Noname manuscript No.
(will be inserted by the editor)

GPU Accelerated Registration of Hyperspectral Images Using KAZE Features

Álvaro Ordóñez · Francisco Argüello · Dora B. Heras · Begüm Demir

Received: date / Accepted: date

Abstract Image registration is a common task in remote sensing, consisting in aligning different images of the same scene. It is a computationally expensive process, especially if high precision is required, the resolution is high, or consist of a large number of bands, as is the case of the hyperspectral images. HSI–KAZE is a registration method specially adapted for hyperspectral images that is based on feature detection and takes profit of the spatial and the spectral information available in those images. In this paper, an implementation of the HSI–KAZE registration algorithm on GPUs using CUDA is proposed. It detects keypoints based on non–linear diffusion filtering and is suitable for on–board processing of high resolution hyperspectral images. The algorithm includes a band selection method based on the entropy, construction of a scale-space through of non-linear filtering, keypoint detection with position refinement, and keypoint descriptors with spatial and spectral parts. Several techniques have been applied to obtain optimum performance on the GPU.

Keywords image registration · hyperspectral data · KAZE features · remote sensing · CUDA · GPU

Álvaro Ordóñez
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain
E-mail: alvaro.ordonez@usc.es

Francisco Argüello
Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, Spain
E-mail: francisco.arguello@usc.es

Dora B. Heras
Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain
E-mail: dora.blanco@usc.es

Begüm Demir
Faculty of Electrical Engineering and Computer Science, TU Berlin, Berlin, Germany
E-mail: demir@tu-berlin.de

1 Introduction

Nowadays, the capture of hyperspectral images is easier thanks to the new advances in image sensor technologies. Hyperspectral images consist of hundreds of continuous spectral bands. This large amount of information had extended the use of these images to a multitude of applications such as land use classification [9], quality control [18], agriculture [14], and medical science [31] among others.

Image registration is a previous fundamental task in many of these applications. It consists in estimating a geometrical transformation that maps one image to another. Image registration methods can be classified according to their nature in two categories [34]: area–based methods and feature–based methods. The first group, area–based methods, work directly with image intensity values while the second group, feature–based methods, seek to detect relevant features such as regions, lines or points. This representation at a higher level makes the feature–based methods more suitable for multisensor registration or illumination changes. Two stages may be identified in a feature–based method: feature detection and feature description. In the first stage, the features are usually extracted by building a pyramidal scale–space to achieve scale invariance. In the second stage, a description is computed for each detected feature. This description encodes the feature information into a sequence of numbers in order to have an unique identification per feature.

The scale-invariant feature transform (SIFT) [15] is the most popular feature–based method. It tries to detect points with distinctive features called keypoints. Keypoints are detected by building a Gaussian scale–space following a pyramidal model. This scale–space is built by smoothing and downsampling the original images in order to achieve scale invariance. Different orientations are also computed and assigned to each keypoint to achieve the rotation invariance. Finally, the detected keypoints are described using a 128 value descriptor computed from different regions of the pixel neighbourhood.

Since the publication of SIFT, different features–based methods have been published following the same scheme. For example, the Speeded Up Robust Features (SURF) [6] was proposed to be computationally more efficient than SIFT replacing the pyramid of images of SIFT with a pyramid of filters, and proposing a new descriptor with only 64 values. A more recent method, KAZE [4], proposes a selective blur in the construction of the scale–space using a nonlinear diffusion filter instead of Gaussian filters. This selective blur maintains the edges of the image. Moreover, it uses the Modified SURF (M–SURF) descriptor [2], a modification of the original SURF descriptor to be faster and handle the boundaries better. The same author of KAZE proposes a version of the method called A–KAZE [5]. Unlike KAZE, A–KAZE uses a Fast Explicit Diffusion (FED) scheme to build the scale–space and a binary descriptor, both changes contribute to reduce the computation times.
The described methods were designed to work with grayscale images and not with multi or hyperspectral images. Different authors proposed adaptations of the methods in order to be specially efficient for these images. Yi et al. [32]

propose a scale restriction criteria for keypoint matching using SIFT. Li et al. [12] also use a scale restriction but adding an orientation restriction. Vural et al. [16] use both restrictions and improve the feature matching stage with a shorter descriptor decreasing the CPU computation time. Teke et al. [30] introduce the scale restriction of SIFT applied to SURF. In [11] the matching stage is improved taking advantage of neighborhood information of the SIFT keypoints. However, all these methods only use one band of the spectral images. Mukherjee et al. [17], nevertheless, propose the use of a principal component analysis (PCA) to combine information across spectral bands. They build a scale–space and perform an individual keypoint detection for each selected principal component. In Dorado-Muñoz et al. [7] the scale–space is generated taking into account all the spectral bands of the image and uses a nonlinear diffusion in order to preserve the edges in the image. Al-khafaji et al. [3] present a 3D spectral-spatial SIFT for hyperspectral images. The method exploits both, spatial and spectral, dimensions simultaneously detecting the keypoints in a 3D data cube. Moreover, they propose a new descriptor based on the distribution of the spectral-spatial gradient magnitude in the 3D neighborhood of the keypoint. However, the method was not tested with remote sensing images. Li et al. [13] introduce a spatial-spectral SIFT for hyperspectral classification based on an unified model of spectral value and gradient change. Finally, in [24], the authors propose HSI–KAZE, a method to register hyperspectral remote sensing images based on KAZE but considering the spectral information. The algorithm is oriented towards extreme situations in which the images are very different in terms of scale, rotation, and other variations.

All the processes involving the usual tasks performed over hyperspectral images as, for example, registration, classification, object detection or change detection, are computationally demanding. The main reason is the large amount of spectral information available in each image comprising hundreds of spectral bands in many cases. The number of images available in space agencies databases continuously increases. At the same time, new applications of hyperespectral images to solve new problems are also emerging due to the affordable price of the sensors that are mounted over a variety of platforms, for example UAVs. This situation makes necessary to produce algorithms and, in particular registration algorithms, which are efficient from the computational point of view even when real-time computation is not always required for the registration process.

Most of the registration algorithms mentioned in this section were designed ignoring the computational performance. In the literature, some registration algorithms for two-dimensional images in GPU have been proposed [8, 19, 26, 27, 29, 33]. In particular, [26] focuses on the KAZE algorithm. But none of these algorithms is adapted to hyperspectral images, which contain more than a hundred times the data of a typical image. The only reference of registration algorithm on GPU specially adapted for hyperspectral images is the HYperspectral Fourier–Mellin algorithm (HYFM) [25]. The algorithm exploits the information contained in the different bands of the images and it is based

on principal component analysis, the Fast Fourier Transform (FFT), and the combination of log-polar maps.

