École Polytechnique Fédérale de Lausanne

Mars Image registration to track changes on Mars

Matching very large images and detection of south polar caps of Mars

Author:
Miryam Chaabouni

Supervisor:
Dr. Anton Ivanov

June 26, 2014

Abstract

The purpose of this project is to find the adequate tools to compare pairs of very large images of the surface of Mars taken by a satellite in a time interval of two to ten years, and to detect changes that occurred during this period. In addition to matching the identical areas between two images, we are particularly interested in finding specific spherical pits that appear in the south pole of Mars, often called 'Swiss Cheese features' or SCF.
The algorithm selected for this project is David Lowe's scale invariant feature detector SIFT [4], or more specifically, a GPU version [15] that is more adapted to the big dimensions of our images. This program will be used for the images registration, but also to detect the regions of interest by matching the images with selected samples of SCF images.

Contents

1 Introduction

Figure 1 shows an example of a Martian terrain that contains "Swiss cheese" features. Such terrains can only be found in the southern polar cap, mostly composed of frozen carbon dioxide and water[8]. It is believed that these circular depressions are caused by successive erosions and deposit of CO2 and water ice, whose rates depend on the season. According to a summary paper from 2009 [10], these pits have a basis layer of water ice, topped by a thinner layer of carbon dioxide ice, that will sublimate during the summer. Whereas the CO2 layer will completely disappear during the summer in the North Pole, the caps may survive in the southern summer where the temperature is lower. An outward growth of the depressions has been observed. Studying the evolution of these features gives the scientists information about the climate composition of Mars.

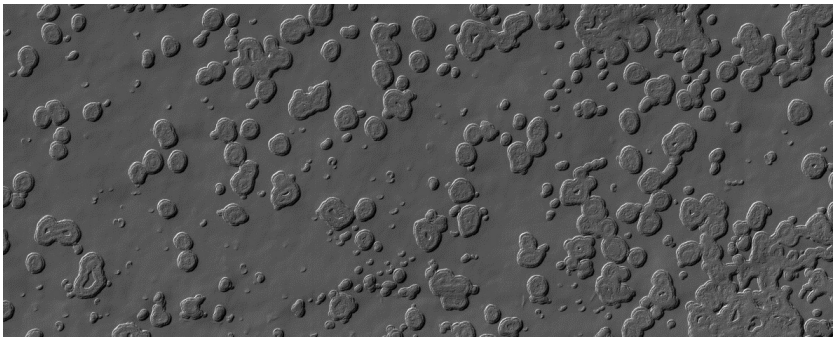

Figure 1: Pits in south polar ice cap. Extracted from HiRISE image ESP_012271_0940[8]

'The problem of image alignment and 2D motion estimation has been studied in computer vision over the past thirty years' [14], therefore there is a wide variety of registration algorithms and feature detectors that were created, each adapted to specific requirements and constraints.
Performance is often an issue in this field, as many detectors use operations that are computationally expensive, and some applications require real-time execution. In this context, many attempts to improve efficiency of these algorithms have already been made as in[16].

The next section explains the details of the project and its main challenges. The main goal is to find a (preferably existing) algorithm that would be adapted for the registration of Mars images. This is usually made using *feature detectors* that come in different types exposed in section3.
One of the most famous algorithms for feature detection is called SIFT[4]. It has proven to be one of the best detectors [5], and has been many times extended for various purposes. The characteristics of this algorithm will be explained in section 4.
A recent paper published in February 2014 exposes how to perform matching between large photogrammetry images using a parallel Version of SIFT ([9] and [15]). Although it could not be applied here, it has inspired the key steps of a SiftGPU based algorithm described in section 5.2.
The algorithm will be tested on HiRISE images of the surface of Mars that contain the particular features mentioned above. The results of the program for different parameters choices are exposed and discussed in the results section.

2 Project description and objectives

We will work on very large images (over 1Gb) of the surface of Mars, that come in the shape of long stripes (20'000x40'000 pixels). The data was taken by the Mars Reconnaissance Orbiter camera HiRISE (High Resolution Imaging Science Experiment). It comes in the JPEG2000 format, a complex compression standard that requires specific decoders. The images may have different sizes, and the angle from which they were taken may vary (vertical or diagonal of the area) thus producing image pairs that do not completely overlap. In addition to that, the images were taken under different illumination conditions, which may create shades that will modify the pits shape from an image to another.

As a first step, we need to find an optimal way to manipulate very large images. Among the wide variety of options available, an appropriate program to perform the registration of the images should be investigated. Once the matching between two images is performed, we should find a way to reduce the number of matching points to keep only the regions of interest that correspond to south polar caps

of Mars. This may be done manually, but we would like to create a automatic procedure that will perform this operation efficiently.
Finally, we want to detect and measure the changes that occurred over time. Once we have extracted small portions of images that we need, we can use a different technique for the comparison and measurement of the pits we are interested in. This can be done by creating a projection of an image on another then simply subtract the *warped* image to observe the motion as explained in section **??**.

3 Feature detectors : a literature review

In this section, we explore the methods that are usually used in computer vision to detect specific features, and that are used afterwards to perform image registration.
"A local feature is an image pattern which differs from its immediate neighbourhood" either by intensity, color or texture. It can be a point, a region or an edge segment. The survey made by Tuytelaars and Mikolajczyk[12] provides a complete and comprehensive description of various types of detectors. Though this survey is not exhaustive, it lists the most popular detectors, whose performance has been tested by several papers and that are more likely to be found as open source. The methods cited are categorized according to the type of feature detected, and the properties fulfilled.
As we will use grayscale data, the detectors of interest in our context are intensity based, e.g., they rely on the intensity gradient or its second order derivative.
The most important properties that a feature detector should have in our context are :

Repeatability We want to be able to detect the same feature in two images taken under different viewing conditions. This is important for matching two pairs of images as well as for motion tracking.

