Images stitching for panorama reconstruction
Introduction to Computer Vision
Final Project Report

Maxence Gélard
CentraleSupélec
MDS - Mention SDI
maxence.gelard@student-cs.fr

Abstract

Reconstructing panoramas from a set of pictures using images stitching is a classic computer vision applications namely based on the use of features matching from keypoints detectors (SIFT - ORB detectors). This work aim at reproducing a classic pipeline of image stitching that would lead to reconstruction of full panoramas, given separated images of the same landscape. However, through this project, we aimed at having a deeper understanding of the inner operations on such a stitching pipeline. We thus namely asses the performance of the computations of homographies, mandatory to stitch images, through the use of different features detectors and matchers (Brute Force - KDTrees). The performance will be both evaluated from a matching error perspective as well as on computational time. Moreover, we also consider a use case where one would be given a set of images that don't necessarily correspond to the same global panorama. Therefore, we propose a benchmark of two clustering algorithms that use images keypoints and descriptors in order to group images from the same panorama before performing stitching on each cluster. This additional step upstream from the panorama reconstruction pipeline thus allows finally to have an end-to-end images stitching process, from isolated and shuffled images in a dataset to a set of panoramas.

1. Introduction and Motivation

In this project, I aim at tackling the problem of images stitching for panorama reconstruction. Image stitching consists in building, from similar and overlapping images of the same scene, a larger image with a wider view. This is a widely used technique which is implemented in the majority of cameras or cell phones. Automatic image stitching is a very well studied area of computer vision, and several methods have been developed in that direction. The key point in registering images in order to build a panorama is to identify features in the images that will allows to compare them.

Moreover, this project aims at adding a clustering step before applying the images stitching pipeline. Indeed, in a classic setup for panorama reconstruction, one could imagine that the algorithm may be given a set of pictures (say 2 for simplicity) corresponding to the same scene, and the goal would be to stitch one image to the other. We are here interested in reproducing such behavior but also trying to extend it to the case where we are given a set of different images, taken from different scenes. The goal would then to be able to group images that belong to the same scene and then stitch them. A possible application in smartphones could be to automatically be able to get panoramas from pictures in the gallery, without having to manually use the panorama mode that require the user to do it by moving slowly the smartphone while the camera incrementally build the panorama. Instead, from several single images from different viewpoints, the smartphone could propose panorama automatically built with our pipeline. Below is given an illustration of a reconstructed panorama taken from [2]:

Figure 1. Image stitching result

2. Problem definition

2.1. General image stitching problem and algorithm

Let's formalise a bit more the problem of image stitching, by taking the example of 2 images (the case of $n \geq 2$ images will be handled afterwards). We are given two images, represented as matrices X_1 and X_2. We thus aim at finding a transformation H, such that the concatenate image $[X_1, H(X_2)]$ would represent a panorama of the scene, built from the two parts X_1 and X_2. The need for a transformation H, that we'll see will be given by a homographic transform, is to take into account the different "shooting angle" of the two images.

Below is given the general algorithm in pseudo-code for panorama reconstruction. Such pipeline will be described in more details in the related work part, but is meant here to introduce formally the different steps that we will be reproducing / evaluating or improving in the rest of the report:

Algorithm 1: Panorama reconstruction pipeline

Data: Images: X_1, X_2
Result: Panorama: X
$keypoints, descriptors \leftarrow detect(X_1, X_2)$;
$H \leftarrow getTransform(keypoints, descriptors)$;
$X_1^{'}, X_2^{'} \leftarrow stitch(X_1, X_2, H)$;
return $X = [X_1, X_2]$

The algorithm above thus allows to formalize a list of 3 main steps that will be discussed in order to perform image stitching:

- From input images, **compute features** (their name *keypoints* and *descriptors* will be explained for the different algorithm we chose.
- By using a **features matching algorithm**, we are able to construct a **homography** matrix H that estimate the transformation that we need to apply either to X_1 and X_2 (this choice will be a discussion point in the Methodology part), to perform the reconstruction
- We transform the images (**stitching**) using H and by some post-processing (concatenation, cropping ...), we end up with the final panorama.

