Ukrainian Catholic University

Faculty of Applied Sciences

IT & Business Analytics Programme

Image Stitching

Linear Algebra Project

Authors:
Alina Voronina
Viktoriia Maksymiuk
Yuliia Maksymiuk

29 May 2022

1 Introduction and problem setting

The Image Stitching project is a work in the field of computer vision. Image stitching algorithms are widely used in various spheres: panorama photography, space imaging, and satellite image data processing. The primary purpose is to represent several images as a single one, solving the problem of limited resolution and frame dimensions. The obtained "stitched" image is easier to process and is more informative than tens or hundreds of smaller shots. Therefore, the development of such an algorithm is essential for computer vision. In addition, it is interesting to try to create one's own image stitching algorithm from scratch using knowledge of linear algebra.

The aim is to implement an algorithm that composes multiple overlapping images captured from different viewing positions into one natural-looking panorama, using linear algebra principles and algorithms. It will help obtain a full-range view in a single photo, excluding the necessity of manual stitching in photo editors. The final goal is to create a program that receives some images of an object/surface/scene as an input and outputs a single picture of that.

Why would one want to use the image stitching algorithm? Imagine visiting the Burj Khalifa - the highest building in the world. With the regular camera or the smartphone, one will not be able to capture the tower in detail, maintaining a high resolution and pretty detailed view. One would most probably use a panorama mode or try to photoshop several images into a single one to post on social media to get likes from friends. Cropped Burj Khalifa will not look as impressive on the Instagram page **:)** Another situation is taking pictures with a large group of people. If one is lucky enough to have friends, they most likely had a situation where they wanted to take a picture of them all. In this case, no one wants to be cropped or left outside of the frame.

As photoshop tools may appear complicated to use for a regular person, the image stitching program will save them. Using Linear Algebra techniques and existing libraries, the project's algorithm will solve the problem of several images composition very precisely.

2 Work review

There are three main steps in implementing the image stitching algorithm. Many approaches are proposed to perform each of them, every one having its pros and cons.

A Keypoint detection and local invariant descriptors extraction

To be stitched together, the images need to have some common elements. The view of these elements may differ, depending on the angle, scaling, rotation, or spatial position. The most straightforward technique to use is the algorithm called **Harris Corners**. It directly takes the corner score's differential into account concerning direction [1], and it happens to be invariant to rotation. Nevertheless, the input images may differ not only by rotation angle but also by scale. The Harris Corner detector (being not scale-invariant) will not work in such a case, which is a clear disadvantage.

Among the other techniques that may be used in keypoint detection are the following: scale-invariant features transform (**SIFT**), speeded up robust features (**SURF**), **Oriented fast**, and rotated BRIEF (**ORB**), and so on. SURF is a little similar to SIFT, and it is slightly quicker than the latter one. However, SURF is not stable to rotation, and it does not work correctly with illumination [2]. In this project, the classic SIFT will be used, as it is a more reliable method used in many works. This technique is both rotation and scale-invariant, although it happens to be slightly slower than the alternatives.

B Feature matching

After obtaining features and descriptors for the images, the next required step is to establish preliminary matches between these photos. For this task, two matching methods can be considered: **Brute Force Matcher** and **K-Nearest Neighbors** (KNN). Brute Force calculates the distance between two points using *Euclidean distance*. Thus, for each descriptor in image 1, it returns the nearest descriptor (*single best match*) in image 2 and vice versa. However, this is not very efficient because there will be *many false positives*, resulting in the homography matrix *not being correct*.

Contrastingly, KNN (*K-Nearest Neighbors*) is useful when considering more than one candidate match, as this method returns *K best matches* for a given descriptor. One of the main advantages of KNN over Brute-Force is the possibility of manipulating the matches obtained by implementing a *ratio test*. It is useful when a correct match has a larger distance than an incorrect one or when many similar descriptors of repetitive features in these images are present.

Nevertheless, the KNN-matching method can be very effective, but it requires *storing* the whole training set; therefore, it may use much memory and be relatively *slow*. Another weakness is selecting the *optimal K value* to achieve the maximum accuracy of the model, which is challenging. However, K=2 is optimal for image stitching when *performing David Lowe's ratio test*.