In this paper, the first GPU algorithm for feature-based registration adapted for hyperspectral images based on HSI–KAZE [24] is presented. The resulting algorithm combines the information of a set of preselected bands, and adapts the keypoint description and the matching stages to take into account the spectral information. The whole algorithm has been developed in CUDA to obtain an effective exploitation of the GPU architecture. Execution times in GPU are also compared to an OpenMP CPU implementation of the algorithm.

2 HSI–KAZE algorithm

In this section, we summarize the algorithm for registering hyperspectral images based on KAZE [4] and A–KAZE [5] features called HSI–KAZE [24]. HSI–KAZE is a feature–based method and as such it searches for distinctive features, in this case, distinctive pixels called keypoints. The main idea is to detect keypoints present in the two images that we want to align. Then, knowing the correspondence between a number of keypoints in both images, the transformation to align one in relation to the other can be calculated.

The method consists of seven stages: band selection, keypoint detection, keypoint description, keypoint matching, band combination and registration. Figure 1 shows the outline of the algorithm. The different stages are described in more detail in the pseudocode of Algorithm 1.

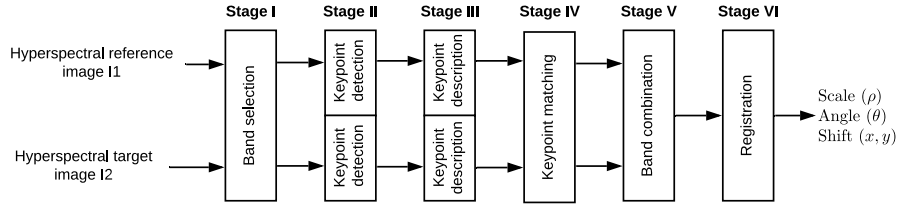

Fig. 1: Scheme of HSI–KAZE algorithm.

The algorithm has two hyperspectral images as input data. One of the images is referred to the reference image and the other to the target image, the image that we want to register with respect the reference image. In the first stage, band selection, a set of most representative bands are selected according their entropy and inter–band distance. Next, in the second stage, keypoint detection, each selected band is smoothed and downsampled in order to detect keypoints with scale invariance (scale–space). The keypoints are detected in these scale–spaces. Then, in the third stage, a descriptor made up of two parts (a spatial and a spectral one) is computed for each detected keypoint in the previous stage. In the fourth stage, the keypoint matching process is carried out based on the computation of the Euclidean distance for the spatial part

of the descriptor and the cosine similarity for the spectral part. One keypoint of the target image is assigned to each keypoint of the reference image if some distance conditions are fulfilled. In the fifth stage, all matched keypoints from the different bands are joined in order to exploit the spectral information. Finally, in the last stage, registration, knowing the correspondences between a number of keypoints of both images, the scale factor ρ, rotation angle θ, and translation (x, y) are computed.

Algorithm 1 HSI–KAZE pseudocode.

Input: Hyperspectral reference image $\mathbf{I}_1$ and target image $\mathbf{I}_2$.
Output: Scale factor ρ, rotation angle θ, and translation (x, y).

Stage I. Band selection.
Feature selection over both images $\rightarrow \mathbf{B}_1$ and $\mathbf{B}_2$

for each band b **in** images $\mathbf{B}_1$ and $\mathbf{B}_2$ **do**
Stage II. Keypoint detection.
    Extract keypoints of $\mathbf{B}_1^b \rightarrow \mathbf{P}_1^b$
    Extract keypoints of $\mathbf{B}_2^b \rightarrow \mathbf{P}_2^b$

Stage III. Keypoint description.
    Calculate the M–SURF descriptor of each keypoint in $\mathbf{P}_1^b$ and append the spectral signature $\rightarrow \mathbf{K}_1^b$
    Calculate the M–SURF descriptor of each keypoint in $\mathbf{P}_2^b$ and append the spectral signature $\rightarrow \mathbf{K}_2^b$

Stage IV. Keypoint matching.
    Match keypoints in $\mathbf{K}_1^b$ and $\mathbf{K}_2^b \rightarrow \mathbf{M}_b$
end for

Stage V. Band combination.
Combine all the matched keypoints $\mathbf{M}_b \rightarrow \mathbf{M}$

Stage VI. Registration.
Exhaustive search of registration parameters $\rightarrow \rho, \theta, (x, y)$

3 HSI–KAZE algorithm on GPU

In this section, we introduce the main concepts behind the CUDA programming model as well as the CUDA implementation of the HSI–KAZE algorithm on GPU enhancing the relevant features that makes the algorithm in CUDA efficient.

3.1 CUDA GPU Programming Fundamentals

CUDA is a parallel computing platform and programming model that makes using a GPU for general purpose computing. It allows running programs using parallel functions called kernels [28]. Each kernel executes a set of parallel threads. Thus, each thread runs an instance of the kernel following a single instruction multiple thread (SIMT) programming model. The programmer have to organize these threads into a grid of blocks. A block is a set of threads that work together. Threads have access to different memory spaces. All threads have access to the global memory. It is the largest memory space but it has the lowest throughput. On the other hand, the shared memory is only visible by the threads of the block and provides faster access to data than global memory. The Pascal GP102 architecture provides 96 KB/SM of dedicated shared memory and 48 KB/SM L1/texture cache. Moreover, it features a 3072 KB L2 cache.

Different optimization strategies have been applied in order to obtain the best performance from the NVIDIA GPU:

1. Search for the best kernel configurations.
2. Reduce the data transfers between the host and device memories.
3. Minimization of memory usage by performing some computations in place.
4. Use of shared memory vs global memory.
5. Use of vectorial instructions to maximize instruction parallelism and optimize memory accesses.
6. Reduction of the number of operations using warp-level primitives to avoid shared memory latency.
7. The use of atomic operations prevents the race conditions among threads.
8. Change the order of operations to efficiently exploit the SIMT programming model.
9. Unroll loops to eliminate loop control instructions.
10. Efficient computation using optimized CUDA libraries.

3.2 CUDA Implementation

In this section, we describe the GPU implementation of the HSI–KAZE algorithm. In Figs. 2, 4, 5 and 6 the pseudocode of HSI–KAZE on GPU is presented. Each process executed in GPU is placed between <> symbols and involves more than one kernel. The GM acronym indicates that this process is executed in global memory and SM in the shared memory.

The optimization strategies applied globally are the following. A thread block size of 16×16 (256 threads) was configured in the sixty per cent of kernels. In the remaining kernels, a thread block size of 8×8 (64 threads) or 32×16 (512 threads) was chosen to adequate the execution to the size of data (CUDA optimization strategy 1 explained in Section 3.1). Most of the functions run on GPU to reduce the data transfers between the host and device memories (strategy 2). When it is possible, the computations are performed in place

to minimize the memory usage (strategy 3). Moreover, optimization strategy 1 has been applied using NVIDIA Visual Profiler tool to analyze each kernel and identify potential performance bottlenecks. The best kernel configurations have been chosen using this tool. Loops were unrolled in kernels where this compiler optimization allows us to eliminate loop control instructions and reduce the computation time (strategy 9). The highly optimized libraries CUB [20], NVIDIA Performance Primitives (NPP) [23] and CUBLAS are used to achieve a high-performance computation (strategy 10).