Invariance As mentioned previously, the images we will analyse can be taken from different angles and may have different sizes. Therefore, the algorithm should be affine invariant and scale invariant.

Efficiency As our images are very large, we want an efficient algorithm.

The survey[12] provides a table that summarises their analysis of feature detectors. It is partly reproduced in table 1.

Feature Detector	Rotation invariant	Scale Invariant	Affine Invariant	Repeatability	Efficiency
Harris-Laplace	√	√		+++	+
Hessian-Laplace	√	√		+++	+
DoG (SIFT)	√	√		++	++
SURF	√	√		++	+++
Harris-Affine	√	√	√	+++	++
Hessian-Affine	√	√	√	+++	++
Salient Regions	√	√	(√)	+	+
Edge-Based	√	√	√	+++	+

Table 1: Overview of feature detectors (extracted from table 7.1 from Tuytelaars and Mikolajczyk [12])

In this table, we see that SIFT is rotation and scale invariant, but not affine invariant. Besides, other detectors seem more efficient than SIFT. However, many studies have proved SIFT to perform better than most detectors[5]. Consequently, it became a very popular tool that has been improved and extended for various contexts. Its many versions can easily be found in open source. SURF is actually based on SIFT but was made to run faster. A parallel version of SIFT was produced by Wu in 2007[15], then by Yang and al. in 2011 [16].
Concerning the missing invariance characteristic, an affine invariant SIFT was proposed in 2009 by Morel and Yu [2], and two GPU version were published simultaneously in 2013 ([1] and [11]). Unfortunately, both versions are unavailable to the public. However, some basic tests show that SIFT is so robust that it can accomplish correct matching between two images taken from different angles, even though it is theoretically not affine invariant (see example in figures 2 and 3).

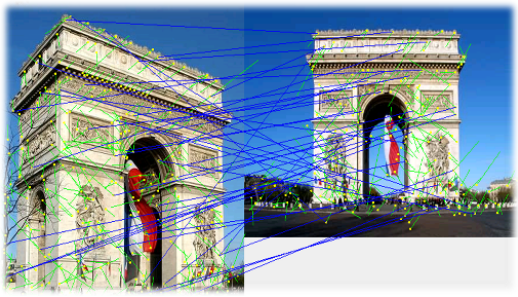

Figure 2: Example of matching points found by SIFT from *developpez.com*[7]

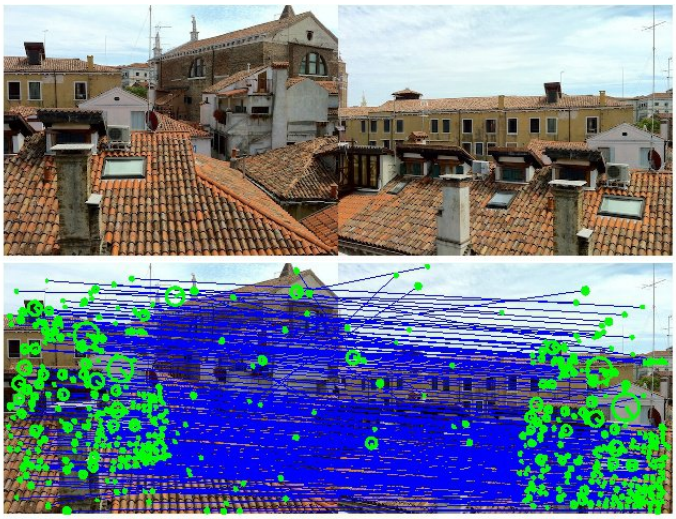

Figure 3: Example of matching points found by SIFT from *VLFeat open source library* [13]

4 Scale invariant feature detector : SIFT

SIFT will find interesting points called Sift Key Points based on their gradient of intensity, and calculate a descriptor of each point. To perform the matching, the same operation is applied to the second image. Then, the descriptors of each key point is compared between the two images, and a score is computed. The matching points are the key points that have a high score.
The main strength of SIFT is that it is scale invariant, e.g., if the same object is repeated but has a very different size in each image, SIFT will be able to detect it anyway. In fact, SIFT will successively smooth the image by a Gaussian filter, then builds an image pyramid formed of sub-samples of the image at different scales. The features of interest are those which persist over selected scales. This property is very useful when working with images taken by a satellite from different heights.
The main steps of SIFT are[6] :

1. Feature extraction (detection of key points)
   (a) Apply difference of Gaussians operator to identify candidate key points.
   (b) Analyse features to determine location and scale by comparing to adjacent neighbours at different scales.
   (c) Local curve fitting to reject points with low contrast or poorly localized
   (d) Uniform thresholding to remove key points with low contrast
   (e) Determine orientation by computing the local gradient direction
2. Feature description : for each key point, a descriptor is created by sampling magnitudes and orientations of the gradient in the region of the key point.

4.1 Parallel implementations : SiftGPU and L^2-SIFT

As each SIFT key point is accompanied by a descriptor (vector of size 128), the algorithm is data intensive and computation intensive. With the rapid development of multimedia data, and the necessity of dealing with huge databases, many researchers became interested in

developing parallel implementations of features detectors for real-time image processing. When working with very large images, a parallel implementation of SIFT is most adequate.

SiftGPU is a implementation of SIFT for GPU. It is an open source program developed in the university North Carolina in 2007. As explained by the author : "SiftGPU processes pixels parallely to build Gaussian pyramids and detect DoG Keypoints. Based on GPU list generation, SiftGPU then uses a GPU/CPU mixed method to efficiently build compact keypoint lists. Finally keypoints are processed parallely to get their orientations and descriptors."[15].
The advantages of this program is that it can work on almost any graphic card. Apart from the SIFT detector, it also provides a method to perform matching. It comes with several examples and two GUI applications. It was implemented in C++ with Visual Studio 2010©.

