Robust feature extraction algorithm suitable for real-time embedded applications

Abiel Aguilar-González, Miguel Arias-Estrada, François Berry

Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Abiel Aguilar-González · Miguel Arias-Estrada · François Berry

Robust feature extraction algorithm suitable for real-time embedded applications

Received: date / Revised: date

Abstract Smart cameras integrate processing close to the image sensor, so they can deliver high-level information to a host computer or high-level decision process. One of the most common processing is the visual features extraction since many vision-based use-cases are based on such algorithm. Unfortunately, in most of cases, features detection algorithms are not robust or do not reach real-time processing. Based on these limitations, a feature detection algorithm that is robust enough to deliver robust features under any type of indoor / outdoor scenarios is proposed. This was achieved by applying a non-textured corner filter combined to a subpixel refinement. Furthermore, an FPGA architecture is proposed. This architecture allows compact system design, real-time processing for Full HD images (it can process up to 44 frames/91.238.400 pixels per second for Full HD images), and high efficiency for smart camera implementations (similar hardware resources than previous formulations without subpixel refinement and without non-textured corner filter). For accuracy/robustness, experimental results for several real world scenes are encouraging and show the feasibility of our algorithmic approach.

Keywords · Robust feature extraction · Smart camera · FPGA · 3D reconstruction.

Abiel Aguilar-González (✉)
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), Luis Enrique Erro # 1, 72840 Tonanzintla, Puebla, Mexico E-mail: abiel@inaoep.mx

Miguel Arias-Estrada
Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), Luis Enrique Erro # 1, 72840 Tonanzintla, Puebla, Mexico E-mail: ariasmo@inaoep.mx

François Berry
Université Clermont Auvergne, Institut Pascal, 4 Avenue Blaise Pascal, 63178 Aubière, Clermont-Ferrand,France E-mail: francois.berry@uca.fr

1 Introduction

Smart cameras are image/video acquisition devices with self-contained image processing algorithms that simplify the formulation of a particular application. For instance, algorithms for smart video surveillance could detect and track pedestrians [23], but for a robotic application, algorithms could be edge and feature detection [2]. In recent years, advances in embedded vision systems such as progress in microprocessor power and FPGA technology allowed the creation of compact smart cameras with low cost and, this increased the smart camera applications performance, as shown in [11, 6, 7, 8]. In current embedded vision applications, smart cameras represent a promising on-board solution under different application domains: motion detection [25], object detection/tracking [32, 31], inspection and surveillance [16], human behavior recognition [20], etc. In any case, flexibility of application domain relies on the large variety of image processing algorithms that can be implemented inside the camera. Algorithms highly used by smart cameras are feature extraction algorithms since extracted features represent medium level abstractions of the images and this can be used as rough reference for scene understanding. There are two types of features that can be extracted from an image. Global features describe the image as a whole; they can be interpreted as a particular property of the image. On the other hand, local features aim to detect key points/feature points within the image. In smart cameras context, there is a tendency for local features (edges, blobs, corners), as the only visual features extracted by the algorithms inside the camera. Several distributed vision systems like object tracking [44, 29], virtual reality [34] and human 3D pose reconstruction [51, 43] have applied smart camera networks in which every node provides local image features. In these configurations, nodes cooperation deliver high level information to a host computer/robot.

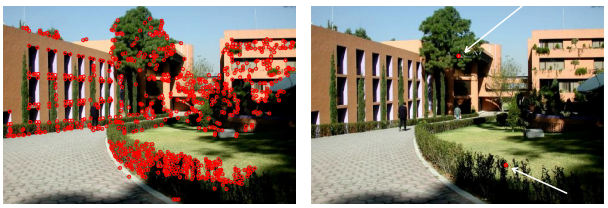
1.1 Visual features for smart cameras

Local feature detection is an image processing operation that aims to deliver medium-level abstractions from an image, and often it is used as initial step of several computer vision algorithms. In previous work, several local visual feature detection algorithms were proposed: algorithms such as Canny or Sobel [10], deliver image edges that often are used in applications like object detection [13], image labeling[50], image segmentation [27], stereo vision [41], etc. Other algorithms are corner detection like Shi & Tomasi [36], Harris & Stephens [21], FAST [35], and they are the cornerstone of several computer vision applications such as, 3D reconstruction [39], camera calibration [49], Structure from Motion (SfM) [40], Simultaneous Localization and Mapping (SLAM), etc. Nowadays desktop computers can process most of the corner detection algorithms in real-time. Unfortunately, in some cases (mobile applications, autonomous robotics and compact smart vision systems) such approaches could be low efficient since they require relatively high computational resources, then power consumption and sizes can be not compatible with an embedded system. One solution to this problem is the use of dedicated hardware as Field Programmable Gate Arrays (FPGAs). This is because FPGAs are devices with low power consumption and its size is small (suitable to embedded/mobile applications). In addition, FPGAs are structured as a customizable circuit where image processing operations can be performed in parallel using a data-flow formalism. For corner detection , in previous work several FPGA-based smart cameras have integrated corner detection algorithms [6, 2, 7] inside the camera fabric, as result, these cameras can simplify the formulation of applications like 3D reconstruction, SfM, object tracking and camera calibration [39, 49, 40]. This is because in these algorithms the first step is for visual feature extraction, considering that the sensor (smart camera) deliver images and feature extraction simultaneously, then, the problem become partially solved. i.e., in all cases the first step of the algorithmic formulation become solved.

1.2 Performance of Corner detection algorithms

Previous corner detection algorithms such as Shi-Tomasi [36] or Harris & Stephens [21] provide good performance for datasets and/or for geometrical scenes (building images, text images and calibration patterns). There are several computer vision applications that used these corner detection algorithms successfully [39, 49, 40], and several smart cameras included them in their self- contained algorithms [6, 2]. Unfortunately, in several applications the corner detection algorithms are not compatible with high textured regions [22] or can not perform real-time processing on full HD images [24, 33, 4]. We can mention the three most important limitations affecting the current corner detection algorithms:

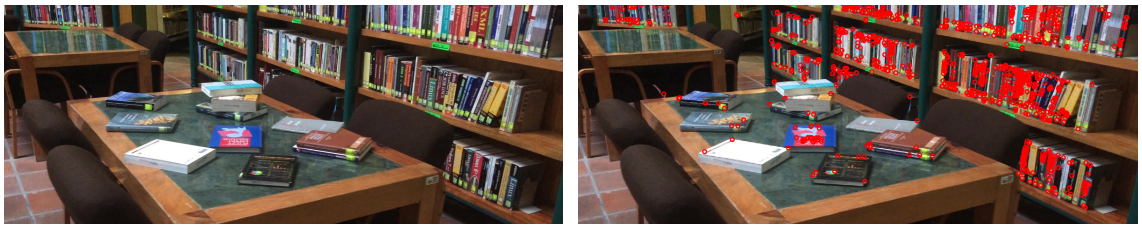
1 *Low performance under complex textured regions.* One limitation occurs when the input images have complex textured regions such as tree foliage or flowerbed (**Fig.** 1a). In these regions, most algorithms detect features that have low temporal stability: i.e., its illumination or orientation changes in time and it is difficult to track. This problem is well documented, for example, there are several works that study the SLAM systems scope/performance under different feature extraction algorithms [18, 30, 47]. In several applications (3D reconstruction, SfM, SLAM), one solution frequently used is to retain only the features with high dominance since it is assumed that the retained features should have high stability and should be easy to track. In practice the stability of high dominant features is not necessarily consistent since retained features still can be located within complex textured regions, as shown in **Fig.** 1b. In addition, retaining features with high dominance implies the use of high threshold values but in several cases these values retain low number of features, and in SfM/SLAM applications these few features often do not provide sufficient information for the camera pose estimation, more details about this problem can found at [17] .



(a) Shi & Tomasi algorithm, threshold = 0.1. **(b)** Shi & Tomasi algorithm, threshold = 0.8.

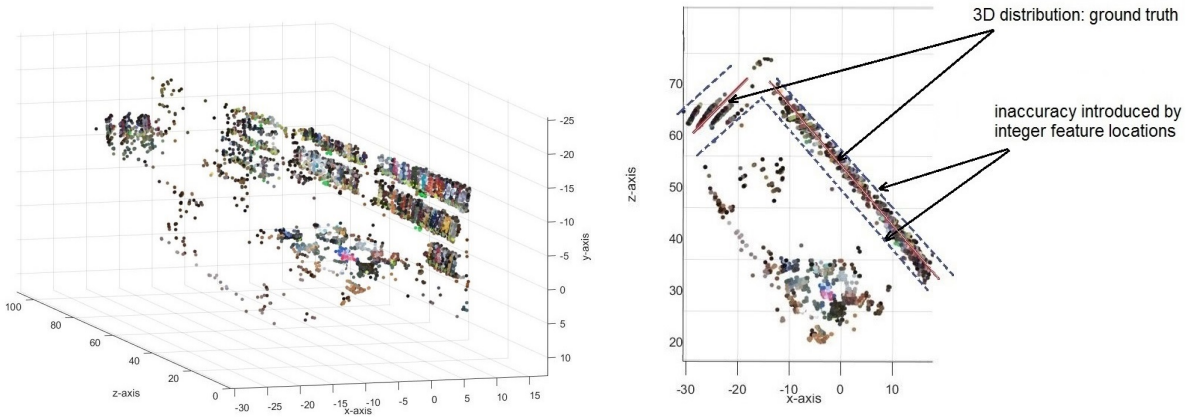
Fig. 1: Shi & Tomasi algorithm: performance for complex textured regions. (a) Low threshold values detect features that are difficult to track. (b) High threshold values deliver few features that often are not compatible with SfM and SLAM applications. Most previous algorithms: Harris & Stephens, FAST, etc., deliver similar results.

