Ukrainian Catholic University

Faculty of Applied Sciences

Business Analytics & Computer Science Programmes

Linear Algebra
Final report

Panorama Stitching

Team:
Daryna Kuzyshyn
Olha Havryliuk
Anastasiia Dynia

27th of April 2024

1. General overview of the problem

In the field of computer vision, one area that stands out for its practical application and visual appeal is panorama stitching. The process involves taking multiple images from different angles and smoothly merging them into a single panoramic image. This technique allows for a broader view that is much wider than what can be captured by regular photography methods, offering a comprehensive perspective of landscapes, cityscapes, or any other scenery.

Even though many of today's digital devices can stitch panoramas, creating a perfect panoramic image still remains an issue because of technical challenges. The primary issues come from the need to accurately align and blend overlapping images without visible seams, distortions, or exposure differences. Each image in the series can vary in terms of angle, scale, lighting, and perspective, all these things make the stitching process complex. Additionally, the technical process of finding exact points that match up in different images, figuring out the best way to line them up, and seamlessly combining them into one picture without any flaws requires a strong and reliable method.

Our project aim is to develop a panorama stitching algorithm that addresses these challenges by using advanced techniques in Computer Vision and Linear Algebra. The authors want not just to develop the stitching process algorithm but to enhance the quality of panoramic photography.

2. A review of related work and possible approaches to solutions

2.1. Related work

The related work that was discovered is named "Automatic Panoramic Image Stitching using Invariant Features". The authors utilize SIFT features for their invariance to scale and rotation, ensuring that feature matching is resilient to changes in image scale, orientation, and illumination. They describe an automatic algorithm that combines feature matching with a multi-image matching strategy, using techniques like RANSAC for homography estimation to align and stitch the images accurately. The result of such approach is the fully automatic panorama stitching system. It effectively stitches images taken from different viewpoints or with different zoom levels. The algorithm can process images in an unordered dataset and identify multiple panoramas, ensuring that the transitions between stitched images are seamless. But the effectiveness of the algorithm heavily relies on the quality and reliability of the feature matches, moreover, the processes of feature extraction, matching, and image stitching are computationally intensive.

Another related work is "360-Degree Panoramic Image Stitching for Un-ordered Images Based on Harris Corner Detection". The system successfully automates the arrangement of unordered images. By utilizing Harris corner detection, the system effectively identifies keypoints in images, then the distance between points is calculated and matches are determined. Then the estimation of the shifting matrix between images is calculated using the Random

Sample Consensus (RANSAC) algorithm. The implementation of blending techniques results in high-quality panoramic outputs without visible seams. However, the system's performance is highly dependent on the effectiveness of the Harris corner detection. In scenarios with low-texture environments or poor lighting, the detection and, consequently, the stitching quality might be degraded.

2.2. Possible approaches

Image panorama techniques are classified into two categories:

- Direct techniques
- Feature-based techniques

Let's take a closer look at each of them:

Direct Techniques:

The direct technique is dependent on the comparison of pixel intensities of the images and it decreases total differences among overlapping pixels. Each pixel intensity is contrasted with every other pixel intensity, making it an extremely complex method.

Feature-based techniques:

Feature-based techniques are aimed at determining a relationship between the images based on extracting different features. Feature-based methods are used to match different shapes between images such as points, lines, etc. The main steps for image stitching are **feature detection and description, feature matching and homography estimation**.

Step 1. Feature detection and description

Feature detection involves identifying specific points, regions, or structures in an image that are significant and can be used as references for further analysis. Once relevant features are detected, the next step is to describe these features in a way that allows for efficient matching and recognition. The goal is to create a representation of the features that captures their distinctive characteristics while being resistant to variations that might occur in real-world images. There are such techniques that allows us to detect and describe necessary features:

- *Harris Corner Detector.* Harris algorithm is a point extract algorithm to get all the corners in the image. The Harris detector looks at the average directional strength.
- *SURF Detector.* This algorithm is used based on multi-dimensional space theory and acceleration calculations using fast matrix approximation and Hessian's definition of "integrated images".
- *ORB Detector.* The ORB is a fast binary descriptor based on BRIEF key points and FAST detectors. BRIEF is a new feature descriptor that uses a smooth image patch binary test between pixels.
- *SIFT Detector.* SIFT calculates the local image descriptor of each key point based on the value of the image gradient and the direction of each point of the image sample in the area centered on the key point.