3.2.1 Band selection stage

Algorithm 2 Entropy–based Band Selection (EBS) on GPU.

Input: Hyperspectral reference image $\mathbf{I}_1$ and hyperspectral target image $\mathbf{I}_2$.
Output: Set of selected bands $\mathbf{B}_1$ and $\mathbf{B}_2$
Parameters: Number of selected bands N_B, minimum inter–band distance D_B.

1: **for each** band b **in** images $\mathbf{I}_1$ and $\mathbf{I}_2$ **do**
2: $< \text{Entropy of band } b \text{ of } \mathbf{I}_1 > \rightarrow e_1$ ▷ GM+SM
3: $< \text{Entropy of band } b \text{ of } \mathbf{I}_2 > \rightarrow e_2$ ▷ GM+SM
4: $\min(e_1, e_2) \rightarrow E[b]$
5: **end for**

6: Sort the elements of $\mathbf{E}$ in descending order.
7: Select the N_B bands of highest entropy with an inter–band distance greater or equal than a minimum inter–band distance D_B between consecutive pairs. $\rightarrow \mathbf{B_1}$ and $\mathbf{B_2}$

With the objective of reducing the number of bands of the hyperspectral images a band selection method is applied. The objective is selecting bands as different as possible in order to reduce the amount of redundant information and, as a consequence, reduce the computational cost of the algorithm. This process is carried out by an entropy-based method called Entropy–based Band Selection (EBS) method [24]. It maximizes the discriminant information provided by the selected bands. Histograms are required by the entropy calculation in order to calculate the frequency of the pixel values.

Firstly, both hyperspectral images are copied to global memory. In Algorithm 2, the pseudocode of EBS on GPU is presented. First, the entropy of each band of each hyperspectral image is computed on GPU (see Algorithm 2, lines 2-3). The calculation of the entropy for each band is detailed in Algorithm 3. First, the maximum and minimum values per band are computed using the

functions *cub::DeviceReduce::Min* and *cub::DeviceReduce::Max* of the CUB library [20] (strategies 4, 5, 6 and 10) (see Algorithm 3, lines 1-2). Then, the histogram to compute the entropy is calculated using *cub::DeviceHistogram::HistogramEven* (strategies 4, 5, 7 and 10) (see Algorithm 3, line 3). Next, the entropy value is computed performing a reduction approach (strategies 6 and 7) (see Algorithm 3, line 4).

The remaining features of EBS are calculated in the CPU because of its simplicity. Finally, the entropies of both images are copied to host memory and the minimum entropy of each band between the two images is selected (see Algorithm 2, line 4). The N_b bands of highest entropy with an inter–band distance greater or equal than D_B between consecutive pairs are selected (see Algorithm 2, lines 6–7). The selected bands are copied to a new array in global memory while the the hyperspectral data structures are deallocated from memory.

Algorithm 3 Entropy of a band on GPU

Input: Band $\mathbf{d_B}$.
Output: Entropy of band $\mathbf{d_B}$.

1: $<$ cub::DeviceReduce::Min $>$ $(\mathbf{d_B}) \rightarrow min$ ▷ GM+SM
2: $<$ cub::DeviceReduce::Max $>$ $(\mathbf{d_B}) \rightarrow max$ ▷ GM+SM
3: $<$ cub::DeviceHistogram::HistogramEven $>$ $(\mathbf{d_B}, min, max) \rightarrow histogram$ ▷ GM+SM
4: $<$ reduce_entropy $>$ $(histogram) \rightarrow entropy$ ▷ GM

3.2.2 Keypoint detection stage

As it was explained in Section 2, after the band selection, a loop over the selected bands performs two stages, keypoint detection and keypoint description. In this section the keypoint detection stage in GPU is explained. The objective is detecting points with distinctive features called keypoints that are scale invariant. A point is considered a keypoint if it is the local minimum or maximum compared to its neighbours. With the objective of detecting these extreme values the original images are smoothed at different levels using a non linear diffusion filter to build a scale–space.

A scale–space consists of a pyramid of images and is obtained by iteratively applying two operations, downsampling and smoothing. Subsamppling produces different N_{oct} octaves while smoothing produces N_{sub} sublevels. An additional initial upsample is applied in order to increase the resolution of the original image.

Once the scale–space has been built, it is necessary to detect the keypoints. A pixel will be considered as keypoint if it is the maximum of its neighbourhood when the Hessian matrix is calculated. The determinant of the Hessian matrix at the different scale levels needs to be calculated. The required derivatives to

compute the Hessian matrix are approximated by Scharr filters which provide better rotation invariance than other popular filters.

Algorithm 4 presents the pseudocode of the keypoint detection stage that calculates the keypoints for one band. The keypoint detection begins calculating the optimal number of octaves N_{oct} according to the spatial size of the band [24] (see Algorithm 4, line 1). Then, the band is upsampled using cubic interpolation to obtain a image whose size is divisible by the number of octaves N_{oct} with the help of the *nppiResizeSqrPixel_64f_C1R* function of the NVIDIA Performance Primitives (NPP) library [23] (strategies 4 and 10) (see Algorithm 4, line 2).

Algorithm 4 Keypoint detection stage.

Input: Band B.
Output: A set of keypoints $\mathbf{K}$.
Parameters: Number of sublevels N_{sub}.

```
1:  Calculate the optimal number of octaves → N_oct
2:  < Upsample the band to obtain images whose size is divisible by N_oct >        ▷ GM
3:  stage Build the pyramidal scale space
4:      < Upsample the band by a factor of 2 using cubic interpolation >           ▷ GM
5:      < Compute the contrast factor k >                                          ▷ GM+SM
6:      < Smooth the upsampled band using a Gaussian filter >                      ▷ GM+SM
7:      for o ← 1, N_oct do
8:          for s ← 1, N_sub do
9:              if o == 1 and s == 1 then continue
10:             < Smooth using a Gaussian filter >                                 ▷ GM+SM
11:             < Compute the Scharr derivatives >                                 ▷ GM+SM
12:             < Compute the conductivity g >                                     ▷ GM
13:             < Discretized the nonlinear diffusion equation >                   ▷ GM+SM
14:         end for
15:         < Subsample the last sublevel image by a factor of 2 >                 ▷ GM+SM
16:     end for
17: end stage

18: stage Locate the keypoints in the scale space
19:     < Compute the determinant of the Hessian matrix >                          ▷ GM+SM
20:     < Detect keypoints by searching in their neighbourhood > → K_1^b, K_2^b    ▷ GM
21:     < Refine the position and the scale of each keypoint >                     ▷ GM+SM
22: end stage
```

A different scale–space is built for each image band following a pyramidal scheme (see Algorithm 4, lines 3-17). First, the band is upsampled using the same NPP function but this time by a factor of 2 in order to minimize aliasing artifacts and extract a higher number of keypoints (GPU optimization strategies 4 and 10 listed in Section 3) (see Algorithm 4, line 4).