2 *Location accuracy is low.* Most of current feature extraction algorithms determine if a candidate pixel p is a corner or not, then, if the pixel p is a corner, the location of the pixel p fits with the location of the corner. In practice, these locations introduce an imprecision since the real position of the corner points can be spatial positions between two or more pixels (subpixel location). In the case of camera calibration and 3D reconstruction, these imprecisions have high impact in the global performance [37], as shown in **Fig.** 2. One solution is to add subpixel refinements as a post-processing step as shown in [48]. However, this increases the computational requirements and processing time.



(a) Input scene: interior of the INAOE library.

(b) Shi & Tomasi algorithm, threshold = 0.2



(c) 3D reconstruction: front view.

(d) 3D reconstruction: top view.

Fig. 2: 3D reconstruction using the Shi & Tomasi algorithm [36]. The extracted features (b) are tracked across different viewpoints from the same scene. (c) Then, we compute the corresponding 3D reconstruction. (d) In this case, feature locations without subpixel refinement introduce an imprecision in the 3D reconstruction.

3 *Low performance for embedded applications.* Nowadays computers can process several corner detection algorithms in real-time. Unfortunately, in embedded applications such as, mobile applications, autonomous robotics or compact smart vision systems, the use of computers is difficult due to their high power consumption and size. The use of FPGA technology is an alternative, but the architectural design requires a good background in HDL (Hardware description language) and digital design. Furthermore, most accurate algorithms like Shi & Tomasi/Harris & Stephens, have relatively complex mathematical formulation (quotients, roots, etc.), that require high hardware resources [24, 33, 4], hardware requirements are $\times 4$ more than corner detection algorithms with straightforward FPGA implementation. On the other hand, corner detection algorithms with straightforward FPGA implementation like FAST [35] have low hardware demand but accuracy and robustness is low.

2 Related work

1. *Low performance under complex texture regions.* Recent works in [22] proposed a FAST corner detector modification that defines corners as similarity in terms of intensity, continuity and orientation in three different circular areas. Then, by using the area similarities, a cascading tree decision process delivers robust features under complex texture regions. Unfortunately, the proposed modification has high impact in the global performance: processing speed is near double that the original FAST-N algorithm. In [46] it was introduced a method for selecting a subset of features that are highly useful for localization and mapping in SLAM/SfM applications. The algorithm is formulated with temporal observability indexes and efficient computation strategies for the observability indexes that are based on incremental Singular Value Decomposition (SVD). The proposed method improves the localization and data association but computational requirements are increased. The algorithm reaches real-time constraints only with sparse point clouds/3D reconstructions. In [15] a Spatial-Temporal Monitor (STM) identifies "good features to track" by monitoring their spatiotemporal appearances without any assumptions about motion or geometry. The STM can be used with any spatial visual descriptor, like SIFT or SURF and, in particular HOM, increasing the accuracy and robustness for SFM and 3D reconstruction applications. However, the algorithm formulation is exhaustive and real-time performance is limited to less than 10 corners per frame.

2. *Subpixel refinement.* In the last decade several works introduced post-processing steps that increase the accuracy of the extracted corners. In [19]the original Harris & Stephens algorithm is improved by introducing sub-pixel locations based on the Gauss surface method on the extracted corner locations. The detected corners were applied on image tracking applications in order to prove the higher precision than the original Harris & Stephens algorithm. In [38] a corner detection algorithm, which includes subpixel locations, was proposed. This algorithm detects corners as the intersection of two or more edges, and subpixel location is found by using a small neighborhood centered on the estimated corner positions, the corner orientation and the dihedral angle of the corner. Experimental results show an average error of 0.36 pixels. In [48] a location method is proposed to improve the precision of the corners extracted by the original Harris & Stephens algorithm. First, a least squares that fits a parabolic function to the image grayscale surface is employed to determine a weight. Subpixel locations are obtained by calculating maximum of the fitting surface. Experimental results show accuracy of 0.15 pixels. In all cases [19, 38, 48], post-processing steps deliver subpixel accuracy suitable to applications like camera calibration or 3D reconstruction. Nevertheless, a more efficient solution could consist in processing the subpixel refinement within the corner detection step. In this way, an FPGA architecture can benefit since data processing is reduced to a single step and there is no need to transfer intermediate results to a memory. Furthermore, subpixel refinement and the corner detection could be computed in parallel.

3. *Embedded applications.* Recent works have studied the hardware implementation of the original Harris & Stephens algorithm [5, 4, 12], in all cases, the FPGA architectures were focused in an efficient hardware resources utilization. In [4], an FPGA implementation based on sliding processing window for Harris corner algorithm is presented. The purpose of the sliding window is to avoid storing intermediate results of processing stages in the external FPGA memory or to avoid the use of large line buffers typically implemented with BRAM blocks. Therefore, the entire processing pipeline benefits from data locality. In [12], the "repetitive feature" extraction procedures were exploited in order to develop a full-parallel FPGA architecture. There is other works that have focused on the FAST-N formulation where direct hardware parallelization implies low hardware resources demand, compact system design and real-time processing with low-grade FPGAs [9]. Those benefits have been used in applications such as analysis of traffic images [14] or mobile robotics [26].

In our case, our work focuses on robust corner detection algorithms suitable for embedded applications. Thus, a corner detection algorithm is proposed with high spatio-temporal robustness to complex textured regions. The keystone of this algorithm consists in applying a non-textured corner filtering combined to a subpixel refinement. The algorithm is fully compliant with a hardware implementation and an FPGA architecture suitable for real-time embedded applications is proposed. Unlike to previous works, this new formulation reuses the information processed by the corner detection algorithm. Then, subpixel refinement and non-textured corner filtering are a part of the corner detection formulation. They cannot be considered as post-processing steps.

3 Robust/accurate feature extraction algorithm suitable for smart cameras.

Our work is based on the Shi-Tomasi formulation[36]. The Shi-Tomasi algorithm is a good trade-off between performance for real world scenarios and high speed processing. As explained above, new feature extraction, subpixel refinement and a non-textured corner filter are combined to increase the performance and robustness of the original Shi-Tomasi corner extraction algorithm. In **Fig.** 3 an overview of our algorithm is shown.

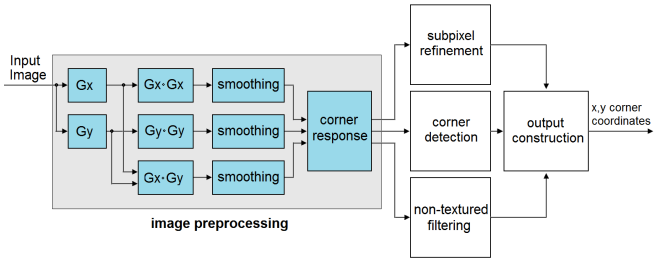


Fig. 3: Block diagram of the proposed algorithm

3.1 The preprocessing module

Given a gray scale image $I(i,j)$, horizontal and vertical gradients are given by: $G_x(i,j) = |I(i-1,j) - I(i+1,j)|, G_y(i,j) = |I(i,j-1) - I(i,j+1)|$. Absolute values in the gradient formulation are used to avoid signed variables and reduce the hardware resources utilization. Of course, this modification changes the performance of the original Shi-Tomasi algorithm, however, performance detriment is minimum in comparison with the decreasement of hardware resources. The difference between the formulation using signed values and formulation using absolute values is around 1% for the corner response image. From gradients, matrices A, B, C (x, y and xy gradient derivatives) are defined as: $A(i,j) = G_x(i,j) \cdot G_x(i,j), B(i,j) = G_y(i,j), \cdot G_y(i,j), C(i,j) = G_x(i,j) \cdot G_y(i,j)$.

A Gaussian filtering is applied on the A, B, C matrices to reduce noise and to remove fine-scale structures that affect the performance of the corner response. In order to reach high performance for embedded applications, convolution steps of our feature extraction algorithm takes inspiration from our previous work [3] where, in order to reach straightforward FPGA implementation, a fixed kernel with values simplified/rounded to fixed point binary representation was proposed. In this work, we propose a fixed binary kernel as shown in **Eq.** 1. This kernel performs a 5x5 Gaussian kernel with $\sigma = 5/3$, and simplifies multiplication in the FPGA implementation by replacing them with shift register operations. This decreases the hardware resources during FPGA implementation, facilitates parallel/pipeline design and has low compromise compared with the original Gaussian kernel accuracy. This because, the average difference between the original Gaussian kernel and the modified kernel has a difference of 0.0112 (1.12%) then, it is possible to assume that results using the modified kernel has to be very close than the results using the original.

$$\begin{pmatrix} \frac{8}{512} & \frac{16}{512} & \frac{16}{512} & \frac{16}{512} & \frac{8}{512} \\ \frac{16}{512} & \frac{32}{512} & \frac{32}{512} & \frac{32}{512} & \frac{16}{512} \\ \frac{16}{512} & \frac{32}{512} & \frac{32}{512} & \frac{32}{512} & \frac{19}{5122} \\ \frac{16}{512} & \frac{32}{512} & \frac{32}{512} & \frac{32}{512} & \frac{16}{512} \\ \frac{8}{512} & \frac{16}{512} & \frac{16}{512} & \frac{16}{512} & \frac{8}{512} \end{pmatrix} \tag{1}$$

3.2 Corner detection

The original Shi & Tomasi corner response **Eq.** 2, provides a high response value for corners and low response otherwise, as illustrated in **Fig.** 4b. In order to determine if a pixel P is a corner or not, the maximum values of the corner response could be retained. Of course, many pixels around each corner are also detected in spite of a filtering with a threshold σ. These pixels are false feature candidates and have low temporal stability.

