Dense Feature Matching Core for FPGA-based Smart Cameras

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

Dense Feature Matching Core for FPGA-based Smart Cameras

Abiel-Aguilar-González[1,2], Miguel Arias-Estrada[1], François Berry[2]
1. Instituto Nacional de Astrofíosica, Óptica y Electrónica (INAOE),Tonantzintla, Mexico
2. Université Clermont Auvergne (UCA), Institut Pascal, Clermont-Ferrand, France
Contact: abiel@inaoep.mx

ABSTRACT

Smart cameras are image/video acquisition devices that integrate image processing algorithms close to the image sensor, so they can deliver high-level information to a host computer or high-level decision process. In this context, a central issue is the implementation of complex and computationally intensive computer vision algorithms inside the camera fabric. For low-level processing, FPGA devices are excellent candidates because they support data parallelism with high data throughput. One computer vision algorithm highly promising for FPGA-based smart cameras is feature matching. Unfortunately, most previous feature matching formulations have inefficient FPGA implementations or deliver relatively poor information about the observed scene. In this work, we introduce a new feature-matching algorithm that aims for dense feature matching and at the same time straightforward FPGA implementation. We propose a new mathematical formulation that addressed the feature matching task as a feature tracking problem. We demonstrate that our algorithmic formulation delivers robust feature matching with low mathematical complexity and obtains accuracy superior to previous algorithmic formulations. An FPGA architecture is lay down and, hardware acceleration strategies are discussed. Finally, we applied our feature matching algorithm in a monocular-SLAM system. We show that our algorithmic formulation provides promising results under real world applications.

Keywords

Feature matching, Feature tracking, Smart camera; FPGA

1. INTRODUCTION

Smart cameras are image/video acquisition devices with self-contained image processing algorithms that simplify the formulation of a particular application. i.e., algorithms for video surveillance could detect and track pedestrians, but for a robotic application, algorithms could be edge and feature detection. In recent years, 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, so in current embedded vision applications, smart cameras represent a promising on-board solution under different application domains: motion detection, object detection/tracking, inspection and surveillance, human behavior recognition [7, 9], etc. In any case, flexibility of application domain relies on the large variety of image processing algorithms that can be implemented inside the camera. One task highly used by computer vision applications is feature matching between different camera views. In computer vision, feature matching aims for pixel/point correspondences across different viewpoints from the same scene/object (**Fig.** 1) and it is the basis of several computer vision applications such as, augmented reality, object recognition [2], etc. The most common formulation consists in detecting a set of feature points and associate each point with a visual descriptor. Once feature points and their descriptors have been extracted from at least two images, it is possible to match features across the images. In this context, feature-tracking seems to be a simple point matching problem, nevertheless, in practice it is a complex task since matching performance depends on the feature extractor/visual descriptor properties. Specific detectors and descriptors, appropriate for the input images content have to be used in specific applications. i.e., if input images are a microscopic view of bacteria or cells, a blob detector should be used. On the other hand, if the images are a city view, a corner detector is more suitable to find building structures. In addition, if input images have high degradation (rotation, orientation or scale changes), complex and intensive visual descriptors considering the image degradation are required in order to guarantee stability.

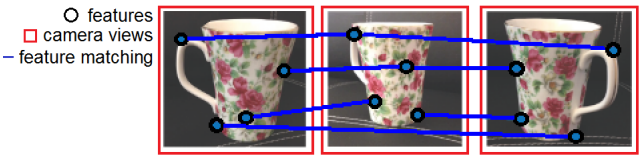


Figure 1: The feature matching problem: visual features (circles) have to be matched (lines) across different viewpoints from the same scene/object (squares).

2. RELATED WORK

In current computer vision systems, several applications use feature matching as keystone of their mathematical formulations, so a smart camera that contains feature matching in their self-contained algorithms is highly desirable. In recent work, there are several approaches that aim for an embedded feature matching core; several FPGA architectures have been developed and several solutions have been proposed. [24, 17]. In [8] an embedded system architecture for feature detection and matching was presented. The proposed FPGA architecture implements the FAST [15] (Features

ACM ISBN .
DOI:

from Accelerated Segment Test) feature detector and the BRIEF [5] (Binary Robust Independent Elementary Features) feature descriptor in a customizable FPGA block. The developed blocks were designed to use hardware interfaces based on the AMBA AXI4 interface protocol and were connected using a DMA (Direct Memory Access) architecture. The proposed architecture computes feature matching over two consecutive HD frames coming from an external memory at 48 frames per second. In [26] a FPGA architecture of SIFT (Scale Invariant Feature Transform) visual descriptor associated to an image matching algorithm was presented. For an efficient FPGA-SIFT image matching implementation (in terms of speed and hardware resources usage), the original SIFT algorithm was optimized as follows: 1) Upsampling operations were replaced with downsampling, in order to avoid interpolation operations. 2) Only four scales with two octaves were used. 3) Dimension of the visual descriptor was reduced to 72 instead of 128 in the original SIFT formulation. This implementation is able to detect and match features in 640×480 image resolution at 33 frames per second. More recently, Weberruss [25] have proposed a FPGA architecture for ORB [16] (Oriented FAST) descriptor associated to a feature matching algorithm. An "harris corner" detection [10] was the feature extractor and ORB visual descriptors were computed at each "corner". Finally, the previous features (stored in a 2D Shift Register) and the current features were matched using the hamming distances as discrimination metric. In 2017, Vourvoulakis [23] presented an FPGA-SIFT architecture for feature matching. In order to achieve high hardware parallelism, procedures of SIFT detection and description were reformulated. At every clock cycle, the current pixel in the pipeline is tested and if it is a SIFT feature, its descriptor is extracted. Furthermore, every detected feature in the current frame is matched with one among the stored features of the previous frame, using a moving window, without breaking the "pixel pipeline". False matches are rejected using RANSAC (Random Sample Consensus) algorithm. The architecture was implemented on Cyclone IV. Maximum supported clock frequency was set as 25 MHz and the architecture was capable to process 81 frames per second, considering 640×480 image resolution.