Next, the contrast factor k that controls the level of diffusion of each pyramid level is computed from a histogram with the distances between the vertical and horizontal derivatives of a smoothed version of the band [4] (see Algorithm 4, line 5). Firstly, the band is smoothed using a Gaussian filter that is com-

puted using two convolutions with a Gaussian kernel (strategies 4 and 6). Each convolution is performed by a different kernel in order to take advantage of the memory locality depending on the direction of the operation. Secondly, the Scharr derivatives are computed using shared memory for the different sublevels [1] (strategies 4 and 6). Finally, a histogram with the distances between these vertical and horizontal derivatives is computed to calculate the contrast factor k. In this calculation, the strategies 5 and 6 are used to calculate the maximum and minimum values required to compute the histogram, and the strategies 4 and 7 to store atomically the histogram in global and shared memory. Third, once the contrast factor k was calculated, the first sublevel of the first octave is computed performing a Gaussian blur over the upsampled band (see Algorithm 4, line 6) as we explained before.

Then, the next sublevels are computed following these steps (see Algorithm 4, lines 7-16):

1. Copy the previous sublevel as temporal new sublevel.
2. Smooth it using the Gaussian filter (strategies 4 and 6) (see Algorithm 4, line 10).
3. Compute the Scharr derivatives of this smoothed image (strategies 4 and 6) (see Algorithm 4, line 11).
4. Calculate the conductivity function $g = \frac{1}{1+\frac{|\nabla X_\sigma|^2}{k^2}}$ [24] using atomic operations (strategy 7) (see Algorithm 4, line 12).
5. Discretize the nonlinear diffusion equation using the FED scheme [4, **?**] (strategy 4) (see Algorithm 4, line 13).

The computation of the next octave begins subsampling the last sublevel image by a factor of 2 using *nppiResizeSqrPixel_64f_C1R* (strategies 4 and 10) (see Algorithm 4, line 15). Finally, the rest of sublevels are obtained repeating these steps.

Once the scale–space is constructed, the stage of locating the keypoints can begin. First, the determinant of the Hessian matrix of each sublevel is computed (see Algorithm 4, line 19). The second order horizontal and vertical derivatives and the second order cross derivative necessaries to compute the determinant are calculated using three kernels in order to take advantage of the memory locality as well as shared memory (strategy 4). The set of first and second order derivatives are approximated by a Scharr filter reusing the same kernels in the scale–space construction (strategies 4 and 6). Second, the keypoints are located by searching for pixels that are the maxima of their neighbourhood in the Hessian matrix (see Algorithm 4, line 20).

An important optimization is performed here with respect to the CPU version (strategy 8). In the CPU version, when a possible keypoint is detected, it is immediately compared with all previous keypoints in order to ensure that it was not previously detected in the same and in the previous sublevels. In the proposed GPU implementation, first, we locate all possible keypoints in the sublevel, and, then, we compare these keypoints with only the detected keypoints in the same and the previous sublevels. Both steps are implemented in

different GPU kernels and both use atomic operations to prevent the race conditions among threads (strategy 7). Finally, when the detection is completed along the entire scale–space, the detected keypoints are refined in position and scale (see Algorithm 4, line 21). Keypoints could be removed in this stage due to their instability. An optimized filtering approach with atomic operations and computed in shared memory is performed to remove them (strategies 4 and 7).

3.2.3 Keypoint description stage

The next stage of the HSI–KAZE algorithm is the keypoint description. A descriptor is a vector computed for each keypoint that is distinctive and invariant to variations such as rotation, scale, illumination, etc. It is used in the matching stage to try to find the same keypoint in the other image.

In this stage, the spatial and spectral keypoint descriptor that we call HSI–SURF descriptor is computed for each detected keypoint in the previous stage. The process followed is detailed in Algorithm 5.

Firstly, the spatial part of the HSI–SURF descriptor, the M–SURF descriptor [2], is computed. First, the main orientation of each keypoint is estimated, and next, the M–SURF descriptor is computed [24]. These steps are characterized by a high computational use of registers due to the high use of mathematical functions to compute the derivatives and to apply the Gaussian filters. As a result, a 64 value spatial descriptor for each keypoint is generated. Finally, the spectral signature of each keypoint is assigned to the spectral part of the descriptor.

Algorithm 5 Keypoint description stage.

Input: Band **B**.
Output: A set of keypoints **K**.

1: **for each** keypoint **in K do**
2: < Calculate the main orientation > ▷ GM
3: < Compute the M–SURF descriptor > ▷ GM
4: Append the spectral signature
5: **end for**

3.2.4 Keypoint matching and band combination stages

After the keypoint description, two sets of keypoints are stored, one for each image. As we mentioned before, the HSI–SURF descriptor for each keypoint is made up of two parts: a spatial and a spectral one. The matching stage has the objective of finding pairs of keypoints in each band of both images. The keypoints of each band are independently matched. Two conditions must be fulfilled in order to consider the matching of a pair of keypoints, one is based on a ratio of Euclidean distances, the other is based on the cosine similarity

between the spectral signature of the keypoints [24]. The pseudocode of the keypoint matching and the band combination stages is shown in Algorithm 6.

For each band, we have two matrices of keypoint descriptors $\mathbf{K}_1^b$ and $\mathbf{K}_2^b$ of dimension $64 \times n_1^b$ and $64 \times n_2^b$, respectively. n_1^b and n_2^b are the number of keypoints detected in each band b and image, i.e., the keypoint descriptors are stored by columns. Given a keypoint p of $\mathbf{K}_1^b$, we have to find the two nearest keypoints $q_0,\ q_1$ of $\mathbf{K}_2^b$ to that keypoint. An approximation of the Euclidean distance is used to rewrite it to involve matrix operations [10]:

$$d^2_{Euclidean}(p,q) = (p-q)^\top (p-q) = \|p\|^2 + \|q\|^2 - 2p^\top q \tag{1}$$

where $\|.\|$ is the Euclidean norm. We can extend it to handle set of keypoints,

$$d^2_{Euclidean}(\mathbf{K}_1^b, \mathbf{K}_2^b) = \mathbf{SN}_1^b + \mathbf{SN}_2^b - 2\mathbf{K}_1^b {\mathbf{K}_2^b}^{\mathrm{T}} \tag{2}$$

where $\mathbf{SN}_1^b$ and $\mathbf{SN}_2^b$ are matrices of one row composed by Euclidean norm of each column of $\mathbf{K}_1^b$ and $\mathbf{K}_2^b$, respectively. The square norm of the reference keypoints and target keypoints are computed in global memory (see Algorithm 6, lines 3-4).

