people is a single object as long as it takes birth, dies and moves as a single unity; if the group splits and two subgroups take 2 significantly different trajectories, it will probably be considered (it depends on the assumptions and implementations) as two objects; this *may* be retroactive.

This difficult task is achieved by keeping the whole history of blobs' births and deaths and relationships between pairs of objects (merge/split). The frame-by-frame decisions are taken by matching the new blobs with the old

objects in the neighborhood. The tricky part is to correctly associate more than one blob to a single object (or viceversa) by correctly discriminating noise from 'positive signal'. This is difficult because the acquisition can tear a single person in two different connected parts for a short while then put it all together as the scene evolves (see fig. 1.3, pag. 5). Often a choice is to be made at the very first frames when the object is born; and backtracking should be rare.

Another big trouble in this module is the correcting tracking during *occlusions*, when more objects intersect themselves making it difficult to distinguish between one and another. Some systems keep tracking objects during partial occlusions; some can even manage total occlusions. In general, the more accurate the model, the more specific it is: it has to do many hypotheses and to apply much a-priori knowledge in order to correctly deal with difficult cases.

Module 3: Decision Making

The previous levels were layers that offered to this level the abstraction of single moving object. This level *must* be specialized in some way: i.e. to detect traffic scenes (objects are vehicles), parking lots (objects are typically vehicles or pedestrians) and so on. The first step of this module is a *classifier*: this is important in order to correctly interpret a scene. When the objects are labelled as belonging to a given taxonomy of classes, another level of understanding can be achieved: the interaction between object gotten by the tracking module (person enters car, 2-person rendezvous, and so on).

1.2 Objective of the work

In all video surveillance tasks a good blob-detection module is highly desirable for a good elaboration. Among the problems of detection, the shadow-removal is yet an unresolved issue; many techniques have been proposed to remove shadows but they mostly rely on manually tuning parameters which are critic to the task. No work studied by our group can automatically tune these parameters. We tried to study the phenomenon and to produce some heuristics to tune them.

The aim of this work is to produce (and test) an algorithm that can automatically detect the parameters that give the 'best' possible shadow removal (or, at least, a good one) without the help of a human operator. This constrain is a must in many environments: in many occasions it's preferable not to remove shadows at all than to accept

the need for constant human maintenance.

The algorithm must prove robust in many environments, though it does not have to be fast, as it can just run in a bootstrap phase (although it has to be light in the *upkeep* phase).

1.3 Organisation of this work

In the second Chapter, vision models found in literature will be discussed and compared to the ones used in the system; new ones will be introduced, too. In the third chapter, the full *iter* of auto-tuning shadow suppression will be followed (models, algorithms, and experimental proofs). In the fourth, some description is provided about the C++ classes that implement this work. In the fifth chapter, conclusions will be thrown along with some possible future work.

Chapter 2

Models used and Related Research

2.1 Shadows

In this section some theory about light and shadow will be explained, as well as the basic assumptions most researchers do when dealing with shadows.

2.1.1 Introduction on shadows

Shadow derives from the occlusion of light source by an object. There is a discrimination between the shadows belonging to the object (*self-shadows*) and projected somewhere in the scene (*cast-shadows*).

In a Video Surveillance task, the shadows can either be *static* (if they belong to a still object, and they are supposed not to move in time) or *moving-shadows*; the latter are simpler to detect (as a background subtraction module exalts these zones as well as moving objects).

A cast shadow can be *attached* to the object projecting it or not.

The main goal of shadow suppression (in our research group) is very important to keep in mind, as it justifies many choices taken in the shadow model. Our first aim is to correctly segment moving objects; the most dangerous shadow is, for instance, one that wrongly connects two separate objects; if a shadow is just a little stripe aside, it can simply be left right there.

We tried to remove just *moving shadows* supposed *attached* to the shadowing body.

In computer vision, much has been done to correctly describe this phenomenon; complex light models are taken into account at a first glance, then reductive hypotheses are assumed in order to justify choices that greatly simplify the algorithms involved (or just make them feasible).

A couple of definitions must be made to better understand some models.

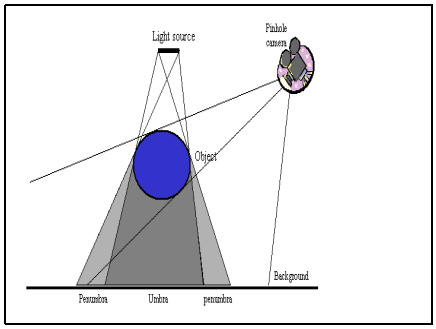

Figure 2.1: Penumbra: a simple physical explanation. (**DEBUG**: nome fotoric: '*02umbrapenumbraparbuona*')

Penumbra is the part of the shadow (typically in the outer contour of the shadow) where light (that has a non-null extent) is just *partially* occluded by the object. **Umbra** is the remaining part of the shadow (typically central), where light is totally occluded.

A penumbra zone tends to have a constant gradient in the luminance ratio between lit and occluded spot (because the percentage of incident light is roughly linear from the beginning to the ending of the penumbra 'stripe'). On the contrary, this ratio tends to be null in the umbra core.

In our video surveillance applications, geometric optics is sufficient to describe the distribution of light (as wavelengths haven't to be accounted for). The appearance of a cast shadow can be described as a temporal signal.

$$s(x, y, t) = E(x, y, t) \cdot \rho(x, y, t) \tag{2.1}$$

This formula can be read as: luminance in the point (x, y) at time instant t is the product between the irradiance E and the reflectance ρ of the object surface. The irradiance E is the amount of light power per object area. As it can be seen from fig. 2.2, it is a function of (1) the angle between the normal to the surface being lit ($\mathbf{N}$) and the direction where the light comes from ($\mathbf{L}$); (2) of the intensity of the source light (c_P) and (3) of the intensity of ambient light (c_A):

$$E(x, y, t) \quad = \quad \begin{cases} c_A + c_P \cos \angle(\mathbf{N}(x, y), \mathbf{L}), & \textit{if illuminated} \\ c_A + \alpha(x, y) c_P \cos \angle(\mathbf{N}(x, y), \mathbf{L}), & \textit{if penumbra} \\ c_A, & \textit{if umbra.} \end{cases} \tag{2.2}$$

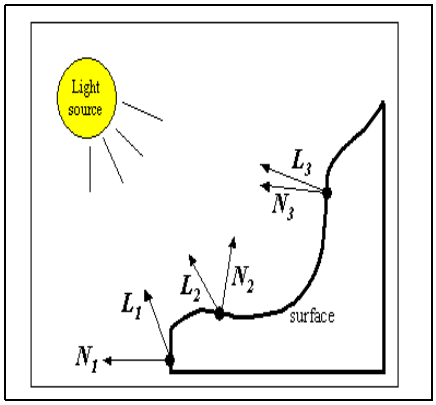

Figure 2.2: An example of vectors $\overrightarrow{\mathbf{N}}$ and $\overrightarrow{\mathbf{L}}$ in multiple points.

The term $\alpha(x, y)$ $(0 <= \alpha <= 1)$ describes the transition inside the penumbra between umbra and illumination. The intensity c_P of the light source is proportional to $1/r^2$, with r being the distance between the object and the light source.

In order to simply cope with shadows, many hypotheses are to be made. In the works many common hypothesis are taken ([1], [2]):

Single Source of Light Among the many sources of illumination in the scene, just one is accounted for, and the other are supposed negligible with respect to it.

Strong Illumination changes The difference in illumination between a covered spot and an uncovered one is considerable. This justifies the threshold on frame difference between lit and unlit spot.

Static camera The camera is assumed to be static, so any frame difference would reveal real movement and no apparent movement.

Static background This implies that any apparent movement belongs to objects in the foreground (no '*waving leaves*' possible);

Textured background This assumes it's easy to detect foreground from background by means of segmentation and to detect shadow (for it generally alters the luminance but not the texture).

