Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs

L^2-SIFT: SIFT feature extraction and matching for large images in large-scale aerial photogrammetry

Yanbiao Sun [a,b], Liang Zhao [b], Shoudong Huang [b], Lei Yan [a,*], Gamini Dissanayake [b]

[a] Institute of Remote Sensing and GIS, School of Earth and Space Science, Peking University, Beijing 100871, China
[b] Centre for Autonomous Systems, Faculty of Engineering and Information Technology, University of Technology, Sydney, NSW 2007, Australia

ARTICLE INFO

Article history:
Received 17 July 2013
Received in revised form 30 January 2014
Accepted 1 February 2014
Available online 3 March 2014

Keywords:
Aerial photogrammetry
Large-scale
SIFT
Feature extraction and matching
Bundle adjustment

ABSTRACT

The primary contribution of this paper is an efficient feature extraction and matching implementation for large images in large-scale aerial photogrammetry experiments. First, a Block-SIFT method is designed to overcome the memory limitation of SIFT for extracting and matching features from large photogrammetric images. For each pair of images, the original large image is split into blocks and the possible corresponding blocks in the other image are determined by pre-estimating the relative transformation between the two images. Because of the reduced memory requirement, features can be extracted and matched from the original images without down-sampling. Next, a red-black tree data structure is applied to create a feature relationship to reduce the search complexity when matching tie points. Meanwhile, tree key exchange and segment matching methods are proposed to match the tie points along-track and across-track. Finally, to evaluate the accuracy of the features extracted and matched from the proposed L^2-SIFT algorithm, a bundle adjustment with parallax angle feature parametrization (ParallaxBA[1]) is applied to obtain the Mean Square Error (MSE) of the feature reprojections, where the feature extraction and matching result is the only information used in the nonlinear optimisation system. Seven different experimental aerial photogrammetric datasets are used to demonstrate the efficiency and validity of the proposed algorithm. It is demonstrated that more than 33 million features can be extracted and matched from the Taian dataset with 737 images within 21 h using the L^2-SIFT algorithm. In addition, the ParallaxBA involving more than 2.7 million features and 6 million image points can easily converge to an MSE of 0.03874. The C/C++ source code for the proposed algorithm is available at http://services.eng.uts.edu.au/sdhuang/research.htm.

1. Introduction

Feature extraction and matching plays a crucial role in the field of photogrammetry, in which accurate and reliable features are often provided as original input information for estimating camera orientation, 3D scene reconstruction, digital surface model generation and, especially, for bundle adjustment (BA), which minimises the reprojection error of 2D image points to produce optimal 3D structures and estimate camera parameters simultaneously. As a basic procedure for photogrammetry, feature extraction and matching has attracted considerable attention in recent decades.

In the literature, many approaches have been proposed to extract and match features from images. The earliest feature extraction approach can date back to the work of Moravec (1981). The Moravec detector was improved, resulting in the Harris detector (Harris and Stephens, 1988). The Harris detector took the Gaussian window function instead of the binary window function and set different weights according to the distances between the central pixel and the surrounding pixels, aiming to decrease the noise effect. It used a Taylor series to approximate more directions and adopted new criteria to identify candidate features. Although the Harris detector outperformed the Moravec detector, both detectors were variant to transformation, scale and illumination changes. To extract features from images taken from different viewpoints and under different illumination conditions, the Scale Invariant Feature Transform (SIFT) detector was first proposed by Lowe (2004). It

* Corresponding author. Tel.: +86 13910821927.
E-mail addresses: syb51@pku.edu.cn (Y. Sun), Liang.Zhao-1@uts.edu.au (L. Zhao), Shoudong.Huang@uts.edu.au (S. Huang), lyan@pku.edu.cn (L. Yan), Gamini.Dissanayake@uts.edu.au (G. Dissanayake).

[1] The ParallaxBA source code is available open-source at OpenSLAM http://openslam.org/ParallaxBA.html. The features of the test datasets in this paper as the input to ParallaxBA are also available at OpenSLAM.

http://dx.doi.org/10.1016/j.isprsjprs.2014.02.001
0924-2716/© 2014 International Society for Photogrammetry and Remote Sensing, Inc. (ISPRS) Published by Elsevier B.V. All rights reserved.

exploited the scale space theory that was first proposed by Lindeberg (1998) to produce scale-invariant features. Furthermore, SIFT used a Difference-of-Gaussian (DoG) instead of scale-normalised Laplacian of Gaussian (LoG) because DoG is much faster. Moreover, the local gradient orientation and magnitudes were computed so that reliable features may be extracted even when there are rotation and translation between two images. In the matching procedure, a 128-dimension descriptor was utilised to match reliable features. However, SIFT most likely failed to extract and match reliable correspondences of images captured from very different viewpoints. To overcome this drawback, two improved SIFT algorithms have been proposed, including the Gradient location-orientation histogram (GLOH) and Affine-SIFT (ASIFT) (Zhang et al., 2012). GLOH was an extension of the SIFT descriptor and computed the SIFT descriptor for a log-polar location grid with 3 bins in a radial direction and 8 bins in an angular direction instead of a 4×4 chessboard location bin. In addition, its gradient orientations were quantised in 16-bin and 272-bin histograms, and a principal component analysis (PCA) was applied to reduce the descriptor dimension (Mikolajczyk and Schmid, 2005). ASIFT is a fully affine invariant method that includes six parameters of affine transform, whereas SIFT is only invariant to four of the six parameters of affine transform. It simulated the image views obtained by the longitude and the latitude angles and then applied the other parameters (Morel and Yu, 2009).

High computational and memory storage costs are required by SIFT; thus, it is impossible to use the primal SIFT algorithm in some applications such as real-time tracking with a sequence of images. To decrease memory usage and improve its efficiency, many approaches have been proposed in the last few years. The SIFT++ software package, which is a lightweight C++ implementation of SIFT, was published in (Vedaldi, 2010). SIFT++ requires less memory compared with other SIFT implementations, so it can process large images if the parameter "first octave" is set as 0 rather than -1. However, for very large images such as aerial images and satellite images in the photogrammetry field, the software needs other operations such as down-sampling or tiling to process these images. LDAHash reduced the dimensionality by mapping the descriptor vectors into the Hamming space and represented descriptors with short binary strings whose similarity can be measured by the Hamming distance, so it could speed up matching and reduce memory consumption (Strecha et al., 2012). PCA-SIFT adopted the dimensionality reduction strategy to express feature patches and projected high-dimensional samples into low-dimensional space (Ke and Sukthankar, 2004). It utilised PCA to normalise gradient patches instead of histograms, significantly decreasing the dimension of feature vectors and improving matching efficiency. Speeded-Up Robust Features (SURF) was a fast scale and rotation invariant interest point detector and descriptor (Bay et al., 2008). It had less runtime than SIFT because it used integral images, which drastically reduced the number of operations for simple box convolutions, independent of the chosen scale. These algorithms can only slightly decrease the computational cost; fortunately, the SIFT implementation designed on a graphic process unit (GPU) can achieve considerable computational savings. Among them, the SiftGPU (Wu, 2007) open-source software package has been released, which is used in the implementation of the L^2-SIFT algorithm proposed in this paper.

With the development of camera hardware, image resolution is becoming higher and higher. Nevertheless, with the limitation of memory storage, the SiftGPU algorithm cannot directly provide a solution to feature extraction and matching using large images taken by professional metric cameras in aerial photogrammetry. It is impossible to allocate enough memory when the processed images are very large because the software needs to build a series of pyramid images to create Gaussian and DoG scale space. SiftGPU reduces the size of large images with down-sampling, but it will give rise to a loss of detail in the images and decrease the number of extracted features. In 2011, a recursive tiling method was proposed for feature matching for close-range images to extract more features and improve matching correctness; this method can be used for dealing with large images (Novák et al., 2011). To diminish the perspective distortion and reduce the number of similar points from repetitive textures, this method recursively splits a tile into four smaller tiles so that homography becomes a better local approximation of the perspective distortion. The main advantage of this algorithm is that it can increase an average of 2–3 times more matches compared to conventional wide-baseline matching; at the same time, the main drawback is the high computational cost, as it usually takes about three times longer to measure points than in the standard case. In this paper, 2D transformation estimations have shown to be sufficient to create aerial image relationships in aerial photogrammetry. Both the number of matches and the computational cost are considered; we only adopt a 2D rigid transformation, which is a special case of homography, and a single recursion level to create image transformation compared with the method used by Novák et al. (2011). In this paper, we design a Block-SIFT method to split two large images into blocks and extract features from the corresponding blocks using SiftGPU to decrease the memory consumption and thus speed up the feature extracting and matching process. The first "L" in L^2-SIFT represents "large images".

In this paper, correspondence refers to a pair of corresponding image points, and tie points amount to a point set that includes two or more image points that correspond to a same 3D point in the environment. Tie points provide the geometric information to determine the camera poses, relative scales and 3D structure. Thus, tie points must be matched from two or more images according to the along-track and across-track distribution of images. An important aspect in digital aerial triangulation is the need to match several images simultaneously. In the earlier years, tie points must be marked on every photograph by the stereovision ability of a human operator (Schenk, 1997). In 2010, Barazzetti et al. proposed automatic tie point extraction (ATiPE) for unordered close-range images (Barazzetti et al., 2010). It divided the all n images into $n-2$ triplets T_i where $T_i = \{I_i, I_{i+1}, I_{i+2}\}$ and I_i represented the ith image. For the single triplet T_i matching, after finishing a pair-wise matching between images $C_i = \{I_i, I_{i+1}\}$ and $C'_i = \{I_{i+1}, I_{i+2}\}$, correspondences from image I_i and I_{i+2} can be achieved by directly comparing the same image points appearing on image I_{i+1}. For multiple triplet matching such as other triplet $T_{i+1} = \{I_{i+1}, I_{i+2}, I_{i+3}\}$, their tie points can be transferred with a simple comparison based on the value of the image coordinates. Because the number of images is also large for large-scale photogrammetry experiments and the number of features that are extracted and matched from a pair of corresponding large aerial images by Block-SIFT method is enormous as well, it is very hard to match all the tie points rapidly. In this paper, a red-black data structure is applied to create a features index to reduce the search complexity, and a tree key exchange method and segment matching method are presented to match tie points along-track and across-track. The original structure of the red-black tree, which is called the symmetric binary B-tree, was proposed in 1972 (Bayer, 1972); it was named a red-black tree in 1978 (Guibas and Sedgewick, 1978). In many fields, red-black trees are used for different purposes. In 2004, Fink et al. used a red-black tree as an indexing structure to search for approximate matches in a large medical database (Fink et al., 2004). In 2012, Crisp and Tao sorted merge costs using a red-black tree to rapidly find the cheapest merge cost in fast region merging for image segmentation (Crisp and Tao, 2010). The proposed algorithm in this paper optimises memory usage to ensure that a computer can provide enough memory to match a very large number of image points into tie points in large-scale photogrammetry experiments. The second "L" in L^2-SIFT represents "large-scale photogrammetry experiments". The overall flowchart of L^2-SIFT is shown in Fig. 1.

This paper is organised as follows. Section 2 briefly introduces the SIFT feature extraction and matching algorithm and then SiftGPU implementation. Section 3 states the Block-SIFT method for a pair of corresponding images taken along-track or across-track. Section 4 details tie point matching using a red-black tree structure, tree key exchange method and segment matching method. In Section 5, seven sets of aerial image tests are used to evaluate the proposed algorithm, and their detailed runtime and convergence of bundle adjustment are listed. Finally, Section 6 presents the paper's conclusions and addresses possible further studies.

2. SIFT feature extraction and matching

The SIFT algorithm can extract stable features, which are invariant to scaling, rotation, illumination and affine transformation with sub-pixel accuracy, and match them based on the 128-dimension descriptors. Therefore, SIFT is an ideal feature extraction and matching method for photogrammetry. To date, the algorithm has been widely used in robotics, image-based computer graphics, digital photogrammetry, surveying and geodesy, etc.

