CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A LoCATe-based Visual Place Recognition System for Mobile Robotics and GPGPUs

L. Bampis[1]*, S. A. Chatzichristofis[2], C. Iakovidou[1], A. Amanatiadis[1], Y. S. Boutalis[1] and A. Gasteratos[1]

[1] *School of Engineering, Democritus University of Thrace, GR-67132, Xanthi, Greece*
[2] *Neapolis University, Department of Information Science, CY-8042, Paphos, Cyprus*

SUMMARY

In this paper, a novel visual Place Recognition (vPR) approach is evaluated based on a visual vocabulary of the Color and Edge Directivity Descriptor (CEDD) in order to address the loop closure detection task. Even though CEDD was initially designed so as to globally describe the color and texture information of an input image addressing Image Indexing and Retrieval tasks, its scalability on characterizing single feature points has already been proven. Thus, instead of using CEDD as a global descriptor, we adopt a bottom-up approach and utilize its localized version, Local Color And Texture dEscriptor (LoCATe), as an input to a state-of-the-art visual Place Recognition technique based on Visual Word Vectors. Also, we employ a parallel execution pipeline based on a previous work of ours using the well established GPGPU computing. Our experiments show that the usage of CEDD as a local descriptor produces high accuracy vPR results, while the parallelization employed allows for a real-time implementation even in the case of a low-cost mobile device. Copyright © 2010 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Visual Place Recognition; Image Description; Mobile Robotics; GPGPU Computing

1. INTRODUCTION

The subject of visual Place Recognition (vPR) refers to the task of a robot to recognize scenes/places that has encountered in the past using only visual sensors. Due to its generic nature, vPR can be applied on a variate of mobile robotic applications, one of which is the loop closure detection task. In the context of graph-based Simultaneous Localization and Mapping (SLAM), a loop closure detection engine is responsible for detecting revisited regions of a robot's trajectory and create additional edge constraints between the corresponding pose nodes [1, 2, 3]. Using this supplementary knowledge, the overall SLAM output can be further improved leading to a more accurate overall representation of the traversed route and the observed environment [4, 5].

Due to its demanding nature, the subject of loop closure detection has received great attention during the last decade leading to a vast variety of approaches. Considering the type of data that they associate, loop closure detection approaches can be divided into the following tree categories: map-to-map, map-to-image and image-to-image [6]. In the first category, revisited scenes are identified by finding correspondences between the appearance and the relative position of the detected features. In the second category, the correspondences are extracted between the current input image and a 3-dimensional representation of the already seen word. Finally, the methods that fall into the last category (also referred to as "appearance based place recognition") are proven to scale better on long trajectory cases and create associations only between the features that the input images contain.

*Correspondence to: lbampis@pme.duth.gr

In the general case of appearance based place recognition, the model of Bags of Visual Words (BVW) is applied. Local feature descriptors are extracted from every input image and converted into visual words quantizing the vectors' space. Then, each one of those words votes on the respective bin of a common histogram/visual-word-vector (VWV) creating a description footprint for each input frame. Thus, a loop closing pair of images can be distinguished by calculating vector distances between the currently obtained (query) VWV and all the previously acquired (database) ones. This technique was first applied in order to address the problem of Image Retrieval [7], yet in the case of loop closure detection, time proximity is also employed in order to assist image matches that persist over time.

Two of the greatest and most acknowledged examples in the field of appearance based place recognition are FAB-MAP [8] and its sparse approximation, FAB-MAP 2.0 [9]. Both of those methods were based on co-occurrence probabilities between the obtained visual words and provoked thought for a vast variety of other approaches. Angeli et al. in [10] based their description on two visual vocabularies (SIFT features and color histograms) taking into account the matching Bayesian probability from previously acquired images. Aiming to speedup the quantization of the feature description vectors, Schindler et al. [11] used a tree-based representation of the formulated visual vocabulary (vocabulary tree) and drastically reduced the computations required for identifying the corresponding VWVs. More recent techniques, instead of adopting a probabilistic method, enforced a temporal consistency constraint between the detected loop closing pairs of camera measurements. One representative example of this approach was described by Gálvez-López and Tardós in [12], with their algorithm DBoW2 [13]. Mur-Artal and Tardós [14], in a later work, made use of DBoW2 in order to recognize places, relocalize and detect loop closure events in a real-time key-frame SLAM system.

Recently, in order to provide execution time efficiency in the loop closure detection task, parallel programming have been applied based on GPGPU computing. In Collier et al. [15] work, features from both the image and the 3-dimensional measurements of a range sensor were utilized so as to detect revisited scenes. In their case, the GPU undertook the extraction of VD-LSD vectors using a parallel algorithm for describing feature points. Another vPR approach that utilizes a GPU for the calculations is described in [16], where the characterization of the individual frames was based on an average representation of local feature descriptors. In that case, the parallelization referred to the distance calculations in the feature vector space.

In the approaches described above, it is a common practice to utilize the functionality of traditional local feature point descriptors. Floating-point descriptors, like SIFT [17] or SURF [18], have proven to be very effective so as to characterize the content of a given feature point in a huge variety of applications. Moreover, with the aim to provide some efficiency to the calculations, great attention has also been given to the competencies of the more modern binary descriptors, e.g. ORB [19] or FREAK [20], which are more compact and faster to formulate and match. Those methods induce a bottom-up approach for the loop closure detection task since the created VWVs can be characterized as global description vectors obtained from the properties of local feature points.

Lately, the robotics' community turned its attention to the task of long-term visual navigation [21, 22] as well, which requires vPR methods capable of identifying revisited trajectory regions under extreme environmental/appearance changes. Those techniques need to deviate from the aforementioned bottom-up notion, sacrificing some of their invariance over the camera's viewpoint changes [23]. In general, methods that fall into this category holistically describe each image without considering the individual local features that compose it. A great example of such an approach was introduced by Arroyo et al. in [24]. Their system characterized sequences of illumination invariant images by using a concatenated and global version of the "Local Difference Binary" (LDB) descriptor [25]. The same sensitivity to the observation viewpoints can be found in another family of algorithms that base the image description on the application of Convolutional Neural Networks (CNNs) [26]. In the general case, a CNN trained for object detection is applied to the whole image, while the output of a specific convolutional layer is used as a description vector. Sünderhauf et al. [27] performed an evaluation of the AlexNet [28] network and tested each layer's robustness under appearance and viewpoint changes. The obtained results revealed that different layers perform better

DOI: 10.1002/cpe

for different kinds of changes eliminating the possibility of a holistic solution. To cope for that, the same group of authors later proposed a vPR system [26] that firstly detected visual landmarks in a given image and then used the output of a specific CNN layer (preserving invariance over the appearance changes) to describe them. Although this example provided some robustness over both appearance and viewpoint changes, it suffered in terms of computations due to the complexity of the adopted landmark detection algorithm. Finally, Arroyo et al. [29] proposed a CNN-based vPR system that simultaneously utilized the information from different convolutional layers by concatenating their output into a single description vector. This concatenation offers another way for introducing some invariance over both environmental and viewpoint changes at the expense of increasing the matching functionality's complexity. The authors in [29] reduced some of the extra computations by shortening and quantizing the description vector.

In this work, we are interested in addressing the loop closure detection subclass of the vPR problem. Our goal is to offer a real-time application capable of detecting revised scenes from a freely moving hand-held camera even in the case of a low power device. For these reasons, the choice of BVW was adopted as a more intertwined with the overall SLAM problem solution. By utilizing the VWV-based approach, our algorithm is also capable of producing the necessary for SLAM local feature descriptors and fundamental matrices undertaking some of the necessary odometry computations.

We propose for the first time the inclusion of an alternative description algorithm capable of characterizing both color and texture information of a given feature point, i.e. the "Color and Edge Directivity Descriptor" (CEDD) [30]. Even though this algorithm was initially designed so as to describe a given image as a whole, recently its capability of describing local feature points was proven as well. As shown in Iakovidou et al. work [31], the localized equivalent of CEDD, referred to as Local Color And Texture dEscriptor (LoCATe), outperformed the matching accuracy of many other descriptors, like SIFT or SURF, for Image Retrial purposes. In addition to the promising local description capabilities, CEDD also presents great parallelization properties and thus allows for a real-time implementation. In our previous work presented in [32], we evaluated a parallel execution pipeline, based on GPGPU computing, capable of producing global image description vectors of CEDD on at least 25 frames per second (fps), even in the case of a low-cost hardware setup. Extending our previous proposal, we alternate a state-of-the-art vPR method, namely DBoW2, and use LoCATe feature descriptors as an input for detecting loop closure events. Furthermore, we reformulate our GPGPU implementation in order to the extract local description vectors introducing an additional speedup. For our timing experiments, a mobile tablet device was used, provided by Google's Project Tango [33], proving the efficiency of our algorithm.

The rest of this paper is organized with the following structure. Section 2 is dedicated to the brief description of LoCATe, the local descriptor used by our method. In Section 3, the proposed vPR system is described in details, while Section 4 apposes some of the GPGPU terminology that we will refer to in the rest of our proposal. Section 5 explains in details the parallelization techniques used by our vPR algorithm. Section 6 contains the timing and accuracy evaluation of the presented approach and finally, Section 7 draws our final conclusions and plans for future work.