Very recently, a program based on SiftGPU was published which is called L^2-SIFT[9]. It was specifically designed to perform matching of large scale aerial photogrammetry images. Using this program, the authors matched large images of the city of Toronto taken successively by a camera flying over the city. This problem is extremely similar to ours. Unfortunately, it could not be used for HiRISE Mars images for technical reasons.
First of all, the size of the images, and the JPEG2000 format is not supported by the program. In fact, it only accepts TIF format with 8bits/channel. As will be explained later, the consequent size of the image was a major obstacle to convert them into this specific format.
Second of all, the output of the program, or more specifically, of the function that matches two images *L2Sift_2Img*, is a list of the coordinates of the matching points, and do not give information about the surroundings. Therefore, it is not sufficient to find the Martian pits.
Finally, the program must be provided with indexes indicating the relative positions of the camera in the two images. The input images are divided into blocks, and the position of corresponding areas is calculated to avoid unnecessary computations. As the position of the satellite is usually indicated for each image, the Block-SIFT method developed in this paper would constitute an efficient way to treat our problem. However, some features might be missed when using this method.
Therefore, a different program will be implemented based on some

concepts of L^2-SIFT. In their work, the authors have proven that L^2-SIFT runs faster than an equivalent program, Mic-Mac, thanks to the efficiency of SiftGPU[15]. For that reason, the program written for this project will also use SiftGPU.

5 Using SiftGPU on Mars images

5.1 Decompressing JPEG2000 images

The major difficulty of this project is to handle the full resolution images. Apart from their specific compression in JPEG2000 format, it is their abnormally big size that raises a number of serious problems. In fact, although many programs exist that support the JPEG2000 format, most of them will abort during execution, producing a memory error (Matlab©, JPEG Resampler).
After thorough investigations a program that was capable of handling our full resolution samples, was found. Kakadu software© [1] is a JPEG2000 developer kit that allows to easily and efficiently manipulate JPEG2000 images. Apart from a GUI (kdu_show) that provides an environment for viewing, rotating and zooming, it also come along with applications compress and decompress (kdu_expand) the image from JPEG2000 to the following formats : BMP, PGM, PPM, TIF, TIFF, RAW and RAWL.
Using the option -region, one can decompress only a specified region of the image. Because SiftGPU will fail to execute when presented with the full size decompressed images, this option will be used to cut the image into smaller pieces.
In order to produce the totality of the image components, the program will make successive calls to Kakadu via the C++ function CreateProcess. As Kakadu is very efficient, performing this step, even thousands of times on small regions takes a relatively short amount of time. However, as the size of the sub-image grows, the sub-sampling step will be slower. Kakadu works in parallel with the CPU threads (but there is a single thread option).
There exist GPU based JPEG2000 decoders that should be mentioned

[1] An evaluation version is available online and can be used for six months at `http://www.kakadusoftware.com/index.php?option=com_content&task=view&id=26&Itemid=22`

: *cuj2k* for NVIDIA graphics cards (CUDA based) [2] and *gpu_jpeg2k*[3]) for Ubuntu operating systems.

5.2 Algorithm

After generating the sub-images in PGM format[4], the data is passed to SiftGPU. The first step of the program is to do registration between two images. Given the size of the images, the number of matching Keys-points generated will be huge. In order to reduce this number, one can select only the ones that correspond to Swiss-cheese features. To do so, the idea is to manually extract the image of such a feature, then perform matching with the previously obtained Key-points.
However, comparing with a single pit image, arbitrarily chosen is not sufficient. Some circular features could be excluded by the matching function just because of different illumination conditions, or for other reasons. Ideally, the comparison should be made with several reference. In order to keep a reasonable complexity, only three reference images are used.
The key steps of the algorithm are:

1. Run SiftGPU over all sub-images of the first image to obtain the key points and descriptors.
2. Do the same for the second image.
3. Run SiftGPU on three reference images of SCF (figure 5).
4. Matching part : for all sub-images,
   (i) perform matching between all descriptors of the first image and all those of the second image.
   (ii) If two sub-images have enough matching points, perform matching with the SCF references.
   (iii) If there are enough matches with the reference images, write the coordinates of the matching points to a file.

In step (ii), we use an arbitrary threshold on the number of matching points to decide whether two areas are similar or not. Similarly,

[2]Freely available at `http://cuj2k.sourceforge.net/`

[3]Copyright 2009-2013 Poznan Supercomputing and Networking Center. Freely available at `http://apps.man.poznan.pl/trac/jpeg2k`

[4]For some reason, this is better accepted by the SiftGPU than TIF or RAW files.

we use a second threshold in the step (iii) to decide if the key point corresponds or not to a south polar cap of Mars. The effect of these thresholds will be discussed in the results section.

The complexity of the algorithm depends almost exclusively on the number of the sub-images. The two first steps consist of loops over the number of sub-samples N. Inside each loop, Kakadu software is called, which runs in time roughly proportional to the size of the sub-image, e.g., inversely proportional to N. Therefore, the complexity of each of the two first steps is roughly $N(1 + 1/N)$. This tends to N when N is large, which correspond to the choice of a fine grid for the sampling. Assuming that the third step runs in constant time, we find that the program will run in approximately $\Theta(2N + N^2) \approx \Theta(N^2)$. Therefore, the number of sub-samples will dramatically increase the computation time of the algorithm.
Comparing with L^2-Sift, we find that this complexity could be reduced to a linear growth if the relative positions of the two images are known. Indeed, L^2-Sift will first find mathematically the corresponding sub-image in the second image before performing the matching (figure 4). This would reduce the double loop of step (i) to a simple loop.