To extract stable features from images with different sizes, the SIFT algorithm builds pyramid images and uses a staged filtering approach based on the DoG function. Pyramid image space consists of two parts (see Fig. 2). The first part is Gaussian scale space, in which each octave is computed by convolution between a down-sampled image with different Gaussian kernels, such as $L(x,y,\sigma)$ in Eq. (1) where $G(x,y,\sigma)$ is a variable-scale Gaussian and $I(x,y)$ is an input image (Lowe, 2004). The convolution method with Gaussian kernels can reduce the noise effect and improve the invariance of the features when the scales of the images are changed. The second part is DoG scale space, in which DoG is a close approximation to LoG in Eq. (2). Compared with LoG, DoG has lower computational costs and approximately the same function to extract stable features.

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{1}$$

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma); \end{aligned} \tag{2}$$

where k is a constant multiplicative factor.

Feature extraction in SIFT includes pixel and sub-pixel localisations of a keypoint. A candidate pixel keypoint is chosen when it is an extreme pixel compared with 26 neighbouring pixels in DoG scale space. The sub-pixel keypoint localisation is implemented by a quadratic approximation, which is computed using a second-order Taylor expansion. Additionally, to improve stability, keypoints with low contrast are removed.

Matching the SIFT features extracted above can be achieved by comparing two sets of feature descriptors with the nearest neighbour method. The correspondences can be found when the ratio between the shortest distance and the second shortest distance is smaller than a given threshold. Considering the effect of image scale, rotation and affine distortion, the SIFT algorithm uses 128-dimension descriptors to conduct feature matching. The SIFT feature descriptors represent image gradient magnitudes and orientations in a 4 × 4 sub-region around the candidate feature. The descriptors can describe both the shape and the orientation of a feature; thus, features with different rotations between two images can be easily matched.

To obtain reliable and accurate corresponding features, it is necessary to use pyramid images and 128-dimension descriptors to conduct SIFT extraction and matching. Unfortunately, this strategy needs high computational power and memory consumption, which limit the use of SIFT in real-time implementations or aerial photogrammetry with large images. The SiftGPU has three implementations including GLSL Unpacked/Packed and CUDA; the five steps of the primal SIFT can run on GPU in a parallel way. It is shown that the performance of SiftGPU is nearly real-time for webcam-sized images, and the quality of the features extracted by SiftGPU is approximately the same as that extracted by the primal SIFT. However, because the GPU memory is limited, large images need to be down-sampled to reduce the size before applying SIFT.

Due to the limitations of GPU and CPU memory, SiftGPU and most of the SIFT implementations cannot address large images directly without other operations such as down-sampling or tiling. In the next section, a Block-SIFT method based on SiftGPU is proposed to extract and match features from large aerial photogrammetric images more efficiently.

3. Block-SIFT features extraction and matching for one pair of images

The SIFT feature extraction and matching algorithm is originally designed to process images in computer vision, in which images are taken by general cameras with low resolution (e.g., 1024 × 768 pixels). However, in aerial photogrammetry, to efficiently and economically reconstruct large-scale terrain, the images are always acquired by professional metric cameras with large frames (e.g., DMC camera, 7680 × 13824 pixels). Applying SIFT to these large images directly will bring about "out of

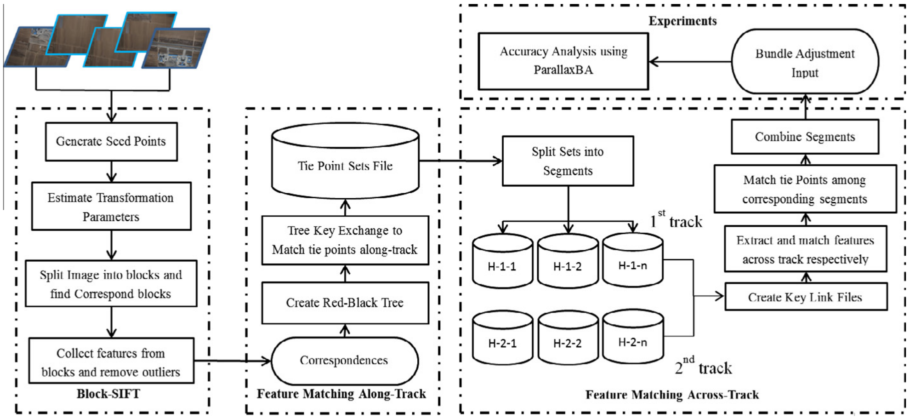

Experiments
Accuracy Analysis using ParallaxBA
Bundle Adjustment Input
Generate Seed Points
Estimate Transformation Parameters
Split Image into blocks and find Correspond blocks
Collect features from blocks and remove outliers
Block-SIFT
Tie Point Sets File
Tree Key Exchange to Match tie points along-track
Create Red-Black Tree
Correspondences
Feature Matching Along-Track
Split Sets into Segments
H-1-1
H-1-2
H-1-n
1st track
H-2-1
H-2-2
H-2-n
2nd track
Combine Segments
Match tie Points among corresponding segments
Extract and match features across track respectively
Create Key Link Files
Feature Matching Across-Track

Fig. 1. The overall flowchart of L^2-SIFT.

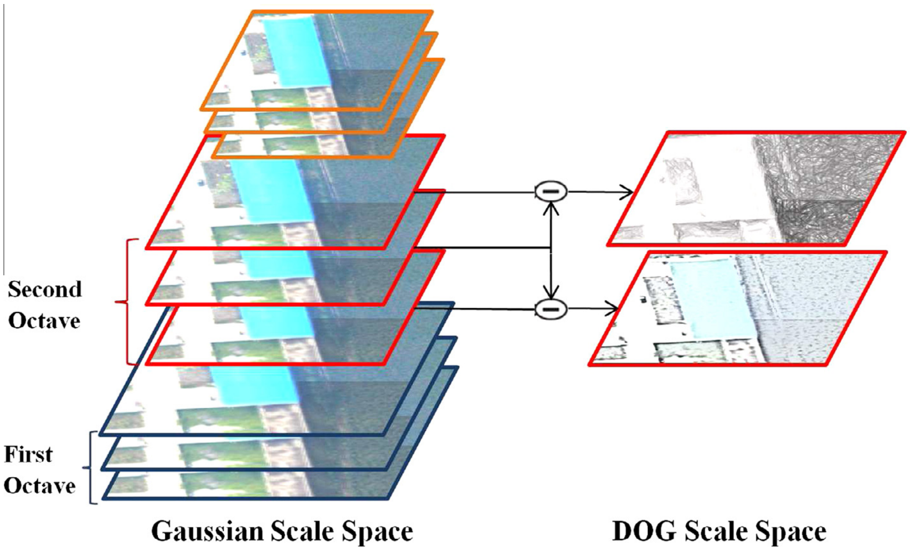


Fig. 2. SIFT scale space.

memory" (Wu, 2007), so some implementations such as SiftGPU first down-sample the original images such that they can be processed within the memory capacity of the CPU or GPU. However, this will cause information loss, and fewer features can be extracted from the images, which is inappropriate in aerial photogrammetry where high precision in both feature extraction and 3D locations is required. To extract and match features from photogrammetric images without losing any information, the Block-SIFT method is proposed in this paper. The main idea is to split the original images into blocks and extract and match features from the corresponding blocks using SiftGPU. The strategy contains three major steps. First, a few seed points are generated to estimate the relative transformation parameters between the two original images; second, the first original image is split into blocks, and the corresponding blocks in the other image are found based on the relative transformation parameters; finally, we collect all the features that are extracted and matched by SiftGPU from the different block pairs (see Fig. 1).

3.1. Generate seed points

To find the corresponding block in the other image and extract and match features by the Block-SIFT method, the geometric relative relationship between two images needs to be computed first. In this paper, we assume that extra information such as GPS/IMU is not available and that finding the corresponding block is based on 2D transformation parameters of two images computed by seed points.

Although down-sampling will suffer a loss of the detail of images and thus is not suggested for extracting and matching features with high quality, it can still be used to generate seed points that refer to a small amount of correspondences. In this paper, three or more pairs of seed points are sufficient to obtain an approximate estimation of the transformation parameters. To generate seed points, SIFT is used on the down-sampled images to obtain features, and Random Sample Consensus (RANSAC) (Fischler and Bolles, 1981) is used to remove the outliers. The resulting matched features are used as the seed points to estimate the transformation parameters by using the linear SVD algorithm described in Section 3.2.

When estimating transformation parameters, the accuracy of seed points is more important than the number of the seed points. Although few inliers are retained after RANSAC with a small threshold, the estimation of transformation parameters is still accurate enough to find the corresponding blocks, as shown in the results section of this paper.

3.2. Estimate transformation parameters

In traditional aerial photogrammetry, the aircraft platform used in photogrammetry always attempts to impose zero pitch and roll angles; thus, the transformation between two adjacent corresponding aerial images can be approximated as a simple 2D rigid transformation in this paper. The transformation parameters including rotation matrix R, translation t and scale c can be estimated by minimising the mean squared error using the seed points described in Section 3.1, which can be implemented according to the linear SVD method (Umeyama, 1991).

Because the images are scaled by down-sampling, the translation t and scale c computed by using seed points are different from the real translation and scale parameter between the original corresponding images. The real translation t_L and scale c_L can be obtained by using the down-sampling scale λ as

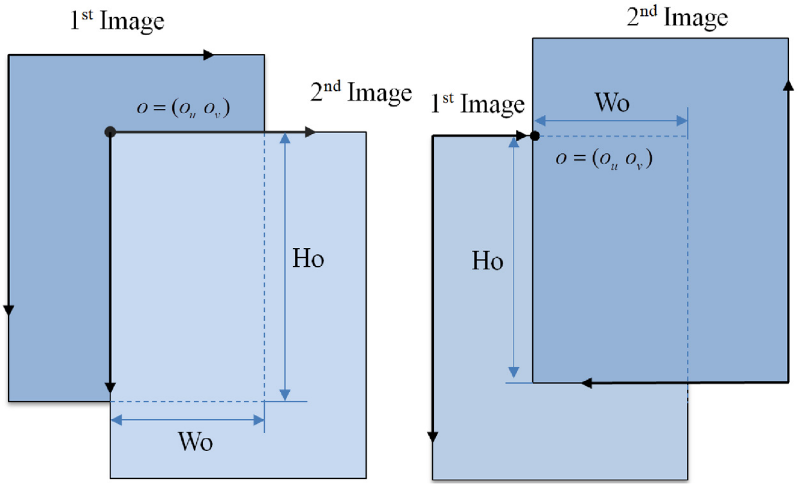
$$\begin{cases} t_L &= \lambda t \\ c_L &= \lambda c \end{cases} \tag{3}$$

Because the rotation matrix does not have scale issues, the rotation parameters between the original images can be represented as $R_L = R$.

3.3. Split large image into blocks

3.3.1. Estimate the overlaps

Because the overlap area between two adjacent images accounts for some portion of the whole large image, we only split the overlap part rather than the whole image into blocks to improve the algorithms efficiency. The overlap region, which is described by the size and the origin of the overlap area on the first image, must first be estimated. In this paper, the overlap region is described using three parameters, H_o, W_o and o, where H_o and W_o refer to the width and height of the overlap area, respectively, and $o = (o_u, o_v)$ represents the origin of the overlap area in the first image, as shown in Fig. 3.



(a) Overlap region along-track. (b) Overlap region across-track.

Fig. 3. Overlap region between two images.

The overlap region is determined by the flight direction and camera orientation as shown in Fig. 3, where Fig. 3(a) shows the overlap region between two along-track corresponding images, while Fig. 3(b) describes the overlap region between two across-track corresponding images. The bold arrow lines represent the axis of the image coordinates, and the bold point is the origin of the overlap region in the first image.

Suppose the width and height of the original image are w and h, respectively; the centre μ_{P2} of the four corners of the second image can be computed by Eq. (4)

$$\begin{cases} \mu_{P2u} = w/2 \\ \mu_{P2v} = h/2 \end{cases} \tag{4}$$

where μ_{P2u}, μ_{P2v} are the two-dimensional image coordinates of μ_{P2}. Then, the corresponding points μ'_{P2} in the first image can be computed by using the estimated transformation parameters as follows

$$\mu'_{P2} = (1/c_L)R_L^{-1}(\mu_{P2} - t_L) \tag{5}$$

Meanwhile, the centre of the first image is $\mu_{P1} = (w/2, h/2)$. Thus it is clear that the overlap region is not empty only when $|\mu'_{P2u} - \mu_{P1u}| < w$ and $|\mu'_{P2v} - \mu_{P1v}| < h$ hold simultaneously, where μ'_{P2u}, μ'_{P2v} are the u, v components of the image coordinate μ'_{P2}, and μ_{P1u}, μ_{P1v} are those of μ_{P1}.

