Parallel Image Gradient Extraction Core For FPGA-based Smart Cameras

Luca Maggiani, Cédric Bourrasset, François Berry, Jocelyn Sérot, Matteo Petracca, Paolo Pagano, Claudio Salvadori

Parallel Image Gradient Extraction Core For FPGA-based Smart Cameras

Luca Maggiani *‡, Cédric Bourrasset ‡, François Berry‡, Jocelyn Sérot*,
Matteo Petracca*, Paolo Pagano*, Claudio Salvadori*
* TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
‡ Institut Pascal, Universite Blaise Pascal, Clermont Ferrand, France

***Abstract*—One of the biggest efforts in designing pervasive Smart Camera Networks (SCNs) is the implementation of complex and computationally intensive computer vision algorithms on resource constrained embedded devices. For low-level processing FPGA devices are excellent candidates because they support massive and fine grain data parallelism with high data throughput. However, if FPGAs offers a way to meet the stringent constraints of real-time execution, their exploitation often require significant algorithmic reformulations.**

In this paper, we propose a reformulation of a kernel-based gradient computation module specially suited to FPGA implementations. This resulting algorithm operates on-the-fly, without the need of video buffers and delivers a constant throughput. It has been tested and used as the first stage of an application performing extraction of Histograms of Oriented Gradients (HOG). Evaluation shows that its performance and low memory requirement perfectly matches low cost and memory constrained embedded devices.

I. Introduction

With the rapidly increasing interest in decentralized approaches for Smart Camera Networks (SCN), *embedded computer vision* has become a research field *per se*. The deployment of complex and smart applications on SCNs now crucially depends on the the possibility to perform complex and time-consuming vision algorithms directly at the node level. This places stringent constraints on the processing elements embarked on these nodes. This is specially true for the so-called low-level processing stages, operating on the raw pixels coming from the sensor, for which the increasing sensor resolutions lead to data throughputs which cannot be sustained by embedded general purpose CPUs.

Because they naturally support massive and fine grain data parallelism, FPGAs have often been advocated as a solution to the aforementioned issue. However, in most cases, this is only true if the algorithm to be implemented can be formulated to operate on *sequential streams of pixels*, without the need to memorize and iterate on intermediate results. This reformulation, which essentially amounts to convert an *imperative, stateful* model to a *dataflow, stateless* one is often not trivial or even feasible in case of non-Turing-complete algorithms. Moreover, the efficiency of the final implementation directly depends on it. Yet, it is seldom explicitly described, if not mentioned, in papers dealing with embedded vision.

Let's take for example a classical operation in low-level vision : the computation of the gradient. This operation is used in many higher-level algorithms such as the Hough transform, the Canny edge detector or the Histogram of Oriented Gradient (HOG). It relies on regular, pixel-wise operations but some of them – the square root and the arc-tangent – are non linear [**?**] and thus do not map easily on hardware computing devices such as FPGAs. To circumvent this problem, two solutions have typically been proposed : using pre-computed look-up tables as in [**?**], [**?**] or relying on run-time iterative algorithms such as CORDIC or derivatives [**?**]. The former approach requires large amount of memory, which can quickly be a problem with tightly space and/or cost-constrained devices, especially when the required precision increases. The second approach has a strong impact on performances since it is no longer possible, in the general case, to maintain a 1-to-1 ratio between the frequency of the input and output pixels respectively. In certain cases, by taking into account the algorithmic context – i.e. the subsequent utilisation of the gradient information computed by the operation –, it is however possible to overcome the limitations of the above mentioned solutions. This is the case, in particular, for the gradient used in the HOG application. The HOG algorithm [**?**] relies on two steps : first the computation of the image gradient followed by a local spatial aggregation of the histogram of the computed gradients. This algorithm is the basis of several well-known methods for detecting objects like pedestrians or vehicles in image sequences [**?**], [**?**], [**?**]. In this context, the resulting histograms are then normalised and compared to reference models by a SVM linear classifier in order to make decision about the presence/non-presence of the target objects.