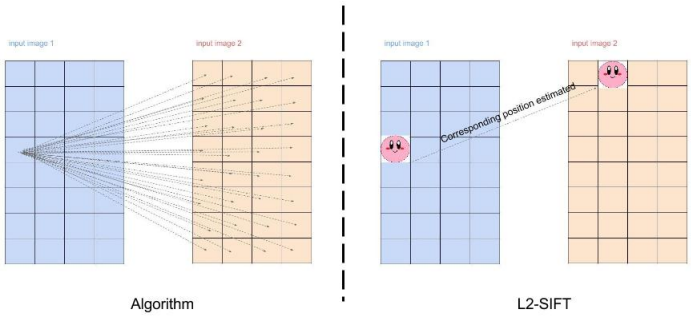

Figure 4: Difference between the algorithm and L^2-SIFT. The matching is performed between each pair of sub-images, wheras L^2-SIFT calculates the corresponding block in the second image and then performs matching.

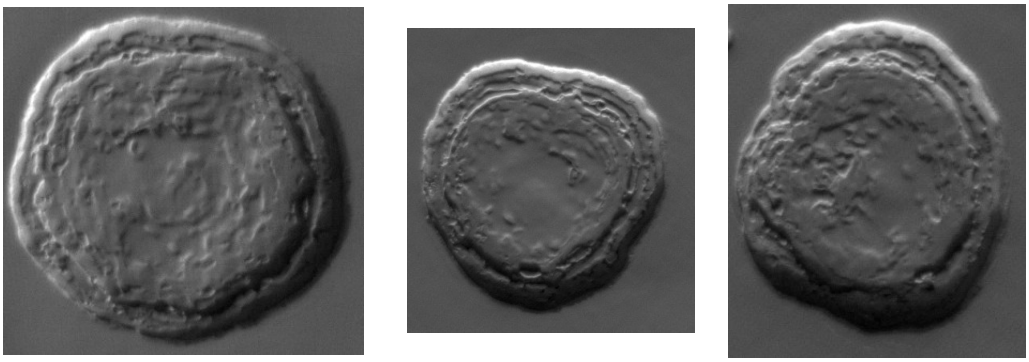

Figure 5: Samples of Swiss cheese features used by the program to detect key points corresponding to Mars pits, extracted from ESP_012271_0940 [8]

6 Results and discussion

The data used for testing the program is a HiRISE images ESP_012271_0940. Three reference circular shapes were extracted from the aforementioned image and are shown in figure 5.

The output of the algorithm comes as a text file with four columns of numbers corresponding to the absolute coordinates of the matching points in the original image. A simple way to visualize the results is to use the Matlab function **match_plot**[3] which was specifically designed to visualise matching SIFT feature points. However, Matlab has the disadvantage of not supporting the full size images; therefore, all the visual representations will be made using small blocks of the full size image.

6.1 Grid size

A primary test shows that, SiftGPU will automatically down-sample the image if it exceeds a certain size. This limitation comes from the OpenGL library, which SiftGPU is based on. Empirically, we find that the down-sampling can be avoided if the sub-samples have a size of $\approx 500 \times 500$ pixels. Therefore, the image should be divided in a 40×80 grid, which represents exactly 3200 sub-images. As explained in section 5.2, such grid size will dramatically increase the running time of the algorithm, exceeding two hours for $20'000 \times 40'000$ pixels images. Thus, most of the basic tests below will be performed using

a 4×8 grid, which gives sub-images of about 5000×5000 pixels.

6.2 Accuracy

In order to appreciate the accuracy of the algorithm, the same image is used for inputs (ESP_012271_0940). For now, the parameter thresholds are arbitrarily set to very low values : 3 for the SCF shape threshold, and to 100 for the area threshold. A deeper analysis of the influence of these parameters will be discussed in the next paragraphs. Intuitively, we can suppose that the program is correct if the matched points are the same in both images.

An example of the result is shown in figure 6. Due to Matlab's

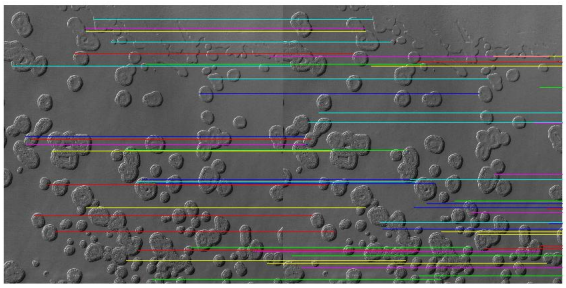

Figure 6: Example of matching points provided by the algorithm on a sub-image extracted from ESP_012271_0940. Sub-image size 5012×5000 pixels. Parameters : 8×4 grid, t_a= 100, t_{SCF} = 3.

memory limitations, only one sub-image (upper left corner) is shown. However, we can verify by a simple subtraction that the outputted points are the same; which indicates that the program accuracy is satisfactory. We notice that some matching points do not correspond to Martian pits, and that many of them ignored. This can be improved by adjusting the parameters.

6.3 Area threshold

The area threshold gives a measure of the similarity between two sub-images. It is the minimum number of common features that two identical areas should have. It also has a major influence on the running

time of the algorithm, as it determines whether or not the steps (ii) and (iii) (see section 5.2) will be executed.
The goal of this section is to find the optimal area threshold that will rightly determine whether two areas are the same while reducing the computations time. To do so, we will work again with two identical inputs. We also use a coarse grid of 32 sub-images to speed up the operations.
In figure 7, we can see the total number of matches found by the program as function of the threshold area. For very low values of t_a (smaller than 100), the program will find matches between any pair of sub-images, and the results will not be too noisy (see figure 8). Starting from 100, we can observe a plateau indicating that increasing the threshold will not affect the number of matches found. This means that a threshold value of 100 is sufficient to ascertain that two areas are identical. If the threshold value is too high, the program will never find similar areas, and the number of matches will be zero. We can verify that the program will find about 4000 matches between two identical sub-images, which is in accordance with the figure 7.
Actually, a limit of 4096 matches is arbitrarily imposed by the pro-