In most of cases, previous FPGA-based feature matching formulations, [24, 17, 8, 15, 5, 26, 25, 16, 23] provide relatively good performance under real world scenarios. Unfortunately, in several applications and in particular smart cameras applications, these algorithms are not compliant due to their relatively high hardware requirements and their algorithmic formulation. We can mention three important limitations affecting the current feature matching algorithms:

1. Low performance for embedded applications: nowadays computers can process several feature matching algorithms in real-time. Unfortunately, in embedded applications such as, smart cameras, 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 there are hard challenges due to previous visual descriptors (SIFT, BIERF, ORB) and matching techniques were designed for software implementation, and often, there are several iterative operations that could not be parallelized. As result, most previous FPGA architectures have high hardware requirements and relatively low processing rate.

2. Sparse matching: in order to maintaining high discrimination between descriptors only features with high thresholding response are matching (since it is assumed that these features have to be associated with high responsive visual descriptor that has low probability to be similar in other features). In practice, this assumption ensures consistency in the matching process, however, there is an important limitation because only a few image points are matched, then, scene/object information are available only at certain sparse points of the image, as shown in **Fig.** 2.

3. Outliers: In certain cases, the image ambiguities around features (color/texture repeatability, occlusion, etc.) generate similar visual descriptors for two or more different features, in such scenario the matching techniques deliver wrong results that can affect the global performance of several computer vision applications (camera calibration, structure from motion, visual odometry, etc.), see **Fig.** 2. To solve this problem, statistically robust methods like RANSAC have to be applied as outlier filter. In this case, statistical methods remove wrong matches, but they increment the matching cost.

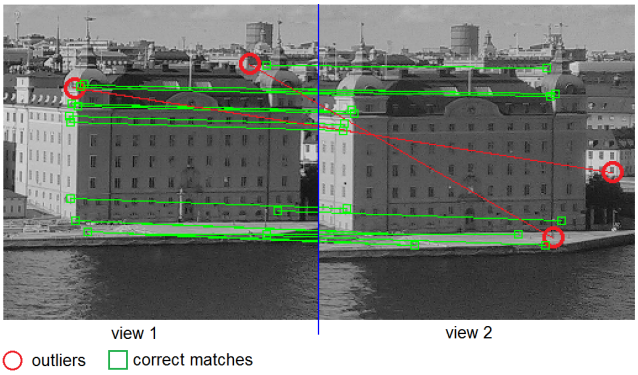


Figure 2: Feature matching algorithms limitations: most previous formulations work with few image points. Therefore, scene information is limited to a certain sparse points in the image. On the other hand, most previous work deliver outliers that effect performance in real world applications. (Figure modified from [25])

3. THE PROPOSED ALGORITHM

Most previous FPGA-based feature matching formulations provide relatively good performance under real world scenarios, however, there are several important limitations (low performance for embedded applications, sparse matching and outliers) that affect performance. In this work, we assume that a more efficient solution consists in addressing the feature matching task as a feature tracking problem. In this way, we consider that a feature tracking approach will provide more data parallelism than previous formulations based on SIFT/ORB visual descriptors. In current state of the art, there are some FPGA architectures for feature tracking [22, 20]. Unfortunately, most previous work addressed the problem via the KLT (Kanade-Lucas-Tomasi) tracking algorithm, that is highly exhaustive and has high hardware requirements for the case of FPGA implementation.

3.1 Feature matching and Feature tracking.

The basis of feature matching is to extract visual features from two or more different viewpoints from the same scene/object and then, match these visual features by comparing visual descriptors computed around each feature, as shown in **Fig.** 3. On the other hand, feature tracking consists in extracting visual features from an image and then, try to find the same features back in a similar image (commonly the next frame from a video sequence), as illustrated in **Fig.** 4.

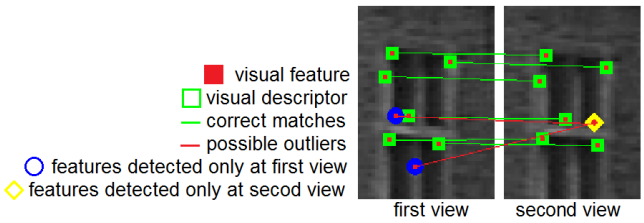


Figure 3: The feature matching problem: for two or more different viewpoints visual features and visual descriptors around each feature are computed. Then, feature matching is computed based on visual descriptors comparison. Since some features could appear only at one unique view, high discriminant visual descriptors are desirable in order to avoid outliers.

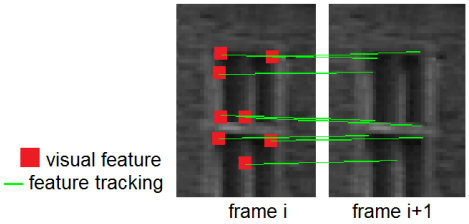


Figure 4: The feature tracking problem: considering two consecutive frames from a video sequence, visual features are extracted within the first frame. Then, it is tried to find the same features back in the next frame. Since there are several image ambiguities around features (color/texture repeatability, occlusion, etc.), complex and intensive pixel similarity metrics are required in order to guarantee accurate tracking.