In this paper, we describe an optimized implementation of the gradient extraction operation which can be used as the first step of the HOG algorithm. By taking into account the specific needs of the subsequent steps and reformulating the extraction in terms of purely dataflow basic operators, we have obtained a highly parallel, fully pipelined implementation, producing a valid result at every clock cycle and with a memory footprint considerably smaller compared to other state of the art solutions. The accuracy of the produced results also meets the precision requirements of the HOG algorithm. The rest of this paper is organized as follows. Sec. II is a brief overview of the existing solutions proposed for computing the gradient, especially for the HOG algorithm. Our solution, named *HOG-Dot* is described in Sec. III. It is evaluated, in terms of accuracy, in Sec. IV. The corresponding hardware

implementation is exposed in Sec. V and the performances of the resulting core are assessed in Sec. VI.

II. Related work

The state of the art of the gradient implementation is mainly related to its use in the HOG algorithm [**?**]. Since spatial gradients techniques are based on magnitude $\| \nabla I \|$ and angle orientation θ, gradient can be expressed in polar coordinates by:

$$\| \nabla I \| = \sqrt{{G_x}^2 + {G_y}^2} \quad \theta = \arctan \frac{G_y}{G_x} \tag{1}$$

where G_x and G_y are respectively the gradient in x and y direction defined as:

$$G_x = I * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \text{ and } G_y = I * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T \tag{2}$$

where I is the input image. These operations require strong computing capabilities for real-time processing. In particular, the $arctan$ evaluation in a hardware implementation either requires many resources or produces non efficient results. In [**?**] a low resolution VLSI gradient module has been proposed. The gradient approximation is carried out by deploying a steering filter architecture within a custom CMOS image sensor. More recently, a fixed point implementation has been proposed by Kadota et al. [**?**]. The authors designed a memory-based core that extracts the gradient with precomputed Look-Up table (LUT) entries. The memory contains all the possible entries of $\left(G_x, G_y\right)$. Therefore the memory footprint would grow as $G_x \times G_y$ sizes increase, i.e. for luminance range extensions or to achieve better gradient resolution. Although this system reaches real time performance, it requires a huge memory footprint ($n \times 2^{2n+1}$ bits where n is the G_x representation) and a square root based on multi-cycles operations per pixel. Mizuno et al. [**?**] has improved the Kadota et al. proposal by using a CORDIC method [**?**] for the orientation evaluation. While the CORDIC method provides an acceptable precision for the orientation, it requires a variable number of iterations which depends on the initialisation value of the algorithm and the input operand.

The most used technique for the gradient extraction has been introduced by Bauer et al. in [**?**] and then reproduced in more recent implementations [**?**], [**?**]. Bauer et al. propose an orientation binning scheme without computing the orientation angle explicitly. The x and y derivatives directly index the histogram using a LUT, while the magnitude is extracted with an hardware square root module. Even though this technique improves the performance with respect to CORDIC-based solutions, the square root module increments the processing latency and limits the maximum operating frequency as well.

III. HOG-Dot algorithm

One of the main issue in developing the HOG pipeline for FPGA-based SC is the polar coordinates computations through non linear floating-point operations. Even though floating point arithmetic could be exploited within FPGA, as shown in [**?**], its implementation require a non-negligible amount of hardware resources and furthermore it will decrease the output throughput. Thus, we propose a highly parallel architecture capable to approximate the *square-root* and the *arc-tangent*. It is based on a set of linear operations that can be easily implemented within a pipelined hardware circuitry. The proposed methods considers a discrete gradient orientation space divided into N uniformly spaced samples, which are denoted by $\hat{i_k}$ with $k = 0, ..., N-1$. In Fig. 1 the considered direction versors are shown over the semi-circumference space, where θ_k represents the angle orientation of the k^{th} gradient component.

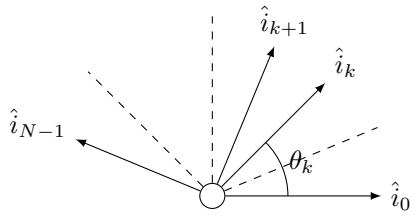

Fig. 1. The versors beam used to compute the dot product.

