POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

HSG Algorithm for Pedestrian Detection

Relatori:
Prof. Guido Masera
Prof. Maurizio Martina

Candidato:
Giuseppe Davidde

Luglio 2018

Acknowledgments

Table of contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Road traffic safety and security

A big amount of injuries, in many parts of the World, is due to the not proper car usage. In order to estimate and try to devise ideas that are able to reduce the number of people died because of car crashes, a division of *"World Health Organization"*[1] , called *"Road Safety"*[1] , has been founded.
Based on the data written in the *"Global status report on road safety"*[1] it can be noticed that, roughly, 1.2 million of World's inhabitants lose their lives each year due to road traffic injuries.
This large number has a huge impact in different aspects of daily life, such as, health and society progress. The reduction of these incidents has also an economic goal, because nowadays, it is possible to estimate that the road traffic crashes have a cost, approximately, equal to 3% of GPD (Gross Domestic Product).
The leading cause of death for young people, having an age between 15 and 29 years, is this kind of accidents (fig.1.1) and it is easy to understand that it is a dramatic result. An accurate analysis of the report shows that low and middle-income countries are the places where the highest percentage of death is recorded (fig.1.1).

Looking at the growth rate of population it can be observed that the increment is equal to 4% between 2010 and 2013, meanwhile, the number of registered cars increases of 16% [1].
This trend needs to be stopped for many reasons. According to the *"WHO"* projection, *road traffic injuries* are now estimated as the ninth leading cause of death and they will become the seventh leading cause of death by 2030.
Other bad aspects of the uncontrolled growth of the number of cars are represented to the increase of traffic congestion and then the rising vehicle tailpipe emissions, that lead to an increase of respiratory diseases.

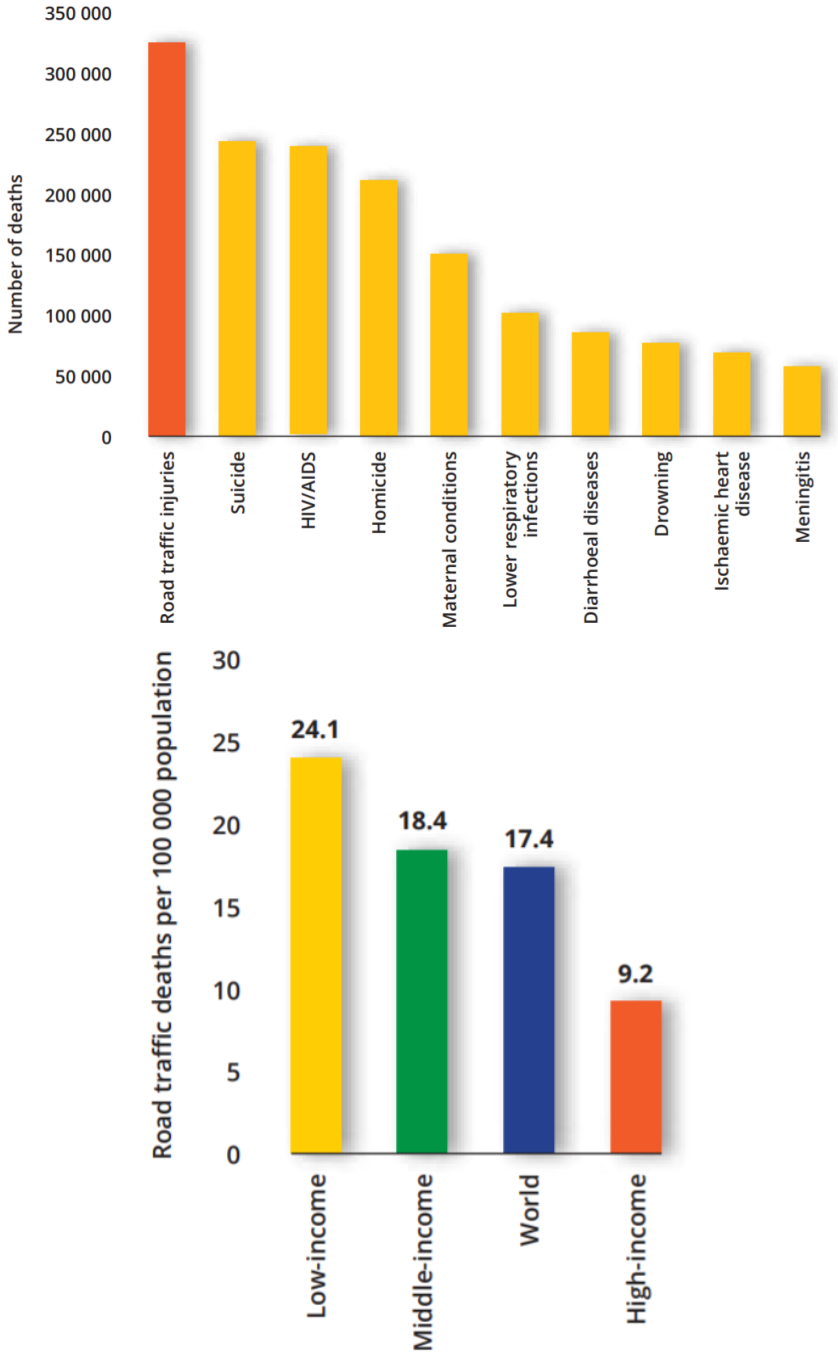


Figure 1.1: **(1)**Causes of death for people aged 15-29 years **(2)**Places with highest percentage of fatal injuries [1].

In the end, another not negligible problem is the reduction of physical activities such as walking and cycling, with well known healthy consequences.
In order to overcome these problems, in 2015 *"WHO"* has published the *so-called "2030 Agenda for Sustainable Development"* [1], where the main goal is represented to obtain a 50% decrease of global death rate due, in particular, to car traffic. Analysing the problem, this target can be reached through action on at least two sides:

a) Improvements about road safety legislation;

b) Actions to make road infrastructure more safer.

Having strict laws people become more responsible about, at least, five key risk factors:

- Driving speed;
- Drink-driving;
- Use of helmets;
- Use of seat-belts;
- Child restraints

This way of thinking is correct, because, exist a great inequality, about road safety, between countries with good legislation and ones with poor rules.

About the second side, it is based on two main aspects :

- A safer cohabitation between cars and other, most vulnerable, road users (as pedestrians, cyclists) is the result of in-depth and aware analysis about urban and road design;
- A production of smarter cars able to interact with the external environment in order to make them always safer.

Regarding to the percentage of car traffic fatalities that involve weak road users (pedestrians, motorcyclists and cyclists), it is, almost, equal to 50%. In this percentage, about the 22% is identified by the pedestrian (Table 1.1). Comparing this

data with different World regions, it can be seen that Africa has the highest value, this is due to the fact that people mostly use walking for moving (1.1). Based on these values, it is clear how it is important to develop car systems able to prevent injuries. With the purpose of summarizing the distribution of road victims in the World divided for road users categories, the Table (1.1) has been realized.

Table 1.1: **Distribution of road injuries among road users categories (in percentage)**

(data in percentage %)	**Cy**	**Ped**	**Motorcy**	**Car Pass**	**Others**
Eastern Mediterranean	**3**	**27**	**11**	**45**	**14**
Europe	**4**	**26**	**9**	**51**	**10**
South-East Asia	**3**	**13**	**34**	**16**	**34**
Africa	**4**	**39**	**7**	**40**	**11**
Western Pacific	**7**	**23**	**34**	**22**	**14**
America	**3**	**22**	**20**	**35**	**21**
World	**4**	**22**	**23**	**31**	**21**

1.2 Smart Driving

Until recently, improvements in automobile safety have been focused on the reduction of damages during accidents. Based on this way of thinking, technologies like airbags, seat belts pretensioners and crash-smoothing features have been realized. Nowadays, in contrast with the previous trend, a big amount of work is done with the aim of trying to directly avoid injuries. The meaning of this work is to implement clever on-board systems with the capability of control the environment and send a proper alert message to the driver when a dangerous situations arise. The research field that attract many IT companies and car factories, is focused on the realization of efficient driverless cars. Unfortunately, nowadays, the main drawback, that reduce the effort, financial speaking, in this direction, it is the deficiency of laws that allow to commercialize the autonomous cars. Nevertheless, in order to overcome these issues, European Union has defined a roadmap that has the goal of defining basic rules about automated cars development [2]. The guidelines have a validity up to 2030 , where it can be observed that the excepted technological

evolution is divided in 5 steps:

1. Driver assistance : autonomous features are activated only when an unavoidable incident is detected (i.e. *Pre Collision System* (PCS));

2. Partial automation : automated driving or parking can be turned on by the human driver;

3. Conditional automation : the car is able to detect automatically friendly conditions and then active the autonomous capabilities;

4. High automation : the vehicle is thinking to drive itself, the human takes the control only in emergency cases;

5. Full automation : in this futuristic environment no human interactions are needed during the driving.

The projections indicate that, with a massive autonomous cars usage, it is possible to fall down steeply the amount of traffic collisions. So all of this translates into save hundreds of billions of dollars in terms of car damages on one side and on the other side, the health-costs decrease suddenly [3]. The biggest difficulty about the on-board systems development is due to the fact that chips have to capabilities to "discover" and "understand" the environment [4]. These kind of systems are dubbed *Advanced Driver Assistance Systems (**ADAS**)*. Regarding to the basic features, these electronic architectures need to have, at least, an alarm that starts in dangerous situations and execute corrective operations. In the commercial World, some examples are given to the *adaptive cruise control*, where the distinctiveness is to maintain a safe gap between other cars, and to the *departure warning* that alerts the driver when the car is out of the lane.

1.3 Pedestrian detection overview

In ADAS World one of the most interesting type is called *Pedestrian Protection System* (PPS) . Its distinguishing feature is to detect both stationary and moving people inside a specific region of interest (ROI), such as the vehicle perimeter and act

specific actions if the collision is unavoidable. Based on [5], an interesting statistic indicates that 70% of pedestrians, in an accident, are in front of the car ; in addition, 90% of these are moving. With the purpose of reducing significantly the number of humans killed by cars, every PPS has to, as mandatory elements, frontal sensors. Looking at other application fields, *Pedestrian Detection* is very suitable for many intelligent video surveillance systems. The kinds of operations that involve the detection of pedestrians are really hard to perform for, at least, three aspects :

1. High changing of scenarios : Humans, in this case intended as pedestrians, tend to have different poses and clothes, but, this is not the only problem, because, due to the perspective, it is possible to view the same thing with different sizes.

2. Adverse surroundings conditions : In order to detect pedestrians it is mandatory to work in outdoor urban area contexts. These working areas have many elements that can reduce the efficiency and reliability of the systems, such as:

   - crowded background (it is typical of urban areas);
   - big variations of ambient light conditions;
   - partial overlap of the human by other objects.

3. Real-time requirement : These systems have to consider as life-saving devices and then the reaction and its accuracy (in terms of ratio between false alarms and misdetections) are crucial aspects.

From research point of view, in the last few decades many algorithms have been developed and proposed but, the exponential computational complexity has meant that only in the few past years it was possible to start with the real implementations. In order to do this there are two potential ways to follow :

- Software Implementation : this kind of realization has many positive aspects, among them, sticks out its flexibility. It is a mandatory condition, especially, when the requested system needs to be tuned pursuant to experimental results, before to be commercialized. As stated above, the *Pedestrian Detection* algorithms need to respect, strictly, real-time standards (it means to work, at

least, with 30 fps). For many years, this was the main problem, because, in order to maintain the requirements on a general-purpose CPU, the only solution was a significant reduction of accuracy and, of course, it was not the right solution. With the aim of overcoming these issues the parallel approach has been exploited. The great improvement was given by the *Graphical Processor Unit* (GPU) usage, because, in this way, the designers could use *"Divide et Impera"* method by splitting the algorithm in many threads executed in parallel. This way of thinking was really attractive, because, the final result was a noticeable increase of performance associated to no leaks in terms of flexibility. All of these good things, otherwise, have a cost. The main drawback stems from the fact that this complex and expensive architectures (PPS) can be placed only on luxury cars.

- Hardware Implementation : this development way has the aim of producing systems suitable also for cheaper cars. By deeply studying the maximum internal parallelism and the needed basic elements an *ad hoc* integrated circuit can be realized. It is possible to summarize the positive aspects of this line of thought as follows:

  – Higher performance than SW implementation;

  – Lower per-unit realization cost if an enough number of samples is produced.

  Looking at the drawback, it is pretty obvious the waste of flexibility. Based on the previous concern, a crucial thing is to propose proper integrated architectures for the most interesting *pedestrian detection* algorithms, such that, the customized hardware can be regarded as a long-term device and then car industries, for instance, are more likely to invest their moneys.

Chapter 2

Current promising Pedestrian Detection systems

2.1 Pedestrian Protection System high-level view

In figure (2.1) a generic splitting of PPS is proposed [5]. In order to understand how

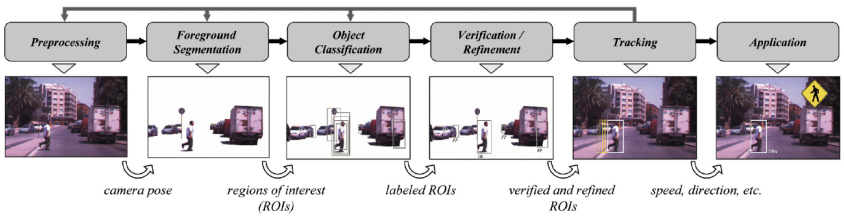


Figure 2.1: **Block diagram of a generic on-board PPS** [5].

these systems work, it is useful to describe each block:

- Preprocessing : the goal of this part is to interact with sensors then acquire and send the image in reliable and efficient way. Thanks to the current technology, the duties, such as exposure time, camera calibration and illumination, are accomplished by a single chip in very easy way.

- Foreground Segmentation : the aim of this block is to determine and extract the *Regions of Interest* of the image to be analysed. This task is based on the fact that weak road users are detected in a very limited band localized in horizontal position respect to the center of the acquired picture (2.2).

- Object Classification : the block works receiving, as input, a group of *ROI* with, the possibility of pedestrian presence inside of each of them. The idea

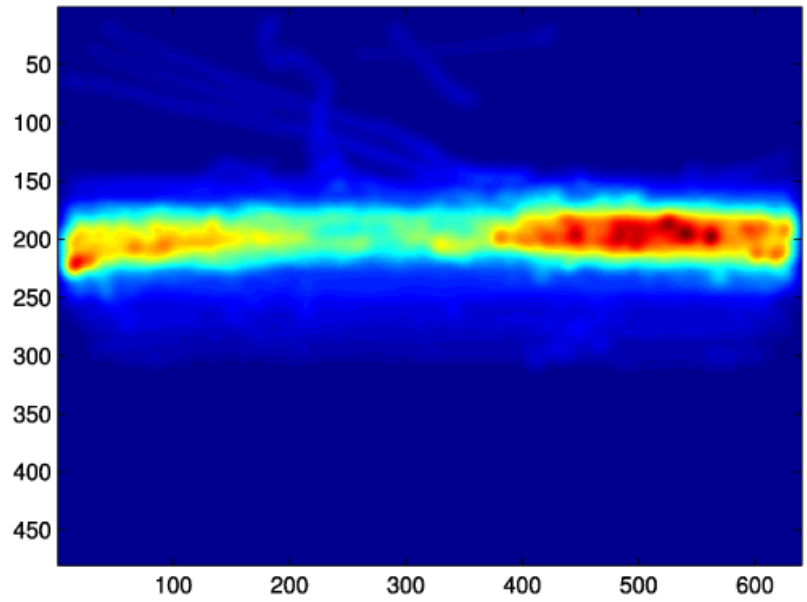


Figure 2.2: **Typical location of pedestrian in an image. Based on statistical distribution obtained with the Caltech dataset** [6].

is to split these *ROI* into two categories : *pedestrian and no pedestrian categories*. This kind of operation is very crucial to do. For this reason many scientists offer their brain effort to try solving this problem in efficient way. Regarding to 2D object classification, the usual division is in *silhouette matching* and *appearance* technique. The most reliable and faster algorithms base their operations on the second, previous mentioned, technique. Looking at *appearance* method in detail, it can be noticed that it is based on so-called *descriptors*, also known as a *space of image features*. Moving to the *Classifier*, the operation, called in the jargon, *"training"* is carried out by using different *ROI*, where their distinctiveness is to include either positive and negative cases. Nowadays, the most used classifiers are the following ones:

- *Support Vector Machines (SVM)*;
- *AdaBoost*;
- *Neural networks*.

In the end, with the purpose of evaluating the performance of these systems,

proper benchmarks have been developed. In this way it is possible to test the capability to distinguish between false negatives and positive ones. In particular, some used Datasets are : *Caltech* and *INRIA*.

- Verification/Refinement : the module works in way to take out false positives, by enhancing *ROI* shape. In detail, this section executes also a segmentation of the pedestrian in order to estimate the distance.

- Tracking : this step, present only on the most advanced systems, is able to monitor the walking of the detected pedestrian. The operation is very useful, on one side, to reduce the amount of false detections, by the virtue of the fact that consecutive frames have been processed, and on the other side it is possible to realize a proper feedback network. Mentioned network has the aim of *predicting the following detections*.

- Application : the final step consists in to take a decision founded on the signals produced by the previous blocks.