Step 2. Feature matching

After extracting features for pair of an image, the next step is to match the feature extracted. The matching process uses descriptor data to compare matching points. This step is intended to compare the best features of an image with the other image. Then if the features of the input images match, to achieve accurate feature matching, the locations of the identical features will be identified as matched pairs. Possible methods for feature matching are provided below:

- *Brute-Force Matcher.* The matches are evaluated on the basis of Euclidean Distance between the two key points. Euclidean distance of every keypoint in the first image is calculated with every other keypoint in the second image. The good matches are then separated by certain minimum distance criteria.
- *KNN matcher.* After identifying features in each image, the KNN matcher considers each feature point from one image and searches for the k closest feature points in the other image, based on a defined distance metric, for example, the Euclidean distance.

Step 3. Homography estimation

Homography estimation is a technique used to find the relationship between two images of the same scene, but captured from different viewpoints. It is used to align images, correct for perspective distortions, or perform image stitching. In order to transfer similar points from one image to another, the 3x3 homography transformation matrix is used, which shows the correspondence between two images' corresponding points' pixel coordinates. Geometric changes including translation, rotation, scaling, and perspective distortion are possible using homography. There are such techniques that can be used to perform homography estimation:

- *RANSAC.* This algorithm estimates the homography matrix by picking a subset of the matched keypoints iteratively. The optimal homography matrix with the most inliers is chosen after assessing the number of inliers that suit the predicted model.
- *Hough transforms.* The Hough transform is a feature extraction technique that converts an image from Cartesian to polar coordinates. Any point within the image space is represented by a sinusoidal curve in the Hough space. In addition, two points in a line segment generate two curves, which are overlaid at a location that corresponds with a line through the image space.

3. A brief explanation of the algorithm chosen and its pros and cons

For this project the feature-based approach was chosen , because it has such advantages:

- It can accurately match features across images that are taken from different angles or positions
- The features can be matched even if the images have been captured from different distances or orientations
- It helps reduce the impact of noise in the resulting picture

As it was already mentioned, there are **3 key steps for image stitching.**

- For feature detection and description **SIFT** was chosen. It is more accurate than any other descriptors and is rotation and scale invariant. But one of the drawbacks is that it is mathematically complicated and computationally heavy.
- For feature matching **KNN** is used. The pros of this method are that it is really simple to implement and it has relatively high accuracy. But, on the other hand, sometimes it can be computationally expensive and it has a high sensitivity to the choice of k and the distance metric.
- For homography estimation **the RANSAC** will be implemented. It is highly robust to outliers and is conceptually simple and straightforward to implement. But the cons are that sometimes too many iterations are required and this algorithm is highly sensitive to the threshold parameter (ϵ) that determines which points are considered inliers.

4. The theoretical part behind the algorithms

Here's a closer look at the theory and some of the formulas behind each of the stages of panorama stitching implementation:

4.1. Feature detection and description

In general, Scale-invariant feature transform(**SIFT**) algorithm can be decomposed into four steps:

- **Feature point (also called keypoint) detection:**

The first stage of SIFT is to identify keypoint candidates that are invariant to scale and orientation. To achieve scale invariance, the algorithm starts by constructing a scale space, which is typically done using the Gaussian function. The scale space of an image I is defined as a function $L(x, y, \sigma)$ that is produced by convolving $I(x, y)$ with a Gaussian kernel $G(x, y, \sigma)$ at different scales σ:

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

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

A process, when the image is repeatedly scaled and blurred is called generating octaves. It's typically ideal to scale the image four times and then blur each scaled image five times.

Then the SIFT uses the Difference of Gaussians (DoG) function to efficiently approximate the scale-normalized Laplacian of Gaussian (LoG), $\nabla^2_\sigma G$, which is computationally more expensive. The DoG is obtained by subtracting two nearby scales:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{3}$$

After a construction of a DoG, we can find the proposal keypoints by searching for an extrema in the DoG images. For each pixel in the i-th image, we compare its value with values

