HOOFR SLAM System: An Embedded Vision SLAM Algorithm and Its Hardware-Software Mapping-Based Intelligent Vehicles Applications

Dai-Duong Nguyen, Abdelhafid Elouardi, Sergio A. Rodriguez Florez, and Samir Bouaziz

***Abstract*— This paper deals simultaneously with the trajectory estimation and map reconstruction by means of a stereo-calibrated vision system evolving in a large-scale unknown environment. This problem is widely known as Visual SLAM. Our proposal optimizes the execution time of the VSLAM framework while preserving its localization accuracy. The contributions of this paper are structured as follows. First, a novel VSLAM approach based on a "Weighted Mean" of multiple neighbor poses is detailed and is denoted as HOOFR SLAM. This approach provides a localization estimate after computing the camera poses (6-DOF rigid transformation) from the current image frame to previous neighbor frames. Taking advantage of the camera motion, we conjointly incorporate two types of stereo modes: "Static Stereo" mode (SS) through the fixed-baseline of left-right cameras setup along with the "Temporal Multi-view Stereo" mode (TMS). Moreover, instead of computing beforehand the disparity of SS mode for all key-points set, the disparity map in scale estimation step is limited to the inliers of the TMS mode so as to reduce the computational cost. This strategy is suitable to be parallelized on a multiprocessor architecture and exhibits a competitive performance with the other state-of-the-art strategies in many real datasets. Second, we report a hardware-software mapping of the proposed VSLAM approach. To this end, a heterogeneous CPU-GPU architecture-based vision system is considered. Third, a thorough and extensive experimental evaluation of our algorithm implemented on an automotive architecture (the NVIDIA Tegra TX1 system) is studied and analyzed. We report hence the localization and timing results through experiments on five well-known public stereo SLAM datasets: KITTI, Malaga, Oxford, MRT, and St_Lucia datasets.**

***Index Terms*— Visual SLAM, scene recognition, feature extraction, hardware-software mapping, embedded systems.**

I. Introduction

VISUAL Simultaneous Localization and Mapping (VSLAM) is a key issue in computer vision and robotics community. VSLAM aims at estimating the camera trajectory while reconstructing a consistent 3D model of the environment. In the literature, existing approaches are based in two predominant perception strategies: monocular and stereo. Stereo VSLAM is generally transposable to RGBD systems [1], [2]. The most versatile of VSLAM approaches is the monocular VSLAM [3]–[6] since its hardware requirement is only one camera to observe the environment. However, "scale drift" remains an open problem of this approach. This is due to the fact that frame-to-frame motion estimates are integrated over time up to an absolute global scale. On the other hand, stereo VSLAM uses two calibrated cameras to capture the scenes so the depth from camera to points can be computed for each frame using the disparity.

Over almost two decades, there have been many successful stereo VSLAM methodologies. Early stereo VSLAM frameworks were based on classical EKF approach enclosing a large Extended Kalman Filter for managing all landmarks [7]–[9]. This approach suffers from two main problems: firstly, the quadratic complexity of the EKF limits the number of processed landmarks; and secondly, the consistency of EKF is known to be poor causing the impossibility of re-linearizing the cost function. In order to tackle such drawbacks, Paz *et al.* [10] proposed to split a large EKF filter into sub-maps. However, this variant limits the application to environments with an area of 100m². An alternative variant is FastSLAM [11] which represents trajectories by means of a set of particles and small EKF filters are assigned to each landmark. FastSLAM framework is also afflicted by its complexity when the number of particles is not bounded for a given environment. It also suffers to achieve loop-closures due to the 6-DOF nature of visual SLAM, making this approach not well-suited for large-scale scenarios.

Since EKF and FastSLAM are filter-based frameworks, they marginalize all past poses and summarize the information gained over time with a probability distribution. In contrast, keyframe-based VSLAM approach performs a windowed optimization (e.g. bundle adjustment) on a small set of past frames to optimize current pose. Then, the global optimization in case of loop closure could be done using Graph-based SLAM method [12], [13]. Strasdat *et al.* [14] compared filter and keyframe-based VSLAM demonstrating that the latter achieves a better balance between computational cost and precision.

Manuscript received December 7, 2017; revised July 3, 2018 and September 17, 2018; accepted October 28, 2018. The Associate Editor for this paper was Z. Duric. *(Corresponding author: Dai-Duong Nguyen.)*

The authors are with SATIE - UMR CNRS 8029, University of Paris-Sud, Paris-Saclay University, 91400 Orsay cedex, France (e-mail: dai-duong.nguyen@u-psud.fr; abdelhafid.elouardi@u-psud.fr; sergio.rodriguez@u-psud.fr; samir.bouaziz@u-psud.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2018.2881556

The requirement for SLAM algorithms in terms of calculation, accuracy and embeddability is a critical factor limiting the use of existing approaches in embedded applications. Meanwhile, trends towards low cost implementations and low power processing require massive parallelism and implementation on heterogeneous architectures.

II. Related Works

Inside keyframe-based VSLAM methodologies, the use of two different data representations can be highlighted: feature and image based. Feature-based strategy comes out earlier and was inspired on several researches. For instance, a scalable stereo visual SLAM have been introduced in frameSLAM [15] and RSLAM [16]. The contribution of frameSLAM consists in reducing the complexity of large loop-closures by constructing sub-maps and simplifying feature constraints into frames constraints. In this way, the mapping task was optimized so as to maintain a subset of frames (skeleton). RSLAM implements a local bundle adjustment with a bounded complexity in order to provide an accurate map and trajectory. Even if RSLAM achieves constant time complexity, the global consistency is not warranted. S-PTAM proposed by Pire *et al.* [17] exploits, in parallel, the tracking and mapping in order to achieve real-time performances. However, it lacks large loop closing which is indispensable in an accurate SLAM system. Recently, ORB-SLAM appears to be one of the most actively developed VSLAM framework. After the monocular version, Mur-Artal *et al.* contributes with a stereo version of ORB-SLAM in [1] to handle the problem of scale drift. They inherit the main spirit of S-PTAM and complement it with a loop closing procedure. A first dense image-based approach is latterly presented in which LSD-SLAM [18] is named as candidate. This approach provides depth estimation and mapping by direct image alignment with affine lighting correction on a rich set of pixels having a high intensity gradient. LSD-SLAM provides good results under low image resolution and small camera motions. The use of high resolution images or video sequences with an important interframe camera motion with LSD-SLAM provides poor localization results and its computational cost becomes a severe issue. An alternative image-based approach was presented recently and named as DSO [19]. Such a method performs following a direct visual odometry framework and does not incorporate loop-closure detection, correction and re-localization. Besides, the authors state that DSO cannot handle dynamic scenes.

The implementation of SLAM algorithms in embedded architectures is often preceded by an algorithm-architecture-adequacy study, which allows formal verifications as soon as possible to warrant the feasibility of the design and to reformulate optimization problems so as to exploit at the best the target architecture. Rodriguez-Losada *et al.* [20] analyzed the acceleration of a laser SLAM on two desktop GPUs: GF8400M and GTX280. The speed-up factors achieved are respectively 8 and 57 in comparison to the execution on a T7250 CPU (@2GHz). More recently, Whealan *et al.* [21] evaluated his approach for dense visual SLAM based RGB-D camera on a powerful heterogeneous system consisting of an Intel CPU (i7 - 3.4GHz) and an Nvidia GeForce GTX 780 GPU. A fast execution is achieved where the average time ranges from 31ms to 45ms per frame. Heterogeneous architectures (CPU-GPU, CPU-DSP or CPU-FPGA) are a common trend nowadays on computing platforms, specially for embedded systems. Therefore, many researchers took advantage of these architectures to accelerate SLAM applications. Vincke *et al.* [22] proposed an efficient EKF-SLAM system based on a low-cost and heterogeneous architecture. The hardware contains an ARM processor, an SIMD coprocessor (NEON) and a DSP core. The system implements low-cost sensors: a camera and odometers. The emergence of embedded systems has lead to several works addressing the embeddability issue of SLAM algorithms. However, few works deal with hardware-software mapping of VSLAM algorithms on embedded architectures which recently offers a high potential to have a great improvement in designing VSLAM systems.

Our Contribution: Inspired from the works in the state-of-the-art where a hardware-software mapping of SLAM algorithms were proposed and evaluated on heterogeneous architectures, we present a novel end-to-end study of an original feature-based stereo VSLAM framework. Our system is designed to work with stereo calibrated cameras, integrating our previous HOOFR extractor [23] for extracting features and matching. The processing is structured to maximize the parallelization. The algorithm complexity is optimized to be suitable for an embedded device. Hence, instead of optimizing camera poses by high cost bundle adjustment on keyframes or saving the map-points history, this algorithm is intended to achieve accurate position for each input frame. In practice, relative poses of current input frame with a set of previous neighbor frames are estimated. The optimal pose is achieved as the mean of relative poses with weighted factors. We name this variant of visual SLAM as HOOFR SLAM. The second contributions of this work consists in presenting a detailed study related to mapping the HOOFR SLAM algorithm on a heterogeneous embedded architecture. So, thanks to a hardware-software mappnig process, the proposed algorithm was implemented on an automotive embedded platform (NVIDIA Tegra TX1) with a low-power CPU associated to a GPU for hardware acceleration. A thorough and extensive experimental evaluation of our algorithm is studied and analyzed. As a third contribution, we present localization and timing results through experiments on five well-known public stereo SLAM datasets (KITTI, Malaga, Oxford, MRT and St_Lucia datasets).