As mentioned above, the most troubled section of PPSs is the *Classification* one. The proposed mechanisms that approach to this problem are called *"Pedestrian detection algorithms"*. In order to elucidate how they work, a briefly description in the following is written.

2.2 Pedestrian detection algorithms

2.2.1 Basic ideas

All current pedestrian detection algorithms share the following elements:

Sliding window

The *sliding window* approach is based on to process the image by using a window of defined size overlayed to the original picture. At each cycle, the window result is classified and then the window is moved on the image of an amount of pixels, called *stride*, and a new result is obtained. It is possible to define the number of windows

(W) computed for each frame as follows:

$$W = \frac{w - w_{extracted}}{s} * \frac{h - h_{extracted}}{s} \tag{2.1}$$

where:

- w = width of the input frame
- h = height of the input frame
- $w_{extracted}$ = width of the extracted window
- $h_{extracted}$ = height of the extracted window
- s = stride of the process

Pyramidal mechanism

A challenging duty is to choose the adequate image dimension. The difficulty consists of the fact that in a picture, an human road user can exhibit different sizes. Based on the data inside of the most common Dataset, in particular for *Caltech dataset*, it can be noticed that 69% of samples have a height ranged from 30 to 60 pixels [6]. The easiest solution requires a dynamic-size window, but, this choice has as consequence a sudden increase of detector complexity. With the aim of overcoming this problem, the solution is to use the so-called *image pyramid* approach. For this purpose, the original image is resized many times to obtain a pyramid where the base is represented by the input picture and the smallest version of this is placed on the top. The *sliding window* process is carried out for each level of the pyramid using a single fixed-size window. Every layer has a distance to the another one equal to one octave.

Features extraction

One of the most important problem for the classifier is identified to the fact that the traditional image encryption is not suitable enough. The previous concern justifies why some elaborations have the aim of highlighting key features in the image to help the classifier to identify the presence of a possible weak road user. In literature, it

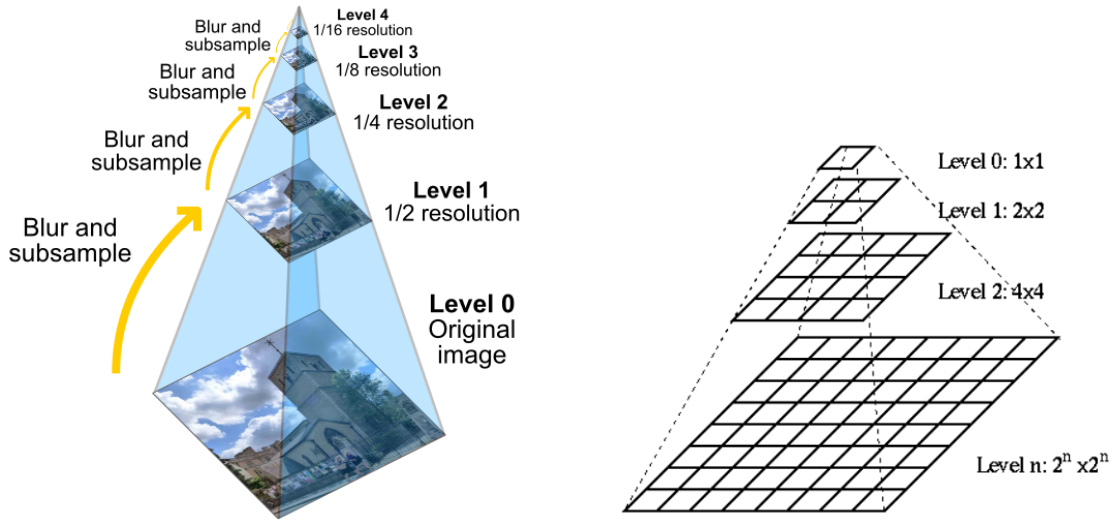


Figure 2.3: **Typical Image Pyramid** [7].

is possible to find a very big amount of *features extraction techniques* that have as output a set of constant length vectors which are sent to the classifier.

Classification

The *classification algorithms* are derived by the *Machine Learning* theory. Thinking about the system, it receives as input many vectors associated to, in general, *N-dimensional spaces* and the output is simply a label that represents the category to whom the input vectors belong. A particular kind of classifier is the *binary classifier* which is defined as defined when only two categories are allowed. Their job can be summarized in two different steps:

- Training : in this stage the target is to help the classifier to build, itself, a classification rule. To do that, the classifier is "nourished" through proper labeled examples having their category. Considering the case of a *binary classifier* , the classification rule commonly is the equation that describes an hyperplane able to split the input space in two semi-spaces. It is possible to distinguish two kinds of representation, illustrated in the following figure (2.4), where on the left a non-linear function and on the right a linear function are depicted:

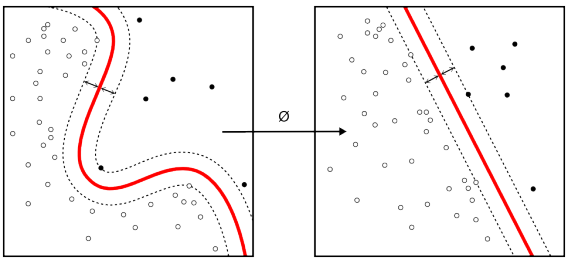


Figure 2.4: **[1]Non-linear binary classifier [2] Linear binary classifier** [8].

As you would except, a classifier with a non-linear function requires a lot of effort to implement it.

- Testing : in this phase the aim is to validate the performance of the classification. With this purpose a robust dataset with realistic pedestrian walking situations is a mandatory thing. In particular, the ratio between *false positives* (FP) and *false negatives* (FN) rates gives the best estimation as concerns the performance of a *pedestrian detector*. This ratio can be measured, adopting at least two mathematical tools:

  1. ROC curves: where the alias means *Receiver Operating Characteristic*. This kind of curves shows the diagnostic ability of a binary classifier at the variation of its discrimination threshold. The graphical plot is obtained by using the *true positive rate* (TPR) against the false positive rate (FPR) when the threshold changes. In particular, the *true positive rate* is also known as *sensitivity* which measures the proportion of positives that are correctly identified and the *false positive rate* can be treated as (1-*specificity*) [9] , where the specificity indicates the proportion of negative that are correctly identified. Based on these previous concerns it is obvious that bigger is the area under the curve, better is the classifier.

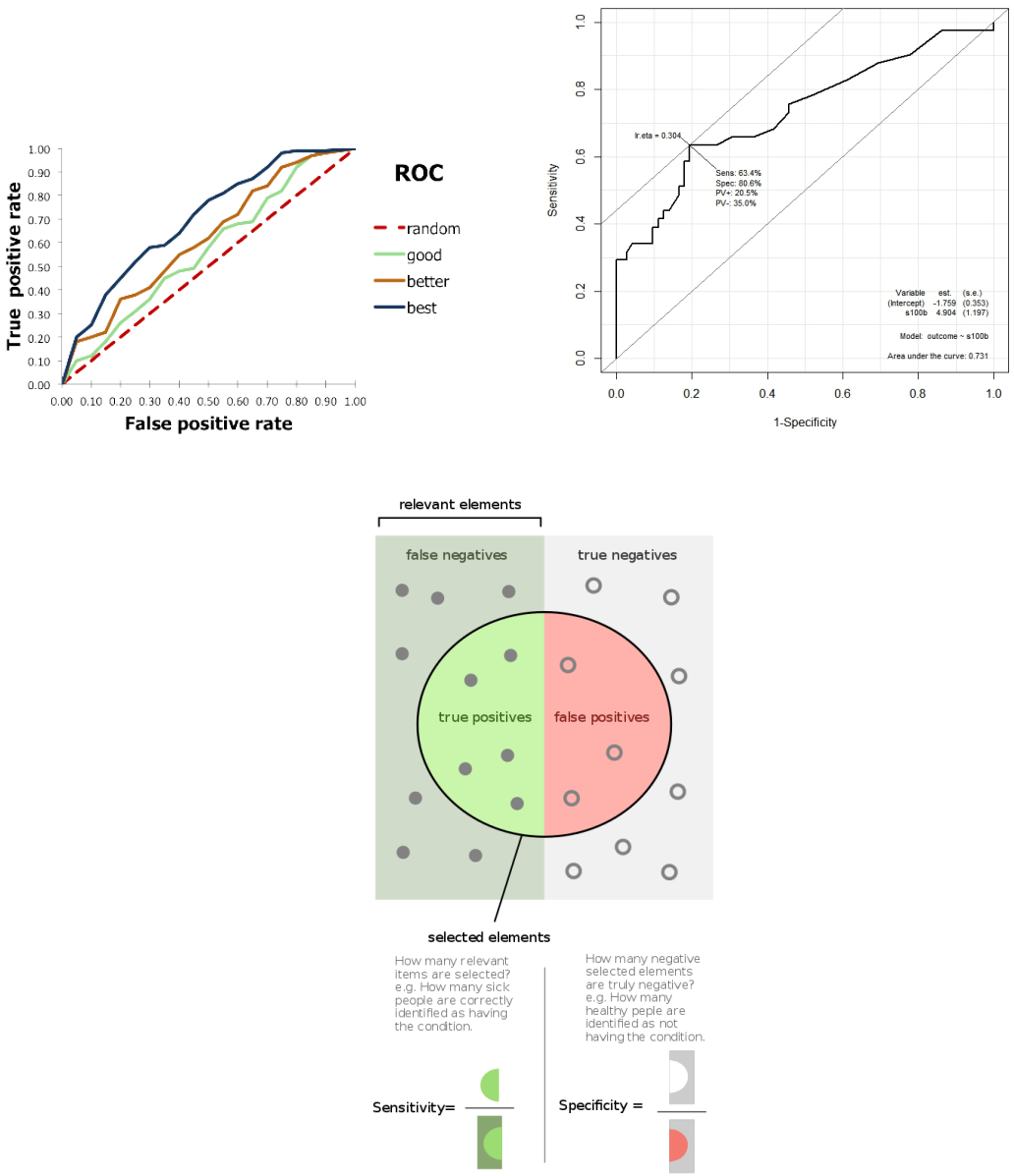


Figure 2.5: **[1,2] Two kinds of representation of *ROC* curves [3] TPR and FPR explanation** [10] [9].

2. Miss rate curves or Detection Error Tradeoff (DET) curves : this is another method to evaluate the false negatives. The plot is obtained by

using *false negatives* and *false positives per image* (FPPI). However, this type of representation is more oriented to evaluate the performance of the whole system and not only the classifier one. An example is shown in figure 2.6:

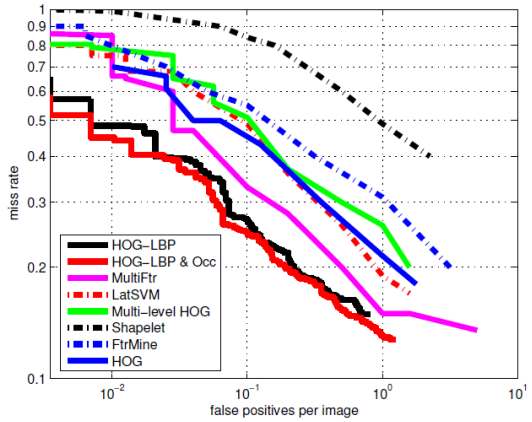


Figure 2.6: **Miss-rate curves that compare the performance of different PPSs** [11] .

In order to train and evaluate in efficient way the performance of pedestrian detection algorithms, many datasets have been realized. The most common are:

1. Caltech;
2. ETH;
3. Daimler;
4. TUD-Brussels;
5. INRIA.

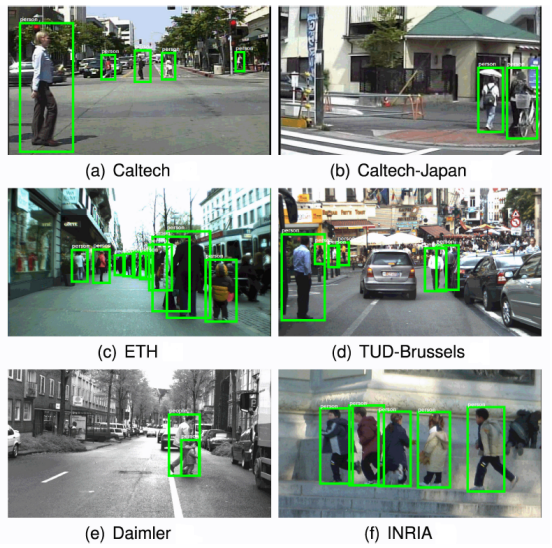


Figure 2.7: **Examples based on different datasets** [6] .

Regarding to the previous mentioned *datasets*, it can be said that the *INRIA* is the eldest one, instead, nowadays, the widely used is the *Caltech* one. This changing is born to the fact that the *INRIA* contains samples too easy to train in proper way the current complex systems.

2.3 Feature detectors based on sliding window approach

Detectors based on Haar features

The main motivation that conducted, in 2000, *Constantine Papageorgiou* and *Tomaso Poggio* to develop, at M.I.T., this detector was the steep growth of image and video stemming from the Internet network [12]. The developed system is very famous due to the fact that it was the first realized with *sliding window* technique. In detail, the *Feature extraction* section is based on *Haar wavelets expansion of the image* and then the *Classification* task is realized by using of a *Support Vector Machine* (SVM).

Haar wavelet overview

In general, the term *Wavelets* indicates a useful mathematical construction for multiresolution analysis. The basic idea is to create a set of approximated subspaces, in a way that the following expression is satisfied:

$$V^0 \subset V^1 \subset .. \subset V^j \subset V^{j+1} \tag{2.2}$$

The meaning of (2.2) is to obtain a vector space V^{j+1} with the property of describe more details respect to space V^j. Looking at the *Haar wavelet*, it represents a sequence of rescaled *"square-shaped"* functions that form a wavelet family or basis [13]. Regarding to the history of the *Haar sequence*, it was proposed by Haar [14] in 1909, with the purpose of describing an orthonormal system for the *square-integrable functions* on the interval [0,1]. The main issue of this kind of *wavelet* derives to the fact that it is not continuous and then not differentiable. However, this drawback can be treated as an advantage when the application field is based on the analysis of signal with drastic transitions. Mathematically speaking, it is possible to describe the *Haar wavelet's mother wavelet function* $\psi(t)$ as follows:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2}, \\ -1 & \frac{1}{2} \leq t < 1, \\ 0 & otherwise. \end{cases} \tag{2.3}$$

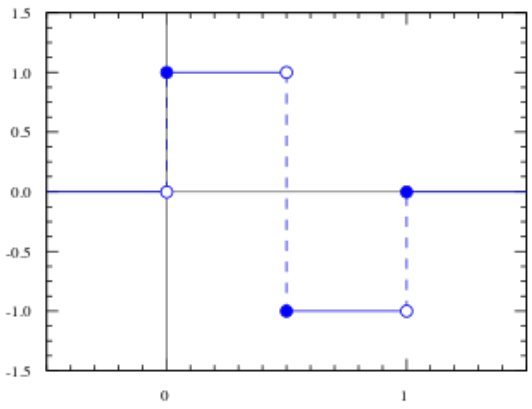


Figure 2.8: **The Haar wavelet example** [13] .

Practical relationship between scaling and wavelet functions

By using the *FFT analysis* the main dilemma is the following one:

> "High resolution in the frequency domain means a poor resolution in time domain. High resolution in time domain, means a poor resolution in the frequency domain."

Wavelets have the aim of solving this problem, by defining the, above written, *Mother wavelet function* (2.3). At this point it is useful to explain how *"Discrete Wavelet Transform"* (DWT) works. To do this, the basic elements to consider are the following [15] :

- Discrete-time signal, called $f_{[.]}$ of lenght N;
- An *analysis filter bank*, composed by the filters $h_{[.]}$ and $g_{[.]}$ of length M. The purpose is to compute $DWT(f_{[.]})$;
- A *synthesis filter bank*, obtained by two filters $\bar{h_{[.]}}$ and $\bar{g_{[.]}}$, respectively. The goal is to compute $IDWT(f_{[.]})$;
- The *"mother wavelet"* (wavelet function), that can be expressed both as (2.3) that as a recursive formulation like the following one:

$$\psi(t) = \sum_k g_k * \phi(2t - k) \tag{2.4}$$

- The *"father wavelet"* (scaling function), that has two kind of expressions.