C Homography estimation with RANSAC algorithm

The most exciting part of creating panoramas is that the two images are not that of a plane but a 3D scene when a person takes them by rotating the camera about its optical axis. To align images of a panorama, at the heart of this technique is a simple invertible 3×3 matrix called **Homography**.

An alternative to using homographs or 3D motions to align images is first to warp the images into **cylindrical coordinates** and then use a pure translational model to align them [3]. Unfortunately, this only works if the camera takes all images at the *same level or with a known inclination angle*.

Pictures can also be projected on a **spherical surface**, which is helpful if the final

panorama *involves an entire sphere or hemisphere view*, not just a cylindrical strip [4]. Algorithms for merging cylindrical images are often used when the camera is known to be level and rotates only around the vertical axis. In such conditions, a **purely horizontal translation** connects images with different rotations. Since homography is **very sensitive** to the quality of data we pass to it, there is a need for an algorithm to filter irrelevant points from the distribution. **RANSAC** is the best-suited candidate for that job.

The most famous RANSAC alternatives are **Least Squares Fit** and **Hough transform**. Both of these procedures are similar to RANdom SAmple Consensus algorithm since they are robust to noise. However, the Least Squares Fit is a simple closed-form solution in addition to the not resistance to outliers.

The Hough transform can fit multiple models; however, its weakness relates to the works for a few parameters (typically 1-4). A RANdom SAmple Consensus algorithm works with a moderate number of parameters (1-8). Even though both of the techniques are robust to outliers and noise, RANSAC is still the best technique to use while working with estimating a homography.

However, there are **drawbacks** too. First of all, when the number of measurements is quite large, RANSAC works slower. That is why there are modifications of RANSAC, which can significantly speed up its performance (*Preemptive RANSAC*). Secondly, in another variant on RANSAC called PROSAC (*PROgressive SAmple Consensus*), random samples are initially added from the most "confident" matches, thereby speeding up the process of finding a (statistically) likely good set of inliers. There are a lot of improved algorithms that can are more useful in end-to-end training of feature detection and matching pipelines. Nevertheless, many state-of-the-art pipelines still rely on methods that **stood the test of time**, such as SIFT or RANSAC, which the authors chose for the project.

3 Methodology and theoretical part behind it

For the image stitching algorithm, several Linear Algebra notions and methods were used. Among them being vectors of keypoints and descriptors, the distance between vectors for the KNN algorithm, and homography matrices for the RANSAC algorithm.

A Keypoint detection and local invariant descriptors extraction

There are four core steps for SIFT algorithms: scale-space extrema detection, keypoint localization, orientation assignment, and keypoint description. These algorithms will allow obtaining the common features on each image so that the images can be stitched according to these features.

Firstly, the features that may be potential keypoints must be discovered. It is essen-

tial that they are not just noise and are not scale-dependent. There are several blurring techniques that can be used to denoise the images, one of them being the **Gaussian Blurring**. After that, the algorithm that defines the scale-invariant keypoints is implemented, and the image is left with a set of best-suited keypoints. Scale-space is defined as a set of images with different scales obtained from the original image. The image must be scaled and blurred several times - those steps are called octaves. The ideal algorithm is to scale the image 4 times and then blur each scaled image 5 times[5].

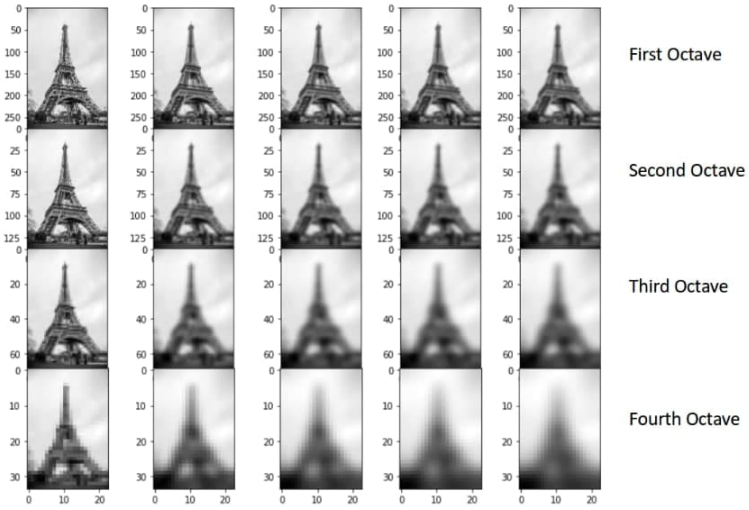

Figure 1. *Scaling and blurring before keypoint detection*

The next step is **Difference of Gaussian**, used for enhancing the features. The algorithm generates a set of images, where each image is a difference between a particular image and its more blurred version.