$$D(i,j) = (A(i,j) + B(i,j)) - \sqrt{(A(i,j) - B(i,j))^2 + 4C(i,j)^2} \tag{2}$$

A way of solving the false feature candidates consists in applying a non-maxima suppression step. In our case, we consider that an appropriate FPGA-based non-maxima suppression step could be defined as follows:

$$\psi(u,v) = D(i-1:i+1\ ,\ j-1:j+1) * M \tag{3}$$

$$\gamma(i,j) = max(\psi) \tag{4}$$

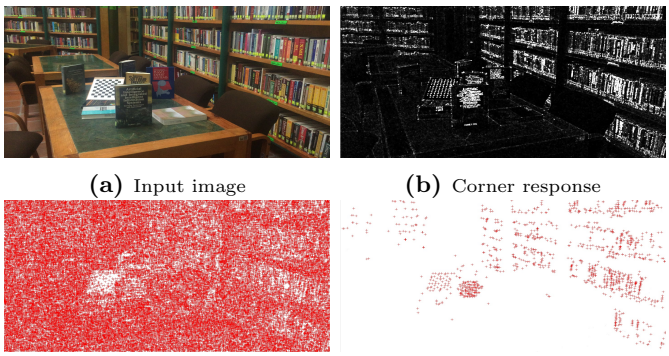
where

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

A thresholding (σ_1) is applied on $D(i,j)$ to select the good corner features.

$$E(i,j) = \begin{cases} 1 & if\ \sigma_1 > D(i,j) > \gamma(i,j) \\ 0 & if\ otherwise \end{cases} \tag{5}$$

This process is illustrated in **Fig.** 4c, where maximum corner response values $E(i,j)$ with a thresholding equal to cero are displayed as crosses. On the other hand, in **Fig.** 4d, a thresholding ($\sigma_1 = 0.1$) applied on the maximum corner response values $E(i,j)$ is shown. Low values imply many corners as in **Fig.** 4c, often with a low stability. On the contrary, higher threshold values improves the robustness and decreases the number of detected corners. In practice, a threshold value between $0.01 > \sigma_1 > 0.1$ provides good balance between number of retained corners and robustness (More details are given in **Section**-5).



(a) Input image **(b)** Corner response

(c) Corner detection, $\sigma_1 = 0$ **(d)** Corner detection, $\sigma_1 = 0.1$

Fig. 4: Corner detection

3.3 Non-textured corner filtering

A robust corner is a point/pixel located at the intersection of two or more edges. Unfortunately, textured regions are highly responsive to corner detectors (like the Shi and Tomasi) and do not represent robust corner features. In this way, we propose a non-textured corner filtering based on a triple surrounding patch around each candidate corner $D(i,j)$. As described in **Fig.** 5, three patches, each composed of four regions of interest, are applied around $D(i,j)$.

Each patch is extracted around $D(i,j)$ and a value of texturization $\tau(i,j)$ is computed such as:

$$\tau_{12}(i,j) = \sum |patch_1(i,j) - patch_2(i,j)| \tag{6}$$

$$\tau_{23}(i,j) = \sum |patch_2(i,j) - patch_3(i,j)| \tag{7}$$

$$\tau(i,j) = \frac{(\tau_{12}(i,j) + \tau_{23}(i,j))}{2} \tag{8}$$

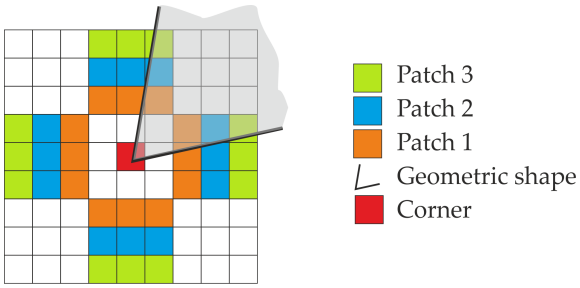
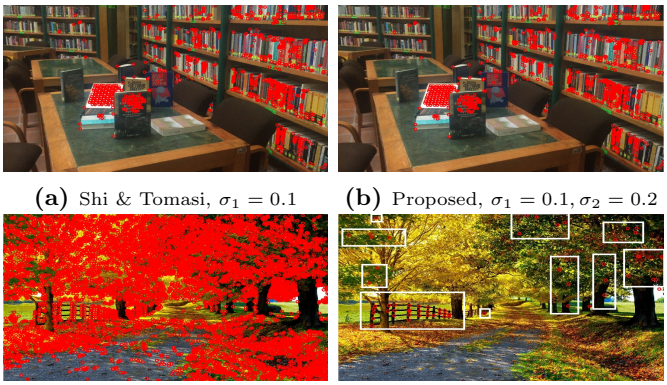


Fig. 5: Patches in the non-textured corner filter. We assume that a robust corner has to be associated with a geometric shape in which all pixels must have similar corner response. Then, it has to be similar corner response across all the patches; otherwise, the detected corner is an isolated point within a complex textured region.

Finally, the candidate corner is selected by a comparison to a threshold (σ_2). An example of non-textured corner filtering is shown is figure 6d.

$$F(i,j) = \begin{cases} 1, \tau(i,j) < \sigma_2 \\ 0, otherwise \end{cases} \quad (9)$$



(a) Shi & Tomasi, $\sigma_1 = 0.1$ **(b)** Proposed, $\sigma_1 = 0.1, \sigma_2 = 0.2$

(c) Shi & Tomasi, $\sigma_1 = 0.1$ **(d)** Proposed, $\sigma_1 = 0.1, \sigma_2 = 0.2$

Fig. 6: For input images with no complex texture regions our algorithm detects high number of features (b), similar to the original Shi-Tomasi formulation (a). For images with complex textured regions our non-textured corner filtering only retains robust features (d).

3.4 Subpixel refinement module

In previous work, one of the most used approaches uses the grayscale values of the input image and then, a Gaussian/Quadratic fitting is applied over the extracted corners in order to refine the location previously computed [19, 38, 48]. Although Gaussian/Quadratic fitting using grayscale values achieves relatively high performance, one solution more suitable for FPGA architectures could be a mathematical fashion that uses the same input that the corner extraction step. In this case, the feature extraction and the subpixel refinement could be computed in parallel, in addition, the use of the same input allows the use of the same buffer, this could decreases the hardware resources usage.

3.4.1 Subpixel location using the Least Squares Fitting

Considering a group of observed data $x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_n, y_n, z_n$,: where x, y are the image pixel location while z is the corner metric response (**Eq.** 2), any fitting function $f(x, y)$ should fulfil with the standard least squares equation:

$$I = \sum_{i=1}^{i=n} (z_i - f(x_i, y_i))^2$$

where x, y are the independent variables, z is the dependent variable, $n \geq k$, k is the number of independent parameters in the function $f(x, y)$, and it is also the least number of samples required [45]. Considering that the fitting technique can be generalized from a best-fit line to a best-fit polynomial, if a low-order polynomial is employed, the fitting accuracy must be bad, while high-order polynomials may lead to unstable fitting results. In this work, we select a quadratic polynomial function ($k = 6$, i.e., $\alpha_0, \alpha_1, \ldots, \alpha_5$) as shown in **Eq.** 10 to fit a parabolic surface, and then, obtain the parameters of this function through least squares adjustment.

$$f(x,y) = \alpha_0 x^2 + \alpha_1 y^2 + \alpha_2 xy + \alpha_3 x + \alpha_4 y + \alpha_5 \quad (10)$$

Considering that the generalized adjustment model could be expressed as follows:

$$L_{n,1} = B_{n,k} \cdot X_{k,1} + d_{n,1}$$

where $X_{k,1}$ denotes the k independent parameters, $L_{n,1}$ are the n samples and $d_{n,1}$ is the constant item in the expression (in this case $d_{n,1} = 0$). In such scenario the problem is to set an appropriate weight determination for the independent variables ($B_{n,k}$) [45]. One approach to solve this problem is to set a Gaussian weight distribution as initial solution, then it is necessary to iterate using an adjustment criterion that refines the first approximation [48]. In our case, we propose a direct weight determination using the Vandermonde Matrix as weight determination for the independent variables. In the past Vandermonde Matrix has been used under polynomial interpolation procedures obtained promising results [42] so, in this work we assume that similar performance could be reach under our application domain. Using the Vandermonde Matrix as weight determination for the independent variables, then, it is possible to avoid the iterative procedures required in previous work and therefore, simplify the hardware implementation. i.e., given the weight determination for the independent variables, ($B_{n,k}$) obtained via the Vandermonde Matrix and considering $X_{k,1}$ as corner responses from an image, then, it is possible to compute subpixel position within a parabolic surface as $L_{n,1} = B_{n,k} \cdot X_{k,1} + 0$.

3.4.2 The proposed approach

Considering that a 3×3 template window as shown in **Fig.** 7, **Eq.** 10 is used to carry out the least squares fitting. The sample values $X_{k,1}(k = 0, 1, \ldots, n)$ are corner metric responses, $n = 9$ is the number of sample values, and set: $X_{k,1} = (s_0, s_1, \ldots, s_n)^T$, where $s_0, s_1, \ldots, s_5$ are the six independent parameters in the Least Squares fitting. Our algorithm selects any six corner responses at reasonably small distances from the center point as independent parameters, for practical purpose we used, $S0, S3, S5, S7, S2$ and $S6$. Then, the parameters in the fitting function can then be calculated as: $L_{n,1} = B_{n,k} \cdot X_{k,1}$, where $B_{n,k}$ is the weight determination for the independent variables and it is defined as follows: given the independent parameters as $\{S0, S3, S5, S7, S2, S6\}$. x, y displacements with respect the center $S4$ could be defined as $x = \{-1, -1, 1, 0, 1, -1\}$, $y = \{-1, 0, 0, 1, -1, 1\}$. Then, two different Vandermonde Matrices **Eq.** 11 and 12 are computed. For poractical prposes we uses the `vander` Matlab fucntion to compute the V_x, V_y matrices. Finally, we defined $B_{n,k}$ as $(V_x * V_y)/S_n$, this because the matrices multiplication between V_x, V_y provide weigh response for the x, y axis using a single matrix [42]. S_n is the number of observations used in the interpolation process, in this case $\{S0, S3, S5, S7, S2, S6\}$.

$$V_x = \begin{pmatrix} -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{pmatrix} \tag{11}$$

$$V_y = \begin{pmatrix} -1 & 1 & -1 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{12}$$

$\mathbf{S}_0$ (–1,–1)	$\mathbf{S}_1$ (0,–1)	$\mathbf{S}_2$ (1,–1)
$\mathbf{S}_3$ (–1,0)	$\mathbf{S}_4$ (0,0)	$\mathbf{S}_5$ (1,0)
$\mathbf{S}_6$ (–1,1)	$\mathbf{S}_7$ (0,1)	$\mathbf{S}_8$ (1,1)

Fig. 7: Template window for least squares fitting.