1)

$$\phi(t) = \begin{cases} 1 & 0 \leq t < 1, \\ 0 & otherwise. \end{cases} \tag{2.5}$$

2)

$$\phi(t) = \sum_k h_k * \phi(2t - k) \tag{2.6}$$

In order to describe completely the system, it is necessary to explain the relationship between the four, previous mentioned, filters:

a) $\bar{h_k} = h_{M-k-1}$

b) $g_k = (-1)^k h_{M-k-1}$

c) $\bar{g_k} = (-1)^{k+1} h_k$

These equations mean that if one of the parameter changes, all the others are directly influenced. The results, after a proper decomposition of $f_{[.]}$ in sub-signals having different frequency contents, are shown in figure (2.9):

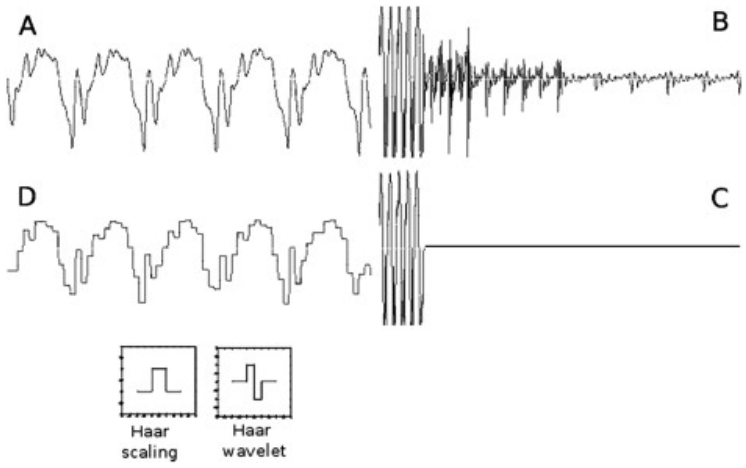


Figure 2.9: **(A): reference signal; (B) its third-level DWT; (C): modified third-level DWT; (D): reconstructed signal; Below (D): father and mother wavelets** [15] .

Looking at *2D DWT* applied to images, it can be observed that both scaling and wavelet filters are applied to rows and columns. The output result depends on the order which these filters are used :

- Wavelet filter applied both on rows/columns : the result is an highlight of diagonal details of the reference image;

- Wavelet filter applied on rows / Scaling filter applied on columns : the final outcome emphasises the vertical details of the initial image;

- Scaling filter applied on rows / Wavelet filter applied on columns : the output points out the horizontal details of the input image.

Papageorgiou and Poggio method

The solution proposed in [12] has the aim of encoding the difference, in terms of average intensity, amidst local regions along various orientations. To do this, a *quadruple density dictionary of wavelets* is computed. In particular, this technique allows to avoid downsampling in some sections of the computation. Looking at the kind of wavelet response, it is possible to detect an intensity change, boundary presence and in particular at which location in the image it appears. Another noticeable thing is represented how the output values are selected and sent to the SVM. In detail, the more relevant *wavelet coefficients* are 1326 [12]. In figure (2.10) are depicted the elements on which this *feature extraction* technique is founded.

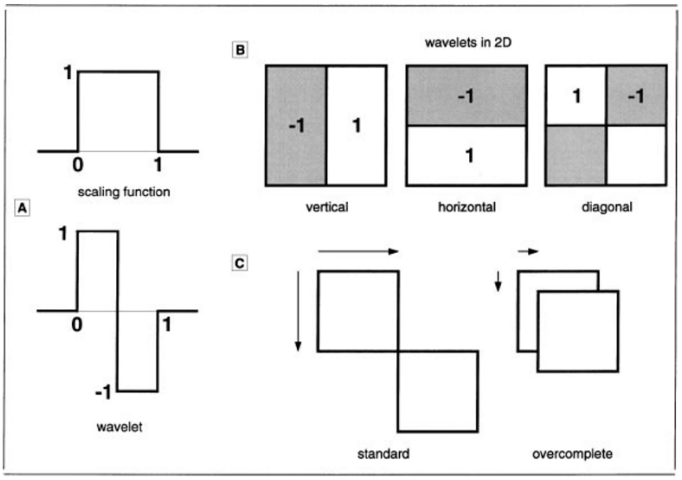


Figure 2.10: **(A): "father" and "mother" Haar wavelets; (B) three 2-D non-standard Haar wavelets; (C): shift in the standard DWT and proposed *quadruple dense shift* to obtain a dictionary of wavelets** [12] .

This proposed detector highlights a very high accuracy performance but at the cost to have an unsustainable latency; in fact, the requested time to elaborate just a single frame was estimated in roughly 20 minutes on a CPU of that age [12].

Fast computation of Haar-like features

In 2004, thanks to *Paul Viola* and *Michael J.Jones* a faster Haar features based detector was developed [16]. This system classifies images according to the value of simple features. The types of features are three and in detail :

- Two-rectangle feature : it is computed as the difference among the sum of the pixels inside two *rectangular regions*;

- Three-rectangle feature : it is derived as the sum within two outside rectangles subtracted from the summ in a center rectangle;

- Four-rectangle feature : it is obtained as the difference between diagonal pairs of rectangles.

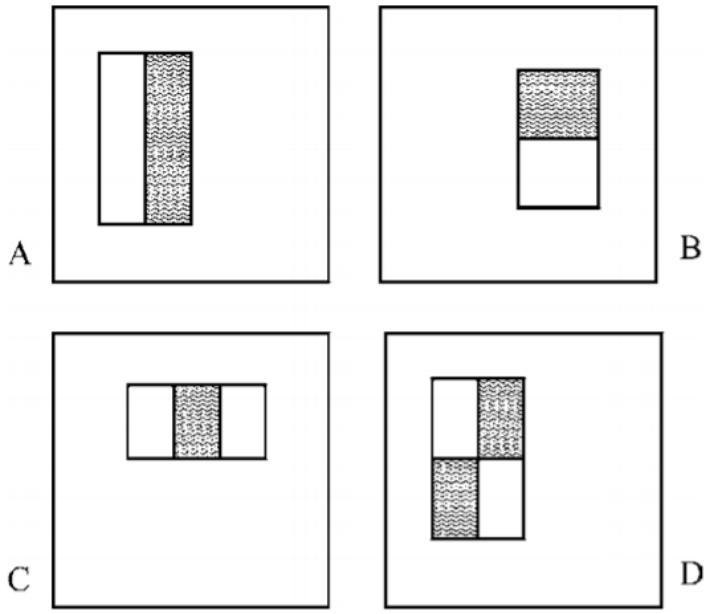


Figure 2.11: **The sum of the pixels which extend inside the white rectangles are subtracted from the sum of pixels inside the grey rectangles. (A),(B): 2-Rectangle features; (C): 3-Rectangle feature; (D): Four-Rectangle feature** [16].

The main innovation provided by this detector was the *Integral Image* concept [16]. By adopting this kind of intermediate representation for the input image, it is possible to compute in very fast way the, previous mentioned, *Rectangle features*. Mathematically speaking, the *Integral Image*, at location (x,y), is defined as follows [16] :

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x',y') \tag{2.7}$$

$$\begin{cases} ii(x,y) & integral \quad image \\ i(x,y) & original \quad image \end{cases}$$

$$s(x,y) = s(x,y-1) + i(x,y) \tag{2.8}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{2.9}$$

$$\begin{cases} s(x,y) & cumulative \quad row \quad sum & s(x,-1)=0 \\ ii(x,y) & integral \quad image & ii(-1,y)=0 \end{cases}$$

This method allows to compute the *Integral Image* in only one pass over the original image.

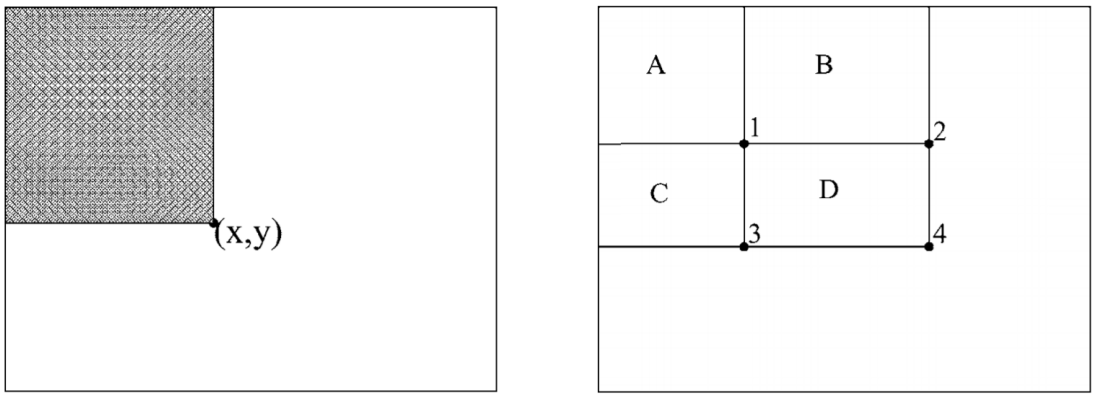


Figure 2.12: **[1] The value of the integral image at point (x,y) is the sum of all the pixels above and to the left; [2] The sum of the pixels inside rectangle *D* can be obtained with only 4 memory accesses. The value at point (1) is the sum of the pixels in A, the value at location (2) is $A+B$, the value at point (3) is $A+C$ and then the point (4) has the value $A+B+C+D$. In detail, the sum within D is computed as $4+1-(2+3)$** [16] .

In the end, another great advantage, in terms of performance, derives to the *AdaBoost classifier* usage [17].

Histogram of Oriented Gradients Detector

This kind of detector, introduced in 2005, thanks to *N.Dalal* and *B.Triggs* [18], is based on *sliding window approach*. One of the most important blocks is called Feature Descriptor, that represents an alternative representation of the reference image, with the aim of simplifying the extraction of useful informations unwrapping the redundant ones. In detail, it takes as input an image of size W×H×N, with N= number of channels , and it is able to produce as output a so-called *feature vector*. This vector is composed by helpful values for an image classification algorithm such

as *Support Vector Machine* (SVM). Looking at the *HOG feature descriptor*, the used *features* are the *distribution (also known as* ***histograms****) of directions of gradients (****oriented gradients****).* In detail, the *Gradients* $\left(G_x = \frac{\partial f(x,y)}{\partial x} \quad and \quad G_y = \frac{\partial f(x,y)}{\partial y}\right)$ of an image are the optimal candidates by virtue of the fact that their *magnitude* is high enough around edges and corners (also defined as *regions of abrupt intensity changes*). These values shall ensure to have very detailed informations about the shape of the *"target"* object. At this point, it is useful to describe all required steps to calculate the final *HOG feature vector* [18] [19] :

- Preprocessing : starting from an image of any size, the first thing to do is to crop it in order to obtain a fixed aspect ratio (*typically* $1:2$) patch of the input image. The final operation consists to have a resized version of this patch, according to [18], the final dimensions are 64×128. In this step, also, sometimes it is common to apply different kinds of *"kernel filter"* [20], with the purpose of obtaining a better version of the image to process. The procedure is illustrated in figure (2.13):

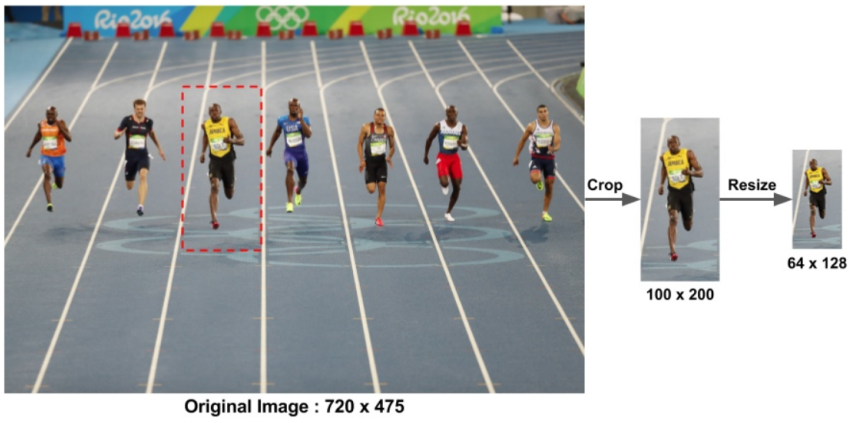


Figure 2.13: **[1] Original Image of any size; [2] Cropped version; [3] Resized version** [19].

- Gradients calculation : this is the first analytical step, where the purpose is to compute both the horizontal (G_x) and vertical (G_y) gradients. This is easily

obtained by filtering the image with proper kernels, like these (2.10) :

$$\begin{cases} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} & and \quad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \end{cases} \tag{2.10}$$

where the left one is used for G_x computation, instead, the right one is used for G_y.

- Magnitude and Direction of gradient : these values can be obtained by the use of the following relations:

$$\begin{cases} |G| = \sqrt{G_x^2 + G_y^2} \\ \theta = atan\left(\frac{G_y}{G_x}\right) \end{cases} \tag{2.11}$$

After these computations, a graphical result is given by the figure (2.14):

Figure 2.14: **[1] Absolute value of G_x; [2] Absolute value of G_y; [3] Magnitude of Gradient** [19].

From (2.14) it can be noticed that :

a) Gradient image : it is able to unwrap a lot of non-essential informations (like constant background), highlighting, instead, the outlines. Looking

at the *gradient image* it is possible to say, in very easy way, if a person is inside of the image.

b) Magnitude of gradient : the purpose is to emphasise every noticeable change in terms of intensity.

- Division in cells and HOG computation : at this point, according to [18], a division in *"cells"* of the input image is done. Considering an image of size 64×128, the process produces 128 blocks, each of them having 8×8 pixels. Each image patch, after the *magnitude* and *direction* computation, exhibits an output composed by 128 numbers (every pixel in the cell is expressed with two numbers. $8 \times 8 \times 2$). The following procedure has the aim of representing in compact, alternative and robust, in terms of noise, way these 128 numbers by using a *9-bin histogram*. By using the so-called *"unsigned" gradients* representation, the angles are between 0 and π and then the nine elements of the array are related to angles 0°,20°,40°,60°,..,160° (*"unsigned gradients representation"*). The steps to fill this vector are explained in the following figure (2.15):

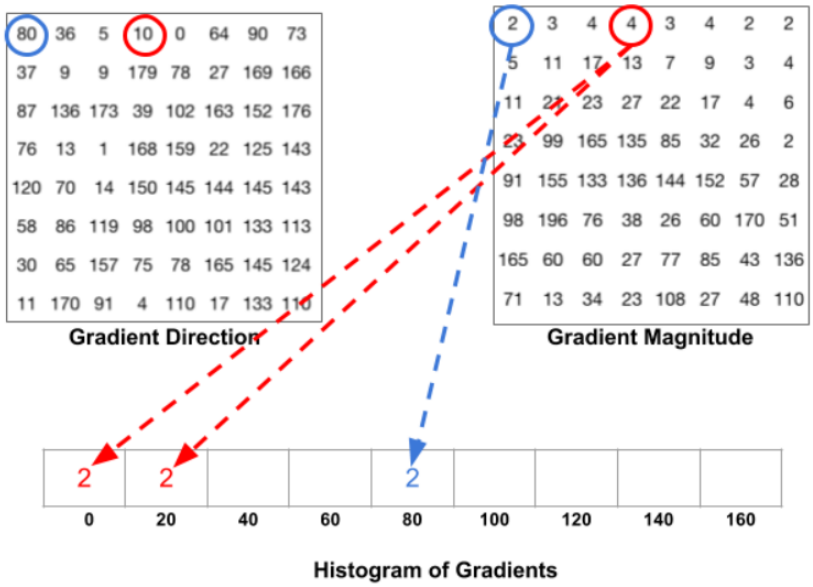


Figure 2.15: **In this example the purpose is to emphasise how the *pixel vote* is splitted in half way between** 0° **and** 20° **due to the fact that** 10° **is in the middle of the range** [19].

The most challenging situation appears when the *gradient direction* is greater than 160°. In this peculiar case, the algorithm defines that the *pixel value* contributes in proportional way for both 0° bin and 160° bin. The figure (2.16) has the aim of clarifying this aspect:

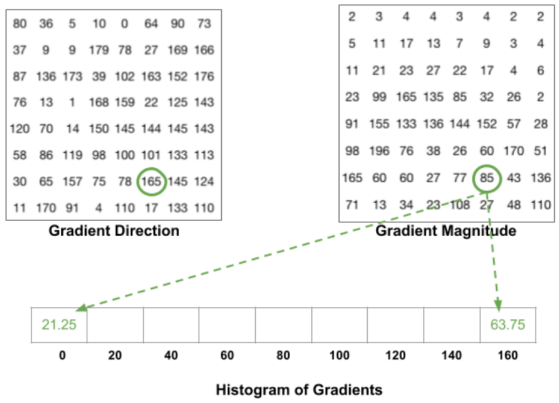


Figure 2.16: **The pixel angle is greater than** 160° **and the relative pixel value is splitted in two angle-bin of the HOG** [19].

At this point the final *Histogram of Oriented Gradients* related to all 8×8 cells present in the image, is obtained by adding whole *pixel contributions*. An example is given by the figure (2.17):

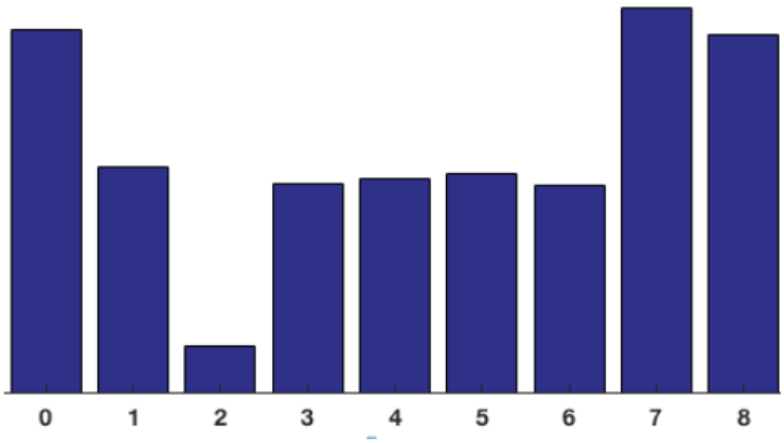


Figure 2.17: **Alternative way to *"visualize"* HOG** [19].

