This a post-print of the article “GPU Accelerated FFT-based Registration of Hyperspectral Scenes” published in the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

The published article is available on https://doi.org/10.1109/JSTARS.2017.2734052.

Á. Ordóñez, F. Argüello and D. B. Heras, "GPU Accelerated FFT-Based Registration of Hyperspectral Scenes," in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 4869-4878, Nov. 2017.
DOI: 10.1109/JSTARS.2017.2734052

GPU Accelerated FFT-based Registration of Hyperspectral Scenes

Álvaro Ordóñez, Francisco Argüello, and Dora B. Heras

***Abstract*—Registration is a fundamental previous task in many applications of hyperspectrometry. Most of the algorithms developed are designed to work with RGB images and ignore the execution time. This paper presents a phase correlation algorithm on GPU to register two remote sensing hyperspectral images. The proposed algorithm is based on principal component analysis, multilayer fractional Fourier transform, combination of log-polar maps, and peak processing. It is fully developed in CUDA for NVIDIA GPUs. Different techniques such as the efficient use of the memory hierarchy, the use of CUDA libraries and the maximization of the occupancy have been applied to reach the best performance on GPU. The algorithm is robust achieving speedups in GPU of up to $240.6\times$.**

***Index Terms*—Hyperspectral imaging, image registration, Fourier transforms, GPU, CUDA, remote sensing.**

I. Introduction

AT THE PRESENT, thanks to the recent advances on the development of image sensor technologies the capture of hyperspectral images is easier than two decades ago [1]. These images are hundreds of bands taken over a wide wavelength range. The high spectral resolution that characterizes the hyperspectral images makes it possible to address many applications such as agriculture [2], medicine [3], target detection [4], environmental monitoring, change detection and quality control, among others.

Image registration is a fundamental task in many of these applications to overlay two or more images taken at different times, from different viewpoints and possibly under different lighting conditions [5], [6]. The registration problem that we study consists in estimating the translation, rotation and scaling parameters between a reference image and a second image of the same scene that have been taken at different times. Several different algorithms to solve automatic image registration have been proposed [7]–[10]. Fourier algorithms are very efficient because the fast Fourier transform (FFT) can be used to compute the cross-correlation (phase correlation) between the two images. Additionally, they are resilient to noise, occlusions, and other defects typical of medical or remote-sensing images. In addition, these algorithms are suitable to be implemented on commodity GPUs because the operations are mostly point by point and can be computed by independent threads, i.e., without data dependencies.

A FFT-based registration technique was proposed by Chen et al. [11] for translation, rotation and scale-invariant image registration using a log-polar grid. This algorithm is also known as Fourier-Mellin invariant symmetric phase-only matched filtering (FMI-SPOMF) [12]–[14]. Different Fourier-Mellin-based techniques have been proposed: Keller et al. introduce the pseudopolar Fourier transform (PPFT) [15], Liu et al. propose the pseudo-log-polar Fourier Transform (PLPFT) [16], and an octa-log-polar version of the PLPFT was proposed by Wu et al [17], among others. Recently, a simpler approach was followed by Pan et al. [18], who proposed an adaptable multilayer fractional Fourier transform (MLFFT) for image registration with lower interpolation errors in both polar and log-polar grids.

However, most of these registration algorithms ignore time performance. In real-time applications, e.g., detection in a search and rescue scenario, disaster and damage control, reconnaissance and surveillance, etc., the execution time becomes crucial. GPUs have been demonstrated to be appropriate in many different problems for a computationally efficient hyperspectral processing: classification [19], spectral unmixing [20], target detection [21] and segmentation [22]. In the literature some registration algorithms for two-dimensional images in GPU have been proposed [23]–[27], but none of them is applied to hyperspectral images, that contain more than a hundred times the data of a typical image.

In this paper we extend the algorithm presented in [28], which is called HYperspectral Fourier-Mellin algorithm (HYFM), to GPU processing to achieve the best performance for use in real-time applications. The algorithm exploits the information contained in the different bands of the images and is based on principal component analysis, multilayer fractional Fourier transform and the combination of log-polar maps. The whole algorithm has been developed in CUDA to get a full exploitation of the GPU architecture. Execution times in GPU are also compared to a CPU implementation of the algorithm.

The paper is organized as follows: Section II describes the algorithm; the GPU implementation of the algorithm is explained in Section III; the results are discussed in Section IV; and, finally, in Section V we present the conclusions.

II. Hyperspectral Fourier-Mellin registration

In this section we present the algorithm to register two hyperspectral images (reference and target) on GPU. The algorithm is based on the computation of Fourier transforms and log-polar maps and on the processing of the peaks detected in these maps. When facing the processing of hyperspectral images a key issue is that they are highly correlated in the spatial and spectral dimensions. This complicates the design of new algorithms and increases the processing times.

The authors are with the Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Spain. (E-mail: alvaro.ordonez@usc.es, francisco.arguello@usc.es, dora.blanco@usc.es)

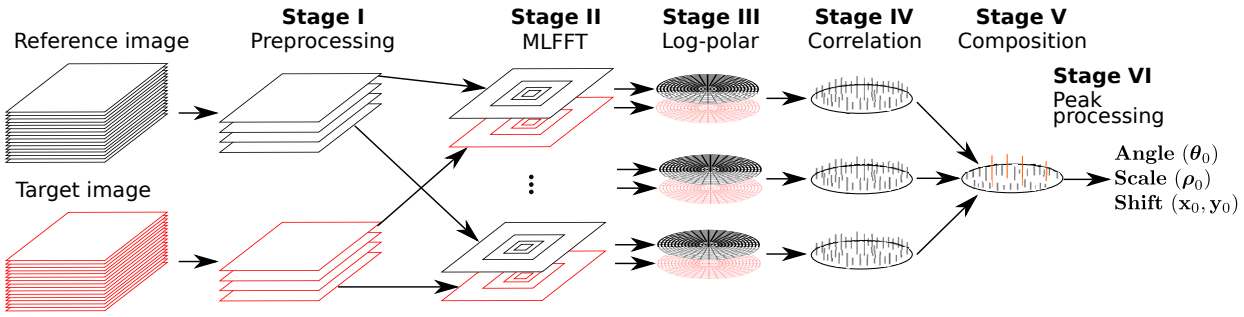


Fig. 1. Proposed HYFM scheme for registration of two hyperspectral images.

The registration algorithm that is projected to GPU is outlined in Fig. 1. The algorithm comprises six main stages. In order to deal with the high dimensionality of the images and to perform dimensionality reduction, a Principal Component Analysis (PCA) [29] is performed in the first stage. The registration is then applied to pairs of PCA components.

Next, the MLFFT technique is calculated for each PCA component to obtain a more robust log-polar maps in the third stage. In order to take into account the hyperspectral character of the images, the phase correlation maps are computed for each pair of log-polar maps in the fourth stage. Then these maps are combined in stage five. Finally, in the last stage, the highest peaks are examined to determine the scaling, rotation, and translation parameters.