2.2. Images clustering problem

We will now formalize the problem of image clustering in a very general way, the precise algorithms which will be used will be described in the Methodology part. For this problem, we are given a set $S = \{X_1, X_2, ..., X_n\}$ images, that come from different scenes (but multiple images are from the same scenes). The goal is thus to build a clustering algorithm that will return a set of label $l \in [0, n_{clusters}]^n$, where $n_clusters$ might be a parameters that need to be tuned.
Once the images clustering had been performed, we will end up with group of images (=images that share the same clustering label), and we will be applying the image stitching algorithm to all the images on the group. For cluster composed of 2 images, *Algorithm 1* can be directly applied. For more than 2 images, we recursively apply *Algorithm 1* to either the two initial images, or a new image and a partially reconstructed panorama (obtained by a previous iteration of the algorithm).

3. Related work

In this section, we will present the main works that lead to image stitching algorithm. These work have in fact introduced the general pipeline described in *Algorithm 1*, by using specific algorithm for each of the steps, and the goal of this section is to give an overview of these techniques.

Let's starts with the features detection. Two main families of features are being uses: direct features [7], where all the pixels are used, and local features [3] that allow to reduce the number of considered variables. A very popular category of features is the family of transformation invariant features, with especially the well known *SIFT* detectors, proposed by D.G. Lowe [5]. Let's detail how this algorithm lead to a list of keypoints and descriptors that will be used in order to match images.

SIFT constructs a scale space of a given image, and for each octave, we proceed to a Gaussian blurrings (5 for each octave generally) of each scale. We then create a set of DoG (Difference of Gaussian) for each octave by subtracting successive blurred version of images in the same octave. This is illustrated below (picture taken from the original paper [5]:

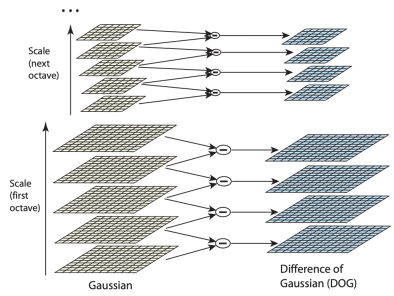
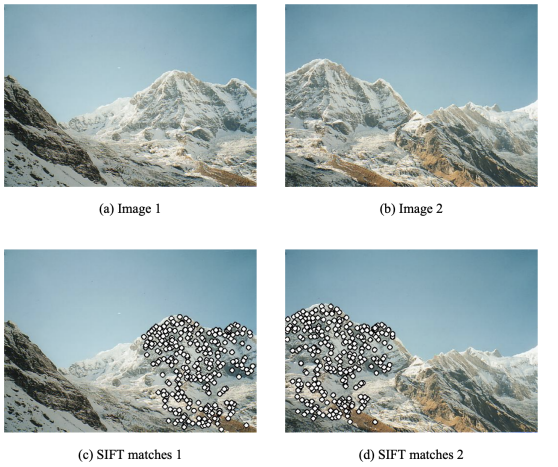


Figure 2. SIFT - illustration of the octaves

Then keypoints are extracted by identifying the local maxima and minima of these DoG (and a keypoints selection is also done by removing low contrast ones). Then using histograms of magnitudes and orientations of keypoints neighbors, we are able to build the descriptors which are "fingerprints" that describe each point.

Given a set of features for a pair of images, the second step is then to set up a matching algorithm, followed by a transformation of the images in order to align them. The *RANSAC* algorithm has namely been proposed to this end in [2], that builds the homography transforms to perform the image stitching. The idea is, for two images X_1 and X_2, a sufficient number of matching descriptors (that we got using SIFT), in order to build a transformation matrix which will map one set one the descriptors to represent them in the coordinate system of the second image.

An illustration is given of features detection on a pair of images (from [3]), in Figure 1.



(a) Image 1 (b) Image 2

(c) SIFT matches 1 (d) SIFT matches 2

Figure 3. Images matching result

4. Methodology

This section will describe in more details the methods and algorithm that have been used. It is divided in two subparts:

- 1) **Images stitching**: details on the features detection, matching and homographic transformation.
- 2) **Images clustering**: used features and algorithms

The image stitching pipeline that will be described here is limited to 2-dimensional images, and would need to be adapted, namely the geometric transformation part, in order to be applicable to 3-dimensional images.