- Block Normalization : looking at the results arising from the *Gradients computation* it is really clear that these values are sensitive to the *light changing*. With the purpose of overcoming this issue, the *Block Normalization* operation is needed. Considering, for instance, an *RGB color vector* of dimensions $\begin{bmatrix}128 & 64 & 32\end{bmatrix}$, by a simple arithmetical operation it is possible to determine the length of this vector as $|v| = \sqrt{128^2 + 64^2 + 32^2} = 146.64$. A common kind of normalization is the so-called ℓ^2-norm, where the *"normalized vector"* is obtained dividing each element of the input vector by $|v|$. An interesting result is that the final vector $\begin{bmatrix}0.87 & 0.43 & 0.22\end{bmatrix}$ is the same, independently from the scale (if the aspect ratio between the dimensions is fixed). According to [18], the best results are reached by adopting 16×16 block, where the internal 4 histograms can be concatenated in order to build a 36×1 vector. In the end, other operation, that is able to suppress the *"aliasing presence"* is to move the window in *"overlapped way"* (i.e. 8 pixels per time) :

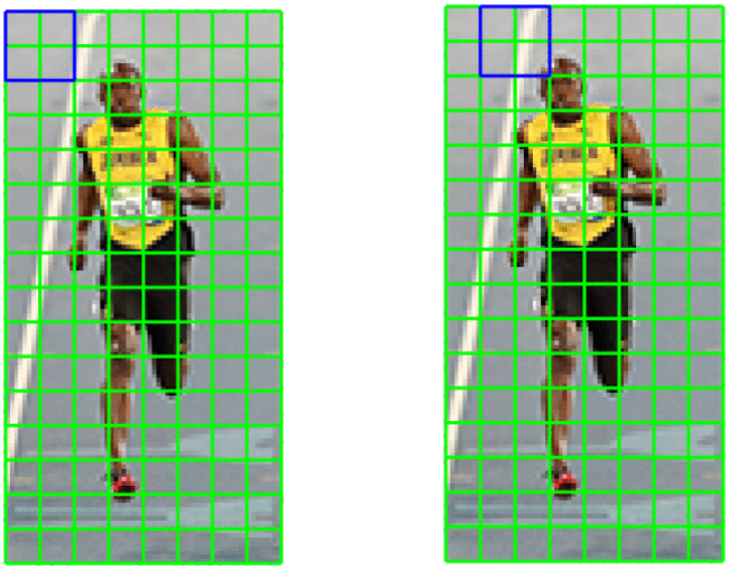


Figure 2.18: **[1] First** 16×16 **block; [2] Second** 16×16 **block moved with** 50% **overlap** [19] .

- HOG feature vector computation : this step requires to concatenate the 36×1 vectors into one massive vector. In order to compute the dimension of this final vector, the considerations to do are (i.e. 64×128 input image divided in 128 blocks of 8×8 pixels):

  a) Number of positions of the 16×16 blocks : for the previous mentioned example, the number of position are, respectively, 7 for the horizontal axis and 15 for the vertical one. In the end, the total is obtained as the product $7 \times 15 = 105$ positions;

  b) Each 36×1 vector is represented by a 16×16 block : it means to have a one giant vector of $36 \times 105 = 3780$ dimensional vector.

  An optional task is to try plotting the 9×1 normalized histograms based on 8×8 cells, in such a way as to have an idea of which data are sent to the classifier (SVM) (figure (2.19)):

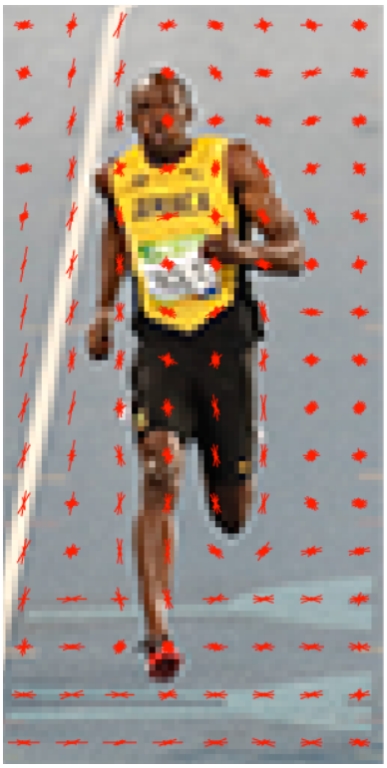


Figure 2.19: **It can be noticed that as the dominant direction of the histogram is able to follow the shape of the person, particularly around the torso and legs** [19] .

Histogram of Significant Gradients Detector

Starting from the fact that HOG [18] is promoted as superior to all other single features proposed for pedestrian detection, the purpose of many scientists was to merge HOG with more elaborated features in *"cascade schemes"* in order to obtain higher detection rates. These ways of thinking, however, lead to very complex and power expensive hardware and software implementations. The main drawback born to the fact that *HOG*, in its implementation, requires inherently difficult *floating point operations* and *repeated memory accesses*. The work proposed in [21] addresses these issues by adopting a fast and proficient *object detection environment* requiring a *low-complexity feature* based on *Histogram of Gradients* combined with a Lookup table (LUT)-based kernel *SVM* classifier. In detail, the proposed detector, called *Histogram of significant gradients (HSG)*, albeit being substantially less computationally hungry than the *HOG*, exhibits better classification when *INRIA* and *ETH* datasets are used, as shown in figure(2.20):

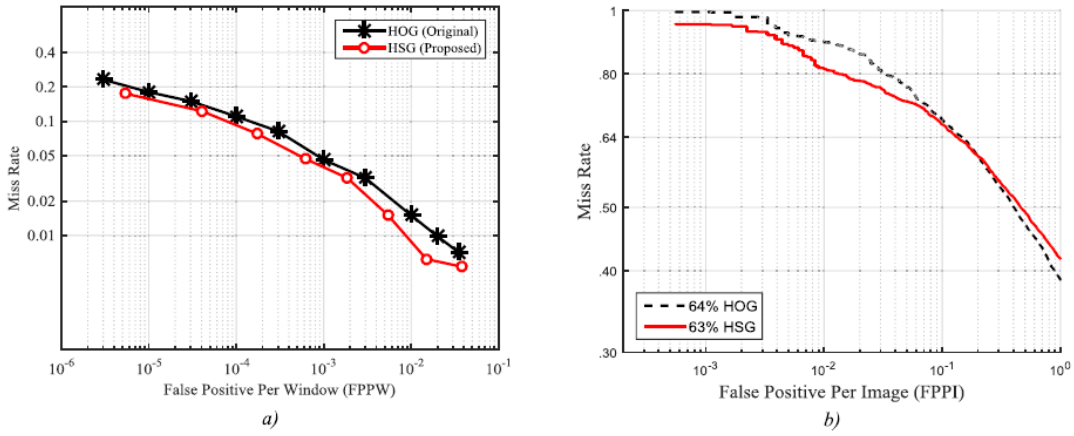


Figure 2.20: **Comparison between HSG framework and the original HOG. a) DET curves based on INRIA data set. b) ROC curves obtained with ETH data set.** [21] .

At this point, it is useful to summarize the differences which make the *HSG implementation* better, in terms of performance, respect to the *HOG* one :

- Padding procedure of histogram : starting from an image of size 64×128, divided into blocks 8×8. The procedure to compute both *gradient magnitude*

and *orientation* is equal to HOG one. The main difference is that instead of populating histogram by the use of bilinear interpolated magnitudes, with HSG, **the *average gradient magnitude in the block* is used as parameter** [21]. This value is helpful to define a threshold which allows only those edges to make a binary vote to the *orientation histogram*. *HSG* compared to *EOH* [22], however, shows in more efficient way the shape contours, due to the fact that the only more significant edges are used. A graphical comparison is depicted in figure (2.21):

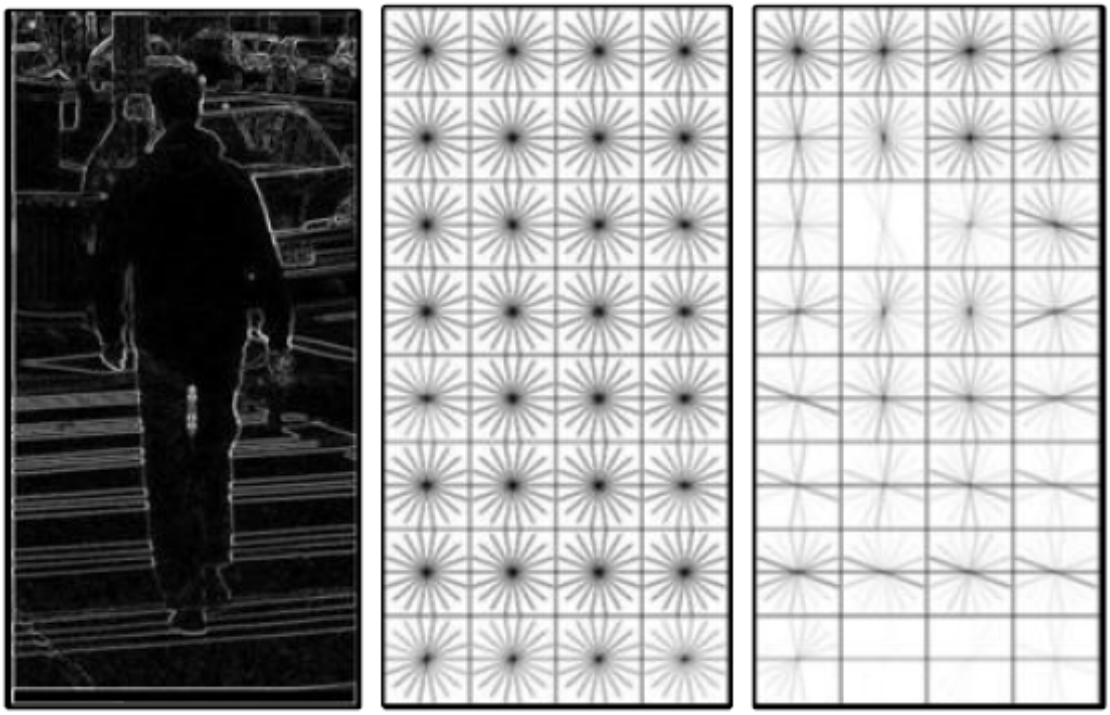


Figure 2.21: **Left : Grayscale reference image. Middle: EOH histograms appears as uniformly distributed with no noticeable shapes, due to the fact that *EOH* allows every gradient to vote [22]. Right : HSG histograms are more related to the shape of objects in the edge image. This is the result of the use of only dominant edges for voting procedure.** [21] .

- Alternative way to implement SVM : in the work proposed by *M.Bilal and et.* [21] another great advantage, in terms of performance, is due to the fact that the *SVM* algorithm is implement by using a *LUT-like* architecture. Adopting

the formulation (2.12) for describe *SVM classification function "h"* :

$$h(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \alpha_j K(x(i),SV_j(i)) \right) + b \tag{2.12}$$

- $\mathbf{x} \triangleq$ input feature vector of n elements
- $\mathbf{SV_j} \triangleq jth$ support vector
- $\mathbf{i} \triangleq$ access index for x and SV_j
- $\alpha_\mathbf{j} \triangleq$ support vector learned coefficient
- $\mathbf{b} \triangleq$ learned bias
- $\mathbf{K} \triangleq$ kernel function [23]

Assuming that $x(i)$ is a linear term and the elements in the bracket can be computed preemptively, it is possible to simplify (2.12) as follows :

$$h_{linear}(x) = \sum_{i=1}^{n} (x(i) * C(i)) + b \tag{2.13}$$

- $\mathbf{C(i)} \triangleq$ precomputed coefficient vector

Looking at equation (2.13) it can be noticed that, for the linear case, *SVM classification* is a simple dot product among input feature vector and a coefficient vector added to the learned bias [21]. Remembering that the *real-time* computation is a mandatory requirement, the speed of the overall system can be increased if a LUT, with only integer values in a limited dynamic range of $x(i)$, is used. An helpful example is to assume that the *kernel function "K"* finds the **minimum of the elements related both input feature vector and the current support vector**. According to [24], the best solution consists to store inside a *2D LUT* all the possible values depending on $x(i)$ (2.12). To this end, the final mathematical expression for the *classification function "h"* is (2.15).

$$T(i,k) = \sum_{j=1}^{m} (\alpha_j min(k,SV_j(i))) \tag{2.14}$$

$$h_{HIK}(x) = \sum_{i=1}^{n}(T(i,x(i))) + b \tag{2.15}$$

Regards to (2.14), the noticeable things are represented to :

a) **T** contains the precomputed values for the full dynamic range of $x(i)$;

b) **T** assumes only positive integer values of $x(i)$ and then k has a range that starts from 0 to the maximum value that $x(i)$ can take;

c) **T** has floating point values stemming from the multiplication between α_j and the integer-values of $min(k,SV_j(i))$.

The most relevant optimization depends on the fact that, thanks to the use of LUT, the number of operations required per classification is only n floating point additions (2.15) instead of *n floating point multiplications plus n floating point additions* needed by *linear SVM* [25]. This leads to obtain a dramatical speedup and also gaining better discrimination power than *HIK SVM* [21].

Chapter 3

Architecture for HSG pedestrian detector

3.1 General Overview

The following hardware implementation of *HSG detector* is the result of the instructions of *M.Bilal and et.* in [21]. This work has the purpose of proposing many dedicated architectural solutions for different sections of the algorithm. This way of thinking leads to implement the structures on one side with a *parallel approach* and on the other side looking at the *power consumptions*. A general description of how the architecture has been thought is depicted in figure (3.1):

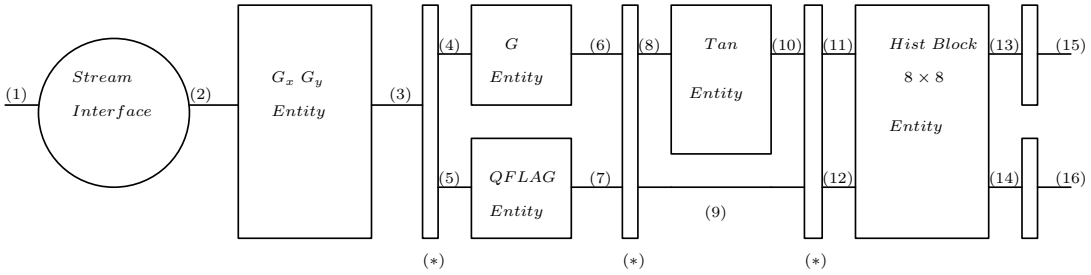


Figure 3.1: **High Level View of *HSG detector***

Proposed implementation, shown in figure (3.1), is customized for elaborating *RGB* images having a resolution of (128×64) pixel, according to [18]. However, the *parametric description* allows to adapt in easy way the entire structure for different, higher resolutions. Focused on the first operation it can been noticed that a conversion from *RGB* to *Gray scale* colour space is done. The goal of this operation is to unwrap the useless informations given by *RGB* image (such as : background colours etc.) because, the detector needs to highlight the pedestrian's

Table 3.1: *Key of (3.1)*

Signal	bits	Signal	bits	Signal	bits	Signal	bits
(1) RGB data	64×24	**(6)** $\|G\|$, $\|G_x\| \times 8$, $\|G_y\| \times 8$, $G_{Threshold}$	$(8 \times 8) \times 4$, $(8 \times 8) \times 16$, $(8 \times 8) \times 16$, 4	**(10)** $gxtan_{20,..,80}$	16	**(14)** Valid Output	1
(2) *Gray scale data*	64×8	**(7)** $QFLAG_{1,..,4}$	16	**(11)** $gxtan'_{20,..,80}$	16	**(15)** $HSG8 \times 8'$	36×4
(3) $\|G_x\|$, $\|G_y\|$	64×9	**(8)** $\|G_x\| \times 8'$	$(8 \times 8) \times 16$	**(12)** $\|G\|''$, $\|G_y\| \times 8''$, $G''_{Threshold}$, $QFLAG''_{1,..,4}$	$(8 \times 8) \times 4$, $(8 \times 8) \times 16$, 4, 16	**(16)** Valid Output'	1
(4),(5) $\|G_x\|'$, $\|G_y\|'$	64×9	**(9)** $\|G\|'$, $\|G_y\| \times 8'$, $G'_{Threshold}$, $QFLAG'_{1,..,4}$	$(8 \times 8) \times 4$, $(8 \times 8) \times 16$, 4, 16	**(13)** $HSG8 \times 8$	36×4	**(*)** Pipe Registers	

shape. A detailed explanation can be read in *section (3.2)*. In the wake of colour space conversion, according to [21], the computation of the gradients is done (see section (3.3)). This operation is useful to derive two important quantities, which:

a) *Magnitude* : expressed as $|G|$;

b) *Sign of* $\left(\frac{G_y}{G_x}\right)$: expressed as $QFLAG_i$ with i=1..4;

The section below, according to [26] (more details in section (3.6)), computes *tangent* values for different angles with the aim of avoiding a wasteful mathematical operation such as the *arctangent* one. Finally, the block called *Histogram* 8×8, executing many operations (see section (3.7)), is able to compute a *1-D* vector composed by 36×4 bits related to the incoming 8×8 pixels block. Another noticeable thing to observe is the *pipe registers* presence and, in particular, their amount. This choice has the purpose of trying stopping the combinational path and then incrementing the overall system performance. In detail, at the end of each section, one pipe register has been placed. This is not a casual positioning, because by adopting proper software (section (5)), the results, in terms of timing, area and power consumption, lead to conclude that solution is nearly to the optimum one. In the end, the two final registers which inputs/outputs are, respectively, ***(13),(14),(15),(16)*** have the goal of ensuring that the internal timing is unrelated to the *external world*.