A. Stage I. Preprocessing

In this first stage, a Blackman window is applied to each hyperspectral band to remove the high frequencies that deteriorate the precision of the registration [30]. Then, a PCA computation starts to reduce the dimensionality of each hyperspectral image, consisting of a large number of inter-related variables, while retaining as much as possible of the variation present in the hyperspectral image. This is achieved by transforming to a new set of variables, N_{PCA} components for each image, which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables [31]. PCA computation is done by Eigenvalue Decomposition (EVD) of the covariance matrix of the image.

B. Stage II and III. MLFFT and log-polar

Firstly, in the *MLFFT* stage, a simple high-pass emphasis filter transfer function is applied to each PCA component. Its purpose is to reduce the aliasing effects that degrade the precision of image registration based on phase correlation and that are strongest in the low frequencies [12].

A challenge when computing the log-polar maps is to evaluate the log-polar Fourier transform efficiently and accurately. We use the MLFFT [18], since it has lower interpolation errors in both polar and log-polar grids. The MLFFT uses a multilevel grid to approximate the log-polar grid. Since the point spacing in the log-polar map follows a logarithmic law relative to the radius, MLFFT uses a denser grid in the lower levels. Each level of the MLFFT can be computed by means of a fractional Fourier transform (FRFT) with a particular parameter α_l. The FRFT for a complex sequence $\boldsymbol{x}$ of size m, where m is a power of 2, is defined as

$$G_k(\boldsymbol{x},\alpha) = \mathrm{e}^{-\pi i k^2\alpha}\sum_{j=0}^{2m-1} x_j \mathrm{e}^{-\pi i j^2\alpha}\mathrm{e}^{\pi i (k-j)^2\alpha} \tag{1}$$

$$= \mathrm{e}^{-\pi i k^2\alpha}\sum_{j=0}^{2m-1} x_j y_j z_{k-j}, \tag{2}$$

where the $2m$-long complex sequences $\boldsymbol{y}$ and $\boldsymbol{z}$ are as follows

$$y_j = \mathrm{e}^{-\pi i j^2\alpha}, \qquad 0 \le j < m, \tag{3}$$

$$y_j = 0, \qquad m \le j < 2m, \tag{4}$$

$$z_j = \mathrm{e}^{\pi i j^2\alpha}, \qquad 0 \le j < m, \tag{5}$$

$$z_j = \mathrm{e}^{\pi i (j-2m)^2\alpha}, \qquad m \le j < 2m, \tag{6}$$

and where

$$x_j = 0, \qquad m \le j < 2m. \tag{7}$$

To calculate the 2D-FRFT of an image $\mathbf{I}$, first, we apply a 1D fractional Fourier transform (2) to each row and then we apply a 1D fractional Fourier transform to each column [28]. Thus, the L-level MLFFT grid can be written as

$$\boldsymbol{P} = \bigcup_{l=1}^{L} \boldsymbol{P}_l \tag{8}$$

where $\boldsymbol{P}_l = \{P_{lp}; 0 \le p < N\}$, $P_{lp} = G_p(\boldsymbol{x}, \alpha_l)$, $\boldsymbol{x}$ is a row or column of image $\mathbf{I}$, and $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_L = 1$. P_{lp} is first calculated for all the N rows of $\mathbf{I}$ and then for all the N columns. Usually, a number of levels L between two and four is used. Finally, the log-polar grid is generated by interpolating on the MLFFT grid.

An example of calculation of a 4-level MLFFT over an image is illustrated in Fig. 2. First, four $2N\times 2N$-long complex sequences $\boldsymbol{y}$ and $\boldsymbol{z}$ are calculated, one for each α where each point corresponds to an exponential as indicated in (3)-(6). Secondly, the size of the image $\mathbf{I}$, in our case a PCA component of the image, is expanded to $2N \times 2N$ size by allocating the image in the upper left corner and padding the rest with zeros as indicated in (7). Finally, the four FRFTs are computed and the MLFFT grid is calculated using (8).

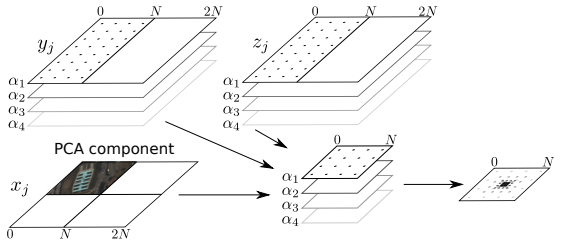


Fig. 2. Example of 4-level MLFFT grid calculation over a PCA component of a hyperspectral image.

C. Stage IV. Correlation

Phase correlation is used in the *Correlation* stage and in the *Peak processing* stage. Fourier based automatic registration relies on the Fourier shift theorem, which states that a circular shift in the spatial domain is equivalent to a phase ratio in the frequency domain.

Let $I_1(x, y)$ and $I_2(x, y)$ be two images, where (x, y) are the spatial coordinates, being the second image a translated replica of the first one,

$$I_2(x, y) = I_1(x + x_0, y + y_0), \tag{9}$$

the shift theorem states that their Fourier transforms are related by

$$F_2(u, v) = \mathrm{e}^{i2\pi(ux_0+vy_0)} F_1(u, v), \tag{10}$$

where (u, v) are the coordinates in the transform space and i the imaginary unit.

Applying the phase correlation in the frequency domain,

$$\frac{F_1(u, v)F_2^*(u, v)}{|F_1(u, v)F_2^*(u, v)|} = \mathrm{e}^{-i2\pi(ux_0+vy_0)}, \tag{11}$$

where $*$ denotes the complex conjugate, and finally, computing the inverse Fourier transform (IFFT), we obtain a matrix with approximate zeros everywhere except for a peak at coordinates (x_0, y_0), that can be used to register the images. To recover the rotation and scaling parameters, Chen et al. [11] proposed to use the log-polar Fourier transform (Fourier-Mellin transform).

D. Stage V. Composition

The registration experiments have revealed that combining the log-polar maps after phase correlation allows achieving the best results, registering a higher number of scales. This combination is computed averaging the different phase correlation maps calculated for each pair of PCA components. This approach allows each PCA pair to contribute to the cross correlation independently.

E. Stage VI. Peak processing

Once the phase correlation over the log-polar map and the combination of maps of the different pairs of PCA components have been performed, a log-polar map is obtained with approximate zeros everywhere except several peaks. If the images to register were exactly equal and neither aliasing nor border effects occur, only one peak should be obtained. In practice, a great number of peaks in the log-polar map are obtained.

The peak processing stage has two steps: process the peaks of phase correlation in the log-polar map to detect the scale factor and the rotation angle, and process the peaks of phase correlation in the cartesian grid (after correcting the scaling and rotation of the target image) to determine the translation.

In order to find the best peak, all the highest peaks in the log-polar map are processed, and the second phase of correlation in the cartesian grid is used to select the best peak. The value of the highest peak in the cartesian grid will determine the correct peak in the log-polar grid. The coordinates of these peaks determine the scale factor ρ_0, the rotation angle θ_0, and the translation parameters (x_0, y_0).