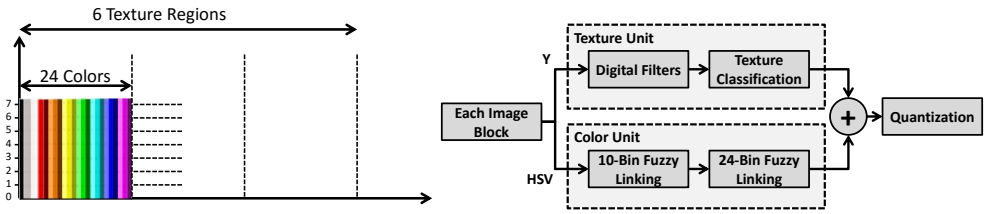
2. LOCATE: THE LOCALIZED VERSION OF CEDD

This section provides a brief presentation of the structural elements of the LoCATe. For a more detailed description, kindly refer to [31]. LoCATe is the localized extension of CEDD, generated through the combination of a local-point detector and the CEDD global descriptor.

2.1. The Detection Stage

Given an input image, the SURF detector is employed to locate points of interest. The SURF detector works on achromatic information and uses the determinant of the Hessian (approximated using a set of box-type filters) to detect both the location and the scale of blob-like structures. After locating a point using the detector, a square patch is marked on the image around it, whose size is determined

DOI: 10.1002/cpe



(a) Descriptor's structure. (b) Implementation flowchart.

Figure 1. LoCATe overview.

by the scale it was detected in by SURF. By the end of the detection stage, the image is represented by patches varying in size, on which the CEDD descriptor is applied, as if they were standalone images.

2.2. *The Description Stage*

LoCATe is capable of quantifying both color and texture attributes of a given patch. The original CEDD implementation demands a division of the image patch into 40×40 Blocks of at least 2×2 pixels each. However, the latest version of CEDD[†] adapts to the description of smaller sized image patches and defines a minimum of 20×20 Block division of at least 2×2 pixels each. Thus, in our case, 400 Blocks are produced for each detected image patch and forwarded to the two main information extraction units, i.e. the Color and the Texture Extraction Units. The corresponding information vectors are then combined and quantized into 8 predefined levels in order to produce one final local descriptor, as the one presented in Fig. 1a.

2.2.1. Color Extraction Unit: As depicted in Fig. 1b, each detected patch enters the Color Unit after its RGB (Red, Green, Blue) values are converted into the HSV (Hue, Saturation, Value) color space. Then, a two-staged fuzzy system is employed to produce a Fuzzy Linking histogram. Linking is defined as the combination of more than one histograms to a single one [34]. The first stage of the fuzzy system has the three mean HSV channels of an Image-Block as inputs and forms a 10-bins histogram output. The three inputs of the fuzzy system are described as follows: H is divided into 8 fuzzy areas, S is divided into 2 fuzzy regions, while the channel V is divided into 3 areas (kindly refer to Fig. 2). The output of the fuzzy system is enabled by a set of 20 rules and returns a crisp value ranging from 0 to 1 (TSK like fuzzy system) to produce the 10-bins first-stage histogram. The first three bins represent Black, Grey and White, respectively, while the rest seven bins represent a preset color each.

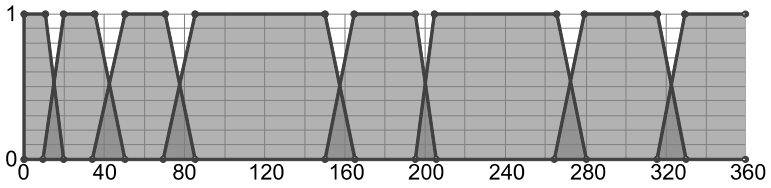
The second-stage fuzzy linking system (TSK) is responsible for adding the brightness value to the seven colors (Black, Grey and White are not computed). Again the S and V mean values of an image patch's Block become fuzzy inputs, as illustrated in Fig. 3. The output is a 3 bin histogram of crisp values, indicating if the color will be characterized as light, normal or dark-hued.

The two outputs (first and second stage histograms) are combined and the final 24-bin Color Histogram is produced. Each bin represents a color as follows: (0) Black, (1) Grey, (2) White, (3) Dark Red, (4) Red, (5) Light Red, (6) Dark Orange, (7) Orange, (8) Light Orange, (9) Dark Yellow, (10) Yellow, (11) Light Yellow, (12) Dark Green, (13) Green, (14) Light Green, (15) Dark Cyan, (16) Cyan, (17) Light Cyan, (18) Dark Blue, (19) Blue, (20) Light Blue, (21) Dark Magenta, (22) Magenta, (23) Light Magenta.

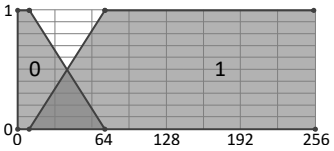
2.2.2. Texture Extraction Unit: In parallel with the Color Unit, image patch's Blocks enter the Texture Unit, after being converted to the grayscale color space. For the extraction of the texture information, the method employs the five digital filters proposed by the MPEG-7 Edge Histogram

[†]The latest version of CEDD can be found in http://tinyurl.com/CEDD-Descriptor.

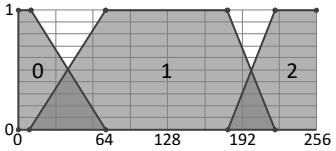
DOI: 10.1002/cpe



(a) H channel's membership function.

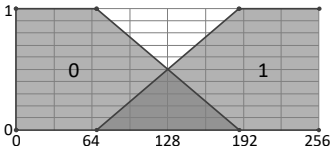


(b) S channel's membership function.

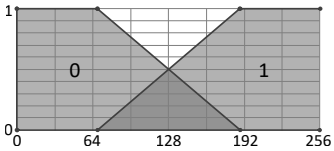


(c) V channel's membership function.

Figure 2. Membership functions used in the first stage of the fuzzy system [34].



(a) S channel's membership function.



(b) V channel's membership function.

Figure 3. Membership Functions used in the second stage of the fuzzy system [34].

Descriptor-EHD [35] which represent five broadly grouped edge types: vertical, horizontal, 45 diagonal, 135 diagonal and isotropic (non-directional) as shown in Fig. 4a, along with an additional Non-Edge filter. In order to employ the filters, each Block must be subdivided into four Sub-Blocks. The value representing each Sub-Block is the mean value of the luminosity (Y) of the pixels consisting the Sub-Block.

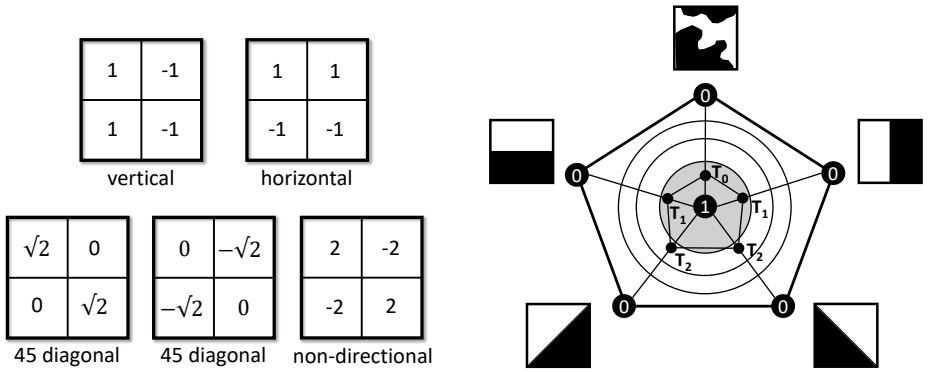
The digital filters are applied and the obtained responses became inputs to the fuzzy mapping scheme, illustrated in Fig. 4b. In its essence, this mapping system is responsible for indicating which kinds of edges are present for every Block. Please note that more than one edge types can be simultaneously present.

The normalized maximum responses (edge magnitudes) from the applied filters (per Block) are placed in the heuristic pentagon diagram (Fig. 4b). Each value is placed along the line that pertains to the filter it emerged from. If that value is greater than the corresponding line's threshold, the Block is classified in the respective type of edge. If none of the five thresholds are met, the Block is categorized as Non-Edge.

The Texture Unit produces a 6-bin vector output for each Block. Every bin represents one of the five employed textures, while the first bin represents the Non-Edge case. When an edge type was found present in a Block the corresponding bin is marked with "1". Otherwise, it is marked as "0", producing the binary texture vector.

2.2.3. Producing the LoCATe descriptor: When the 24-bins Color Histogram and the 6-bins texture vector have been calculated, the two are combined and a 144-bins vector for every image patch's Block is generated as follows: the bins are divided into six regions (that represent a different texture) of 24-bins each. According to the Block's texture vector, and for those of its bins that were marked as "1", the respective region in the 144-bins vector is filled with the calculated 24-bins Color Histogram. Then, all Block descriptors are added to form the patch's descriptor. This vector is normalized and quantized into 8 predefined levels. On completion, the LoCATe descriptor has been

DOI: 10.1002/cpe



(a) Filter Coefficients for Edge Detection.

(b) Edge Type Diagram (Heuristic Pentagon Diagram).

Figure 4. Edge Histogram Descriptor-EHD.

formed and characterizes the visual content of the corresponding patch in a compact and distinct fashion.

3. VISUAL PLACE RECOGNITION: LOCATE-BASED DBOW2

As mentioned before, in this paper we utilize a bottom-up image description approach in order to address the loop closure detection problem. Thus, we chose the well-established algorithm of DBoW2 and we replace its binary visual vocabulary with a LoCATe-based one. The main objective of DBoW2 is to create a VWV for every input image using a tree-based BVW of $L = 6$ levels and $K = 10$ branches per node. In [12] an off-line training step is necessary in order to formulate the required vocabulary tree using the binary description of BRIEF [36]. With the aim to change the local feature description, we are obligated to alternate this off-line step as well. In this work, the most prominent 300 feature points from each one of the 30K indoor and outdoor samples of Bovisa 2008-09-01 [37] dataset, indicated by the SURF detector, are described using LoCATe. This generic set of descriptors is used as an input to a k-means hierarchical clustering with k-means++ seeding [38]. The leaf nodes of the created tree represent the final vocabulary with a total size of $W = K^L$ visual words.