Plane background The surface normal is supposed roughly constant throughout the background . This greatly simplifies light models as it implies illumination changes due to moving cast shadows are smooth and that $(\cos \angle(\mathbf{N}, \mathbf{L}))$ is constant (see [1]).

'Wide' light The light source has a certain *extent*: shadows have a *penumbra*.

2.1.2 Assumptions made in this work

In this work some assumptions are taken for granted on shadow and background models:

- *Self shadows* are neglected; shadows are supposed to be *cast shadows*.
- *Penumbra* isn't modelled distinctly from *umbra*; penumbra behavior will be implicitly considered, though, when considering shadow segmentation while varying α (see [2]).
- Objects are opaque; no light reflection is considered.
- Single Strong Light assumption is taken for granted; this is *very* important for validating mathematical models on shadows. In general, all systems dealing with shadows assume just two light sources: a strong direct light (that may be occluded or not) and a diffuse light that permeates the environment (if that wasn't present shadows would be utterly *black*).

2.1.3 Related Work

Previous works have been analyzed on the shadow removal topic. A few words will be spent about them, as they partly inspired this work.

In [3], moving objects in traffic scenes are detected mixing statistic and knowledge-based approaches in background update. Background is modelled through a short time-window history. Interestingly, mean values are discarded (for lack of robustness) and the mode value is used instead of median for computational ease. Segmentation heavily exploits Optical Flow (which is, renowned to be much time consuming). Knowledge has to be applied in order to overcome OF's intrinsic limits (ghosts, uniform objects, ...).

In [2], HSV color space is used instead of the common RGB. It is considered closer to a 'natural' representation of the shadows. Some ideas inspired part of this work, as for instance the constancy of the ratio in a strict neighborhood.

In [1], a short introduction to luminance theory is given. It's been used *as-is* in this article. Interesting models are used and justified to detect *umbras* and *penumbras* (simplified as *linear* transitions from umbra and lit places).

2.1.4 Models introduced

In this section new models will be presented that were introduced to better manage shadows.

The Direction of the Light Source

Systems that manage shadows often suppose (see sec. 2.1.1, pag. 10) that a single light source is present in the scene. This is important in order to obtain a simplified model of the luminance field.

The light source univocally determines the direction of the cast shadow of an object, providing it's joint to the object itself. With this simplification, knowing the source of the light let us know the direction where to *expect* the shadow. Assuming the light source is rather constant during the time, although it can change slightly during time (as the Sun, for example), we can find the '*sun direction*' (as it will now on be called) by feedback, then updating it slowly in order to adapt to eventual changes in the position of the light source.

The drawback is in the feedback mechanism: we need to *already* know how to find shadow, and then use this information to create a robust model of where the light comes from, then we can use this information in order to better find and segment shadows.

The hypotheses to be made in order to apply this model are very strong: a sufficient condition for the model to be correct is to suppose the background plain and to suppose the object to have a vertical, positive extent. Then, the shadow direction is opposite to the light direction.

Given a 'tritonal' image $I(x, y)$ (with three possible labels: B (background), F (foreground), S (shadow)), we can define a simple 'shadow matrix' $\mathcal{M}_s$:

$$\mathcal{M}_s \doteq \begin{pmatrix} N_{135^\circ} & N_{90^\circ} & N_{45^\circ} \\ N_{180^\circ} & 0 & N_{0^\circ} \\ N_{215^\circ} & N_{270^\circ} & N_{315^\circ} \end{pmatrix}, \left(= \begin{pmatrix} N_{NW} & N_N & N_{NE} \\ N_W & 0 & N_E \\ N_{SW} & N_S & N_{SE} \end{pmatrix} \right) \tag{2.3}$$

or:

$$\mathcal{M}_s \doteq \begin{pmatrix} N(-1,1) & N(0,1) & N(1,1) \\ N(-1,0) & 0 & N(1,0) \\ N(-1,-1) & N(0,-1) & N(1,-1) \end{pmatrix}, \tag{2.4}$$

where the former (eq. 2.3) is in '*directional* form' and the latter (eq. 2.4) in '*cartesian* form'. They are obviously equipollent; for mathematical ease

the second will be used. Note the $M_{rc} = N(c-1, 1-r)$[1].

A possible definition of M_{ij} is (using the eq. 2.4):

$$N(i,j)[\mathcal{I}] \doteq card\{(x,y) \in \mathcal{I} \,\Big|\, \mathcal{I}(x,y) = F, \mathcal{I}(x+i, y+j) = S\} \tag{2.5}$$

From now on, foreground, background and shadow pixels will be called **FP**, **BP** and **SP** respectively. As one can argue from eq. 2.5, $N(\alpha)$ is the occurrence of OPs *8-connected* to SPs along the chosen direction. Another way to strengthen the computation is:

$$\hat{N}(i,j)[\mathcal{I}] \doteq card\Big\{ \quad (x,y) \in \mathcal{I} \,\Big|\, \mathcal{I}(x,y) = F \wedge \mathcal{I}(x+i, y+j) = S \;\wedge\; \mathcal{I}(x-i, y-j) = F \quad \Big\}. \tag{2.6}$$

This second definition associates to each direction the occurrence of OPs connected to SPs along the chosen direction α *having another OP in the opposite direction* $(180° + \alpha)$. This makes the calculation more robust as it prunes 'isolated' (yet lucky) points from the matrix. This second version was the one used in the implementation of the algorithm.

This matrix has to be used in some way. Two important informations can be extracted from this matrix: the simplest way is to compute the direction of the light.

$$\vec{\mathbf{d}}_{light} \equiv \begin{bmatrix} d_x \\ d_y \end{bmatrix} \doteq \begin{pmatrix} \sum_j N(1,j) - \sum_j N(-1,j) \\ \sum_i N(i,1) - \sum_i N(i,-1) \end{pmatrix} = \begin{pmatrix} \sum_{i,j} i \cdot N(i,j) \\ \sum_{i,j} j \cdot N(i,j) \end{pmatrix} \tag{2.7}$$

[2]

The vector $\vec{\mathbf{d}}_{light}$ wraps two different informations: its *direction* is the 'sun direction' we were searching for; its module quantifies in some way the number of pixels voting for that direction: it seems useless so far.

Experiments show that this model is very good in order to detect the main light direction in the condition of outdoor vehicle tracking. In this hypothesis the shadow is always attached to the vehicle; the junction between object and its shadow is typically wide. This is very important for the algorithm to work well. The concern with tall objects with little horizontal extension (such as

[1] r stands for row, c for column. Furthermore, i and j range from -1 to 1.

[2] An interesting research branch could be considering the properties of the 'second momentum' matrix defined as: $M_2 = \begin{pmatrix} \sum i^2 \cdot N(i,j) & \sum ij \cdot N(i,j) \\ \sum ij \cdot N(i,j) & \sum j^2 \cdot N(i,j) \end{pmatrix}$. This wasn't developed at all, for time lack. Interesting properties could be the *determinant* and the *direction* obtained considering M as an inertia matrix.

standing people) is that the shadow is computed by scanning the boundary between the object and its shadow: if this is very short with respect to the two areas the result is very subject to noise and one pixel or two can easily alter the result by $45 - 90°$.

The **computational effort** of this algorithm is quite the same as a 3x3 blur: in the actual implementation, all FPs are scanned and for each of them the neighborhood is processed and the matrix filled. For big areas, it could be convenient to extract the border of the foreground areas and just process the borders. However, in real scenes with plenty of noise it seems less convenient than the used approach.

Another idea that will come useful for the core of the shadow removal algorithm is the goodness of the direction d. It comes from a simple concept (that will be explained in more detail later).