III. Hyperspectral Fourier-Mellin registration on GPU

In this section, we introduce some CUDA programming fundamentals as well as the CUDA implementation of the HYFM algorithm on GPU for the registration of hyperspectral images.

A. CUDA GPU Programming Fundamentals

CUDA is a parallel computing platform and programming model that allows running programs using parallel functions called kernels [32]. 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. A warp is the number of threads that can run concurrently on a multiprocessor. The programmer organizes these threads into a grid of blocks. A grid is a set of blocks that are executed in parallel and which are independent. A block is a set of threads that work together.

In the new architecture called Pascal, the memory hierarchy was changed [33]. The Pascal GP104 architecture provides 96 KB/SM of dedicated shared memory and 48 KB/SM L1/texture cache, i.e., it is not necessary to select a preference of the shared memory and L1 cache split to get the optimal performance in contrast to earlier architectures. The shared memory is only visible by the threads of the block. This unified L1/texture cache acts as a coalescing buffer for memory access. Moreover, Pascal GP104 features a unified 2048 KB L2 cache.

In order to reduce the computational time a set of optimization strategies have been applied:

1) **Reduce the data transfers between the host and device memories.** Implementing an algorithm that runs entirely on GPU allows reducing computation times.
2) **Minimize the use of memory.** Some computations are performed in-place, i.e., the outputs are stored over the space previously allocated for the inputs.
3) **Use of texture memory whenever possible.** The texture memory is a dedicated read-only cache which includes hardware filtering. It can also perform linear floating point interpolation as part of the read process. Compared to a typical CPU caching scheme, the texture memory cache is

optimised for accelerating access patterns, for 2D spatial locality (in the coordinate system of the texture).

4) **Avoid divergences.** Divergences negatively affect performance.
5) **Efficient computation using libraries.** Use the computing libraries implemented in the literature in order to get the best performance.
6) **Use of low-level optimized instructions.** The use of low-level optimized functions provides better performance and better accuracy, especially for repeatedly executed code [34].
7) **Search for the best kernel configurations.** To get the highest possible occupancy is the only way to hide latencies and keep the hardware busy [34]. To achieve this, the maximum block size for each kernel is selected with the requirement that the number of registers and the shared memory usage do not act as limiters of the occupancy.

B. CUDA implementation

In this section, we describe the GPU implementation of the HYFM algorithm. The computational complexity for a d-dimensional $N \times N$ image is $O(d^3 + d^2N^2)$ for the PCA computation and $O(N_{\mathrm{PCA}}N^2 \log N)$ for the remaining tasks, where N_{PCA} is the number of Principal Components (PCs) retained. The maximum amount of global memory required is that corresponding to the size of the largest original image plus the space required to compute their PCA components.

In Fig. 3 the pseudocode for the HYFM registration 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, SM in the shared memory and TM in the texture memory.

The optimization strategies applied globally are the following. The algorithm runs entirely on GPU to reduce the data transfers between host and device memories (CUDA optimization strategy 1 explained in Section III-A). Regarding the data neighborhood required for the computation of each pixel, two situations have been identified. The first situation occurs for most of the computations per pixel, which do not require the values of the neighbors. The other situation occurs when interpolation is required. In this last case, the texture memory is used (see Section III-A, strategy 3). Moreover, most of the computations are in-place, i.e., the input is overwritten by the output (see Section III-A, strategy 2).

Furthermore, optimization strategy 7 (see Section III-A) has been applied using the NVIDIA Visual Profiler tool to analyze each kernel and identify potential performance bottlenecks. In general, a 256-threads block have been chosen with the exception of kernels of PCA for which the block size is 512. Analyzing the resources that each kernel needs, we customize each kernel configuration. This allows us to obtain the best performance and optimal hardware occupancy. Moreover, the chosen block sizes are multiple of the warp size, which is 32 on all current hardware, therefore, this facilitates memory accesses by warps that are aligned to cache lines. In addition,

HYFM registration (GPU algorithm)

Input: Hyperspectral reference image $\mathbf{I}_1$, hyperspectral target image $\mathbf{I}_2$.
Output: Scale factor ρ_0, rotation angle θ_0, and translation (x_0, y_0).
Parameters: number of PCA components N_{PCA}, number of alpha parameters L, FRFT α_l parameters, phase correlation peaks N_{Peaks}.

Preprocessing and initializations
1: $<$ Initialize auxiliary data for high-pass filter, FRFT computation, and log-polar computation $>$ ▷ GM
2: **for each** input image **do**
3: $<$ Apply a Blackman window to each hyperspectral band $>$ ▷ GM
4: $<$ PCA computation $>$ (Detailed in Fig. 4) ▷ GM+SM
5: $<$ Extend each band to $N \times N$ pixels, with N power of 2 $>$ ▷ GM+SM
6: **end for**

MLFFT, log-polar, and correlation (Detailed in Fig. 5)

Composition
7: $<$ Average the log-polar maps $>$ ▷ GM

Peak processing
8: $<$ Sort the log-polar peaks by magnitude $>$ ▷ GM+SM
9: **for** $peak := 0$ (with coordinates ρ' and θ') **to** N_{Peaks} **do**
10: $<$ Scale and rotate (θ' and $\theta' + \pi$) the first PCA of target image $>$ ▷ GM+SM
11: $<$ Compute phase correlation in the cartesian grid $>$ ▷ GM+SM
12: $<$ Select the highest peak (with coordinates x' and y') $>$ ▷ GM+SM
13: **end for**
14: Use the highest peak of all cartesian grids to determine $(\rho_0, \theta_0, x_0, y_0)$

▷ GM: Global Memory, SM: Shared Memory, TM: Texture Memory

Fig. 3. Pseudocode for the HYFM registration in CUDA.

the size of the input images is extended to the closest power of 2 to obtain a more efficient computation of the FFT [35] which is required many times in the algorithm. Also, it allows avoiding divergences (see Section III-A, strategy 4) in the operations performed by the different threads.

1) Stage I. Preprocessing and initializations: This first stage is mostly computed using global memory. As shown in Fig. 3, the algorithm begins with both images stored in global memory.

First, the initialization of some arrays repeatedly required in the high-pass filter, log-polar and FRFT computations are executed (Fig. 3, line 1). Due to their large size, these arrays are stored in global memory. The cosine and sine calculations with a factor π, which are required in these initializations, have been replaced by the optimized instructions *cospi* and *sinpi* (see Section III-A, strategy 6).

Then, a Blackman window is applied to each hyperspectral band of each image (Fig. 3, line 3). It is computed as a point by point multiplication which is carried out in global memory. After this, the PCA computation starts to reduce the dimensionality of a hyperspectral image (Fig. 3, line 4). As mentioned previously, it is done by Eigenvalue Decomposition (EVD) of the covariance matrix of the image. Fig. 4 includes the pseudocode of the PCA algorithm in GPU [36]. The image must be centered by subtracting the average pixel value of the band's pixels (Fig. 4, lines 1-2). This step is performed using the cuBLAS function *cublasSgemv* [37] and the shared memory (see Section III-A, strategies 2 and 5).