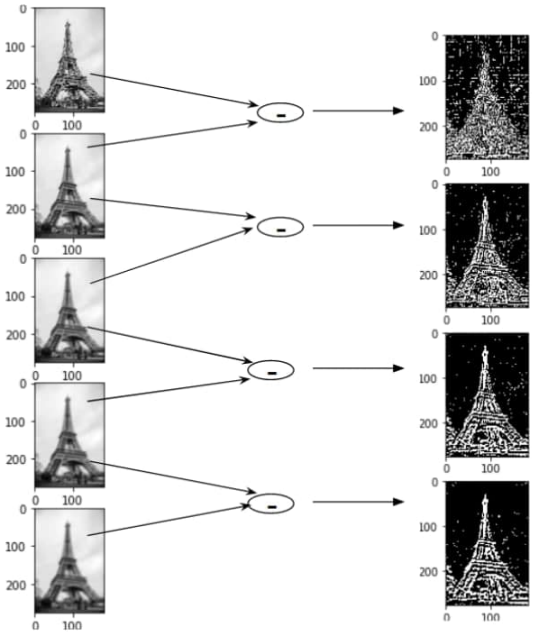

Figure 2. *Example of DoG performance*

The new set of images is used in the following steps.

Later the images are **searched for local minima and maxima over scale and space**. For instance, 1 pixel in an image is compared with its eight neighbors, and 9 pixels in the next scale and 9 pixels in previous scales [6]. Local extremas are potential keypoints that are also filtered for relevance by comparing their intensity with the chosen threshold (lower limit, intensities under which are defined as low and not suitable for keypoints; therefore, keypoints with such intensities are not chosen to be processed on the next steps). Keypoints also must not be located at the edges of the image, as they may vanish at the second image one wants to stitch it to. The result of this stage is a set of good keypoints.

Orientation assignment step speaks for itself. The keypoints obtained in the previous step are assigned an orientation so that they are rotation-invariant.

The last stage for SIFT algorithm is **keypoint description**. In this step, the descriptors of the keypoints are generated, taking into consideration the latter is scale-invariant and rotation-invariant. For each keypoint the neighbouring points are taken, which *describe* the keypoint itself. This stage is done using gradients.

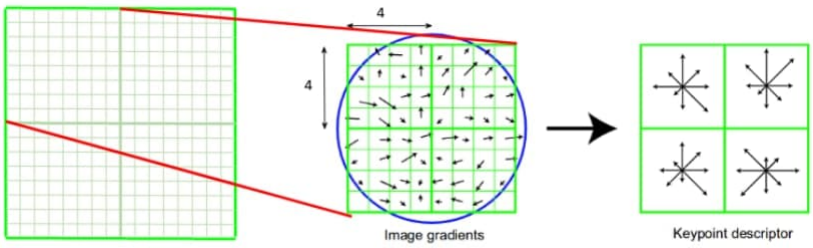

Figure 3. *Keypoint descriptors generation*

The result of the overall SIFT algorithm is a set of keypoints and a set of their descriptors for a particular image.

B Feature matching

After applying SIFT algorithm to extract keypoints and their descriptors for each image, the KNN matching procedure is used to find k closest matches *for each feature* in each image. To keep only the *strong* matches, **David Lowe's ratio test** is also implemented to increase the robustness of the SIFT algorithm and *eliminate false matches*. The general idea behind this test is that there needs to be a sufficient difference in distance between the first-best and the second-best candidates to match for the keypoint to be preserved. In other words, if both features are similarly close to the keypoint, then there is *no best match*, and the keypoint is eliminated from further calculations.[7] This method compares the ratio of two nearest neighbors with some fixed threshold, so to accept the first-best match as a valid match for the initial keypoint, the ratio should be smaller than a threshold value. An optimal threshold value to compare distance ratio with is 0.8, which helps *eliminate* 90% *of the false matches* while discarding *less than* 5% of the correct matches[8].