In our case, we propose a new feature tracking formulation that can be extended for feature matching. Our contributions are twofold: first, we propose a tracking/matching framework that improves most of the current limitations of the previous feature matching algorithms and that is fully compliant with FPGA architectures. Second, we lay down an FPGA architecture, suitable for smart cameras implementation. In **Fig.** 6 an overview of our algorithmic formulation is shown. In general, our algorithm first, stores two consecutive frames from an input video sequence. Visual features are extracted from the both frames. In addition, curl of the intensity gradient is computed over both frames. In this case, curl of the intensity gradient aims to remove image ambiguities around features (color/texture repeatability, pixel similarity, etc.). Then, a fully parallelized feature tracking algorithm computes preliminary matches between features in first frame and pixel points in second frame. For that, curls of the intensity gradient are considered as input since it is assumed that it guarantees consistent tracking around features. Finally, considering feature points in second frame, a feature matching steep refines the tracking result by comparing tracked points in second frame with visual feature coordinates computed at the same frame.

3.2 Image storage

Considering that in most cases the image sensor provides data as a stream, a storage is required to get two consecutive frames at the same t time. More information/details about the storage architecture will be presented in **Section** 4.1. For mathematical formulation, first frame (frame at t time) is noted $I_1(x, y)$ while the second frame (frame at $t + 1$ time) is $I_2(x, y)$.

3.3 Feature extractor

In this work, the Shi-Tomasi feature extractor is used. it provides a good trade-off between accuracy/robustness, speed processing and hardware requirements. This extractor is based on spatial gradients such as:

$$\begin{aligned} A(x,y) &= \frac{\partial I}{\partial x} \cdot \frac{\partial I}{\partial x} \\ B(x,y) &= \frac{\partial I}{\partial y} \cdot \frac{\partial I}{\partial y} \\ C(x,y) &= \frac{\partial I}{\partial x} \cdot \frac{\partial I}{\partial y} \end{aligned}$$

A gaussian filtering is applied over the A, B, C matrices in order to reduce noise and to remove fine-scale structures that affect the performance of the corner response. Smoothed matrices are defined by A', B', C'. Original Shi-Tomasi corner metric **Eq.** 1, provides a high response value for corners and low response otherwise, as illustrated in **Fig.** 5b.

$$D(x,y) = (A'(x,y)+B'(x,y)) - \sqrt{(A'(x,y) - B'(x,y))^2 + 4C'(x,y)^2} \quad (1)$$

In order to determine if a pixel P is a corner or not, maximum value of the corner response is retained. However, many pixels around each corner are detected in spite of filtering with a threshold α. These pixels are false feature candidates and are difficult to match/track. A way to remove these false feature candidates consists in applying a non-maxima suppression step. An appropriate FPGA-based non-maxima suppression step could be defined as follows: Considering $D(x, y)$ as the corner response image and $\Omega(x, y)$ as fixed neighborhood size of 3×3 around $D(x, y)$, a "good features" is computed as :

$$\beta(x,y) = max \left[\Omega(x,y) * \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \right]$$

Finally, a thresholding (α) have to be applied over $D(x, y)$ in order to select the "good features/corners", see **Eq.** 2.

$$corners(x,y) = \begin{cases} 1 & if\ \beta(x,y) > \alpha \\ 0 & if\ otherwise \end{cases} \quad (2)$$

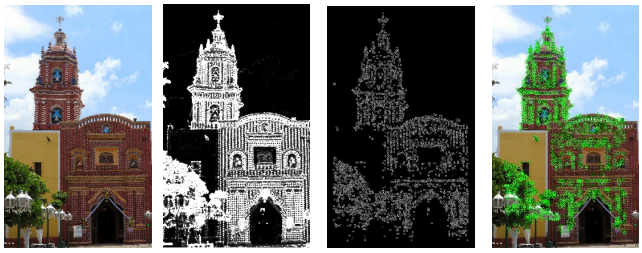


Figure 5: The Shi-Tomasi feature extractor: (a) Input image. (b) Corner metric response, **Eq.** 1. (c) Corner response after our non-maxima suppression step, $\beta(x, y)$. (d) Output image considering $\alpha = 0.08$, **Eq.** 2

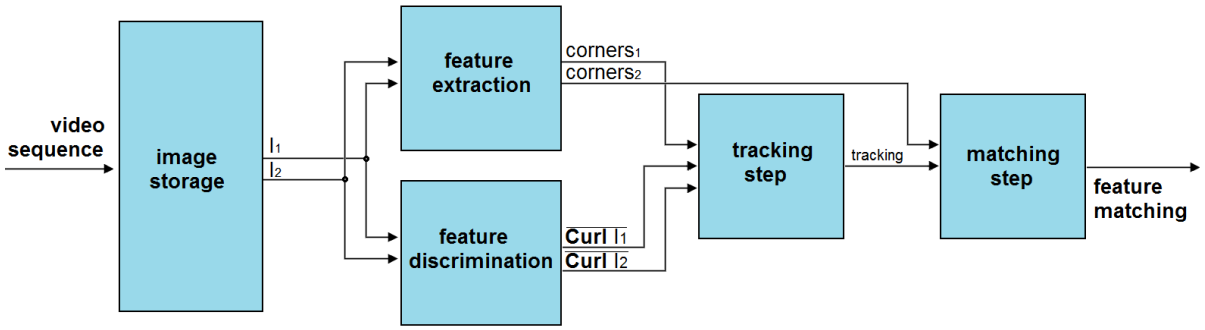


Figure 6: Block diagram of the proposed algorithm.