3.2 Stream Interface architecture

The architecture has been thought on one side to reduce the complexity of the data treated by the downstream blocks and on the other side, by the *FIFO* usage, to prevent the loss of input data. In detail, the *RGB* input data can be expressed as a *3-D* matrix of dimensions : $8 \times 8 \times 3$. After the procedure, these data become a *2-D* matrix of dimensions : 8×8. Speaking about the data parallelism, it means to execute the following reduction : 192 numbers of 8 bits $\rightarrow$ 64 numbers of 8 bits. With the goal of using the minimum amount of hardware resources, the exploited algorithm to do the conversion from *RGB* colour space to the *Gray* one is :

$$Gray = 0.3125 \times R + 0.5625 \times G + 0.125 \times B \tag{3.1}$$

Looking at the eq.(3.3), it can be observed that it is possible to achieve the result by using only simple additions of right shifts of power of two. In particular, eq.(3.3) can be reworded as follows :

$$Gray = R >> 2 + R >> 4 + G >> 1 + G >> 4 + B >> 3 \tag{3.2}$$

A more detailed description of the internal structure of this block is given by the figure (3.2):

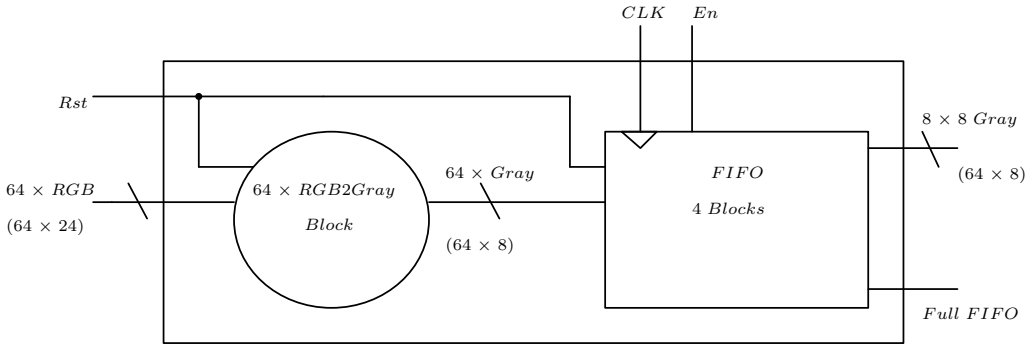


Figure 3.2: **High Level View of *Stream Interface* block**

The behaviour of the *interface* depends both the correct sampling of the *"En"* signal and the presence of valid *"Gray"* input data. According to the first constraint, the generation of *"En"* stems from a video source (such as camera). The timing

diagram, depicted in figure (3.3), clarifies how this architecture works :

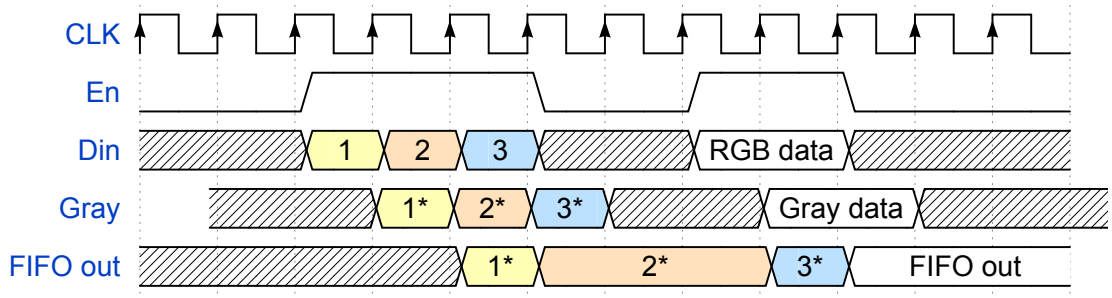


Figure 3.3: **Timing diagram of Stream Interface block**

At this point, it is clear that the combinatorial delay requested by the *RGB2Gray Entity* is a mandatory condition to satisfy, in order to obtain the correct working of the block. Regarding to the figure (3.2), a thing that stands out is the presence of 64 *RGB2Gray* blocks that work in parallel. The eq.(3.2) can be mapped in hardware in many approaches, but, the most efficient one, in terms of asymptotic behaviour, is based on a *tree-like* structure. In this way, it is possible to obtain a total delay $T(n)$, depending on the data parallelism, with a logarithmic law : $T(n) \propto log_2(n)$. In figure (3.4) are represented two possible hardware implementations of *RGB2Gray* block:

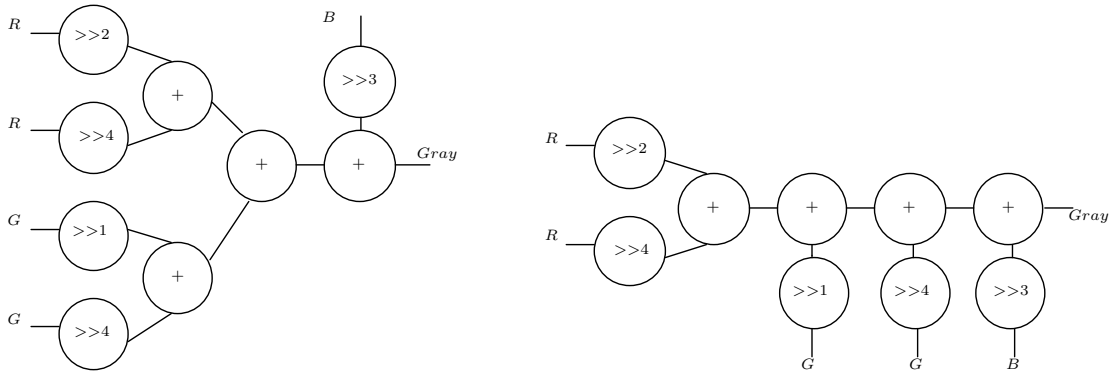


Figure 3.4: **Left : Implementation having a logarithmic behaviour ; Right : Implementation with a linear behaviour.**

Looking at the figure (3.4), a remarkable thing is that the left implementation exhibits a critical path composed by one *shifter block* plus *three adders*, instead

of a delay composed by one *shifter block* and *four adders* inherent to the right solution. The two solutions are equivalent in terms of required hardware resources, but, in order to have a faster conversion, the left representation is the preferred one. To describe the *"logarithmic"* structure, the *VHDL* code was chosen and its implementation is the following one :

```
-------RGB2Gray Entity-----------------
RGB2Gray_process : process(R,G,B,Rst)
variable R_int,G_int,R_G_int : std_logic_vector(7 downto 0);
begin
if Rst = '1' then
Gray <= (others=>'0');
R_int := (others=>'0');
G_int := (others=>'0');
R_G_int := (others=>'0');
else
--Gray = 0.3125*R + 0.5625*G + 0.125*B-------
R_int := std_logic_vector(shift_right(unsigned(R),2)+shift_right(unsigned(R),4));
G_int := std_logic_vector(shift_right(unsigned(G),1)+shift_right(unsigned(G),4));
R_G_int := R_int + G_int;
Gray <= R_G_int + std_logic_vector(shift_right(unsigned(B),3));
end if;
end process RGB2Gray_process;
-----------------------------------------
```

In the end, the brief description of the *FIFO (First Input First Ouput) memory*, proposed in figures (3.5) and (3.6), is useful to clarify its functioning.

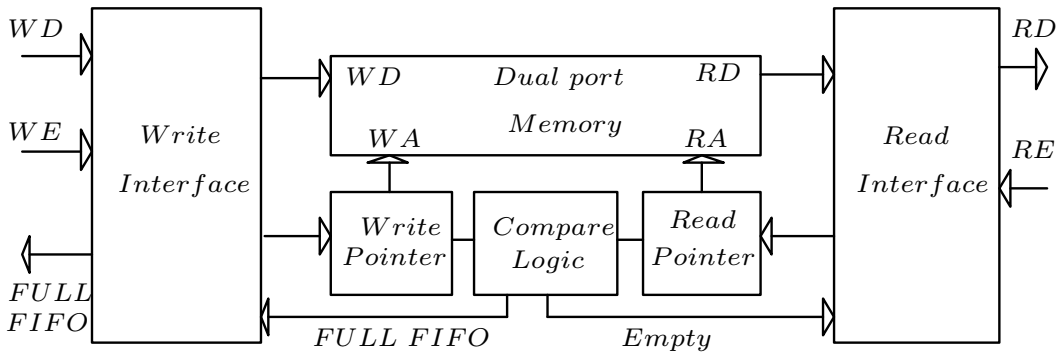


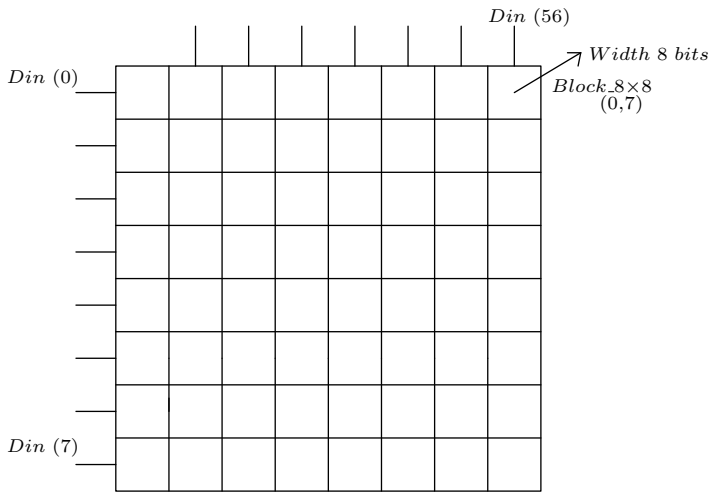
Figure 3.5: **(1) Block diagram of a FIFO Memory**.

One data pushed to fifo:							
RP	**WP**						
5 more data pushed to fifo							
RP						**WP**	
3 data popped from fifo							
			RP			**WP**	
3 more data pushed to fifo							
	WP		**RP**				
2 more data pushed to fifo: **fifo full**							
			WP, RP				
6 data popped from fifo							
	RP		**WP**				

Figure 3.6: **FIFO working explanation [27], where WP = Write Pointer and RP = Read Pointer**.

The proposed array size (*4 Blocks*) is the optimal one and it was obtained through many simulations, paying attention to the *FULL FIFO* flag. In presence of an active *FULL FIFO*, the current input stream is irrecoverably lost.

3.3 Gradients computation architecture




```
-------INPUT DATA FROM STREAM INTERFACE-------
        for r in 0 to 7 loop
        Block_8x8(r,0) := Din(r);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,1) := Din(r+8);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,2) := Din(r+16);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,3) := Din(r+24);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,4) := Din(r+32);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,5) := Din(r+40);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,6) := Din(r+48);
        end loop;
        for r in 0 to 7 loop
        Block_8x8(r,7) := Din(r+56);
        end loop;
-----------------------------------------------
```

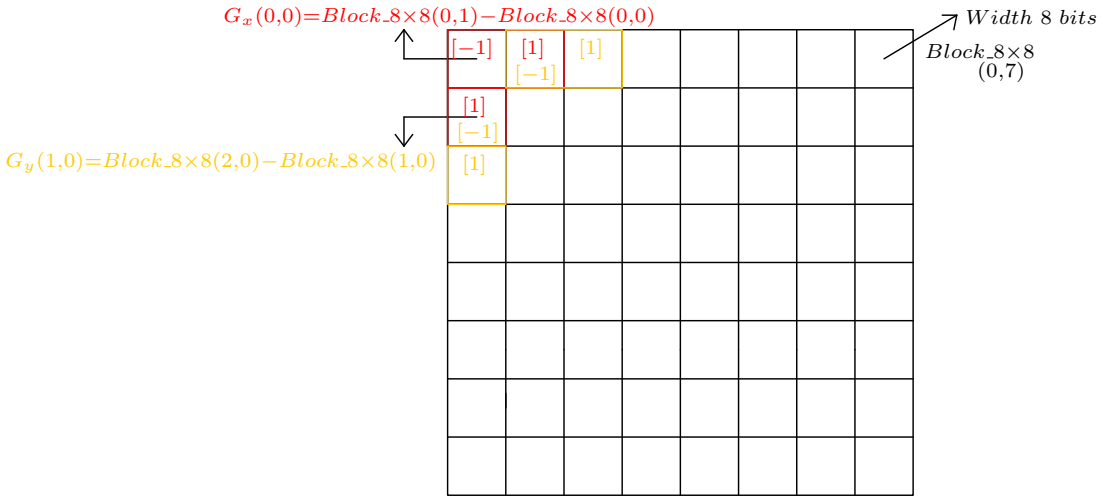
Figure 3.7: **(1) Parallel Load of 2-D Matrix, (2) VHDL Description**.

According to the figure (3.1) (*signal (2)*), this architecture has been thought to work with blocks having, respectively 8 rows and 8 columns of data with a width equal to 8 bits (*"unsigned" representation in the Gray colour space*). The first operation requires the filling of a *2-D* matrix. To do this, a parallel approach to

describe the hardware was exploited, as clearly explained in figure (3.7) and the relative code (3.7). This particular choice allows to execute the entire procedure in one clock cycle speeding up the system performance. In the wake of the filling procedure, the emphasis shifts to the gradients computation. With the goal of using the minimum number of hardware resources, instead of use more complex kernels (such as (2.10)), the following very simple masks were exploited:

$$\left\{ \begin{bmatrix} -1 & 1 \end{bmatrix} \quad and \quad \begin{bmatrix} -1 \\ 1 \end{bmatrix} \right. \tag{3.3}$$

where the left one is used for G_x and the right one for G_y.




```
--Gx and Gy Matrix by using 'intermediate' technique--
for r in 0 to 7 loop
        for c in 0 to 6 loop
Gx(r,c) <= (Block_8x8(r,c+1))-(Block_8x8(r,c));
        end loop;
end loop;
for c in 0 to 7 loop
        for r in 0 to 6 loop
Gy(r,c) <= (Block_8x8(r+1,c))-(Block_8x8(r,c));
        end loop;
end loop;
```

Figure 3.8: **(1) Graphical representation of Gradients computation, (2) VHDL Description**.

A noticeable thing is related to the fact that also if this technique is very basic,

it is able to produce very accurate results (see section (4)). Regarding to the used hardware resources, this block requires :

- *Block*_8 × 8(r,c) : 64 registers (each of them of 8 bits) to store the input data;
- 128: 2's complement adders to compute both G_x and G_y;
- $G_x(r,c)$ and $G_y(r,c)$: 128 registers (each of them of 8 bits) to store the computed gradients values.

In the end, the timing diagram, depicted in figure (3.9), clarifies the behaviour of the entire section:

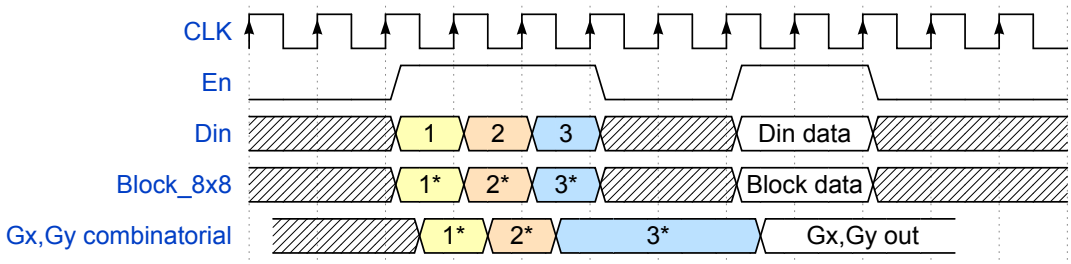
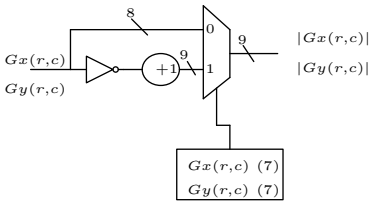


Figure 3.9: **Timing diagram of G_x G_y block**

Looking at the above timing diagram it is clear that, thanks to the parallel approach, all the needed operations are accomplished in only one clock cycle. In detail, the critical path of this section is equal to one adder (*subtractor*), having inputs on 8 bits.

3.4 Magnitude and Threshold computation architecture

This section has as inputs two blocks 8×8, respectively for G_x and G_y. According to [21], with the aim of saving hardware resources, the *magnitude computation* is accomplished by using the following approximate expression : $|G| = |G_x| + |G_y|$. Looking at the *"sign bit" (MSB)* of G_x and G_y it is possible to compute their absolute values. For reasons that will become clear later, in this section also a left shift of a factor 3 (in other words, a multiplication by 8) has been performed for either gradients. The architecture and code, in figure (3.10), have the aim of explaining in which way the first duty is accomplished:




```
--------Gradient ABS COMPUTATION--------
for k in 0 to 7 loop
for l in 0 to 7 loop
--------2's complement--------------------
if (Gx(k,l)(7) = '1') then
Gx_abs_int(k,l) := not(Gx(k,l))+1;
else
Gx_abs_int(k,l) := Gx(k,l);
end if;
if (Gy(k,l)(7) = '1') then
Gy_abs_int(k,l) := not(Gy(k,l))+1;
else
Gy_abs_int(k,l) := Gy(k,l);
end if;
-----------------------------------------
end loop;
end loop;
-----------------------------------------
```

Figure 3.10: **Gradients absolute value computation**

Regarding to the above mentioned section code, the hardware costs can be summarized as follows:

- Inverter : 64×2;
- Adder : 64×2;
- Mux : 64×2;
- Register : 64×2.