Nearest Neighbor Distance Ratio *(NDDR)* or ratio test $= \frac{d_1}{d_2}$
where d_1, d_2 - 1st and 2nd nearest neighbor distances respectively.

To exclude some possibly incorrect matches, the **Crosscheck** technique is introduced. This approach validates obtained matches by *repeating the same procedure a second time*, but in reverse order, finding the best match from the first image for each keypoint of the second image. Valid matches are only those that produce the *same pair of keypoints* in both directions (thus, such keypoints are the best matches for each other), which means that the CrossCheck approach makes the results *invariant* to image order and *more accurate*.

Implementing the K-Nearest Neighbours Matcher is relatively simple because the central concept of LA underlying it is the **Euclidean distance**. For each descriptor from image1, one finds K nearest neighbors from image2 after calculating the Euclidean distance between two points in the Euclidean space.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

The *optimal value of K* for this algorithm is 2 to keep the two best descriptors for each descriptor on both images to perform Lowe's ratio test and subsequent Crosscheck validation to *eliminate some possible incorrect matches*. After applying such feature matching methods, the resulting matches will be *accurate enough* for further calculation and obtaining a stitched image.

C Homography estimation with RANSAC algorithm

The last part of the project consists of **distorting one image** to have the *same perspective* as the other one and then stitching the two photos together. To change the perspective of the other image, the **homography transformation**, described in the previous and this sections, is used. Stitching is done based on selecting pairs of features in different images so that each attribute in that pair corresponds to the exact locations in both pictures. After that, the optimal homography matrix is calculated for projecting one image into the same projection as the other one. As a result, the features in both images are on top of each other.

A homography can be computed when there are **4 or more corresponding points** in two images. Automatic feature matching does not always produce 100% accurate matches. It is not uncommon for **20-30%** of the matches to be incorrect after applying SIFT and KNN techniques. Fortunately, a reliable estimation technique called RANSAC gives the correct result even when there are a lot of bad matches.

The abbreviation stands for **RANdom SAmple Consensus**, an algorithm proposed

in **1981** to estimate the parameters of a particular model starting from a set of data contaminated by *large amounts of outliers*. The RANSAC algorithm calculates a whole estimation of the homography matrix based on a threshold value (ε) (limit for finding inliers of the model). RANSAC practically guarantees an optimal homography matrix by naively computing many matrices and choosing the one with the most points that *agree with a transformation* (**maximum number of inliers**). This technique finds the best perspective transformation **H** between the source and destination planes

$$s_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

so that the ***back-projection error (below) is minimized***.

$$\sum_i (x_i' - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}})^2 + (y_i' - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}})^2$$

Homography has multiple benefits in image processing. This projective transformation can map *any four points* in the plane to any other four. *Rotations, translations, scalings, and shears* are all special cases. Hence, homography can become accustomed to changing the plane and the perspective of an image. Here **Homogeneous Coordinates** are helpful, which increase the dimension of current coordinates. In this case, the **2D points** of the image will become **3D points** in homogeneous coordinates.

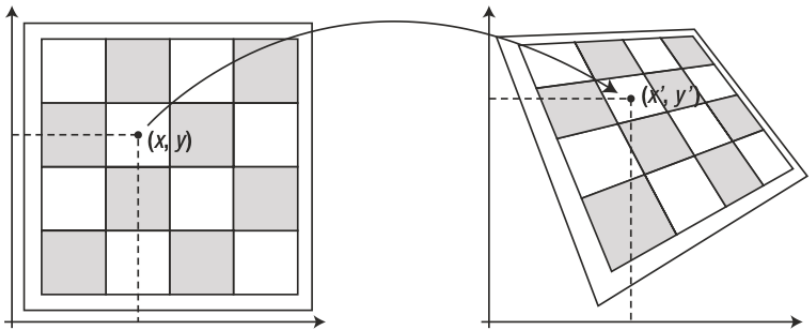

Figure 4. *Homography application to change the perspective of an image*

For any given homography, there is a need to compute values of a **3x3 matrix** (**H**), so it can be written as:

$$H = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix}$$

For mapping, consider a set of points - (x_1, y_1) in the first image and (x_2, y_2) in the second one. The homography is needed so that if the authors multiply the matrix containing the (x_1, y_1) coordinate values of one image by **H**, the result will be the (x_2, y_2) coordinate values of the resulting image to which the first image needs to be transformed. So the Homography matrix **H** maps in the following way:

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

Although there are nine entries in the matrix , the homography only contains **eight degrees of freedom** (the entries are redundant concerning scale). The reason is that the bottom right element h_{22} in the Homography matrix is **always 1**, so there are **8 unknowns**.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1 * X_1 & -y_1 * X_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2 * X_2 & -y_2 * X_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3 * X_3 & -y_3 * X_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4 * X_4 & -y_4 * X_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1 * Y_1 & -y_1 * Y_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2 * Y_2 & -y_2 * Y_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3 * Y_3 & -y_3 * Y_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4 * Y_4 & -y_4 * Y_4 \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}$$

Equation 1. *Homography calculation*

The lower case **x** and **ys** correspond to coordinate values in the original image, and the upper case **Xs** and **Ys** correspond to coordinate values in the target image/coordinate system we want to morph the original image to. In order to estimate the matrix, **4** points in the input image are selected and mapped to the desired locations in the unknown output image according to the use case (*8 equations and 8 unknowns*, which can be easily solved by solving a linear matrix equation $\mathbf{Ax = b}$ using function **np.linalg.solve(A,b)**).

RANSAC repeatedly selects 4 random pairs and *calculates a homography matrix* based on these 4 pairs. Then it takes each pair of features, multiplies the first coordinate by the newly found homography matrix, and looks to see if the calculated points are close to the actual points in the second coordinate. If the **SSD** (*Sum of Squared Differences*) of the two points **is less than some limit** (ε), the pair is an **"inlier"**, which means that the point of image 1 is successfully converted to image 2 using our homography matrix. The whole process is repeated **n** times, and as a result, the homography matrix with the most extensive set of inliers is chosen, which will be used to project the first image.

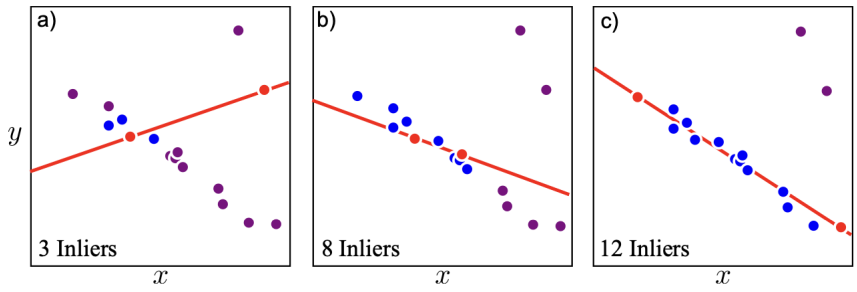

Figure 5. *RANSAC procedure. The best fit (the most inliers) is* ***c)***

Once the transformation matrix **H** is calculated, the perspective transformation is applied (**cv2.warpPerspective()** function) to the entire input image to get the final transformed image. As a result, the proper image stitching will be done, and the resulting picture will look like one beautiful panorama.

4 Data and implementation

Several sources were used to execute the algorithm, including own photo materials and open-source images on the Internet. The image stitching algorithm gets a set of two or more images in *jpg* or *png* format (specified by the user when entering a path) and outputs one single wide-view panorama stitched from those pictures. Examples of data input and output are presented in the **Results** section.

Images can be stitched either horizontally or vertically. In the latter case, each image is rotated by 90 degrees clockwise, stitched, and rotated back. Therefore, it is essential for the user to specify the type of stitching they want to perform. It can be done by typing **H** for horizontal stitching and **V** for vertical stitching, respectively, in the particular input field.

Link to the implementation of the Image Stitching on **Git**. [9].

A Implementation

The main module in the project is stitching.py, where the stitching process is implemented. Function *read_input()* takes filenames (images) as input and, in the end, returns stitched panorama photo. Inside *main()*, function *feature_matching()* for performing feature matching (KNN Algorithm with Lowe's ratio test from knn.py module and Cross-Check validation) is called, which computes keypoint matches for images, which are used further for RANSAC algorithm. Additionally, it draws the found matches of keypoints from images.