Given an input frame I, the strongest 512 SURF feature points are extracted (this number will be rationalized in the sections bellow) and converted into description vectors using LoCATe. Using the tree structure of our visual vocabulary, each one of the obtained local descriptors is converted into a visual word using only $K * L$ comparisons. With the new set of 512 visual words, each frame is assigned with one global image descriptor (VWV), the respective bins of which denote a weighted existence (or not) for each member of the visual vocabulary. This vector ($\hat{\boldsymbol{v}}^{(I)} = [v_1^{(I)}, v_2^{(I)}, ..., v_i^{(I)}, ..., v_W^{(I)}]$) is created using the "term frequency-inverse document frequency" (tf-idf) [39] using:

$$v_i^{(I)} = \frac{N_i^{(I)}}{N^{(I)}} \log \frac{N^{(D)}}{N_i} \tag{1}$$

where $N_i^{(I)}$ represents the number of occurrences of the visual word i in image I, $N^{(I)}$ the total number of visual words in image I, $N^{(D)}$ the total number of visual words in the training dataset and finally N_i represents the total number of occurrences of the i-th word in the training dataset.

After the extraction of the VWVs, DBoW2 asserts a loop closure when a set of requirements are met, as described in the rest of the paragraph. Firstly, normalized $L1$-scores (n-scores) between the query ($\hat{\boldsymbol{v}}_q^{(I)}$) and all the previously acquired database images ($\hat{\boldsymbol{v}}_d^{(I)}$) that contain some common

DOI: 10.1002/cpe

words are calculated using:

$$L_1\left(\hat{\boldsymbol{v}}_q^{(I)}, \hat{\boldsymbol{v}}_d^{(I)}\right) = 1 - 0.5\left|\frac{\hat{\boldsymbol{v}}_q^{(I)}}{\left|\hat{\boldsymbol{v}}_q^{(I)}\right|} - \frac{\hat{\boldsymbol{v}}_d^{(I)}}{\left|\hat{\boldsymbol{v}}_d^{(I)}\right|}\right| \tag{2}$$

$$n\left(\hat{\boldsymbol{v}}_q^{(I)}, \hat{\boldsymbol{v}}_d^{(I)}\right) = \frac{L_1\left(\hat{\boldsymbol{v}}_q^{(I)}, \hat{\boldsymbol{v}}_d^{(I)}\right)}{L_1\left(\hat{\boldsymbol{v}}_q^{(I)}, \hat{\boldsymbol{v}}_{q-1}^{(I)}\right)} \tag{3}$$

where $\hat{\boldsymbol{v}}_{q-1}^{(I)}$ denotes the VWV that corresponds to the most similar with query image one can find in the database, i.e. the previous one. The database images achieving higher scores than a predefined threshold th_n form a subset of matching candidates. This subset is later divided into groups based on the images' time-intervals, forming islands of images. Islands are assigned with a new score created by the accumulation of their members' ones, while the highest scoring island is considered to contain the possible image candidates for loop closure. As in [40], a temporal consistency check is also applied, which eliminates matches between query images and islands not being consistent for at least k time-intervals. This essentially means that an image-to-island match (image i with island j) needs to persist in appearing for the last k time-intervals (images from $i-k$ to i with islands from $j-k$ to j), with the islands containing some overlapping regions. Lastly, a geometrical constraint needs to be checked so as to accept a loop closure event. A match between the query image and a member of the chosen island will be accepted if more inliers than a minimum threshold (th_R) can be found during the estimation of the fundamental matrix using a RANSAC scheme. In order to provide computational efficiency direct indexing is applied, which associates the features' corresponding visual words to the tree's parent nodes. This indexing reduces the candidates of the feature-to-feature matching from the two images, since only features that share the same parent, at a certain level of the tree ($l = 4$), are going to be checked. For a more detailed explanation of the DBoW2 algorithm please refer to the description of the original paper [12].

4. GPGPU TERMINOLOGY

In this work, we utilize CUDA [41], the general purpose parallel computing architecture introduced by NVIDIA, as a means for assigning calculations to the GPU. Thus in this section, a small description of the terms used in the rest of the paper is provided.

Concerning the two main processing units of a computing setup, the CPU is referred to as *Host* while the GPU as *Device*. The GPU unit contains a number of multi-cores or multiprocessors (*MPs*). Each *MP* complies with a "Single Instruction on Multiple Threads" (*SIMT*) architecture[‡], meaning that the respective sub-processors are restricted on executing the same instruction at every clock cycle. Violating the aforementioned constraint causes a serialization of the computations leaving some sub-processors idle.

The term *Grid* refers to a set of *Thread Blocks*, while every *Thread Block* is a set of *Threads*. From the software point of view, *Threads* are responsible for providing the instructions to the sub-processors of a given *MP*. It is important to also note here that the *Grids* and the respective *Thread Blocks* can be mentally arrange in one, two or three dimensional structures (x, xy or xyz respectively) to make the coding design straightforward. The *Host* is responsible for determining the arraignment and size of the respective *Grids*/*Thread Blocks* that are going to execute a specific function (*Kernel*) on the *Device*, while the maximum number of *Thread Blocks* in a *Grid* and *Threads* per *Thread Block* is defined by the GPU model. Given a specified *Kernel*, the respective *Threads* are combined into groups (*Warps*) of 32 members. A *Warp* is the fundamental execution unit that performs one instruction over all its *Thread*-members, thus it is a common practice to create *Thread Blocks* of size $r \times 32$ ($r \in \mathbb{N}^+$) in order to occupy the whole GPU.

[‡] *SIMT* is an extension of the more traditional "Single Instruction on Multiple Data" (*SIMD*) architecture.

DOI: 10.1002/cpe

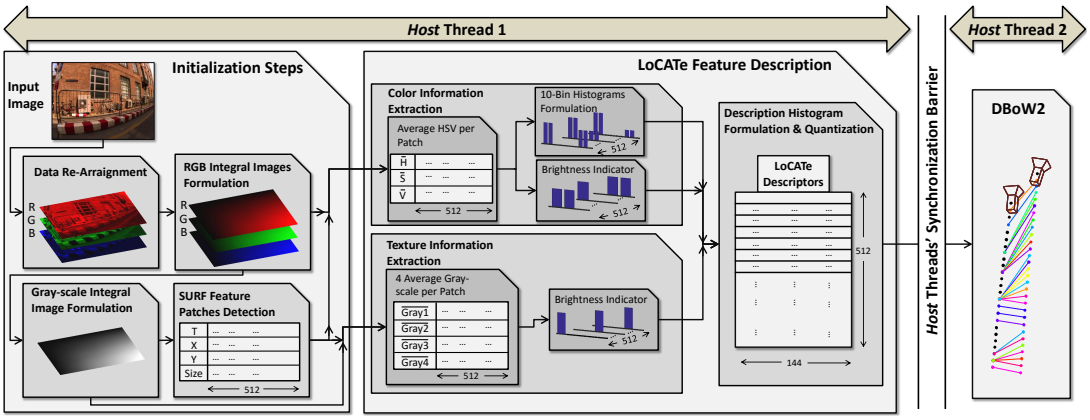


Figure 5. The GPGPU based pipeline of the proposed vPR system. The approach can be divided into three main processing components: (Left) the initialization steps, (Middle) the LoCATe descriptors calculation and (Right) the loop closure detection algorithm.

A GPU has an efficient memory architecture as well, divided in global and local memories. The *Global* memory can be accessed and shared among every *Thread* that is running on the *Device*. It is also the only memory accessible by the *Host*, while *read* operations from *Threads* with indexes that follow the memory alignment guidelines can be coalesced. *Texture* and *Constant* memories are also parts of the *Global* memory (can be accessed by every *Thread*) and are cached. On the one hand, *Texture* memory is read-only by the *Device* and optimized for 2-dimensional accesses. On the other hand *Constant* memory is in general slower but does not introduce extra latencies when the individual *Threads* access the same addresses simultaneously. *Threads* that are members of the shame *Thread Block* share access to common *Shared* memory chunks which are roughly 100 times faster than the *Global* one. Finally *Registers* are privet for each *Thread* and presents the smallest accessing latency.

The above terms are going to be used in the description of our GPGPU-based implementation while for a more detailed description one could refer to [42, 43].

5. GPGPU IMPLEMENTATION

In this section, our parallel pipeline is described in details. The main objective of the presented work is the GPGPU implementation of the proposed vPR system's back-end, e.i. the local feature detection and description. In our previous work [32], a parallel pipeline of CEDD (the global description ancestor of LoCATe) was described addressing the problem of real-time image indexing. However, here we present a new and improved version taking into account the new characteristics and implementation properties that the features' locality induces.

This section additionally justifies our selection of LoCATe description. Considering the required processing steps of a state-of-the-art parallel SURF features extraction method, the most computationally demanding procedures that LoCATe requires can be counted as pre-calculated, as to be discussed in the following subsections.

Figure 5 presents the main processing steps of our proposed implementation, all of them efficiently design around the architectural principles of GPGPU programming. For a given input image, some initialization pre-processing steps are required in order for our proposal to be generic and applicable to any GPU model (viz. Data Arraignment, RGB Integral Images Formulation, Gray-scale Integral Image Formulation and SURF Features Detection). The main difference between [32] and this work is that the image patches to be described are no longer of a stable size. Instead, their pixel-members multitude varies with respect to the scale that SURF detected each local feature. This new property arises a challenge regarding the individual *Threads'* workload since for an efficient implementation, the calculations are required to be the same, regardless the size of every described

DOI: 10.1002/cpe

patch. Thus, we propose an architecture based on Integral Images aiming to efficiently provide the SURF feature points detections as well as the LoCATe description vectors.