3.4 Improvement feature discrimination

Previous work [19] demonstrated that simple pixel similarity metrics such as SAD (Sum of Absolute Differences), Hamming distances or NCC (Normalized Cross-Correlation) deliver poor results over real world scenarios. This is due to several image ambiguities around features. i.e., due to color/texture repetition, different features could have low difference between similarity metrics (close to zero). To solve these problems, previous feature tracking formulations [22, 20] used more complex approaches such as Eigenvalues of the Gradient matrix or Jacobian matrix as similarity metric. Unfortunately, they require high hardware resources for FPGA implementation.

In this work, we propose to improve the feature discrimination by using the curl of the intensity gradient $\frac{dI(x,y)}{d\mathbf{x}}$ in each point. Let curl as a vector operator that describes the infinitesimal rotation, then, at every point the curl of that point is represented by a vector where attributes (length and direction) characterize the rotation at that point. In our case, we use only the norm of **Curl** $I(x,y)$ given by:

$$\mathbf{Curl}I(x,y) = \nabla \times \frac{dI(x,y)}{dx} \tag{3}$$

where ∇ is the **Del** operator.

$$\overline{\mathbf{Curl}I}(x,y) = \sqrt{\left(\frac{\partial}{\partial y}\frac{\partial I}{\partial x} - \frac{\partial}{\partial x}\frac{\partial I}{\partial y}\right)^2} = \left|\frac{\partial}{\partial y}\frac{\partial I}{\partial x} - \frac{\partial}{\partial x}\frac{\partial I}{\partial y}\right| \tag{4}$$

3.5 Feature tracking

Tracking process assumes that features displacements between frames is such as it exists an overlap on two successive "search regions". A search region is defined as a patch around a feature to track. This process is illustrated in **Fig.** 7. Considering that between I_1 and I_2, the illumination is stable, a similarity-based metric provides a good accuracy. This similarity is calculated by a SAD (Sum of Absolute Difference). This process is defined in **Eq.** 5, where r is the search region size, $\mathbf{Curl}I_1(x,y), \mathbf{Curl}I_2(x,y)$ are the norm curl images on two consecutive frames from a video sequence (I_1, I_2 respectively), x, y are the spatial coordinates of features extracted in I_1 and, a, b are the spatial coordinates for all points within the search region constructed in I_2. Thus, considering a video sequence, first, the feature extractor (**Section** 3.3) obtains g feature points from $I_1(x,y)$, defined as $corners_1(x,y)$. Let $x_1(g), y_1(g)$ as the x, y are the spatial position for all extracted features in I_1. Then, tracking of extracted features is performed by applying **Eq.** 6 and 7; where $x_2(h),\ y_2(h)$ are the tracked positions in I_2.

$$\begin{aligned}\mathrm{SAD}(a,b) = \sum_{u=-r}^{u=r}\sum_{v=-r}^{v=r} &|\overline{\mathbf{Curl}I_1}(x+u, y+v)| \\ &-|\overline{\mathbf{Curl}I_2} = (x+u+a, y+v+b)|\end{aligned} \tag{5}$$

$$x_2(h) = \sum_{h=1}^{h=g} x_1(h) + min_a SAD(a,b) \tag{6}$$

$$y_2(h) = \sum_{h=1}^{h=g} y_1(h) + min_b SAD(a,b) \tag{7}$$

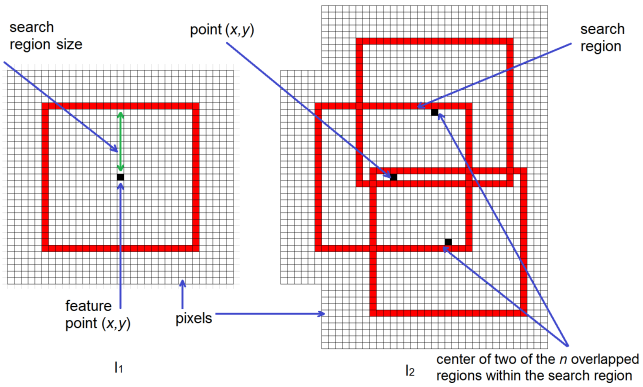


Figure 7: The proposed feature-tracking algorithm formulation. For each feature point, n overlapped regions are constructed in I_2, n region centers are all points within the search region created in I_1. n region center that minimizes SAD is the tracked position of the feature point (x,y) in I_2.