The algorithm is robust towards noise. If a false SP is added in the middle of a FP area (that is, changing a FP - surrounded by 8 FP - to SP), as one can easily behold, the matrix will grow of 1 in each of the 8 directions. Symmetrically, if a false FP is added in the middle of a FP, the same happens. This is not true if the definition (eq. 2.6, pag. 13) is used instead, but the concept holds: direction computation is robust towards small isolated changes in the middle of an area. This will prove useful in the automatic tuning of the shadow removal.

From the previous reasoning, we can try to estimate the goodness of the matrix. For every *noise* point, the direction vector doesn't change (*not* just its direction α, which is less strict), but the sum $\mathcal{S}$ of the eight values $N(i, j)$ does: the lesser $\mathcal{S}$, the greater the goodness of the acquisition. Why bothering about the correctitude of the classification of SP and FP? Because the algorithm was projected to work in *feedback*, so that it could tune to a good classification providing a catch-all parameter initialization and a good heuristic function in order to move the parameters in the right direction.

Another interesting property of robustness is that the $\vec{\mathbf{d}}_{light}$ is immune to deformation of the contour. This is both true in the case of continuum, from a theoretical point of view, and in the discrete case, where the quantization noise is not null. This can be easily shown. The vector $\vec{\mathbf{d}}_{light}$ can be thought as the integral along the contour Γ between FPs and SPs of the single contributions of direction along the contour, which is the normal to the trajectory $d\mathbf{r}$, where $\mathbf{n} = \mathbf{k} \wedge \mathbf{r}$:

$$\vec{\mathbf{d}}_{\mathbf{light}} = \int_{\mathbf{r}\in\Gamma} d\vec{\mathbf{n}}(\mathbf{r}) = \int_{\mathbf{r}\in\Gamma} d(\mathbf{k} \wedge \mathbf{r}) = \mathbf{k} \wedge \int_{\mathbf{r}\in\Gamma} d\mathbf{r} = \mathbf{k} \wedge (B - A), \qquad (2.8)$$

where A and B are respectively the starting and the ending point of the trajectory Γ. So, it's shown that changing the middle of the contour doesn't

change the result as long as the two extremes don't change. For a correct verse of $\vec{\mathbf{d}}_{light}$, moving from A to B the contour should have on its *left* the object and on its right the shadow (as the cross product rotates $\mathbf{r}$ 90° on the left). With this hypotheses, $\mathbf{d}$ represents the provenance of the light source, whilst $-\mathbf{d}$ represents the very direction of the light (which is the opposite).

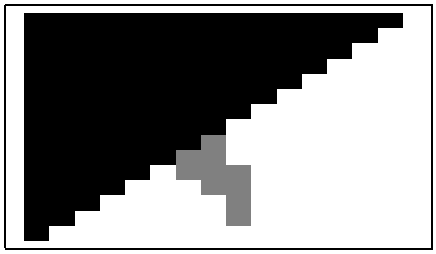

Figure 2.3: An example of the effect of a perturbation (in gray) of a shadow region. In black, the FPs. In white, the SPs. In gray, the FPs added for the differential approach.

In the discrete case, as we can see from the simple example in fig. 2.3 , the variation in the matrix is simply the difference betweeen the number of occurrences of $N(\alpha)$ the 'extra limb' adds (δM^+) and the number of occurrences it subtracts (δM^-).

$$\delta\mathcal{M}_s = -\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 0 & 2 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 7 \\ 4 & 0 & 6 \\ 7 & 3 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 7 \\ 4 & 0 & 4 \\ 7 & 1 & 1 \end{pmatrix} \tag{2.9}$$

For linearity, $\delta dlight$ can be easily found with the original formula:

$$\delta\vec{\mathbf{d}}_{light} = \begin{bmatrix} \delta d_x \\ \delta d_y \end{bmatrix} = \begin{bmatrix} 7+4+1-1-4-7 \\ 1+1+7-7-1-1 \end{bmatrix} = \mathbf{0}. \tag{2.10}$$

It could probably be proven that the result doesn't change if the two extremes are kept as they are.

2.1.5 Goodness definition

From the previous example, one can argue that: (1) the measure of the $\vec{\mathbf{d}}_{light}$ doesn't change if the edge between SPs and OPs is changed as long as A and B remain unchanged, and (2) noise pixels don't change the measure. The objective is to exploit Light Source Model to find out a measure of this noise: this is the goodness .

Goodness η is defined as follows:

$$\eta \doteq \frac{||\vec{\mathbf{d}}_{light}||_2}{||\mathcal{M}_s||_1} = \frac{|\vec{\mathbf{d}}_{light}|}{\sum_{i,j} N(i,j)} = \\ = \frac{\sqrt{d_x^2 + d_y^2}}{N_{NW} + N_N + N_{NE} + N_W + N_E + N_{SW} + N_S + N_{SE}}. \quad (2.11)$$

As N_{ij} are non-negative, it can be shown that η is bound to the range $[0, 1]$. More interestingly, this value is affected by noise pixels and decreases as noise points grow up. If a single SP is added in a FP zone (or viceversa), the matrix $\mathcal{M}_s$ will grow by 1 in all directions; $\vec{\mathbf{d}}_{light}$ won't change at all, while the denominator in eq. 2.11 will grow by 8.

The main property of this definition is that, if two images share the same $\vec{\mathbf{d}}_{light}$, their goodness discriminates the 'fitness' of those decisions. Let's see an example:

$$\mathcal{M}_1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 18 \\ 0 & 18 & 36 \end{pmatrix}; \mathcal{M}_2 = \begin{pmatrix} 1 & 1 & 7 \\ 4 & 0 & 22 \\ 7 & 19 & 37 \end{pmatrix}; \mathcal{M}_3 = \begin{pmatrix} 10 & 10 & 10 \\ 10 & 0 & 28 \\ 10 & 28 & 46 \end{pmatrix} \quad (2.12)$$

This three matrixes represent (in order): the shadow matrix of the black straightforward path in fig. 2.3 and the shadow matrix of the gray-perturbated path; the third is the same as the first, with 10 random noise pixels added somewhere. It can be deduced that:

$$\mathbf{d}_1 = \mathbf{d}_2 = \mathbf{d}_3 = \begin{bmatrix} 18 \\ 18 \end{bmatrix}; ||\mathcal{M}_1|| = 72; ||\mathcal{M}_2|| = 98; ||\mathcal{M}_3|| = 152 \quad (2.13)$$

So, the goodness for the matrixes in the example are:

$$\eta_1 = 35,36\%; \;\; \eta_2 = 25,98\%; \;\; \eta_3 = 16,75\%. \quad (2.14)$$

It has been shown that goodness is sensible in some way to noise and alteration in the straightforwardness of the path from A to B.

This parameter will be heavily used in the shadow removal algorithm.

2.2 Alpha and Beta definition: a needed note

The focus of this work was to automatically tune up the most critical parameters in shadow removal tasks. These were acknowledged to be α and β in the MSP test.

Note that in most works ([2], [1]) alpha and beta have different semantics from our work's. We'll call α_{SH} and β_{SH} the one used in our work (inverse form), and α_{norm} and β_{norm} the ones used in normal form. It's easy to compute the translation from normal to inverse form from α_{SH} to α_{norm} (the same method applies to β):

$$\alpha_{norm} = \frac{254}{1 + \frac{\alpha_{SH}}{64}} \tag{2.15}$$

So, for instance, 'our' 0 would be mapped to 254, 254 to 26 (the limit of our model), and so on.

What do alpha and beta represent? They are the limit to the possible ratio between the current frame value and the unoccluded value (taken from the background). During the experiments, it was noted that α_{SH} and β_{SH} had two interesting behaviors:

- Starting to move α_{SH} from 0 up, a first (positive) effect is that false OPs get filled with true BPs; then, after a given value, a (negative) effect happens: some BPs begin being considered as FPs. This effect begins from the border of the shadow opposite to the connection between the blob area and the shadow area. This property will be heavily exploited in the algorithm of automatical alpha tuning.