Next, $\mathbf{H} = -2\mathbf{K}_1^b {\mathbf{K}_2^b}^{\mathrm{T}}$ is calculated using the function *cublasDgemm* of CUBLAS library [21] (strategies 4 and 10) (see Algorithm 6, line 5). Then, each i element of $\mathbf{SN}_2^b$ is added to every element of the i^{th} row of $\mathbf{H}$ using shared memory (strategy 4). The result is stored in $\mathbf{D}$ (see Algorithm 6, line 6).

A modified insertation sort algorithm is used to find the 2 smallest values for each column of matrix $\mathbf{D}$, i.e., for each reference keypoint (see Algorithm 6, line 7). Each thread sorts all the computed distances for a reference keypoint. Assuming that the first smallest value of the array of distances D is already sorted (in $D[0]$), an element i will be inserted in the correct position only if $D[i] < D[0]$. This modification reduces the number of stores to global memory minimizing the memory usage.

Finally, the i element of $\mathbf{SN}_1^b$ is added to the 2 smallest values in $\mathbf{D}$ and the square root is computed. It can be done after the sorting because these operations do not influence the distance order [10].

With the two nearest reference keypoints $q_0,\ q_1$ to the keypoint p, to consider the match of p and q_0 two conditions must be fulfilled (see Algorithm 6, lines 9-15). This part of the implementation is computed on CPU due to the low computational complexity of the task. First, the ratio of the Euclidean distances between keypoint p of the reference image and the two nearest keypoints q and r of the target image must be smaller than a distance ratio D_{ratio}. Second, the cosine similarity between the spectral signature of keypoints p and q must be lower than a value R. The spectral part of the descriptor allows removing false matches. D_{ratio} was fixed to 0.6 and R is fixed to 0.9. Both values were chosen experimentally. In the case of considering a match, the matched keypoints are inserted in the array $\mathbf{M}$ (see Algorithm 6, line 12).

The matched keypoints extracted corresponding to the different selected bands are joined at the end of this stage. This way, different features not

present in all bands are taken into account. Finally, the repeated matches are removed (see Algorithm 6, line 17).

Algorithm 6 Keypoint matching and band combination.

Input: Matrix of keypoint descriptors $\mathbf{K}$ for each selected band of both images.
Output: Array $\mathbf{M}$ of N matched pairs.
Parameters: Distance ratio D_{ratio}, spectral similarity ratio R.

1: $N \leftarrow 0$
2: **for each** band b **in** images $\mathbf{B}_1$ and $\mathbf{B}_2$ **do**
3: $< \text{Compute the Euclidean norm of each column of } \mathbf{K}_1^b >\rightarrow \mathbf{SN}_1^b$ ▷ GM
4: $< \text{Compute the Euclidean norm of each column of } \mathbf{K}_2^b >\rightarrow \mathbf{SN}_2^b$ ▷ GM
5: $<\text{Compute } -2\mathbf{K}_1^b{\mathbf{K}_2^b}^{\mathrm{T}} \text{ using cuBLAS} >\rightarrow \mathbf{H}$ ▷ GM+SM
6: $< \mathbf{SN}_2^b + \mathbf{H} >\rightarrow \mathbf{D}$ ▷ GM+SM
7: $<$ For each column of matrix $\mathbf{D}$ (i.e. for each reference keypoint) finds the 2 smallest values $>$ ▷ GM
8: $< \sqrt{\mathbf{SN}_1^b + \mathbf{D}} >\rightarrow \mathbf{D}$ ▷ GM
9: **for each** keypoint p **in** $\mathbf{K}_1^b$ **do**
10: Select $q_0,\ q_1$ as the two nearest keypoints of $\mathbf{K}_2^b$ to the keypoint p according $\mathbf{D}$
11: **if** $\frac{D[q_0][p]}{D[q_1][\pi]} < D_{ratio}$ **and** $\mathrm{d}_{\mathrm{Cosine}}(f_p, f_{q_0}) < R$ **then**
12: Match p and $q_0 \rightarrow M[N]$
13: $N \leftarrow N + 1$
14: **end if**
15: **end for**
16: **end for**
17: Remove repeated matches in $M[N]$

3.2.5 Registration stage

Once the correspondence between keypoints present in both images is found, the geometrical transformation to map the target image to the reference image must be computed. In this last stage, an exhaustive search based on histograms is performed to register the images [24]. For each combination of two matched pairs (4 keypoints), scale factor, rotation angle, and translation parameters are calculated. Then, a selection using histograms is performed. This stage is computed in CPU due to its low computational cost.

4 Results

This section presents the results obtained. First, the test images, the PC and the GPU used are described. The results are first shown in terms of registration accuracy showing that the same results as in CPU are obtained by the GPU registration algorithm proposed. Then, the results are analyzed in terms of execution time for the GPU version comparing to an efficient OpenMP algorithm executed in CPU. Finally, some insights on the results obtained are given.

4.1 Test Images and Experimental Setup

The algorithm was evaluated on a PC with a quad-core Intel Xeon E5–2623v4 at 2.6 GHz and 128 GB of RAM under Ubuntu 16.04.6. The CPU version was compiled using the gcc and the g++ 5.4.0 version. Regarding the GPU implementation, the CUDA code runs on a Tesla P40 with 30 SMs and 128 CUDA cores each. The CUDA code was compiled using nvcc with version 9.0.176, as well as the CUB 1.8.0 version. Performance results in terms of registration precision, computation time and speedup are presented. In the computation times and speedup results, we provide the average of ten independent executions for each experiment.

The algorithm was evaluated over seven different scenes covering a variety of land uses. Information about the images can be seen in Table 1. They present different sizes, number of bands and resolutions in order to test the algorithm in different conditions.

Table 1: Sensor, size, number of spectral bands, resolution (m/pixel), and location of the test hyperspectral images.

Scene	Sensor	Size	Bands	Spatial Resolution
Indian Pines	AVIRIS	145×145	220	20
Salinas Valley	AVIRIS	512×217	204	3.7
Pavia University	ROSIS–03	610×340	103	1.3
Pavia Centre	ROSIS–03	1096×715	102	1.3
Santa Barbara Front 2009	AVIRIS	900×470	224	16.4
Santa Barbara Front 2010	AVIRIS	900×470	224	11.3
Jasper Ridge 2006	AVIRIS	1286×588	224	3.3
Jasper Ridge 2007	AVIRIS	1286×588	224	3.4
Santa Barbara Box 2013	AVIRIS	1024×769	224	15.2
Santa Barbara Box 2014	AVIRIS	1024×769	224	15.2

The first four images were taken by the ROSIS–03 (Reflective Optics System Imaging Spectrometer) sensor, the Pavia University and Pavia Centre images, and the AVIRIS (Airbone Visible/Infrared Imaging Spectometrer) sensor, the Indian Pines and Salinas Valley images. For these four images as only one image of the scene is available, the set of target images to evaluate the algorithm in terms of registration precision is created by applying different rotation angles and scale factors to the original images. This way, we can test the method in a controlled environment.