3.6 Feature matching

Our feature tracking algorithm do not consider the occlusion problem i.e., it tracks any feature extracted from I_1, then, some outliers could be present because some features extracted in reference image I_1 could have occlusion at I_2. To solve this problem, a matching technique is used as outlier filtering. Considering $x_2(h)$, $y_2(h)$ as reference spatial coordinates for the feature matching (**Eq.** 6 and 7) and given $corners_2(x,y)$ the feature extraction from I_2 (**Eq.** 2), a pixel tracking is correct only if there is one unique feature in I_2 that is located in the region that surrounds the previously computed tracking localization (see **Eq.** 8 - 10).

$$filter_h(x,y) = \sum_{u=-1}^{u=1} \sum_{v=-1}^{v=1} corners_2(x+u, y+v) \quad (8)$$

$$x_f(h) = \begin{cases} x_2(h) & if\ filter_h(x,y) == 1 \\ 0 & otherwise \end{cases} \quad (9)$$

$$y_f(h) = \begin{cases} y_2(h) & if\ filter_h(x,y) == 1 \\ 0 & otherwise \end{cases} \quad (10)$$

4. FPGA ARCHITECTURE

In **Fig.** 8, an overview of the FPGA architecture for the feature-matching algorithm is presented. The architecture is centered on an FPGA implementation where all recursive/parallelizable algorithms are accelerated in the FPGA fabric. In general, the basis of the proposed architecture is the frame storage unit. In this block, frames captured by the imager are feed to/from an external SDRAM memory using a DMA. Two consecutive frames are read out into the buffers used to hold local sections of the frames that are being tracked and allow for local parallel access that facilitates parallel processing.

4.1 Frame storage unit

Images from the image sensor are stored in an external SDRAM that holds at least 2 frames from the sequence, and later the SDRAM is read by the FPGA to cache parts of the frames into buffers. The **frame storage** unit is responsible for data transfers in segments of the image (usually several rows of pixels) to/from the SDRAM. The core of the FPGA architecture are the buffers attached to the local processors that can hold temporarily as cache, for image sections from two frames, and that can deliver parallel data to the processors. For the SDRAM controller, both Xilinx and Altera have IPs for this proposes. For the buffers, we use a circular buffer schema in which input data from the previous N rows can be stored using memory buffers till the moment when a n×n neighborhood is scanned along subsequent rows. This approach has high hardware reutilization and high flexibility for computer vision applications. For more details, see [1].

4.2 Feature extraction unit

Fig. 9a gives an overview of the **feature extractor** unit. First, the architecture computes the vertical/horizontal gradients, $\frac{\partial I_1}{\partial x}$, $\frac{\partial I_1}{\partial y}$, respectively. Then it computes the $A(x,y), B(x,y), C(x,y)$ variables. After that, a buffer delivers parallel data for the Gaussian filtering. Then, the reconfigurable convolution units (see [1]) compute the smoothing operation. Finally, the FPGA architecture computes the corner response metric and the non-maxima suppression step. In order to simplify the square root operation implementation in the feature extraction step, we adapted the architecture developed by Yamin Li and Wanming Chu [11]. This architecture uses a shift register mechanism and compares the more significant/less significant bits to compute square root from **Eq.** 1 with relatively high accuracy and low hardware resources.

4.3 Feature discrimination unit

Fig. 9b, an overview of the **feature discrimination** unit is shown. It resuses the gradient computation carried out in the **feature extraction** unit. Then, curl of the intensity gradient is computed as illustrated in **Section** 3.4. Two logical processes compute the curl of the intensity gradient for I_1 and I_2 in parallel.

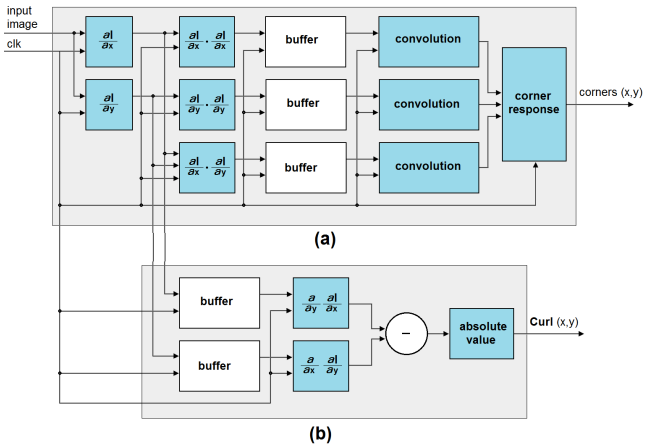


Figure 9: (a) FPGA architecture for the feature extractor unit. (b) FPGA architecture for the feature discrimination unit.

4.4 Feature tracking unit

For the **feature tracking** unit, we consider that the tracking problem can be seen as a generalization of the dense stereo matching problem. i.e., stereo matching algorithms track (searching on the horizontal axis around points in the reference image), all points/pixels within a stereo pair. Feature tracking aims to track features points between two consecutive frames from a video sequence (searching around spatial coordinates of the features in the reference frame). Then, it is possible to adapt previous stereo matching FPGA architectures to fulfill with our application domain. In this work, we adapted the FPGA architecture presented in [14], which has low hardware requirements and high parallelism. In **Fig.** 10, the developed architecture is shown. Considering that feature points for I_1 are known, these are obtained by the feature extraction unit. Then, the search region modules (see **Fig.** 7), construct n search regions, where search regions are constructed via logical pointers under the input buffer for I_2. For each feature point in the reference image (I_1), search region centers correspond to all patches within the search region on frame I_2. Once the search regions are constructed, similarity SAD modules compute the correlation response (applying the sum of absolute differences as similarity metric response). i.e., it compares all search regions with the reference region. Finally, a multiplexer tree can determine the a, b indices that minimize the correlation function, and therefore, the tentative position in I_2 of the feature points extracted in the reference image (I_1).