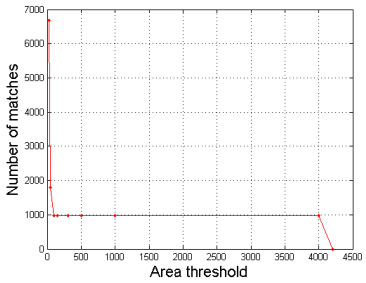

Figure 7: Number of matching points corresponding to an SCF. Upper left corner of ESP_012271_0940. Sub-image size 5012×5000. Parameters : grid size 32. th_{SCF} = 3.

gram in order to restrain the memory consumption.
We can conclude from these observations that the area threshold can be set anywhere from 100 to 4096 for this configuration. If another grid or a different image is chosen, the optimal area threshold might be slightly different. As explained above, the size of the sub-images used here will induce a down-sampling by SiftGPU. Thus, if a finer grid is chosen, there will be a greater number of features detected which might result in a more important noise. Besides, the total number of

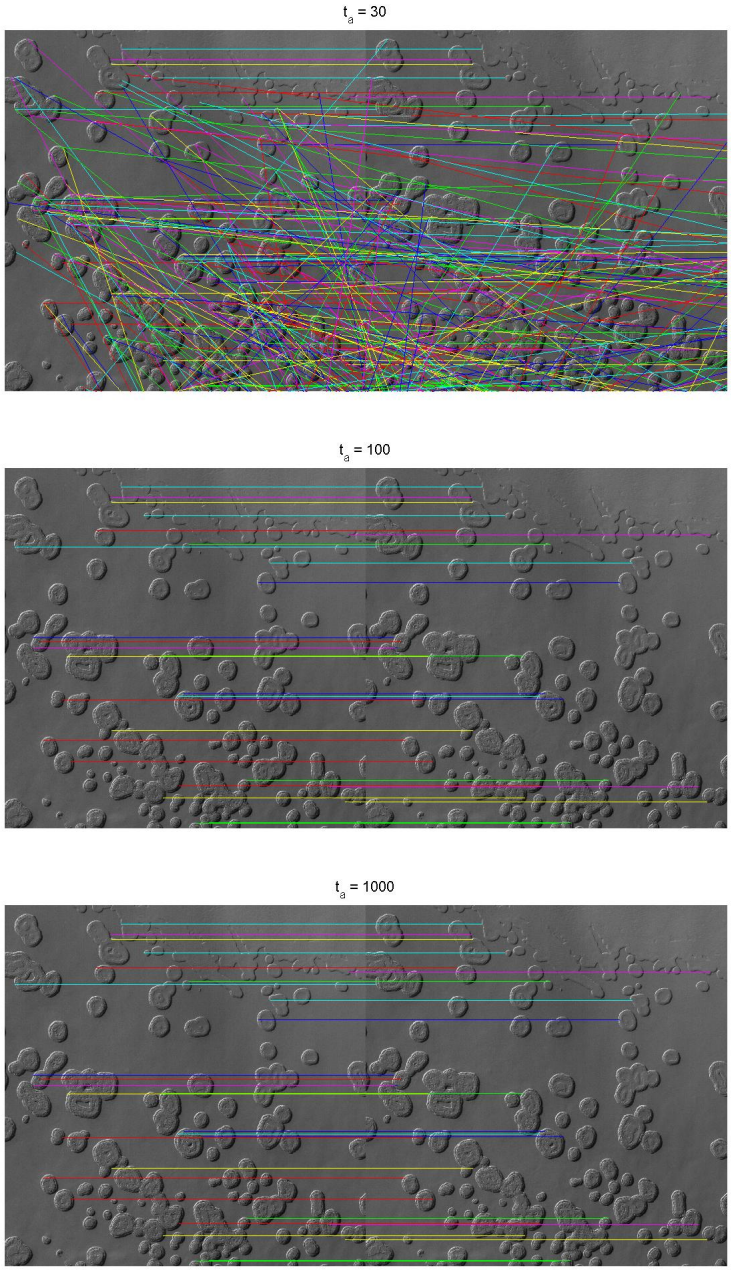

Figure 8: Effect of area threshold on the matching points found by the algorithm in the upper left corner of ESP_012271_0940. From top to bottom t_a = 30, 100 and 1000. Noisy results for $t_a = 30$. No difference between $t_a = 100$ and 1000.

features depends on whether the image represents a dense or a sparse area. Consequently, the values found for this simulation only give a range of possible values for the area threshold, and should be adapted to the configuration.

6.4 SCF threshold

The SCF threshold was set to eliminate the key points that did not correspond to Swiss-cheese features. In the previous simulations, the threshold was set to a low value and many features that were matched by the program did not correspond to Mars circular pits, but rather to edge points. Comparing the matching points with three reference Swiss cheese features should eliminate some of the unwanted key-points. On the other hand, several shapes corresponding to the desired patterns were ignored. Therefore, for this part of the simulations, we would like to avoid the down-sampling performed by SiftGPU to have a better precision. In order to keep a fast running time, a smaller image will be studied. It is the same upper-left corner sub-image that was represented in figure 8. As its size is still too large (5012×5000 pixels), we will divide it into 25 sub-images, using a 5×5 grid.
Once again, we will use the same image for both inputs. The area threshold is set to 1000.
In table 2, we can see the number of matching points similar to the templates found by the program, for a few values of SCF threshold. For a value $t_{SCF} = 40$, the program cannot find any circular shape

t_{SCF}	5	20	25	30	40
Number of matches	697	684	593	482	0

Table 2: Number of matching points of different values of SCF threshold. Simulations were made on 5012×5000 pixels sub-image extracted from ESP_012271_0940 (upper left corner). Area threshold $t_a = 500$.

in our sample. Though, it will probably find the original templates in the full image. Indeed, the tests made with the previous configuration (full size image, 8 by 4 grid), the program found 178 matching points for $t_{SCF} = 40$.
For a threshold value below this limit, the program found hundreds of points that may belong to a Mars pit. In figure 9, we can observe these matching points. First of all, we notice that we have much more

matching points than in figure 9, thanks to the higher precision. We can see that for t_{SCF}=5 or 20, we still have many key points that do not point to a Swiss cheese feature. We can verify that, for this configuration, the program has detected between 20 and 40 matching points for each sub-images. Therefore, the best threshold value should be comprised in that interval.
The result for $t_{SCF} = 20, 25$ and 30 is shown in figure 9. We can see that several matching points will vanish as the threshold increases.