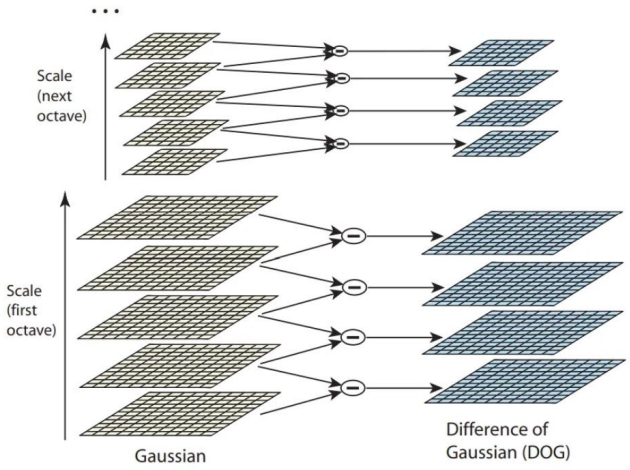

Figure 1: Constructing a DoG

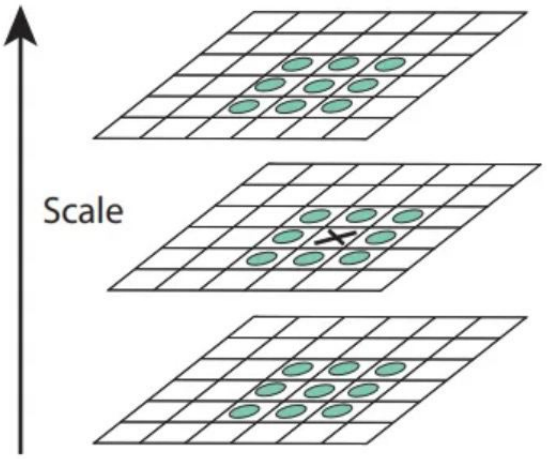

Figure 2: Search for extremum

of all 8 neighbors in this image and with values of 9 neighbors in $(i-1)$-th and $(i+1)$-th images.

For each proposal point, we fit the quadratic model to find the actual extremum and check whether it has enough contrast and doesn't lie on the edge. The result of this stage is a set of good keypoints.

- **Feature point localization:**

Once keypoint candidates have been found, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge.

Consider D, which is the result of DoG method:

$$D(\mathbf{z_0} + \mathbf{z}) \approx D(\mathbf{z_0}) + \left(\frac{\partial D}{\partial \mathbf{z}}\bigg|_{\mathbf{z_0}} \right)^{\mathrm{T}} \mathbf{z} + \frac{1}{2}\mathbf{z}^{\mathrm{T}} \left(\frac{\partial^2 D}{\partial \mathbf{z}^2}\bigg|_{\mathbf{z_0}} \right) \mathbf{z}, \tag{4}$$

where the derivatives are evaluated at the proposed point $\mathbf{z_0} = [x_0, y_0, \sigma_0]^T$ and $\mathbf{z} = [\delta_x, \delta_y, \delta_\sigma]^T$ is the offset from this point.

The Hessian matrix, used to reject features with low contrast or those located on edges, is defined as:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Then we compute the ratio of the eigenvalues λ_1 and λ_2:

$$\frac{\mathrm{Tr}^2(\mathbf{H})}{\mathrm{Det}(\mathbf{H})} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} < \frac{(r+1)^2}{r}, \tag{5}$$

where

$$r \text{ is a threshold ratio,}$$
$$\mathrm{Tr}(\mathbf{H}) = D_{xx} + D_{yy},$$
$$\mathrm{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2.$$

Points with a high ratio of eigenvalues are identified as edge points and are excluded from consideration.

- **Orientation assignment**

The keypoints obtained in the previous step are assigned an orientation so that they are rotation-invariant.

In this step an **orientation histogram** will be used. It is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint.

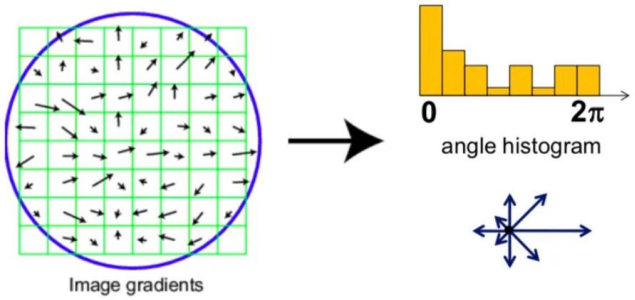

Figure 3: Orientation assignment

The peaks observed in the orientation histogram represent the dominant directions of local gradients. By identifying the highest peak in the histogram, we choose the dominant orientation. Subsequently, any other local peak within 80 % of the highest peak is also considered,

creating the keypoints with corresponding orientations. In locations with multiple peaks of similar magnitude, several keypoints are created at the same location and scale but different orientations.