5.1. Initialization Steps

5.1.1. Data Re-Arrangement: Given an input image, the first step of our algorithm refers to the re-arrangement of the pixel values in such a way so as to be efficiently accessible by the *Device*. Thus, a buffer in the *Global* memory is allocated with the same size as the input images' resolution multiplied by three ($N_{rgb} = W \times H \times 3$, W being the frame's width, H its height and three color channels per pixel). Since we are interested in accessing each individual color channel independently from the other two, we need to deviate from the traditional image storage format which uses a color-major arrangement. Looking at an image as a 3-dimensional array with storage order [color, width, height], the required rearrangement procedure is the equivalent of a permutation that exchanges the first and the last dimension ([width, height, color]). For this reason, the aforementioned buffer is bound by a 3-dimensional *Texture* memory configured so as to provide coalesced accesses when neighboring pixel values of the same color channel are required. This *Texture* memory is bound only once while the *Host* is responsible for updating the buffer's values for each different input image in the stream. The aforementioned process is performed by a *Kernel* of N_{rgb} *Threads*, while its output is three sequential buffers (B_r, B_g and B_b) containing the red, green and blue channels respectively.

5.1.2. RGB Integral Images Formulation: In order to provide a generic solution, the required by LoCATe average color values (Color Extraction Unit) of a given patch are calculated through Integral Images. As a reminder, each value of an Integral one-channeled Image I^S is calculated using:

$$I^S(i,j) = \sum_{u=0}^{i} \sum_{v=0}^{j} I(u,v) \tag{4}$$

where I denotes the original one-channeled image. Our algorithm needs to calculate an Integral product from each color channel and here we present the procedure of creating one of them. The rest of the channels' Integral products can be formulated with the exact same steps.

To efficiently acquire the I_c^S of a channel I_c ($c \in \{r,g,b\}$) we make use of the parallel algorithm proposed by Timothy et al. [44], which refers to the 2-dimensional extension of Blelloch's well-established method for prefix summations [45]. As in [44] we utilize three pyramids during the up-sweep step (${}^{up}P_B$, ${}^{up}P_H$ and ${}^{up}P_V$) and one during the down-sweep (${}^{dn}P$) in order to formulate the prefix sum along both channel's dimensions simultaneously. The up-sweep steps are governed by the following equations:

$$\begin{aligned} {}^{up}P_B^{(k)}(i,j) = {}^{up}P_B^{(k-1)}(2i,2j) + {}^{up}P_B^{(k-1)}(2i+1,2j) + \\ + {}^{up}P_B^{(k-1)}(2i,2j+1) + {}^{up}P_B^{(k-1)}(2i+1,2j+1) \end{aligned} \tag{5}$$

$${}^{up}P_H^{(k)}(i,j) = {}^{up}P_B^{(k-1)}(2i,2j) + {}^{up}P_B^{(k-1)}(2i+1,2j) \tag{6}$$

$${}^{up}P_V^{(k)}(i,j) = {}^{up}P_B^{(k-1)}(2i,2j) + {}^{up}P_B^{(k-1)}(2i,2j+1) \tag{7}$$

where k refers to the corresponding pyramid's level and i,j to the indexes in axes x (horizontal) and y (vertical) respectively. ${}^{up}P_B^{(k)}$ contains the 2-dimensional sum of neighboring values at each layer k, while the first pyramid level (${}^{up}P_B^{(0)}$) is initialized with the input image channel I_c. The last two pyramids are used to compute the row-wise and column-wise summations via:

$${}^{up}X^{(k)}(i,j) = \sum_{u=0}^{i-1} {}^{up}P_H^{(k)}(u,j) \tag{8}$$

$${}^{up}Y^{(k)}(i,j) = \sum_{v=0}^{j-1} {}^{up}P_V^{(k)}(i,v) \tag{9}$$

DOI: 10.1002/cpe

After reaching the highest pyramid level, the down-sweep steps of ${}^{dn}P$ pyramid are calculated by:

$$ {}^{dn}P^{(k)}(i,j) = \begin{cases} {}^{dn}P^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor), & i\ even, j\ even \\ {}^{dn}P^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + {}^{up}Y^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor), & i\ odd, j\ even \\ {}^{dn}P^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + {}^{up}X^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor), & i\ even, j\ odd \\ {}^{dn}P^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + {}^{up}X^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + \\ +{}^{up}Y^{(k+1)}(\lfloor i/2 \rfloor, \lfloor j/2 \rfloor) + {}^{up}P_B^{(k)}(i-1, j-1), & i\ odd, j\ odd \end{cases} \quad (10) $$

where $\lfloor x \rfloor$ denotes the *floor* function mapping each value x to the smallest following integer. Starting from the highest layer and moving towards layer 1, we end up with ${}^{dn}P^{(1)}$ containing the corresponding channel's integral image.

The above set of equations comply with the computational guidelines of the GPGPU architecture and require two layers of *Texture* memories per input channel I_c in order to provide the respective I_c^S. Those *Textures* are binded on *Global* buffers only once at the beginning of our application and enable a ping-pong access procedure avoiding the problem of read-after-write [46]. One 3-dimensional *Texture* (*RGB Texture*) is binded on the B_r, B_g and B_b buffers and plays the role of the first *Texture* layer for each individual channel. The final Integral RGB Images, formulated by the above steps, are finally stored in the second 3-dimensional layer ($RGB\_Integral\_Texture$) with the arraignment: [width, height, color].

5.1.3. Gray-scale Integral Image Formulation: The calculation of the gray-scale Integral equivalent of a given input image will highly benefit the rest of our architecture. More specifically, both the SURF detector (application of the second-order Gaussian filter Kernel) and the LoCATe descriptor (Texture Extraction Unit) require the calculation of the sum/average of the images' patches in the gray-scale domain (I_{gray}). Since the conversion between a colored image I_{rgb} and the respective I_{gray} one is described by a linear transformation, the Integral gray-scale product can be formulated directly from the I_c^S channels using:

$$ I_{gray}^S(i,j) = 0.299 * I_r^S(i,j) + 0.587 * I_g^S(i,j) + 0.114 * I_b^S(i,j) \quad (11) $$

The created I_{gray}^S is once again stored in a dedicated *Global* buffer binded by a 2-dimensional *Texture* cache ($Gray\_Integral\_Texture$) using $N_{gray} = W \times H$ *Threads*.

5.1.4. SURF Feature Patches Detection: The SURF feature detector is based on the convolution of a second-ordered Gaussian filter with the pixel values of a given gray-scale image. This filter is approximated by a Gaussian box kernel and evaluated at a variety of different scale levels achieving scale invariance. The responses of this convolution are then thresholded returning the feature point detection on a scale-space coordination system $\hat{\boldsymbol{p}} = [i, j, s]$, where s refers to the scale corresponding to the maximum response. For a more detailed description of the method, please refer to the original paper of SURF [18].

Having the Integral image I_{gray} pre-calculated the detection of the SURF feature points can be very cost-effective since the Gaussian filtering can be applied on different scale levels with negligible computational steps. For this *Kernel* we utilize the implantation described by [47] since we verified its efficiency both in terms of computations and achieved accuracy. As described in [48], the SURF algorithm has many parallelization potentials which can be categorized as follows: (a) Block-Level Parallelization, (b) Scale-Level Parallelization and (c) Pipeline Parallelization. Regarding the first category, the parallelization is achieved by treating every input image as a set of individual image-blocks and concurrently checking each one of them for containing a feature point. In the second approach, each different scaling level of SURF is computed in parallel, while the last category refers to a pipeline scheme that separates and simultaneously executes the stages of detection and description. Looking for a scalable solution, in [47] the fist parallelization approach was adopted since its architectural characteristics seamlessly comply with the implementation guidelines of a GPGPU.

DOI: 10.1002/cpe

As an output of this *Kernel*, we obtain 4 vectors stored in the *Global* memory containing the detected feature points' responses (T), their position along x and y axes, as well as the corresponding patch sizes. These vectors are then sorted with respect to the achieved responses and the 512 strongest ones are used for the rest of the pipeline. This sample size of local features is selected as the closest to the one used by DBoW2 (300 per input frame) while still being an integral multiple of the standard *Warp* size (i.e. 32).

5.2. LoCATe Features Description

The LoCATe descriptor characterizes image patches, the size of which is provided by the feature detector. As described in Section 2, LoCATe is fundamentally based on the description of two image properties: color and texture. Those properties are extracted from each individual feature patch's Block formulating two main description vectors (*24-Bin Color Vector* and *6-bin Texture Vector*). Those vectors are firstly combined in order to produce a Block description vector and later to create the patch's description histogram. With the aim to provide more efficiency to the computations, we slightly change the description procedure and we treat each image patch as a single Block, drastically reducing the computational complexity and omitting the Blocks' histograms addition and normalization procedures of the original implementation. As it was proven by our previous work [32], the most computationally demanding part of the process is the calculation of the average values between each one of the color and grayscale channels. Using the initialization steps described above, those values can be computed very efficiently justifying the selected approach.