In detail, exploiting the parallel approach, this computation is done in one clock cycle with a critical path composed by an inverter plus one adder and a *mux*.

```
----Gx e Gy AUGMENTED PRECISION----------
for k in 0 to 7 loop
for l in 0 to 7 loop
Gx_abs_x8(k,l) <= shift_left(unsigned(Gx_abs_int(k,l)),3);
Gy_abs_x8(k,l) <= shift_left(unsigned(Gy_abs_int(k,l)),3);
end loop;
end loop;
-----------------------------------------
```

Figure 3.11: **Left shift of Gradients absolute value**

The code (3.11) represents a *"trick"* to overcome the poor resolution arising from the precomputed *tangent values* (see section 3.6). In particular, basing on experimental results (see chapter 4), the parallelism of each output data was chosen equal to 16 bits. In the end, this block is able to compute a peculiar parameter, called *"G_Threshold"*. This value is *crucial* to compute in correct way [21] the various *"orientation histograms"*. Regarding to the hardware implementation, the parallel approach has been chosen and the proposed architecture is depicted in figure (3.12)

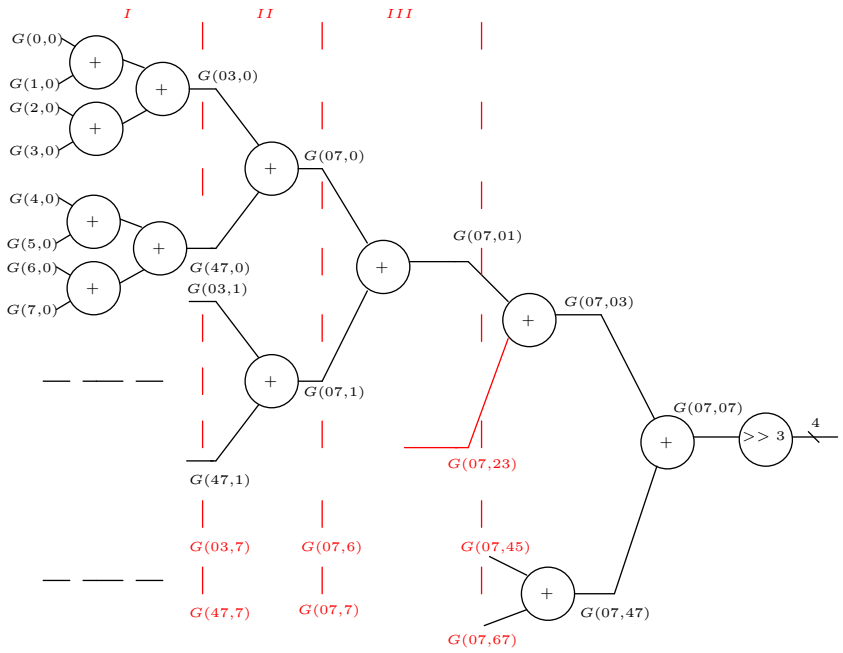


Figure 3.12: **Threshold computation parallel architecture**

Looking at figure (3.12) it is possible to estimate a critical path composed by 6 adders plus a right shifter (its aim is to realize a division of the final result by a factor 64). The needed hardware can be summarized as follows :

- Section I : $(4 \times 8) + (2 \times 8)$ adders;
- Section II : 8 adders;
- Section III : 4 adders;
- Final Section : $2 + 1$ adders.

3.5 QFLAG architecture

This block, starting from G_x and G_y values, is able to give an information about the *sign* of $\frac{G_y}{G_x}$ ratio. This kind of parameter is very useful to associate the vote to the correct histogram *bin*. In order to speed up this computation, a parallel approach has been exploited. In particular, the incoming 8×8 block has been divided into four 4×4 blocks. The basic idea is to compare the MSBs of the two inputs, subsequently of these results, proper registers are filled with $0s$ or $1s$. With the aim of clarifying how this operation is done, the proposed architecture is depicted in the following figure (3.13):

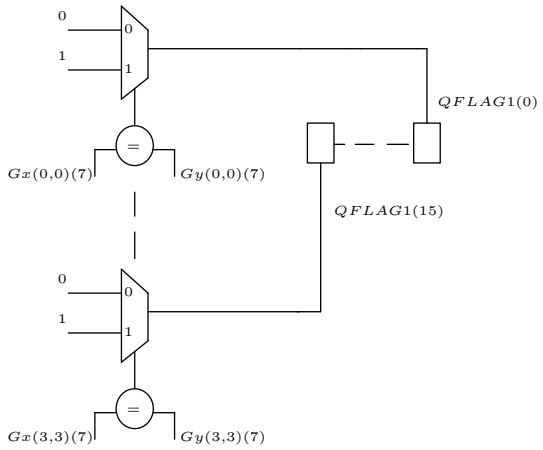


Figure 3.13: **QFLAG computation parallel architecture**

By a simple graphical inspection, the hardware cost can be summarized as follows:

- Mux : 16×4;
- Comparators : 16×4;
- Registers : 4 of 16 bits.

In detail, the critical path of this architecture is very small, in fact it is composed by a $1 - bit$ comparator plus a 2to1 mux.

3.6 Tangent computation architecture

This section speeds-up significantly the performance of the overall system. The main drawback, in terms of delay and complexity, for the other kinds of hardware implementations is represented by the computation of the arctangent function. For the algorithm nature, the orientation values, expressed as:

$$\Theta = atan\left(\frac{G_y}{G_x}\right) \tag{3.4}$$

are used to fill correctly the histogram bins. Various hardware implementations in literature uses complex architectures such as *CORDIC* [28] to do it, but, in many cases the reached precision is worthless. In order to overcome this issue, following the instructions written in [26] it is possible to avoid the arctangent computation. In detail, the solution consists to solve the inequality (3.5), instead of the above mentioned trigonometric function. According to [26], the mathematical definition of $tan(\Theta)$ has a very interesting characteristic which is represented by the following inequality:

$$tan(\Theta_i) \leq tan(\Theta) < tan(\Theta_{i+1})$$

By harnessing (3.4), it is possible to write the final expression :

$$G_x(x,y) * tan(\Theta_i) \leq G_y(x,y) < G_x(x,y) * tan(\Theta_{i+1}) \tag{3.5}$$

By adopting the so-called *unsigned representation*, it is possible to limit the calculation for a restricted range of angles that starts from 0° to 180°. Following [21], the above mentioned range is divided into 9 bins, it means to have a representation like the figure (3.14):

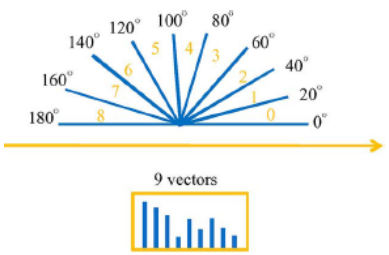


Figure 3.14: **Angle to Bin conversion** [26]

In detail, as will become clear in the section that involves the *Histogram* 8×8 computation, the data produced by the *"QFLAG architecture"* are helpful to reduce the amount of values to compute. Regarding to the previous concern, the approximate values computed by this architecture can be summarized through the table (3.2):

Table 3.2: *Approximate Values of tanΘ*

tangent	**Approximate value**
$tan0°$	0
$tan20°$	$2^{-2} + 2^{-3}$
$tan40°$	$2^{-1} + 2^{-2} + 2^{-4}$
$tan60°$	$1 + 2^{-1} + 2^{-2}$
$tan80°$	$2^{2} + 1 + 2^{-1} + 2^{-3} + 2^{-5}$

Looking at the above table the noticeable things are:

- Power of two relations: the five required values are obtained only by simple shift and add operations. It means that the effort to obtain the final result is very limited;
- Parallel Approach: this idea to implement in hardware the relations is the best one to obtain all results in one clock cycle.

Talking about the proposed parallel approach architecture, an example is depicted in figure (3.15):

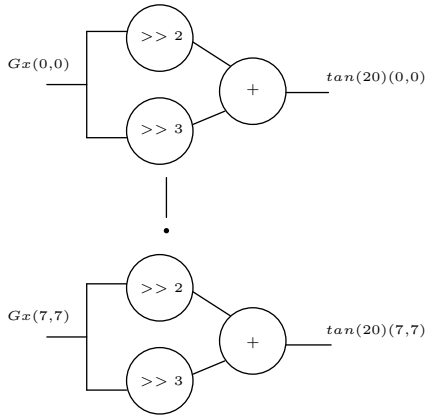


Figure 3.15: **Tangent approximate architecture for tan**$20°$

The related code that describes entirely the structure is the following one:

```
        ----------TANGENT APPROXIMATE ENTITY----
        for k in 0 to 7 loop
                for l in 0 to 7 loop
gxtan20(k,l)<=std_logic_vector(resize(unsigned(shift_right((Gx_abs_x8(k,l)),2)+
shift_right((Gx_abs_x8(k,l)),3)),16));
gxtan40(k,l)<=std_logic_vector(resize(unsigned(shift_right((Gx_abs_x8(k,l)),1)+
shift_right((Gx_abs_x8(k,l)),2)+
shift_right((Gx_abs_x8(k,l)),4)),16));
gxtan60(k,l)<=std_logic_vector(resize(unsigned((Gx_abs_x8(k,l))+
shift_right((Gx_abs_x8(k,l)),1)+shift_right((Gx_abs_x8(k,l)),2)),16));
gxtan80(k,l)<=std_logic_vector(resize(unsigned(shift_left((Gx_abs_x8(k,l)),2)+
(Gx_abs_x8(k,l))+shift_right((Gx_abs_x8(k,l)),1)+
shift_right((Gx_abs_x8(k,l)),3)+shift_right((Gx_abs_x8(k,l)),5)),16));
                end loop;
        end loop;
        ---------------------------------------
```

Figure 3.16: **Tangent values computed starting from the *"augment precision"* version of G_x values**

In code (3.16) can be seen that the final results are on 16 bits. This internal parallelism, based on experiments (see chapter(4)), produces the most accurate results. In the end, the hardware cost can be estimated in this way :

Table 3.3: **Summary of Tangent approximate architecture hardware cost**

tangent	**Hardware Cost**
*tan*0°	0
*tan*20°	$64 \times 4 \times 2^{-2}$ *shifter* + $64 \times 4 \times 2^{-3}$ *shifter* + 64×4 *adder*
*tan*40°	$64 \times 4 \times 2^{-1}$ *shifter* + $64 \times 4 \times 2^{-2}$ *shifter*+$64 \times 4 \times 2^{-4}$ *shifter* + $64 \times 4 \times 2$ *adder*
*tan*60°	64× *connection* +$64 \times 4 \times 2^{-1}$ *shifter*+$64 \times 4 \times 2^{-2}$ *shifter* + $64 \times 4 \times 2$ *adder*
*tan*80°	$64 \times 4 \times 2^{2}$ *shifter*+64× *connection* + $64 \times 4 \times 2^{-1}$ *shifter*+$64 \times 4 \times 2^{-3}$ *shifter*+$64 \times 4 \times 2^{-5}$ *shifter* + $64 \times 4 \times 4$ *adder*

3.7 Histogram 8×8 block architecture

This final section has the aim of producing a vector of 36 elements, each of them represented on 4 bits. The amount of 36 elements is related to the fact that the entire 8×8 block is divided in 4 blocks having dimensions 4×4. In detail, each 4×4 block produces a vector of 9 elements, according to the explanation given in section (3.6). Following the algorithm in [21], every output block is called *"histogram"*. The simple nature of its computation can be depicted through the flow diagram (3.17):

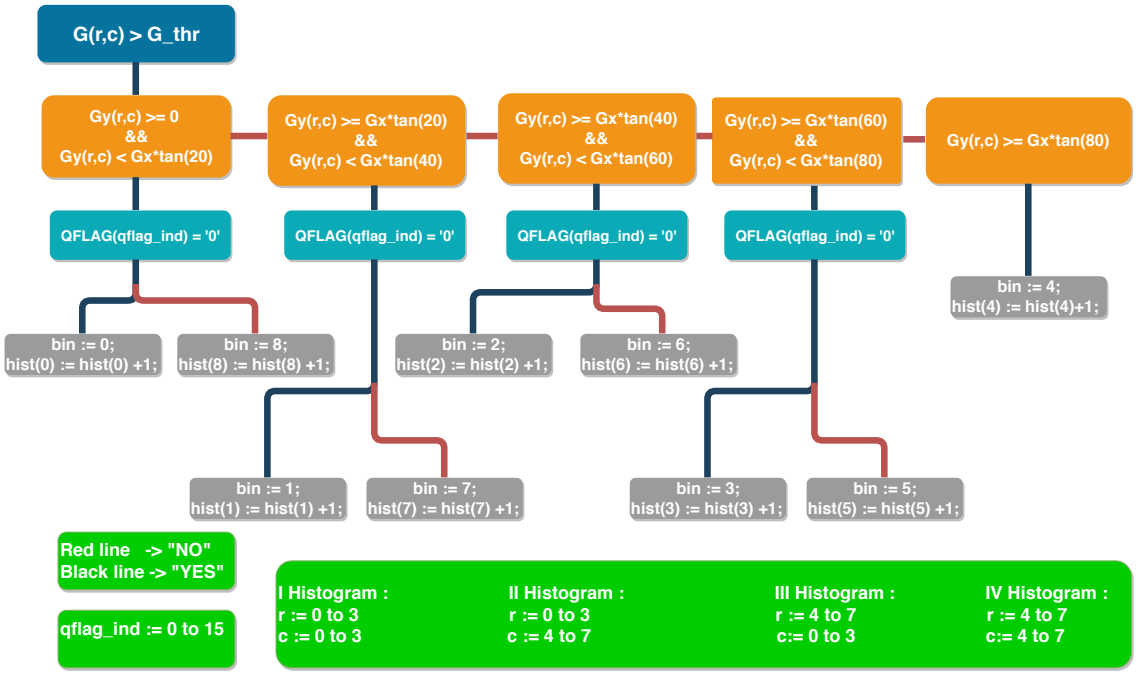


Figure 3.17: **Flow diagram to compute an histogram based on HSG**

Chapter 4

Testbench and Simulations

4.1 Testbench environment

With the aim of verifying the correct behaviour of the proposed architecture, the testbench environment is depicted in figure (4.1):

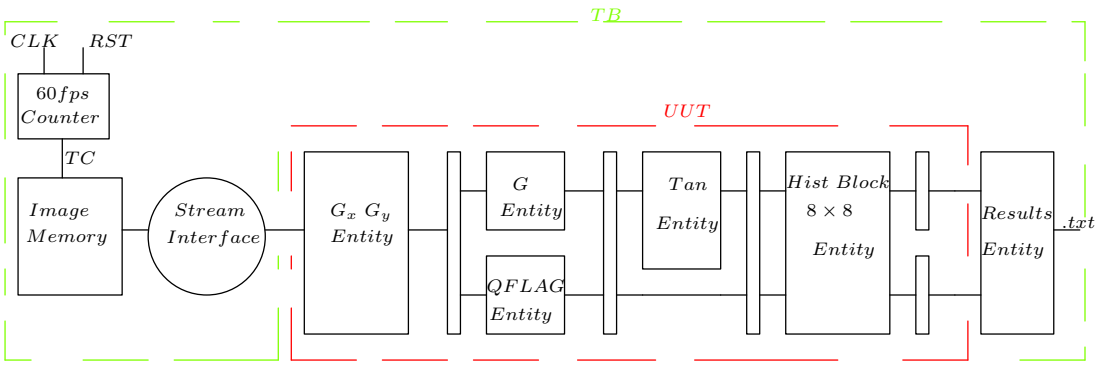


Figure 4.1: **Testbench environment of the developed architecture**

The *60fps Counter* has the purpose of ensuring that the overall structure is *"nourished"* with 60 frames per second.