4.1. Images stitching

4.1.1 Features detection and matching

For the features detection part, I have been using two different detectors in order to retrieve keypoints and detectors: SIFT and ORB (*Oriented FAST and Rotated BRIEF*)[6] detectors. The details of SIFT detector can already be found in the Related Word section. Let's give some details on the ORB detector. ORB combines two algorithm, one for the location of keypoints (FAST algorithm), and one to build the descriptors (BRIEF). The main claim of ORB is to perform as well as SIFT while being two orders of magnitude faster. Here is the brief overview of the algorithm [1]

- First for the keypoints detection, ORB uses FAST (*Features from Accelerated and Segments Test*), that find keypoints as pixels that have a brightness significantly different than its surroundings. To make this detection scale invariant, ORB uses the same trick as SIFT, i.e build a pyramid scale space of the images and locale keypoints at these different scales ; and a orientation is already given as in SIFT, but here at a patch level.
- Then, ORB uses BRIEF(*Binary robust independent elementary feature*) to build the descriptors. For each keypoints, BRIEF is using a patch neighborhood and build binary features vectors by selecting pairs of pixels in the patch and comparing their intensities.

Below is given an example of the keypoints / descriptors detection (here using SIFT detector):

Figure 4. Features detection - SIFT

The next step is then, given a pair of images' keypoints and descriptors, to be able to match them, i.e identify similar keypoints between the two different images. The idea is, given two list of descriptors, to compute the distance between each descriptors' pair (descriptors being vectors in high dimension, typically 128 for SIFT detectors). Multiple parameters can be considered:

- Choice of the distance metric: typically L_1 or L_2 norm are used (i.e $distance(d_1, d_2) = ||d_1 - d_2||_1$ or

[1] More details on this blog-post: https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

$distance(d_1, d_2) = ||d_1 - d_2||_2)$. Another commonly used distance when using ORB descriptors is the Hamming distance, which count the number of non-zeros elements after having applied a XOR operations between two descriptors.

- Choice of the matching algorithm: either a brute force algorithm, which will compute all distance pairs, or a KD-Tree based algorithm.

Once we are given a list of match, i.e of corresponding keypoints in the two images, we can consider two ways of using them:

- We can order the match by distance (using the distance that has been used during the matching, i.e L_1, L_2 or Hamming), and only keep the K matches than have the lowest distance (typically we chose $K = 50$).

- While this technique may sound logical if using the brutforce algorithm, when using a KD-Tree based method, another possible method would be, for each descriptors, to find its k-nearest-neighbors (typically $k = 2$). Then, we need to define which are the best matches. Previously, when using brutforce algorithm, best matches were the matches with the lowest distances. Here, we will be using a trick from Lowe (author of the SIFT paper) that consider that a good match (here composed of 2 candidates as we are using 2-nearest-neighbors) is thus that $distance_1 < ratio * distance_2$, where $distance_1$ is the distance for the first match and $distance_2$ is the distance for the second match ; and the $ratio$ coefficient (typically $ratio \in [0.6, 0.8]$) is to be tuned.

Below are given examples of features matching :

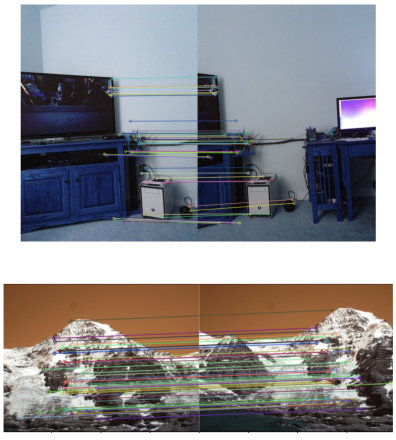


Figure 5. Features matching - SIFT + bruteforce algorithm (L1 distance)

4.2. Homography estimation

Given matching descriptors between two images, we aimed at finding a 2D transformations to map points from the first image to the second one. This transformation is called a homography, and reflect the fact that the two images are taken from the same scene but with different rotation of the camera. More precisely, given two images X_1 and X_2, and matching keypoints (repsectively $keypoints_1$ and $keypoints_2$), we want to find a homographic transformation to map $keypoints_1$ to the coordinates system of $keypoints_2$, i.e we want to find a 3×3 matrix H:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Such that, by noting $(x_2, y_2, 1)$ the initial coordinates of a pixel of X_2 (using homogeneous coordinates system), we want to find the new coordinates of this pixels $(x_2^{'}, y_2^{'}, 1)$, which will be given by:

$$\begin{pmatrix} x_2^{'} \\ y_2^{'} \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix}$$

In order to uniquely determine such transformation, one needs to have a minimum of 4 points, i.e for two images that we want to stitch, we need to have at least 4 matching keypoints. For a panorama reconstruction, the rotation of the camera is small enough between images to get a relatively high number of matching keypoints. However, if the two images have a totally different orientation, for a fix value of $ratio$ (the one used in the 2-nearest-neighbors matching), we could get just a few matching keypoints. This difficult case will be explored in more details in the evaluation part of this report.