- Moving β_{SH} from 255 down has a similar initial effect, as it changes some false OPs into FPs; when it overcomes its optimal value, another behavior is experienced: it tends to erode the borderline between object and shadow, in the direction of decreasing shadow area. This property was very difficult to use, and most experiments in this direction failed.

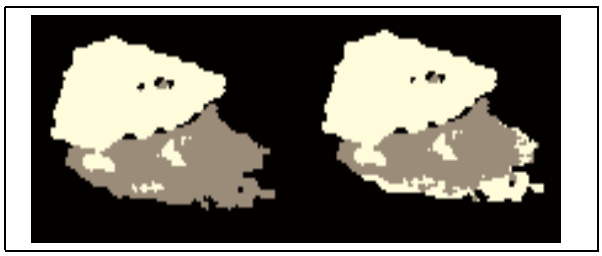

Figure 2.4: An example of different alpha shadow classification: on the left, α_{SH} is too low; on the right, it's too high.

Chapter 3

Adaptive Tuning of α and β

In this section the main experiment will be presented, along with each aspect of the problem: theory, models, choices, implementation, and *experimental proofs*. For all data, a PC was used with the following parameters: Pentium III 1000 MHz, 512 MB RAM, with MS Visual Studio 6. C++ was used along with Intel's **ipl** [10] and **openCV** [11] libraries.

3.1 Algorithm overview

The adaptive alpha-tuning algorithm works in feedback ; its block scheme is represented in fig. 3.1.

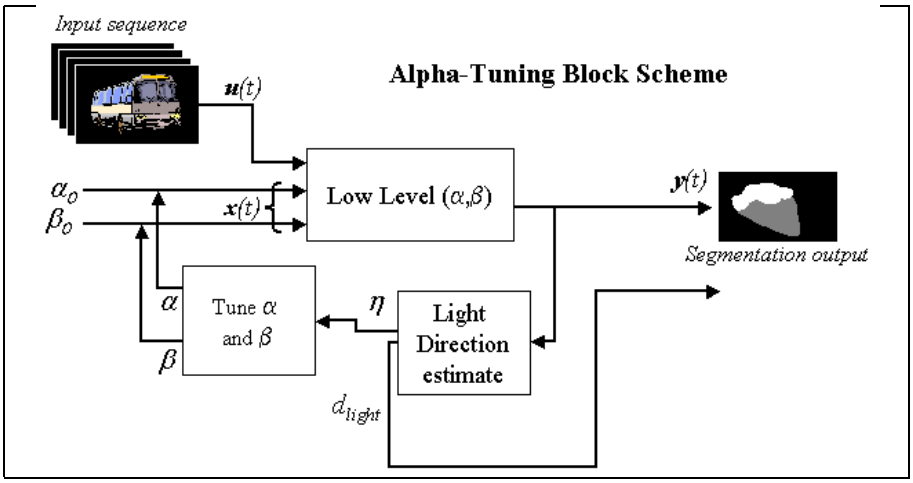

Figure 3.1: The block scheme of the alpha-tuning algorithm.

Concept:

- Images are processed by a simple low-level algorithm (described later in this chapter) that is parametric in (α, β). Its starting values will be (α_0, β_0); these values are equal in all sequences and are chosen in order not to prevent shadows from being detected: they are quite *lask* as we prefer computing false SPs than losing true SPs.

- The LL module returns a tritonal image (BPs, FPs, SPs). With this partition a simple 'sunlight' algorithm estimates the direction where the main light comes from (see DIREZIONELUCE). This algorithm also provides the goodness of the measure. This goodness is a measure of how good the partitioning algorithm was.

- With this information, a better (α, β) can be extracted and used in the next frame, and so on.

This paradigm is just a concept, as the real implementation was *not* in feedback.

Implementation:

- In a ***bootstrap phase***, images are processed by the low-level algorithm varying (α_0, β_0) (in practise, only α was moved throughout its full range, from 0 to 255, thus doing a *complete* scan on α. For robustness, this scan is made for many (10 right now) frames.

- Once the function $goodness(\alpha, t)$ is at disposal, the result is averaged through time t, so it becomes $\widehat{goodness}(\alpha)$. Then its graph is analyzed and the optimal value for α is found, looking for a value just before a sudden fall in goodness . For further explanation, see sec. 3.3.1, pag. 22.

- Then, the ***real acquisition*** phase begins; the optimum α was already found. From now on, the system works 'statically' with the same values.

Note. The *bootstrap phase* is rather expensive, but it has to be done just once, and this cost is acceptable. On the other hand, this algorithm should adapt during time, as light direction, intensity and existence itself drastically change during 24-hour time lapse. A simple 'refresh' module could be distributed over time (for little CPU usage) and slowly adapt to light changes, but this direction wasn't studied at all; it seems feasible, though, and quite easy to implement.

3.2 Low Level implementation

In this section the first module's (i.e., low level's) structure will be treated and analyzed without code details. In fig. **??**, a simple block scheme is presented. The input is processed as an AVI. A function (`elabImage()`) takes the n-th image (which will be now on called *current image*, or simply *current*), outputs background and other static models, and returns a tritonal image in which SPs, FPs and BPs are classified.

Each step will be presented in the next subsection, underlining the interface between blocks.

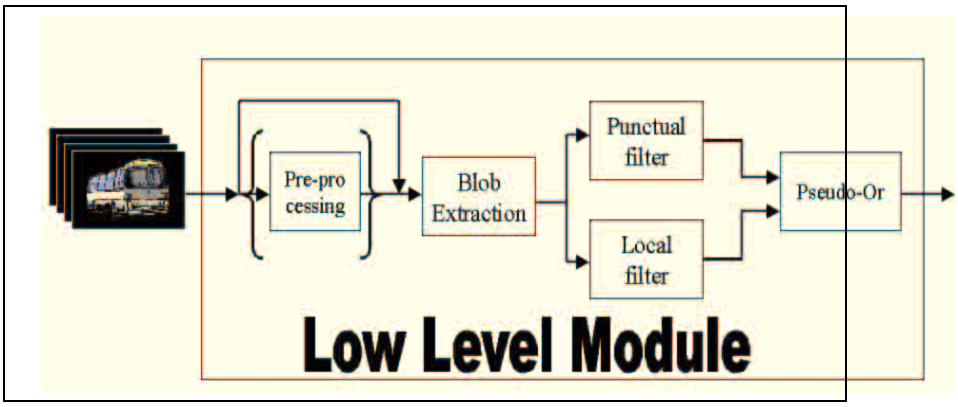

Figure 3.2: A simple block scheme of the Low Level module.

3.2.1 Video Sequence Processing scheme

The scheme presented in fig. **??** will be now analyzed in more detail.

For simplicity purposes, a static background was used, acquired in a moment where no objects were present in the scene. A simple description is provided here; more on shadow identification issues is provided in the next subsection.

Note. *When talking about ratios, ratio between the value in the current frame divided for the value of the background in that point (recorded some frames before). Usually, ratios are intended between low values and high values, in order to obtain a normalized value (0..1).*

0: Preprocessing In this phase, a simple preprocessing is applied both to the background and the current image. For instance, noisy sequences will be pre-blurred, big scenes shrank and so on.

1: Blob Extraction In this part, a simple *color background subtraction* technique extracts the foreground from the background . Foreground will be then subdivided in SPs and real FPs. It's important to note that what this module considers background will be always considered as such: no change of mind in the following module is implemented in this simple framework. The algorithm is simple: for each pixel, the maximum difference between foreground image and background image over the 3 channels (R,G and B) and if it's more than a fixed threshold (about 25/255) it's object.

2a: Punctual filter In this module shadows are discriminated from object pixels on a per-pixel basis: points identified as MSPs (see sec. 3.3.2, pag. 25) are then filtered with a **constant color ratio** reasoning.