III. HOOFR SLAM Algorithm

A. Algorithm Overview

The transformation (translation and rotation) of a stereo camera can be computed in homogeneous coordinates (up to scale) by one image chain (left or right). Therefore, in our system, we employ only the left image for relative motion estimation between two camera positions. The right image is used in further step to calculate the real scale. For each input stereo frame, HOOFR features are extracted in the left image and HOOFR description is then computed for both motion

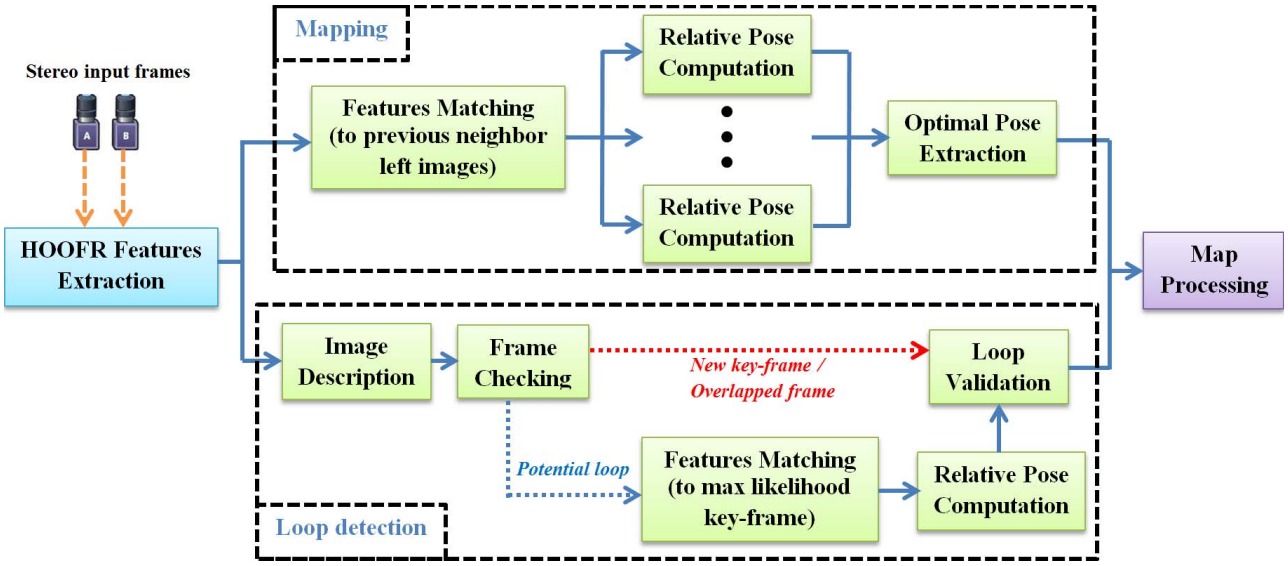


Fig. 1. Functional blocks of the algorithm flow at each input stereo frame.

estimation and loop detection. HOOFR features are matched with those of previous left image in order to estimate relative transformations between the current frame and the previous frame. We define the "previous neighbor frame" (PNF) as the frame that the camera transformation is successfully estimated from it to current frame through essential matrix.

Essential matrix (E) [24] corresponding to one relative transformation is computed from the matching sets using RANSAC [25]. The camera translation, camera rotation and landmarks positions are extracted from E by triangulation but they are in homogeneous coordinates (up to scale). In order to have a real scale, we use stereo matching to match the position of triangulated landmarks in the left image to those corresponding in the right image. The real landmark-camera distance is computed based on this stereo matching and the distribution of the left and right cameras. Scale factor is the ratio of real distance on the triangulated distance. Finally, the camera motion is estimated as the product of the homogeneous motion and the scale factor. To achieve the optimal camera pose, the main idea of our design is that we do not employ bundle adjustment which presents a high processing cost. Instead, we propose another method denoted as "windowed filtering" which estimates camera pose of current frame from a set of previous neighbor frames. For each previous neighbor frame, we apply the entire motion estimation to achieve one prediction of current pose. Each predicted pose is associated with a weight corresponding to its confidence in comparison to others predictions. The optimal current pose is then the mean of all predictions by their weights respectively.

In parallel with current-to-neighbors motion estimation, we perform a loop detection test for left image. HOOFR binary feature description is used one again to extract image description. The current left image is queried in key-frame set to find the max-likelihood. In the case of low matching score, current image is considered as a new key-frame, we add current position attached with its image description to pose graph. In contrast, potential loop closing is considered when high matching score is presented and the max-likelihood key-frame is far from current frame in pose graph. The motion estimation between current frame and max-likelihood key-frame is then computed to validate the loop closure. A real loop is taken into account only if motion estimation is successful.

Our algorithm pipeline is shown in detail in figure 1. After HOOFR features extraction, we launch at one time the Loop detection and Mapping threads. Inside Mapping thread, Features Matching block finds the correspondences for each key-point of the current frame in each PNF. We offload this block to GPU due to its computational cost. Then, a number of Relative Pose Computation tasks are executed, each of them computes one predicted camera pose from one PNF. The number of PNFs ($nframes$) hugely depends on the camera movement speed. However, in practice, due to the architecture constraints, $nframes$ is fixed to 3 or 4 for the maximum number of neighbor frames. The "Optimal Pose Extraction" block evaluates the predictions to get an optimal current pose. Otherwise, inside Loop detection thread, Image Description block describes the current frame by comparing the descriptions of relevant key-points to a bag of words (BoW). The image description is then passed to Frame Checking block to find the max-likelihood in key-frames set. We define an overlapped frame as a frame having a max-likelihood near to it in pose graph with high matching score. Normally, when we have a new key-frame, some of the following frames could be overlapped frames. Features Matching and Relative Pose Computation between current frame to max-likelihood key-frame in case of potential loop are processed by stricter condition than that of Mapping thread to warrant an accurate loop closure. "Map Processing" block gathers the result of two main threads (Loop detection and Mapping). It always updates current pose and points to the map if mapping is successful,

updates the key-frame set if loop detection determines that current frame is a new key-frame or corrects the map by distributing error along the pose graph when a real loop is presented.

B. HOOFR Features Extraction

1) Bucketing HOOFR Features: HOOFR [23] extracts key-points that are used for motion estimation. To ensure a precise estimation, many correspondences are required so that many points should be detected and described in an image. Due to this reason, we need a high speed extractor. HOOFR detector is the combination of ORB detector with Hessian score and provide better compromise between execution time and matching precision [26]. The main idea of HOOFR detector is that it detects features in an image by applying FAST detector over multiple scales of an image pyramid. Then, the detected features are filtered to keep the most relevant key-points based on their Hessian score (instead of Harris score in ORB). This filtering provides a good repeatability. It is eliminated in the processing flow of some works such as LSD-SLAM [27]. However, in our system, we maintain this filtering step to improve the matching result for an enhanced pose estimation.

In order to warrant a homogeneous distribution of features, we employ bucketing technique as used in all others systems. The input frame is divided into a grid where the number of cells depend on the image resolution. HOOFR features are then detected with adapting threshold trying best to extract enough points. We fix the maximum number of points retained in one cell to pts. In each cell, at the first detection, FAST threshold is set to a high value fa. Unless the number of key-point is higher than pts, the second detection is operated with lower FAST threshold value ($fa/2$). After detection, if the number of points is higher than pts, we compute Hessian score for each point and maintain only pts points having the highest score. The orientation and description are computed by HOOFR descriptor for the most relevant points retained by each cell. HOOFR descriptor (256 bits) is an enhanced version of FREAK (512 bits). It has a low sensitive to viewpoint and is fast to compute and match.

2) Binary Descriptor for Place Recognition: Scene recognition is the fundamental step for loop closure. Typically, this step uses SIFT or SURF full-featured descriptors due to their high matching score among the existing approaches. Nevertheless, their computational cost has degraded performances of SLAM systems. Recently, binary bag of words [28] is proposed with a competitive performance by an order of magnitude faster than floating approaches. this approach is widely used and has remarkable results in visual SLAM systems such as [4] and [29]. Thus, in our system, we integrate the idea of binary word so as to keep place recognition process light. Among the relevant key-points provided by HOOFR detector in the whole image, we select K points ($K = 150$ in our implementation) having the highest Hessian score to get corresponding words based on their HOOFR description. Image description is built from these binary words.

C. Mapping Thread

1) Features Matching: In mapping thread, features matching is carried out between the current frame and the previous neighbor frames. We note that the camera frequency is high (10-50 fps) leading to a little change between consecutive images. For this reason, the correspondence in neighbor frame is located not too far from key-point position in current frame, so we can limit the searching region instead of the whole image. For each key-point I in PNF, we perform "Brute-Force" matching with all key-points locating in same cell or "neighbor cell" in the current frame. We find the most and the second correspondences (J and J') by the smallest and the second smallest Hamming distance respectively. The result of feature matching has an important role in the precision of pose estimation so that it must be done carefully. Hence, we apply further three following conditions to select pairs among the most matching pairs $I - J$ (smallest Hamming distance):

Firstly, the matching pair must have a high distinction in comparison to its second matching. It means that the ratio of Hd_{I-J} to $Hd_{I-J'}$ must be lower than a threshold φ where Hd_{I-J} represents the Hamming distance of the pair $I-J$. The value of φ is 0.85 giving a good exhibition in our experiments.

Secondly, if the positions of I and J have a small difference in the images ($\|p_I - p_J\| < 2$), it means probably that the point is too far from camera or the camera does not move significantly compared to the previous pose. Such two cases do not provide a good estimation so that these matches should be also rejected. Furthermore, if too many matches have a small position change, the camera is considered staying nearby the previous pose.

Thirdly, in contrary to the second condition, if I and J have too big differences in positions ($\|p_I - p_J\| > 90$), it could be a false matching and also must be eliminated.

In some other researches like ORB-SLAM or LSD SLAM, people use guiding search to find correspondences. They rely on the last transformation of camera and point position in the previous frame to predict the point position in the current frame. This method has a good performance when the transformation is small and stable but it is easy to loose the tracking when the transformation becomes more critical. In our algorithm, we apply Brute-Force to find the best candidate in a large set of local features. After 3 test conditions above, we get a reliable matching set for the following step.

2) Relative Pose Computation: The goal of Relative Pose Computation (RPC) block is to compute the relative pose between two frames (always from left images of a stereo camera) and to triangulate a set of map points. There are many RPC blocks executed in parallel. We defined these execution as sub-threads inside mapping thread. Each of them estimates one relative motion from current frame to one previous neighbor frame. RPC block consists of 3 principal steps: rotation and translation extraction from essential matrix, solution determination and scale estimation. We assume the image domain to be given in stereo-rectified coordinates, the intrinsic (focal length, center points) and extrinsic (baseline, relative angle) camera parameters are calibrated a-priori.