To better visualize the points eliminated, we plotted them in figure 10. Though many of the points eliminated do not correspond to Swiss cheese features, we observe that some correct shapes were also discarded by the program. This shows the limit of the threshold model. If the goal is to detect all the Martian caps in an image, increasing the threshold over 30 will not improve the results of the program, and a more complex strategy should be applied. Increasing the number of the reference images could give better results, but will increase the running time of the algorithm. Alternatively, one could use a single reference and try to track the evolution of a particular pit.

Another limitation of the threshold model, is that the total number of circle-like key-points will depend on the concentration of south-polar caps in the region of the sub-image studied. If the threshold is set too high, then the areas that have a small number of pits will not reach it, and the isolated caps will be discarded.

Now that we have an estimation of the right parameters to use, we can move to the next step, and try to match different images.

6.5 Motion tracking

Due to the complexity of the HiRISE data base, it was difficult to find two images of the same area taken at different times by the satellite. Instead, we are going to manually modify the previously used image and use the result as a second input. Again, to speed up the simulations while working on full resolution, we will take a 5012×5000 sub-image extracted from the upper left corner of ESP_012271_0940 .

As a first test, we will rotate the image with an angle of 90° [5]. As SIFT is rotation invariant (see section 4), it should be able to match

[5] using Kakadu -rotate option

the same features from the original to the rotated image. The results are shown in figure 11. As expected, the program has no difficulties matching the correct points. For a better visibility, a second plot with a smaller set of matches is represented[6].

The next challenge is to try to track a feature that has moved from its place over time. To simulate this case, we will use as a second input a sub-image of the same size and from the same area, but whose position is slightly shifted compared to the previously used sub-image (see figure 12).

The result is shown in figure 13. We observe that the matching is correct except for a few lines at the bottom. In this area the concentration of the south polar pits is so high that it is difficult to distinguish them. Increasing the area threshold here would result in a great loss of correct matching points. This again shows the limitations of simple threshold models, which cannot recognize precisely a circular Martian pit. The program could benefit from a proper pattern recognition component. However, the general behaviour is correct, and using simple considerations we can have a good estimation of the localisation of Swiss cheese feature and of their 2D motion.

[6]using Matlab's function **datasample** `http://www.mathworks.ch/ch/help/stats/datasample.html`

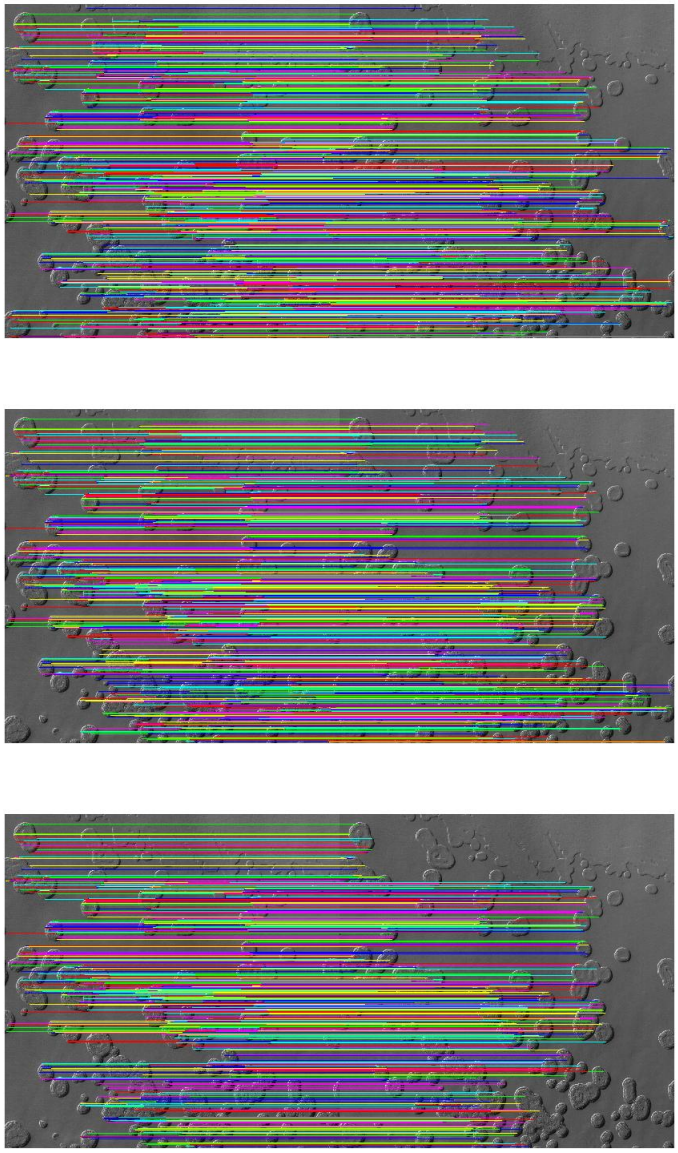

Figure 9: Effect of SCF threshold on the matching points found by the algorithm in the upper left corner of ESP_012271_0940. From top to down t_{SCF} = 20, 25 and 30. Increasing the threshold has eliminated several matching points that did not correspond to circular shapes (see upper region). Some correct shapes were lost