With 4 random points pairs, one could determine the necessary transformations, however this wouldn't be robust to noise. Therefore, an algorithm called *RANSAC* can be used to find a more robust transformation. For this algorithm, we need to define "inliners" as points that are consistent with the homography. These points will be considered as inliners if the sum of squared difference (*SSD*) between the target points and the transformed points, is less than a parameters ϵ. The RANSAC algorithm thus can be described as follows:

- Select 4 random pairs of matching keypoints (i.e matching points from $keypoints_1$ and $keypoints_2$.

- These 4 pairs uniquely determine a homography H.

- Find the inliners points as defined above.

- Repeat the first step for a given number of iterations by adding to the new 4 random points the inliners points,

and keep the transformation that minimizes the number of outliers.

So we are now provided a transformation H which will be applied to the image X_2. And thus allow us to stitch the image together and finally obtain the final panorama! Below is given the result of the images stitching for the examples that were in Figure 5:

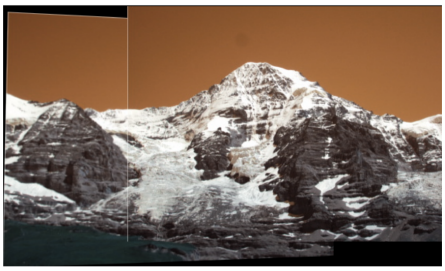


Figure 6. Panorama results

4.2.1 Some "limitation" and proposed solutions

In this section, I would like to mention to small issues that I have encountered and the simple solutions that I added to the image stitching pipeline in order to improve the results.

Firstly, even if it is not said explicitly, this image stitching pipeline assume that there is a "target image" and a "candidate image" that will be transformed through the homography H. Therefore, a choice had to be made, by saying that the image that will go on the left side of the panorama is the target and the one on the right is the candidate. However, if we recall the final aim of the project, we aim at being able to be given any pairs or n-tuples of images in any order (we don't know in advance which one is the left one or the right one). Therefore, I wanted to build a function that can take as input images pair (X_1, X_2) or (X_2, X_1), and allows return the correct panorama. To this end, I have developped a small heuristic that look at the keypoints x-coordinates of the two images, and saying that the left image is the one whose keypoints mean x-coordinate is the greatest. That leads to the following algorithm:

Algorithm 2: Selection of the left and right images

Data: Images: $keypoints_1$, $keypoints_2$, X_1, X_2
Result: Panorama: X_{left}, X_{right}
$keypoints_{1,x} \leftarrow$keypoints$_1[:, 0]$;
$keypoints_{2,x} \leftarrow$keypoints$_2[:, 0]$;
if *mean*$(keypoints_{1,x}) \geq$ *mean*$(keypoints_{2,x})$ **then**
  $X_{left}, X_{right} = X_1, X_2$;
else
  $X_{left}, X_{right} = X_2, X_1$;
end
return X_{left}, X_{right}

The second issue, which is more related to the aesthetic of the output relies on the fact that after the homographic transform and the images stitching, one could get a result with black strips on top and bottom of the resulting panoramas. To solve this issue, I added a post-processing step that consists in removing the rows of the image that contains a too large proportion of black pixels, this proportion being tunable.

4.3. Images clustering

In this subsection, we will discuss the details of the clustering part of the pipeline, which is, as mentioned in the introduction, meant to be run before the main images stitching pipeline.

For this clustering pipeline, I have developed two approaches, which are described below

4.3.1 K-Means algorithm

The first technique that I have been using is a K-Means algorithm. It takes as inputs the concatenation of all the descriptors of all my corpus of images $S = \{X_1, X_2, ..., X_n\}$ and build clusters based on these vectors. Then to perform clusters assignments, for every image, I look to which cluster each of its descriptors has been assigned to. Then, the cluster that is assigned to the image is the one that contain the maximum number of descriptors of the image. This algorithm, though simple, needs as hyperparameters the number of clusters we want to build, which is a huge limitation to our usecase because in a photos gallery, we don't know how many panoramas may be built with the images corpus. One could imagine some trick using for example photos localization that are provided by smartphones to determine an approximate number of different places, but that would just be a very approximate heuristic.

4.3.2 Affinity Propagation

In order to solve the problem of finding the number of clusters, I proposed another approach which uses Affinity Prop-

agation [4] algorithm.