Image Memory

This element has been thought to the aim of feeding the system with an image having a rate equal to $60fps$. To do this a *Matlab* script has been developed to generate an *.hex* file, where each line represents a single *24-bits RGB pixel* of the reference image. In detail, the proposed code is the following one (4.2):

```
function [RGB,RGB_hex,Gray_mat]=RGB_hex(rgb_image)

%%%Three main components%%%%%%%%%
red_channel = rgb_image(:,:,1);
green_channel = rgb_image(:,:,2);
blue_channel = rgb_image(:,:,3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%3 Matrixes each of 8x1024 elements%%%%%%%%
%%%%Image 128x64%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for r = 1:1:16 %%%128 rows divided by 8
if r == 1
a = 1;
b = 8;
a1 = 1;
b1 = 8;
c_in = 1;
c_end= 8;
else
a = 1;
b = 8;
c_in=c_in+8;
c_end=c_in+7;
end
for c = 1:1:8 %%%64 columns divided by 8
Block_R(1:8,a1:b1) = red_channel(c_in:c_end,a:b);
Block_G(1:8,a1:b1) = green_channel(c_in:c_end,a:b);
Block_B(1:8,a1:b1) = blue_channel(c_in:c_end,a:b);
a1 = a1+8;
b1 = a1+7;
a=a+8;
b=a+7;
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%8x8 blocks (fixed columns and varying row)%%
RGB = cat(3,Block_R,Block_G,Block_B);
% hex version, where i.e Red = 4E Green = 5B Blue = 66%
RGB_hex = [ dec2hex(RGB(:,:,1)) dec2hex(RGB(:,:,2)) dec2hex(RGB(:,:,3)) ];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

%%%%.txt file%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dlmwrite('./VHDL_input/img.ixxx/RGBHex.txt',RGB_hex,'delimiter','');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 4.2: **Matlab script able to generate an .hex file that represents an *RGB* reference image**

The *VHDL* memory has the capability to *"read and store"* the *RGB.txt* file content. In detail, at each clock cycle, according to specified conditions, the output memory is composed by a block of 8×8 values. A portion of the code that implements this kind of memory is (4.3):

```
type rom_array is array (0 to m-1) of std_logic_vector(n-1 downto 0);

impure function read_data_ROM return rom_array is

file infile1: text open READ_MODE is "./tb/RGBHex.txt";
variable buf1 :line;
variable value1 : std_logic_vector(n-1 downto 0);
variable ROM:rom_array:=(others=>(others=>'0'));
variable I : integer range 0 to m ;

begin

while not endfile(infile1) loop
readline(infile1, buf1); ---infile1 in buf1
hread(buf1, value1);    ---buf1 hex in std_logic_vector value1
ROM(I):=std_logic_vector(unsigned(value1));
I := I+1;
end loop;
return ROM;
end function;

constant ROM: rom_array:=read_data_ROM;
```

Figure 4.3: **Portion code that explains how the Image Memory is able to read and store the content of a *.txt* file**

Figure 4.4: **Left: Reference image; Right: Image sent to the VHDL architecture**

Results Entity

This block was described to give a graphical meaning to the outputs produced by the proposed architecture. This entity is composed by two processes, divided as follows:

I) Gradients process: the behavioural description is reported in (4.5):

```
Gx_Gy_results_proc: process(Gx,Gy,Rst)
file Gx_file : text open WRITE_MODE is "./results/Gx_results.txt";
file Gy_file : text open WRITE_MODE is "./results/Gy_results.txt";
variable Gx_line, Gy_line : line;
variable count_line : integer range 0 to 8191;
begin -- process
if Rst = '1' then     -- asynchronous reset (active high)
null;
valid_Gx_Gy <= 0;
count_line := 0;
else
if valid_Gx_Gy = 0 then
valid_Gx_Gy <= valid_Gx_Gy + 1;
end if;

if valid_Gx_Gy > 0 and count_line < 8191 then
for k in 0 to 7 loop
for j in 0 to 7 loop
count_line := count_line + 1;
write(Gx_line, to_integer(signed(Gx(k,j))));
writeline(Gx_file, Gx_line);
write(Gy_line, to_integer(signed(Gy(k,j))));
writeline(Gy_file, Gy_line);
end loop;
end loop;
end if;
end if;
end process;
```

Figure 4.5: **Reading procedure of Gx, Gy values computed by the VHDL architecture**

Looking at the above code (4.5), it is possible to observe the presence of particular signals, like : *valid_Gx_Gy* and *count_line*. The first mentioned allows to discard the initial no valid outputs (due to the pipelined structure), instead, the second one has the aim of validating the end of the procedure based on 128 blocks, each of them of 8×8 dimensions (i.e. a 128×64 image produces a vector of length equal to 8192 elements). At this point, in order to depict in graphical way these results a Matlab was developed (4.6) :

```
function [Gx_VHDL,Gy_VHDL] = Gx_Gy_VHDL(Gx_VHDL_temp,Gx_VHDL_vector,Gy_VHDL_temp,Gy_VHDL_vector)

Gx_VHDL = zeros(128,64);
Gy_VHDL = zeros(128,64);

%%%%%%%%%%%%%%%%%%%%%%%%128x64 Gx,Gy VHDL Matrixes%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for r_V_t = 1:1:1024 %%%8192 elements to 1024x8 matrix
if r_V_t == 1
r_V_v_t = r_V_t;
else
r_V_v_t = r_V_v_t + 8;
end
Gx_VHDL_temp(r_V_t,1:8)= (Gx_VHDL_vector(r_V_v_t:(r_V_v_t+7)));
Gy_VHDL_temp(r_V_t,1:8)= (Gy_VHDL_vector(r_V_v_t:(r_V_v_t+7)));
end
% Gx_VHDL(1:8,1:8) = Gx_VHDL_temp(1:8,1:8);
% Gx_VHDL(1:8,9:16)= Gx_VHDL_temp(9:16,1:8);
start = 1;
for r_VHDL = 1:1:16 %%% 16*8= 128 rows (r_V_int+8) , 8*8= 64 columns (c_V_int+8)
if r_VHDL == 1
r_V_int = r_VHDL;
else
r_V_int = r_V_int+8;
end
for c_VHDL = 1:1:8
if c_VHDL == 1
c_V_int = c_VHDL;
else
c_V_int = c_V_int+8;
end
if c_VHDL == 1 && start == 1
ind_V_int = c_VHDL;
start = 0; %%%%restart the counting procedure
else
ind_V_int = ind_V_int+8;
end
Gx_VHDL(r_V_int:r_V_int+7,c_V_int:c_V_int+7) = (Gx_VHDL_temp(ind_V_int:ind_V_int+7,1:8));
Gy_VHDL(r_V_int:r_V_int+7,c_V_int:c_V_int+7) = (Gy_VHDL_temp(ind_V_int:ind_V_int+7,1:8));
end
end

end
```

Figure 4.6: **Matlab script to draw the G_x,G_y values coming from VHDL architecture**

These computations are not influenced by any rounding procedure, to verify it, the Matlab command *isequal*(G_x VHDL,G_x Matlab) was issued. As expected, the answer was 1, due to the fact that the architecture executes the same operations of the Matlab algorithm ('intermediate'). In the end, figure (4.7) summarizes the operations flow :

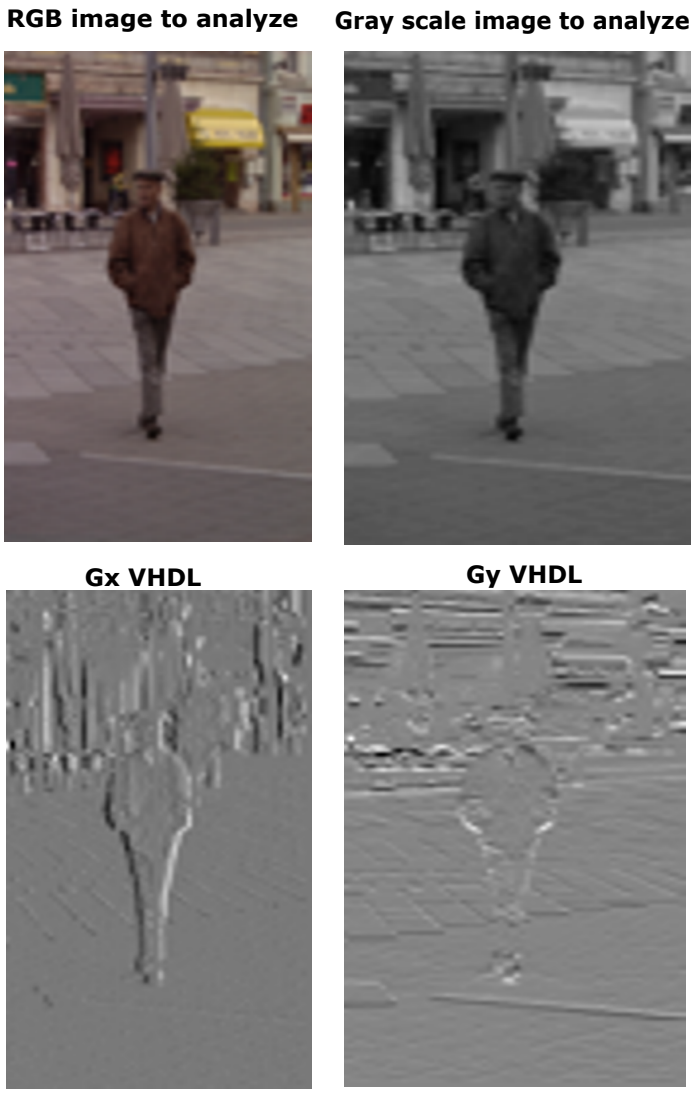


Figure 4.7: **Operation flow**

Starting from the RGB image, the first operation is the Gray scale conversion (top right). In the bottom left, a gradient image in the x direction measuring horizontal change in intensity. On the right, a gradient image in the y direction measuring vertical change in intensity. A useful consideration is that gray pixels have a small gradient; black or white pixels have a large gradient.

II) HSG final result process: the behavioural description is reported in (4.8):

```
HSG_results_proc: process(CLK,Rst)
file HSG_file : text open WRITE_MODE is "./results/HSG_results.txt";
variable count_elements : integer range 0 to 4608;
variable HSG_line : line;
begin -- process
if Rst = '1' then                    -- asynchronous reset (active low)
null;
END_SIM <= '0';
count_elements := 0;
elsif CLK'event and CLK = '1' then
if Valid_Output = '1' and count_elements < 4607 then
for k in 0 to 35 loop
count_elements := count_elements + 1;
write(HSG_line, to_integer(unsigned(HSG_feature_vector_int(k))));
writeline(HSG_file, HSG_line);
end loop;
elsif count_elements > 4607 then
END_SIM <= '1';
end if;
end if;
end process;
```

Figure 4.8: **Generation of the final result of the HSG algorithm**

According to the *HSG* algorithm [21], for every iteration four vectors, each of them of 9 elements, are produced, it means to have 36 values for a single iteration. The total procedure, for a 128×64 image, requires to work on 128 blocks (each of them of 8×8 dimensions). It means to have, in the end, $128 \times 36 = 4608$ values. The above code has the aim of producing a text file containing all produced values with a vector-like style. At this point, in order to depict in graphical way these results a Matlab was developed (4.9) :

```
function [HSG_VHDL] = HSG_VHDL_plot(HSG_VHDL_vector)

HSG_VHDL=zeros(32,16,9);
HSG_VHDL_temp = zeros(512,9);
HSG_vector = HSG_VHDL_vector';

start_H = 1;

%%512X9 MATRIX WHERE EACH COLUMN IDENTIFIES IS EQUAL TO A BIN %%%%%%
for r = 0:1:511
for c = 0:1:8
if c==0 && start_H ==1
c_int = c+1;
start_H = 0;
else
c_int = c_int+1;
end
HSG_VHDL_temp(r+1,c+1) = HSG_vector(c_int);
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start_A = 1;

%%%%%%%%%%%%%%THE OVERALL IS OBTAINED BY FOLLOWING THIS RULE : %%%%
%%%%%%%%%%%%%%4 HISTOGRAMS OF A BLOCK ARE PLACED IN THIS ORDER %%%%
%%%%%%%%%%%%%%%%%%%%[r_0 ,(c_0 ,c_1 )](hist-1,hist-2)%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%[r_1 ,(c_0 ,c_1 )](hist-3,hist-4)%%%%%%%%%%%%%%%%%%%
for i = 0:2:30
for j = 0:2:14
for r = 0:1:1
for c = 0:1:1
if c == 0 && start_A == 1
start_A = 0;
acc = c+1;
else
acc = acc+1;
end
HSG_VHDL(r+i+1,c+j+1,1)=HSG_VHDL_temp(acc,1);
HSG_VHDL(r+i+1,c+j+1,2)=HSG_VHDL_temp(acc,2);
HSG_VHDL(r+i+1,c+j+1,3)=HSG_VHDL_temp(acc,3);
HSG_VHDL(r+i+1,c+j+1,4)=HSG_VHDL_temp(acc,4);
HSG_VHDL(r+i+1,c+j+1,5)=HSG_VHDL_temp(acc,5);
HSG_VHDL(r+i+1,c+j+1,6)=HSG_VHDL_temp(acc,6);
HSG_VHDL(r+i+1,c+j+1,7)=HSG_VHDL_temp(acc,7);
HSG_VHDL(r+i+1,c+j+1,8)=HSG_VHDL_temp(acc,8);
HSG_VHDL(r+i+1,c+j+1,9)=HSG_VHDL_temp(acc,9);
end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

Figure 4.9: **Portion of the Matlab script for obtain a graphic visualization of the HSG descriptor output**

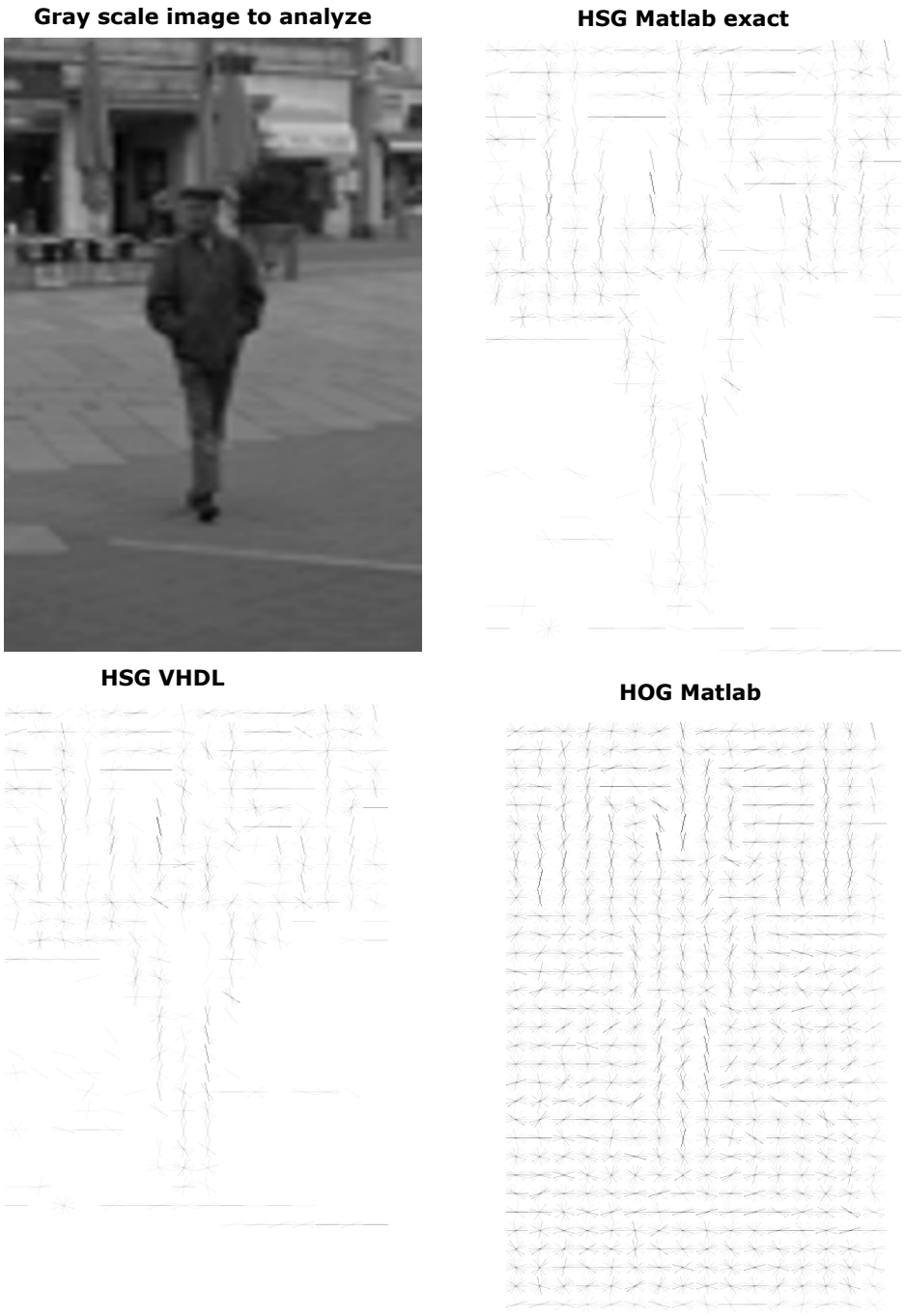


Figure 4.10: **HSG and HOG comparison final results**

Regarding to the figure (4.10), it can be noticed how HSG algorithm is "more descriptive" than HOG one. The great difference is due to the fact that the

"vote" is executed based on a threshold and then only relatively big intensity variations are reported in the final histogram. In particular, comparing the figure named *"HSG Matlab exact"* and *"HSG VHDL"* it is possible to notice slight differences. The reason is that in the first case both the magnitude and orientation are computed in exact way by using, respectively, the equations $|G| = \sqrt{G_x^2 + G_y^2}$ and $\Theta = atan\left(\frac{G_y}{G_x}\right)$.