The covariance matrix of matrix $\mathbf{I}$ is calculated using the cuBLAS function *cublasSgemm* (see Section III-A, strategy 5). This function performs the $\mathbf{I} \cdot \mathbf{I}^T$ operation (Fig. 4, line 3).

The PCA stage continues with the computation of

PCA computation (GPU algorithm)

Input: Hyperspectral image $\mathbf{I}$ stored in global memory.
Output: PCA components of image $\mathbf{I}$.

Image centering phase
1: $<$ Obtain the sum of all pixels in each band of image $>$ ▷ SM + GM
2: $<$ Image centering $>$ ▷ SM

PCA phase
3: $<$ Calculate the covariance matrix $\mathbf{I} \cdot \mathbf{I}^T$ of centered image $>$ ▷ SM + GM
4: $<$ Calculate the matrix of eigenvectors $\mathbf{V}$ $>$ ▷ SM + GM
5: $<$ Obtain the PCs by $\mathbf{IV}$ $>$ ▷ SM + GM

Fig. 4. Pseudocode for PCA computation in CUDA.

MLFFT, log-polar, and correlation (GPU algorithm)

Input: PCA components of image $\mathbf{I}_1$ and $\mathbf{I}_2$.
Output: N_{PCA} phase correlation maps.
Parameters: number of PCA components N_{PCA}, number of alpha parameters L, FRFT α_l parameters.

1: **for each** PCA component **in** images $\mathbf{I}_1$ and $\mathbf{I}_2$ **do**
2: $<$ Apply a high-pass filter to each component $>$ ▷ GM+SM
3: (1) Compute FFT
4: (2) High-pass kernel
5: (3) Compute IFFT
6: **for each** α_l **from** $l = 1$ **to** L **do**
7: $<$ Compute FRFT with parameter α_l $>$ ▷ GM+SM
8: $<$ Compute module $>$ ▷ GM
9: $<$ Approximate a $N/2 \times N/2$ log-polar grid $>$ ▷ GM+TM
10: (1) Initialize texture memory
11: (2) Log-polar kernel (interpolation)
12: **end for**
13: $<$ Compute phase correlation in the log-polar grids $>$ ▷ GM+SM
14: (1) Compute FFT in $\mathbf{I}_1$ and $\mathbf{I}_2$ log-polar grids
15: (2) Correlation kernel
16: (3) Compute IFFT
17: (4) Compute module
18: **end for**

▷ GM: Global Memory, SM: Shared Memory, TM: Texture Memory

Fig. 5. Detail of the pseudocode for the MLFFT, log-polar, and correlation stages in CUDA.

$$\mathbf{I} \cdot \mathbf{I}^T = \mathbf{USV}^T$$

using the cuSOLVER function *cusolverDnSgesvd* (see Section III-A, strategy 5) [38], where the diagonal elements of $\mathbf{S}$ are the singular values of $\mathbf{I} \cdot \mathbf{I}^T$ and the first columns of $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of $\mathbf{I} \cdot \mathbf{I}^T$ (Fig. 4, line 4). The final step is to compute the PCs using the cuBLAS function *cublasSgemm* (see Section III-A, strategy 5), that calculates the product between $\mathbf{I}$ and $\mathbf{V}$ (Fig. 4, line 5).

When the PCA computation is finished, each band of the images (PCA components from now on) is extended to an $N \times N$ size with N power of 2 adding a border as explained at the beginning of this section (Fig. 3, line 5).

2) Stage II and III. MLFFT and log-polar: The next stages, *MLFFT, log-polar,* and *correlation*, are illustrated in Fig. 5. It begins with the application of a high-pass filter to each PCA component (Fig. 5, lines 1-5) with the objective of allowing to pass signals with higher frequency and attenuate signals with lower frequencies. To get the best performance, the cuFFT library is used in the calculations of the FFT that are required by the high-pass filter (see Section III-A, strategy 5) [35].

Once the high-pass filter is applied, the process for computing the L-level MLFFT grid and the log-polar can begin (Fig. 5, lines 6-12). Fig. 6 illustrates this computation. First, four FRFTs with $\alpha = \{1, 1/4, 1/16, 1/64\}$ are computed which allows to analyze the image at different resolutions. The cuFFT library is used to accelerate the computations of the FFT that are required for the FRFT computations. Besides, the FRFT computation requires a high number of multiplications over data in global memory: first matrix multiplications and then element-wise multiplications. Consequently, this requires a high number of memory accesses. Secondly, using (8) the MLFFT grid is calculated.

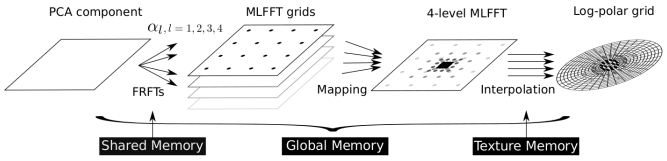


Fig. 6. Computation of a log-polar grid on GPU using a four-level MLFFT over a PCA component.

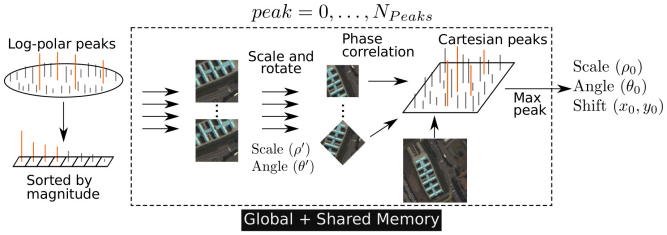


Fig. 7. Peak processing stage on GPU.

Lastly, to get the log-polar grid, the different levels of MLFFT are combined to integrate the information (Fig. 5, lines 9-11). To compute this process, it is necessary to perform a large number of memory accesses and interpolation calculations, for which the use of texture memory is crucial (see Section III-A, strategy 3).

3) Stage IV. Correlation: The last process to be executed in the first loop is the computation of phase correlation in the log-polar grids (Fig. 5, lines 13-17), and after it the result is added to a cumulative variable. The cuFFT library is used to compute the required FFTs in the phase correlation (see Section III-A, strategy 5).

4) Stage V. Composition: Returning to Fig. 3, when the *Correlation* stage is finished, the combination is calculated averaging log-polar maps after phase correlation (Fig. 3, line 7).

5) Stage VI. Peak processing: Fig. 7 illustrates the computation of this stage on GPU. First, in order to find the best peak, all the highest peaks in the log-polar map are processed (Fig. 3, lines 9-13). For this, this last stage starts sorting the log-polar peaks by magnitude using the *sort_by_key* function of the Thrust library (see Section III-A, strategy 5) (Fig. 3, line 8). This library automatically selects the most efficient implementation for sorting [39].