The versor $\hat{i_k}$ can be expressed by its components as follows:

$$\hat{i_k} = \left\{ \hat{x} \cos\theta_k + \hat{y} \sin\theta_k \middle| k \in 0, \cdots, N-1 \right\} \tag{3}$$

For each $\hat{i_k}$ versor, we express the overlaying gradient component as the scalar projection of ∇I over $\hat{i_k}$. In Eq. 4 the scalar product is given.

$$\frac{\partial I}{\partial \hat{i_k}} = \nabla I \cdot \hat{i_k} \tag{4}$$

Then by replacing Eq. 3 in Eq. 4, we obtain:

$$\frac{\partial I}{\partial \hat{i_k}} = \cos\theta_k \frac{\partial I}{\partial x} + \sin\theta_k \frac{\partial I}{\partial y} \tag{5}$$

Finally, by recalling Eq. 2, in Eq. 6 the spatial gradient component over a generic $\hat{i_k}$ versor is shown.

$$\begin{aligned} \frac{\partial I}{\partial \hat{i_k}}(x,y) = \cos\theta_k \left[I(x+1,y) - I(x-1,y) \right] \\ + \sin\theta_k \left[I(x,y+1) - I(x,y-1) \right] \end{aligned} \tag{6}$$

As in Eq. 6, the k^{th} projection of the spatial gradient ∇I can be straightforward computed by applying linear arithmetic operations to the image pixel. In particular in Eq. 7 the complete matrix expression is shown. Once the number of the orientation samples N has been defined, the co-sinusoidal values are then a-priori fixed for each θ_k thus resulting a constant coefficients matrix.

In Eq. 7 a generic k^{th} component can be computed as convolution product between the kernel matrix and the image pixels. By applying the same approach for each $\hat{i_k}$ within the N orientation samples, all the gradient projections are computed. In Fig. 2, $N = 8$ is assumed. The eight co-sinusoidal convolutions are carried out over the same 3×3 mask pixels, thus computing the gradient projections. The spatial gradient is then evaluated by extracting the greater magnitude projection among the available set. The resulting projection represents the closest gradient approximation among the available N

$$\frac{\partial I}{\partial \hat{i}_k}(x,y) = \begin{bmatrix} 0 & -\sin\theta_k & 0 \\ -\cos\theta_k & 0 & +\cos\theta_k \\ 0 & +\sin\theta_k & 0 \end{bmatrix} \cdot \begin{bmatrix} I(x-1,y-1) & I(x,y-1) & I(x+1,y-1) \\ I(x-1,y) & I(x,y) & I(x+1,y) \\ I(x-1,y+1) & I(x,y+1) & I(x+1,y+1) \end{bmatrix} \tag{7}$$

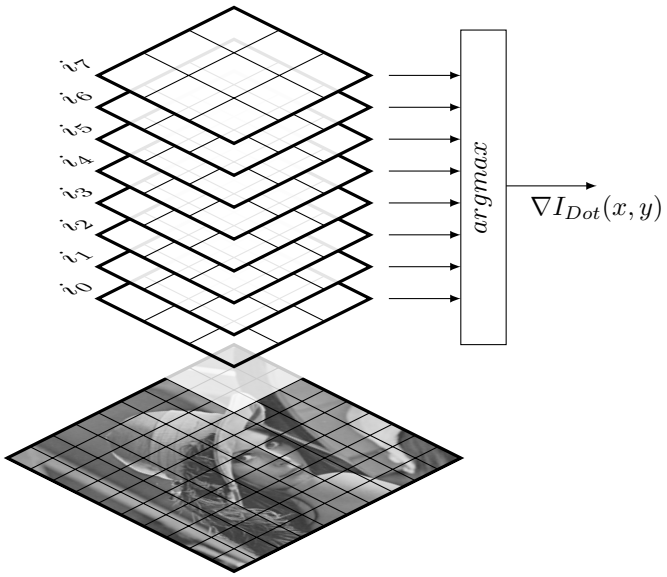

Fig. 2. The proposed HOG-Dot parallel implementation.