After performing feature matching, *ransac()* and *get_homography()*, implemented in ransac.py, are called to help with creating a natural panorama. Function *ransac()* applies RANdom SAmple Consensus method and calls *get_homography()* function to find the best Homography matrix. In the end the *stitch()* function is called to make the stitch between the images less noticeable.

B Pseudocode for Several Images Stitching

Data: *images* = n images;
Result: Stitched image

foreach *image* ∈ *images* **do**
  perform Image Stitching algorithm on two neighbouring images;
  change the path of the second image to the new stitched image;
end
return *images[-1]*;

Algorithm 1: Several images stitching

C Pseudocode for Image Stitching

Data: images = 2 images;
rotate = boolean;
Result: Panorama image

if *rotate is True* **then**
  rotate image1;
  rotate image2
end
detect keypoints, extract local invariant descriptors using SIFT;
match features between images using KNN, Lowe's ratio test and CrossCheck;
estimate Homography matrices using matched features by applying RANSAC;
apply a warping transformation using H obtained;
obtain panorama image;
if *rotate is True* **then**
  rotate panorama
end
return panorama;

Algorithm 2: Image Stitching process

D Pseudocode for Feature Matching

Data: Q = set of query points;
R = set of reference points;
Result: *matches* = dictionary, key - *query* point, value - *best reference* point;
foreach *query point* $q \in Q$ **do**
    compute distances between q and all $r \in R$;
    sort the computed distances;
    select k-nearest reference points corresponding to k smallest distances (k=2);
    eliminate false matches by applying *Lowe's ratio test*;
    filter keypoint matches using *CrossCheck* validation;
end
return matches;

Algorithm 3: KNN & Lowe's ratio test & CrossCheck methods

E Pseudocode for RANSAC Method

Data: img1_points = array with points from image1;
img2_points = array with points from image2;
numIterations = maximal number of iterations;
n = number of random points to pick at every iteration (i.e. number of points to find the tranformation matrix);
Result: Homography matrix

bestHomography ← *None*;
best_inliers ← 0 `/* the greatest amount of inliers */`
num_points = img1_points.length();
for *i in numIterations* **do**
    randomSub = n random indices from 0:num_points;
    random_img1 = img1_points[randomSub];
    random_img2 = img2_points[randomSub];
    Homography = **ComputeHomography**(random_img1, random_img2);
    num_inliers = **ComputeInliers**(Homography);
    if *num_inliers* > *best_inliers* **then**
        bestHomography = Homography;
        best_inliers = num_inliers;
    end
end
return bestHomography;

Algorithm 4: RANSAC method

F Pseudocode for Calculating Homography

Data: *image1* = list of **4** random keypoints from image1;
image2 = list of **4** random keypoints from image2;
Result: *values* = 2D array with subarrays as homography matrix values;
estimate a **8x8** matrix **A** using image1 and image2;`/* Equation 1 */`
estimate a **8x1** matrix **b** using image2 keypoints coordinates;
solve a linear matrix equation **Ax = b** to get homography matrix values as a 2D array (*values*);
return values;

Algorithm 5: Homography matrix estimation

5 Results

The result of the project is an image stitching algorithm. It stitches several images into one wide-view panorama, using *cv2 library* built-in **SIFT** algorithm for keypoint detection, **KNN** algorithm for feature matching, and **Homography estimation with RANSAC algorithm** for obtaining the best-stitched image.

By inputting the paths to the images the user wants to stitch, they can get one full image obtained from cropped ones.

The following images of Paris are an example of the algorithm's work on stitching the images horizontally.