- Rotation (R) and Translation (t) Extraction From Essential Matrix (E): Essential matrix is estimated from HOOFR matching set returned by the previous step. The epipolar geometry is described by equation (1):

$$[p_I; 1]^T K^T E K [p_J; 1] = 0 \quad (1)$$

where K is the intrinsic camera matrix, p_I (x_I, y_I) and $p_J(x_J, y_J)$ are respectively positions in PNF and current frame of a correspondence $I - J$. Each matching pair gives a constraint to solve E. In others works such as ORB-SLAM, people use 5-point algorithms [30] inside a RANSAC scheme to extract an optimized model E_{op} from matching set. They assume a standard deviation of 1 pixel in the measurement error. Then, they consider R and t extracted from E_{op} as the initial state for the optimization of bundle adjustment (BA).

In our proposal, we intend to avoid BA which has a high computational cost. Hence, we focus on the method to improve the precision of estimating E, which makes the most difference of our system with others in state of art. Before computing E, number of matching pairs (np) in each RPC block is checked. If np is under a threshold λ, the corresponding RPC block is considered as an invalid estimation and its sub-thread will be stopped immediately. In contrariwise, E estimation is processed when np is bigger than λ. Through experiments, we found that a high precise localization is presented when the measurement error (me) of inlier in RANSAC scheme is smaller than 0.4. However, when we apply RANSAC with $me = 0.4$ to the initial matching set, the execution time severely increases. The reason is that the number of iterations in RANSAC is updated during the estimating process. After each iteration, the remaining number of iterations is computed by equation (2):

$$N_I^i = max(N_I^{i-1} - 1, log\frac{1-c}{1-(n_e/N_e)^5}) \quad (2)$$

where N_I^i is the remaining number of iterations at time i, c is the parameter of confidence (normally between 0.95 and 0.99), $n_e$is the number of inliers in the best model at time t and N_e is the total number of elements in the whole set. When the measurement error is smaller than 0.4, it is obvious that n_e decreases leading to the increase of remaining number of iterations. Therefore, in order to accelerate the processing, we propose the Algorithm 1 for estimating E_{op}:

We mark the inliers of E_{op}, while outliers are rejected. Given that E_{op} has been determined; our method for estimating rotation R, translation t and 3D points triangulation is based on performing single value decomposition (SVD) of E (mentioned in Hartley & Zisserman's book [24]). Due to the fact that E is "up to scale" so that SVD provides the solution of $[t]_m$ in homogeneous coordinates (scale is not defined). Furthermore, we have 2 opposite directions which are possible for translation (t) and two different rotations (R) which are compatible with an essential matrix. This gives four classes of solutions in total for the relation between two camera coordinates. However, there is only one correction solution where the triangulated point is in front of camera at both positions (current and reference positions).

Algorithm 1 Essential Matrix Estimation From Matching Set

1) Apply 5-points algorithm inside RANSAC scheme to the initial matching set with the measurement error equal to 1.0.
2) Select the inliers corresponding to the optimal model of step 1 to form another set (refined matching set).
3) Apply 5-points algorithm inside RANSAC scheme to the refined matching set with the measurement error equal to 0.4 to get a final optimal model (E_{op}).
4) Test the final optimal model of step 3 on the whole initial matching set to select the inliers (measurement error reclaims the value of 1.0).
5) Compute the mean of measurement errors returned by inliers from step 4. The inverse of this value represents the score of the estimated model.

- Solution Determination: In order to select the correct solution among the four possibilities, for each inlier matching pair, we compute 3D triangulated position in the 4 solutions. The point is arranged to the solution in which it is in front of camera at both reference and current positions. The chosen solution is the one containing the most points seen in comparison to others. In theory, if the estimation of E is noiseless, one solution will contain all triangulated points. In that case, we can check only one matching pair to find the valid solution. However, matching is affected by noise in practice, so checking all matching pairs provides a more robust method. In particular, if image is too degraded by noise leading to no clear winner solution, the relative pose estimation of the corresponding sub-thread is stopped immediately and will be marked to be invalid.

- Scale Estimation: As the essential matrix is "up to scale", the translation and 3-D triangulated points computed above are in unit coordinates. Therefore, the residual problem after selecting the valid solution is determining the "real scale" of map and camera motion.

The most advantage of a stereo camera is providing two images from different physical cameras, taken at the same time. Hence, the depth from 3D point to camera can be estimated without scale ambiguity using stereo-disparity (static stereo). Assuming that we have a point in the left image, the correspondence of this point is searched along epipolar line in the right image. In our case of rectified-stereo, this search is performed along the horizontal lines.

As stereo correspondence measure, we use 5 pixels-SSD method [31] along the scan-line. In our system, we obtain *a-priori* a point in the left image. Hence, if we consider the same position in the right image, the correspondence is located undoubtedly on the left side of this position. In practice, the disparity range in the right image is constrained to $[(x_J - \sigma, y_J), (x_J, y_J)]$ where σ is the limited search region (σ=30 in our experiment). Once the correspondence is defined, the real 3D point $\bar{P}_J(\bar{X}_J, \bar{Y}_J, \bar{Z}_J)$ with reference to camera will be extracted by well-known static-stereo triangulation (using disparity, baseline and camera focal length)

as mentioned in [24]. We compute the real distance for all triangulated points arranged to the selected solution. Scale factor (k_J) is the ratio of the real distance of static stereo on the triangulated distance of temporal stereo. In fact, this factor is simply computed by the ratio of ($\bar{Z}_J/Z_J$). In the case of noise absence, all points have the same scale factor. However, it is never the case in practice. To have an appropriate value, we consequently employ 1-point scheme on the scale factor set as in Algorithm [2].

Algorithm 2 RANSAC Scheme for Scale Estimation

1) Take the value (k_{av}) of one element in the factors set.
2) Find the number of inliers in the entire set. A factor k_J will be classified as an inlier if the difference is small enough ($|k_J - k_{av}|/min(k_J, k_{av}) < \varepsilon$). ε is set to 0.1 in our test. Mark the scale value if it is the best model (contain the most inliers).
3) Repeat the processing for all other elements in factors set. The value of the best model is considered as the estimated scale.

In order to have reliable scale estimation, after doing 1-point scheme, we additionally evaluate the best model if the number of iterations reaches to the bound. The model is invalid when the number of inliers do not attain an acceptable value ($N_{inliers} < \gamma$). In this case, we also reject the current process, the sub-thread returns invalid estimation. Otherwise, camera translation and 3-D point position are multiplied with the scale to get the non-scale value and only 3-D points computed from inliers are maintained. Through experiments, we found that the value of γ is set to 10 giving a good performance.

3) Optimal Pose Extraction: This block is the summary step in mapping thread and takes into account all predictions from sub-threads to calculate the optimal camera pose. In practice, for each sub-thread, we notice that relative pose is extracted from Essential matrix which is obtained beforehand from features matching set. Therefore, we propose to use inverse of mean error retained by inliers after essential matrix estimation as a weight factor of predicted pose. Equation (3) shows the computation of optimal current left camera pose C^l (also defined as $[R|t]$ in some references) from all predictions:

$$C^l = \Sigma_{n=1}^{N} \frac{\sigma_n}{\Omega} \hat{C}_n^l \tag{3}$$

where $\hat{C}_n^l$ is the predicted position of the sub-thread Th_n, σ_n is the inverse of mean error of inliers and $\Omega = \Sigma_{n=1}^{N} \sigma_n$. Besides, N represents the number of valid sub-threads which compute a prediction with positive weight. Contrariwise, when a sub-thread is marked to be invalid, its weight takes the value of 0 and it will be ignored in optimal pose extraction. When all prediction are invalid, current frame is not tracked. In this case, map is not updated and we proceed directly to the next frame.

D. Loop Detection Thread

Loop detection thread runs in parallel with mapping thread. It processes the current frame and tries to detect a loop closure.

1) Image Description: For loop detection and re-localization, our system implements a bag of words place recognition based on FAB-MAP 2.0 module exhibiting a robust performance as shown in [32]. We use the HOOFR key-points and their HOOFR descriptor to extract the bag of words in a large images set. A 256-bit binary descriptor contains in total 2^{256} different words. However, a huge vocabulary not only takes much time to build image description but also has a poor efficiency in loop detection. The issue is that we have a low tolerance when too many words are maintained in the vocabulary. In such case, a 3D point will be assigned easily to two different words when camera has little position change. Consequently, a low similarity between 2 images is presented through these images of the same scene. In experiments, we found that a vocabulary of 10000 words provides a favorable compromise between precision and execution time. The vocabulary is created offline one time from a large set of random images and is used for all test sequences.

2) Map and Key-Frame Set: In our system, map is represented as a set of $M_i = C_i^l, T_i^l, L_i$. Each M_i contains position of left camera C_i^l in global coordinates, relative transition T_i^l to previous left camera position and all 3-D landmarks positions L_i in camera coordinates. Our developed system is similar to recent SLAM systems that do not consider all processed frames as key-frames due to 2 main reasons:

- For each input frame, each element in key-frames set will be queried to compute the likelihood percentage. The frame sampling is aimed to reduce the size of key-frames set. Hence, the computation will be light. This strategy is suitable for implementing the application on an embedded system where memory resources are limited.
- There exists always an overlap between consecutive frames. It means that when images are taken from close times, they contain many common words. In many cases, two images take exactly the same words from environment. The overlap will cause problem in computing likelihood percentage when these two images attain the same value. In this case, all percentages have small values causing the ambiguity in loop detection.

Therefore, we consider a frame as a new key-frame when there is no likelihood percentage value bigger than η (0.99 in our experiments). Each key-frame is then updated into key-frame set $KE_i = id_i, V_i, D_i$ where id_i is the index of the key-frame position in the map, V_i and D_i are respectively features and their HOOFR description extracted from key-frame.

3) Frame Checking : First of all, in "Frame Checking" block, K binary words retrieved from the most K relevant features in HOOFR extraction are employed as well as vocabulary to build image description. Specifically, each of K binary words will be queried in vocabulary to find the best matching word (lowest Hamming distance). Image descriptor is formed by taking into account which words that image takes from the vocabulary. Likelihood percentage is then computed for all elements in key-frame set based on their image descriptor. If the maximum likelihood is less than η, "Frame Checking" decides that current frame is a new key-frame and we will update it to the key-frame set in "Map Processing" block.