When the overlap region is not empty, the origin of the overlap in the first image can be computed by

$$o_u = \begin{cases} \mu'_{P2u} - \mu_{P1u}, & \mu'_{P2u} > \mu_{P1u} \\ 0, & \mu'_{P2u} \leqslant \mu_{P1u} \end{cases} \tag{6}$$

$$o_v = \begin{cases} \mu'_{P2v} - \mu_{P1v}, & \mu'_{P2v} > \mu_{P1v} \\ 0, & \mu'_{P2v} \leqslant \mu_{P1v} \end{cases} \tag{7}$$

and the size of the overlap area can be calculated as

$$\begin{cases} W_o = w - |\mu'_{P2u} - \mu_{P1u}| \\ H_o = h - |\mu'_{P2v} - \mu_{P1v}| \end{cases} \tag{8}$$

3.3.2. Split image and find corresponding block

After the overlap region is computed, we can split the overlap area in the first large image into blocks and find the corresponding blocks in the second image easily by using the transformation parameters computed in Section 3.2. Suppose the width and height of one block image is represented as W_s and H_s, respectively, as shown in Fig. 4. For a specified block in the first image with the index of the column and row as (i, j) (starting from 0), the origin of this block is computed as

$$o_i = \{o_u + W_s \times i, o_v + H_s \times j\} \tag{9}$$

Then the four corners can be easily determined by the width W_s and height H_s in the first image. By using transformation parameters, the four corners are transformed into the second image and the corresponding block can be found.

Because only simple two-dimensional rigid transformations are considered, the estimated block in the second image is inaccurate. To match more features, the size of the corresponding block in the second image should be expanded when finding the corresponding block in Fig. 4. How large the block size and expanded size should be will be discussed in Section 5.2.

3.4. Remove outliers

After all the blocks from original large images are processed using SiftGPU, the outliers must be removed. In this paper, RANSAC (Fischler and Bolles, 1981) is used for outlier removal. Because RANSAC is based on the geometric relationship between the two corresponding images with a pinhole camera model, all of the features extracted and matched in different blocks with respect to the original images must be transformed into the original image coordinates before applying RANSAC. This transformation can be achieved by adding the global coordinates of the local origin of each block to the feature locations in the blocks.

The RANSAC algorithm applied in this paper makes use of an epipolar geometric estimation based on a fundamental matrix. To estimate the fundamental matrix from a minimal sample when fitting the model to determine inliers or outliers, a 7-point algorithm is used. For the final step to estimate the fundamental matrix from all inlier points, the normalised 8-point algorithm is used.

In this paper, our main objective is to reconstruct a more accurate scene; thus, a multi-level RANSAC (Zhao et al., 2010) is applied for outlier removal. The idea of a multi-level RANSAC is to firstly prune out outliers with large error, thereby leaving a higher inlier ratio for the next-step RANSAC. A two-level RANSAC is adopted in the experiments of this paper, in which the threshold of the first level is set to 2.0 pixels, and the threshold of the second level is set to 1.0 pixel. Our experiments show that the two-level RANSAC can provide more accurate feature matching (with slightly fewer

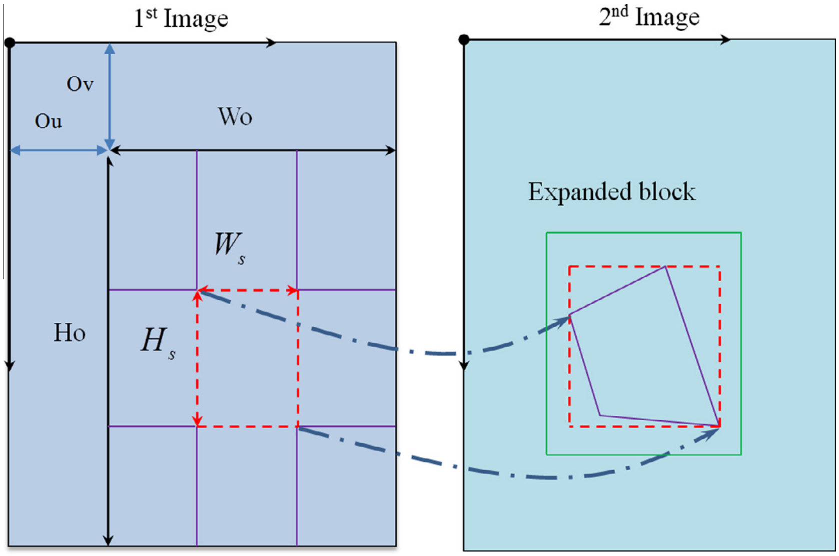
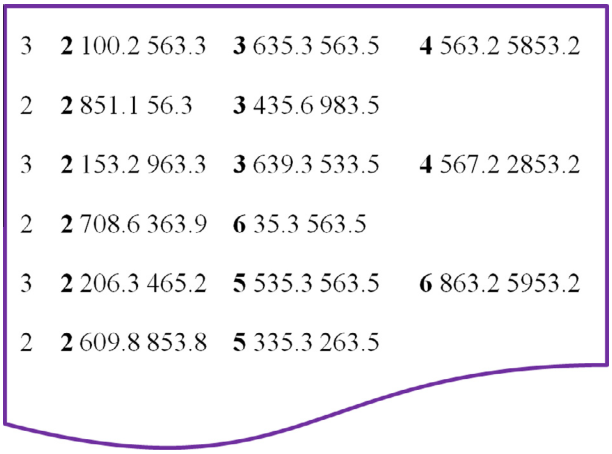


Fig. 4. Split image into blocks and find corresponding blocks in the second image.

inliers identified) compared to a normal one-step RANSAC and LO-RANSAC (Chum et al., 2003) with a threshold of 1.0 pixel (the same as that in the last level in the two-level RANSAC).

4. Tie points matching along-track and across-track

In this paper, tie points refer to image points that appear on different visual images and correspond to the same 3D feature, and finding the sets of tie points is called "tie points matching". Tie points are very important for reconstructing terrain scenes and estimating camera poses. A tie point sets file with a specified format needs to be generated as the input of a bundle adjustment. For example, Fig. 5 shows the format of the file as the input of ParallaxBA, which is almost the same as that of SBA (Lourakis and Argyros, 2009) (In the input of SBA, an initial value of the 3D feature positions is needed. However, it is unnecessary to provide initial values of the features in ParallaxBA because they are automatically calculated within the algorithm). Each row in the file represents a tie point set denoted as R_F^i where i is the index of the 3D feature. The first element of R_F^i is an integer N_i denoting the number of image points corresponding to the same 3D feature i; the rest of R_F^i contains the image points $\left\{F_i^{j_1}, F_i^{j_2} \ldots, F_i^{j_{N_i}}\right\}$ where each F_i^j includes the image index j and the corresponding image point coordinates (u, v).



```
3   2 100.2 563.3   3 635.3 563.5   4 563.2 5853.2
2   2 851.1 56.3    3 435.6 983.5
3   2 153.2 963.3   3 639.3 533.5   4 567.2 2853.2
2   2 708.6 363.9   6 35.3 563.5
3   2 206.3 465.2   5 535.3 563.5   6 863.2 5953.2
2   2 609.8 853.8   5 335.3 263.5
```

Fig. 5. A file of tie points which serves as the input to bundle adjustment (ParallaxBA).

For one pair of images, the Block-SIFT method presented in Section 3 can be used to generate correspondences. For multiple images, tie points matching is used to find all image points of each of the same features among a large amount of correspondences from many pairs of images. To accelerate tie points matching among numerous putative correspondences, a red-black tree structure is used in this paper. Furthermore, a tree key exchange method and a segment matching method are proposed to match along-track and across-track tie points. The process is detailed in the following subsections.

4.1. Red-black tree structure for tie points matching

In an aerial photogrammetry study case, there are many large images with rich texture information, whilst Block-SIFT can extract and match numerous image point correspondences. Consequently, it is extremely slow to conduct tie points matching by searching and comparing the image indexes and the image point coordinates directly. To reduce the computational cost of searching among many image point correspondences, it is necessary to utilise a special data structure. In this paper, a red-black tree data structure is applied.

A red-black tree is a type of advanced binary search tree and has already been applied in search-based applications (including text mining and nucleotide sequence matching). For a binary tree with n nodes, basic operations which include search, delete and insert, take O(lg n) time on average and take O(n) time in the worst case, when the binary tree is a linear chain of n nodes. A red-black tree as one form of balanced search trees is proposed to solve this issue and it takes O(lg n) time on basic operations in any case. Compared with the standard binary tree structure, it has extra colour property besides tree keys and tree pointers. Thus, a red-black tree has the least possible tree height and the searching complexity of traversing elements from root to leaf is O(lg n) (Cormen et al., 2009).

A node of a red-black tree for matching tie points contains a key and a node value, where the key can be set as the first image point, and the node value is all the image points in each line in Fig. 5.

When creating a red-black tree with tie point sets, nodes are arranged according to the image point index j and the image point coordinate (u, v). If the image point indexes are the same, the first image point coordinate u will determine where a node should be placed in the tree. Similarly, if both j and u values are the same, then the second image point coordinates v will be used to arrange the node positions. Fig. 6 gives an example of a red-black tree structure using the data in Fig. 5 according to the red-black tree rules, where the u values in the second image are used to arrange the node positions. In this red-black tree, one node of a tree represents a tie point set in which the left (red or black) colour element is a node key and the right element is the corresponding node value. In Fig. 6, the first image points in each row of Fig. 5 are set as the node keys, and all image points in each row R_F^i are their node values.

Supposing that the red-black tree has been constructed as in Fig. 6, we now illustrate how to perform the tie points matching when another two tie point sets, R_F^7 and R_F^8, become available. Assume the key of R_F^7 is the same as that of R_F^4, while the key of R_F^8 is different from any existing keys. To match R_F^7, two searching steps from the root is enough to find the same key that is located at the second level of the tree. The green dotted line in Fig. 7 describes the searching path. Now, the node value R_F^7 can be combined with the node value R_F^4 as shown in Fig. 7. To match R_F^8, because the same key cannot be found in the tree (within four searching steps as shown by the green dotted line in the left figure in Fig. 8), R_F^8 will be inserted into the tree. Afterward, the tree structure should be adjusted with the restriction of the red-black tree's properties; the new tree structure is in the form of the right figure in Fig. 8 instead of that as shown in the left figure of Fig. 8. For the details of the operations of the red-black tree, please refer to Chapter 13 in (Cormen et al., 2009).

In the following subsections, a red-black tree will be used as a basis in tie points matching both along-track and across-track.

4.2. Tree key exchange method for along-track tie points matching

Along-track tie points matching means to match the tie point sets from a sequence of images taken along-track. Supposing that there are n images in one track that will be processed by Block-SIFT, we can obtain $n-1$ groups of correspondences $\zeta = \{\zeta_1^2, \zeta_2^3 \ldots, \zeta_{n-1}^n\}$, where $\zeta_j^{j+1} = \left\{(F_i^j, F_i^{j+1}) : i \in M_{j(j+1)}\right\}$ is the group of correspondences between the jth image and the $(j+1)$th image (here $M_{j(j+1)}$ denotes the set of matched points between the jth image and the $(j+1)$th image). Similarly, another group of correspondences $\zeta_{j+1}^{j+2} = \left\{(F_k^{j+1}, F_k^{j+2}) : k \in M_{(j+1)(j+2)}\right\}$ is generated from the $(j+1)$th image and the $(j+2)$th image. Supposing that F_i^{j+1} and F_k^{j+1} are identical, F_i^j, F_i^{j+1}, F_k^{j+2} are projected from the same 3D feature and should be combined into a single tie point set $\left\{F_i^j, F_i^{j+1}, F_k^{j+2}\right\}$.