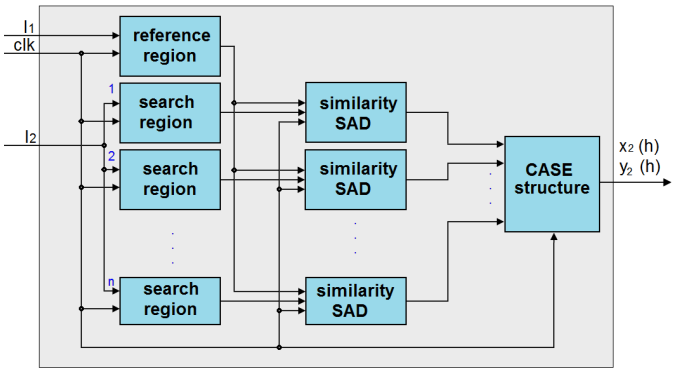


Figure 10: FPGA architecture for the feature tracking unit

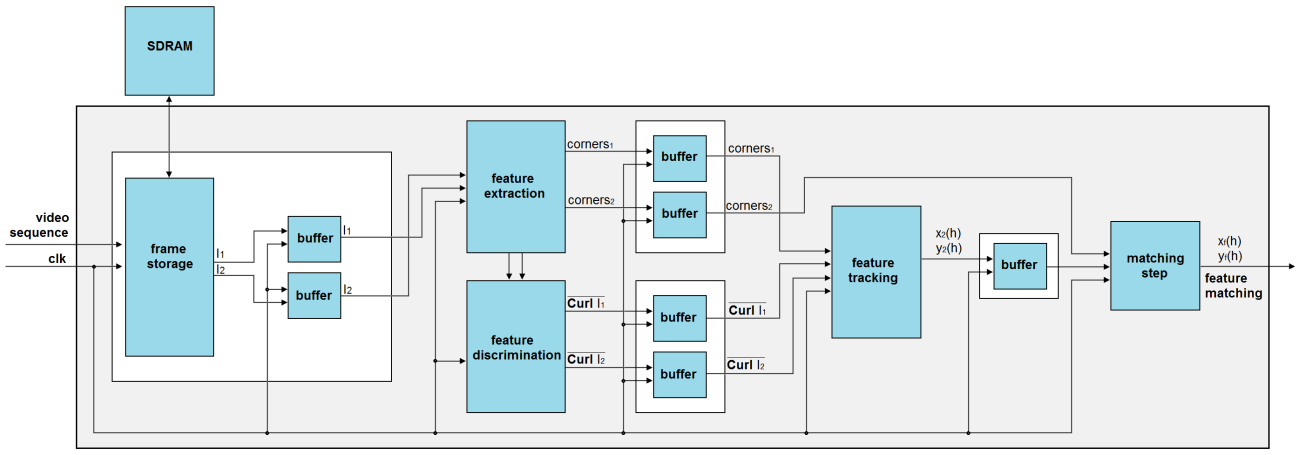


Figure 8: FPGA architecture for dense/robust feature matching

4.5 Feature matching unit

The feature matching unit consists in a unique module that compares the feature tracking results with visual features extracted in I_2. Then, the feature matching $x_f\{h\}$, $y_f\{h\}$ (final result) is the result obtained after comparisons between tracking positions $x_2\{h\}$, $y_2\{h\}$ and visual features $corners_2$ in frame I_2.

5. RESULTS

In order to validate our mathematical formulation, we implemented our feature-matching algorithm in a MatLab R2016b code that captures video sequences from a web camera. Then, feature points (corners) are tracked along the video sequence. In all experiments, feature points are obtained by applying the algorithm presented in **Section** 3.3. Video sequences of 1920×1080 pixel resolution and 800 frames were used. In **Fig.** 11, results by applying our algorithm over an outdoor scenario are shown.

Figure 11: Feature matching under outdoor scenarios. Our mathematical formulation reach: accurate matching (without outliers) and dense matching (more than 14000 pixels are matched).

5.1 Performance for dataset scenes

For performance comparisons we compared our feature matching algorithm with previous feature matching/tracking algorithms. For feature tracking, we applied the KLT, KL and Mean-shift algorithms. For feature matching we used three classic feature matching frameworks, based on ORB, SIFT and SUFR visual descriptors, respectively. We we evaluate the algorithms with several video sequences, all videos were obtained from [18]. Two performance tests were conducted. In **Table** 1, accuracy comparisons are shown.

To validate the accuracy in numerical form, RMS error is computed as: $\epsilon = \sqrt{x^2 + y^2}$; where x is the error in x axis, defined as the average difference between the ground truth x position in each frame of the video sequence and the position in same frame computed by the testing algorithm (visual features ground truth were computed using the camera localization ground truth provided by the dataset). y is the error in y axis, it is computed similar to x. In all cases, our algorithm outperforms the KL and the MS tracking algorithms, and outperforms feature matching algorithms based on visual descriptors like ORB, SIFT and SURF. This is because the KL algorithm does not consider the occlusion problem. In addition, the KL algorithm uses simple similarity metrics, that introduces erroneous measurements under image ambiguities. On the other hand, MS algorithm was formulated for object-tracking in dynamic scenes, therefore, performance under rigid scenes is low. For algorithms that use robust visual descriptors (ORB, SIFT, SURF), occlusions, ambiguities and perspective changes between frames introduce outliers, therefore, accuracy is lower than tracking approaches. Although KLT algorithm outperforms our algorithmic formulation, KLT is highly exhaustive, processing speed is low, and implementation in real-time/embedded applications is highly limited.