When maximum likelihood is bigger than η, the frame is not a new key-frame. However, we fall into two possibilities: the overlap with previous images or potential loop detection. In practice, to manage key-frame set, a variable called "historical time" (ht) is additionally attached to each element in the set. Once key-frame is added to the set, this value is initialized as the size of the set at that moment. Besides, when loop closing is successfully processed at this key-frame, ht is updated by the size of the set at the update moment. A "newcomer" (newly added or processed) is identified when ht is closely to actual size of key-frame set. Moreover, after a loop closing, it is probably that we have many loop points nearby. In order to avoid too many loops processing, we count the number of new key-frames added from a loop point. "Frame Checking" recognizes a potential loop when two conditions below are satisfied:

- Historical time of maximum likelihood key-frame ht_m is smaller than the size of actual key-frame set by t ($ht_m < keyframeset.size() - t$).
- Ne new key-frames have been already added from the last loop point.

where t, Ne are respectively set to 5 and 10 in our experiments. Otherwise, it is recognized as an overlapping frame.

The features matching and relative pose estimation between current frame and maximum likelihood frame are performed only in the case of potential loop. We use the word "potential" because the current frame must finally be validated by pose estimation. In our experiments, most of the frames are recognized as a new key-frame or overlap with the previous frame. Features matching and Pose estimation tasks are processed only when a loop point is closely attained. Nevertheless, we found that some particular frames are potential loops but they are not the real loops. This is inevitable and it occurs when two images of different places take too many common points in the vocabulary. However, these frames are rapidly rejected after features matching due to the lack of valid correspondences or rejected in pose estimation based RANSAC since there is not enough inliers.

4) Features Matching: In loop detection thread, features matching block is between current frame and its max-likelihood frame when a potential loop is detected. As a precise loop requires severe checking conditions, we propose to use "cross Brute-Force" matching instead of the high distinction checking. The idea is that we keep the second and the third checking conditions as in features matching of mapping thread. However, we change the first condition as following:

- Two feature sets are matched using local Brute-Force matching in two direction. For each point I in the max-likelihood frame, we find the best local match J (smallest Hamming distance) in the current frame and vice-versa.
- The matches verify the first condition if they have the same matching results in two direction ($I \rightarrow J$ and $J \rightarrow I$).

This stricter condition allows us to detect the "false positive" of potential loop (a high similarity but not a same scene) where few matching pairs retained after checking.

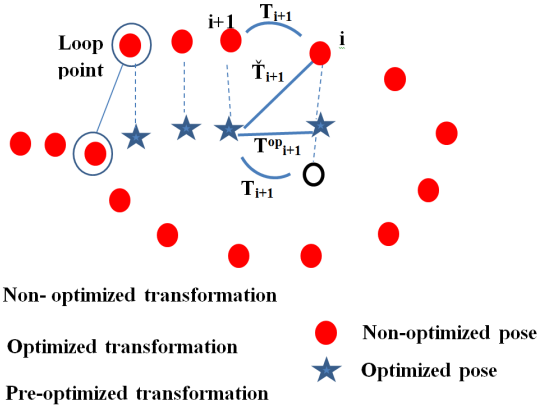


Fig. 2. Loop correction.

5) Relative Pose Estimation: RPC block in loop detection is similar to that in mapping thread except the change of the threshold λ. We also increase the value of λ to insure that only "true positive" of potential loop is handled. The reason is that after a tightening matching, we require more number of matching pairs retained to compute essential matrix. In experiments, we found that this combination exhibited a tremendous performance with no "false positive" loop passing.

E. Map Processing

Map Processing block considers results returned from mapping and loop detection threads to make the decision. Table I resumes all possibilities that the system can meet. If mapping consecutively fails after a fixed number of frames due to some reasons such as abrupt movement or occlusion, our system turns into tracking-lost state (tracking lost = true). In this state, each frame is processed only by loop detection thread. Mapping thread is disabled. Once the camera is relocated in the map, we return to tracking-active state. However, map optimization will be neglected as the lack of previous poses. Moreover, map will be discrete at the relocated point and the incoming optimization is limited to this point. In a normal situation when tracking-lost is false, if mapping is invalid for current frame while loop detection provides a legal result, we buffer the loop information. In the limited following frames, in the case that mapping revives, loop closing will be performed.

"Map correction" is called only if both loop and mapping threads return valid estimations. Once it is activated, the trajectory is optimized by distributing loop closing error along pose graph. The propagation starts from loop point, follows the trajectory back to the point to which loop point is attached. Figure 2 shows the correction applied for each position in the map. Assuming that position $(i+1)$ is optimized, we compute the transformation $\check{T}_{i+1}$ between the optimized pose C^{op}_{i+1} and the non-optimized pose C_i, while T_{i+1} is the transformation between two non-optimized poses already maintained in the node $(i+1)$. T^{op}_{i+1} is then estimated by equation (4) where μ represents the "propagation coefficient". In our experiments, we propose to compute μ depending on the

TABLE I
POSSIBILITIES AND DECISION OF "MAP PROCESSING" BLOCK

Blocks									
Tracking lost	TRUE			FALSE					
Mapping	-			Valid			Invalid		
Loop detection	Key-frame	Neutral	Loop	Key-frame	Neutral	Loop	Key-frame	Neutral	Loop
Decision									
Update pose graph	-	-	+	+	+	+	-	-	-
Add key-frame	-	-	-	+	-	-	-	-	-
Activate Map Correction	-	-	-	-	-	+	-	-	-
Re-localization	-	-	+	-	-	-	-	-	-

number of optimized positions (N_p) in total by equation (5). The optimized pose of node i (C_i^{op}) is finally computed by equation (6).

$$T_{i+1}^{op} = \mu \check{T}_{i+1} + (1-\mu) T_{i+1} \tag{4}$$

$$\mu = \pi / N_p \tag{5}$$

$$C_i^{op} = C_{i+1}^{op} * T_{i+1}^{op^{-1}} \tag{6}$$

The execution time of map correction depends on the map size. Following time, this step becomes costly with a large loop closure. To warrant frame-rate processing, we launch "map correction" as a thread in parallel and continue to process next frame. However, the key-frames set is blocked in order to avoid memory accessing dump during map correction. As a consequence, any new key-frame is added and we just update pose graph until the current correction thread is finished.

IV. HARDWARE-SOFTWARE MAPPING ON A CPU-GPU ARCHITECTURE

The heterogeneous CPU-GPU architecture is considered in our system due to its popularity in embedded computing platforms. The algorithm is analyzed based on the data flow for each functional block. The evaluation methodology consists on the identification of blocks consuming a significant processing time or having a low data dependence. During the experiments, we found that Features Matching block has a high computational cost but could be parallelized thanks to the independence in data flow. In this functional block, each point correspondence in an image will be found by comparing the HOOFR 256-bits descriptor of this point to that of each point in the other image. For a high localization precision, a large number of points detected is required, leading consequently to a high matching cost. However, processing of each point is not related to others so a parallelization can be performed. Otherwise, HOOFR features extraction is also accelerated using OpenMP to exploit all the computing cores of the CPU.

A. OpenMP Implementation of HOOFR Extraction

The HOOFR detection is more suitable to implement on CPU than GPU architecture due to 2 main reasons. Firstly, HOOFR is based on FAST detection which employs a segmentation test to accelerate feature extraction processing. In the segmentation test, a pixel can be rejected after one or two pixel tests. Such a strategy makes the difference in processing cost for each pixel (some pixels require much more time to be processed than others). Hence, it is not suitable to be implemented on a GPU architecture where each work-item requires the same complexity to make use of computation resources. Secondly, the next step after FAST detection is Hessian filtering. Hessian score is computed for all the features returned by FAST detector and then only some relevant features with the highest Hessian score are kept. This filtering is much more rapid on CPU thanks to the binary classification (used in std::nth_element function of C++ stdio.h library). However, binary classification needs a dynamic memory allocation which is not supported on GPU. Hence, in our system, we employed OpenMP to implement HOOFR feature extraction.

There are two parts in the images: passive zone and active zone. As we select a pattern of surrounding points to make its description, passive zone is a part of image where the pixels are close to the border so that the description pattern is out of image. Passive zone is determined by *edge_threshold* in HOOFR descriptor and it is useless to detect key-point inside this part. Otherwise, active zone is the part where key-points could be described without doubt by HOOFR descriptor. Active zone is divided into grid. The number of cells in X and Y axes are set based on the image resolution in such a way that each dimension of one cell is about 80 – 150 pixels. The detection performing on one cell is independent to that on others cells.

HOOFR detection is demonstrated on the first part of algorithm 3. Each OpenMP thread processes an image cell and individual key-point sets are created for each cell to assure data independence. *NUM_THREADS* represents the number of cells handled in parallel. We assign value to *NUM_THREADS* by the total number of cores inside the processors to make use of computing resources. A bigger value of *NUM_THREADS* is meaningless in practice since a maximum parallelism was employed. In each cell, FAST detection is performed with adapting threshold. Then we extract the relevant key-points corresponding to the highest HESSIAN scores. At the end of detection phase, all key-points are regrouped in one global set to which a structure defined as *Points_Distribution* is attached. This structure represents the distribution of key-points in the image and is later required in Matching block to specify the searching regions. We chose the static mode for OpenMP

Algorithm 3 OpenMP Implementation of HOOFR Extraction

```
//////******Detection******//////
#pragma omp parallel for num_threads(NUM_THREADS)
For each image cell do
   keypoints_cell ← FAST_Detection(fa);
  ///***adapting detection***///
  if ( keypoints_cell.size() < pts)
     keypoints_cell ← FAST_Detection(fa/2);
  end if
  if ( keypoints_cell.size() > pts)
     Compute_Hessian_score;
     keypoints_cell ← Retain_relevant_points;
  end if
end for
//////******Key-points Regrouping******//////
  [Points_Distribution, keypoints_set] ← Regroup_keypoints;
//////******Description******//////
#pragma omp parallel for num_threads(NUM_THREADS)
For each keypoint in keypoints_set do
   Compute_keypoint_descriptor;
end for
```

scheduling instead of the dynamic mode. The reason is that computational complexity in each thread is comparable to that in another thread. Static mode is hence more suitable in which the chunks are scheduled to threads during compilation while dynamic mode is not efficient due to the more locking. Similar to the detection phase, features description is also parallelized using OpenMP but the strategy is modified. We note that the number of key-points detected in each image cell is not constant. Specially, when non-texture parts appear in the scene, some image cells contain very few key-points in comparison to other cells. If we keep the parallelism on image cell level, the threads handling many key-points will be extremely more costly than the threads with few key-points. In such case, some computing units finish the work too fast and have wasting time to wait others. To avoid this issue, we propose to use OpenMP at key-point level as shown on the second part of algorithm 3. Orientation and description for each key-point are extracted without dependence on any another key-point. The same complexities are presented for all threads leading to an efficiency in work distribution among computing units.