This algorithm takes as input not directly the descriptors but a similarity matrix (details below) between the features. The idea of the algorithm is then to find "exemplars" points, i.e points that are the most likely to represent a given set of other datapoints. From the similarity matrix, the affinity propagation will iteratively compute two matrices, the responsibility and the availability matrix. The first one quantifies how well a given points can be an exemplar poit for the other, while the second one quantifies how much a given point can be represented by each other points. At the end of the algorithm (for example after a given number of iterations), we get the points that are the most appropriate to be exemplars points, and assign to their clusters the points that are the most likely to be represented by this point.

In our use-case, the similarity matrix is defined using the same 2-nearest-neighbors algorithm defined previously. To be more precise, for two images X_i and X_j, and their respective descriptors, $descriptors_i$ and $descriptors_j$, we count the number of good matches points, i.e matches thus that $distance_1 < ratio * distance_2$, where $distance_1$ is the distance for the first match and $distance_2$ is the distance for the second match (the distance being the one used for the matching, like L_1, L_2, ...). So the entry (i, j) of the similarity matrix will be this number of good matches, and we normalize each row of the matrix by the sum of the row' values. Finally, we make the matrix symmetric so that it does correspond to a similarity matrix.

4.4. Putting clustering and stitching together

We thus end up with the final complete pipeline, that combine the clustering algorithm and the panorama reconstruction, which is described in the following algorithm.

Algorithm 3: Clustering and panoramas reconsturction

Data: Images set: $S = \{X_1, ..., X_n\}$
Result: Panoramas: $P = \{P_1, ..., P_m\}$
$P \leftarrow []$;
$clustering.fit(S)$;
$clustersLabels \leftarrow clustering.labels$;
for *label in unique(clustersLabels)* **do**
    clusterImages = groupImages(S, label);
    P.append(imageSnitching(clusterImages));
end
return P;

5. Evaluation

In this section, we will discussed the performance of the two different pipelines. We namely aims at evaluating the accuracy of the homographic transform as well as the quality of the clustering algorithms.

5.1. Images stitching evaluation

In order to evaluate the images stitching pipeline, I have been using the Extreme View Dataset (EVD). EVD is a set of 15 image pairs with ground truth homographies (i.e we are given the 3×3 matrix that has been applied to the reference image descriptors to get the descriptors of the second image of the pair). The idea is thus to apply our image stitching pipeline to each pairs of images up to the computation of the homography (the last step of stitching the images together being just the application of the homography). Below is represented some pairs of the EVD datasets:

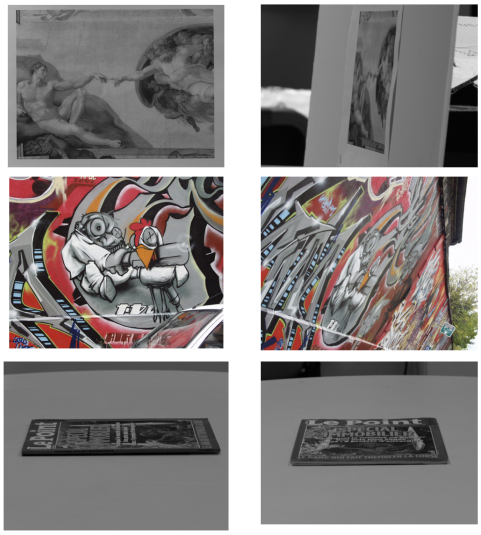


Figure 7. EVD samples

To evaluate our pipeline, we applied our features matching algorithm using the 2-nearest neighbors trick, and computed the number of inliners, i.e matching descriptors that verify a criterion of the form $distance < threshold$.

Below is given an example of features matching result on one of the images pair of the dataset:

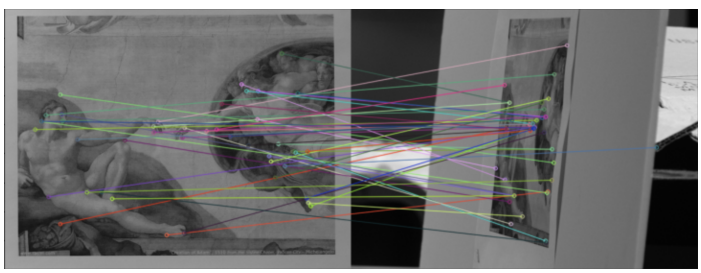


Figure 8. Features matching on one EVD sample

We have been using two metric: the relative number of inliners (with respect to the total number of good descriptors matches), and the time it takes for each image to estimate the homography matrix. A descriptors match is consider to form an "inliner" if, once transformed using the estimated homography, the keypoint location in the image is not far from a given distance from the location of the corresponding keypoint in the second image (this distance has been set to 2 pixels in this experiment). We evaluated these metrics for the two considered detectors - SIFT and ORB - as well as for different descriptors matcher (L_1 brute force and *KDTree* for SIFT ; L_1 brute force and Hamming distance brute force for ORB).