For density comparisons: traditional feature matching algorithms extract and match visual features via visual descriptor comparisons. Unfortunately, in all cases, the maximal number of features that can be matched varies between 0.5% and 1.0% of all pixels in the image, depending on the selected descriptor and its particular configuration. In practice, computer vision systems work with configurations that allow extracting near 1% of the pixels from an image. This is illustrated in **Table** 2, where feature matching based on visual descriptors reach less than 200 matches per frame. This limits the real-world applications performance since less than 1% of the image points are available, thus, the visual environmental understanding, high-level descriptors application and objects/structures recognition in the scene could have low stability under real-world scenarios. Even the most current and popular feature matching approaches, for example such based on the ORB descriptor, are limited to sparse matching. On the other hand, tracking algorithms such as KLT and KL deliver high density compared with matching approaches, as shown in **Table** 2. Both, KL and KLT can track near 2% of all points in the scene (×44 more than the matching algorithms). In the case of our feature matching algorithm, it can track/match near 5% of the points within the scene (×2 more than previous feature tracking approaches and ×85 more than previous

feature matching approaches). This was achieved by applying low threshold values for the feature extraction process and, due to vectors transformations whose formulation guarantee consistence between low responsive features. For KL and KLT these low thresholds generate outliers that affect the tracking accuracy. Since the MS algorithm was formulated for object-tracking, density is low since this algorithm only track shape features (that are a few of pixels compared with all the points in the analyzed scenes).

5.2 Performance for real world applications

We implement our feature-matching algorithm in a monocular-SLAM system. We applied our algorithm to obtain point correspondences across the video sequence. In this case, point correspondences allow fundamental matrix estimation. Finally, using the fundamental matrix, is it possible to estimate a 3D reconstruction and camera pose along the video sequence. In this case our feature matching algorithm is capable to increase the point cloud density, as shown in **Fig.** 12. We consider that this could be highly useful under several computer vision applications that use feature matching in their mathematical formulations (SLAM, SfM, 3D reconstruction), since more information ($\times 85$ more than previous feature matching approaches and $\times 2$ more than previous feature tracking formulations) are available. Thus, visual environmental understanding, high-level descriptors application and objects/structures recognition performance could be improved. For our FPGA architecture, we consider that our architectural formulation could be implemented within a smart camera fabric. In this scenario, our feature matching algorithm could be an important contribution for smart cameras, this because several computer vision applications uses point correspondences between frames/camera views as keystone of their mathematical formulation. For more details about the performance of the proposed algorithm see the material adjoint to this manuscript, all material can found from https://1drv.ms/f/s!AqkoNeNXKa6ijF10rKB5LFp30_CP.

6. CONCLUSIONS

In this work, we have introduced a new feature matching algorithm that deliver accurate/dense feature matching under indoor/outdoor scenarios. We proposed a new mathematical formulation that addressed the feature matching task as a feature tracking problem, and we have used the curl of the intensity gradient as feature discrimination technique. An FPGA architecture was lay down and, hardware acceleration strategies were discussed. Since several computer vision applications use feature matching a keystone of their mathematical formulations, we consider that feature matching within a smart camera fabric could be promising under current computer vision applications. We have applied our feature matching algorithm in a monocular-SLAM system. We have shown that our algorithmic formulation improves the performance under SLAM applications. As work in progress we are implementing our feature matching algorithm inside the DREAMCAM [4], a robust/flexible smart camera.

7. ACKNOWLEDGMENTS

This work has been sponsored by the French government research program "Investissements d'avenir" through the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01), by the European Union through the program Regional competitiveness and employment 2007-2013 (ERDF Auvergne region), and by the Auvergne region. This work has been sponsored by the National Council for Science and Technology (CONACyT), Mexico, through the scholarship No. 567804.

8. REFERENCES

[1] A. Aguilar-González, M. Arias-Estrada, M. Pérez-Patricio, and J. Camas-Anzueto. An fpga 2d-convolution unit based on the caph language. *Journal of Real-Time Image Processing*, pages 1–15, 2015.

[2] J. D. Anderson, D.-J. Lee, B. Edwards, J. Archibald, and C. R. Greco. Real-time feature tracking on an embedded vision sensor for small vision-guided unmanned vehicles. In *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 55–60. IEEE, 2007.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[4] M. Birem and F. Berry. Dreamcam: A modular fpga-based smart camera architecture. *Journal of Systems Architecture*, 60(6):519–527, 2014.

[5] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[6] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 2, pages 142–149. IEEE, 2000.

[7] M. Fularz, M. Kraft, A. Schmidt, and A. Kasiński. The architecture of an embedded smart camera for intelligent inspection and surveillance. In *Progress in Automation, Robotics and Measuring Techniques*, pages 43–52. Springer, 2015.

[8] M. Fularz, M. Kraft, A. Schmidt, and A. Kasiński. A high-performance fpga-based image feature detector and matcher based on the fast and brief algorithms. *International Journal of Advanced Robotic Systems*, 12(10):141, 2015.

[9] I. Haritaoglu, D. Hanrwood, and L. Davis. Real-time surveillance of people and their activities. *IEEE Trans. on Pattern Analysis and Machine Intlligence*, 22(8):809–830, 2010.

[10] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[11] Y. Li and W. Chu. 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*, pages 538–544. IEEE, 1996.

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[13] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[14] M. Pérez-Patricio, A. Aguilar-González, M. Arias-Estrada, H.-R. Hernandez-de Leon, J.-L. Camas-Anzueto, and J. de Jesús Osuna-Coutiño. An fpga stereo matching unit based on fuzzy logic. *Microprocessors and Microsystems*, 42:87–99, 2016.

[15] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Computer vision–ECCV 2006*, pages 430–443, 2006.