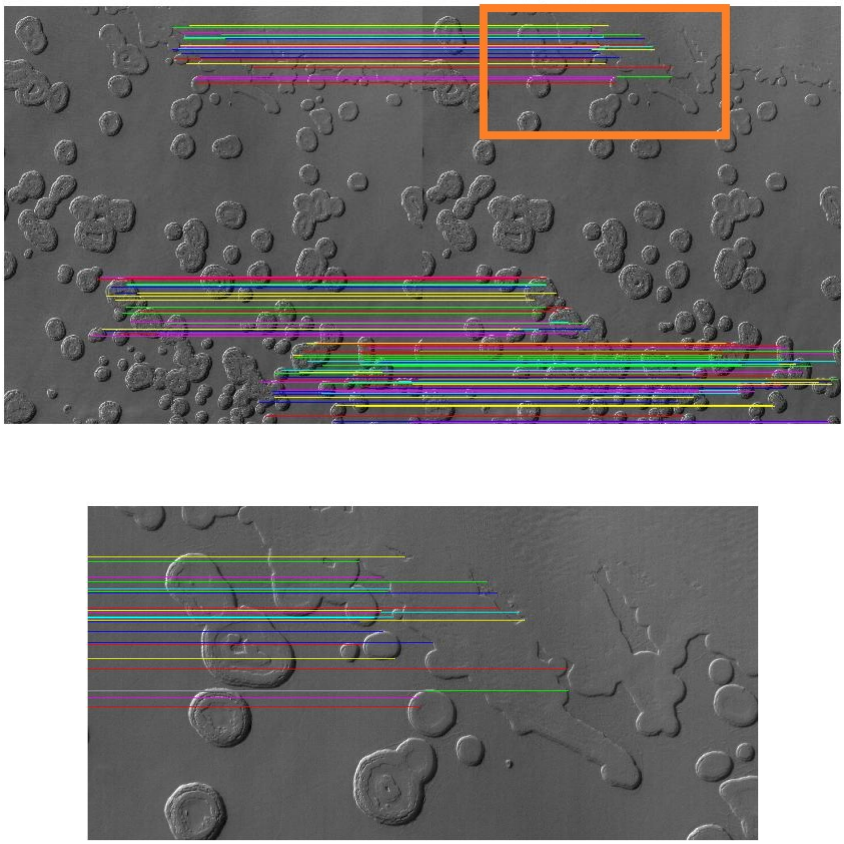

Figure 10: Matching points eliminated when increasing the SCF threshold from 25 to 30 (up), and zoom on the upper region (bottom). Most of the points eliminated did not correspond to a circular pit, but some correct shapes were lost.

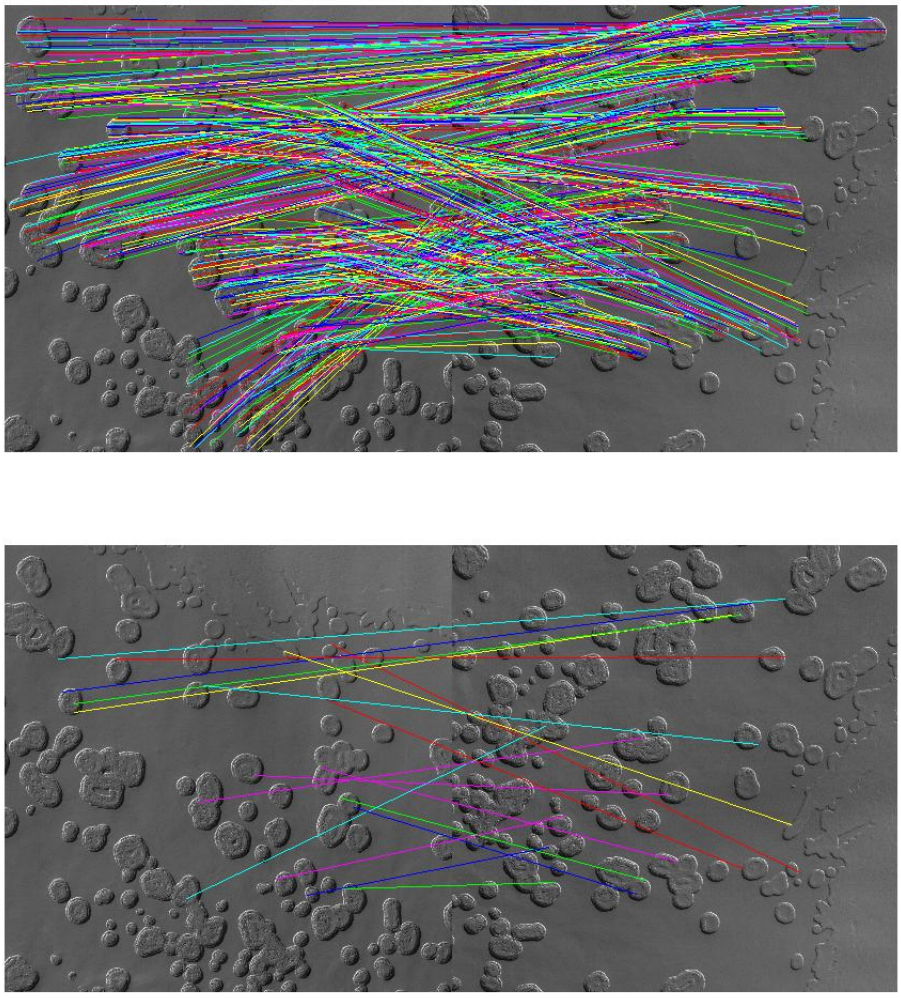

Figure 11: Matching points between upper left corner of ESP_012271_0940, and the same sub-image rotated by 90° (upper). The bottom figure is the same but with a fewer number of matches selected ranomly. Parameters : $t_a = 500$, $t_{SCF} = 30$.