5.2.1. Color Information Extraction: In order to formulate the required Color Histograms, the average HSV values for each patch is required. Thus, we first need to calculate the sums between every color channel in the RGB color-space, find the respective average values and then convert them to the HSV domain. Using one *Thread* for each feature point, the Color Information Extraction *Kernel* can be computed by Algorithm 1. Note that $\gg$ denotes the bitwise "shift-right" operation. The algorithm makes perfect use of the created RGB Integral Images downgrading one of the most computationally demanding procedures of LoCATe to some few operations, all of them executed in parallel. Note that all *Global* memory accesses are aligned with the *Threads*' indexes, while the 2D-neighboring memory *read* operations are greatly benefited by the offered caching since the $RGB\_Integral\_Texture$ was utilized.

With the average HSV values per feature patch in hand, we now proceed to the formulation of the Color Histograms. For every LoCATe descriptor two kinds of vectors are required, one referring to the absolute color information and one to the brightness information. Both of those vectors are obtained based on the same notion of Fuzzy Linking and thus they are addressed by a common solution. We follow our previous proposal presented in [32] and we make use of a Parallel Participation Identifier (PPI) which is capable of determining the membership (or not) of a given value to some specific sub-region of a range in constant-time. Assuming a value range $\mathbb{V} = [v_0, v_e]$ and m sub-regions R_j, $j \in [0, m-1]$, the problem in hand refers to the association of an input query value q with one of the regions R_j. The serialized version of a Participation Identifier is obliged to compare the value of q with each one of the sub-regions' limits one after another in order to determine its membership. On the contrary PPI make these checks in parallel by assigning each sub-region to one individual *Thread*. Thus, only two checks need to be performed by each *Thread* (lower and upper bound per sub-region) regardless the multitude of sub-regions. The output of a PPI engine for a given patch p_i is a column vector ${}^f\hat{\boldsymbol{v}}^i$ of size m, whose values (${}^f v^i_j$) represent the participation of q in R_j as defined by the corresponding membership functions. The sub-regions' limits are stored in the *Constant* memory allowing different *Threads* to access the same values simultaneously. The two instances of PPI (color and brightness) are executed using two asynchronous *Kernels*. On the one hand, the fuzzy linker producing the absolute color information histogram contains 48 possible sub-region participations (8 for H, 2 for S and 3 for V) and thus a total of $[512 \times 48]$ *Threads* are utilized. On the other hand, for the case of the brightness histogram, the possible participation sub-regions are 4 (2 for S and 2 for V) meaning that $[512 \times 4]$ *Threads* are required. At this point our choice for retaining the most prominent 512 SURF feature points is clarified. Since each one of

DOI: 10.1002/cpe

Algorithm 1 The Color Averaging *Kernel*.

Require: $RGB\_Integral\_Texture[\ ,\ ,\ ,]$: 3D *Texture* binded on RGB Integral Images
Require: $X[\ ]$: X axis coordinates per feature point
Require: $Y[\ ]$: Y axis coordinates per feature point
Require: $Size[\ ]$: sizes per feature point patch
Output: $avgH\_vector$: average H values per feature point patch
Output: $avgS\_vector$: average S values per feature point patch
Output: $avgV\_vector$: average V values per feature point patch

1: **for all** $threadIDx.x, x \in [0, 511] \cap \mathbb{N}$ **do in parallel**
2: $x = X[threadIDx.x]$
3: $y = Y[threadIDx.x]$
4: $size = Size[threadIDx.x]$
5: $size\_div = size \gg 1$
6: $sumR = getSum\,(x, y, size\_div, 0)$
7: $sumG = getSum\,(x, y, size\_div, 1)$
8: $sumB = getSum\,(x, y, size\_div, 2)$
9: $size\_sq = size * size$
10: $avgR = sumR/size\_sq$
11: $avgG = sumG/size\_sq$
12: $avgB = sumB/size\_sq$
13: $[avgH, avgS, avgV] = rgb2hsv\,(avgR, avgG, avgB)$
14: $avgH\_vector[threadIDx.x] = avgH$
15: $avgS\_vector[threadIDx.x] = avgS$
16: $avgV\_vector[threadIDx.x] = avgV$
17: **end for**

Function $getSum\,(x, y, size\_div, color)$

1: $sum = RGB\_Integral\_Texture[x + size\_div, y + size\_div, color] +$
$+ RGB\_Integral\_Texture[x - size\_div, y - size\_div, color] -$
$- RGB\_Integral\_Texture[x + size\_div, y - size\_div, color] -$
$- RGB\_Integral\_Texture[x - size\_div, y + size\_div, color]$
2: **return** sum

the aforementioned *Kernels* is executed by a set of *Threads* whose multitude is an integral multiple of 512, the whole device is occupied utilizing all the available resources. Slightly alternating the original approach, the computed from PPI vectors (${}^{c}\hat{\boldsymbol{v}}^{i}$ for the color and ${}^{b}\hat{\boldsymbol{v}}^{i}$ for the brightness information) are not combined in this step so as to create the *24-Bin Color Vector*, since we found that this procedure can be included in the histogram quantization unit with less computational steps, as to be described in the following subsections. Finally, note that the output storing format of those vectors can be visualized with the following row-major arrays:

$$ {}^{c}\hat{\boldsymbol{v}} = \left[\ {}^{c}\hat{\boldsymbol{v}}^{0}\ \middle|\ {}^{c}\hat{\boldsymbol{v}}^{1}\ \middle|\ ...\ \middle|\ {}^{c}\hat{\boldsymbol{v}}^{511}\ \right] \qquad {}^{b}\hat{\boldsymbol{v}} = \left[\ {}^{b}\hat{\boldsymbol{v}}^{0}\ \middle|\ {}^{b}\hat{\boldsymbol{v}}^{1}\ \middle|\ ...\ \middle|\ {}^{b}\hat{\boldsymbol{v}}^{511}\ \right] \tag{12} $$

, meaning that for both cases, the sequentially stored values are those of ${}^{f}v_{j}^{i}$, ${}^{f}v_{j}^{i+1}$, ${}^{f}v_{j}^{i+2}$, etc ($f \in \{c, b\}$). This feature is induced by the *Threads* memory accessing pattern (sequentially indexed *Threads* is preferred to access sequential memory addresses) and greatly reduces the *write* operations' latency, even though it deviates from a straightforward notion.

5.2.2. Texture Information Extraction: The second main description step of LoCATe refers to the extraction of texture information from the detected feature patches. To that end, LoCATe divides

DOI: 10.1002/cpe

Algorithm 2 The Gray-scale Averaging *Kernel*.

Define: $offsetX[\,] = \{0, 1, 0, 1, -1, 0, -1, 0, 0, 1, 0, 1, -1, 0, -1, 0\}$
Define: $offsetY[\,] = \{0, 0, 1, 1, -1, -1, 0, 0, -1, -1, 0, 0, 0, 0, 1, 1\}$
Require: $Gray\_Integral\_Texture[\,,\,]$: 2D *Texture* binded on the gray-scale Integral Image
Require: $X[\,]$: X axis coordinates per feature point
Require: $Y[\,]$: Y axis coordinates per feature point
Require: $Size[\,]$: sizes per feature point patch
Output: $avgG\_vector$: average gray-scale values per feature point patch

1: **for all** $threadIDx.x/y, x \in [0, 511] \cap \mathbb{N}, y \in [0, 3] \cap \mathbb{N}$ **do in parallel**
2: $x = X[threadIDx.x]$
3: $y = Y[threadIDx.x]$
4: $size = Size[threadIDx.x]$
5: $size\_div = size \gg 1$
6: $sumG = RGB\_Integral\_Texture[x + size\_div * offsetX[threadIDx.y], y + size\_div * offsetY[threadIDx.y]]$
7: $sumG = +RGB\_Integral\_Texture[x + size\_div * offsetX[threadIDx.y + 4], y + size\_div * offsetY[threadIDx.y + 4]]$
8: $sumG = -RGB\_Integral\_Texture[x + size\_div * offsetX[threadIDx.y + 8], y + size\_div * offsetY[threadIDx.y + 8]]$
9: $sumG = -RGB\_Integral\_Texture[x + size\_div * offsetX[threadIDx.y + 12], y + size\_div * offsetY[threadIDx.y + 12]]$
10: $size\_sq = size * size$
11: $avgG = sumG/size\_sq$
12: $avgG\_vector[512 * threadIDx.y + threadIDx.x] = avgG$
13: **end for**

the patches (p_i) into 4 equal Sub-Blocks (B_j^i, $i \in [0, 511]$ and $j \in [0, 3]$) and calculates the average gray-scale value for each one of them ($\bar{B}_j^i$). Given the initialization steps required by the detection of SURF features, the most computationally demanding step of this procedure can now be calculated with low cost by means of Algorithm 2. The corresponding *Kernel* is executed using $[512 \times 4]$ *Threads* utilizing once more all the GPU's resources. Note that the *Global* memory accesses are once more coalesced, while the employed *Texture* caching reduces the memory's latency in cases of *read* operations from 2D-neighboring regions (i.e. the Integral Image patches). We chose to store the $offsetX$ and $offsetY$ vectors in the *Constant* memory only once in the beginning of our application, allowing multiple *Threads* to access the same memory addresses simultaneously without reducing the performance. The output of this procedure is one vector (stored in the *Global* memory) containing the calculated gray-scale average values for each Sub-Block of every detected patch p_i: $avgG\_vector = \left[\hat{\boldsymbol{B}}_0, \hat{\boldsymbol{B}}_1, \hat{\boldsymbol{B}}_2, \hat{\boldsymbol{B}}_3\right]$, where $\hat{\boldsymbol{B}}_j = \left[\bar{B}_j^0, \bar{B}_j^1, ..., \bar{B}_j^i, ..., \bar{B}_j^{511}\right]$.