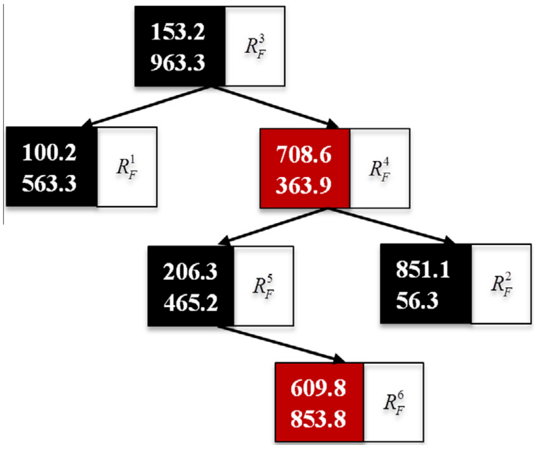


Fig. 6. Red-black tree structure using data in Fig. 5. All the keys have the same image index 2 which is ignored in the figure.

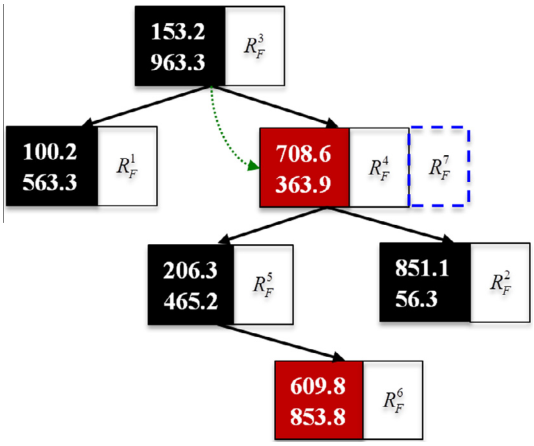


Fig. 7. Match R_F^7 – the same key is found and the tie point sets are combined.

In this paper, the tree key exchange method is proposed to rapidly match tie points among the numerous correspondences generated by Block-SIFT. The detailed implementation is described in Algorithm 1. First, the groups of correspondences ζ_j^{j+1} are extracted from each of two adjacent large images along-track using the Block-SIFT method in Section 3. Because there is not enough memory to store all of the correspondences in one track, ζ_j^{j+1} is firstly saved into a file. Second, load $\zeta_1^2 = \left\{\left(F_i^1, F_i^2\right) : i \in M_{1(2)}\right\}$ from the file and initialise a red-black tree R_b. Here, each tree node is $R_b^i = \left\{F_i^2 \middle| \left(F_i^1, F_i^2\right)\right\}$, whose node key is set as the second image point F_i^2. Third, for each $\zeta_j^{j+1} (j \geqslant 2)$, search the same keys in the tree, update the tree and exchange the tree keys for next step. Assuming that $\zeta_1^2, \ldots, \zeta_{j-1}^j$ have been processed, the current tree node is in the format of $R_b^i = \left\{F_i^j \middle| \left(F_i^1, \ldots, F_i^{j-1}, F_i^j\right)\right\}$. Now, load $\zeta_j^{j+1} = \left\{\left(F_k^j, F_k^{j+1}\right) : k \in M_{j(j+1)}\right\}$ from the file, and for each correspondence in ζ_j^{j+1}, compare F_k^j with the existing tree keys. If F_k^j is the same as the key of $R_b^i = \left\{F_i^j \middle| \left(F_i^1, \ldots, F_i^{j-1}, F_i^j\right)\right\}$, that is, $F_k^j = F_i^j$, then F_k^{j+1} can be directly added to the node R_b^i to get $R_b^i = \{F_i^j | \left(F_i^1, \ldots, F_i^{j-1}, F_i^j, F_k^{j+1}\right)\}$. If F_k^j is different from all of the existing keys in the tree, then the correspondence (F_k^j, F_k^{j+1}) is set as a new node $R_b^k = \left\{F_k^j \middle| \left(F_k^j, F_k^{j+1}\right)\right\}$ (the node key is F_k^j) and inserted into the tree. To prepare for the next step in matching pairs of correspondences in $\zeta_{j+1}^{j+2} = \left\{(F_k^{j+1}, F_k^{j+2}) : k \in M_{(j+1)(j+2)}\right\}$, tree key exchange is performed to change the key into the image point in the $(j+1)$th image, that is, to change $R_b^i = \left\{F_i^j \middle| \left(F_i^1, \ldots, F_i^{j-1}, F_i^j, F_k^{j+1}\right)\right\}$ into $R_b^i = \left\{F_k^{j+1} \middle| \left(F_i^1, \ldots, F_i^{j-1}, F_i^j, F_k^{j+1}\right)\right\}$ because it is impossible to match a correspondence in ζ_{j+1}^{j+2} with a node that does not contain an image point in the $(j+1)$th image. All of the nodes that do not contain an image point in the $(j+1)$th image are released from tree R_b and saved to the file. Finally, after ζ_{n-1}^n is processed, all along-track tie point sets are saved to the file ξ.

In the proposed tree key exchange method, when determining whether a new available tie point set should be combined with an existing tie point set, it is only necessary to search the key among the key sets of the red-black tree instead of all of the image points, the search cost is low, and the search process is efficient. Moreover, the proposed method only loads necessary candidate tie point sets and always immediately releases unnecessary ones; therefore, it can keep the number of nodes in the tree bounded and will not increase search runtime when dealing with more tie point sets. Hence, this method is also feasible and efficient when processing many along-track images without breaching the memory capacity.

Algorithm 1. Tree Key Exchange Method for Along-Track Tie Points Matching

```
Input: $n$ large images in one track
Output: all along-track tie point sets $\xi$
Block-SIFT
for $j = 1; j < n; j++$ do
    Apply Block-SIFT to the jth and $(j+1)$th image to get the group of correspondences $\zeta_j^{j+1} = \left\{ \left( F_i^j, F_i^{j+1} \right) : i \in M_{j(j+1)} \right\}$
    save $\zeta_j^{j+1}$ into file
end for
Initialize red-black tree $R_b$
Load $\zeta_1^2 = \left\{ \left( F_i^1, F_i^2 \right) : i \in M_{1(2)} \right\}$
for each $i \in M_{1(2)}$ do
    insert the node $R_b^i = \left\{ F_i^2 \middle| \left( F_i^1, F_i^2 \right) \right\}$
end for
Search same keys, update the tree and exchange tree keys
for $j = 2; j < n; j++$ do
    load $\zeta_j^{j+1} = \left\{ \left( F_k^j, F_k^{j+1} \right) : k \in M_{j(j+1)} \right\}$
    for each $k \in M_{j(j+1)}$ do
        search $F_k^j$ in $R_b$
        if $F_k^j$ is found $\left( F_k^j = F_i^j \right)$ then
            $F_k^{j+1}$ is directly added in $R_b^i$, $R_b^i = \left\{ F_i^j \middle| \left( F_i^1, \ldots, F_i^{j-1}, F_i^j, F_k^{j+1} \right) \right\}$
        else
            insert a new node $R_b^k = \left\{ F_k^j \middle| \left( F_k^j, F_k^{j+1} \right) \right\}$
        end if
    end for
    release the nodes that do not contain an image point in the $(j+1)$th image from the tree, save them into file $\xi$
    exchange tree keys for all the other nodes, set the new key as the image points in the $(j+1)$th image
        $R_b^i = \left\{ F_k^{j+1} \middle| \left( F_i^1, \ldots, F_i^{j-1}, F_i^j, F_k^{j+1} \right) \right\}$
        $R_b^k = \left\{ F_k^{j+1} \middle| \left( F_k^j, F_k^{j+1} \right) \right\}$
end for
Save all along-track tie point sets into file $\xi$
```

4.3. Segment matching method for across-track tie points matching

The across-track tie points matching refers to finding all the corresponding image points among a very large quantity of along-track tie point sets of multiple tracks obtained by the approach described in Section 4.2. Consequently, it is extremely time-consuming to search and match tie points within numerous tree nodes in a red-black tree if all of the tie point sets are loaded. In addition, it is impossible to load all of the tie point sets into a red-black tree due to the memory limit. To rapidly match across-track tie points without breaching the memory restriction, a segment matching method is proposed in this paper. The idea is: we split all of the along-track tie points into segments and only load the corresponding segments that possibly contain matched tie points to conduct matching. The detailed across-track tie points matching algorithm is stated below and illustrated by a simple example.

First, we discuss the size of the segment. The size of each segment should be determined by the forward overlap percentage. Supposing that the forward overlap percentage is ϵ, then the maximum number of images that observe a same 3D point is approximately $n_\varepsilon = \lceil 1/(1-\epsilon) \rceil$ where $\lceil a \rceil$ means the smallest integer greater than or equal to a (for example, when $\epsilon = 0.6$, $n_\varepsilon = 3$). Thus, if an image point in the kth image in the second track is matched with an image point in the jth image in the first track, then the images in the first track that may contain an image point of the same matched 3D feature include the jth image itself and its $2n_\varepsilon - 2$ neighbouring images ($n_\varepsilon - 1$ images on the left and $n_\varepsilon - 1$ images on the right). Therefore, the size of each segment is chosen as $2n_\varepsilon - 1$ in this paper. For example, when the forward overlap percentage is $\epsilon = 60\%$, the size of each segment is $2n_\varepsilon - 1 = 5$, as shown in Fig. 9.

Next, we show and explain the detailed steps of the method. In general, the segment-matching method contains five main steps: (i) split all along-track tie points into segments; (ii) create links between every image point and the first image point in an along-track set point set; (iii) match across-track tie points between overlapped images; (iv) load segments and match across-track points within putative along-track tie point sets; (v) collect and combine along-track and across-track tie points from all segments. The detailed implementation of tie points matching for two across-tracks is described in Algorithm 2. The input of the algorithm incorporates the two along-track tie point sets ξ_1 and ξ_2 and the n_1 and n_2 images from the two tracks. The output is all of the matched tie point sets. For more than two tracks, the operation is similar.

The first step of Algorithm 2 is to split all along-track tie points into segments. For each tie point set $R_F^i = \left\{ F_i^{j_1}, F_i^{j_2}, \ldots, F_i^{j_{N_i}} \right\}$ in ξ_1 or ξ_2, its segment index S_j can be determined according to the image index of the first image point $F_i^{j_1}$. If $F_i^{j_1}$ lies in the first track, then $S_j = [j_1/(2n_\varepsilon - 1)] + 1$ is the segment index for this tie point set and the segment is denoted as $\xi_1^{S_j}$. Here, $[a]$ denotes the largest integer less than or equal to a. If $F_i^{j_1}$ belongs to the second track, then appoint the tie point set into segment $\xi_2^{S_j}$ ($S_j = [(j_1 - n_1)/(2n_\varepsilon - 1)] + 1$, where n_1 is the total number of images in the first track). After appointing all tie point sets to different segments in this way, we only need to load the necessary segments instead of the whole ξ_1, ξ_2 in the fourth step of the algorithm. For example, in Fig. 9, there are 8 segments ξ_1^1, ξ_1^2, ξ_1^3, ξ_1^4 and ξ_2^1, ξ_2^2, ξ_2^3, ξ_2^4 each with the size of 5.

The second step is to create links between every image point in each tie point set and its first image point. The reason is that the keys of red-black tree are set as the first image points of the tie point sets; thus, an image point that is not the first one in a tie point cannot be found within tree keys. Therefore, we have to create a relationship between each image point and the first image point in a tie point set. In this step, we load each tie point set $R_F^i = \left\{ F_i^{j_1}, F_i^{j_2}, \ldots, F_i^{j_{N_i}} \right\} \in \xi_1 \bigcup \xi_2$, and the link $K_{j_p} = \left\{ F_i^{j_p} \rightarrow F_i^{j_1} \right\}$, $j_p \in \{j_1, j_2, \ldots, j_{N_i}\}$ ($F_i^{j_1}$ is the first image point in R_F^i) is constructed

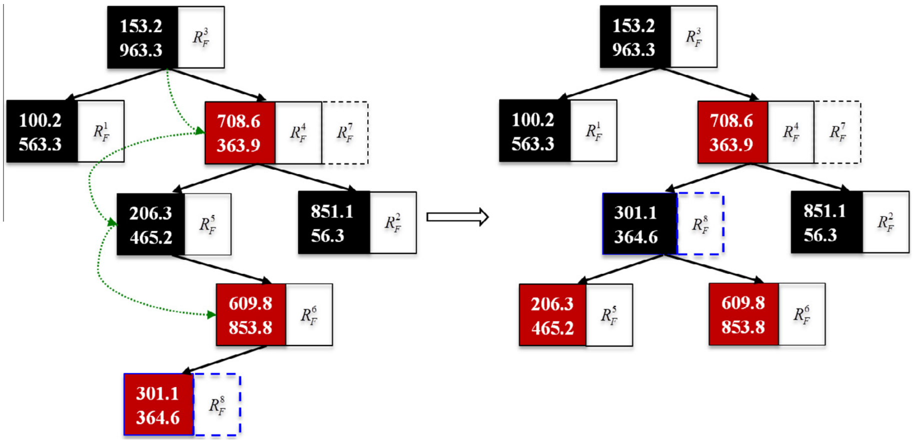


Fig. 8. Match R_F^8 – the same key cannot be found, adjust the tree structure and insert the tie point set as a new node.