[16] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[17] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. Gpu-based video feature tracking and matching. In *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, volume 278, page 4321, 2006.

[18] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[19] O. Suominen. Transform-based methods for stereo matching and dense depth estimation. 2012.

[20] B. Tippetts, S. Fowers, K. Lillywhite, D.-J. Lee, and J. Archibald. Fpga implementation of a feature detection and tracking algorithm for real-time applications. *Advances in Visual Computing*, pages 682–691, 2007.

[21] C. Tomasi and T. Kanade. Detection and tracking of point features. 1991.

[22] M. Tomasi, S. Pundlik, and G. Luo. Fpga–dsp co-processing for feature tracking in smart video sensors. *Journal of Real-Time Image Processing*, 11(4):751–767, 2016.

[23] J. Vourvoulakis, J. Kalomiros, and J. Lygouras. Fpga accelerator for real-time sift matching with ransac support. *Microprocessors and Microsystems*, 49:105–116, 2017.

[24] J. Wang, S. Zhong, W. Xu, W. Zhang, and Z. Cao. A fpga-based architecture for real-time image matching. In *Eighth International Symposium on Multispectral Image Processing and Pattern Recognition*, pages 892003–892003. International Society for Optics and Photonics, 2013.

[25] J. Weberruss, L. Kleeman, and T. Drummond. Orb feature extraction and matching in hardware. In *Proceedings of the Australasian Conference on Robotics and Automation, the Australian National University, Canberra, Australia*, pages 2–4, 2015.

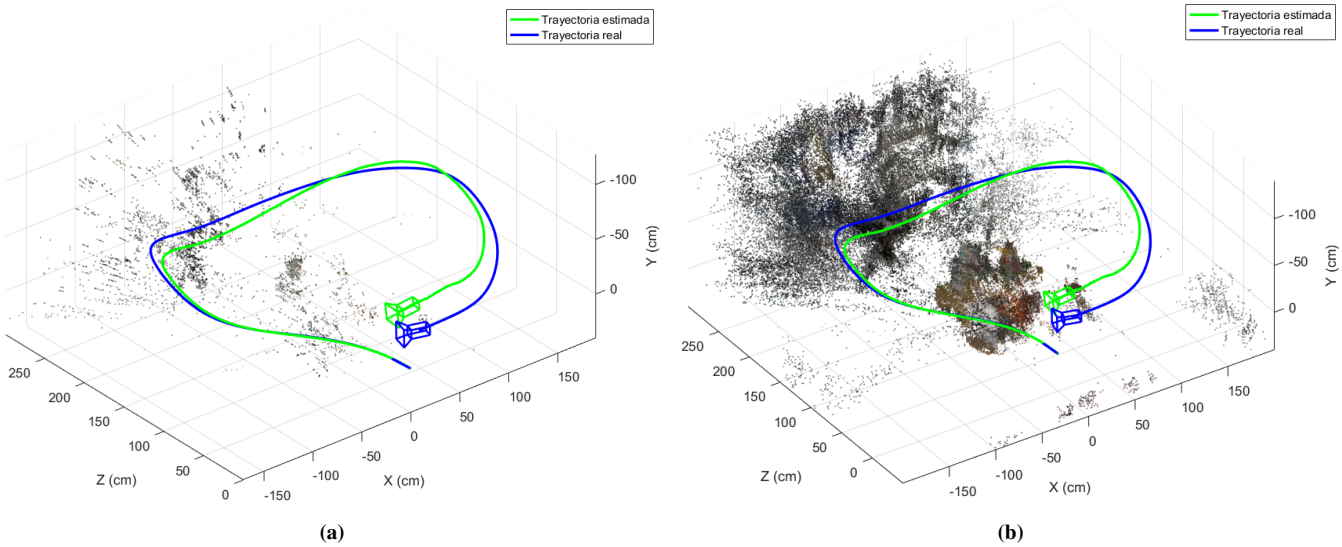
[26] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao, and W. Feng. An architecture of optimised sift feature detection for an fpga implementation of an image matcher. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 30–37. IEEE, 2009.

Table 1: The proposed algorithm compared with previous feature matching/tracking algorithms (accuracy). Error is measured in pixels

Dataset	**SIFT**[12]	**SURF**[3]	**ORB**[25]	**KLT**[21]	**KL**[13]	**MS**[6]	**Proposed**
fr1/room	67.22	79.38	76.24	0.21	4.81	9.62	1.87
fr2/desk	69.83	81.12	73.63	0.45	4.9	9.81	1.55
fr1/plant	59.29	77.74	75.24	0.39	4.12	8.25	1.7
fr1/teddy	75.38	83.53	76.73	0.47	4.91	9.82	1.71

Table 2: The proposed algorithm compared with previous feature matching/tracking algorithms (density). Density is measures as the feature matches number per frame

Dataset	**SIFT**[12]	**SURF**[3]	**ORB**[25]	**KLT**[21]	**KL**[13]	**MS**[6]	**Proposed**
fr1/room	167	174	79	7469	7546	85	14286
fr2/desk	183	188	77	7942	7252	77	14598
fr1/plant	124	158	78	6264	6576	74	13547
fr1/teddy	172	183	75	7722	7112	92	14968



(a) **(b)**

Figure 12: The proposed algorithm applied in a monocular-SLAM system. (a) Feature matching via the BIREF visual descriptor. (b) Feature matching via the proposed algorithm. For our algorithm, 3D density is increased, then, visual environmental understanding, high-level descriptors application and objects/structures recognition performance can be highly improved.