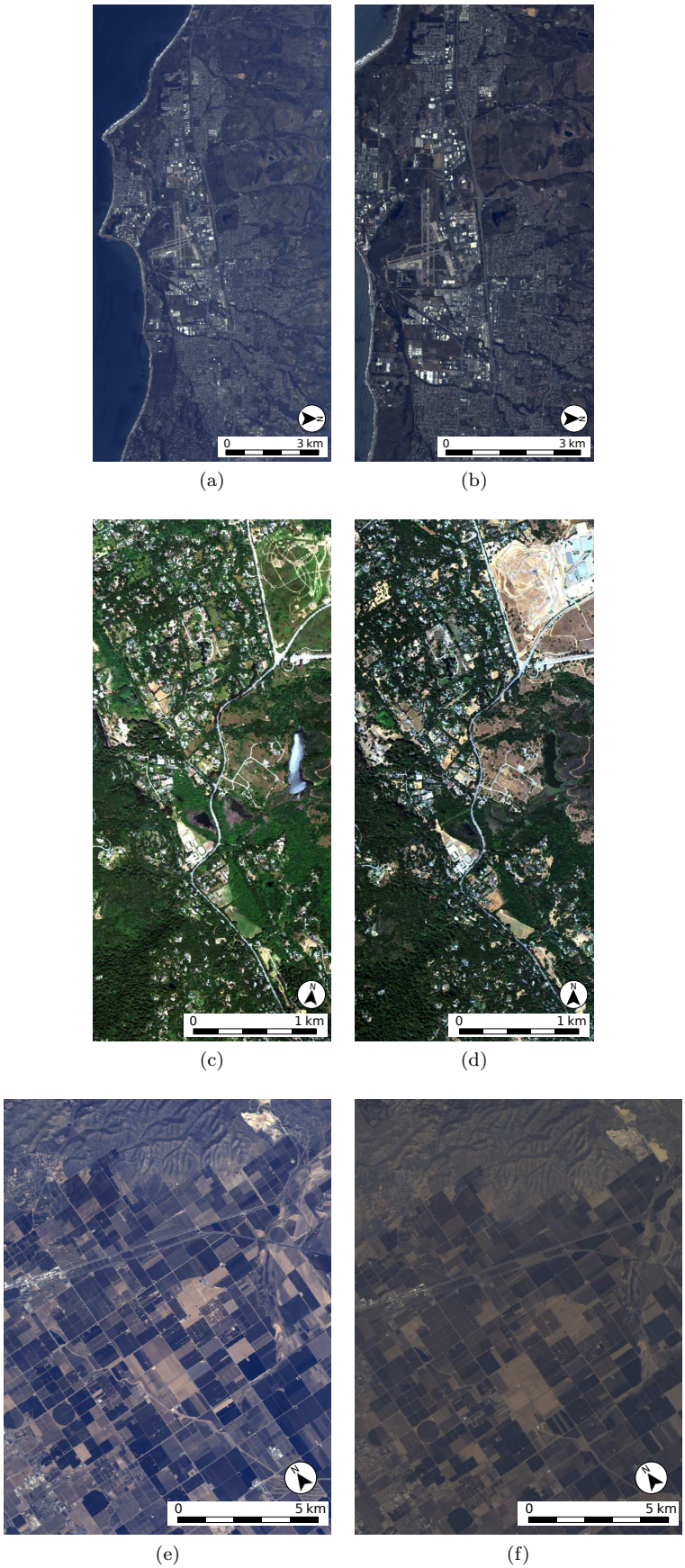

Fig. 2: Pairs of test hyperspectral images taken by AVIRIS: (a) Reference Santa Barbara Front image taken on 30 March 2009, (b) Target Santa Barbara Front image taken on 30 April 2010, (c) Reference Jasper Ridge image taken on 5 December 2006, (d) Target Jasper Ridge image taken on 13 August 2007, (e) Reference Santa Barbara Box image taken on 11 April 2013, and (f) Target Santa Barbara Box image taken on 16 April 2014.

The last six images correspond to three different scenes and are well–known hyperspectral images in the remote sensing area. These were taken by the AVIRIS sensor at different year. They present changes in illumination, vegetation, buildings, roads, etc., as well as different scale factor, rotation angle and translation. Moreover, different rotation angles and scale factors are applied to the second image of each pair in order to extender our experiment. An RGB image of these images can be seen in Figure 2. The complete dataset is available in `https://gitlab.citius.usc.es/hiperespectral/RegistrationRepository`.

4.2 Performance Results in Terms of Registration Precision

In this section, the evaluation of HSI–KAZE in terms of registration precision is presented. The registration process consists in registering one image, the reference image, with respect to second image, the target image, of the scene. In order to increase the experimental range, different scale factors and rotation angles are applied to the target image of each pair. The scale ranges go from $1/15\times$ to $24.0\times$ in increments of 0.5. In total, 63 scale factors. For each scale factor, 72 rotation angles are applied. From 0 to 360 in increments of 5. As a result, a total of $4,464$ cases was proved for each scene.

Table 2 summarizes the successfully registered cases for each scene using HSI–KAZE on GPU. The implementation on GPU achieves the same results as the CPU version. A detailed comparison with other methods of literature and an exhaustive evaluation in terms of registration precision can be seen in [24].

Table 2: Successfully registered cases for each scene using HSI–KAZE on GPU. The number in parentheses summarizes the number of scales that were correctly registered for all angles. If an angle is incorrectly registered, the whole scale factor is considered incorrect, i.e., this case is not included in the table.

Scene	Range of scales factors correctly registered
Pavia University	$1/11\times$ to $13.0\times$ (35)
Pavia Centre	$1/15\times$ to $24.0\times$ (61)
Indian Pines	$1/4\times$ to $5.0\times$ (12)
Salinas Valley	$1/7\times$ to $6.0\times$ (17)
Jasper Ridge	$1/11\times$ to $12.0\times$ (33)
Santa Barbara Front	$1/8\times$ to $9.0\times$ (24)
Santa Barbara Box	$1/9\times$ to $8.5\times$ (24)
Number of scalings (average)	(29.43)

4.3 Performance Results in Terms of Computation Times

In this section, the performance results in terms of computation times for HSI–KAZE are presented. The procedure test is the same as in previous section but the last correctly registered scale is considered for each scene according to Table 2.