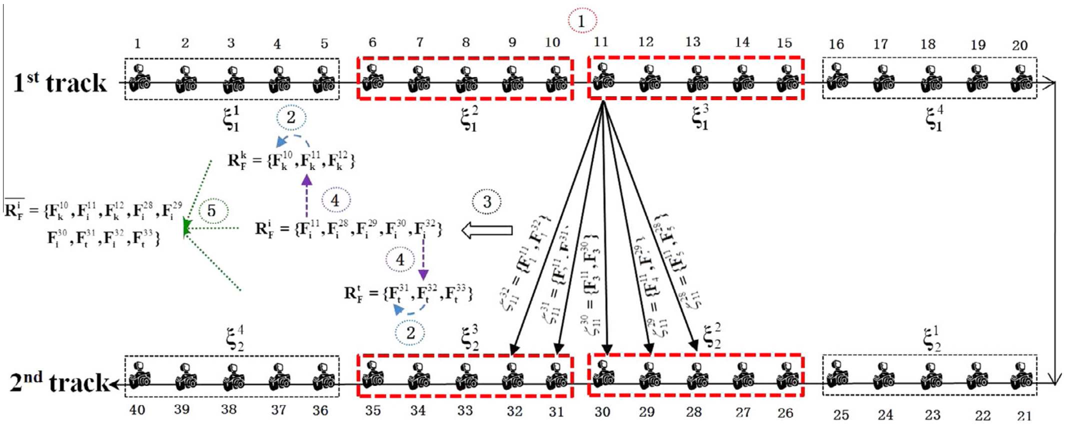

1st track
2nd track
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
ξ_1^1
ξ_1^2
ξ_1^3
ξ_1^4
ξ_2^4
ξ_2^3
ξ_2^2
ξ_2^1
1
2
3
4
5
$R_F^k = \{F_k^{10}, F_k^{11}, F_k^{12}\}$
$R_F^i = \{F_i^{11}, F_i^{28}, F_i^{29}, F_i^{30}, F_i^{32}\}$
$R_F^t = \{F_t^{31}, F_t^{32}, F_t^{33}\}$
$\overline{R_F^i} = \{F_k^{10}, F_i^{11}, F_k^{12}, F_i^{28}, F_i^{29}, F_i^{30}, F_t^{31}, F_i^{32}, F_t^{33}\}$
$\zeta_{11}^{32} = \{F_i^{11}, F_i^{32}\}$
$\zeta_{11}^{31} = \{F_i^{11}, F_i^{31}\}$
$\zeta_{11}^{30} = \{F_i^{11}, F_i^{30}\}$
$\zeta_{11}^{29} = \{F_i^{11}, F_i^{29}\}$
$\zeta_{11}^{28} = \{F_i^{11}, F_i^{28}\}$

Fig. 9. Across-track tie points matching among two adjacent tracks. Segment matching method is used. Only 2 segments from the first track and 2 segments from the second track are needed when matching one image in the first track to the images in the second track.

and saved to the key file. After such key files are created, if there are two corresponding image points that are not the first in the two tie point sets, we can match them easily by searching their first image points within tree keys with the help of the key files. For example, in Fig. 9, F_k^{11} of R_F^k and F_t^{32} of R_F^t can be linked with their first image points F_k^{10} and F_t^{31}.

The third step is to match across-track tie points between the overlapped images among two tracks. In aerial photogrammetry, an image in one track will overlap with many images in the adjacent track. For the jth image in the first track, the overlapped image set P_2^j in the second track can be calculated according to the snaked shape of the airline track. The groups of correspondences $\zeta_j^{jp} = \left\{(F_i^j, F_i^{jp}), i \in M_{j(jp)}\right\}$ between the jth image in the first track and the jpth image in the second tracks ($jp \in P_2^j$) are obtained using the Block-SIFT method described in Section 3. Then, construct a tree R_b where each node is denoted as $R_b^i = \left\{F_i^j \middle| \left(F_i^j, F_i^{jp}\right)\right\}$, while the node keys are set as image point F_i^j appearing on the jth image. The process is as follows: Initialise the tree using ζ_j^{j1} and traverse other groups of correspondences ζ_j^{jp}. Search the first image point of ζ_j^{j1} among the node keys. If found, the correspondence is added to the node value; if not, it will be inserted into R_b and become a new node. After this process, the node values become matched tie points between overlapping images P_2^j. Then, save all node values to the file and destroy the tree. For example, in Fig. 9, the 11th image has overlap areas with the 28th, 29th, 30th, 31th, and the 32th images, so five groups of correspondences can be obtained by Block-SIFT, which are $\zeta_{11}^{28} = \left\{(F_i^{11}, F_i^{28}),\ i \in M_{(11)(28)}\right\}$, $\zeta_{11}^{29} = \left\{\left(F_i^{11}, F_i^{29}\right),\ i \in M_{(11)(29)}\right\}$, $\zeta_{11}^{30} = \left\{\left(F_i^{11}, F_i^{30}\right),\ i \in M_{(11)(30)}\right\}$, $\zeta_{11}^{31} = \left\{\left(F_i^{11}, F_i^{31}\right),\ i \in M_{(11)(31)}\right\}$ and $\zeta_{11}^{32} = \left\{\left(F_i^{11}, F_i^{32}\right),\ i \in M_{(11)(32)}\right\}$. Because the 11th image is the common image for the pair-wise matching, F_i^{11} are set as the tree node keys. After the tree is constructed, all of the node values are the matched image points from the 11th, 28th, 29th, 30th, 31th, and the 32th images, and they are save to the file. For example, one matched tie point set can be represented as $R_F^i = \left\{F_i^{11}, F_i^{28}, F_i^{29}, F_i^{30}, F_i^{32}\right\}$.

The fourth step is to match across-track points within putative along-track tie point sets in the corresponding segments. For each image in the first track, load tie points from the file in the third step and search the corresponding tie points in ξ_1 and ξ_2. When processing the jth image in the first track, only segments $\xi_1^{s_j}$, $\xi_1^{s_j+1}$, $\xi_2^{s_{\bar{j}}}$ and $\xi_2^{s_{\bar{j}}+1}$ are required to be loaded, where $s_j = [(j + n_\varepsilon - 1)/(2n_\varepsilon - 1)]$ and $s_{\bar{j}} = [(\bar{j} - n_1 + n_\varepsilon - 1)/(2n_\varepsilon - 1)]$ (here $\bar{j}$ is the index of the image located in the second track that has the most overlap areas with the jth image). If $\xi_1^{s_j}$ or $\xi_2^{s_{\bar{j}}}$ equals zero, only segment $\xi_1^{s_j+1}$ or $\xi_2^{s_{\bar{j}}+1}$ is loaded. Similarly, if $\xi_1^{s_j+1}$ or $\xi_2^{s_{\bar{j}}+1}$ is larger than the maximum number of segments, then only $\xi_1^{s_j}$ or $\xi_2^{s_{\bar{j}}}$ is necessarily loaded. These segments can provide all possible corresponding tie points. Now, use $\xi_1^{s_j}$ and $\xi_1^{s_j+1}$ to build a red-black tree R_{b1}, and use $\xi_2^{s_{\bar{j}}}$ and $\xi_2^{s_{\bar{j}}+1}$ to build another tree R_{b2}. The tree keys are set as the first image points of each tie point set. Because the first image point of any tie point set in file in the third step comes from the first track and all the other image points in the tie point set are from the second track, we will then search for the corresponding image points of the first image point in R_{b1} and search for the corresponding image points of the other image points in R_{b2}. With the help of the link files created in the second step, it is easy to find the first image point in a tie point set. Assume that F_k^{j1} is found when searching F_i^j in key file, then we search F_k^{j1} in the tree R_{b1} and assume that the node value R_F^k is found. For each other image point F_i^{jp} $(jp \in P_2^j)$ in R_F^i, the first image point F_t^{j1} of the tie point set is found in the key file when searching F_i^{jp}. Then, search F_t^{j1} among the node keys of R_{b2} and find the corresponding node value R_F^t. Furthermore, R_F^i, R_F^k and R_F^t are combined into a new tie point set $\bar{R}_F^i$. For example, in Fig. 9, for $j = 11,\ \bar{j} = 30,\ s_j = 2,\ s_{\bar{j}} = 2$ and four segments $\xi_1^2,\ \xi_1^3,\ \xi_2^2$ and ξ_2^3 (with red bold rectangle) need to be loaded when matching across-track tie points between the 11th image and its overlapped images. Suppose there are two tie point sets $R_F^k = \{F_k^{10}, F_k^{11}, F_k^{12}\} \in \xi_1^2 \bigcup \xi_1^3$ and $R_F^t = \left\{F_t^{31}, F_t^{32}, F_t^{33}\right\} \in \xi_2^2 \bigcup \xi_2^3$. F_k^{11} and F_i^{11} are corresponding image points, whilst F_t^{32} and F_i^{32} are also corresponding image points. When matching R_F^k and R_F^t with $R_F^i = \left\{F_i^{11}, F_i^{28}, F_i^{29}, F_i^{30}, F_i^{32}\right\}$, it is necessary to search their first image points F_k^{10} and F_t^{31} within tree keys. Now, R_F^k, R_F^i and R_F^t can be combined to become a new matched tie point set. In the last step, the tie point sets in different segments are collected and combined. Thus, all of the tie points are obtained.

In the proposed segmentation method, only the necessary tie point sets among corresponding segments are loaded into a red-black tree, and the number of nodes in the tree is kept bounded, so it is efficient to match across-track tie points with numerous along-track tie points.

Algorithm 2. Segment Matching for Across-Track Tie Points Matching

```
Input: two along-track tie point sets $\xi_1$ and $\xi_2$, $n_1$ and $n_2$ images from two tracks
Output: all along-track and across-track tie points
Split $\xi_1$ and $\xi_2$ into segments
compute segment size $s_g$
for each tie point set $R_F^i = \left\{F_i^{j_1}, F_i^{j_2}, \ldots, F_i^{j_{N_i}}\right\} \in \xi_1 \bigcup \xi_2$ do
    if $R_F^i \in \xi_1$ then
        $S_j = [j_1/(2n_\varepsilon - 1)] + 1$
        appoint $R_F^i$ into $\xi_1^{S_j}$ segment
    else
        $S_j = [(j_1 - n_1)/(2n_\varepsilon - 1)] + 1$
        appoint $R_F^i$ into $\xi_2^{S_j}$ segment
    end if
end for
Create link between every image points with the first image point in a tie point set in $\xi_1 \bigcup \xi_2$
load $R_F^i = \left\{F_i^{j1}, F_i^{j2}, \ldots, F_i^{j_{N_i}}\right\}$ in $\xi_1 \bigcup \xi_2$
for each image point in $R_F^i$ do
    create link $K_{jp} = \left\{F_i^{j_p} \rightarrow F_i^{j_1}\right\},\ j_p \in \left\{j_1, j_2, \ldots, j_{N_i}\right\}$
    save $K_{jp}$ into key file
end for
Match across-track tie points between overlapped images
for $j = 1; j < n_1; j++$ do
    compute the set of overlapped images $P_2^j$ in the second track
    for each image $jp$ $(jp \in P_2^j)$ do
        extract and match $\zeta_j^{jp} = \left\{\left(F_i^j, F_i^{jp}\right),\ i \in M_{j(jp)}\right\}$ using Block-SIFT
    end for
    construct a red-black tree $R_b$, each node $R_b^i = \left\{F_i^j \middle| \left(F_i^j, F_i^{jp}\right)\right\}$
    for each $\zeta_j^{jp}$ do
        search $F_i^j$ in $R_b$
        combine node values or insert new nodes
    end for
    destroy tree $R_b$ and save matched nodes into file
end for
Load segments and match across-track points within putative along-track tie point sets
for $j = 1; j < n_1; j++$ do
    load segments $\xi_1^{s_j}$, $\xi_1^{s_j+1}$, $\xi_2^{s_{\bar{j}}}$ and $\xi_2^{s_{\bar{j}}+1}$ from files
    $s_j = [(j + n_\varepsilon - 1)/(2n_\varepsilon - 1)]$
    $s_{\bar{j}} = [(\bar{j} - n_1 + n_\varepsilon - 1)/(2n_\varepsilon - 1)]$
    create tree $R_{b1}$ using $\xi_1^{s_j}$ and $\xi_1^{s_j+1}$, whose keys are the first image points in the tie point sets.
    create tree $R_{b2}$ using $\xi_2^{s_{\bar{j}}}$ and $\xi_2^{s_{\bar{j}}+1}$, whose keys are the first image points in the tie point sets.
    load $R_F^i = \left\{\left(F_i^j, F_i^{jp}\right), jp \in P_2^j\right\}$ from file
    search $F_i^j$, find $F_k^{j1}$
    search $F_k^{j1}$ in $R_{b1}$, find node value $R_F^k$
    for each $F_i^{jp}$, $jp \in P_2^j$ do
        search $F_i^{jp}$, find $F_t^{j1}$
        search $F_t^{j1}$ in $R_{b2}$, find node value $R_F^t$
    end for
    combine $R_F^k, R_F^i$ and $R_F^t$ into $\bar{R}_F^i$
end for
Collect and combine along-track and across-track tie points from all segments
```