2b: Local filter MSPs are here filtered with a local shadow classification pattern. In a shadow, the ratio should be roughly constant in a small neighborhood. For each pixel, the mean value of the ratio image is computed (just like a blurring) and confronted with the pixel value: if they differ too much, the point is considered object. A not-MSP is *always* considered FP.

3: Pseudo-Or (Merging module) This module simply merges the results of 2a and 2b modules. It was initially implemented as an OR (using easy parameters on the former ones); now it's a **voting neighborhood function**. In fact, for each pixel, a neighborhood of dimension $3x3$ is analyzed and the occurrences of BPs, FPs, SPs are kept in 3 variables. The result is the most frequent value, with some exceptions: in practise BPs are weighted less so that it's easier for FPs or SPs to 'win'. This was made in order to fill little empty spaces between foreground areas and shadow areas; this was made to make more robust the 'direction of the sunlight' computation (for more detail, see sec. 2.1.4, pag. 12): please note that its definition regards moving through a borderline between object area and shadow area, and any one hole weakens the computations...

Modules 2a and 2b were imported from a previous work by M. Mola.

Notes on ratio definition

A key concept in shadow-detection is computing the *ratio* between two images. As image processing is a time-consuming and CPU-critical task, some simplifications are used to optimize speed.

The quantized model used herein for *Ratio Image* description will be now discussed in some detail. Ipl images were used with 3 8-bit channels. A ratio image was computed and used in the 2a/2b modules; this ratio was a 8-bit gray-level image. The correspondent value $v_q \in [0..255]$ is (keep in mind that in the original frames 0 is black, 255 is white, and dark values are lesser than light values):

$$\rho_H \doteq \frac{P_{light}}{P_{dark}}; \qquad f(\rho_H) \doteq 64 \cdot (\rho_H - 1) \tag{3.1}$$

$$v_q \doteq \begin{cases} 0, & \rho_H < 1.01 & \textit{(shadow lighter than original)} \\ f(\rho_H)(\in [1..254]), & \rho_H \in [1.01, 4.97] & \textit{(normal values)} \\ 255, & \rho_H > 4.97 & \textit{(too dark)} \end{cases} \tag{3.2}$$

256 bins were used in a smart way: if the MSP is lighter than the original object, it deserves just a symbol (0) to represent it: it won't be considered a shadow. A maximum darkness value was decided (≈ 5), beyond which all ratios are considered equal; and the remaining levels (between 1 and 5, approximately) were linearly divided in equal bins. This legacy was given from the previous algorithm by Mola. Another possible (more used in literature and perhaps more natural) correspondence could be:

$$\rho_L \doteq \frac{P_{dark}}{P_{light}}; \tag{3.3}$$

$$v_q \doteq \begin{cases} 255, & \rho_H > 1 & \textit{(shadow lighter than original)} \\ 254 \cdot \rho_L(\in [0..254]), & \rho_H \in [0, 1] & \textit{(normal values)} \end{cases} \tag{3.4}$$

In the former, normal values for shadows would result in numbers ranging from 1 to inf; in the latter the range would be $[0, 1]$ instead, as P_{dark} and P_{light} were exchanged. The latter will be now on called 'normal form'; the first will be called 'inverse form'.

3.3 Alpha tuning

The aim of the experiment is to find the best α_{SH} while not knowing the best β_{SH} yet.

3.3.1 The algorithm

Many AVI scenes were processed in order to have a rich benchmark to work with. In a first moment, these scenes were manually analyzed and eventually

some pre-processing was set for the trickiest scenes. Then, a simple ground truth was extracted by moving alpha from 0 up to an optimal value. *Then*, beta was moved from 255 down to its sub-optimal value: because it was difficult to obtain an optimal value, a 2-way cheap approach was preferred (two 256-long cycles) to a 1-way costly approach (one 65536-long cycle). This is more inaccurate, but it's easier to compute for a human and better reflects the computer algorithm. A strong alpha property is described in sec. 2.2, pag. 17: the tuning of alpha from 0 up to 255 begins filling false negative SPs, then - after the optimal value - tends to erode the shadow area in the external part. Recalling the goodness formula, the reaction of goodness to these 2 behaviors has two different impacts:

- In the first part ($\alpha_{SH} < \alpha_{SH}^{opt}$) the direction $\vec{\mathbf{d}}_{light}$ shouldn't vary much (points filling) whilst the goodness should rise.
- In the second part ($\alpha_{SH} < \alpha_{SH}^{opt}$), where SPs belonging to the outer border get eroded, the direction should begin varying and goodness should crash down: the impact on the matrix would be to 'pump up' values opposite to the 'winning' ones. This effect doesn't usually overcome the true value of $\vec{\mathbf{d}}_{light}$, but is enough to significantly lower the goodness .

The experiment consisted of moving α_{SH} from 0 to a maximum value (not necessarily 255; 200 has been used). *A simple full iteration was used, although it could be better for performance purposes to apply a dichotomic search.* A graph of $goodness(\alpha_{SH})$ was extracted and inspected. An ideal behavior would be: slowly decreasing to a fixed value α_{opt}, then abruptly fall down, then keep quite constant. The researched value is then α_{opt}. In fig. **??** some graphs can be analyzed.

The true implementation was divided into the following steps: divide the function $goodness(\alpha_{SH})$ in bins, in order to reduce the effect of noise; find the maximum bin B_{max}; then, find in the vicinity of B_{max} the greatest negative variation between two adjacent α values (to detect when the function falls with highest derivative). The computed value is α_{opt}. The case of 'α not found' is also considered: if the maximum bin is found very late in the diagram (see for example the first image in fig. **??**: a value was strictly found, just because we were forced to find one).

For robustness, the algorithm was not applied to a single frame; in practise, the function was computed for N[1] frames and then the algorithm was

[1]About 10 was chosen for N. It varied from scene to scene, depending on the dimension (width x height) of the images.

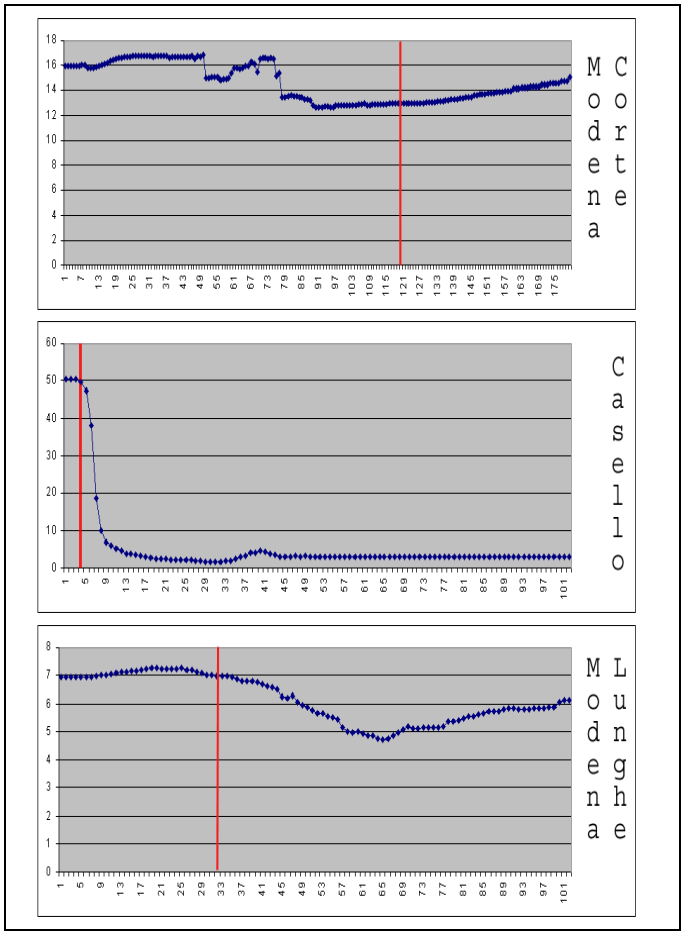

Figure 3.3: The algorithm of alpha tuning is tested in three sequences and confronted with the optimal value.

applied to average values of $goodness(\alpha_{SH})$. They were computed in moments where there was some motion. One may object that in real cases this can't be done; if it had to be done automatically, though, it would be sufficient to integrate the values for a long time. If a scene hasn't got a FP, there won't be shadows at all, so the function $goodness(\alpha_{SH})$ would be flat (null derivative) and, for linearity, this wouldn't have effect upon the result. A simple correction could be added: weighting the function with the number of FPs in an image; this would drastically reduce the impact of 'dead scenes' on the result.