After calculating the sample values in the Least Squares Fitting $(L_{n,1})$, we apply the formulation presented in [48], therefore, the decimal part of the features extracted can be calculated as:

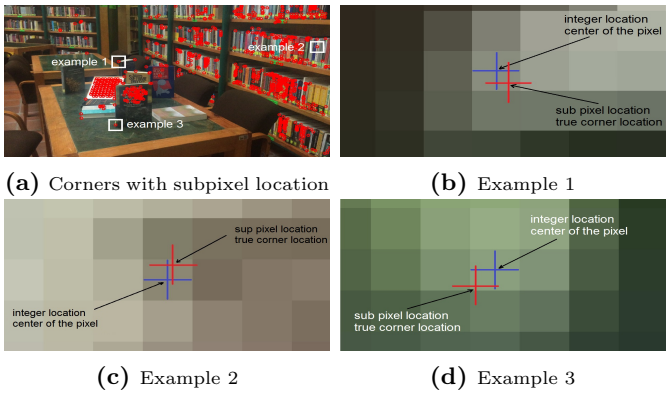
$$\begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix} = \begin{pmatrix} \frac{2L_{1,1}L_{2,1} - L_{2,1}L_{4,1}}{(L_{2,1})^2 - 4L_{1,1}L_{0,1}} \\ \frac{2L_{0,1}L_{4,1} - L_{2,1}L_{3,1}}{(L_{2,1})^2 - 4L_{1,1}L_{0,1}} \end{pmatrix} \tag{13}$$

So the sub pixel location of the extracted features is:

$$x_{sp}(i, j) = i + \Delta_x \tag{14}$$

$$y_{sp}(i, j) = j + \Delta_y \tag{15}$$

where x_{sp}, y_{sp} are the refined subpixel locations, this process is illustrated in **Fig.** 8.



(a) Corners with subpixel location **(b)** Example 1

(c) Example 2 **(d)** Example 3

Fig. 8: Subpixel refinement by our mathematical formulation

3.5 Output construction module

The final step is the subpixel location of the corners retained after the non-textured filtering. Thus, when $E(i, j) = 1$ AND $F(i, j) = 1$, the coordinates of each corner are computed by:

$$X_{Corner(i,j)} = i + x_{sp}(i, j) \tag{16}$$

$$Y_{Corner(i,j)} = j + y_{sp}(i, j \tag{17}$$

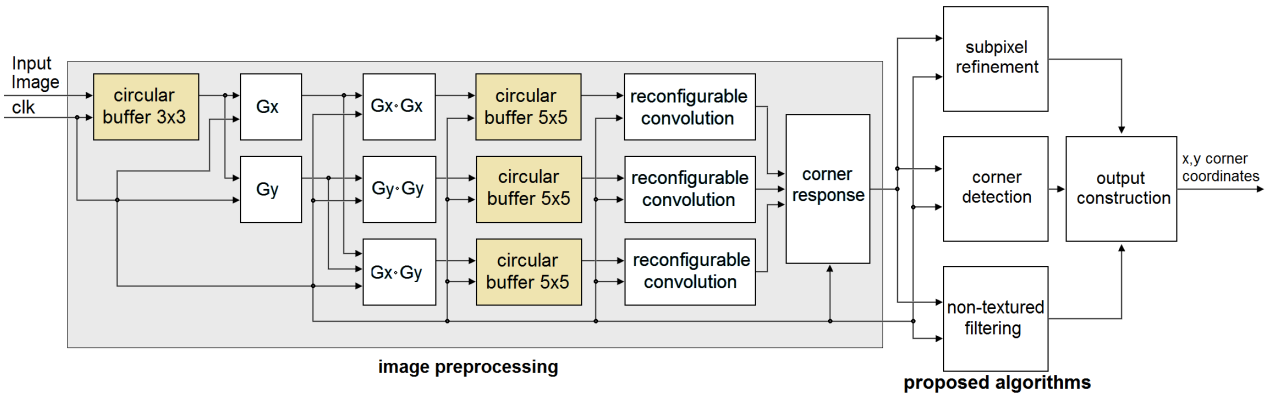


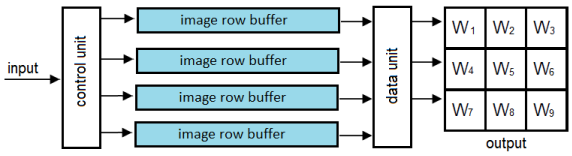
Fig. 9: FPGA architecture for the proposed algorithm. The FPGA architecture computes the vertical/horizontal gradients and the x, y, xy gradient derivatives in parallel. Also, the corner detection, subpixel refinenemt and non-textured filter are computed in parallel. Circular buffers attached to the local processors hold temporarily as cache and deliver parallel data to the processors.

4 FPGA architecture for the feature extraction algorithm.

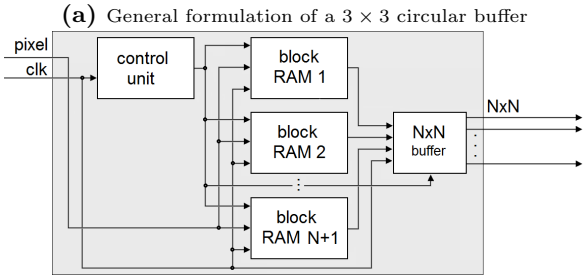
An overview of the developed FPGA architecture is illustrated in **Fig.** 9. The structure of the architecture are composed by four hardware processing elements: image preprocessing, subpixel refinement, corner detection and non-textured corner filter. The core of the FPGA architecture are circular buffers attached to the local processors that are used to hold local sections of the image and allow local parallel data access for parallel processing. In general, input images are processed in stream. First, the architecture reads/stores data/parts of the frames into circular buffers that can hold rows temporarily as cache, store image rows from the input images, and that can deliver parallel data to the image preprocessing module. For the image preprocessing module, the architecture computes the vertical and horizontal gradients. Then it computes the $A(i,j), B(i,j), C(i,j)$ variables. Circular buffers delivers image pixels for the smoothing operations and, reconfigurable convolution units (see [3]) compute the smoothing operation. Finally, the FPGA architecture computes the corner response metric, $D(i,j)) = (A(i,j) + B(i,j)) - \sqrt{(A(i,j) - B(i,j))^2 + 4C(i,j)^2}$, for that, we adapted the architecture developed by Yamin Li and Wanming Chu [28]. This architecture uses a shift register mechanism and compares the more significant/less significant bits, it allows to compute square root with low hardware resources, therefore, it allows for a convenient square root framework, suitable for our algorithmic formulation. Using the $D(i,j))$ computed by the image preprocessing module, three parallel modules carry out the corner detection, subpixel refinement and the non-textured corner filter in parallel. Finally, the output construction module delivers the refined positions for the corners retained after the non-textured filtering.

4.1 The circular buffers

In [3] we introduced a circular buffer schema in which input data from the previous N rows of an image can be stored using memory buffers (block RAMs/BRAMs) till the moment when a $N \times N$ neighborhood is scanned along subsequent rows. In this work, we follow a similar approach to achieve high data reuse and high level of parallelism. Then, our algorithm is processed in modules where all image patches can be read in parallel. First, a shift mechanism (**control unit**) manages the read/write addresses of N+1 BRAMs, in this formulation N BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, data inside the read mode BRAMs can be accessed in parallel and each pixel within a $N \times N$ region is delivered in parallel (**N × N buffer**), as shown in **Fig.** 10a. For more details see [3].



(a) General formulation of a 3 × 3 circular buffer



(b) FPGA architecture for the circular buffers

Fig. 10: The circular buffers architecture. For a N × N patch, a shift mechanism (**control unit**) manages the read/write addresses of N+1 BRAMs, in this formulation N BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, the **N × N buffer** delivers logic registers with all pixels within the patch in parallel.

4.2 FPGA architecture for feature extraction

In **Fig.** 11, the FPGA architecture for feature extraction is shown. First, a circular buffer delivers parallel data for a 3x3 processing window. i.e., for all pixel in the input image, the circular buffer delivers the nine pixels centered in a 3×3 patch in parallel. Using pixels within the patch, the processor computes the non-maxima suppression step, and then, based on the values obtained, a thresholding operation determines if the patch center is a corner or not.

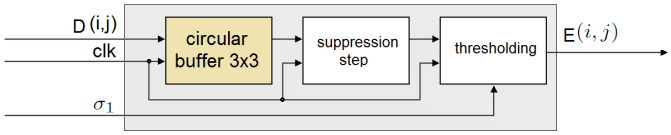


Fig. 11: FPGA architecture for the feature extraction. In a first instance, a suppression step over the corner response are computed. Then, a thresholding (σ_1) is applied in order to select the "good" corner features.