5. Experimental results

In the experiments, seven datasets are used to check the validity and accuracy of the proposed L^2-SIFT algorithm. First, we discuss the influence of the two parameters, namely, the block size and the expended size, on the number of correspondences extracted and matched by the proposed L^2-SIFT algorithm. Then, for two large images, we compare the number of correspondences extracted using the L^2-SIFT algorithm and that extracted using the SiftGPU algorithm. For all of the test datasets in large-scale photogrammetry listed in Table 1, the detailed runtime of extracting and matching the tie points is listed in Table 2. To evaluate the accuracy of the extracted features, ParallaxBA (Zhao et al., 2011) is applied where reprojection error in the objective function can

Table 1
Overview of the test datasets.

	Toronto	Vaihingen	DunHuan	Jinan	Village	College	Taian
Camera	UCD	DMC	Cannon	Cannon	DMC	Cannon	DMC
#Images	13	20	63	76	90	468	737
#Tracks	3	3	3	2	4	14	12
Image Size	11500 × 7500	7680 × 13824	5616 × 3744	5616 × 3744	7680 × 13824	5616 × 3744	7680 × 13824
Forward Overlap	60%	60%	60%	60%	80%	60%	60%
Side Overlap	30%	60%	30%	30%	50%	30%	30%

measure the accuracy of the feature extraction and matching results of L^2-SIFT. In addition, three-dimensional terrains are reconstructed based on the ParallaxBA method with a large amount of image points extracted and matched by the L^2-SIFT algorithm. Finally, L^2-SIFT and some functions of MicMac software (Pierrot-Deseilligny and Clery, 2011) are compared.

5.1. Datasets

Detailed parameters of the seven test datasets are shown in Table 1. All of the images are taken by high-quality cameras equipped on aircraft platforms, and they contain abundant terrain texture information. Among these datasets, publicly available Toronto and Vaihingen datasets are from the ISPRS Test Project on Urban Classification and 3D Building Reconstruction (Cramer, 2010; Rottensteiner et al., 2011; Rottensteiner et al., 2012). All experiments are executed on an Intel CPU i5-760 computer on the Windows platform with a 2.5 GHz CPU and 1 G GeForce GT 220 graphics card.

5.2. Analysis on the number of correspondences influenced by block size and expanded size

Block size and expanded size are two important parameters of the L^2-SIFT algorithm. The number of correspondences that can be extracted and matched strongly depends on these two parameters. In this section, we will discuss what size blocks should be split in the first image and what size corresponding blocks should be expanded in the second image, aiming to extract more correspondences. We will experiment on a pair of along-track images from the Vaihingen dataset. The size of the images is 7680 × 13824 pixels. The first octave of SIFT starts from -1, and the two thresholds of RANSAC are set as 2.0 pixels and 1.0 pixel, respectively.

To analyse the influence of the block size, the expanded size is set as 50 pixels and the block sizes in the first image are set as 250 × 250, 500 × 500, 1000 × 1000, 2000 × 2000, 3000 × 3000, 4000 × 4000, 5000 × 5000, 6000 × 6000, 7000 × 7000 pixels, respectively. The number of correspondences is counted and depicted in Fig. 10. The results show that fewer correspondences will be generated with larger block sizes and the number of correspondences begins to drop quickly when the size of the split block is up to 3000 × 3000 pixels. The reason for this result is that for many candidate correspondences, the ratio of the smallest Euclidian distance between two descriptor sets to the second smallest one is smaller than a given threshold when the block size is larger and there are many unrelated feature descriptors; thus, fewer features are matched. In addition, when the block size is large, L^2-SIFT must first down-sample the split block instead of extracting and matching them directly. Thus, if the block size is smaller, more correspondences will be extracted. However, considering that this will greatly increase the total runtime when the block is very small, we set the block size as 500 × 500 pixels in this paper.

To analyse the influence of the expanded size, the block size is set as 500 × 500 pixels and the expanded sizes are set as 0, 25, 50, 100, 200, 300, 400, 500, and 1000 pixels, respectively. The number of correspondences is shown in Fig. 11. It is clearly shown that the number of correspondences increases when the expanded size in the second image ranges from zero to 50 pixels, and it reduces when the expanded size ranges from 50 to 1000 pixels. Because the transformation parameters are inaccurate, the image block in the second image cannot contain the specified image block in the

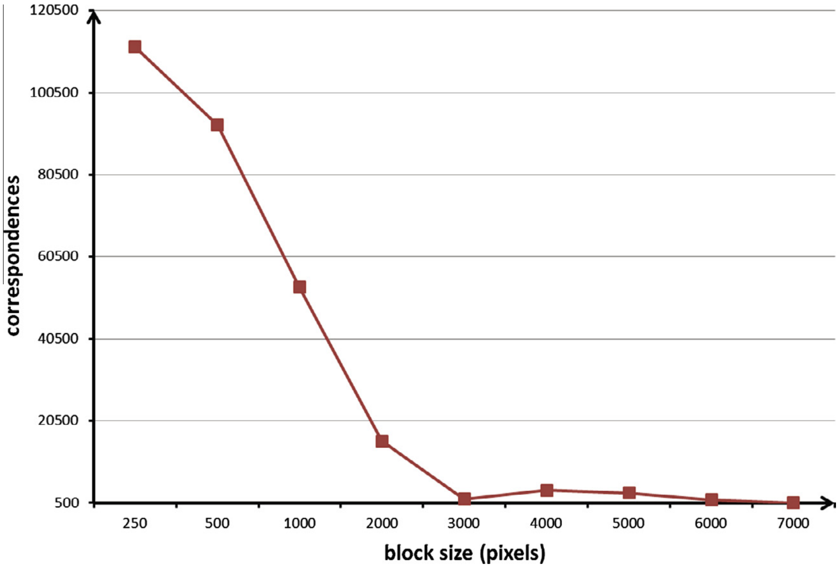


Fig. 10. Number of correspondences affected by the block size in L^2-SIFT.

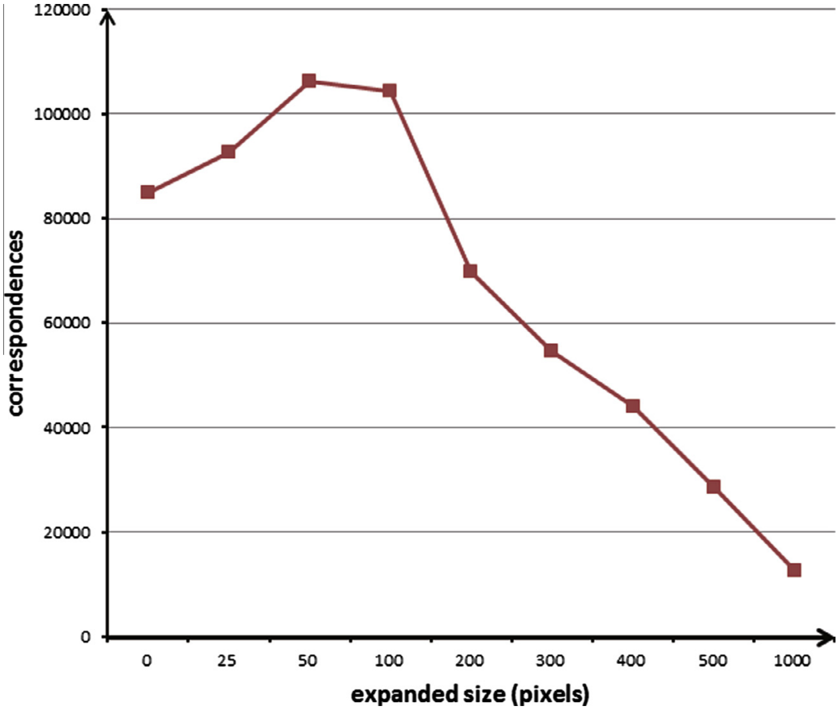


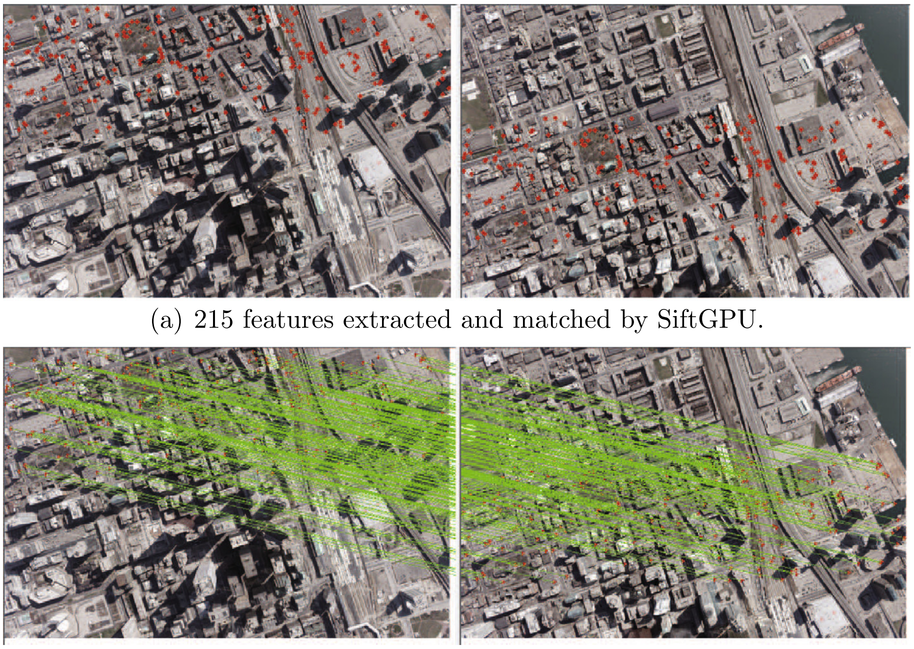
Fig. 11. Number of correspondences affected by the expanded size in L^2-SIFT.

first image entirely if it does not expand or merely expands a little. However, if the image block in the second image expands too much, there will be many unrelated feature descriptors, and many candidate correspondences cannot satisfy the matching rule and are discarded in the matching procedure, as described previously.

To sum up, considering the number of correspondences and runtime, the block size in the first image should be neither too small nor too large. In this paper, the block size is set as 500 × 500 pixels. In the meantime, the expanded size in the second image should be large enough such that the corresponding block in the second image can contain the specified block in the first image. However, it should not be too large to avoid redundant feature descriptors. In this paper, the expanded size is set as 50 pixels.