3.3.2 Shadow identification criterions

Some Shadow identification criterions are described in more detail: 'MSP-ness', punctual shadow decision, local shadow constancy and the voting neighborhood function.

MSP A point is MSP ('*Maybe Shadow Point*') if and only if the ratio between the luminance in the point of the current frame $CF(x, y)$ and the background $BKG(x, y)$ lies in a specified range $[\alpha, \beta]$[2].

punctual (constant color) ratio test A MSP point is considered a shadow if the red, blue and green ratios are similar to the luminance ratio; that is, if $max_{ch\in\{r,g,b\}}|\rho_{gray} - \rho_{ch}| < \delta_{TH}$. This is a punctual relation.

local test This is a local relation, as it regards just a fixed neighborhood of the examined point, just like *blurring*.

pseudo-or This function was projected with the aim to remove blank points (BPs) between FPs and SPs, in order for the 'sunlight algorithm' to work better[3].

3.3.3 Benchmark analysis

It's important to spend some time upon the scenes that were observed: each of them presents some difficulties and recurring characteristics (dark and uniform vs light wide shadow, good vs noisy foreground detection, ...)

In the following subsections a description for each sequence observed is provided, along with a picture; this picture always contains 2 or 3 images. If they're 3, they are (from top to bottom, for a sample image extracted from the sequence):

1. the original RGB image.
2. the tritonal output for that image computed with ground truth (α, β)
3. the tritonal output for that image computed with auto-extracted (α, β) (in fact, just α was extracted, along with the ground truth β value).

Visually, FP are white, SP are gray and BP are black.
If just two images are present, an explanation is given in the text.

[2] see sec. 2.2, pag. 16 for more details.

[3] This algorithm has proven weak in case the edge/borderline (luigi: preferisci davvero edge in questo caso?!?) between SPs and FPs were not well connceted. This is easy to understand if one thinks to the implementation of the algorithm.

camioncino This sequence proved very difficult to evaluate, as the (α, β) model failed for every possible couple to roughly approximate the real conception of FPs, BPs, and SPs. In fig. 3.10, just ground truth is provided: the algorithm didn't find a good α value to use (it always grew from 0 to 200). This is interesting because it's the only case in which no α was available.

vista2 This sequence presented the movement of students and car in front of the main entrance of Faculty of Ingegneria, Bologna. The value found was similar to the ground truth. A little displacement can be watched in the boundary of some shadows. This sequence had to be *preprocessed*, thus diminishing the sensibility of the algorithm to the single frames (as they had to be pre-blurred).

ingressoing This sequence is similar to 'vista2', as it focuses the same spot of Faculty of Ingegneria. The result of the algorithm is quite similar to ground truth. It's to note that the slab near the car provides both false positive and false negative FPs when it moves. This scene was noisy, too, so it had to be preprocessed.

vespa2 This sequence focuses on the passing of a motorcycle casting a wide shadow on the terrain. The shadow found is very similar to the ground truth (substantially the same).

casalecchio primo In this sequence (highway with moving vehicles, near Casalecchio, Bologna-Firenze) the background subtraction model was less robust than in other ones. The segmentation of vehicles wasn't good enough: as it can be seen, much of the vehicle is erroneously mistaken for shadow. The parameters found, though, fail as well as the ground truth parameters.

casalecchio secondo This sequence (another view of the same highway as 'casalecchio primo') presents the same flaws as the previous one. They both had to be preprocessed for a better segmentation.

casalecchio This sequence (yet another view of the same highway) was preprocessed, too. This is the sequence with the greatest discrepancy between the ground truth α and the auto-computed one. Regardless, the segmentation is bad (many parts of the landscape are mistaken for objects) and the partitioning of foreground in FPs/SPs is bad in both figures. The reason for a high difference between the two values was attributed to the failing of the α-tuning model: shadows in this sequence don't follow the rules described in sec. 2.2, pag. 17.

casello This is one of the most interesting sequences. It provides a view of a large truck approaching a highway's exit. The shadow is smooth, gradual and large. Apart from some segmentation faults (an arrow painted on the street is mistaken for object), the response of the algorithm is very similar to the ground truth.

Figure 3.4: The *"ingressoing"* sequence.

Figure 3.5: The *"vista2"* sequence.

3.3.4 Experimental Results

The main work was to automatically find (α, β) values from a scene with a parameter-tuning algorithm. During this 6-month time lapse, it was found that a good value of α could be extracted, while β was very tricky to evaluate.

In table 3.3.4, pag. 30, the ground truth values for α are provided along with the ones extracted by the goodness -algorithm.

In the last sequence, the algorithm couldn't find a value for α.

3.4 Beta tuning

Many direction were followed in order to cope with β. Some of them will be lightly discussed just to provide some cues for future works.

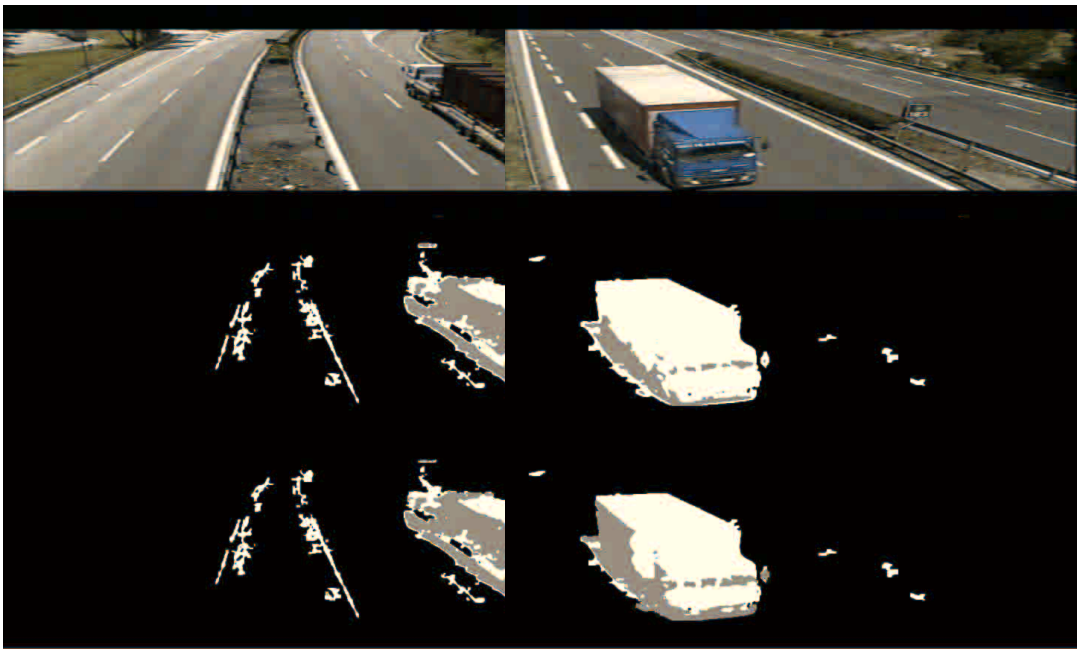

Figure 3.6: The *"primo"* sequence.

Figure 3.7: The *"secondo"* sequence.

An important fact has to be pointed out once more: α proved to be the most critical parameter to tune up, as it is the most sensible to change; this means that changing α from 1 to 10 or from 10 to 20 changed drastically the segmentation. The same is not true for β: a 254 value is veri similar to 200, for instance.