In order to numerically interpret the texture information, LoCATe applies a set of 5 different filtering masks on each detected patch. The obtained filter responses are then thresholded determining if the patch can be considered as member of a respective edge directivity class or if it is of non-texture. Thus, one individual *Thread* is utilized for every one of the 512 detected feature patches, undertaking the calculation of every mask's response. The *Kernel* produces 512 binary column vectors ${}^t\hat{\boldsymbol{v}}^i$ of size 6, the values of which (${}^tv_j^i$) declares the patch's membership (or not) to one of the five texture classes ($j \in [1, 5]$) or to the texture-less one ($j = 0$). For an instance of patch p_i, if the maximum mask response does not overcome a predefined threshold T_m, the value of ${}^tv_0^i$ becomes one, while the rest of vector ${}^t\hat{\boldsymbol{v}}^i$ is zeroed. Otherwise, the respective responses are normalized by the maximum and compared with the corresponding thresholds formulating the membership vector ${}^t\hat{\boldsymbol{v}}^i$. One may argue that in this step the calculation of each mask's response could be performed simultaneously using 5 different *Threads* per patch. In that case, even though

DOI: 10.1002/cpe

Algorithm 3 The Gray-scale Averaging *Kernel*.

Require: ${}^{c}\hat{\boldsymbol{v}}$: stored row-major array containing the ${}^{c}\hat{\boldsymbol{v}}^{i}$ vectors in its columns
Require: ${}^{b}\hat{\boldsymbol{v}}$: stored row-major array containing the ${}^{b}\hat{\boldsymbol{v}}^{i}$ vectors in its columns
Require: ${}^{t}\hat{\boldsymbol{v}}$: stored row-major array containing the ${}^{t}\hat{\boldsymbol{v}}^{i}$ vectors in its columns
Output: ${}^{d}\hat{\boldsymbol{v}}$: stored row-major array containing the ${}^{d}\hat{\boldsymbol{v}}^{i}$ vectors in its columns

1: **for all** $blockIDx.x/y, x \in [0,7] \cap \mathbb{N}, y \in [0,7] \cap \mathbb{N}$ **do in parallel**
2: **for all** $threadIDx.x/y, x \in [0,511] \cap \mathbb{N}, y \in [0,2] \cap \mathbb{N}$ **do in parallel**
3: **if** ${}^{t}\hat{\boldsymbol{v}}[blockIDx.x] == 0$ **then**
4: $value = {}^{c}\hat{\boldsymbol{v}}[512 * threadIdx.y + threadIDx.x] * {}^{t}\hat{\boldsymbol{v}}[512 * blockIDx.y + threadIDx.x]$
5: **else**
6: $value = {}^{b}\hat{\boldsymbol{v}}[512 * threadIdx.y + threadIDx.x] * {}^{c}\hat{\boldsymbol{v}}[512 * (blockIDx.x + 2) + threadIDx.x] * {}^{t}\hat{\boldsymbol{v}}[512 * blockIDx.y + threadIDx.x]$
7: **end if**
8: $j = blockIDx.y * 24 + blockIDx.x * 3 + threadIdx.y$
9: ${}^{d}\hat{\boldsymbol{v}}[512 * j + threadIDx.x] = value$
10: **end for**
11: **end for**

less steps are required to be executed by each *Thread*, there is no efficient way to identify the maximum response without violating the SIMT architecture. Additionally, the color and texture *Kernels* are asynchronous with each other and thus executed simultaneously on the *Device*. This essentially means that even if we assign more *Threads* to the *Kernel*, their execution will be serialized (on some extend) since the GPU will always be fully occupied. Finally, the vectors ${}^{t}\hat{\boldsymbol{v}}^{i}$ are stored with an array row-major format of:

$$ {}^{t}\hat{\boldsymbol{v}} = \left[\; {}^{t}\hat{\boldsymbol{v}}^{0} \;\middle|\; {}^{t}\hat{\boldsymbol{v}}^{1} \;\middle|\; \ldots \;\middle|\; {}^{t}\hat{\boldsymbol{v}}^{511} \; \right] \tag{13} $$

for the same reasons described in Section 5.2.1.

5.2.3. Description Histogram Formulation and Quantization: The final processing step of LoCATe includes the combination of vectors ${}^{c}\hat{\boldsymbol{v}}^{i}$, ${}^{b}\hat{\boldsymbol{v}}^{i}$ and ${}^{t}\hat{\boldsymbol{v}}^{i}$ from each feature patch p_i into a total description histogram ${}^{d}\hat{\boldsymbol{v}}^{i}$. Subsequently, this histogram is quantized forming a final description vector ${}^{L}\hat{\boldsymbol{v}}^{i}$.

Algorithm 3 describes the procedure for combining the histograms and creating one ${}^{d}\hat{\boldsymbol{v}}$ array. This array is stored in a row-major format and contains the ${}^{d}\hat{\boldsymbol{v}}^{i}$ column vectors with equivalent structure to the ones of eq. 12 and 13. The specification of this algorithm is that it calculates one LoCATe value simultaneously for every feature patch. The *Kernel* is executed by 8×8 *Thread Blocks* of 512×3 *Threads* each. The $threadIdx.x$ indexing dimension of every *Thread* determines the i-th column of each respective array ${}^{F}\hat{\boldsymbol{v}}^{i}$ ($F = \{c, b, t\}$), while the rest of the employed indexing terms ($threadIdx.y$, $blockIDx.x$ and $blockIDx.y$) are responsible for defining the corresponding row (${}^{F}\hat{\boldsymbol{v}}^{i}_{j}$). As defined by LoCATe, the first three values of each ${}^{c}\hat{\boldsymbol{v}}^{i}$ do not depend on the ${}^{b}\hat{\boldsymbol{v}}^{i}$ ones and thus, they are just copied to the final description histogram.

With the ${}^{d}\hat{\boldsymbol{v}}$ in hand, we choose to address the quantization procedure with another PPI engine. A total of $[8 \times 144]$ *Threads* are assigned for each one of the 512 ${}^{d}\hat{\boldsymbol{v}}_{i}$ histograms, determining the participation of every bin to one of quantization levels. The output of this procedure is the final ${}^{L}\hat{\boldsymbol{v}}$ array which contains the LoCATe descriptors for all the detected feature points. In order to be efficiently accessed by the *Host*, this array is transposed [49] (so as to contain one LoCATe descriptor in every row) and transferred to the CPU memory.

DOI: 10.1002/cpe

Table I. Details of the used datasets.

Dataset	Description	Camera position	Image size	Number of Images
Lip6 Indoor	Indoors Static	Frontal	240×192	388
Lip6 Outdoor	Outdoors Slightly dynamic	Frontal	240×192	531
Malaga 2009 Parking 6L	Outdoors Slightly dynamic	Frontal	1024×768	3474
City Centre	Outdoors, Urban Dynamic	Lateral	640×480	1237

Figure 6. An instance of our on-line vPR application running on Google's Project Tango development kit.

5.3. *DBoW2*

A loop closure detection system requires a constant image stream as an input. Each time a new frame is acquired, the aforementioned steps are performed on the *Device* meaning that the *Host* remains idle. As in [50, 51, 52], we adopt a pipeline architecture in order to utilize all of the device's available resources. Using two *Host* threads, during the GPU-based SURF features detection and LoCATe descriptors formulation from image i, we assign to the CPU all processing steps required to detect a possible loop closure event between the image $i-1$ and the database. Thus, the computational time required by the rest of the DBoW2 algorithm, liberated from the feature detection and description part, is overlaid by the parallel algorithm described in the previous subsections.

6. EXPERIMENTAL RESULTS

In this section, an experimental evaluation of our whole system is provided. The goal of our proposal is to provide a vPR place recognition system capable of running in real time even in the case of a low power mobile device. For this reason, we tested our algorithm measuring both its accuracy on detecting loop closure events and the achieved execution time. The obtained results justify our choice for adopting the local description of LoCATe for every one of the assessed datasets.

6.1. *Experimental Protocol*

Four different testing datasets were used in order to validate our algorithm, namely Lip6 Indoor [10], Lip6 Outdoor [10], Malaga 2009 Parking 6L [53] and City Centre [8]. Table I contains their brief description. One of the main restrictions that we considered during our choice of testing cases

DOI: 10.1002/cpe

Table II. Average execution time for each one of the tested datasets.

		Average Time (ms/frame)			
		Lip6 Indoor	Lip6 Outdoor	Malaga 2009 Parking 6L	City Centre
SURF Detection	Serial Version	148.4	147.2	461.1	316.7
	Parallel Version	4.4	4.3	14.5	10.1
LoCATe Description	Serial Version	41.1	41.7	41.9	41.2
	Parallel Version	4.7	4.7	4.6	4.7
DBoW2 (- BRIEF)	Serial Version	9.8	10.1	12.9	11.5
Whole algorithm	Serial Version	199.4	199.2	516.0	368.8
	Parallel Version	10.5	10.7	19.7	15.4

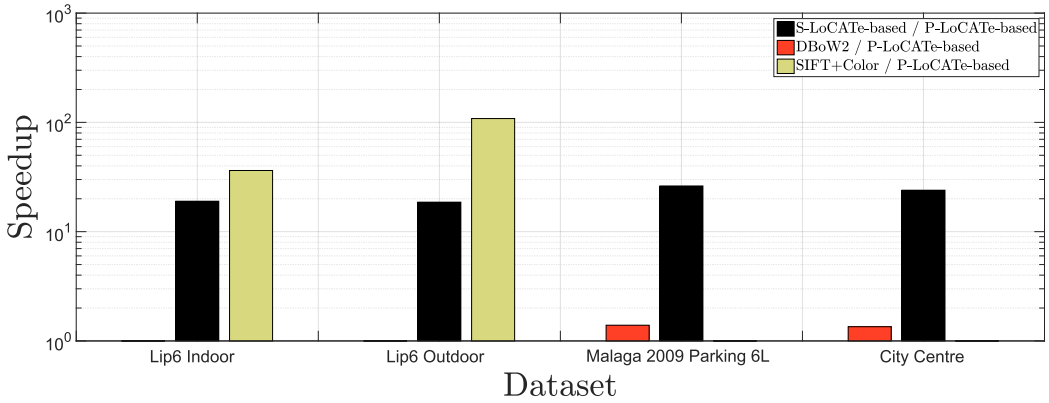


Figure 7. Achieved speedup of the proposed parallel pipeline over every assessed dataset.

was that each dataset should contain colored images and not gray-scale ones in order for LoCATe to be applicable.