5.3. Feature extracting and matching for two large images

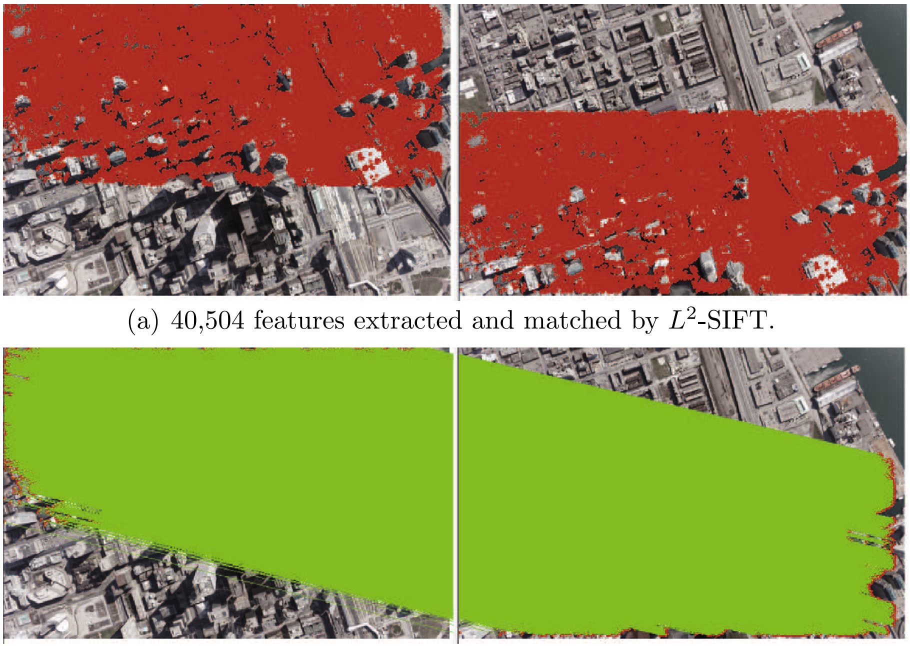
With limited memory capacity, the original SIFT algorithm cannot process large images directly. In many open software packages, such as SiftGPU, the down-sampling procedure is used to extract and match features for large images, which brings out a loss of texture information and results in few or even no features matched in aerial photogrammetry. In the L^2-SIFT algorithm, Block-SIFT can



(a) 215 features extracted and matched by SiftGPU.

(b) Correspondences are linked with green lines.

Fig. 12. Features from Toronto dataset extracted by SiftGPU after outliers removal using RANSAC.



(a) 40,504 features extracted and matched by L^2-SIFT.

(b) Correspondences are linked with green lines.

Fig. 13. Features from Toronto dataset extracted by L^2-SIFT after outliers removal using RANSAC.

first split the overlapped part of a whole image into blocks and each of the corresponding blocks is processed separately so that many reliable features can be extracted without reaching the memory limit. In this subsection, the numbers of correspondences from two large images using the SiftGPU algorithm and the L^2-SIFT algorithm are compared. The two images used are from the Toronto dataset with a size of 11500 × 7500 pixels. For the L^2-SIFT algorithm, we set the block size as 500 × 500 pixels and the expanded size as 50 pixels. Meanwhile, RANSAC is used to remove outliers. For SiftGPU, because the original images are down-sampled into 2875 × 1875 pixels before feature extraction and matching, only 215 correspondences can be extracted using SiftGPU, as shown in Fig. 12, where the red points represent features, whereas L^2-SIFT can extract 40,504 correspondences, depicted in Fig. 13. From the results, we can see that the L^2-SIFT algorithm can avoid texture information loss and extract a very large number of reliable features when processing one pair of large images.

5.4. Feature extracting and matching for large-scale photogrammetry

In this section, the number of features and image points extracted and matched from the seven datasets and the detailed runtime of each procedure and the total time are listed in Table 2.

The L^2-SIFT algorithm contains six procedures, which are extraction and matching from adjacent along-track images by Block-SIFT (SIFT-ALT), matching tie points along-track by the tree key exchange method (Key-Exchange), splitting all tie point sets in one track into segments (Split-Seg), extraction and matching from overlapped across-track images by Block-SIFT (SIFT-ACT), matching across-track tie points by the segment matching method (SegMat) and combining segments (Combine-Seg). For all test datasets, the start octave of SIFT is set as 0, and the two-level RANSAC thresholds are 2.0 pixels and 1.0 pixels. For the College dataset, it takes approximately 3 h to extract 3,305,361 features, in which there are 8,302,182 image points, and for the Taian dataset, it takes approximately 21 h to extract 33,208,549 features, in which there are 74,464,965 image points (the first row in Table 2 represents the number of features, and the second row shows the number of image points).

5.5. Accuracy analysis by bundle adjustment

Although the RANSAC algorithm has been used to remove outliers, there may still be outliers in the tie point sets. Bundle adjustment is currently the most accurate and widely used method in 3D reconstruction; therefore, it is an ideal tool to assess the quality of tie points by using the objection function about the reprojection errors. In this section, we use ParallaxBA to evaluate the accuracy of the features extracted and matched from the L^2-SIFT algorithm, which has more advantages than other bundle adjustment methods in terms of convergence (Zhao et al., 2011).

In ParallaxBA, a camera pose is represented as rotation angles and position of the camera. The ith camera pose $\mathbf{P}$ is represented by

Table 2
Runtime statistic of each procedure (in minutes).

	Toronto	Vaihingen	DunHuan	Jinan	Village	College	Taian
Features	113685	554169	250784	1229011	3282901	3305361	33208549
Image points	239279	1201982	597294	2864868	8683526	8302182	74464965
SIFT-ALT	5.6	10.8	11.5	8.9	55.8	33.6	400.8
Key-Exchange	0.4	1.6	0.4	0.7	7.1	4.7	43.8
Split-Seg	0.3	1.2	0.5	0.5	8.3	4.4	81.8
SIFT-ACT	8.5	19.1	21.2	14.7	82.4	120.5	372.3
SegMat	3.1	2.5	3.3	0.7	49.6	20.5	306.6
Combine-Seg	0.3	1.1	0.4	0.4	6.1	3.4	39.8
Total time	18.2	36.3	37.3	25.9	209.3	187.1	1245.1

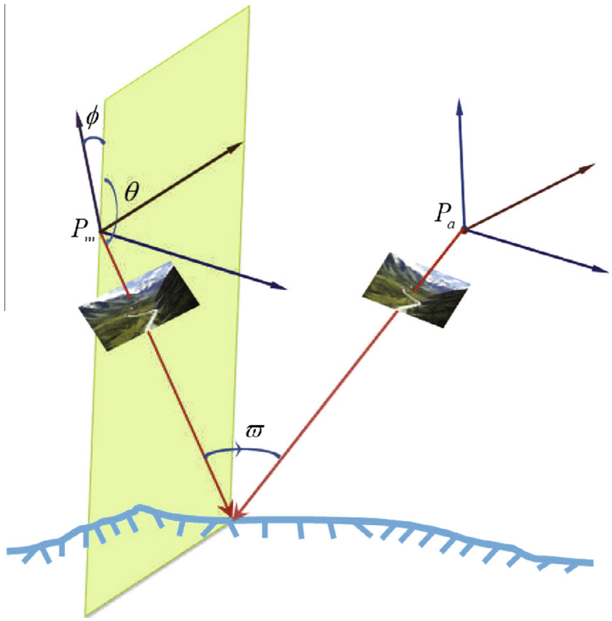


Fig. 14. Parametrization for a 3D feature in ParallaxBA.

$$\mathbf{P} = [\varphi, \omega, \kappa, x, y, z]^T \tag{10}$$

where $[\varphi, \omega, \kappa]^T$ are the yaw, pitch, roll angles of **P** and $[x, y, z]^T$ are the three-dimensional coordinates of the exposure position of camera. In this paper, the global coordinate system in ParallaxBA optimisation locates the first image coordinate. That is, the 1^{st} camera pose is $[0, 0, 0, 0, 0, 0]^T$.

A feature is parameterized by three angle in ParallaxBA instead of Euclidean XYZ (see Fig. 14).

Supposing that a feature **F** is observed by two cameras $\mathbf{P}_m$ and $\mathbf{P}_a$, define $\mathbf{P}_m$ as the main anchor and $\mathbf{P}_a$ as the associate anchor of **F**. $\mathbf{X}_m^F$ and $\mathbf{X}_a^F$ are corresponding rays which are from the main and associate anchor to the feature, respectively. The feature is described as follows:

$$\mathbf{F} = [\phi, \theta, \varpi]^T \tag{11}$$

where ϕ is the azimuth angle and θ is the elevation angle. ϖ is the parallax angle from the main corresponding ray $\mathbf{X}_m^F$ to the associate corresponding ray $\mathbf{X}_a^F$.

The tie points extracted and matched by L^2-SIFT are used as the input of ParallaxBA for all seven aerial photogrammetry datasets. The MSE of the feature reprojection errors is an important index to assess the features quality, computed by

$$\epsilon^2 = \frac{1}{N}\sum_{i=1}^{N}((\bar{u}_i - u_i)^2 + (\bar{v}_i - v_i)^2) \tag{12}$$

where N is the total number of image points, $[\bar{u}_i,\ \bar{v}_i]$ denotes the reprojected image point coordinates in ParallaxBA, and $[u_i, v_i]$ denotes the extracted image point using L^2-SIFT.

Table 3
Accuracy analysis of L^2-SIFT by MSE using ParallaxBA.

Dataset	#features	#image points	#outliers	MSE
Toronto	113685	239279	0	0.041
Vaihingen	554169	1201982	0	0.119
DunHuan	250782	597289	2	0.168
Jinan	1228959	2864740	52	0.126
Village	1544021	4098528	58	0.0953
College	1236502	3107524	75	0.734
Taian	2743553	6016649	148	0.03874

Because of the memory limitations, it is not possible to load all the features from the Village, College and Taian datasets into ParallaxBA. We only use part of the features to execute the ParallaxBA optimisation for these datasets. The number of features, the number of image points, the number of outliers removed, and the final MSE of ParallaxBA using the seven datasets are listed in Table 3.

Terrains are reconstructed with tie points, which are extracted and matched by L^2-SIFT. The three-dimensional point clouds from the results of ParallaxBA are shown in Fig. 15, in which the point clouds of the Toronto, Vaihingen, DunHuan and Jinan datasets are drawn in Fig. 15(a)–(d) respectively. Similarly, the point clouds of the Village, College and Taian datasets are shown in Fig. 15(e), (g) and (i), respectively, and for the Village, College and Taian datasets, the features that were not used in ParallaxBA are triangulated by using the camera poses optimised by ParallaxBA. The full point clouds are shown in Fig. 15(f), (h) and (j). Because 3D point clouds are reconstructed by ParallaxBA in a relative coordinate system in which the first camera's coordinate system is regarded as the global coordinate system, the colours in the figures have no practical implications.