1. A similar approach to α was followed: to find a value for beta in which the goodness value changed abruptly. This nearly didn't happen at all. A good reason for that was found: while moving α beyond its optimal value created a false border in the shadow (thus changing qualitatively the shadow 'spot' allowing this fact to be caught), a different behavior was discovered for β; moving this parameter along its optimal value just eroded or expanded the boundary between object and shadow. This change wasn't perceived by the goodness algorithm, as it can be expected by its definition.

2. As β is little sensible, with respect to α, a simple classifier was tried. A function that roughly estimated $\beta = f(\alpha)$ was designed. A simple idea could be: dark shadows have low α and low β, and light shadows have high α and high β. This proved not to be true in all cases.

3. The same idea of the previous sentence (dark vs light shadows) was

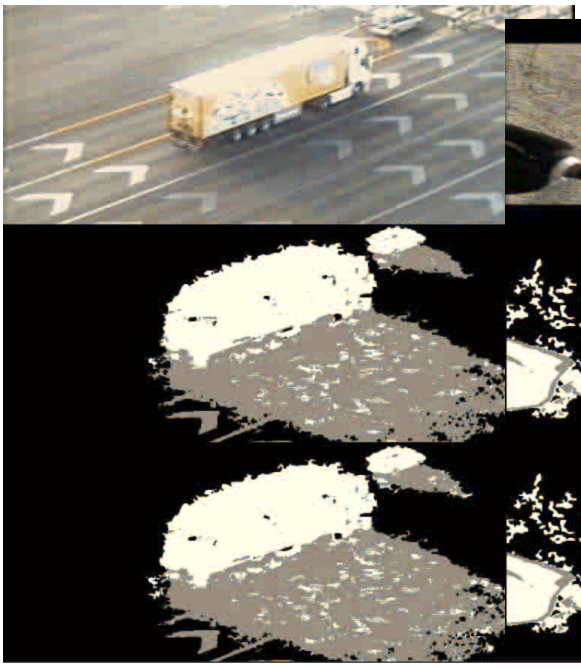

Figure 3.8: The *"casello"* sequence.

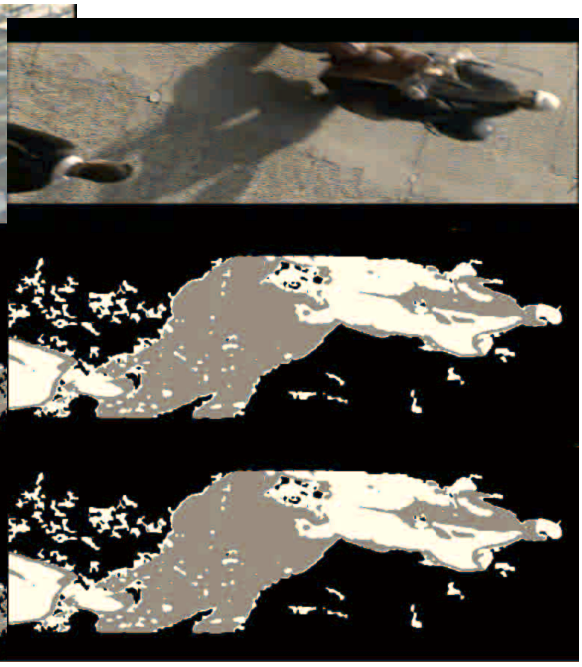

Figure 3.9: The *"vespa"* sequence.

studied in figure **histograms**. For each sequence, a histogram was extracted for the division image (each pixel meaning the ratio between the foreground image and the background image). This provided a 'spectrum' of the occurrences of intensity values for the shadows. This study, too, came to a dead end. The operative idea to extract value will be given, though: approximating the histogram to a single gaussian $g(\mu, \sigma)$, α and β could be estimated as $(\alpha = \mu - \sigma; \beta = \mu + \sigma)$. This wasn't robust enough so a *median* approach was preferred to a *mean* approach: first and third quartile were substituted to $(\mu - \sigma; \mu + \sigma)$.

Figure 3.10: The *"camioncino"* sequence.

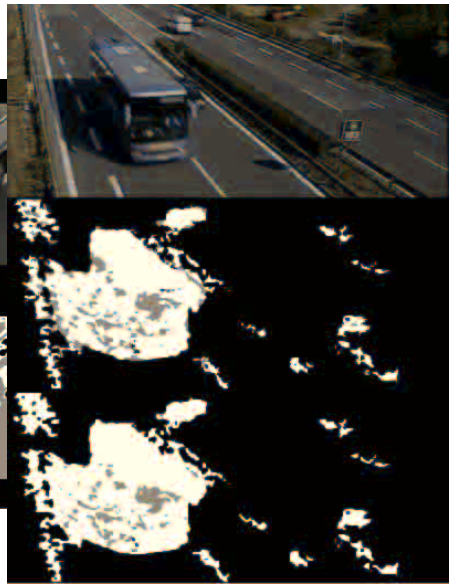

Figure 3.11: The *"casalecchio"* sequence.

seq.	α_{opt}	β_{opt}	α found
SEQ_ING_VISTA2	15	130	10
SEQ_ING_VESPA	18	190	17
SEQ_CASAL_PRIMO	32	255	24
SEQ_CASAL_SECONDO	80	255	17
SEQ_CASALECCHIO	1	255	86
SEQ_INGRESSOING	20	254	22
SEQ_CASELLO	3	128	5
SEQ_ING_CAMIONCINO	17	180	-

Table 3.1: A summary of Ground Truth (α, β) values and the α_{opt} found.

Chapter 4

C++ Classes Documentation

This chapter will treat in some detail the classes written for the job. Ipl and CV libraries were used for their good implementation of Computer Vision efficient algorithms.

4.1 Classes Overview

Inheritance was little or no used at all in this project, so classes shall be enumerated. The main classes will be described only. The others can be thought of as little utilities (or servers), to help doing the job in a structured and data-driven way.

TLowLevel This class wraps the concept of Low Level: it is initialized with some parameters having to do with background and the way the class has to process images; then it's evoked by the main method `void elabImage(IplImage* pImg)` which embodies the image into the background model and computes the OPs/BPs/SPs. Then, it can save them into BMP file or visualize them.

CProvaApp The MS Visual Studio application class: doesn't do anything.

CProvaDlg It's the dialog class; the dialog was structured to visualize up to 6 mini-windows in real time (originally foreground, background , 2-bit blob detection, punctual shadow classifier, local shadow classifier and joint shadow classification); other visual tools allowed to dynamically change the most important parameters and to dynamically inspect some points of the scene (for debug).

CUtil This is a collection of static utilities. They are often used in many part of the program; they have been released in a single static class for easy distribution and - obviously - for reuse.