When the peaks are sorted, the higher ones are processed. In this processing the first component of the target image is rotated and scaled several times with the help of the *nppiRotate_32f_C1R* and *nppiResizeSqrPixel_32f_C1R* func-

TABLE I
INFORMATION FOR THE TEST HYPERSPECTRAL IMAGES: SENSOR, SIZE, NUMBER OF SPECTRAL BANDS, AND RESOLUTION (m/pixel)

Image	Sensor	Size	Bands	Spatial Resolution
Pavia University	ROSIS-03	610×340	103	1.3
Pavia Centre	ROSIS-03	1096×715	102	1.3
Indian Pines	AVIRIS	145×145	220	20
Salinas Valley	AVIRIS	512×217	204	3.7
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
Santa Barbara Front 2009	AVIRIS	900×470	224	16.4
Santa Barbara Front 2010	AVIRIS	900×470	224	11.3

tions of the NVIDIA Performance Primitives (NPP) library (see Section III-A, strategy 5) (Fig. 3, line 10).

Next, to determine the translation parameters and the correct values for the rotation angle and scale factor, a phase correlation is performed on the cartesian grid (Fig. 3, line 11). To recover these parameters it is necessary to find the maximum value in the phase correlation map (Fig. 3, line 12) using the *cublasIcamax* function of cuBLAS library (see Section III-A, strategy 5).

Finally, the highest peak of all phase correlations maps is selected and its coordinates determine the translation parameters. The coordinates of the selected peak in the log-polar map decide the scale factor and the rotation angle (Fig. 3, line 14). These four values are the output parameters that will be applied to the target image in order to register it.

IV. RESULTS

This section presents some experimental results obtained by the HYFM algorithm. First, the experimental conditions and test images will be described in Section IV-A. Then, the results in terms of registration and computation time are detailed (Sections IV-B and IV-C).

A. Experimental conditions and test images

The CPU version of the algorithm has been evaluated on a PC with a quad-core Intel Core i5-6600 at 3.3 GHz and 32 GB of RAM. The code has been compiled using the gcc 4.8.4 version under Linux. Regarding the GPU implementation, the CUDA code runs on a Pascal NVIDIA GeForce GTX 1070 with 15 SMs and 128 CUDA cores each. The CUDA code has been compiled under Linux using nvcc with version 8.0.26 of the toolkit, as well as the libraries used. All computations are performed in single-precision arithmetic. Performance results in terms of registration precision, computation time and speedup will be presented. In the computation times and speedup results, we provide the average of ten independent executions for each experiment.

Seven hyperspectral images were used to evaluate the algorithm proposed in this work. Table I shows detailed information for each image (the sensor, size, number of spectral bands, and spatial resolution).

TABLE II
SUCCESSFULLY REGISTERED CASES FOR EACH SCENE. THE NUMBER IN PARENTHESES INDICATES THE NUMBER OF SCALINGS THAT WERE CORRECTLY REGISTERED FOR ALL ANGLES

Scene	FMI-SPOMF	MLLFT	SURF
Pavia University	$1/4\times$ to $4.5\times$ (11)	$1/4\times$ to $4.5\times$ (11)	$1/4\times$ to $3.0\times$ (8)
Pavia Centre	$1/5\times$ to $6.0\times$ (15)	$1/5\times$ to $6.0\times$ (15)	$1/6\times$ to $5.5\times$ (15)
Indian Pines	$1/2\times$ to $3.0\times$ (7)	$1/2\times$ to $3.0\times$ (7)	$1.0\times$ (1)
Salinas	$1/2\times$ to $4.0\times$ (8)	$1/2\times$ to $4.0\times$ (8)	$1/2\times$ to $1.5\times$ (3)
Jasper Ridge	$1/3\times$ to $2.5\times$ (6)	$1/3\times$ to $2.5\times$ (6)	(0)
Santa Barbara Front	$1/4\times$ to $2.5\times$ (7)	$1/4\times$ to $2.5\times$ (7)	$1.0\times$ (1)
Santa Barbara Box	$1/2\times$ to $2.0\times$ (4)	$1/3\times$ to $2.0\times$ (5)	$1.0\times$ (1)
Average number of scalings	(8.29)	(8.43)	(4.14)

Scene	SIFT-1	SIFT-2	HYFM on GPU
Pavia University	$1/6\times$ to $5.5\times$ (15)	$1/2\times$ to $2.0\times$ (4)	$1/4\times$ to $5.5\times$ (13)
Pavia Centre	$1/8\times$ to $6.5\times$ (19)	$1/5\times$ to $2.5\times$ (8)	$1/5\times$ to $7.5\times$ (18)
Indian Pine	$1/3\times$ to $3.0\times$ (7)	$1/2\times$ to $1.0\times$ (2)	$1/2\times$ to $4.0\times$ (8)
Salinas	$1/4\times$ to $5.5\times$ (13)	(0)	$1/2\times$ to $4.5\times$ (9)
Jasper Ridge	(0)	$1/2\times$ to $1.5\times$ (3)	$1/5\times$ to $3.0\times$ (9)
Santa Barbara Front	$1.0\times$ to $1.5\times$ (2)	$1/3\times$ to $8.0\times$ (17)	$1/4\times$ to $3.5\times$ (9)
Santa Barbara Box	$1/3\times$ to $1.5\times$ (4)	$1/2\times$ to $1.0\times$ (2)	$1/4\times$ to $6.0\times$ (14)
Average number of scalings	(8.57)	(5.14)	(11.43)

The first four scenes were taken by the AVIRIS (Airbone Visible/Infrared Imaging Spectometrer) and ROSIS-03 (Reflective Optics System Imaging Spectrometer) sensors and they are commonly used for testing in remote sensing (Fig. 8) [40]. For these images the registration is performed with a rotated and scaled version of the same image. This way, all the details can be investigated in a controlled environment.

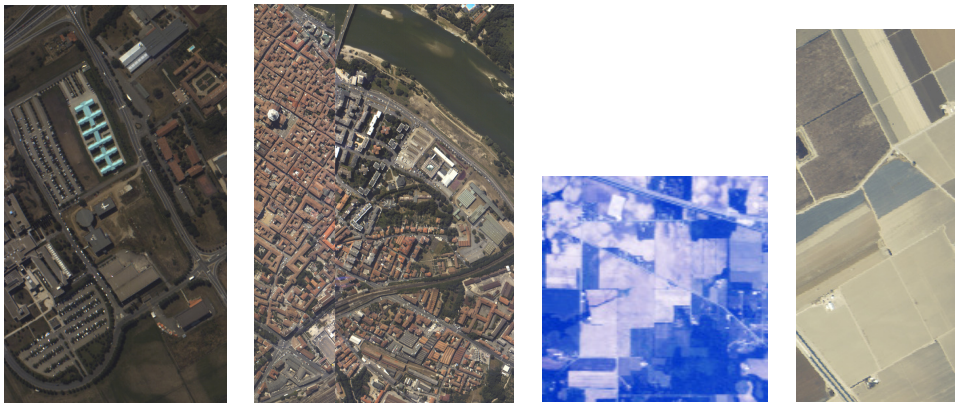
The remaining scenes are pairs of images taken by the AVIRIS sensor at different dates (Fig. 9). Each pair of images, in addition to different scaling and orientation factors, presents changes in vegetation, differences in the lighting conditions, and alterations in buildings and infrastructures. These images were loaded from the AVIRIS database [41] and no additional processing or band removal was performed. A region from both images was selected to remove the background. To increase the range of the study, we have performed these additional operations of scaling and rotation over the second real image of each pair.