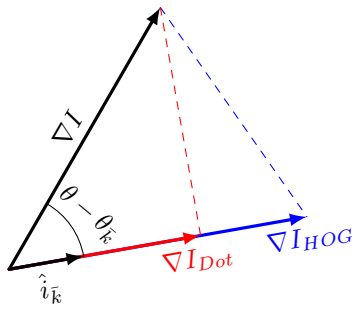

Fig. 3. ∇I, ∇I_{HOG} and ∇I_{Dot} comparisons.

orientation samples. Once the maximum magnitude has been computed, the $argmax$ operator extracts the relative $\bar{k}^{th}$ orientation as follows:

$$\bar{k} = argmax\Big\{\frac{\partial I}{\partial \hat{i}_{\bar{k}}}(x,y) \quad \Big| \quad k \in 0,\cdots,N-1\Big\} \tag{8}$$

Finally, in Eq. 9 the resulting gradient approximation is shown.

$$\nabla I_{Dot}(x,y) = \big(\frac{\partial I}{\partial \hat{i}_{\bar{k}}}(x,y) \quad , \quad \theta_{\bar{k}}\big) \tag{9}$$

The original HOG implementation defines instead the gradient $\nabla I_{HOG}(x,y)$ as:

$$\nabla I_{HOG}(x,y) = \big(\parallel \nabla I(x,y) \parallel \quad , \quad \theta_{\bar{k}}\big) \tag{10}$$

By comparing Eq. 9 and Eq. 10, the resulting vectors have the same orientation direction $\hat{i}_k$ thus both belong to the same histogram bin. With respect to the magnitude differences, in Fig. 3 both approximations are reported along with the mathematical formulation. This expression, defined as ∇I, represents the spatial gradient definition as in Eq. 1 which results without the HOG orientation binning. In particular, it is clearly shown that ∇I_{Dot} always represents a round down approximation of the ∇I_{HOG}. Since HOG-Dot involves a dot-product operation, the ∇I_{Dot} approximation can be also expressed as follows:

$$\parallel \nabla I_{Dot}(x,y) \parallel = \parallel \nabla I(x,y) \parallel \cos(\theta - \theta_{\bar{k}}) \tag{11}$$

All in all, the HOG-Dot technique allows gradient computation by deploying only linear operations rather than square root and arc-tangent operator. By controlling the number of the considered orientations N, the algorithm accuracy can be estimated and then predicted. The higher N means a finer granularity of the gradient projections but requires more computing iterations. Nevertheless the increased iterative operations, our method is suitable for a parallel exploitation thus achieving the highest performance.

IV. HOG-Dot accuracy evaluation

To evaluate the approximation error a mathematical expression has been derived, in terms of gradient angle and magnitude. As previously introduced, the angle approximation depends on the number of orientation samples N. Given a fixed N, the orientation accuracy is then equal to $2\pi/N$ radiant. As far as the magnitude approximation concerns, the Error Vector Magnitude (EVM) metric has been used. The EVM is usually deployed to quantify the difference between transmitted and received constellation points and can be expressed as:

$$EVM \quad = \quad \sqrt{\frac{P_{error}}{P_{ref}}} \tag{12}$$

where the power of the error vector is expressed as P_{error} while the ideal one as P_{ref}. In this context, the EVM can be rewritten as:

$$EVM_{Dot} = \sqrt{\frac{\parallel e \parallel^2}{\parallel \nabla I_{HOG}(x,y) \parallel^2}} = \frac{\parallel e \parallel}{\parallel \nabla I_{HOG}(x,y) \parallel} \tag{13}$$

where e is $\nabla I_{HOG}(x,y) - \nabla I_{Dot}(x,y)$, the difference vector between the HOG and our version. As in Eq. 11 the difference vector between $\nabla I_{HOG}(x,y)$ and $\nabla I_{Dot}(x,y)$ is straightforward evaluated as follows:

$$\begin{aligned} \parallel e \parallel = & \quad \parallel \nabla I_{HOG}(x,y) - \nabla I_{Dot}(x,y) \parallel \\ = & \quad \parallel \nabla I(x,y) \parallel \Big[1 - cos(\theta - \theta_{\bar{k}})\Big] \end{aligned} \tag{14}$$