4.3 FPGA architecture for non-textured corner filtering

First, a circular buffer delivers parallel data for the non-textured corner filtering, in this case, all the pixels within a 9×9 image patch in parallel, the patch center is the location of the pixel being processed. Then, using the pixels within the patch, the processor computes the three patches that surround the tentative corner in parallel, as shown in **Fig.** 12. Then, our FPGA architecture carry out the comparisons between these patches and finally, this module estimate the geometric robustness of the corners. Using this geometric robustness and considering a threshold value provided by the user (σ_2), the module retains only corners with high geometric robustness.

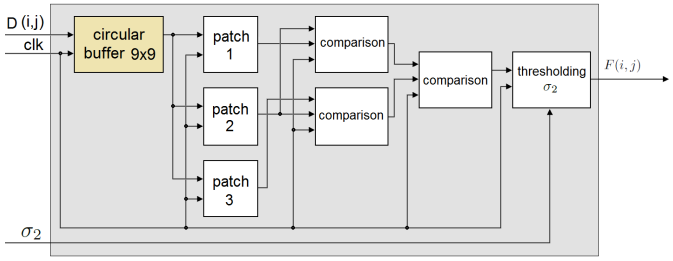


Fig. 12: FPGA architecture for the non-textured corner filter. Three patches that surround any possible corner are computed. Then, comparisons between these patches are computed. Finally, by using the patches comparisons, the geometric robustness of the corners is estimated.

4.4 FPGA architecture for subpixel refinement

For the subpixel refinement, we implement the proposed mathematical formulation using a parallel-pipeline approach, as shown in **Fig.** 13. First, a circular buffer delivers parallel data for the subpixel computation. i.e., the circular buffer delivers all pixels within a 3×3 image patch in parallel, the patch center is the location of the pixel being processed. Using pixels within the patch, the processor computes the sample values (α_1-α_6) in parallel. Then, based on the sample values, the FPGA architecture computes decimal part (Δ_x, Δ_y) for all possible corner points in the input image in parallel. In order to reduce hardware resources consumption, a Look Up Table (LUT) manages the quotients operations in our implementation.

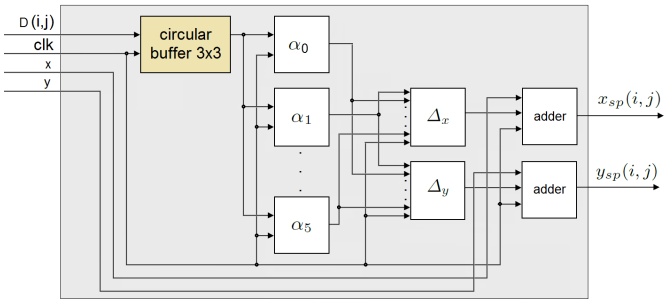


Fig. 13: FPGA architecture for the subpixel refinement. First, six independent parameters for a Least Squares fitting are computed. Then Least Squares Fitting refines the integer locations.

4.5 Output construction

Using the outputs: $E(i,j)$, $F(i,j)$ and $x_{sp}(i,j)$, $y_{sp}(i,j)$, the **output construction** module pplying logic comparisons between registers. Then, it computes the subpixel coordinates of the corner points after the non-textured filter. In practice, these subpixel coordinates could be used by any real world application: SfM, SLAM, camera calibration, etc.

5 Results and discussion

The developed FPGA architecture was implemented in an FPGA Cyclone IV EP4CGX150CF23C8 of Altera. All modules were designed via Quartus II Web Edition version 10.1SP1. All modules were validated via post-synthesis simulations performed in ModelSim Altera. For all test, we consider $\sigma_1 = 0.1, \sigma_2 = 0.1$ since these values provided high number of "good" features (it is possible to obtain more than 10.000 features per frame and these features have high temporality stability, therefore, they are easy to track) under large set of different indoor/outdoor scenarios. In practice, we recommend these values as reference between high number of detected features and high temporality stability. Lower values of σ_1, σ_2 could detect more features, however, temporality stability could be decreased. On the other hand, higher values of σ_1, σ_2 should provide more temporality stability but number of features detected is decreased.

5.1 Performance compared with previous work

The full hardware resource consumption of the architecture is shown in **Table** 1. Our algorithm formulation allows for a compact system design, it requires 4% of the total logic elements. For memory bits, our architecture uses 8% of the total resources, this represents 34 block RAMs consumed mainly in the circular buffers. These hardware utilization enables to target a smaller FPGA device and therefore could be possible a small FPGA-based smart camera, suitable for real-time embedded applications.

Table 1: Hardware resource consumption for the developed FPGA architecture

Resource	Demand
Total logic elements	6,507 (4%)
Total pins	52 (12%)
Total Memory Bits	803,104 (8%)
Embedded multiplier elements	0 (0%)
Total PLLs	0 (0%)

In comparison with previous work, in **Table** 2 we present hardware resource utilization between our FPGA architecture and previous FPGA-based feature extraction algorithms. For the FAST algorithm, there are several works [14, 26, 9] which FPGA implementations take advantages of the mathematical formulation of the FAST algorithm. For all test, we compared Harris and Shi-Tomasi formulations in straightforward form. This because the Shi-Tomasi corner detector is based entirely on the Harris corner detector. In general, one modifications on the corner response function makes the Shi-Tomasi corner detector more robust under illumination changes (that is useful to track the features). Unfortunately, this modification uses one square root that limit the hardware implementation. In this work, we compute square roots adapting the algorithm presented in [28], then, we introduce an FPGA-based implementation on Shi-Tomasi feature extraction algorithm. In general, FAST-based approaches not require block RAM cores since the original FAST formulation allows straightforward pipeline reformulation. Therefore, the hardware resource demand is low compared with our approach (Shi-Tomasi based approach) and low compared with previous Harris/Shi-Tomasi based approaches [12, 4, 24, 33]. The reason for FPGA architectures based on the original Harris-Stephens/Shi-Tomasi is the low robustness of the features detected by the FAST algorithm. In general, features detected by FAST-based approaches have high noise sensitivity and have very low performance for complex texture regions. Compared with previous Harris-Stephens/Shi-Tomasi based algorithms, our algorithm formulation which replaces quotients by Look-up Tables and that uses reconfigurable convolution units [3], our algorithm allows lower hardware consumption than [24] and [33]. In addition, our algorithm allows similar hardware requirements than the more efficient Harris-Stephens implementations [4], and only [4] has lower hardware requirements than our algorithm but without subpixel computation and the robustness of our approach.

Table 2: Hardware resource consumption comparisons

Method	Logic elements	Block RAMs	Approach
Dinh et al [14]	2,960	-	FAST
Kraft et al [26]	3,915	-	FAST
Brenot et al [9]	4,244	-	FAST
Chao and Wong [12]	14,184	69	Harris
Amaricai et al [4]	4,149	23	Harris
Hsiao et al [24]	8,050	36	Harris
Possa et al [33]	8,624	43	Harris
This work*	6,507	28	Shi-Tomasi

*Operating frequency = 100 MHz, image resolution 1200×720

In **Table** 3, speed processing for the proposed feature extraction algorithm for different image resolutions is shown. For that, we synthesized different versions of our FPGA architecture (**Fig.** 9), in these versions, we modified the circular buffers in order to work with all tested image resolutions. Then, we carried out post-synthesis simulation in ModelSim Altera. In all cases, our FPGA architecture allows for real-time processing. When compared with previous work (**Table** 4), our algorithm provides the highest speed processing under Full HD images, it outperforms several previous work [14, 9, 26, 24], and for HD images, our algorithm reaches speed processing similar to the more efficient Harris-based approaches [4].

Table 3: Processing speed for different image resolutions

Resolution	Frames/s	Pixels/s
1920×1080	44	91,238,400
1200×720	105	90,720,000
512×512	346	90,701,824
200×200	2247	89,880,000

*Operating frequency = 100 MHz

Table 4: Processing speed comparisons

Method	Resolution	Frames/s	Pixels/s
Dinh et al [14]	1920×1080	22	45,619,200
Kraft et al [26]	512×512	23	6,029,312
Brenot et al [9]	1920×1080	24	49,766,400
Chao and Wong [12]	640×480	98	30,105,600
Amaricai et al [4]	1200×720	134	114,776,000
Hsiao et al [24]	640×480	50	15,360,000
Possa et al [33]	1024 ×1024	93	97,517,568
This work	1920×1080	44	91,238,400

5.2 Performance compared with the original Shi-Tomasi algorithm

In order to validate the accuracy of our subpixel refinement step, we create a dataset as shown in **Fig.** 14. This dataset consist in eight different images of 166 × 150 pixel resolution. In each image we set a testing point at the 17.60, 129.40 spatial position and, each testing point where configured to fulfil with all possible configurations of the corners points presented in a real world scenario (0° corner, 30° corner, 45° corner, 60° corner, 30° solid corner, 45° solid corner, 60° solid corner, 90° solid corner). In order to generate the testing images, we use Matlab. Thus, we use the Matlab plotting functions in order to set the testing points and to draw the figures within the images. The generated images were saved as .jpg images. Finally, we apply the original Shi-Tomasi algorithm and our algorithm over the generated images. For the original Shi-Tomasi algorithm, integer locations introduce a constant inaccuracy of 0.4 pixels in the x, y coordinates, respectively, as shown in **Table** 5. For our algorithm, the subpixel refinement step decreases the error in the x, y coordinates, error is near $\times 10$ lower than the original Shi-Tomasi algorithm.