In order to compare the GPU implementation of the algorithm according to a high performance baseline, an OpenMP implementation was developed. The achieved speedup over the CPU one thread for the OpenMP implementation for all scenes changing the number of threads is presented in Figure 3. The achieved speedup is higher as the number of threads increase. The algorithm on CPU scales better for the bigger images, i.e., for Santa Barbara Line and Santa Barbara Front.

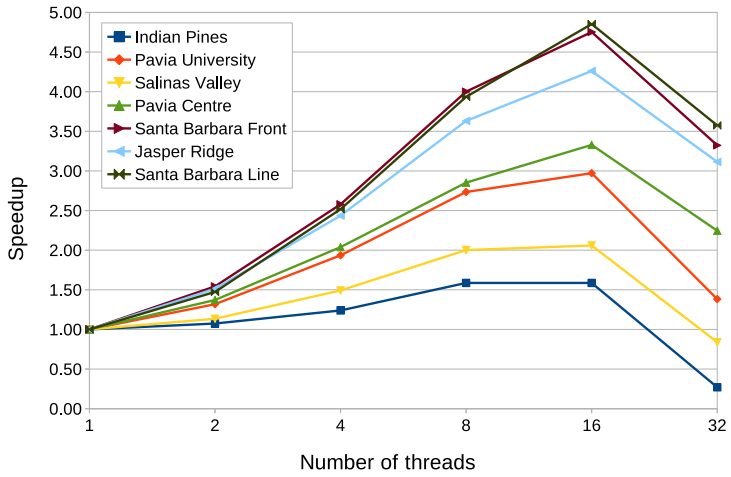

Fig. 3: Speedup for the OpenMP code when the number of threads increases with respect to the OpenMP version for only one thread.

The CPU and GPU execution times and speedup implementation per stage of the algorithm are presented in Table 3 for the registration of the Jasper Ridge scene. The CPU execution times correspond to the best CPU implementation, i.e., considering 16–threads. The speedup of the GPU version is also calculated with respect to the 16–thread OpenMP execution time.

The highest speedup, 47.44×, is achieved in the band selection stage. The computation of the entropy for each band is specially well adapted to the SIMT computation model of GPU. Up to 5 GPU optimization strategies are used at this stage: the use of shared memory and vectorial instructions to optimize the memory access, the use of warp–level primitives avoiding shared memory latency, the use of atomic operation to avoid race conditions, and the efficient computation using the optimized CUB library.

A high speedup is also achieved in the keypoint matching and band combination stages. The approximation of the Euclidean distance using matrix computations allow us to find the 2-nearest keypoints for all the reference keypoints in parallel and efficiently. Two reasons are the key for this high computational efficiency. First, the matrix multiplication computed as part of this approximation is the key to the high degree of parallelism achieved. Second, the use of a modified version of the insertation sort algorithm reduces the memory usage.

In the keypoint detection stage, a speedup of 15.68× is achieved thanks to the use of the proposed optimization strategies, highlighting the change in the order of the operations in the keypoint location substage in order to efficiently exploit the SIMT programming model of the GPU.

Because of the intensive use of registers required in the keypoint description, the speedup in this stage is 2.41×.

Table 3: OpenMP CPU and CUDA GPU execution times (in seconds) and speedup over the OpenMP 16 threads implementation per stage of the HSI–KAZE algorithm for Jasper Ridge scene.

Stage	CPU (s)	GPU (s)	Speedup
Band selection	25.38	0.54	47.44×
Keypoint detection	73.69	4.70	15.68×
Keypoint description	8.73	3.63	2.41×
Keypoint matching and Band combination	61.95	1.94	31.90×
Registration	0.01s	-	-

Table 4 compares the execution times for all scenes between the OpenMP 16–threads version and the CUDA GPU implementation. When the image size is larger and a higher number of keypoints are detected, higher speedups are achieved. For example, for the images of 600 MiB and 1,000,000 detected keypoints, a speedup around 13.00× is achieved.

Table 4: OpenMP CPU and CUDA GPU execution times (in seconds) and speedup over the OpenMP 16 threads implementation for HSI–KAZE for each scene.

Scene	Size (MiB)	Number of detected keypoints	CPU (s)	GPU (s)	Speedup
Indian Pines	18	34,407	3.00	0.91	3.31×
Pavia University	82	305,652	23.03	3.10	7.43×
Salinas Valley	87	123,922	9.59	1.74	5.51×
Pavia Centre	305	1,081,454	109.65	9.86	11.12×
Santa Barbara Front	362	699,935	79.05	6.92	11.42×
Jasper Ridge	647	1,168,880	170.50	12.60	13.53×
Santa Barbara Box	673	1,097,362	168.70	12.12	13.92×

The achieved and theoretical GPU occupancy for each stage of the HSI–KAZE algorithm for the Jasper Ridge scene is shown in Table 5. We obtained these measures using the NVIDIA Profiling Tool (NVPROF) which provides the metrics at kernel level. For this reason, the occupancy values at stage level are calculated by weighting each kernel occupancy by its execution time with respect to the total execution time of the stage. In all stages, the achieved and theoretical GPU occupancy values are limited by registers per multiprocessor due to high number of mathematical operations required, especially in the computation of the spatial descriptor. It is important to note that higher occupancy does not imply necessarily higher performance. There is a point above which additional occupancy does not improve performance [22]. HSI–KAZE on GPU was developed prioritizing low execution time over other metrics.

Table 5: Achieved and theoretical GPU occupancy per stage of the HSI–KAZE algorithm for Jasper Ridge scene.

	Achieved Occupancy	Theoretical Occupancy
Band Selection	0.69	0.80
Keypoint detection	0.23	0.96
Keypoint description	0.26	0.30
Keypoint matching and Band combination	0.65	0.76

5 Conclusions

In this paper a CUDA GPU implementation of the HSI–KAZE method for registering hyperspectral images is presented. The algorithm is based on KAZE and A–KAZE features exploiting the spectral information using different bands and incorporating spectral information to the keypoint descriptors. The proposed implementation efficiently exploits the thousands of available threads on

the GPU, obtaining a considerable reduction in execution time as compared to the OpenMP CPU implementation. The band selection and keypoint matching stages are the most optimized in GPU thanks to the use of the proposed optimization strategies, highlighting the use of parallel reductions using warp–level primitives, atomic operations, vectorial instructions and shared memory. Experiments show a speedup of 13× over an efficient multicore implementation in OpenMP in the case of hyperspectral images of size 600 MB.