Thus, the EVM_{Dot} is a deterministic function of $\theta - \theta_{\bar{k}}$, and, consequently of N ($\theta_{\bar{k}} = \bar{k}\pi/N$). The EVM_{Dot} is then

described by:

$$EVM_{Dot} \quad = \quad 1 - cos\left(\theta - \frac{\bar{k}\pi}{N}\right) \tag{15}$$

In Fig. 4 the EVM_{Dot} is then shown. It is expressed within a single bin span as function of $\theta - \theta_k$. Since the angle resolution directly depends on N, two EVM curves are reported for N equal to 8 and 16 respectively. The error goes to zero whether θ is equal to θ_k, while it achieves the maximum value when ∇I_{Dot} lies in the middle between θ_{k-1} and θ_{k+1} instead. Nevertheless, with $N = 8$ the maximum error is bounded to 2%. To complete the comparison, in Fig. 5 the

Fig. 4. The EVM_{HOG_Dot} evaluation as function of $\theta - \theta_{\bar{k}}$.

maximum and average errors, defined as $max(EVM_{Dot})$ and $avg(EVM_{Dot})$, are reported as function of N. By increasing the orientation samples N, the difference between θ and $\theta_{\bar{k}}$ monotonically decreases, thus the maximum EVM decreases as well. Given all values on its period, the average EVM is evaluated as follows:

$$avg(EVM_{Dot}) = \frac{2}{T}\int_0^{T/2} EVM_{Dot}\ \mathrm{d}\theta \tag{16}$$

$$= 1 - \frac{2N}{\pi} sin(\frac{\pi}{2N})$$

where T is equivalent to the EVM_{Dot} period $\frac{\pi}{N}$. The average value is clearly bounded by Eq. 15 and decreases when N increases. Although we exploit the gradient extraction with a linear kernel based method, our results are comparable with those obtained by the HOG technique. As shown in Fig. 5, by fixing N equal to 8, the resulting average EVM is close to 0.5%. Whether an improved accuracy is requested, the sampling factor N can be further increased to match the output specifications.

Fig. 5. Maximum and average EVM_{Dot} as function of N.

V. Hardware Implementation

In Section III the proposed method has been first formally introduced. In this Section a FPGA hardware implementation will be shown. In particular, the proposed hardware design has been developed as the gradient extraction circuit for the HOG pipeline. Therefore the number of orientation samples N is then equivalent to the number of the HOG histogram bins, usually equal to 8.

The hardware design is obtained following Eq. 7. The matrix convolution between the kernel coefficients and the 3×3 mask pixel is computed by leveraging on pipelined hardware module. Since these coefficients are constant for a given angle θ_k, the k^{th} gradient component can be evaluated by deploying two hardware multipliers and three adder/subtractor modules. In Fig. 6 the hardware circuitry for a single gradient projection is shown. More in detail, the input pixel is represented with 8bit while each coefficient value has been represented with 9bit two's complement. Due to the internal pipeline structure, the module shown in Fig. 6 is capable of processing new input data every clock cycle to achieve the maximum throughput. In

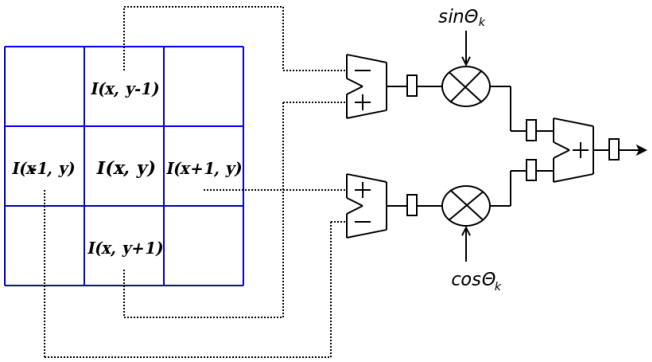

Fig. 6. The hardware implementation for the i_k gradient component.