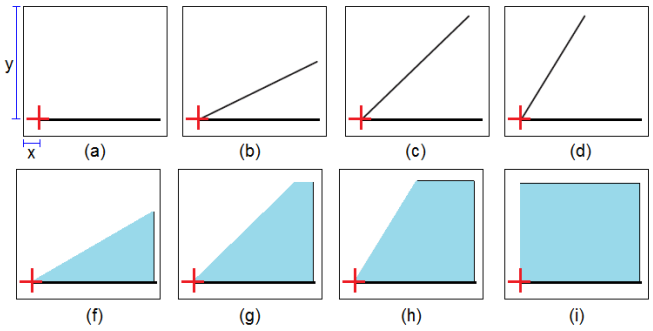


Fig. 14: Synthetic images used for accuracy validation. Interest points are marked as crosses: (a) 0° corner; (b) 30° corner; (c) 45° corner; (d) 60° corner; (e) 30° solid corner; (f) 45° solid corner; (g) 60° solid corner; (h) 90° solid corner.

Table 5: Accuracy comparison between the proposed algorithm and the original Shi-Tomasi algorithm.

Test	Original Shi-Tomasi (x,y)	This work (x,y)
(a) 0° corner	18, 129	17.68, 129.46
(b) 30° corner	18, 129	17.56, 129.47
(c) 45° corner	18, 129	17.67, 129.36
(d) 60° corner	18, 129	17.58, 129.38
(f) 30° solid corner	18, 129	17.64, 129.33
(g) 45° solid corner	18, 129	17.67, 129.46
(h) 60° solid corner	18, 129	17.54, 129.47
(i) 90° solid corner	18, 129	17.54, 129.36
mean error (in pixels)	**0.4, 0.4**	**0.055, 0.051**

In order to validate the non-textured filter performance, we measured the repeatability of the corners detected by the original Shi-Tomasi algorithm and our feature extraction algorithm using our non-textured filter. For that, we recorded and testing eight different video sequences under different real world scenarios, see **Table** 6. First, we extract feature points from frame 1, then, we track feature points using the approach presented in [1]. This approach deliver high accuracy in terms of feature tracking (more accurate than the most used algorithms such as SIFT and SURF) but algorithmic formulation is highly exhaustive and, an FPGA architecture is necessary in order to reach real-time processing. Finally, in order to measure the repeatability, we measure the relation between the input/output features number in the feature tracking algorithm. i.e., the feature tracking algorithm assumes that all input features can be tracked, therefore, input features number must be equal to the output features number. In practice this is not true and the correlation function in the feature tracking algorithm only retains features with high temporal robustness (see **Eq.** 8 in paper [1]), i.e., it measure the robustness of the features detected by any feature extraction algorithm. For practice, in this work we express this robustness measurement as $\rho = \Delta_{out}/\Delta_{in}$, where Δ_{in} is the number of input features in the feature tracking algotimh while Δ_{out} is the output features number. In all tested videos the original Shi-Tomasi algorithm detect near than 5x more feature points that our feature extraction algorithm, however, robustness is .1575. i.e., only 15.75% of the detected features can be tracked with an acceptable accuracy. Considering commonly used feature tracking algorithms (SIFT, SURF), false matches could be a problem, then, a false matches filtering step (RANSAC) could be necessary. Using the proposed algorithm, robustness is .8500, i.e., 85% of the detected features can be tracked with an acceptable accuracy. In practice, simple binary descriptors like BRIEF, BRISK, ORB, without RANSAC post-processing step, could take advantage by applying the proposed algorithm since most of the detected features have high temporal stability and therefore, they are easy to track.

Table 6: Robustness comparison between the proposed algorithm and the original Shi-Tomasi algorithm.

Test	Original Shi-Tomasi (ρ)	This work (ρ)
1	1768 / 9658 = .18	1248/1487 = .83
2	1559 / 8778 = .17	824/983 = .83
3	1676 / 8848 = .18	814/924 = .88
4	1183 / 9444 = .12	1187/1384 = .82
5	997 / 7323 = .13	402/445 = .88
6	1259 / 7645 = .16	507/559 = .90
7	1234 / 8287 = .14	827/984 = .84
8	1538 / 8288 = .18	987/1193 = .82
mean robustness (ρ)	**.1575**	**.8500**

In **Fig.** 15, we show in graphical form the scope and performance for the non-textured corner filter. For low threshold values, the original Shi-Tomasi formulation delivers high number of features that are difficult to track, as shown in the left column of **Fig.** 15. On the other hand, high threshold values often deliver highly robust features, as illustrated in the central column of **Fig.** 15. However, the number of features are low, and in several applications such as 3D reconstruction, SfM, SLAM, etc., low number of features imply sparse 3D reconstructions that make difficult to understand the environment. For SfM, SLAM, low number of features often make difficult to estimate the camera pose. Using our feature extraction algorithm, the non-textured corner filter retains a high number of features (right column of **Fig.** 15), even under input images with complex texture, as shown in **Fig.** 17.

Finally, for practical real world applications, in **Fig.** 16 and 17 we show the performance for a 3D reconstruction application. In both cases squares are the features detected by the feature extractor module while circles are the features retained after the non-textured corner filter. The retained features were tracked across different viewpoints from the same scene. Then, we compute the corresponding 3D reconstruction following the formulation presented in [2]. In **Fig.** 16 we show the performance for indoor scenarios. As shown in **Fig.** 16d the subpixel refinement module increases the accuracy of the 3D reconstruction. In **Fig.** 17 we show the performance for outdoor scenarios. In this case, low threshold values in the original Shi-Tomasi formulation deliver high number of features that are difficult to track while high threshold values deliver low number of features and the 3D reconstruction is sparse. By using our formulation, it is possible to retain a higher number of features that are easy to track and therefore, deliver semi dense 3D reconstruction under input images with complex textured regions.

A Pseudo code for the proposed algorithm

Parameter definition:
I: input image
$X_{resolution}$, $Y_{resolution}$: the size of the imput image
σ_1: threshold for the corner detector
σ_2: threshold for the non-textured corner filtering

preprocessing step:

For all pixels $I(i,j)$ which satisfy $i >= 2; j >= 2$ and $i <= X_{resolution} - 1$, $j <= Y_{resolution} - 1$

1: Compute the horizontal gradient $G_x(i,j)$
End
For all pixels $I(i,j)$ which satisfy $i >= 2; j >= 2$ and $i <= X_{resolution} - 1$, $j <= Y_{resolution} - 1$

2: Compute the vertical gradient $G_y(i,j)$
End

For all pixels $G_x(i,j)$, $G_y(i,j)$

3: $A = G_x(i,j) * G_x(i,j)$
4: $B = G_y(i,j) * G_y(i,j)$
5: $C = G_x(i,j) * G_y(i,j)$
6: $D(i,j)$, **Eq.** 2
End

Corner detection step:

For all pixels $D(i,j)$

7: $\psi(u,v)$, **Eq.** 3
8: $max(\psi)$, **Eq.** 4
9: $E(i,j)$, **Eq.** 5
End

Non-textured corner filtering:

For all pixels $D(i,j)$
10: $\tau_{12}(i,j)$, **Eq.** 6
11: $\tau_{23}(i,j)$, **Eq.** 7
12: $\tau(i,j)$, **Eq.** 8
13: $F(i,j)$, **Eq.** 9
End

Subpixel refinement step:

For all pixels $D(i,j)$

14: $L_{n,1}$, **Eq.**
15: $\begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix}$, **Eq.** 13
16: $x_{sp}(i,j)$, **Eq.** 14
17: $y_{sp}(i,j)$, **Eq.** 15
End

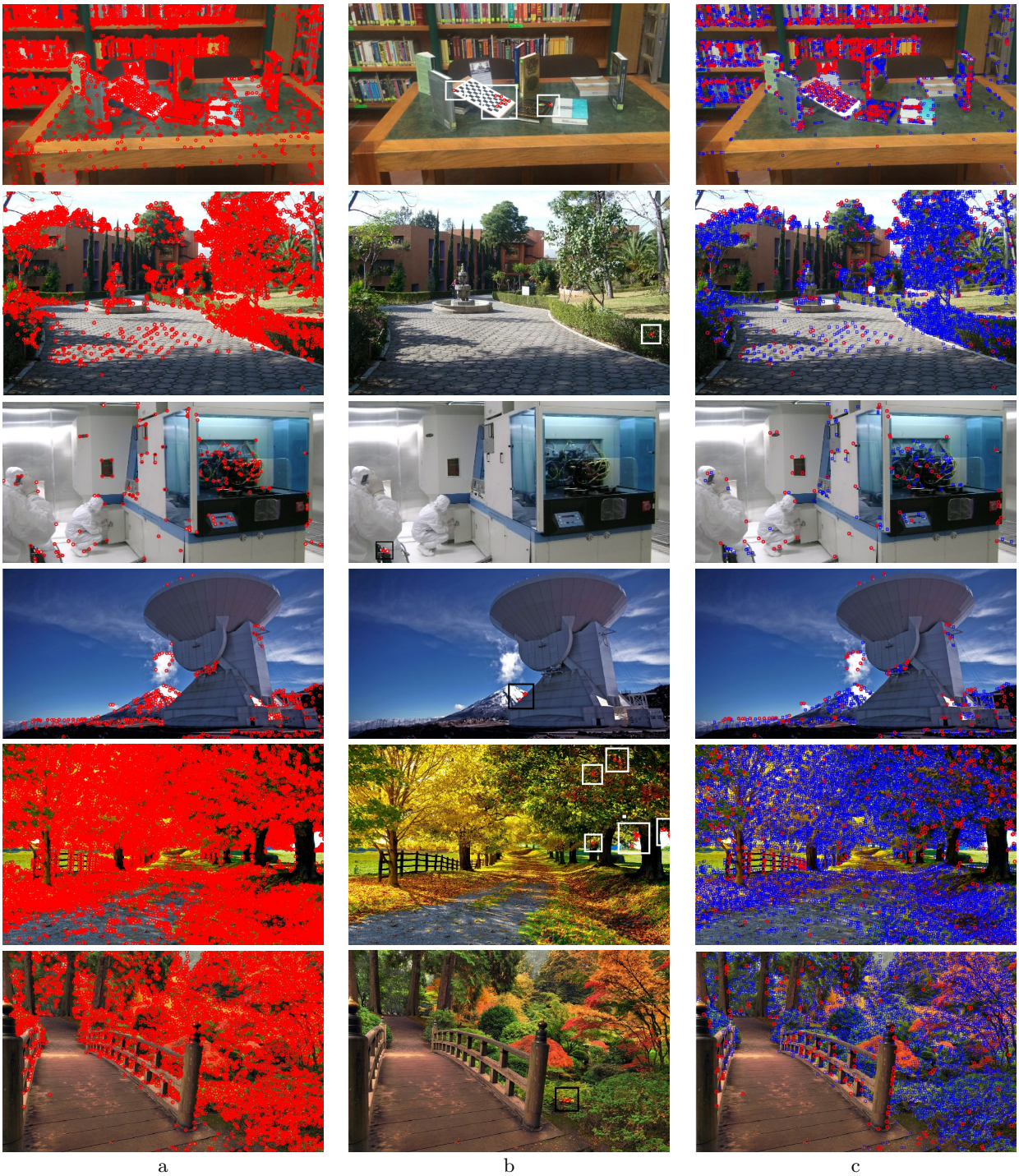
Output construction:

For all pixels $E(i,j), F(i,j), x_{sp}(i,j), y_{sp}(i,j)$

18: $X_{Corner(i,j)}$, **Eq.** 16
19: $Y_{Corner(i,j)}$, **Eq.** 17
End

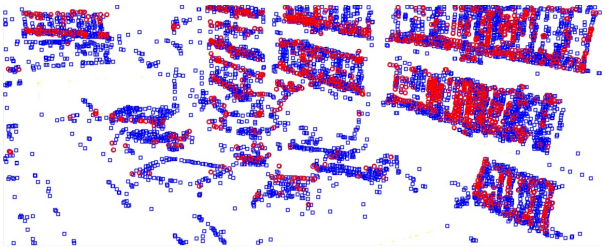
B Conclusions

In this article, we have introduced a new feature extraction algorithm suitable for smart camera implementation. Our algorithm is robust enough to deliver high number of robust features for image sequences with high number of complex textured regions and at the same time it delivers high performance for real-time embedded applications. We have proposed a non-textured corner filter that retain high number of robust features for images with complex textured regions, and we have proposed its subpixel refinement. Both algorithms increase the performance and scope of the original Shi-Tomasi corner detection algorithm. We proposed an FPGA architecture that allows real-time processing and compact system design and we validated our FPGA architecture via post-synthesis simulations. Our results are encouraging and show the feasibility of our algorithmic approach. The FPGA architecture reuses the the corner response values used in the corner detection module, and computes the subpixel refinement and the non-textured filtering modules in parallel. This enables an efficient hardware resources utilization, lower than several previous formulations without subpixel refinement and without non-textured corner filter, and similar hardware resources than the most efficient FPGA-based Harris corner detection. Finally, our FPGA architecture delivers high speed processing (it can process up to 44 frames/91,238,400 pixels per second for Full HD images), higher than most previous work and similar speed processing than the more efficient FPGA-based Harris corner detection reported. Since many vision algorithms rely on finding and tracking features, we consider this work can be useful in several real-time image-processing applications such as: Structure from Motion and Simultaneous Localization and Mapping. As work in progress, we are implementing the developed FPGA architecture into the DreamCam, a robust/flexible smart camera [6].

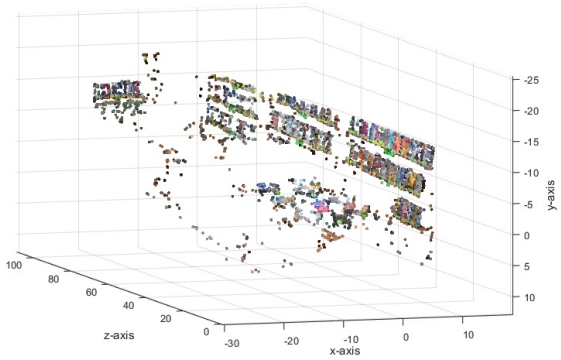


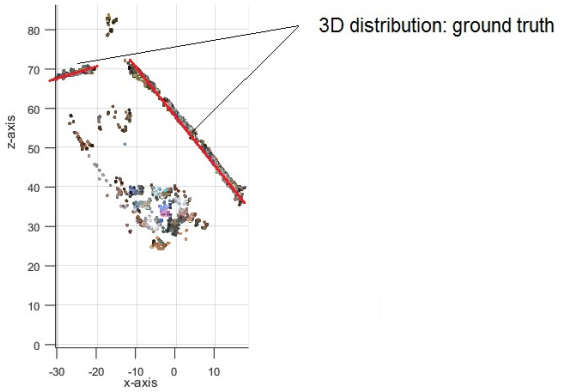
a b c

Fig. 15: Non-textured corner filter performance. (a) Original Shi-Tomasi formulation, red points are the feature extraction considering $\sigma = 0.01$. For low threshold values, several features are detected, however, in most cases, detected features have low temporal stability. (b) Original Shi-Tomasi formulation, red points are the feature extraction considering $\sigma = 0.1$. For high threshold values, detected features have high temporal stability, but detected features number is low. (c) The proposed formulation considering $\sigma_1 = 0.01, \sigma_2 = 0.2$, in this case low threshold values detect high number of features (blue points) but some of them with low temporal stability. In order to solve this problem, the Non-textured corner filter retain only features with high temporal stability (red points).



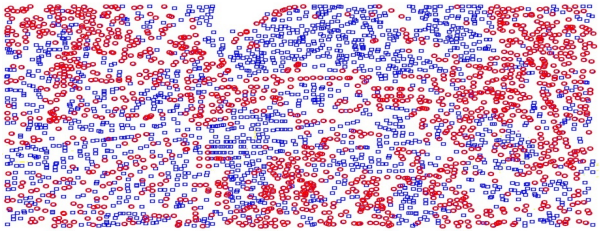
(a) Input scene: interior of the INAOE library.

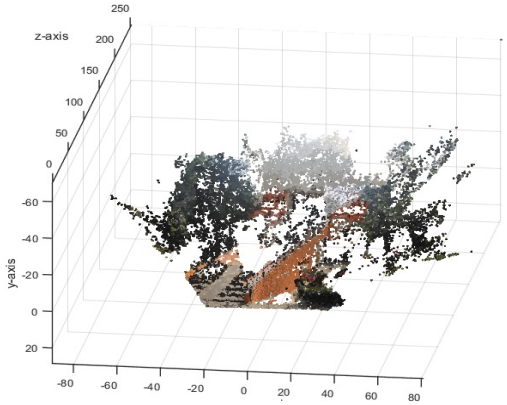


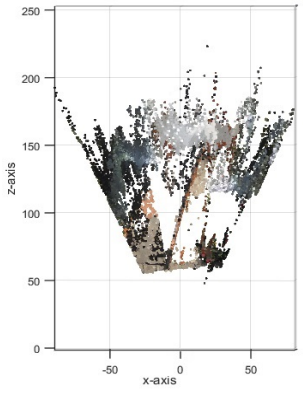


(c) 3D reconstruction: front view. **(d)** 3D reconstruction: top view.

Fig. 16: 3D reconstruction using the proposed algorithm. Blue points are the features detected by the feature extractor module, some of them have low temporal stability. Red points are the features retained after the non-textured corner filter. The retained features in (b) were tracked across different viewpoints from the same scene. Then, we compute the corresponding 3D reconstruction (c). In this case, subpixel refinement decreases the inaccuracy in the 3D reconstruction (d). When we compare with the original Shi-Tomasi algorithm (without subpixel refinement) **Fig.** 2d, the quality of the 3D reconstruction by our algorithm is higher, close to the ground truth.







(c) 3D reconstruction: front view. **(d)** 3D reconstruction: top view.

Fig. 17: 3D reconstruction using the proposed algorithm. Blue points are the features detected by the feature extractor module, some of them have low temporal stability. Red points are the features retained after the non-textured corner filter. The retained features in (b) were tracked across different viewpoints from the same scene. Then, we compute the corresponding 3D reconstruction (c) (d).