Our algorithm was designed with the strict specification of being able to run on a mobile device in real-time (meaning that the output is calculated faster or in equal time with the frequency induced by a key-frame SLAM system [4, 5, 54]). Our timing experiments were conducted on the Tablet device provided by Google's Project Tango [33]. This device contains a GPU based on the NVIDIA's Kepler architecture, as well as an ARMv7 CPU. The onboard camera can be used to acquire the necessary input image stream and the detected loop closing pairs of frames can be displayed on the screen as the user is moving. Figure 6 contains an instant of our running application.

As a means of measuring the achieved vPR performance for the conducted experiments, the Precision-Recall curves were utilized. As a reminder, the Precision term is defined as the ratio between true-positive loop closure detection and the total number of detections returned by our system. Furthermore, Recall is defined as the ratio between true-positive detections and the total number of loop closure events that a testing dataset contains. An accurate vPR system should achieve as high Recall rates as possible for 100% Precision since the inclusion of a false-positive detection will most probably worsen the SLAM output.

Aiming to strengthen our proposal, we compare our approach with the ones presented in [12] and [10]. The first one corresponds to the DBoW2 algorithm (the one that we based our work on), while

DOI: 10.1002/cpe

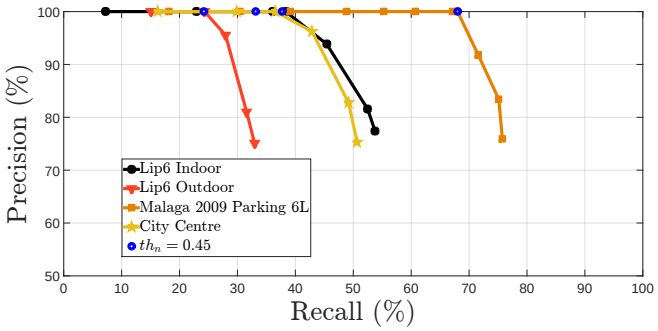


Figure 8. Precision-Recall curves achieved by our technique for every tested dataset.

the second one can be characterized as notably relevant to our technique since it also includes color information to the description. Note that the corresponding timing and accuracy results were pulled straightforwardly from the respective methods' papers.

6.2. Timing Results

In order to test the efficiency of our parallel algorithm, a CPU-only version of the proposed LoCATe-based vPR method was also implemented. This version serves as a reference point, determining the time that the approach would require if no GPGPU parallelization was applied. The average timing results for both parallel and serial versions are illustrated in Table II. Each one of the main processing steps was timed for every testing dataset. Note that the timings labeled as "DBoW2 (- BRIEF)" corresponds to the processing steps that DBoW2 requires excluding the BRIEF features detection and description. The used datasets contain frames with different resolutions. This characteristic only affects the computations corresponding to the SURF feature detection. The rest of the method's steps are performed only among the 512 most prominent SURF feature patches implying a relatively constant-time output. It should also be noticed that the timings corresponding to the CPU's calculations are not perceptible by the total execution time of the parallel version due to the employed pipelining. Figure 7 presents the overall speedup that we obtained through the algorithm's parallelization for each dataset. "*S-LoCATe-based / P-LoCATe-based*" corresponds to the speedup we obtained over the serialized version of the proposed algorithm, "*DBoW2 / P-LoCATe-based*" to the speedup over the original DBoW2 [12] method and "*SIFT+Color / P-LoCATe-based*" over the method presented in [10]. As it can be seen, our approach presents a stable improvement of the vPR efficiency for all tested image sizes. In addition, the reader should note that timing results regarding the DBoW2 method are only available for the Malaga 2009 Parking 6L and City Centre datasets. Correspondingly, [10] only offers timing results for the Lip6 Indoor and Lip6 Outdoor cases.

6.3. System's Accuracy

In order to measure the proposed system's accuracy via Precision-Recall metrics, a loop closure ground truth is required. Datasets Lip6 Indoor, Lip6 Outdoor and City Centre provide this information by indicating the image pairs that are considered to observe the same content. For the case of Malaga 2009 Parking 6L though, no such ground truth is provided. Thus, for this dataset we hand-picked all the loop closure events that could be visually identified by a human so as to distinguish the true-positive from the false-positive detections of our algorithm. Figure 8 presents the Precision-Recall curves for all the testing cases that we obtained by varying threshold th_n. Considering the general case, using a threshold value of $th_n = 0.45$ results to the highest Recall rates for 100% Precision in every assessed scenario. Yet, in a real application scenario, one can adjust threshold th_n to the most beneficial value.

As it was stated by [31], characterizing a frame as an aggregation of local feature descriptors produces more robust image matching results than a global image descriptor. In order to confirm

DOI: 10.1002/cpe

Table III. Comparative accuracy results between the tested vPR approaches.

	Recall rates (for 100% Precision accuracy)			
	Lip6 Indoor	Lip6 Outdoor	Malaga 2009 Parking 6L	City Centre
CEDD-based	24.48%	15.56%	43.71%	21.64%
LoCATe-based	37.92%	23.60%	68.02%	36.24%
Original DBoW2 [12]	N/A	N/A	74.75%	30.61%
Angeli et al. [10]	36.86%	23.59%	N/A	N/A

this argument in the field of vPR, we formulated another experiment testing the effect of a global descriptor to the loop closure detection task. For this experiment, instead of a LoCATe-based VWV, we computed one CEDD descriptor for each input image and tried to detect the loop closing pairs of frames using its 144bin description vector. Table III contains the Recall results, corresponding to 100% precision, for the LoCATe-based, CEDD-based, the original DBoW2 and the method proposed in [10] on every tested dataset. As it can be seen, the localized approach constantly outperforms the global descriptor and the technique of [10] on every available case. Regarding the original DBoW2 approach, which was based on the description of BRIEF, our algorithm produced more accurate results in the case of City Centre dataset, but it was not able to detect as many loop closure events in the Malaga 2009 Parking 6L one. Even though DBoW2 performed better in the last case, it also induces higher execution time (an average of 21.6ms on an Intel Core i7 @ 2.67GHz machine [12]), making our method a better choice when the operational frequency is crucial. Finally, Fig. 9 visualizes the loop closing camera pairs detected by the proposed LoCATe-based vPR system, projected over the robot's executed trajectories[§].

7. CONCLUSION AND FUTURE WORK

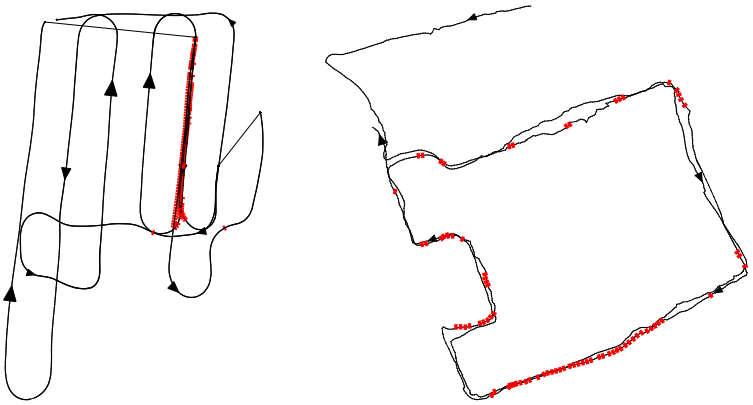
In this work a novel parallel vPR algorithm was proposed, using GPGPU computing, capable of running on the Google's project Tango device in real-time. Instead of relying on a global descriptor, a bottom-up approach was adopted based on the description of local features. The most computationally demanding steps for detecting and describing the feature points were assigned to the GPU, while the CPU undertook all the calculations required for matching between the images. Using a pipeline scheme, the two available processing units can work in parallel, making sure that all the device's resources are utilized. Finally, our choice of combining SURF and LoCATe grants a highly efficient parallel implementation, since those methods contain some common computational steps.

A loop closure detection engine can be used in order to assist the SLAM output accuracy. Therefore, the authors' future plans involve the development of an onboard monocular SLAM system, based on the description of LoCATe, fully compatible with the presented vPR technique. Such a system is expected to induce more computationally demanding modules, and thus further investigation is required in order to define the most appropriate components that the GPU should undertake.

ACKNOWLEDGEMENT

[§]Note that the datasets Lip6 Indoor and Lip6 Outdoor do not provide any odometry information and thus they are excluded from Fig. 9.

DOI: 10.1002/cpe



(a) Malaga 2009 Parking 6L

(b) City Centre

Figure 9. Detected loop closures. The respective camera poses are marked with red.

Special thanks to Nektarios Anagnostopoulos, one of the co-developers of LIRE [55] library, for kindly providing us with the reference LoCATe feature detection code.