B. GPU Implementation of Features Matching

In this paper, our system is developed on a CPU-GPU architecture. CUDA and OpenCL are two well-known languages for GPU programming. OpenCL is supported by several high-end GPUs (NVIDA, AMD, Intel, etc.). It is also a framework for programming across various heterogenous platforms such as: CPU-GPU, CPU-DSP or CPU-FPGA. Otherwise, CUDA is less flexible when only supported by NVIDIA hardware. However, in some powerful embedded platforms (Tegra K1, X1, X2), NVIDIA supports only CUDA programming. In order to make HOOFR SLAM work on several architectures, we developed Features Matching block in three versions: OpenCL and CUDA versions running on a GPU and a standard C++ version running on a CPU.

OpenCL divides gpu memory access into 3 principal types: *global memory*, *local memory* and *private memory*. *Global memory* is accessible by all kernels in the GPU and is the only part which has data interaction (transfer and receive) with CPU memory. Otherwise, *local memory* is accessed by the kernels in the same work-group. It is integrated on-chip and has a fast accessing time in comparison to global memory. However, local memory is limited so that we should avoid abusing it. Finally, *private memory* is the fastest memory but the smallest part seen from kernel. It belongs to a work-item and is accessed only by this work-item. In features matching of HOOFR SLAM, we benefit all kinds of GPU memory to have an optimized implementation. CUDA uses the same manner to observe GPU memory but with a little change in naming: *global memory*, *shared memory* (corresponding to *local memory* in opencl) and *local memory* (corresponding to *private memory* in opencl). The CUDA programming nature is also similar to that of OpenCL. Hence, in the following, we only detail the implementation in OpenCL while the other could be deduced easily.

To implement features matching on GPU, key-point information must be transferred to the GPU global memory. As shown in figure 3, two parameters (*cel*, *des*) are required for each key-point in PNFs. *cel* is in the form of integer number corresponding to the cell where the key-point is located. It takes the values from 0 to (n-1). Besides, *des* is 256-bit HOOFR description of the key-point. In practice, *des* is performed using a matrix having 1 row and 32 columns with 32 elements of type "unsigned char". To regroup all parameters for PNFs, we create two matrices as in equations (7, 8) where *Pnf_Cels* and *Pnf_Dess* are respectively in dimension of (*pnf_np* x 1) and (*pnf_np* x 32), *pnf_np* is the total number of key-points in PNFs.

$$Pnf\_Cels = [cel_{11}\ cel_{12}\ ...\ cel_{1m}\ .....\ cel_{i1}\ cel_{i2}\ ...\ cel_{il}]^T \quad (7)$$

$$Pnf\_Dess = [des_{11}\ des_{12}\ ...\ des_{1m}\ .....\ des_{i1}\ des_{i2}\ ...\ des_{il}]^T \quad (8)$$

On the side of current frame, two parameters are also taken into account. Firstly, we create *Cur_Dess* matrix having the dimension of (*cur_np* x 32) for key-point description. *cur_np* is the number of key-points in current frame. Similar to *Pnf_Dess*, each row of *Cur_Dess* serves as one 256-bit description based on 32 unsigned char numbers. Secondly, key-points set of current frame is organized by the order of image cell so that a structure denoted as *Points_Distribution* is employed. This structure is transformed into an integer matrix with the dimension of (*N_CELLS* x 2) while *N_CELLS* is the number of image cells. In *Points_Distribution* matrix, each row corresponds to the distribution of one cell in the whole key-points set: the first element *ref* is the position where the first key-point of the cell is located in the whole set, the second element *nb* is the number of key-points of the cell.

In practice, *Pnf_Dess*, *Pnf_Cels*, *Cur_Dess* and *Points_Distribution* matrices are transferred to *GPU_Pnf_Dess*,

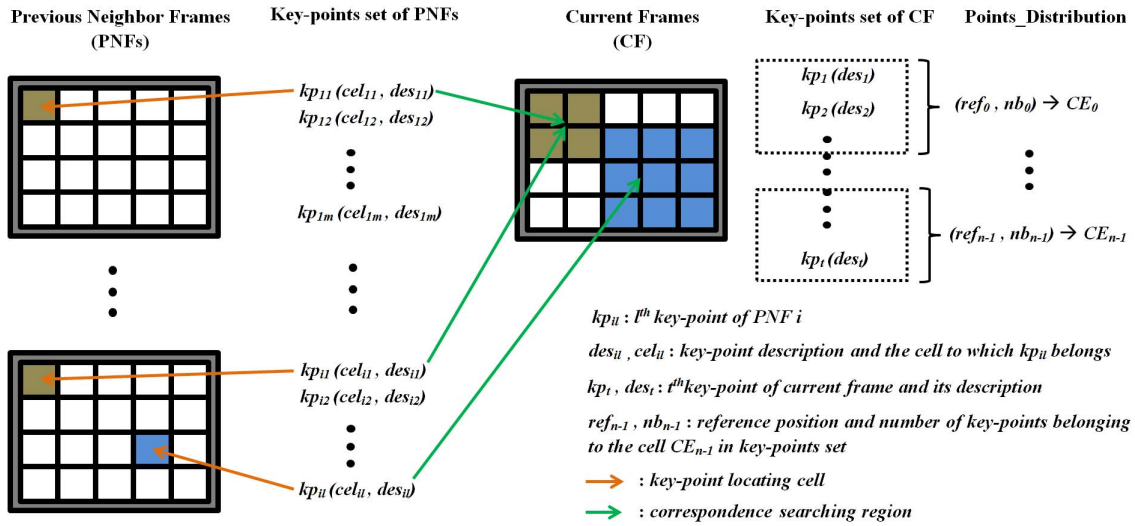


Fig. 3. Matching strategy.

GPU_Pnf_Cels, *GPU_Cur_Dess* and *GPU_Points_Distribution* respectively on GPU global memory. These memory parts are set to "read-only" to not be changed by any work-item. Moreover, we also allocate on GPU global memory a "write-only" integer matrix referred as *GPU_Correspondence* (*pnf_np* x 3) on which matching result is returned. We note that all input matrices are aligned to 1-D array on the GPU memory since GPU programming do not support pointer-to-pointer variable.

A natural implementation at our first try is that we process the whole matching of one key-point on one work-item. However, by this naive approach, we encountered the "overhead computation" problem. In fact, when kernel has too high computational cost, kernel execution takes too much time to complete one work-item. At this time, the "watch-dog" in GPU driver considers that GPU is idle since there is no feedback from kernel during an amount of time. This confusion leads to the GPU frequency reduction which severely decreases GPU timing performance. Therefore, in order to avoid such issue, we keep the kernel light by splitting the matching of one key-point into several work-items. In practice, we search the correspondence in the current frame at the same cell and neighbor cells as mentioned in figure 3. The searching on one cell is rapid due to a small number of key-points so that it is suitable to be operated on one work-item.

Algorithms 4 and 5 show the calling function on CPU and the kernel running on GPU for feature matching in mapping thread. The main idea is to use 9 work-items in a work-group to find correspondence in 9 neighbor cells of current frame. In kernel, *cell_id* variable is the index of image cell where the work-item performs the searching. *cell_id* is one neighbor cell so that it is determined by local_id *kc* of the work-item and image cell *GPU_Pnf_Cels*[i] of the PNF key-point. Keypoints locating in the image cell *cell_id* of current frame are classified from position *ref_l* to position *ref_h* in the key-points set. Besides, *dist_min* and *trainIdx* correspond respectively to the distance and the index in key-points set of

Algorithm 4 Calling Function on Host (CPU)

```
function Matching
.....
workitems = (pnf_np+BLOCK_SIZE-1)/BLOCK_SIZE*BLOCK_SIZE;
global_work_size[] = {workitems,9};
local_work_size[] = {BLOCK_SIZE,9};
clEnqueueNDRangeKernel(cmd_queue, matching_kernel,
2, NULL, global_work_size, local_work_size,
0, NULL, NULL);
clFinish(cmd_queue);
.....
end function
```

the first matching, while *dist_min2* is the distance of the second matching. Opencl local memories are allocated to save 9 local results and are synchronized by **barrier** function. After the synchronization, only one of these 9 work-items (*kc*=0) continues handling the local results to extract the final matching. It specifies final *dist_min* and *dist_min2* from local results and validates the matching if the ratio *dist_min/dist_min2* is lower than *0.85*. BLOCK_SIZE represents the number of PNF key-points processed also in the same work-group. Thus, *local_work_size* is assigned to {BLOCK_SIZE, 9}. The value of BLOCK_SIZE depends on many factors defined in GPU architecture such as the maximum *local_work_size* in each dimension or the local memory capacity. In our implementation, BLOCK_SIZE is set to 16 which provides a good performance. OpenCL programming claims that *global_work_size* must be a multiple of *local_work_size* in all dimension. Hence, the first dimension of *global_work_size* must be the nearest multiple of BLOCK_SIZE that is greater or equal to *pnf_np*. The work-items having the global identification bigger than *pnf_np* will be stopped rapidly after the test at the first line in kernel. The second dimension of *global_work_size*

Algorithm 5 OpenCL Matching Kernel on Device

declare global arrays: GPU_Pnf_Dess, GPU_Pnf_Cels, GPU_Cur_Dess, GPU_Points_Distribution, GPU_Correspondence;
function KERNEL: MATCHING
declare 3 local arrays: DIS_min[9*BLOCK_SIZE elements], DIS_min2[9*BLOCK_SIZE elements], MatchingId_min[9*BLOCK_SIZE elements];
i ← get_global_id(0);
ki ← get_local_id(0); *////from 0 to BLOCK_SIZE-1*
kc ← get_local_id(1); *////from 0 to 8*
////identify neighbor cell
cell_id ← **Get_Neighbor_Cell_ID**(GPU_Pnf_Cels[i], kc);
//// Get keypoint descriptor
point_pnf_des ← **Get_Keypoint_Descriptor**(GPU_Pnf_Dess[32*i]);
////Get local matches to from the neighbor cell
{DIS_min[9*ki+kc], TRAINIdx_min[9*ki+kc], DIS_min2[9*ki+kc]} ← **Find_Local_Best_And_Second_Matches** (point_pnf_des, GPU_Cur_Dess, GPU_Points_Distribution);
barrier(CLK_LOCAL_MEM_FENCE);
*///**At this point, local matching result of 9 neighbor cells are saved at the positions from 9*ki to 9*ki+8 **///*
if (kc==0)
GPU_Correspondence ← **Find_Global_Matches**(DIS_min, TRAINIdx_min, DIS_min2);
end if
end function

takes the value of 9 similarly to the second dimension of *local_work_size*. GPU programming is also employed for features matching in loop detection thread and we use the same approach as in mapping thread to find correspondence. However, matching conditions have a little changes leading to some modifications in matching kernel. Firstly, due to the fact that "cross BruteForce" is used, only the last matching will be searched in each matching direction. *dist_min2* will not be considered so that we do not need to allocate GPU memory to save it. After barrier function, the process is also simpler when only the last matching is extracted from 9 local ones. Secondly, in CPU calling function, two kernel calls (**clEnqueueNDRangeKernel**) are required: one for "*current frame* to *max_lilkelihood frame*" key-points matching and the second is for the opposite direction "*max_lilkelihood frame* to *current frame*". On the other hand, BLOCK_SIZE still keeps the value of 16. *local_work_size* and *global_work_size* in each kernel call are computed by the same manner as used in mapping thread CPU call.