order to extend the gradient computation to all the available projections, N convolution modules are instantiated in parallel. As previously suggested by Fig. 2, the resulting hardware structure generates N parallel gradient projections. Each clock cycle, those resulting projections are compared each other to extract the maximum magnitude value. In Fig. 7 the hardware $argmax$ implementation is shown. This is implemented as $\log_2(N)$ comparative stages within a pipelined architecture.

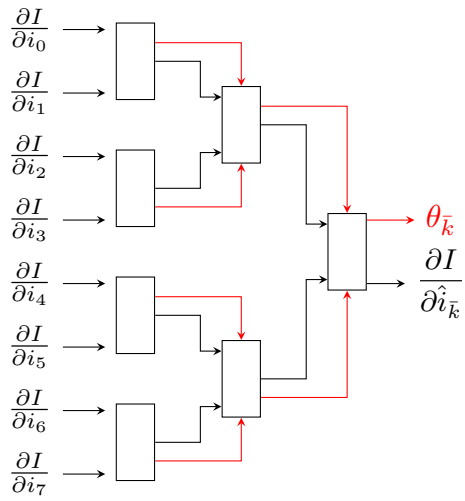

Fig. 7. Pipelined $argmax$ extraction ($N = 8$).

With the assumption of $N = 8$, the first stage is composed by four parallel comparators, which are fed by the computed gradient projections. Each comparator propagates to the next stage the temporary maximum projection and its orientation. In the presented example, the comparison ends at the third stage, where the closest gradient approximation is generated. The resulting couple $\frac{\partial I}{\partial \hat{i}_{\bar{k}}}$ and $\theta_{\bar{k}}$ represents the closest spatial gradient approximation retrieved from the HOG-Dot method, as in Eq. 9.

VI. Performance Evaluation

In this Section the proposed gradient extraction module is evaluated in terms of memory footprint, data throughput, data latency and output accuracy. We adopted the Terasic DE0-nano development board which embeds an Altera Cyclone

IV FPGA with 22k Logic Elements (LE), 600 kbits on-board memory and 144 9x9 bits multiplier modules. Considering a VGA resolution and N equal to 8, the proposed hardware module occupies only 5.4% of the Logic Elements (LEs), 1.2 KByte of the on-chip RAM (0.2%) and uses 16 of 144 available multiplier module. These results are depending on the number of gradient components N and to the maximum image width size W. Higher N would increase the number of the used multiplier as $2 \cdot N$ while an increased image size will increase only the memory footprint due to neighborhood extraction. In Tab. I a comparison with the state of the art is shown. In particular, due to the limited implementation details in the state of the art and the different FPGA technologies involved, numerical LEs comparisons are impractical. We rather compare the on-chip memory occupancy footprint and the used DSP modules among the considered solutions.

TABLE I
HARDWARE RESOURCE UTILIZATION FOR VGA RESOLUTION.

	Platform	RAM	DSPs
Kadota [?]	Altera Stratix II	N/A	12
Mizuno [?]	Altera Cyclone IV	64 KB	68
Bauer [?]	Xilinx Spartan-3	225 KB	18
Negi [?]	Xilinx Virtex-5	162 KB	0
HOG-Dot	Altera Cyclone IV	1.2 KB	16

Although the proposed work uses a higher amount DSP modules with respect to Kadota et al. [?] and Negi et al. [?], HOG-Dot memory footprint is significantly lower than others. Moreover, with respect to the implementations proposed by Bauer et al. [?] and Mizuno et al. [?], our method reduces the DSP usage. This results of the HOG-Dot pipelined architecture, which optimizes the hardware resource occupancy. In Table II the processing latency and the maximum operating frequency are shown with respect to the other implementations. As far as throughput concerns, our implementation achieves the same results as the Negi et al. proposal, while is deployed within a low-end FPGA device. With respect to Bauer et al., the resulting processing latency is increased by removing frame buffering and external memory accesses. The maximum operating frequency is 78MHz, which corresponds to the higher throughput achievable. Hence, the theoretical image frequency of 250 fps can be reached at VGA resolution. Given the limited processing latency, our module is then suitable for real-time processing applications.