B. Performance results in terms of registration precision

The algorithm has three parameters to be selected: the number of the PCA components to be retained ($N_{\mathrm{PCA}} = 8$ in our case), the parameters α in the FRFT and MLFFT ($\alpha = \{1, 1/4, 1/16, 1/64\}$), and the number of the log-polar peaks to be examined ($N_{\mathrm{Peaks}} = 50$ in our case). Cubic interpolation is used. The N_{PCA} and α parameters have been fixed after testing a wide range of values. A high number of log-polar peaks have been selected in order to maximize the probability of successful registration.

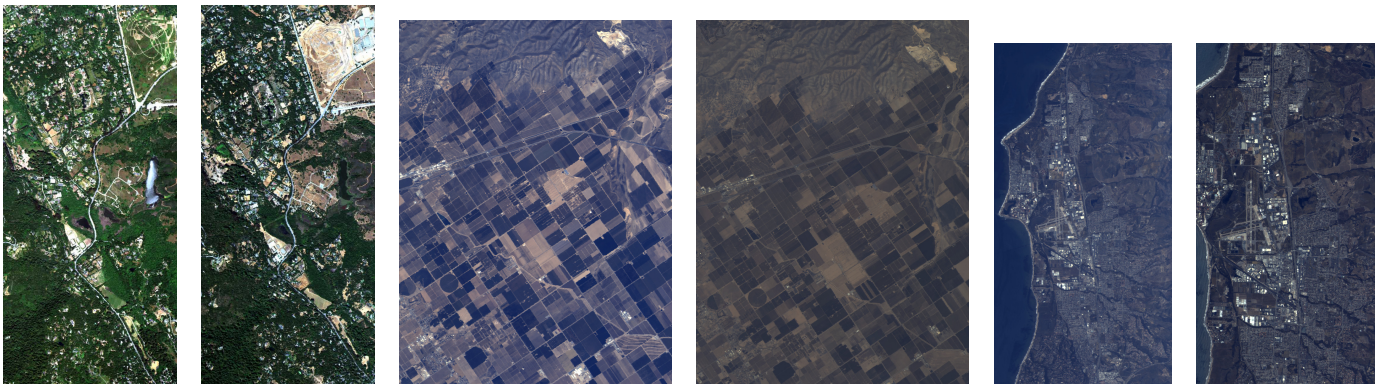
The procedure used to test the algorithm is as follows. When working with the first group of images, the original image is used as a reference while the target image is a rotated and scaled version of it. In the second group, the first image of each pair is used as a reference while the second image is used as a target image. We have performed an exhaustive search with scale ranging from $1/6\times$ to $8\times$ (20 scale factors) and rotation angles from 0 to 360 degrees in increments of 5 degrees (72 angles).

Table II summarizes the cases that were correctly registered for each scene by the proposed scheme on GPU and others in literature. These results were presented in detail



(a) (b) (c) (d)

Fig. 8. Hyperspectral images commonly used for testing in remote sensing: (a) *Pavia University*, (b) *Pavia Centre*, (c) *Indian Pines*, (d) *Salinas*.



(a) (b) (c) (d) (e) (f)

Fig. 9. *Jasper Ridge Biological Preserve 1* and *Santa Barbara* scenes taken for the AVIRIS sensor. (a) Fragment of the first *Jasper* image of size 588×1286 pixels taken on 5/12/2006, (b) fragment of the second *Jasper* image taken on 8/13/2007 with the same number of pixels, (c) fragment of the first *Santa Barbara Box Line 8* image of size 769×1024 pixels taken on 4/11/2013, (d) fragment of the second *Santa Barbara Box Line 8* image taken on 4/16/2014 with the same number of pixels, (e) fragment of the first *Santa Barbara Front Range A2* image of size 470×900 pixels taken on 3/30/2009, and (f) fragment of the second *Santa Barbara Front Range A2* image taken on 4/30/2010 with the same number of pixels.

in [28]. Two methods based on FFT: the original Fourier-Mellin algorithm (FMI-SPOMF) [11], and the MLFFT method [18], and three feature-based methods: the Speeded-Up Robust Features (SURF) method [42] and two Scale Invariant Feature Transform (SIFT) implementations, the original (SIFT-1) [43] and a more recent version (SIFT-2) [44].

The proposed HYFM method provides the best results, specially in images taken on different dates in which any other method fails. For example, for the *Santa Barbara Box* image, a scaling up to $6.0\times$ has been obtained, while the maximum scale recovered by the other methods is $2.0\times$. The accuracy of the method on GPU is the same as the accuracy of the original method on CPU. The GPU projection has been developed to preserve it.

C. Performance results in terms of computation times

The procedure test consists in registering two images. In the case of the pairs scenes taken by the AVIRIS sensor at different dates, these two images are registered. In the remaining images a target image is generated applying a rotation angle and a scale factor to the original image. All tables show execution times without times for CPU-GPU transfers and image allocation in memory, unless otherwise is specified.

TABLE III
CPU AND GTX 1070 GPU EXECUTION TIMES (IN SECONDS) PER STEP OF THE HYFM ALGORITHM FOR *Jasper Ridge* REGISTRATION

Step	Pseudocode lines	CPU	CUDA GPU	Speedup
Blackman	3 in Fig. 3	0.047s	**0.065**s	0.7×
PCA	4 in Fig. 3	0.790s	**0.214**s	3.7×
Extend	5 in Fig. 3	0.111s	**0.005**s	22.5×
High-pass	2-5 in Fig. 5	21.803s	**0.065**s	336.3×
MLFFT	7-8 in Fig. 5	344.637s	**0.953**s	361.6×
Log-polar	9-11 in Fig. 5	2.781s	**0.080**s	34.9×
Phase correlation	11 in Fig. 3 and 13-17 in Fig. 5	128.419s	**0.444**s	289.2×
Composition	7 in Fig. 3	0.007s	**0.001**s	6.0×
Sort peaks	8 in Fig. 3	0.023s	**0.004**s	5.2×
Scale and rotate	10 in Fig. 3	8.443s	**0.198**s	42.7×
Evaluate peaks	12 and 14 in Fig. 3	0.238s	**0.018**s	13.1×

The CPU and GPU utilizations by each step of the algorithm are presented in Table III for the registration of *Jasper Ridge* images. Most of the computation time is spent to calculate the MLFFT, the phase correlation and the high-pass filter. The 75% of the computation time is used for processing these steps because they involve expensive FFT calculations. The best speedup is obtained for these three processes. The use