Figure 4 presents the CPU-GPU mapping of the algorithm. In order to avoid memory access conflict, mapping and loop detection thread work on separate zones of CPU memory. Each zone is pinned respectively to that of GPU global memory where the corresponding matching kernel is performed.

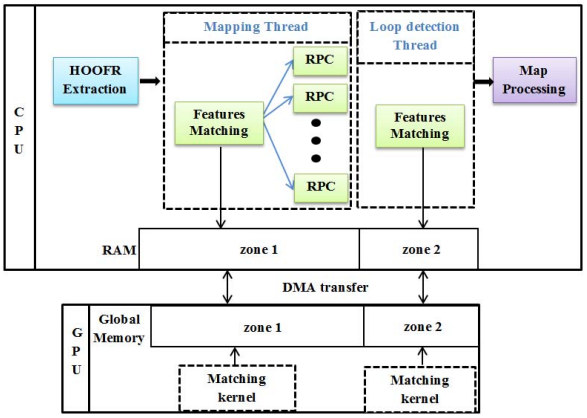


Fig. 4. CPU-GPU mapping.

TABLE II
ALGORITHM PARAMETERS

Parameter name	Value
Number of features	1500-2500
FAST threshold (fa)	12
Difference threshold in feature matching (φ)	0.85
Low threshold of pixel position change	2
High threshold of pixel position change	90
RANSAC threshold in E estimation	1.0-0.4
Inliers scale threshold (ε)	0.1
Number of binary words for loop detection (K)	150
Maximum neighbor frames (*nframes*)	4

Memory pin also allows to active DMA high-bandwidth data transfer between CPU and GPU.

V. SYSTEM IMPLEMENTATION AND EVALUATION

A. Localization and Mapping results

We evaluate our proposed algorithm on five well-known datasets KITTI [33], Oxford [34], Malaga [35], MRT [36] and St_Lucia [37] with full image resolution. Table II regroups the main parameters of our algorithm used in the experiments. In order to warrant the precision, we detect 2000 features per image in KITTI, MRT, St-Lucia sequences; 2500 features in Oxford and 1500 features in Malaga sequences.

1) KITTI Dataset: The KITTI data set consists of 22 sequences from a car driven around a residential area. The car speed is up to 90 km/h, images are recorded at 10fps with a resolution of 1234x376 pixels and many moving objects exist in the scenes. Ground truth is provided in the 11 sequences (00-10) by an accurate GPS and a Velodyne laser scanner. Some sequences contain a significant loop-closure, i.e 00, 02, 05, and 07. We compare the performances with stereo ORB SLAM - one of the most robust algorithm which uses high cost bundle adjustment and contains loop closure in the state-of-the-art. We apply the algorithm on 11 first sequences, blue curves represent the ground truth provided by a precise RTK-GPS.

The entire localization of the 11 sequences are shown in figure 5 observed in 2D of X-Z axis. By reference to

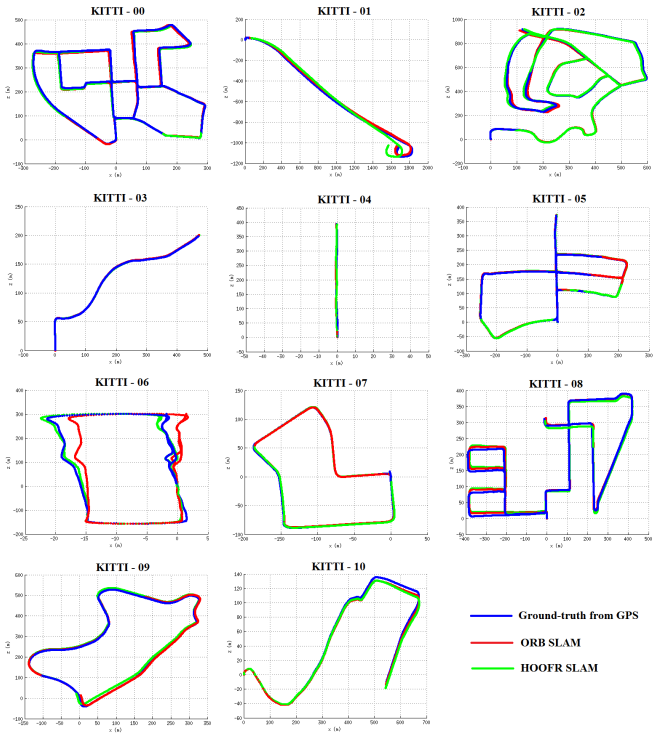


Fig. 5. Localization results of ORB SLAM and HOOFR SLAM in KITTI dataset.

TABLE III
ROOT MEAN SQUARE ERROR (RMSE) IN KITTI DATASET OF STEREO HOOFR SLAM CALCULATED FOR X AND Z AXIS

Seq	Dimension(mxm)	Frames	ORB RMSE(m)	HOOFR RMSE (m)
00	564 x 496	4541	4.7612	3.2306
01	1840 x 1140	1101	17.7170	50.2589
02	599 x 946	4661	6.6243	4.7042
03	471 x 199	801	1.2390	1.2609
04	0.5 x 394	271	0.3677	0.3225
05	479 x 426	2761	1.1884	1.3507
06	23 x 457	1101	1.6343	0.8061
07	191 x 209	1101	0.9304	0.9199
08	808 x 391	4071	4.8629	6.4138
09	465 x 568	1591	4.2835	6.7374
10	671 x 177	1201	2.7623	3.7944

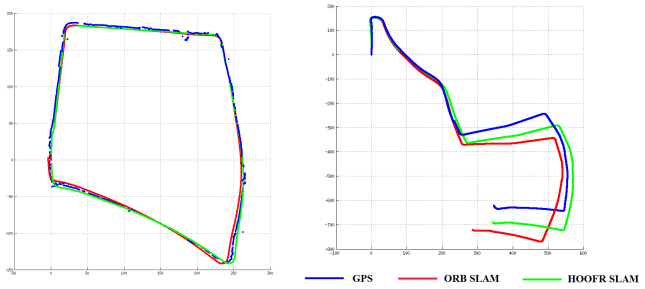


Fig. 6. Localization results of HOOFR SLAM on static (left) and dynamic (right) sequences of Oxford dataset.

camera, X is the horizontal line pointing to the right side, Y is the vertical line pointing to ground and Z is the line pointing forward. Regarding the figure, our proposal have a competitive performance with respect to ORB SLAM except the sequence 01. The reason is that this sequence is captured by a car traveling on a high way with very high velocity. ORB SLAM obtains a precise localization by saving the map-points history and using the high cost bundle adjustment optimization. In contrast, our algorithm is aimed to get high speed processing and reduce memory resources usage, so that the precision is sacrificed in this case.

Table III shows the Root Mean Square Error (RMSE) of trajectory for each sequence computed only for X and Z axis due to the fact that although GPS is corrected by RTK signal, ground-truth in Y axis is still not reliable. The results indicate that our system has a considerable accuracy with a trajectory error around 1% of its dimension (except 2% for sequence 01). The percentage is computed by the ratio of RMSE over the maximum value of 2 dimensions. Despite of the less complexity, our proposal even surpasses ORB-SLAM in some sequences such as 00, 02, 04 or 06.

2) Oxford Dataset: Oxford dataset is recorded by 6 cameras mounted onboard a vehicle traversing a route through central Oxford. The ground truth is provided by the fused GPS+Inertial solution. In order to evaluate HOOFR SLAM, we choose two sequences: the "static sequence" (recorded on 2014/05/06 at 12:54:54 GMT) contains very few moving objects and the "dynamic sequence" (recorded on 2014/06/24 at 14:47:45 GMT) is a challenging by a longer trajectory and in presence of many moving objects in the scenes. Figure 6 shows the performances of HOOFR SLAM on these two sequences.

On static environment, HOOFR SLAM and ORB SLAM present a very robust performance where RMSEs are respectively 2.24m and 2.22m. However, the localization error is increased on "dynamic sequence" where the RMSE is 40m for HOOFR SLAM and 70m for ORB SLAM. One of the most challenge of dynamic sequence is that there are some blurry images caused by sunlight. Hence, the degraded result is explained due to two factors: the moving objects and the poor image quality.