References

1. Aguilar-González A, Arias-Estrada M (2016) Dense mapping for monocular-slam. In: Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on, IEEE, pp 1–8
2. Aguilar-González A, Arias-Estrada M (2016) Towards a smart camera for monocular slam. In: Proceedings of the 10th International Conference on Distributed Smart Camera, ACM, pp 128–135
3. Aguilar-González A, Arias-Estrada M, Pérez-Patricio M, Camas-Anzueto J (2015) An fpga 2d-convolution unit based on the caph language. Journal of Real-Time Image Processing pp 1–15
4. Amaricai A, Gavriliu CE, Boncalo O (2014) An fpga sliding window-based architecture harris corner detector. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL), IEEE, pp 1–4
5. Aydogdu MF, Demirci MF, Kasnakoglu C (2013) Pipelining harris corner detection with a tiny fpga for a mobile robot. In: Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on, IEEE, pp 2177–2184
6. Birem M, Berry F (2014) Dreamcam: A modular fpga-based smart camera architecture. Journal of Systems Architecture 60(6):519–527
7. Bourrasset C, Maggianiy L, Sérot J, Berry F, Pagano P (2013) Distributed fpga-based smart camera architecture for computer vision applications. In: Distributed Smart Cameras (ICDSC), 2013 Seventh International Conference on, IEEE, pp 1–2
8. Bravo I, Baliñas J, Gardel A, Lázaro JL, Espinosa F, García J (2011) Efficient smart cmos camera based on fpgas oriented to embedded image processing. Sensors 11(3):2282–2303
9. Brenot F, Fillatreau P, Piat J (2015) Fpga based accelerator for visual features detection. In: Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2015 IEEE International Workshop of, IEEE, pp 1–6
10. Canny J (1986) A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence (6):679–698
11. Carey SJ, Barr DR, Dudek P (2013) Low power high-performance smart camera system based on scamp vision sensor. Journal of Systems Architecture 59(10):889–899
12. Chao TL, Wong KH (2015) An efficient fpga implementation of the harris corner feature detector. In: Machine Vision Applications (MVA), 2015 14th IAPR International Conference on, IEEE, pp 89–93
13. Choi C, Christensen HI (2012) 3d textureless object detection and tracking: An edge-based approach. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, pp 3877–3884
14. Dinh TH, Vu DQ, Ngo VD, Ngoc NP, Truong VT (2014) High throughput fpga architecture for corner detection in traffic images. In: Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on, IEEE, pp 297–302
15. Feichtenhofer C, Pinz A (2013) Spatio-temporal good features to track. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp 246–253
16. Fularz M, Kraft M, Schmidt A, Kasiński A (2015) The architecture of an embedded smart camera for intelligent inspection and surveillance. In: Progress in Automation, Robotics and Measuring Techniques, Springer, pp 43–52
17. Gauglitz S, Höllerer T, Turk M (2011) Evaluation of interest point detectors and feature descriptors for visual tracking. International journal of computer vision 94(3):335–360
18. Gil A, Mozos OM, Ballesta M, Reinoso O (2010) A comparative evaluation of interest point detectors and local descriptors for visual slam. Machine Vision and Applications 21(6):905–920
19. Han Y, Chen P, Meng T (2015) Harris corner detection algorithm at sub-pixel level and its application
20. Haritaoglu I, Harwood D, Davis LS (2000) W 4: Real-time surveillance of people and their activities. IEEE Transactions on pattern analysis and machine intelligence 22(8):809–830
21. Harris C, Stephens M (1988) A combined corner and edge detector. In: Alvey vision conference, Citeseer, vol 15, p 50
22. Hasegawa T, Yamauchi Y, Ambai M, Yoshida Y, Fujiyoshi H (2014) Keypoint detection by cascaded fast. In: 2014 IEEE International Conference on Image Processing (ICIP), IEEE, pp 5676–5680
23. Hengstler S, Prashanth D, Fong S, Aghajan H (2007) Mesheye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In: Proceedings of the 6th international conference on Information processing in sensor networks, ACM, pp 360–369
24. Hsiao PY, Lu CL, Fu LC (2010) Multilayered image processing for multiscale harris corner detection in digital realization. IEEE Transactions on Industrial Electronics 57(5):1799–1805
25. Köhler T, Röchter F, Lindemann JP, Möller R (2009) Bio-inspired motion detection in an fpga-based smart camera module. Bioinspiration & biomimetics 4(1):015,008
26. Kraft M, Schmidt A, Kasinski AJ (2008) High-speed image feature detection using fpga implementation of fast algorithm. VISAPP (1) 8:174–9
27. Li N, Huo H, Zhao Ym, Chen X, Fang T (2013) A spatial clustering method with edge weighting for image segmentation. IEEE Geoscience and Remote Sensing Letters 10(5):1124–1128
28. Li Y, Chu W (1996) A new non-restoring square root algorithm and its vlsi implementations. In: Computer Design: VLSI in Computers and Processors, 1996. ICCD'96. Proceedings., 1996 IEEE International Conference on, IEEE, pp 538–544
29. Liu J, Wark T, Martin S, Corke P, D'Souza M (2011) Distributed object tracking with robot and disjoint camera networks. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on, IEEE, pp 380–383
30. Mozos ÓM, Gil A, Ballesta M, Reinoso O (2007) Interest point detectors for visual slam. In: Conference of the Spanish Association for Artificial Intelligence, Springer, pp 170–179
31. Norouznezhad E, Bigdeli A, Postula A, Lovell BC (2010) Object tracking on fpga-based smart cameras using local oriented energy and phase features. In: Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras, ACM, pp 33–40
32. Olson T, Brill F (1997) Moving object detection and event recognition algorithms for smart cameras. In: Proc. DARPA Image Understanding Workshop, vol 20, pp 205–208
33. Possa PR, Mahmoudi SA, Harb N, Valderrama C, Manneback P (2014) A multi-resolution fpga-based architecture for real-time edge and corner detection. IEEE Transactions on Computers 63(10):2376–2388
34. Qureshi F, Terzopoulos D (2008) Smart camera networks in virtual reality. Proceedings of the IEEE 96(10):1640–1656
35. Rosten E, Drummond T (2005) Fusing points and lines for high performance tracking. In: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, IEEE, vol 2, pp 1508–1515
36. Shi J, Tomasi C (1994) Good features to track. In: Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on, IEEE, pp 593–600
37. Sroba L, Ravas R, Grman J (2015) The influence of subpixel corner detection to determine the camera displacement. Procedia Engineering 100:834–840
38. Stock C, Mühlmann U, Chandraker MK, Pinz A (2002) Subpixel corner detection for tracking applications using cmos camera technology. Citeseer
39. Tanskanen P, Kolev K, Meier L, Camposeco F, Saurer O, Pollefeys M (2013) Live metric 3d reconstruction on mobile phones. In: Proceedings of the IEEE International Conference on Computer Vision, pp 65–72
40. Torr PH, Zisserman A (1999) Feature based methods for structure and motion estimation. In: International workshop on vision algorithms, Springer, pp 278–294
41. Ttofis C, Hadjitheophanous S, Georghiades AS, Theocharides T (2013) Edge-directed hardware architecture for real-time disparity map computation. IEEE Transactions on Computers 62(4):690–704
42. Turner LR (1966) Inverse of the vandermonde matrix w1th applications
43. Wu C, Aghajan H, Kleihorst R (2008) Real-time human posture reconstruction in wireless smart camera networks. In: Proceedings of the 7th international conference on Information processing in sensor networks, IEEE Computer Society, pp 321–331
44. Yang AY, Maji S, Christoudias CM, Darrell T, Malik J, Sastry SS (2011) Multiple-view object recognition in smart camera networks. In: Distributed Video Sensor Networks, Springer, pp 55–68
45. Yuan JY (1996) Numerical methods for generalized least squares problems. Journal of Computational and Applied Mathematics 66(1):571–584
46. Zhang G, Vela PA (2015) Good features to track for visual slam. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 1373–1382
47. Zhang H, et al (2007) Quantitative evaluation of feature extractors for visual slam. In: Computer and Robot Vision, 2007. CRV'07. Fourth Canadian Conference on, IEEE, pp 157–164

48. Zhu Q, Wu B, Wan N (2007) A sub-pixel location method for interest points by means of the harris interest strength. The Photogrammetric Record 22(120):321–335
49. Zhu W, Ma C, Xia L, Li X (2009) A fast and accurate algorithm for chessboard corner detection. In: Image and Signal Processing, 2009. CISP'09. 2nd International Congress on, IEEE, pp 1–5
50. Zitnick CL, Dollár P (2014) Edge boxes: Locating object proposals from edges. In: European Conference on Computer Vision, Springer, pp 391–405
51. Zivkovic Z (2010) Wireless smart camera network for real-time human 3d pose reconstruction. Computer Vision and Image Understanding 114(11):1215–1222

Author Biographies

Abiel Aguilar-González obtained his B.Eng. in Mechatronics in June 2012, Universidad Politécnica de Chiapas (UPCH), Tuxtla Gutiérrez, Mexico. In June 2015 he obtained his M.Sc. in Mechatronics Engineering with highest honors, Instituto Tecnológico de Tuxtla Gutiérrez (ITTG), Tuxtla Gutiérrez, Mexico. He is currently pursuing his Ph.D. in Computer Science at the reconfigurable computing laboratory of the Instituto Nacional de Astrofísica Óptica y Electrónica (INAOE), Cholula, Mexico. His research interests are real-time image processing, real-time FPGA-based system design and fuzzy logic applications.

Miguel Arias-Estrada obtained his B.Eng. in Communications and Electronics, and his M.Eng in Digital Systems at the FIMEE (University of Guanajuato) in Salamanca, Gto. in 1990 and 1992 respectively. In 1998, he obtained his Ph.D. degree at the Computer Vision and Systems Laboratory of Universit Laval (Quebec city, Canada). He was a professor-researcher at the Computer and Systems Laboratory at Laval University where he worked on the development of a Smart Vision Camera. Since 1998 he is with the Computer Science department of INAOE (National Institute of Astrophysics, Optics and Electronics, Puebla, Mexico) where he continues his research on FPGA architectures for computer vision. His interests are Computer Vision, FPGA and GPU algorithm acceleration for 3D and machine vision.

François Berry received his Doctoral degrees and the Habilitation to conduct researches in Electrical Engineering from the University of Blaise Pascal in 1999 and 2011, respectively. His PhD was on visual servoing and robotics and was undertaken at Pascal Institute in Clermont -Ferrand. Since September 1999, he is currently (Associate Professor) at the University of Blaise Pascal and is member of the Perception System and Robotics group (within GRAVIR, Pascal Institute- CNRS). He is researching smart cameras, active vision, embedded vision systems and hardware/software co-design algorithms. He is in charge of a Masters in Microelectronics and in head of DREAM Research on Embedded Architecture and Multisensor) group. He has authored and coauthored more than 55 papers for journals, conferences and workshops. He has also led several research projects (Robea, ANR, Euripides) and has served as a reviewer and a program committee member. He has been co-founder of the Workshop on Architecture of Smart Camera (WASC) and Scabot.