TABLE II
PROCESSING LATENCY COMPARISONS.

	Latency (clock cycles)	Operating frequency (MHz)
Kadota et al. [?]	24	127.49
Mizuno et al. [?]	N/A	40
Bauer et al. [?]	3262	63
Negi et al. [?]	2	44.85
HOG-Dot	3	78

A. *HOG performance comparison*

In this Section, our HOG-Dot method is then compared with the state of the art HOG implementation. In order to compare the results, an entire hardware HOG pipeline has been deployed following the module presented in [?]. The HOG-Dot hardware module is then followed by the histogram circuit which provides to the classifier an ordered flow of oriented gradient over 8×8 pixel cells. The original HOG and our modified approach have been trained both with a classification SVM provided by SVMLight [?]. The INRIA pedestrian [?] dataset has been used to train and test both systems. The reference HOG implementation has been provided by the OpenCV framework, which implements a double precision floating point arithmetic with arc-tangent and square root operations. The comparison results are summarized with the Receiver Operator Characteristics (ROC) metric and shown in Fig. 8. This curve shows in the x-axis the relationship between the False Positive Rate (FPR) and the True Positive Rate (TPR) on the y-axis. The closer to the upper left the ROC goes, the higher performance are achieved. The plots are made by changing the SVM decision threshold within ± 5 range as proposed by [?]. As a result, the proposed fixed

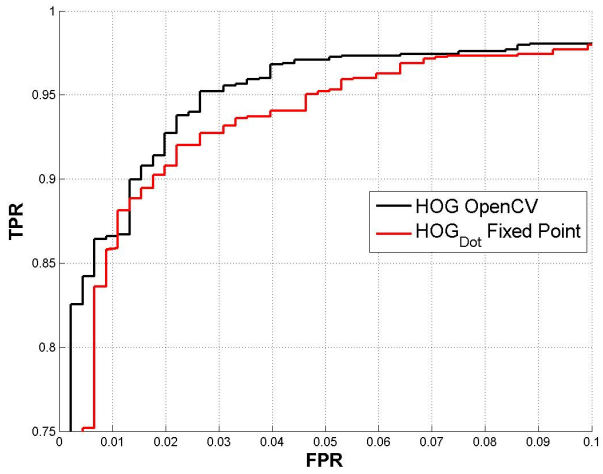

Fig. 8. ROC curves with INRIA dataset

point HOG-Dot algorithm shows TPR up to 91% with FPR at 2%. These performance have been obtained with a L2 histogram normalization as in the OpenCV implementation. The original OpenCV instead results around 93% TPR with 2% FPR. The difference between the reference and our proposed method is clearly due to the approximation error shown in Fig. 4. Although our method only involves fixed-point linear operation, the detection performance shows comparable results with the floating point, non linear reference implementation. As far as the accuracy evaluation is concerned, the proposed approximated algorithm performs up to 2% of error on the gradient magnitude.

VII. CONCLUSIONS

In this paper the parallel gradient extraction HOG-Dot algorithm is presented. This algorithm is suitable for real-time

image processing in FPGA-based low-cost Smart Cameras. The fixed point and linear HOG-Dot implementation has been evaluated in a HOG-SVM pedestrian detection system. It shows comparable detection performance with respect to the floating point, non linear reference implementation. Moreover, the hardware implementation shows benefits in terms of memory footprint and hardware resources occupancy with respect to state of the art FPGA solutions. The final implementation shows real-time processing performance, enabling a wide range of applications in the smart camera scenario.

ACKNOWLEDGMENT

This work has been sponsored by the French government research programme ”Investissements d’avenir” through the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01).