Figure 6. *Input images A*

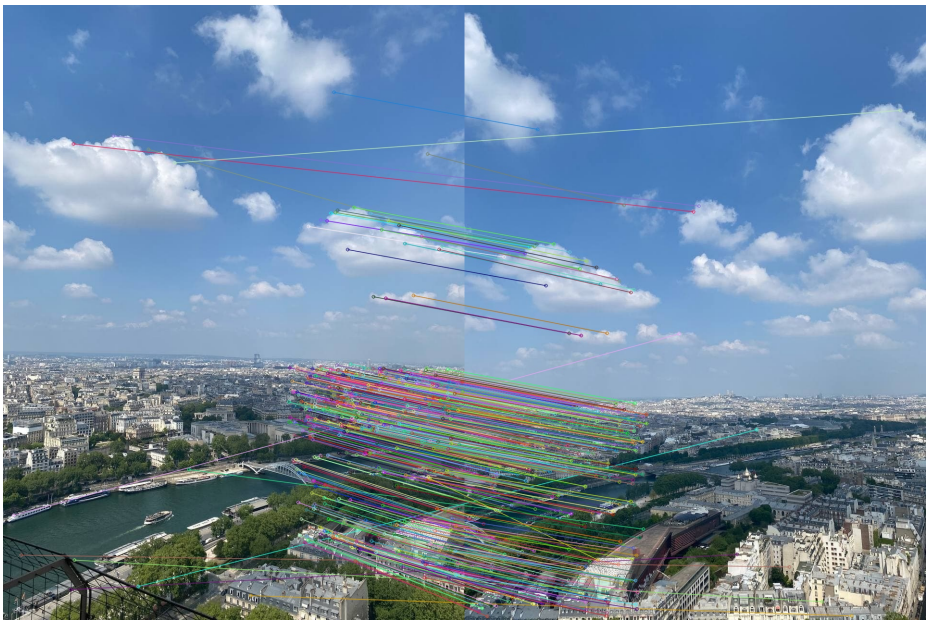

Figure 7. *Image after keypoint detection and matching A*

Figure 8. *Output image A*

The following images of a lake are an example of the algorithm's work on stitching the images vertically.

Figure 9. *Input images B*

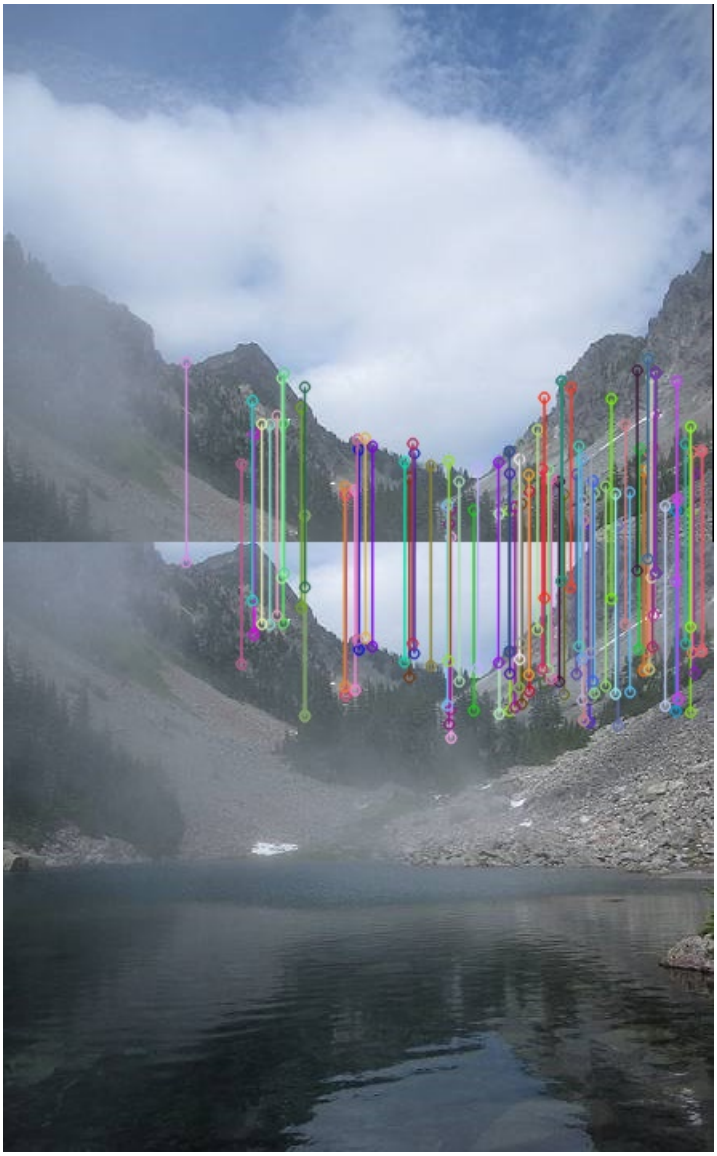

Figure 10. *Image after keypoint detection and matching B*

Figure 11. *Output image A*

The following images of a mountain are an example of the algorithm's work on stitching more than two images horizontally.