REFERENCES

1. Folkesson J, Christensen H. Graphical SLAM-a self-correcting map. *In Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, 2004; 383–390.
2. Thrun S, Montemerlo M. The graph SLAM algorithm with applications to large-scale mapping of urban structures. *The Int. J. of Robotics Research* 2006; **25**(5-6):403–429.
3. Grisetti G, Kümmerle R, Stachniss C, Burgard W. A tutorial on graph-based SLAM. *Intelligent Transportation Systems Magazine* 2010; **2**(4):31–43.
4. Strasdat H, Montiel J, Davison AJ. Scale drift-aware large scale monocular SLAM. *In Proc. Robotics: Science and Systems*, vol. 2, 2010; 5.
5. Mei C, Sibley G, Cummins M, Newman PM, Reid ID. A constant-time efficient stereo SLAM system. *In Proc. British Machine Vision Conf.*, 2009; 1–11.
6. Williams B, Cummins M, Neira J, Newman P, Reid I, Tardós J. A comparison of loop closing techniques in monocular SLAM. *Robotics and Autonomous Systems* 2009; **57**(12):1188–1197.
7. Nister D, Stewenius H. Scalable recognition with a vocabulary tree. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, 2006; 2161–2168.
8. Cummins M, Newman P. Fab-map: Probabilistic localization and mapping in the space of appearance. *The Int. J. of Robotics Research* 2008; **27**(6):647–665.
9. Cummins M, Newman P. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The Int J. of Robotics Research* 2011; **30**(9):1100–1123.
10. Angeli A, Filliat D, Doncieux S, Meyer JA. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics* 2008; **24**(5):1027–1037.
11. Schindler G, Brown M, Szeliski R. City-scale location recognition. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2007; 1–7.
12. Gálvez-López D, Tardós JD. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics* 2012; **28**(5):1188–1197.
13. Gálvez-López D, Tardós JD. DBoW2: Enhanced hierarchical bag-of-word library for C++ 2012. URL `http://doriangalvez.com/software`.
14. Mur-Artal R, Tardós JD. Fast relocalisation and loop closing in keyframe-based slam. *In Proc. IEEE Int. Conf. on Robotics and Automation*, 2014; 846–853.
15. Collier J, Se S, Kotamraju V. Multi-sensor appearance-based place recognition. *In Proc. IEEE Int. Conf. on Computer and Robot Vision*, 2013; 128–135.
16. Kim DH, Kim JH. Visual loop-closure detection method using average feature descriptors. *Springer Robot Intelligence Technology and Applications 2*. Springer, 2014; 113–118.
17. Lowe DG. Distinctive image features from scale-invariant keypoints. *Int. J. of Computer Vision* 2004; **60**(2):91–110.
18. Bay H, Tuytelaars T, Van Gool L. SURF: Speeded Up Robust Features. *In Proc. European Conf. on Computer Vision*. Springer, 2006; 404–417.
19. Rublee E, Rabaud V, Konolige K, Bradski G. ORB: an efficient alternative to SIFT or SURF. *In Proc. IEEE Int. Conf. on Computer Vision*, 2011; 2564–2571.
20. Alahi A, Ortiz R, Vandergheynst P. Freak: Fast retina keypoint. *In Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2012; 510–517.
21. Churchill W, Newman P. Continually improving large scale long term visual navigation of a vehicle in dynamic urban environments. *In Proc. IEEE Int. Conf. on Intelligent Transportation Systems*, 2012; 1371–1376.

DOI: 10.1002/cpe

22. Sünderhauf N, Neubert P, Protzel P. Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons. *In Proc. IEEE Int. Conf. on Robotics and Automation, Workshop on Long-Term Autonomy,*, 2013.
23. McManus C, Upcroft B, Newman P. Learning place-dependant features for long-term vision-based localisation. *Autonomous Robots* 2015; **39**(3):363–387.
24. Arroyo R, Alcantarilla PF, Bergasa LM, Romera E. Towards life-long visual localization using an efficient matching of binary sequences from images. *In Proc. IEEE Int. Conf. on Robotics and Automation*, 2015; 6328–6335.
25. Yang X, Cheng KTT. Local difference binary for ultrafast and distinctive feature description. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2014; **36**(1):188–194.
26. Sunderhauf N, Shirazi S, Jacobson A, Dayoub F, Pepperell E, Upcroft B, Milford M. Place recognition with ConvNet landmarks: Viewpoint-robust, condition-robust, training-free. *In Proc. Robotics: Science and Systems*, MIT Press: Rome, Italy, 2015.
27. Sünderhauf N, Shirazi S, Dayoub F, Upcroft B, Milford M. On the performance of ConvNet features for place recognition. *In Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2015; 4297–4304.
28. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *In Proc. Advances in neural information processing systems*, 2012; 1097–1105.
29. Arroyo R, Alcantarilla PF, Bergasa LM, Romera E. Fusion and binarization of CNN features for robust topological localization across seasons. *In Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2016.
30. Chatzichristofis SA, Boutalis YS. CEDD: Color and Edge Directivity Descriptor: A compact descriptor for image indexing and retrieval. *Int. Conf. on Computer Vision Systems* 2008; :312–322.
31. Iakovidou C, Anagnostopoulos N, Kapoutsis AC, Boutalis Y, Chatzichristofis SA. Searching images with MPEG-7 (& MPEG-7-like) powered localized descriptors: the SIMPLE answer to effective content based image retrieval. *In Proc. IEEE Int. Workshop on Content-Based Multimedia Indexing*, 2014; 1–6.
32. Iakovidou C, Bampis L, Chatzichristofis SA, Boutalis YS, Amanatiadis A. Color and Edge Directivity Descriptor on GPGPU. *In Proc. IEEE Int. Conf. on Parallel, Distributed and Network-Based Processing*, 2015; 301–308.
33. Google project tango. URL https://www.google.com/atap/projecttango/#project.
34. Chatzichristofis S, Zagoris K, Boutalis Y, Papamarkos N. Accurate image retrieval based on compact composite descriptors and relevance feedback information. *Int. J. of Pattern Recognition and Artificial Intelligence* 2010; **24**(2):207–244.
35. Manjunath B, Ohm J, Vasudevan V, Yamada A. Color and texture descriptors. *IEEE Transactions on circuits and systems for video technology* 2001; **11**(6):703–715.
36. Calonder M, Lepetit V, Strecha C, Fua P. BRIEF: Binary Robust Independent Elementary Features. *In Proc. European Conf. on Computer Vision*, 2010; 778–792.
37. RAWSEEDS. Robotics Advancement through Web-publishing of Sensorial and Elaborated Extensive Data Sets (Project FP6-IST-045144) 2007-2009. URL http://www.rawseeds.org/rs/datasets.
38. Arthur D, Vassilvitskii S. k-means++: The advantages of careful seeding. *In Proc. Symp. on Discrete algorithms*, 2007; 1027–1035.
39. Sivic J, Zisserman A. Video google: A text retrieval approach to object matching in videos. *In Proc. IEEE Int. Conf. on Computer Vision*, 2003; 1470–1477.
40. Bampis L, Amanatiadis A, Gasteratos A. Encoding the description of image sequences: A two-layered pipeline for loop closure detection. *In Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2016; 4530–4536.
41. CUDA by NVIDIA. URL http://www.nvidia.com/object/cuda_home_new.html.
42. Hwu WM, Rodrigues C, Ryoo S, Stratton J. Compute unified device architecture application suitability. *Computing in Science & Engineering* 2009; **11**(3):16–26.
43. NVIDIA Corporation. Cuda online programming guide. 2013. URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/.
44. Terriberry TB, French LM, Helmsen J. GPU accelerating speeded-up robust features. *In Proc. IEEE Int. Symp. on 3D Data Processing, Visualization and Transmission*, 2008; 355–362.
45. Blelloch GE. Prefix sums and their applications. *Carnegie Mellon University School of Computer Science, Tech. Rep. CMU-CS-90-190* 1990; .
46. Cornwall J, Kelly P. Efficient multiple pass, multiple output algorithms on the gpu. *In Proc. European Conf. on Visual Media Production*, 2005; 253–262.
47. Zhu F, Chen P, Yang D, Zhang W, Chen H, Zang B. A GPU-based high-throughput image retrieval algorithm. *In Proc. ACM General Purpose Processing with Graphics Processing Units*, 2012; 30–37.
48. Fang Z, Yang D, Zhang W, Chen H, Zang B. A comprehensive analysis and parallelization of an image retrieval algorithm. *In Proc. IEEE Int. Symp. on Performance Analysis of Systems and Software*, 2011; 154–164.
49. Ruetsch G, Micikevicius P. Optimizing matrix transpose in CUDA. *Nvidia CUDA SDK Application Note* 2009; **28**.
50. Bampis L, Iakovidou C, Chatzichristofis SA, Boutalis YS, Amanatiadis A. Real-time indexing for large image databases: color and edge directivity descriptor on GPU. *Springer The J. of Supercomputing* 2015; **71**(3):909–937.
51. Amanatiadis A, Bampis L, Gasteratos A. Accelerating single-image super-resolution polynomial regression in mobile devices. *IEEE Transactions on Consumer Electronics* 2015; **61**(1):63–71.
52. Amanatiadis A, Bampis L, Gasteratos A. Accelerating image super-resolution regression by a hybrid implementation in mobile devices. *In Proc. IEEE Int. Conf. on Consumer Electronics*, 2014; 335–336.
53. Blanco JL, Moreno FA, Gonzalez J. A collection of outdoor robotic datasets with centimeter-accuracy ground truth. *Autonomous Robots* 2009; **27**(4):327–351.
54. Davison AJ, Reid ID, Molton ND, Stasse O. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2007; **29**(6):1052–1067.
55. Lux M, Chatzichristofis SA. Lire: lucene image retrieval: an extensible java cbir library. *In Proc. Int. Conf. on Multimedia*, 2008; 1085–1088.

DOI: 10.1002/cpe