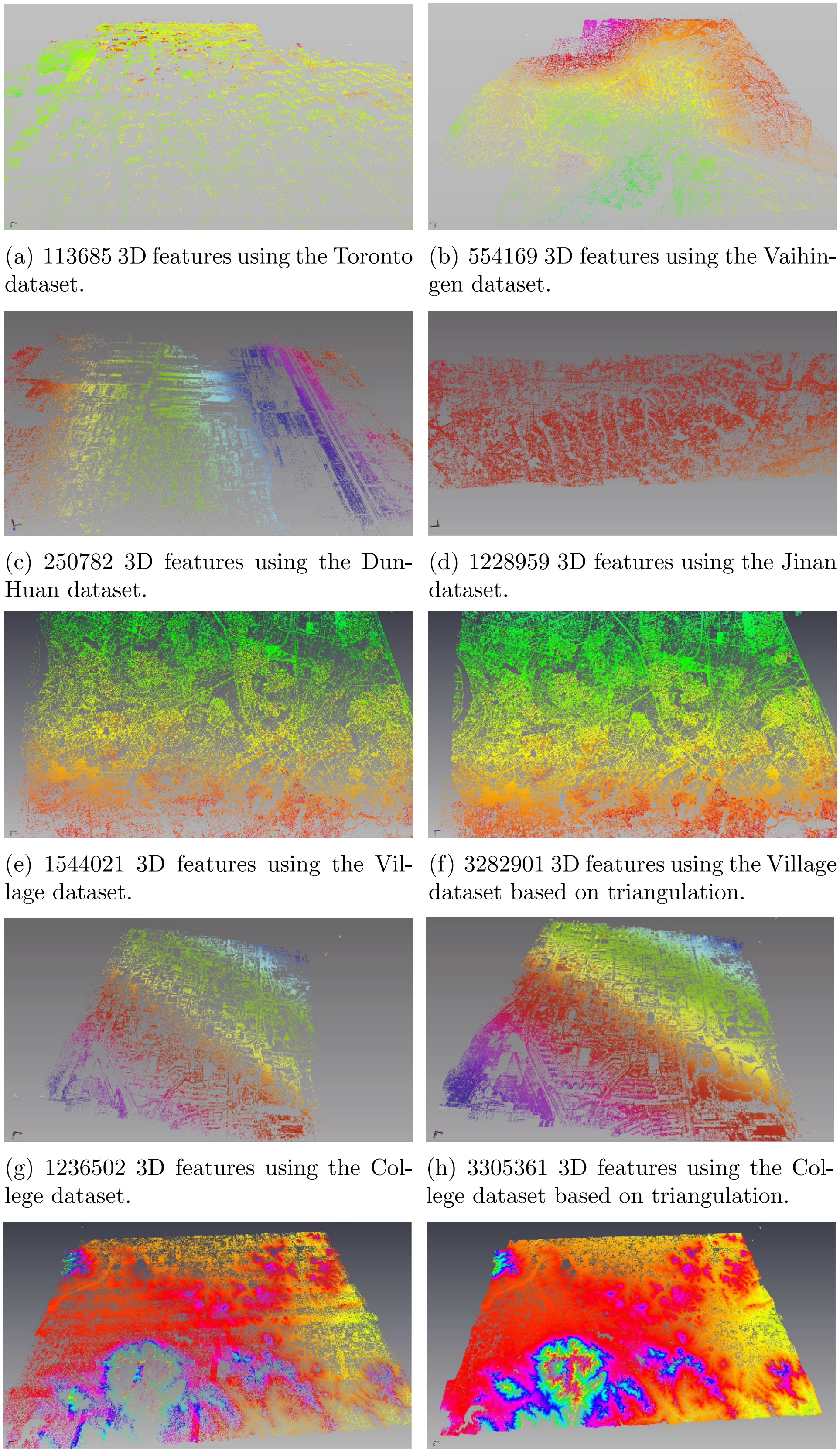
5.6. Compare L^2-SIFT with MicMac

MicMac is an open-source photogrammetric software package developed and published by the MATIS laboratory of the Institute Graphique National (IGN) (Pierrot-Deseilligny and Paparoditis, 2006). As one of the best software programs in photogrammetry to compute and reconstruct 3D models, it has been applied in a wide spectrum of applications (Pierrot-Deseilligny and Clery, 2011). Among all of the functions of MicMac, the Tapioca function, which makes full use of multi-core parallel computation technology, can provide a solution to extract and match features from large images. The features extracted and matched by the Tapioca function are used as the input of the Apero function, which uses one type of bundle adjustment algorithm to optimise camera poses. The Tapioca function utilises an external solution SIFT++ to process large images with low efficiency. In this section, a comparison between some functions of MicMac and L^2-SIFT are performed, including the number of features extracted and matched from two large images, the total runtime of features extraction and matching in a study area. Meanwhile, the MSE of BA with L^2-SIFT is compared with that of MicMac to assess the accuracy of the features extracted and matched by L^2-SIFT.

First, two images in the publicly available Village dataset are used to compare the number of features extracted and matched from two large images whose size is 7680×13824 pixels, as shown in Table 1. The number of features and the total runtime (in minutes) are shown in Table 4, in which the first-octave parameters are set as 0. This table shows that L^2-SIFT is faster than MicMac. In L^2-SIFT, we have also tried to simply use SIFT++ instead of SiftGPU to extract features from the block pairs, and the time used is similar to that of MicMac. This proves that the time advantage of L^2-SIFT mainly benefits from the utilisation of SiftGPU.

Second, to compare the total runtime of features extraction and matching in a study area, the Toronto, Vaihingen and Village datasets are chosen. In MicMac, a specified file that indicates the image distribution is given as an input of the Tapioca tool. From Table 5, it is clear that L^2-SIFT is much faster when processing many images in a study area.

Finally, to verify the accuracy of the features extracted and matched by L^2-SIFT, the MSE of ParallaxBA is compared with that by the Apero tool in MicMac. In Table 6, the MSE with L^2-SIFT is slightly less than that of MicMac, showing that L^2-SIFT can extract reliable features.



(a) 113685 3D features using the Toronto dataset.

(b) 554169 3D features using the Vaihingen dataset.

(c) 250782 3D features using the DunHuan dataset.

(d) 1228959 3D features using the Jinan dataset.

(e) 1544021 3D features using the Village dataset.

(f) 3282901 3D features using the Village dataset based on triangulation.

(g) 1236502 3D features using the College dataset.

(h) 3305361 3D features using the College dataset based on triangulation.

(i) 2743553 3D features using the Taian dataset.

(j) 33208549 3D features using the Taian dataset based on triangulation.

Fig. 15. Reconstructed terrains by ParallaxBA using features from L^2-SIFT.

Table 4
Features and time comparison for two large images between L^2-SIFT and MicMac.

	Number of features	Time used (in minutes)
MicMac	81684	34.7
L^2-SIFT	62898	0.6

Table 5
Time comparison for a study area between L^2-SIFT and MicMac (in minutes).

	Toronto	Vaihingen	Village
MicMac	380.2	1755.6	10109.1
L^2-SIFT	18.16	36.31	209.3

Table 6
MSE of BA comparison between L^2-SIFT and MicMac (unit is square pixels).

	Toronto	Vaihingen	Village
MicMac	0.1775	3.186	1.2435
L^2-SIFT	0.041	0.119	0.0953

6. Conclusions and future work

In this paper, we present the L^2-SIFT algorithm to extract and match features from large images in large-scale aerial photogrammetry. The Block-SIFT method, the red-black tree structure, the tree key exchange method and the segment matching method are proposed in the L^2-SIFT algorithm. The Block-SIFT method is proposed to overcome the memory limitation problem when matching two large images. The red-black tree structure, tree key exchange method and segment matching method are used to improve the efficiency of tie points matching along-track and across-track in large-scale aerial photogrammetry. Finally, the detailed runtime of the L^2-SIFT algorithm is listed, and the accuracy of the extracted features are evaluated by the MSE of reprojection errors in ParallaxBA using seven real-world aerial datasets. As shown in the experiments, the L^2-SIFT algorithm can efficiently extract numerous high-quality features from large images and accurately match tie points among a very large number of unordered image points in large-scale aerial photogrammetry.

In this paper, we assumed near-zero pitch and roll angles of the platform in the method for extracting and matching 2D image features. This assumption may not hold in many applications, such as those using an Unmanned Aerial Vehicle. In future work, we will investigate how to extract and match a pair of large images taken from a platform with large pitch or roll angles; thus, the estimated transformation between two images is not limited to 2D.

Acknowledgements

The authors would like to thank the authors of SiftGPU, MicMac and SIFT++ for making their algorithms available as open-source which is really helpful to the research described in this paper.

References

Barazzetti, L., Remondino, F., Scaioni, M., 2010. Extraction of accurate tie points for automated pose estimation of close-range blocks. In: ISPRS Technical Commission III Symposium on Photogrammetric Computer Vision and Image Analysis.

Bayer, R., 1972. Symmetric binary b-trees: data structure and maintenance algorithms. Acta Inform. 1 (4), 290–306.

Bay, H., Ess, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-up robust features (surf). Comput. Vis. Image Underst. 110 (3), 346–359.

Chum, O., Matas, J., Kittler, J., 2003. Locally Optimized RANSAC. Pattern Recognition. Springer, Berlin Heidelberg, pp. 236–243.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2009. Introduction to Algorithms, 3rd ed. MIT Press, pp. 309–338.

Cramer, M., 2010. The DGPF-test on digital airborne camera evaluation overview and test design. Photogram. Fernerkundung Geoinform. 2010 (2), 73–82.

Crisp, D., Tao, T., 2010. Fast region merging algorithms for image segmentation. In: Proc. the Fifth Asian Conference on Computer Vision, Melbourne, Australia, pp. 412–417.

Fink, E., Goldstein, A., Hayes, P., Carbonell, J.G., 2004. Search for approximate matches in large databases. In: Proc. Systems, Man and Cybernetics, pp. 1431–1435.

Fischler, M., Bolles, R., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24 (6), 381–395.

Guibas, L., Sedgewick, R., 1978. A dichromatic framework for balanced trees. In: Proc. 19th Annual Symposium on Foundations of Computer Science, pp. 8–21.

Harris, C., Stephens, M., 1988. A combined corner and edge detector. In: Proc. Alvey Vision Conference, pp. 147–151.

Ke, Y., Sukthankar, R., 2004. PCA-sift: a more distinctive representation for local image descriptors. In: Proc. Computer Vision and Pattern Recognition (CVPR), pp. 506–513.

Lindeberg, T., 1998. Feature detection with automatic scale selection. Int. J. Comput. Vis. 30 (2), 79–116.

Lourakis, M.I., Argyros, A.A., 2009. SBA: a software package for generic sparse bundle adjustment. ACM Trans. Math. Softw. 36 (1), 1–30.

Lowe, D., 2004. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. 60 (2), 91–110.

Mikolajczyk, K., Schmid, C., 2005. A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell. 27 (10), 1615–1630.

Moravec, H., 1981. Rover visual obstacle avoidance. In: Proc. 7th International Joint Conference on Artificial Intelligence, vol. 2, pp. 785–790.

Morel, J.M., Yu, G., 2009. ASIFT: a new framework for fully affine invariant image comparison. SIAM J. Imaging Sci. 2 (2), 438–469.

Novák, D., Baltsavias, E., Schindler, K, 2011. Reliable image matching with recursive tiling. In: Photogrammetric Image Analysis, pp. 49–60.

Pierrot-Deseilligny, M., Clery, I., 2011. APERO, an open source bundle adjustment software for automatic calibration and orientation of set of images. Int. Arch. Photogram. Remote Sens. Spatial Inform. Sci. XXXVIII (5).

Pierrot-Deseilligny, M., Paparoditis, N., 2006. A multiresolution and optimization-based image matching approach: an application to surface reconstruction from SPOT5-HRS stereo imagery. Int. Arch. Photogram. Remote Sens. Spatial Inform. Sci. 36 (part 1/w41), 73–77.

Rottensteiner, F., Baillard, C., Sohn, G., Gerke, M., 2011. ISPRS Test Project on Urban Classification and 3D Building Reconstruction.

Rottensteiner, F., Sohn, G., Jung, J., Gerkec, M., Baillard, C., Sebastien, B., and Uwe, 2012. The ISPRS Benchmark on Urban Object Classification and 3D Building Reconstruction. In: Proc. XXII ISPRS Congress, Melbourne.

Schenk, T., 1997. Towards automatic aerial triangulation. ISPRS J. Photogram. Remote Sens. 52 (3), 110–121.

Strecha, C., Bronstein, A.M., Bronstein, M.M., Fua, P., 2012. LDAHash: improved matching with smaller descriptors. IEEE Trans. Pattern Anal. Mach. Intell. 34 (1), 66–78.

Umeyama, S., 1991. Least-squares estimation of transformation parameters between two point patterns. IEEE Trans. Pattern Anal. Mach. Intell. 13 (4), 376–380.

Vedaldi, A., 2010. SIFT++. <http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html>.

Wu, C., 2007. SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT). <http://cs.unc.edu/~ccwu/siftgpu>.

Zhang, Z., Ganesh, A., Liang, X., Ma, Y., 2012. TILT: transform invariant low-rank textures. Int. J. Comput. Vis. 99 (1), 1–24.

Zhao, L., Huang, S., Yan, L., Wang, J.J., Hu, G., Dissanayake, G., 2010. Large-scale monocular SLAM by local bundle adjustment and map joining. In: Proc. 11th International Conference on Control Automation Robotics & Vision (ICARCV), pp. 431–436.

Zhao, L., Huang, S., Yan, L., Dissanayake, G., 2011. Parallax angle parametrization for monocular SLAM. In: Proc. IEEE International Conference on Robotics and Automation (ICRA), pp. 3117–3124.