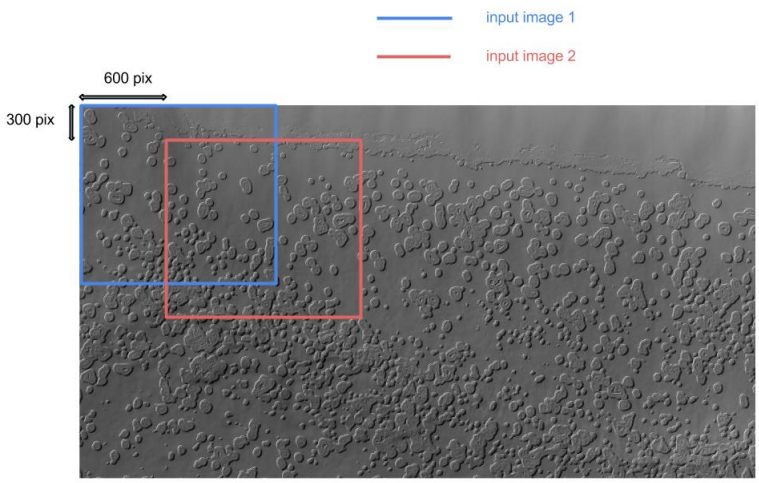

Figure 12: Schematic representation of the two sub-images used as an input for motion tracking. Original image ESP_012271_0940. Size of sub-images 5012×5000 pixels

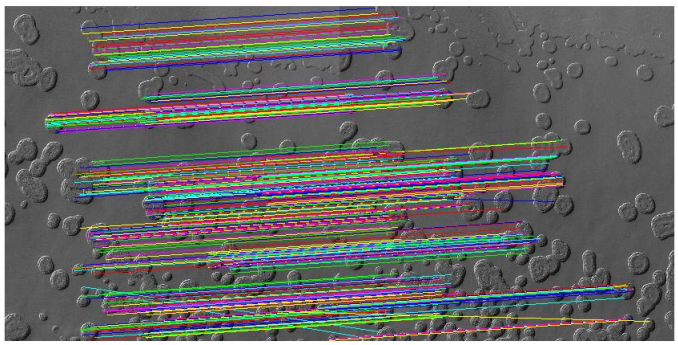

Figure 13: Matching points between two sub-images of ESP_012271_0940. Left starting from $(x_0, y_0) = (0, 0)$. Right starting from $(x_0, y_0) = (600, 300)$. Size of both images 5012×5000. Parameters : $t_a = 1000$, $t_{SCF} = 25$.

7 Conclusion

This project was a good opportunity to overview the methods used in computer vision for feature detection and image registration. Here the performance of the most efficient techniques was tested in the particular context of very large high resolution images, rather than real-time execution.
Many existing software failed to support the specific format and the important size of HiRISE Mars images, but eventually, the adequate tools were found. Kakadu software was used to decompress and manipulate the data, simply and efficiently. Following a recent work on large photogrammetric images, a GPU implentation of SIFT, SiftGPU, was used for the matching and registration of the large images. A program exploiting the capacities of Kakadu and SiftGPU was written to detect and match south polar caps of Mars between two images. It uses simple threshold models to find similar areas and select regions containing Martian pits. The results obtained were satisfactory, as many right matches were found between shifted and rotated images. However, the isolated caps are often ignored by the program; this can only be solved by a more sophisticated pattern recognition procedure.

8 Acknowledgements

I would like to thank the authors of SiftGPU and L^2SIFT for making their software available as open-source, but also for answering my numerous e-mails.

References

[1] Codreanu V. *et al.*, "GPU-ASIFT : a fast fully affine-invariant feature extraction algorithm", High Performance Computing and Simulation (HPCS), 2013 International Conference.

[2] Guoshen Yu, Jean-Michel Morel, "ASIFT: An Algorithm for Fully Affine Invariant Comparison", *SIAM Journal on Imaging Sciences*, 2009.

[3] Li Yang Ku, (2011), "Match Plot", *Matlab Central*. [Online] Available : `http://www.mathworks.com/matlabcentral/fileexchange/31144-match-plot/content/match_plot.m`

[4] Lowe, D.G, "Object recognition from local scale invariant features", *Computer vision*, VOL2, NO60, 2004.

[5] K. Mikolajeczyk and C. Schmid, "A performance evaluation of local descriptors", *IEEE Transactions on pattern analysis and machine intelligence*, VOL. 27, NO. 10, 2005.

[6] M. Nixon, A. Aguado, "Feature extraction and image processing for computer vision", *Oxford Academic Press*, 2012.

[7] J. Odul, (2011) "Utiliser et appliquer l'algorithme SIFT sur une image avec Android", *Developpez.com*. [Online] Available : `http://jodul.developpez.com/tutoriels/android/utiliser-sift/` [22 June 2014]

[8] A. Scott, C. Verba, (2009) "South Pole Residual Cap Features: Swiss Cheese Terrain", *HiRISE* . [Online] Available : `https://hirise.lpl.arizona.edu/ESP_012271_0940` [22 June 2014]

[9] Sun *et al.*, "L^2-SIFT: SIFT feature extraction and matching for large images in large-scale aerial photogrammetry", *ISPRS Journal of Photogrammetry and Remote Sensing*, 2014.

[10] P.C. Thomas *et al.*, "Residual south polar cap of Mars: Stratigraphy, history and implications of recent changes", Icarus 203 (2009) 352-375.

[11] Tong *et al.*, "A GPU-based affine and scale invariant feature transform algorithm", *Journal of information & computational science* 10:13, 2013

[12] T.Tuytelaars and K. Mikolajeczyk, "Local invariant feature detectors : a survey", *Foundations and trends in computer graphics and vision*, VOL. 3, NO. 3, 2007.

[13] VLFeat open-source library, `http://www.vlfeat.org/index.html`

[14] C. Vogel *et al.*, "Optical flow for glacier motion estimation", *22nd ISPRS Congress, Melbourne, Australia 01/2012*

[15] Wu, C., "SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)", 2007. `http://cs.unc.edu/~ccwu/siftgpu/`

[16] D. Yang and W. Zhang, "A parallel analysis on scale invariant feature transform (SIFT) algorithm", *APPT 2011 Proceedings of the 9th international conference on Advanced parallel processing technologies"*, pages 98-111. 2011