4.2 Code fragments

In this section, code fragments are provided as samples in order to help future use of these classes. A fake class (*foo*) is described, calling the most important methods that were implemented.

```
Class foo {

// class that invokes TLowLevel class
// it has to wrap as member variables the following:
CSeqParameters UP; // CSeqparameters class wraps all the
pareameters useful for elaboration purposes (threshold, etc.)




}


void foo::main() {

IplImage* background = ...; // the beginning background

TLowLevel* LL=NULL;

if (LL == NULL)
        LL = new TLowLevel(*background,*background,*background,UP);
            // initialization of the object

IplImage* currentImage = getImage();    // fetch in some way the
current image CDib dibin(currentImage);       // converts ipl to
CDib
```

```
    // much of the interface is for graphical
    // rendering purposes... much can be pruned.
LL->ElabImage(
        dibin,          // current DIB image
        true,           // visualize elaboration in output
        &m_rectRettangolone6,   // ptr to CRect where the bmp is to be drawn
        dibPaccoGrigia,     // graylevel Dib
        this);          // this class wraps necessary
                        // informations (parameters and so on)

LL.ElabImage(currentImage); // the current image must be 'fed' to
the TLowLevel class in order tu update itself. ....

}
```

```
TLowLevel::ElabImage(CDib & DibRgb_input,
             bool   visualizza,
             RECT*  pRettangolo,
             CDib*  dibPaccoGrigia,
             CDialog* pDialog)
{

if (necessary)
    pre_elabImmagine(& DibRgb_input);
        // if used, this feature has got to be used on BKG, too!


// allocate (once for all or as needed) all images to be used


CUtil::modulo1_GetBlobFromCurrentAndBackground(*this);
       // calls module 1 (...)
```

```
bool ok=CUtil::modulo2_GetBlobFromCurrent_Background_Blob(*this);
// calls module 2a (punctual shadow-hood) and check for errors



CUtil::modulo3_GetOmbreWithRegionReasoning(*this); // calls module
2b (local shadow-hood)



    double bonta,dirSole;
    dirSole=CUtil::stimaDirezioneLuce(m_pIC_currentFrame,&bonta);
        // given the current frame, this function estimates
        // both the direction of the sunlight (DIRSOLE)
        // AND the goodness (bonta) of the measure.
        // This will be used in estimating alpha

    if (wanted)
        CUtil::aggiungiSimboloSole(m_pIC_currentFrame, dirSole  ,true);
            // adds the sunlight direction to the output
            // bitmap (in the form of an arrow terminating in a disc)


    CRImgOmbre ombraOr(CRIOresdiv,CRIODaModulo2,FILTRO_OMBRA_RIC1);
        // merges the two shadows models (puctual and local)
        // invisibly updated by modules 1,2,3. (they're member
        // of TLowLevel class , that's CUtil's friends.) into
        // a single robust tritonal class.
        // (CRImgOmbre wraps the concept of 3-tonal IplImage)
        // FILTRO_OMBRA_RIC1 is the kind of merging algorithm
        // used (it's a complex OR explained in the text)
        // this can be changed (for example to a simple OR).
        /*
            the class CRImgOmbre wraps an IplImage that's got to
            be tritonal. To be valid (there's is a bool validator
            in the class and constructors always provdie valid
            images, of course) an image must have, for each pixel,
            a choice between just 3 pixel:
                0   (background),
                142 (shadow),
                255 (foreground).
```

```
            With this representation,
            the simple visualization of the gray-level image show
            an image in which the object is white, the shadows
            gray and the background black.
        */


    // to debug an image to file:
if (want_to_save_img_to_file)
    CUtil::SaveIplimage2bmp(ombraOr.getIpl(),"prova123.bmp");


    post_elabImmagine();

}


void CUtil::stimaAlfa(
                bool modificaParametri, // if TRUE, the taken decision
                            // alters the main SeqParameters object
             TLowLevel& LL,         // the LowLevel Object
        const CString&  nomeFileDib, // DIB filename
              CDib*     dibPaccoGrigia,       // dummy dib for correct visualiza
    /*(OUT)*/ CRAlfaEstimator* &pEstimoUnico)
              // Estimator object used to extract correct alfa w
{
   int  MIN=1,
    MAX=200,
    STEP=1; // looks for all alpha values ranging from 1 to 200

   int nIterazioni=(MAX-MIN)/STEP; // numb. of iterations
   float* arr = new float[nIterazioni];  // container for goodness(alpha)



    if (pEstimoUnico == NULL)
        pEstimoUnico = new CRAlfaEstimator(nIterazioni);

    CRAlfaEstimator pEstimoVoltaxVolta(nIterazioni);

for (ix=0,i=MIN;i<MAX;i+=STEP,ix++)         // cycle on ALPHA {
```

```
CDib dibClonata(nomeFileDib);       // gets the dib
    LL.getPSeqParametri()->alfash=i;    // sets alpha to i
    LL.getPSeqParametri()->betash=255;  // sets beta to 255
    LL.getPSeqParametri()->setSoglieRicBySoglieMartino();
        // converts alfash,betash in alfa,beta (see article)

    LL.ElabImage(dibClonata,false,NULL,dibPaccoGrigia,NULL);
        // elabs the current image
    stimaDirezioneLuce(LL.getCRIOFinale(), & bontaGrigi);
                // extracts the goodness (bontagrigi)
                // from the final elaboration in LL object.
    arr[ix] = bontaGrigi;   // populates the 'internal' array goodness(alpha)

} // end iteration, array fully populated


bool ok=pEstimoUnico->addArray(arr,nIterazioni); // adds
'internal' array to CRAlfaEstimator object
    // it gets added for every image processed
    // in order to grant temporal fairness/robustness.

int alfaOtt=pEstimoUnico->getAlfaOttimo(); // Optimal alpha is
        // given by CRAlfaEstimator object; it's more valuable
        // if it's taken just after many images
}
```

Chapter 5

Conclusions

The aim of this work was to deploy an algorithm that could automatically tune the parameters needed in the task of removing shadows in order to successfully treat many different situations.

The study of the problem showed that the parameters that mostly change in the studied cases are the ones involving the darkness of the shadows. Two parameters were detected to simply (yet strongly) characterize this variability: the ones that were called in this work α and β (α was found to be more critical than β in each studied case). The problem became: tuning up α and β. A novel criterion was found to 'judge' the goodness of a given pair (α, β). A feedback model was then tried in order to converge to a good pair.

This overall task has proven to be:

- ◊ feasible and to work well in some cases;
- ◊ **able to tune** α in most cases and to mainly fail in cases where the system is less dependant on α (that is to say, when there are big discrepancies, changes in α don't affect the segmentation);
- ◊ unable to tune α in cases where a man (with this simplified model) couldn't provide a good segmentation pair;
- ⧫ weak in some situations (preprocessing that in some cases is necessary highly weakens the robustness of the algorithm); in these cases, though, ground truth results in bad segmentation, too (error in the *model*);
- ⧫ unable to well tune β, although the system is less sensitive to this parameter;
- ⧫ rather costly, although it *has* to be done just in bootstrap phase: at runtime it the scene undergoes slow changes, so the adaptive algorithm

can be easily integrated in a cheap way, in little pieces of elaboration distributed over time;

Bibliography

[1] Jurgen Stauder, R. Mech, J. Ostermann, *Detection of Moving Cast Shadows for Object Segmentation*, IEEE Transactions on Multimedia, vol, 1, no. 1, March 1999.

[2] Rita Cucchiara, C. Grana, M. Piccardi, A. Prati, S. Sirotti, *Improving Shadow Suppression in Moving Object Detection with HSV Color Information.*

[3] Rita Cucchiara, C. Grana, M. Piccardi, A. Prati, *Statistic and Knowledge-based Moving Object Detection in Traffic Scenes.*

[4] Martino Mola, *Reasoning during occlusion*, report for ARCES, DEIS, Università di Bologna.

[5] "Special Issue on Video Communications, Processing, and Understanding for Third Generation Surveillance Systems", C.S. Regazzoni, V. Ramesh, G. L. Foresti, Proceedings of the IEEE, Oct. 2001, Special Issue on 3GSS.

[6] "Moving Shadow Detection Using a Physics-based Approach", S. Nadimi and B. Bhanu, University of California, Riverside, California, IEEE 2002.

[7] "Shadow Elimination for Effective Moving Object Detection with Gaussian Models", C.J. Chang, W.F. Hu et al., Department of Electrical Engeneering, Yuan Ze University, Taiwan, ROC, IEEE 2002.

[8] "Adaptive video background modeling using color and depth", Michael Harville et al., Hewlett-Packard Labs, 1501 Palo Alto, CA 94304.

[9] "Cast Shadow Removing in Foreground Segmentation", A. Branca, G. Attolico, A. Distante, CNR, via Amendola 166/5, 70126 Bari, ITALY.

[10] http://www.intel.com

[11] http://sourceforge.net/projects/opencvlibrary/