- **Keypoint description**

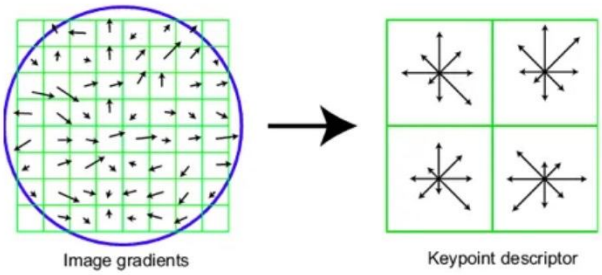

Figure 4: Calculating descriptor

In the final stage of the SIFT algorithm, descriptors are created for each keypoint. Since there is a list of feature points which are described in terms of location, scale, and orientation, one can construct a local coordinate system around the feature point which should be similar across different views of the same feature. This descriptor itself is 128-dimensional normalized vector formed from histogram of gradients.

4.2. Feature matching

Once features are extracted from each image, the next step is to find matching features in different images. For this we use **KNN-algorithm**. For each feature in one image, the algorithm searches for the closest k features in the other images based on a distance metric, for example Euclidean distance.

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{6}$$

The distance metric quantifies how similar two features are, with a smaller distance indicating a higher similarity. The ideal value for K in this algorithm is 2. This ensures that the two most suitable descriptors for each descriptor on both images are left.

To further refine the matches and exclude potentially incorrect ones, the **Davis Lowe's ratio and Crosscheck technique** are applied.

- David Lowe's ratio

  It eliminates the false matches in such way that the distances between the first-best and second-best candidates are compared. If the ratio of these distances is smaller than a threshold value, typically set to 0.8, the first-best match is considered valid for the keypoint. This approach effectively eliminates around 90 % of false matches while discarding less than 5 % of correct matches.

- Crosscheck technique

  It verifies matches by conducting the matching process in reverse, ensuring consistency in the pairs of keypoints between images. Only matches that produce the same pair of keypoints in both directions are considered valid, resulting in more accurate and invariant results irrespective of the image order.

Now, having the resulting matches, we can proceed with further calculations.

4.3. Homography estimation

In the process of stitching together several photos, the fundamental step is homography estimation. It involves using a homography matrix to describe the transformation between different image planes, enabling the transfer of corresponding points from one image to another.

A homography is a 3×3 non-singular matrix that relates points in one plane to points in another plane, under projective transformations.

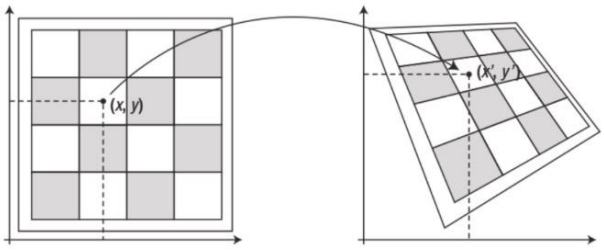

Figure 5: Changing the perspective of an image

Homography is represented by the matrix $\mathbf{H}$:

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

where h_{ij} denotes the element in the i-th row and j-th column of $\mathbf{H}$.

Lets assume $\mathbf{x} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$ and $\mathbf{x}' = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ in homogeneous coordinates.

The relationship between the two points under the homography transformation is given by:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = c \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

By eliminating the scale factor c, the relationship can be written in a linear form $A\mathbf{h} = \mathbf{0}$, where A is a 2×9 matrix constructed from the coordinates of the corresponding points, and $\mathbf{h}$ is a 9×1 column vector containing the flattened $\mathbf{H}$ matrix:

Homography has 8 degrees of freedom even though it contains 9 elements (3x3 matrix) i.e. the number of unknowns that need to be solved for is 8.

Computing a homography using **the RANdom SAmple Consensus(RANSAC) algorithm** involves mapping points from one plane to another with a perspective transformation. The RANSAC determines the complete homography matrix based on a threshold value.