Acknowledgements This work was supported in part by the Consellería de Educación, Universidade e Formación Profesional [grant numbers GRC2014/008, ED431C 2018/19, and ED431G/08] and Ministerio de Economía y Empresa, Government of Spain [grant number TIN2016-76373-P] and by Junta de Castilla y Leon - ERDF (PROPHET Project) [grant number VA082P17]. All are co–funded by the European Regional Development Fund (ERDF). The work of Álvaro Ordóñez was also supported by Ministerio de Ciencia, Innovación y Universidades, Government of Spain, under a FPU Grant [grant numbers FPU16/03537 and EST18/00602].

References

1. Acción, A., Argüello, F., B. Heras, D.: Extended anisotropic diffusion profiles in gpu for hyperspectral imagery. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing pp. 1–13 (2019). DOI 10.1109/JSTARS.2019.2939857
2. Agrawal, M., Konolige, K., Blas, M.R.: Censure: Center surround extremas for realtime feature detection and matching. In: European Conference on Computer Vision, pp. 102–115. Springer (2008)
3. Al-khafaji, S.L., Zhou, J., Zia, A., Liew, A.W.C.: Spectral-spatial scale invariant feature transform for hyperspectral images. IEEE Transactions on Image Processing **27**(2), 837–850 (2017)
4. Alcantarilla, P.F., Bartoli, A., Davison, A.J.: KAZE features. In: European Conference on Computer Vision, pp. 214–227. Springer (2012)
5. Alcantarilla, P.F., Nuevo, J., Bartoli, A.: Fast explicit diffusion for accelerated features in nonlinear scale spaces. IEEE Trans. Patt. Anal. Mach. Intell **34**(7), 1281–1298 (2011)
6. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (SURF). Computer vision and image understanding **110**(3), 346–359 (2008)
7. Dorado-Muñoz, L.P., Velez-Reyes, M., Mukherjee, A., Roysam, B.: A vector sift detector for interest point detection in hyperspectral imagery. IEEE transactions on Geoscience and Remote sensing **50**(11), 4521–4533 (2012)
8. Fan, Z., Vetter, C., Guetter, C., Yu, D., Westermann, R., Kaufman, A., Xu, C.: Optimized GPU implementation of learning-based non-rigid multi-modal registration. In: Medical Imaging, pp. 69142Y–69142Y. International Society for Optics and Photonics (2008)
9. G. Bascoy, P., Quesada-Barriuso, P., B. Heras, D., Argüello, F.: Wavelet-based multicomponent denoising profile for the classification of hyperspectral images. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **12**(2), 722–733 (2019). DOI 10.1109/JSTARS.2019.2892990
10. Garcia, V., Debreuve, E., Barlaud, M.: Fast k nearest neighbor search using GPU. In: 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pp. 1–6. IEEE (2008)
11. Hasan, M., Jia, X., Robles-Kelly, A., Zhou, J., Pickering, M.R.: Multi-spectral remote sensing image registration via spatial relationship analysis on SIFT keypoints. In: Geoscience and Remote Sensing Symposium (IGARSS), 2010 IEEE International, pp. 1011–1014. IEEE (2010)
12. Li, Q., Wang, G., Liu, J., Chen, S.: Robust scale-invariant feature matching for remote sensing image registration. IEEE Geoscience and Remote Sensing Letters **6**(2), 287–291 (2009)

13. Li, Y., Li, Q., Liu, Y., Xie, W.: A spatial-spectral sift for hyperspectral image matching and classification. Pattern Recognition Letters (2018)
14. Lowe, A., Harrison, N., French, A.P.: Hyperspectral image analysis techniques for the detection and classification of the early onset of plant disease and stress. Plant methods **13**(1), 80 (2017)
15. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2), 91–110 (2004)
16. Mehmet, F., Yardimci, Y., Temlzel, A., et al.: Registration of multispectral satellite images with orientation-restricted sift. In: Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009, vol. 3, pp. III–243. IEEE (2009)
17. Mukherjee, A., Velez-Reyes, M., Roysam, B.: Interest points for hyperspectral image data. IEEE Transactions on Geoscience and Remote Sensing **47**(3), 748–760 (2009)
18. Munir, M., Wilson, D.I., Yu, W., Young, B.: An evaluation of hyperspectral imaging for characterising milk powders. Journal of food engineering **221**, 1–10 (2018)
19. Muyan-Ozcelik, P., Owens, J.D., Xia, J., Samant, S.S.: Fast deformable registration on the GPU: A CUDA implementation of demons. In: Computational Sciences and Its Applications, 2008. ICCSA'08. International Conference on, pp. 223–233. IEEE (2008)
20. NVIDIA: CUB Library (2018). URL https://nvlabs.github.io/cub/
21. NVIDIA: cuBLAS Library User's Guide (2019). URL https://docs.nvidia.com/pdf/CUBLAS_Library.pdf
22. NVIDIA: CUDA C Best Practices Guide (2019). URL https://docs.nvidia.com/pdf/CUDA_C_Best_Practices_Guide.pdf
23. NVIDIA: NVIDIA Performance Primitives (NPP) v10.1.1 User's Guide (2019). URL https://docs.nvidia.com/cuda/archive/10.1/npp/index.html
24. Ordóñez, Á., Argüello, F., B. Heras, D.: Alignment of hyperspectral images using KAZE features. Remote Sensing **10**(5) (2018). DOI 10.3390/rs10050756
25. Ordóñez, Á., Argüello, F., B. Heras, D.: GPU accelerated FFT-based registration of hyperspectral scenes. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (2017)
26. Ramkumar B.and Laber, R., Bojinov, H., Hegde, R.S.: GPU acceleration of the KAZE image feature extraction algorithm. Journal of Real-Time Image Processing (2019)
27. Sah, S., Vanek, J., Roh, Y., Wasnik, R.: GPU accelerated real time rotation, scale and translation invariant image registration method. In: Image Analysis and Recognition, pp. 224–233. Springer (2012)
28. Sanders, J., Kandrot, E.: CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional (2010)
29. Shams, R., Sadeghi, P., Kennedy, R., Hartley, R.: Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. Computer methods and programs in biomedicine **99**(2), 133–146 (2010)
30. Teke, M., Temizel, A.: Multi-spectral satellite image registration using scale-restricted surf. In: Pattern Recognition (ICPR), 2010 20th International Conference on, pp. 2310–2313. IEEE (2010)
31. Vince, R., More, S.S.: Hyperspectral imaging for detection of parkinson's disease (2018). US Patent App. 10/098,540
32. Yi, Z., Zhiguo, C., Yang, X.: Multi-spectral remote image registration based on sift. Electronics Letters **44**(2), 107–108 (2008)
33. Zhang, Y., Zhou, P., Ren, Y., Zou, Z.: GPU-accelerated large-size VHR images registration via coarse-to-fine matching. Computers & Geosciences **66**, 54–65 (2014)
34. Zitova, B., Flusser, J.: Image registration methods: a survey. Image and vision computing **21**(11), 977–1000 (2003)