3) MALAGA Dataset: Malaga stereo dataset was gathered entirely in urban scenarios with a car equipped with a Bumblebee2 stereo camera running at a high rate (20fps). We chose the 10th sequence of dataset because it contains a very long trajectory, several loop closing and a huge variation of image brightness during the experiment. We also test localization performances of stereo HOOFR-SLAM in comparison to stereo ORB-SLAM and the result is shown in figure 7. To the best of our attempts, ORB-SLAM did not exhibit a converging trajectory. At some points when the image brightness is low, ORB-SLAM provides a poor localization or even lost tracking. Otherwise, our algorithm shows a remarkable localization result with $nframes = 2$ and the number of keypoints set to 1500. The argument explaining this situation is that ORB-SLAM uses ORB detector while our proposal uses HOOFR detector. Following our previous

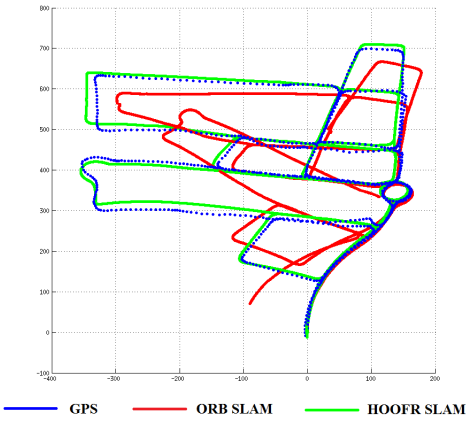


Fig. 7. Localization result using Malaga sequence: GPS (blue), ORB-SLAM (red) and HOOFR-SLAM (green).

publication [23], HOOFR detector has better repeatability than ORB detector in case of brightness change. By reference to GPS result, the reconstructed trajectory of our proposal is more reliable than that of stereo ORB-SLAM.

4) MRT and St-Lucia Datasets: In our experiments, we also validate HOOFR SLAM performances on two old datasets: MRT [36] and St-Lucia [37]. MRT is realized in 2010 using 20Hz calibrated stereo cameras. The stereo images are recorded from the AnnieWAY vehicle driven in a loop at a bridge in the city of Karlsruhe. Besides, St-Lucia dataset is gathered from 30 Hz calibrated stereo cameras embedded on a car driven on 9.5 km around the University of Queensland's St Lucia campus. The reconstruction results of HOOFR for these two datasets are shown in figure 8 including trajectory generated from GPS (no RTK correction). Comparing to GPS data, HOOFR exhibits a considerable localization result. Noting that although GPS devices used in the two datasets are not very precise, it allows us to recognize the general shape of the trajectories.

B. Timing

1) Architecture Description: In experiments, we have implemented HOOFR SLAM on two CPU-GPU platforms: a powerful Intel PC and an NVIDIA JETSON Tegra TX1 development system. Table IV shows their specifications as a recap.

Intel PC provides a mighty CPU containing 8 cores i7 running at 3.4 GHz. The CPU architecture optimizes memory access by offering 8MB smart cache that allows all cores to dynamically share access to the last level cache. The main memory (RAM) is 16 GB allowing storing a very long trajectory. This machine also integrates an NVIDIA GT-740 GPU with 384 shader cores, 2GB global memory and 28.8 GB/s memory interface bandwidth. The GPU programming supports CUDA and OpenCL.

On the other hand, since Tegra X1 (TX1) is a platform for embedded applications, its design is aimed to consume less energy. On this board, NVIDIA has elected to use ARM's Cortex containing a cluster of 4 high performance A57 big cores and a cluster of 4 high efficiency A53 little cores.

TABLE IV
ARCHITECTURE SPECIFICATIONS (JETSON TEGRA X1 EMBEDDED SYSTEM VS POWERFUL INTEL PC)

	TX1	Intel PC
CPU	4-cores ARM A57 4-cores ARM A53	8 intel cores i7
CPU clock rate	1.3-1.9 GHz	3.40 GHz
Cache	2 MB	8 MB
RAM	4 GB LPDDR4	16 GB
GPU	256-core Maxwell	384-core Geforce GT 740
GPU clock rate	1 GHz	1.07 GHz
Operating System	Ubuntu 14.04	Ubuntu 14.04
CUDA version	7.0	7.5
OpenCL version	-	1.2

TABLE V
MEAN OF EXECUTION TIME (MILLISECONDS) USING KITTI DATASET FOR EACH FUNCTIONAL BLOCK IN HOOFR SLAM ON THE INTEL POWERFUL PC AND THE TX1

	nframes = 2			
	Intel		Tegra TX1	
	CPU (8 cores)	CPU-GPU	CPU (4 cores)	CPU-GPU
HOOFR Extraction	8.536	8.555	16.783	16.731
Mapping	52.126	27.332	119.937	99.185
Loop detection average	15.001	7.881	21.916	16.466
Loop detection cost-time	36.553	15.253	95.248	80.223
Map Processing	0.349	0.137	0.584	0.403

TABLE VI
MEAN PER-FRAME EXECUTION TIME COMPARISON ON KITTI

Algorithm	Execution time (ms)	
	Intel PC	Tegra TX1
Stereo ORB SLAM	69.924	190.710
CPU - HOOFR SLAM	62.235	137.235
GPU - HOOFR SLAM	36.154	116.552

However, only one cluster could be activated at a time. The A57 CPU cluster operates at 1.9 GHz, has 2MB of L2 cache shared by the four cores with 48KB L1 instruction cache and 32KB L1 data cache per core. The A53 CPU cluster operates at 1.3 GHz, with 512KB of L2 cache shared by four cores, 32KB instruction and also 32 KB data L1 cache per core. The GPU of TX1 is designed using Maxwell architecture, includes 256 shader cores and clocks at up to 1GHz. GPU memory interface offers a maximum bandwidth of 25.6 GB/s with the capacity of 2GB global memory. NVIDIA supports only CUDA for gpu programming on TX1 (no OpenCL), so that we use CUDA version of HOOFR SLAM matching block during the experiments on this board.

2) Timing Evaluation : We evaluate the mean processing times of the proposed algorithm on 11-first sequences of KITTI dataset. All timings are given in milliseconds. The values are the mean of 11 sequences where timing on each

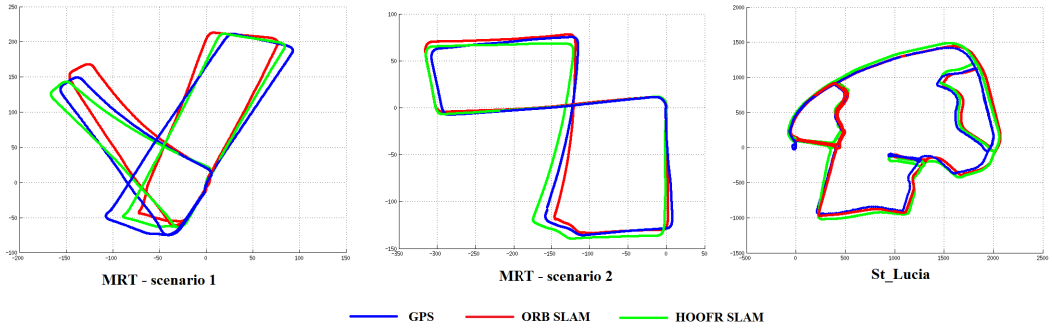


Fig. 8. HOOFR SLAM reconstruction using MRT and St-Lucia datasets.

TABLE VII
KITTI-07 PROCESSING TIME (ms) ON INTEL PC AND NVIDIA TX1 WITH DIFFERENT VALUES OF *nframes*

$nframes$	1			2			3			4		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Intel (CPU)	33.646	58.254	78.156	36.512	65.689	82.241	38.989	76.263	96.358	43.989	81.124	98.416
Intel (CPU-GPU)	18.154	36.487	50.164	19.498	39.456	57.129	20.846	41.354	61.487	22.498	50.462	64.624
TX1	40.268	101.265	130.748	46.894	130.128	170.854	48.657	152.238	201.418	50.658	162.624	240.859

sequence is also the mean after 5 lauches. Table V represents the timing of each functional block in our proposal pipeline. The number of neighbor frames is 2. In the table V, Loop detection average is the sum of execution time divided by the total number of frames. However, this value can not be a good representation because execution time of Loop detection thread is not constant. In fact, with an overlapped frame or in case of not enough inliers, loop detection thread is terminated rapidly. Otherwise, when loop closure is reached, this thread becomes higher cost because the relative movement is estimated. To have a better representation, we presents "Loop detection cost-time" which is the mean time of loop detection thread when the movement estimation is performed.

We notice that Mapping thread and Loop detection thread are launched in parallel. Hence, per-frame time is only the sum of HOOFR extraction and Mapping (the most consuming thread). Moreover, when loop closure is valid, map correction inside Map Processing is launched in other thread so that it does not slow down the new frame acquisition. On PC Intel, without GPU implementation, the algorithm runs at ∼62 ms per frame. By offloading processing to GPU, we have a better performance when the mean of execution time of the whole algorithm is decreased to 36 ms per frame.

For Tegra TX1 embedded system, it is obvious that the processing task is much slower than that of the Intel PC because of many reasons: lower frequency of CPU and GPU, smaller cache memory resources and low number of CPU and GPU cores. On this platform, our partitioning exhibits a considerable performance where the algorithm takes in average ∼116 ms per frame. We also evaluated the timing performance of ORB-SLAM and table VI shows the mean per-frame timing comparison between our proposal (HOOFR SLAM) and ORB SLAM on two platforms using KITTI dataset. With Intel PC, the ORB execution time is approximately 69ms per frame (7 ms costly than CPU-only version or 32ms costly than

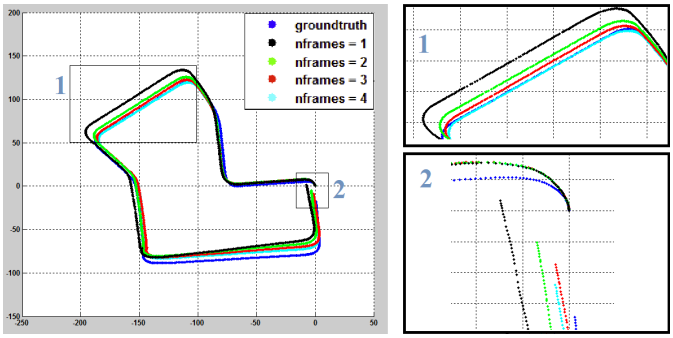


Fig. 9. KITTI-07 localization results using different values of *nframes*.

CPU-GPU version of our algorithm). On TX1 embedded platform, ORB-SLAM takes 190 ms per frame (53 ms costly than CPU version or 74 ms costly than GPU version of our proposal). Compared to ORB performance, our HOOFR algorithm exhibits a lighter processing task while maintaining a competitive accuracy. It will be an advantage when camera frequency becomes higher because HOOFR can handle more frames in this case. For fast camera moving, handling more frames allows avoiding severe change in camera position of each input frame.