Stitching Result		
Algorithm	% of inliners	Time
SIFT (L_1)	26.86	0.58
SIFT (KDTree)	24.75	0.39
ORB (L_1)	**27.31**	0.39
ORB (Hamming)	27.25	**0.27**

Thus we can see a slight improvement by using ORB instead of SIFT in terms of % of inliners we get (i.e the homography is better estimated), and also a small acceleration by using KD-Tree when comparing L_1 brute force / *KDTree* for SIFT detector. However, due to the fact that *KDTree* methods don't test all possible pairs of matches (as a brute force algorithm would do), we have a small decrease in performance.

5.2. Images clustering evaluation

In this subsection, we will now evaluate the performance of the clustering pipeline, and compare the KMeans and the Affinity Propagation approaches. To this end, I have been using a set of 15 images, that represent a total of 4 different scenes. I have fitted the two different algorithm and evaluated them with the normalized mutual information score (NMI) [1], which is a score between 0 and 1 that is a measure of the similarity between two labels of the same data (so that allows to compare the true cluster labels and the predicted cluster labels). Notice that for KMeans, a number of clusters had to be chosen, so I decided to use 4 clusters as I had 4 scenes, but once again, it the application we are aiming, we do not have access to this information *a priori*. Below are given the NMI scores for the two algorithms (with also other number of clusters for KMeans):

Clustering Results (Metric: NMI)	
Algorithm	NMI score
KMeans (2 clusters)	0.52
KMeans (3 clusters)	0.71
KMeans (4 clusters)	0.66
KMeans (5 clusters)	**0.80**
KMeans (6 clusters)	0.52
KMeans (7 clusters)	0.65
KMeans (8 clusters)	0.55
Affinity Propagation	**0.92**

First, one can see that KMeans has a large variance in terms of NMI scores depending on the number of clusters that we choose, making it a not very robust algorithm. On the other hand, one can observe that we get an improvement by using the affinity propagation algorithm compared to the best KMeans results (increase of 0.12 for the NMI score). Therefore, for our use-case one should select this algorithm as it has better performance but above all because it doesn't require to set manually the number of clusters. Below is given the result of the reconstruction of a panorama from a cluster built using Affinity Propagation, that has 3 images (on first row one can see the three separated images, and on the second row the result of the panorama reconstruction as well as the post processing to remove black strips):

Figure 9. Clustering and panorama reconstruction pipeline

6. Conclusion

To conclude, we were able to reproduce the standard panorama reconstruction pipeline that has namely been developed thanks to the scale and rotation invariant features detector like SIFT. We extended this pipeline by considering another possible features detector (ORB), and carried out several experiments, namely on the descriptors match-

ing algorithm that we have been using. We thus concluded that the ORB detectors slightly performs better than SIFT detector, though provides very close results. On the other hand, using a *KDTree* based methods instead of a brute-force method lead to small acceleration on the homography estimation pipeline. Concerning the clustering pipeline, we proposed two approaches, one using descriptors vectors embedded in a KMeans algorithms, while the second uses a similarity matrix of the images' descriptors, that we fitted in an Affinity propagation algorithm. We highlighted in the evaluation part the massive improvement we got by using the latter algorithm. Therefore, through this project, we proposed an end-to-end pipeline that take as input a set of pictures from different scenes, and is able to group them and reconstruct accurate panoramas. One could think of possible improvements that may be the next steps of the project. For example, one can notice that there are often visible separations lines on the reconstructed panoramas. This is namely due to the fact that for most use case, we reconstructed panoramas from 3 images, that were taken from relatively very different angles, thus making the homography estimations harder. A possible solution would be to have more images, with smaller relative angle, which would results in solving more but easier stitching problems. Such datasets may not be always available, or such pipeline requiring more computational resources, one could move this project forward and investigate solution to solve the issue (preprocessing of the image - blur / smooth the borders ; heuristic to change ht homography and get more overlaps etc.).

References

[1] Normalized mutual score - sklearn. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html`. Accessed: 2022-01-14.

[2] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[3] M. Brown, D. G. Lowe, et al. Recognising panoramas. In *ICCV*, volume 3, page 1218, 2003.

[4] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[5] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[7] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 2(1):1–104, 2006.