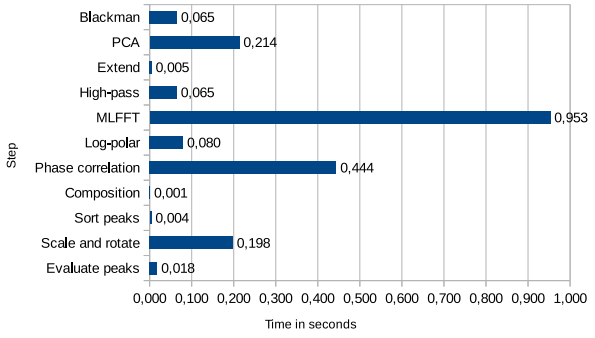


Fig. 10. GTX 1070 GPU utilization (in seconds) per step of the HYFM algorithm for *Jasper Ridge* registration

TABLE IV
CPU AND GTX 1070 GPU COMPUTATION TIMES (IN SECONDS) FOR HYFM FOR EACH SCENE

	CPU	CUDA GPU	Speedup
Pavia University	123.13s	**0.54s**	228.0×
Pavia Centre	507.01s	**1.88s**	269.7×
Indian Pines	6.57s	**0.19s**	34.6×
Salinas	28.62s	**0.27s**	106.0×
Jasper Ridge	507.30s	**2.05s**	247.5×
Santa Barbara Box	120.47s	**0.77s**	156.5×
Santa Barbara Front	120.09s	**0.67s**	179.2×

TABLE V
CPU AND GTX 1070 GPU EXECUTION TIMES (IN SECONDS) INCLUDING TIMES FOR CPU-GPU TRANSFERS AND IMAGE ALLOCATION IN MEMORY (IN CPU AND IN GPU) FOR HYFM FOR EACH SCENE

	CPU	CUDA GPU	Speedup
Pavia University	123.27s	**0.61s**	203.1×
Pavia Centre	507.59s	**2.11s**	240.6×
Indian Pines	6.59s	**0.21s**	31.4×
Salinas	28.79s	**0.34s**	83.8×
Jasper Ridge	508.71s	**2.53s**	201.1×
Santa Barbara Box	121.93s	**1.28s**	95.3×
Santa Barbara Front	120.87s	**0.94s**	128.1×

of the cuFFT library allows obtaining good performance in these steps. The scaling and rotation of the images is also expensive because it requires arithmetic computations and involves interpolation calculations. In addition, the MLFFT, the phase correlation, and the process to scale and rotate the images are executed many more times than the others due to the nature of the algorithm (see Fig. 3). Fig. 10 allows us to easily compare the execution times on GPU for each step.

Table IV compares the execution times of the HYFM algorithm for the different scenes on CPU and CUDA GPU. When the image size is large, better parallelization and speedups are achieved: speedups of $269.7\times$ and $247.5\times$ are obtained for the *Pavia Centre* and *Jasper Ridge* images, respectively. The similar computation times obtained for images of different sizes are associated with the resizing which is applied in the preprocessing stage (Fig. 3, line 5).

The execution times including times for CPU-GPU transfers and image allocation in memory are presented in Table V. As in Table IV, the speedup achieved is better when the image size is larger.

TABLE VI
ACHIEVED AND THEORETICAL AVERAGE OCCUPANCY IN PERCENTAGE BY EACH FUNCTION FOR *Jasper Ridge* REGISTRATION

	Achieved occupancy (%)	Theoretical occupancy (%)
Blackman	0.74	0.75
PCA	0.77	0.94
Extend	0.90	1.00
High-pass	0.76	0.86
MLFFT	0.63	0.73
Log-polar	0.89	0.99
Phase correlation	0.64	0.69
Composition	0.82	1.00
Sort peaks	0.37	0.42
Scale and rotate	0.64	0.74
Evaluate peaks	0.97	1.00

Table VI presents the achieved and theoretical average occupancy for each function, each of them consisting of more than one kernel. We have collected these measures using the NVIDIA Profiling Tool (NVPROF). It provides the metrics at kernel level. For this reason, the occupancies at function level are calculated by weighting each kernel occupancy by its execution time with respect to the total execution time of the function. Higher occupancy does not always correspond to higher performance, there is a point above which additional occupancy does not improve performance [34]. Low achieved occupancy for a function indicates that the GPU is not fully utilized because not enough thread blocks to hide the operation latency are launched. It is important to note that kernels of library functions cannot be configurable by the user and their occupancy is not always the best. Only two of the kernels not including library calls are limited. These are the kernel to apply the Blackman window and the kernel to initialize some constants required to calculate the log-polar grid. Both kernels are limited by number of registers due to the high use of mathematical functions (cosine, power, multiple divisions, etc.).

V. CONCLUSIONS

In this paper, a GPU accelerated algorithm for the registration of hyperspectral images based on the Fourier transform (HYFM) is presented. The algorithm was developed for running entirely in GPU and it exploits the GPU architecture efficiently. The algorithm is based on principal component analysis, multilayer fractional Fourier transform, combination of log-polar maps, and peak processing. The Fourier transform is computed through the FRFT over a multilayer grid, which reduces the interpolation errors in the log-polar map.

We have performed several experiments on hyperspectral images of different sizes and obtained by different sensors, achieving successful results in both precision of the registration and execution time compared to other algorithms available in the literature. The steps that involve FFT computations are the most expensive in execution time and the most optimized in GPU: MLFFT, phase correlation and high-pass filter. For all of them the highest speedups have been reached. A registration time of 2.11s is achieved in GPU for the *Pavia Centre* image with 1096×715 pixels and 102 bands, obtaining a speeding factor of $240.6\times$ when compared to the sequential execution

in CPU. The results have shown that commodity GPUs are an adequate platform to perform efficient registration even for large images.

SUPPLEMENTAL DATA

The underlying research materials for this article can be accessed at http://wiki.citius.usc.es/hiperespectral:hyfm-gpu.

ACKNOWLEDGMENT

This work was supported in part by the Consellería de Cultura, Educación e Ordenación Universitaria [grant numbers GRC2014/008 and ED431G/08] and Ministry of Education, Culture and Sport, Government of Spain [grant numbers TIN2013-41129-P and TIN2016-76373-P] both are co-funded by the European Regional Development Fund (ERDF). The work of Álvaro Ordóñez was also supported in part by the Ministry of Education, Culture and Sport, Government of Spain, under a FPU Grant [grant number FPU16/03537].

REFERENCES

[1] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. M. Nasrabadi, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *Geoscience and Remote Sensing Magazine, IEEE*, vol. 1, no. 2, pp. 6–36, 2013.

[2] D. Haboudane, J. R. Miller, E. Pattey, P. J. Zarco-Tejada, and I. B. Strachan, "Hyperspectral vegetation indices and novel algorithms for predicting green LAI of crop canopies: Modeling and validation in the context of precision agriculture," *Remote sensing of environment*, vol. 90, no. 3, pp. 337–352, 2004.

[3] D. B. Malkoff and W. R. Oliver, "Hyperspectral imaging applied to forensic medicine," in *BiOS 2000 The International Symposium on Biomedical Optics*. International Society for Optics and Photonics, 2000, pp. 108–116.

[4] D. Manolakis, D. Marden, and G. A. Shaw, "Hyperspectral image processing for automatic target detection applications," *Lincoln Laboratory Journal*, vol. 14, no. 1, pp. 79–116, 2003.