To evaluate the timing in more details, we studied the timing and localization precision in terms of the number of neighbor frames (*nframes*). Table VII presents the minimum, the maximum and the mean of per-frame processing time on KITTI-07 when the *nframes* parameter changes from 1 to 4. For the powerful Intel PC, we still have a frame-rate running at less than 100ms when the *nframes* increases to 4 for both GPU and without GPU version. For TX1 embedded system, due to limited resources, processing time did not meet the frame-rate (10 Hz) performances. The variation of time in each frame is primarily as a consequence of the motion estimation step. In fact, in order to compute essential matrix from matching

set, this step uses RANSAC scheme which selects the subset by random choices and the proportion of inliers is not identical for different matching sets. Some of high proportion of inliers normally take less time to compute than that of low proportion. Besides, Figure 9 shows the effect of *nframes* on the localization result. Ground-truth is always presented by the blue curve. We notice that more we take into account the number of neighbor frames, more we get a higher localization precision. The explanation for this exhibition could be found at the features detection level. In fact, at some points in the trajectory, especially in turning scenarios, current frame contains less common points with nearest neighbor frame than with a further neighbor frame. Therefore, the motion estimation with further neighbor frame provides more confidence and has a higher weight. By integrating a more precise prediction in optimal pose extraction, we have a lower localization error.

VI. Conclusions

In this work, a novel estimation algorithm for feature-based stereo VSLAM has been presented. Our parallelized lightweight VSLAM framework was obtained as a result of a hardware-software mapping study addressing feature extraction, data processing, hardware building implementation and benchmarking. We referred to this approach as HOOFR SLAM since it integrates our previous work on HOOFR features [23]. This binary descriptor is employed for motion estimation and loop closure detection. Motion estimates are integrated over time following a hybrid filtering/key frame strategy. That is, position is estimated using a widowed weighted mean using previous neighbor frames. Weights are computed from inter-frame robust feature matching. A thorough experimental research was carried out on five well-known datasets (KITTI, Oxford, Malaga, MRT and St-Lucia) providing considerable localization results in terms of RMSE (around 1% of sequence dimension). Our real-time algorithm implementation on high performance Intel-based PC architecture processes frames at more than 20 Hz using KITTI dataset. On the Tegra TX1 embedded system, the processing time is close to real time performances with 6 fps running rate. The emergence of new heterogeneous CPU-GPU architectures such as Xavier Nvidia (8 Core ARM64 CPU, 512 Core Volta GPU) should help to embed the HOOFR SLAM algorithm with real-time constraints.

However, real-time execution of HOOFR SLAM on an embedded system that consumes few watts remains a perspective of this work. This can be achieved taking advantage of hardware accelerators such FPGAs. An important effort will be done for offloading HOOFR extraction and features matching steps on FPGA following a hardware-software co-design approach so as to achieve a real-time HOOFR SLAM System.

References

[1] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[2] L. Riazuelo, J. Civera, and J. M. M. M. Montiel, "C^2 TAM: A Cloud framework for cooperative tracking and mapping," *Robot. Auton. Syst.*, vol. 62, no. 4, pp. 401–413, 2014.

[3] G. Bresson, T. Féraud, R. Aufrère, P. Checchin, and R. Chapuis, "Real-time monocular SLAM with low memory requirements," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 1827–1839, Aug. 2015.

[4] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[5] G. Pascoe, W. Maddern, M. Tanner, P. Piniés, and P. Newman, "NID-SLAM: Robust monocular slam using normalised information distance," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 1435–1444.

[6] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May/Jun. 2014, pp. 15–22.

[7] A. J. Davison and D. W. Murray, "Mobile robot localisation using active vision," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 1998, pp. 809–825.

[8] A. J. Davison and D. W. Murray, "Simultaneous localization and map-building using active vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 865–880, Jul. 2002.

[9] A. J. Davison and N. Kita, "3D simultaneous localisation and map-building using active vision for a robot moving on undulating terrain," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Dec. 2001, p. I.

[10] L. M. Paz, P. Jensfelt, J. D. Tardos, and J. Neira, "EKF SLAM updates in O(n) with divide and conquer SLAM," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2007, pp. 1657–1663.

[11] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot, "Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association," *J. Mach. Learn. Res.*, vol. 4, no. 3, pp. 380–407, 2004.

[12] G. Grisetti, C. Stachniss, and W. Burgard, "Nonlinear constraint network optimization for efficient map learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 3, pp. 428–439, Sep. 2009.

[13] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, Feb. 2010.

[14] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Editors choice article: Visual SLAM: Why filter?" *Image Vis. Comput.*, vol. 30, no. 2, pp. 65–77, 2012.

[15] K. Konolige and M. Agrawal, "FrameSLAM: From bundle adjustment to real-time visual mapping," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1066–1077, Oct. 2008.

[16] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "RSLAM: A system for large-scale mapping in constant-time using stereo," *Int. J. Comput. Vis.*, vol. 94, no. 2, pp. 198–214, 2011.

[17] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "Stereo parallel tracking and mapping for robot localization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep./Oct. 2015, pp. 1373–1378.

[18] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct SLAM with stereo cameras," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep./Oct. 2015, pp. 1935–1942.

[19] R. Wang, M. Schwörer, and D. Cremers, "Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 3903–3911.

[20] D. Rodriguez-Losada, P. S. Segundo, M. Hernando, P. de la Puente, and A. Valero-Gomez, "GPU-mapping: Robotic map building with graphical multiprocessors," *IEEE Robot. Autom. Mag.*, vol. 20, no. 2, pp. 40–51, Jun. 2013.

[21] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM without a pose graph," in *Proc. Robot., Sci. Syst.*, 2015, pp. 1–9

[22] B. Vincke, A. Elouardi, and A. Lambert, "Real time simultaneous localization and mapping: Towards low-cost multiprocessor embedded systems," *EURASIP J. Embedded Syst.*, vol. 2012, p. 5, Dec. 2012.

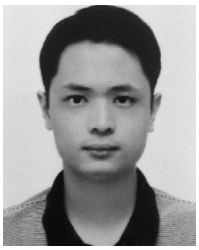
[23] D.-D. Nguyen, A. E. Ouardi, E. Aldea, and S. Bouaziz, "HOOFR: An enhanced bio-inspired feature extractor," in *Proc. 23rd Int. Conf. Pattern Recognit. (ICPR)*, Dec. 2016, pp. 2977–2982.

[24] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[25] E. Michaelsen, W. V. Hansen, M. Kirchhof, J. Meidow, and U. Stilla, "Estimating the essential matrix: GOODSAC versus RANSAC," in *Proc. Symp. ISPRS Commission III Photogramm. Comput. Vis. (PCV)*, Bonn, Germany, Sep. 2006.

[26] D.-D. Nguyen, A. E. Ouardi, and S. Bouaziz, "Enhanced bio-inspired feature extraction for embedded application," in *Proc. 14th Int. Conf. Control, Autom., Robot. Vis. (ICARCV)*, Nov. 2016, pp. 1–6.

[27] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *Proc. 13th Eur. Conf. Comput. Vis.* Zurich, Switzerland: Springer, Sep. 2014, pp. 834–849.
[28] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012.
[29] H. Lim, J. Lim, and H. J. Kim, "Real-time 6-DOF monocular visual SLAM in a large-scale environment," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May/Jun. 2014, pp. 1532–1539.
[30] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 756–770, Jun. 2004.
[31] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: Theory and experiment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 9, pp. 920–932, Sep. 1994.
[32] M. Cummins and P. Newman, "Highly scalable appearance-only SLAM—FAB-MAP 2.0," in *Proc. Robot., Sci. Syst.*, vol. 5, Seattle, CA, USA, 2009, pp. 1–8.
[33] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013.
[34] W. Maddern, G. Pascoe, C. Linegar, and P. Newman, "1 Year, 1000km: The oxford robotcar dataset," *Int. J. Robot. Res.*, vol. 36, no. 1, pp. 3–15, 2017.
[35] J.-L. Blanco-Claraco, F.-Á. Moreno-Dueñas, and J. González-Jiménez, "The Málaga urban dataset: High-rate stereo and LiDAR in a realistic urban scenario," *Int. J. Robot. Res.*, vol. 33, no. 2, pp. 207–214, 2014.
[36] F. Moosmann and C. Stiller, "Velodyne SLAM," in *Proc. IEEE Intell. Vehicles Symp.*, Baden-Baden, Germany, Jun. 2011, pp. 393–398.
[37] M. Warren, D. McKinnon, H. He, and B. Upcroft, "Unaided stereo vision based pose estimation," in *Proc. Australas. Conf. Robot. Automat.*, G. Wyeth and B. Upcroft, Ed. Brisbane, QLD, Australia: Australian Robotics and Automation Association, 2010, pp. 1–9.



Dai-Duong Nguyen received the degree in electrical engineering in 2014 (mention in industrial information) from the Hanoi University of Science and Technology, Hanoi, Vietnam, and the M.S. degree in information, system and technology from Paris-Sud University, Orsay, in 2015. He is currently pursuing the Ph.D. degree with MOSS Group, SATIE laboratory, Paris-Sud University. His research activities are focused on vision systems for SLAM applications.

Abdelhafid Elouardi received the M.S. degree from Pierre & Marie Curie University in 2001, and the Ph.D. degree in electronics from Paris-Sud University, France, in 2005. He was with Henri Poincaré University, Nancy, as a Researcher, from 2005 to 2006. He is currently an Associate Professor with Paris-Sud University. With the Embedded Systems Team of SATIE lab, his research interests include hardware-software co-design, evaluation and instrumentation of embedded systems, design of smart architectures for image and signal processing, and SLAM applications.

Sergio A. Rodriguez Florez received the M.S. and Ph.D. degrees from the University of Technology of Compiègne, France, in 2007 and 2009, respectively. Since 2011, he is an Associate Professor with Paris-Sud University. His research activities are focused on dynamic scene analysis through multi-modal perception intended to enhance intelligent transportation systems applications.

Samir Bouaziz received the Ph.D. degree in electronics from Paris Sud University, Orsay, France, 1992. He is currently a Full Professor with Paris-Sud University. His research focuses on hardware-software designs of embedded systems for autonomous vehicles and robots. His research is led by time constraints and complexity consideration for a good fit between hardware and algorithms, instrumentation and benchmark systems to understand human-vehicle interactions.