4.2 Simulation environment

With the goal of checking the correct behaviour of the overall system in the time domain, *ModelSim* was used. The minimum size of *"60 fps counter"* that allows to work at 60 fps, was obtained by solving this equation :

- $\left(\frac{1}{128 \times 60}\right) = \left(\frac{f_{ck}}{N}\right)$
- $f_{ck} = 1$ MHz
- $N = 130 -> 8$ bits

The timing diagrams that clarify how the architecture works are reported in figures (4.11):

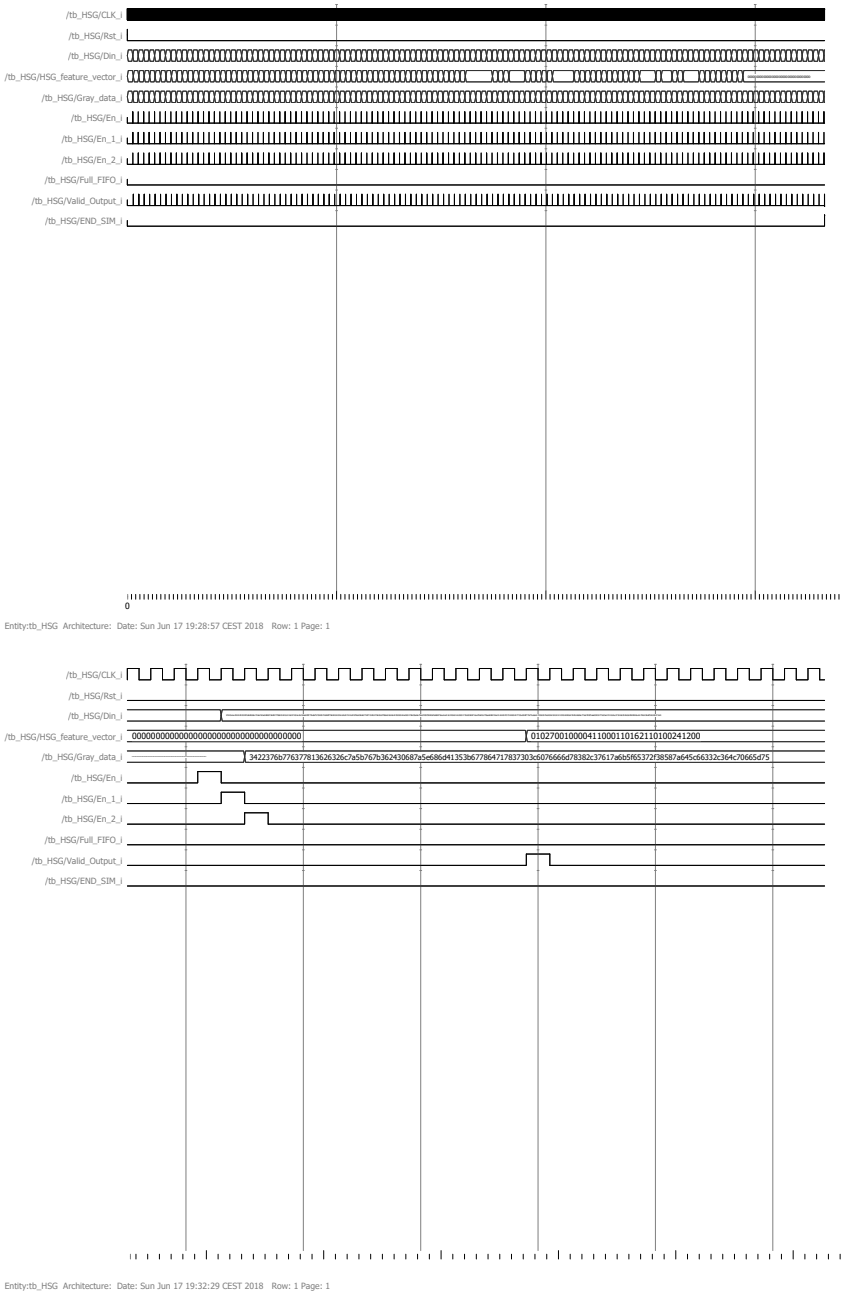


Figure 4.11: **Timing diagram of the overall architecture**

Chapter 5

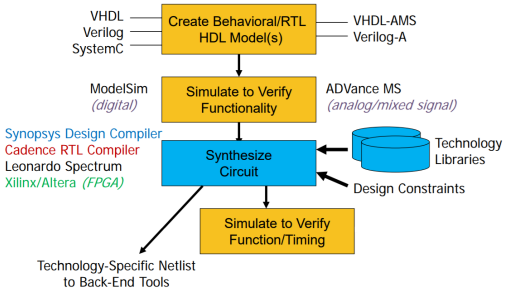
RTL Synthesis and Test

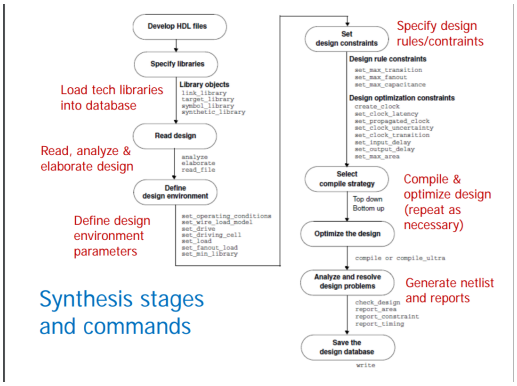
The final step of testing is carried out by adopting *Synopsys Design Compiler©*. This software is able to extrapolate two kinds of important informations from the provided VHDL description, such as :

- Netlist as .v file : a netlist is a description of the connectivity of an electronic circuit. In its simplest form, a netlist consists of a list of the electronic components in a circuit and a list of the nodes they are connected to. [29]

- *.sdf (Synopsys© delay format) file* : SDF file contains the delay value of each timing arc corresponding to each cell in the netlist. These delay values in the SDF file are extracted under a given conditions of the netlist. It may be that the SDF corresponds to just an after synthesis netlist, with wire loads estimated according to some wire load model, or it may be that the SDF corresponds to a neltist which has been laid out, with actual position of cell, actual load on the cell, actual metal wires connected to the cells. [30]

In figure (5.1) is explained the typical flow of operations which allows to map the behavioural architecture into a sequence of physic basic cells:





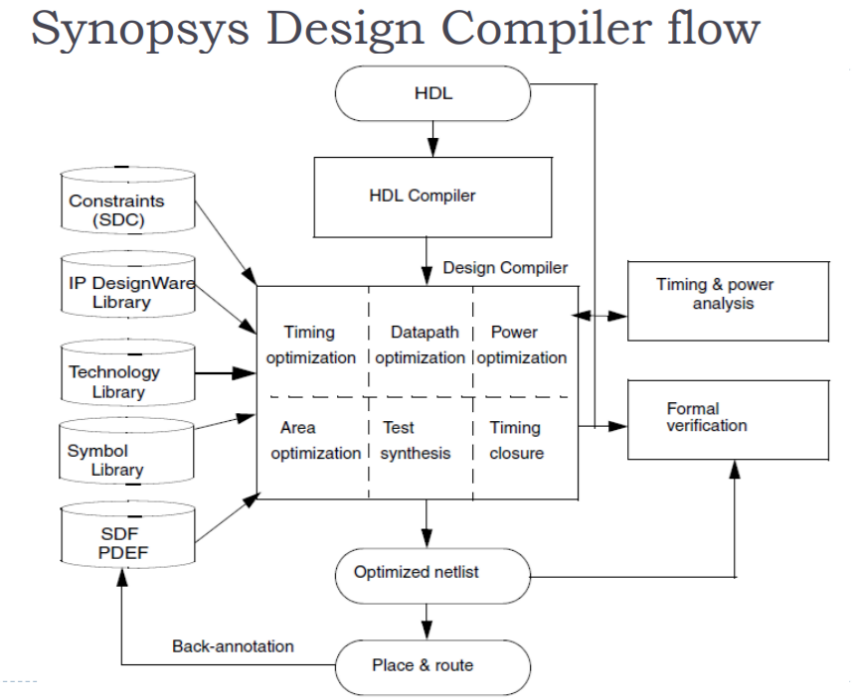


Figure 5.1: **Scheme of how a netlist is generated [31]**

In particular, the used *Technology Library* is *"uk65lscllmvbbr_120c25_tc"* with *BUFM2R* as basic cell. The produced netlist was tested on *ModelSim* to check the correct behaviour. With the purpose of obtaining a power estimation of the system, a proper script was used inside the Verilog Testbench (5.2):

```
initial begin
$read_lib_saif("./saif/uk65.saif");
$set_gate_level_monitoring("on");
$set_toggle_region(UUT);
$toggle_start;
end

always @ ( END_SIM_i ) begin
if (END_SIM_i) begin
$toggle_stop;
$toggle_report("./saif/HSG_back.saif", 1.0e-9, "tb_HSG.UUT");
end
end
```

Figure 5.2: **UUT monitored to estimate its switching activity**

In the following tables (5.1 and 5.2) are summarized the most interesting results

obtained later on the synthesis procedure :

Table 5.1: HSG @maximum frequency

HSG_65nm @ maximum frequency	
frequency (MHz)	***297.61***
power (mW)	***23.17***
area (μm^2)	***178396***

Table 5.2: HSG @1MHz

HSG_65nm @ 1MHz (candidate working frequency)	
frequency (MHz)	***1***
power (μW)	***87.535***
area (μm^2)	***163458***

Regarding to the values written in table (5.2) it can be noticed that a working frequency of 1MHz is enough to ensure processing the input images at 60 fps. This particular description, also, ensures a very low power consumption.

Chapter 6

Conclusions and future works

The HSG algorithm has proved to be an excellent solution for pedestrian detection and other applications (2.20). In this thesis, therefore, an architecture that computes the histogram to send an Hardware Neural Network was developed. The work was organized by following two steps, the first required to check the correct computation, the second one was focused on the optimizations, in particular by describing parallel structures where it was possible. Regarding to the future improvements of this architecture, the first one could be applying the clock gating technique by exploiting the Synopsys© capabilities and another one could be the using of a proper Neural Network to obtain *DET and ROC* curves. The final graphical result is obtained after 128 iterations and it leads to have a final *feature vector* composed by 4608 elements (instead of 3780 elements of the HOG implementations). This overhead is compensated by the fact that on one side all results inside the proposed architecture are obtained by adopting an integer arithmetic and on the other side stride percentage is equal to 0%. In the end, merely looking to the graphical results, it can be noticed that algorithm produces histograms where wasteful informations are unwrapped (6.1):

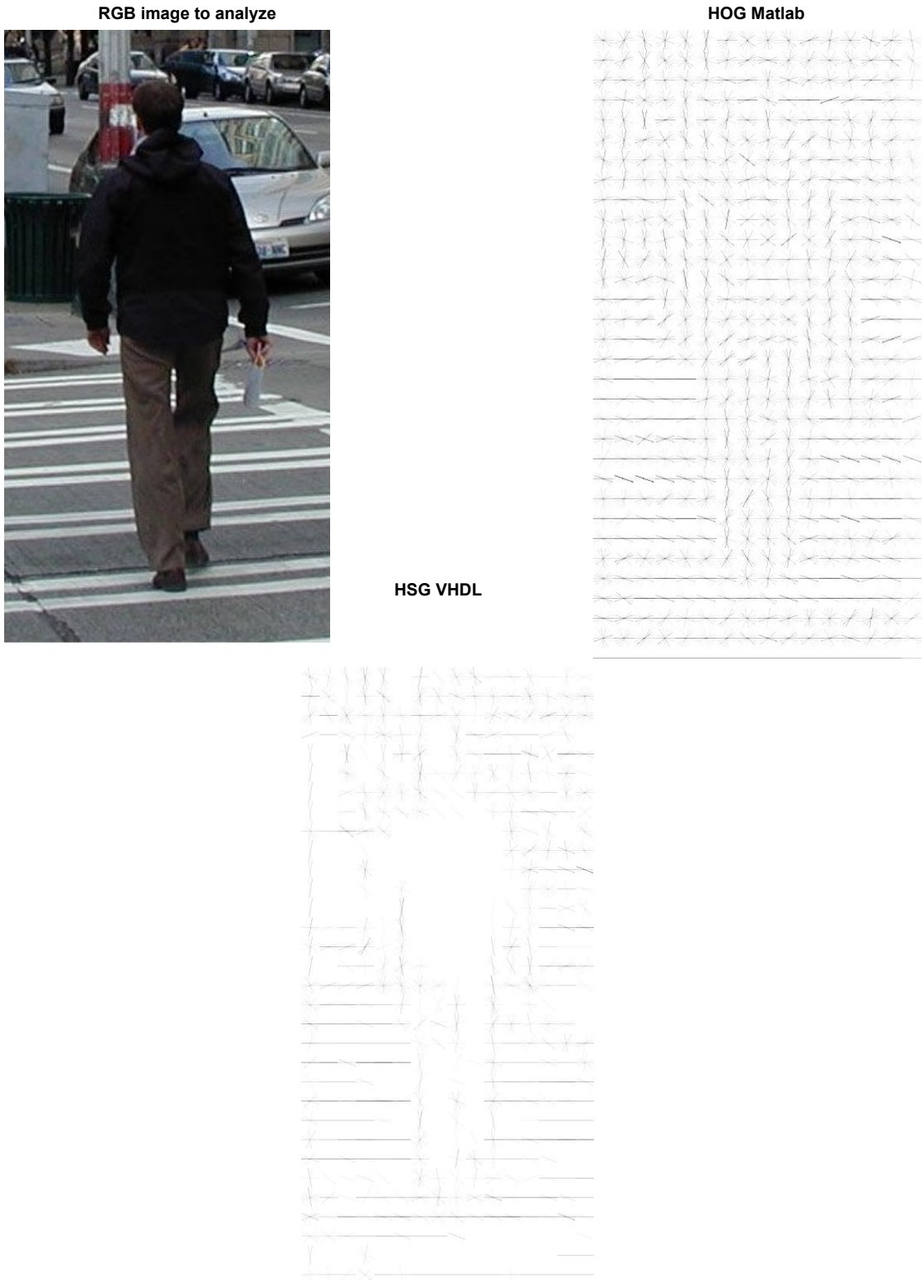


Figure 6.1: **HSG and HOG comparison final results**

Bibliography

[1] W. H. ORGANIZATION, *World Health Organization. Global status report on road safety.* Management of Noncommunicable Diseases, Disability, Violence and Injury Prevention (NVI), 2015.

[2] ERTRAC, *Automated Driving Roadmap*, 2015.

[3] M. Stanley, "Autonomous cars: The future is now." *Blue Papers*, 2015.

[4] W. D. Jones., "Building safer cars," *IEEE Spectrum*, 2002.

[5] D. Geronimo, M. Lopez, D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems." *PAMI*, vol. 32, 2010.

[6] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *PAMI*, vol. 34, 2012.

[7] R. Wang, "Edge detection with image pyramid," http://fourier.eng.hmc.edu/e161/lectures/canny/node2.html, 2013.

[8] "Support vector machine," https://en.wikipedia.org/wiki/Support_vector_machine.

[9] "Roc," https://en.wikipedia.org/wiki/Receiver_operating_characteristic, 2013.

[10] O. S. Software, "Drawing roc curve," https://docs.eyesopen.com/toolkits/cookbook/python/plotting/roc.html, 2018.

[11] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *CVPR*, 2009.

[12] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[13] "Haar wavelet," https://en.wikipedia.org/wiki/Haar_wavelet.

[14] A. Haar, "Zur theorie der orthogonalen funktionensysteme," *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, 1910.

[15] R. C. Guido, "A note on a practical relationship between filter coefficients and scaling and wavelet functions of discrete wavelet transforms," *Applied Mathematics Letters*, vol. 24, no. 7, pp. 1257–1259, 2011.

[16] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[17] D. Peleshko and K. Soroka, "Research of usage of haar-like features and adaboost algorithm in viola-jones method of object detection," in *Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2013 12th International Conference on the*. IEEE, 2013, pp. 284–286.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[19] "Histogram of oriented gradients," https://www.learnopencv.com/histogram-of-oriented-gradients/.

[20] "Image kernels, explained visually," http://setosa.io/ev/image-kernels/.

[21] M. Bilal, A. Khan, M. U. K. Khan, and C.-M. Kyung, "A low-complexity pedestrian detection framework for smart video surveillance systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2260–2273, 2017.

[22] H. Ren and Z.-N. Li, "Object detection using edge histogram of oriented gradient," in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4057–4061.

[23] J. Wu, "Efficient hik svm learning for image classification," *IEEE Transactions on Image Processing*, vol. 21, no. 10, pp. 4442–4453, 2012.

[24] J. Wu and J. M. Rehg, "Beyond the euclidean distance: Creating effective visual codebooks using the histogram intersection kernel," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 630–637.

[25] "Caltech pedestrian detection benchmark," http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/.

[26] P.-Y. Chen, C.-C. Huang, C.-Y. Lien, and Y.-H. Tsai, "An efficient hardware implementation of hog feature extraction for human detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 2, pp. 656–662, 2014.

[27] "Fifo working," http://electrosofts.com/verilog/fifo.html.

[28] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[29] "Netlist meaning," https://en.wikipedia.org/wiki/Netlist.

[30] "Sdf meaning," http://www.vlsiip.com/asic_dictionary/B/back_annotation.html.

[31] "How netlist is generated," http://www.eng.auburn.edu/~nelson/courses/elec5250_6250/slides/LogicSynthesis-Synopsys.pdf.