[5] D. L. Hill, P. G. Batchelor, M. Holden, and D. J. Hawkes, "Medical image registration," *Physics in medicine and biology*, vol. 46, no. 3, p. R1, 2001.

[6] X. Dai and S. Khorram, "The effects of image misregistration on the accuracy of remotely sensed change detection," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 36, no. 5, pp. 1566–1577, 1998.

[7] L. G. Brown, "A survey of image registration techniques," *ACM computing surveys (CSUR)*, vol. 24, no. 4, pp. 325–376, 1992.

[8] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[9] S. Dawn, V. Saxena, and B. Sharma, "Remote sensing image registration techniques: A survey," in *Image and Signal Processing*. Springer, 2010, pp. 103–112.

[10] J. Le Moigne, N. S. Netanyahu, and R. D. Eastman, *Image registration for remote sensing*. Cambridge University Press, 2011.

[11] Q.-S. Chen, M. Defrise, and F. Deconinck, "Symmetric phase-only matched filtering of Fourier-Mellin transforms for image registration and recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 12, pp. 1156–1168, Dec 1994.

[12] B. S. Reddy and B. N. Chatterji, "An FFT-based technique for translation, rotation, and scale-invariant image registration," *IEEE transactions on image processing*, vol. 5, no. 8, pp. 1266–1271, 1996.

[13] S. Derrode and F. Ghorbel, "Robust and efficient Fourier–Mellin transform approximations for gray-level image reconstruction and complete invariant description," *Computer Vision and Image Understanding*, vol. 83, no. 1, pp. 57–78, 2001.

[14] J. Bigot, F. Gamboa, and M. Vimond, "Estimation of translation, rotation, and scaling between noisy images using the Fourier-Mellin transform," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 614–645, 2009.

[15] Y. Keller, A. Averbuch, and M. Israeli, "Pseudopolar-based estimation of large translations, rotations, and scalings in images," *Image Processing, IEEE Transactions on*, vol. 14, no. 1, pp. 12–22, 2005.

[16] H. Liu, B. Guo, and Z. Feng, "Pseudo-log-polar Fourier transform for image registration," *Signal Processing Letters, IEEE*, vol. 13, no. 1, pp. 17–20, 2006.

[17] X.-x. Wu, B.-l. Guo, and J. Wang, "Octa-log-polar Fourier transform for image registration," in *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on*, vol. 1. IEEE, 2009, pp. 601–604.

[18] W. Pan, K. Qin, and Y. Chen, "An adaptable-multilayer fractional Fourier transform approach for image registration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 400–414, March 2009.

[19] J. López-Fandiño, P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity GPUs," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 8, no. 6, pp. 2884–2893, 2015.

[20] A. Plaza, J. Plaza, and H. Vegas, "Improving the performance of hyperspectral image and signal processing algorithms using parallel, distributed and specialized hardware-based systems," *Journal of Signal Processing Systems*, vol. 61, no. 3, pp. 293–315, 2010.

[21] Y. Tarabalka, T. V. Haavardsholm, I. Kåsen, and T. Skauli, "Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and GPU processing," *Journal of Real-Time Image Processing*, vol. 4, no. 3, pp. 287–300, 2009.

[22] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Efficient segmentation of hyperspectral images on commodity GPUs." in *KES*, vol. 243, 2012, pp. 2130–2139.

[23] P. Muyan-Ozcelik, J. D. Owens, J. Xia, and S. S. Samant, "Fast deformable registration on the GPU: A CUDA implementation of demons," in *Computational Sciences and Its Applications, 2008. ICCSA'08. International Conference on*. IEEE, 2008, pp. 223–233.

[24] Z. Fan, C. Vetter, C. Guetter, D. Yu, R. Westermann, A. Kaufman, and C. Xu, "Optimized gpu implementation of learning-based non-rigid multi-modal registration," in *Medical Imaging*. International Society for Optics and Photonics, 2008, pp. 69 142Y–69 142Y.

[25] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley, "Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images," *Computer methods and programs in biomedicine*, vol. 99, no. 2, pp. 133–146, 2010.

[26] S. Sah, J. Vanek, Y. Roh, and R. Wasnik, "GPU accelerated real time rotation, scale and translation invariant image registration method," in *Image Analysis and Recognition*. Springer, 2012, pp. 224–233.

[27] Y. Zhang, P. Zhou, Y. Ren, and Z. Zou, "GPU-accelerated large-size VHR images registration via coarse-to-fine matching," *Computers & Geosciences*, vol. 66, pp. 54–65, 2014.

[28] Álvaro Ordóñez, F. Argüello, and D. B. Heras, "Fourier–mellin registration of two hyperspectral images," *International Journal of Remote Sensing*, vol. 38, no. 11, pp. 3253–3273, 2017.

[29] J. A. Richards, *Remote Sensing Digital Image Analysis: An Introduction*, 5th ed. Springer Publishing Company, Incorporated, 2012.

[30] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[31] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[32] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Addison-Wesley Professional, 2010.

[33] NVIDIA. (2016) Whitepaper: NVIDIA Tesla P100. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[34] ——. (2017) CUDA C Best Practices Guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf

[35] ——. (2017) cuFFT Library User's Guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUFFT_Library.pdf

[36] A. S. Garea, D. B. Heras, and F. Argüello, "GPU classification of remote-sensing images using kernel ELM and extended morphological profiles," *International Journal of Remote Sensing*, vol. 37, no. 24, pp. 5918–5935, 2016.

[37] NVIDIA. (2017) cuBLAS Library User's Guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf

[38] ——. (2017) cuSOLVER Library User's Guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUSOLVER_Library.pdf

[39] ——. (2017) Thrust Quick Start Guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/Thrust_Quick_Start_Guide.pdf

[40] Hyperspectral Remote Sensing Scenes at the University of the Basque Country. [Online]. Available: http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes

[41] AVIRIS database. [Online]. Available: http://aviris.jpl.nasa.gov/data/index.html
[42] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.
[43] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
[44] I. R. Otero, "Anatomy of the SIFT method," Ph.D. dissertation, École normale supérieure de Cachan-ENS Cachan, 2015.

Álvaro Ordóñez received the B.S. in Computer Science and the M.S. in Big Data Technologies from the University of Santiago de Compostela, Santiago de Compostela, Spain, in 2015 and 2016, respectively, where he is currently working toward the Ph.D. degree as an assistant researcher of Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS). His main research interests include image analysis and processing, parallel algorithms and big data technologies.

Francisco Argüello received the B.S. and Ph.D. degrees in Physics from the University of Santiago, Spain in 1988 and 1992, respectively. He is currently an associate professor in the Department of Electronic and Computer Engineering at the University of Santiago, Spain. His current research interests include signal and image processing, computer graphics, parallel and distributed computing, and quantum computing.

Dora B. Heras received a M.S. degree in Physics in 1994 and a Ph.D. in 2000 from the University of Santiago. She is currently an Associate Professor in the Department of Electronics and Computer Engineering at the same University. Her research interests include parallel and distributed computing, software optimization techniques for emerging architectures, computer graphics and image processing.