Figure 12. *Input images C*

Figure 13. *Images after keypoint detection and matching C*

Figure 14. *Output image C*

The algorithm works poorly with photos that have too few keypoints. An example is an entirely black painting on a white wall - the program will not find enough common features to stitch the photos.

When stitching more than two photos together, the "black fields" problem might appear. Because images are stitched iteratively, and the intermediate results are saved in *jpg* or *png* format, the black fields are also preserved, causing the following resulting pictures to have a certain area being black. This problem is particularly solved in code implementation.

When stitching the images vertically, the user must specify this before entering paths to the pictures. It is essential for the algorithm to rotate the images by 90 degrees clockwise, as it primarily stitches only horizontal photos. In the end, the algorithm returns a stitched and rotated by 90 degrees counterclockwise panorama.

6 Conclusions

The authors studied an image stitching problem that combines multiple photos to produce a beautiful natural panorama in this project. This process works for various scenes, including images taken indoors and outdoors.

During the study, the authors compared different approaches for each of the image stitching problem steps to identify the most suitable algorithms for project development. As a result, the authors have implemented several algorithms: K-nearest-neighbors for feature matching with Lowe's ratio test and CrossCheck to extract the most valid keypoint matching pairs. Additionally, the RANSAC algorithm to calculate the most extensive set of features that can be used to form a homography matrix to create a projection. This method is very beneficial in finding the best model when the dataset contains many outliers (for example, half of the points or even more).

The effectiveness of the image stitching method depends on the quality of the resulting stitched image since high-quality images (i.e., a high number of pixels) have thousands of features - hence, thousands of keypoints while low-quality images may have only a few hundreds. Additional techniques can also be incorporated into project implementation to improve the performance of creating panorama images. For example, implementing image blending algorithms to eliminate sharp intensity changes at the image joins or Singular Value Decomposition (SVD) for finding least-square solutions to Homography evaluation pairs.

The algorithm can be ameliorated for several images by finding the best order to be stitched in and dealing with black fields every step. It might worsen the program's effectiveness but produce a more cleaner and better-stitched image.

Due to the methodology used, the algorithm lacks effectiveness on high-resolution pictures. The higher quality of the images, the longer the stitching process is performed. However, if the photos are of average or minor quality, the algorithm yields the result in up to 10 minutes for each pair of photos if the CrossCheck validation is performed too.

7 Literature

1. Alex Caparas, Arnel Fajardo, Ruji Medina, "Feature-based Automatic Image Stitching Using SIFT, KNN and RANSAC", International Journal of Advanced Trends in Computer Science and Engineering, Volume 9, No.1.1, 2020.

2. Matthew Brown and David G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features", Department of Computer Science, University of British Columbia, Vancouver, Canada.

3. Szeliski, R. (2022) Computer Vision: Algorithms and Applications, Second Edition, pp. 511-512.

References

[1] https://en.wikipedia.org/wiki/Harris_corner_detector

[2] Daliyah S. Aljutaili, Redna A. Almutlaq, Suha A. Alharbi, Dina M. Ibrahim , "A Speeded up Robust Scale-Invariant Feature Transform Currency Recognition Algorithm", World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol:12, No:6, 2018.

[3] Chen, S. E. (1995). QuickTime VR – an image-based approach to virtual environment navigation. In ACM SIGGRAPH Conference Proceedings, pp. 29–38.

[4] Szeliski, R. and Shum, H.-Y. (1997). Creating full view panoramic image mosaics and environment maps. In ACM SIGGRAPH Conference Proceedings, pp. 251–258.

[5] https://www.analyticsvidhya.com

[6] https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

[7] https://stackoverflow.com/questions/51197091/how-does-the-lowes-ratio-test-work

[8] https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf

[9] https://github.com/linvieson/image-stitching

Keep strong, stay healthy and learn LA!