Given a set of point correspondences between two images, we denote the points in the source image as $\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$ and the corresponding points in the destination image as $\mathbf{x}'_i = \begin{bmatrix} x'_i \\ y'_i \end{bmatrix}$:

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \approx \mathbf{H} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

We randomly pick four good matches, compute a homography from these four, and test how good this homography is by checking how many of the good matches are consistent with the homography. If x_i, x'_i are a good match(good match means that the Sum of Squared Differences of the two points is less than a threshold(ϵ)), we're assuming that in most cases the homography will map x_i to something near x'_i. Good matches that are consistent with the homography are called inliers, and those that aren't are called outliers.

RANSAC iterates, each time selecting a new subset of points, estimating a new $\mathbf{H}$. The best $\mathbf{H}$ is the one that results in the largest number of inliers. As a result, we get that the features in both images are on top of each other and the stitching process is done.

5. Data description and implementation

5.1. Data description

The photos for the dataset were taken from the internet, particularly the images in both png. and jpg. format. The dataset consists of images taken from different angles and that need to be stitched horizontally and vertically. These images serve as the input for the stitching algorithm, which seamlessly combines multiple images into a well-aligned panorama. Examples of the input images and the resulting panoramas are shown in the **Results** section.

You can also see the dataset and the implementation if you follow the link to the GitHub of this project.

5.2. Implementation

The algorithm supports multiple stitching, enabling to combine more than two images to create comprehensive panoramas. It works in such way: the user first writes whether to stitch images horizontally or vertically and then gives two or more images(pathes to them) for stitching and as a result receives beautiful panorama.

For vertical stitching, the algorithm rotates the images by 90 degrees to ensure accurate alignment before proceeding with the stitching process. This guarantees that the resulting panoramas are visually well-aligned.

The function **read_input_and_stitch()** takes the path to the images from the user and produces a panorama image. Inside this function, **stitch_images()** is repeatedly called to stitch each pair of images. This function performs keypoint detection and feature matching using the **sift.detectAndCompute()** and **knn_matcher()**, enhanced by Lowe's ratio test and cross-check validation. These matches are essential inputs for the RANSAC algorithm to estimate the correct alignment. After performing feature matching within **stitch_images()**, the function **ransac()** is used to estimate the homography matrix that will be used for aligning and transforming the images. Finally, after the images are aligned and transformed, the **trim_black_borders()** function is called to clean up the panorama by removing any black borders resulting from the warping process.

5.3. Pseudocodes for Algorithms

Stitching Images

Data: pathes to photos of the user;
Result: well-aligned stitched panorama;

If user specifies "vertical", set boolean rotate to True, otherwise set to False;
For each pair of consecutive images (img1, img2) in images **do**
  If rotate is True **then**
    Rotate img1 and img2 90 degrees clockwise using cv2.rotate;
  End
  Convert img1 and img2 to grayscale (gray1, gray2) using cv2.cvtColor;
  Compute keypoints (kp1, kp2) and descriptors (des1, des2) using SIFT;
  Match features between des1 and des2 using KNN and store in matches1;
  Perform cross-check with matches2 to get good_matches;
  Estimate the best homography matrix H using RANSAC from good_matches;
  Warp img2 using H to align with img1, resulting in img2_transformed;
  Overlay img1 on img2_transformed to create a partial panorama panorama;
  If rotate is True **then**
    Rotate the panorama back to original orientation using cv2.rotate;
  End
  Trim black borders from the resulting panorama using trim_black_borders function;
  Move to the next pair using the current panorama as img1;
End
Display the final stitched panorama using cv2.imshow;
Wait for user interaction to close the display window using cv2.waitKey(0);
End

KNN Matcher

Data: des1 = list of descriptors from image1;
des2 = list of descriptors from image2;
k = number of nearest neighbors to find (default 2);

Result: matches = dictionary where each key is an index in des1 and the value is a list of k-nearest descriptors from des2;
Initialize a dictionary "matches"with keys from 0 to length of des1, each key having an empty list as value;
For each descriptor index i in des1 do
Initialize a list "distances"to store distances from des1[i] to each descriptor in des2;
    For each descriptor index j in des2 **do**
        Compute Euclidean distance between des1[i] and des2[j] using numpy's linalg.norm;
        Append a tuple of (distance, j) to "distances";
    End
    Sort "distances"based on distance values;
    Set "matches[i]"to the first k elements of "distances";
End
Return "matches";

Cross Check

Data: matches1 = dictionary of matches from des1 to des2;
matches2 = dictionary of matches from des2 to des1;
ratio = threshold for filtering matches (default 0.7);
Result: good_matches = list of tuples representing good matches after cross-check and ratio test;

Initialize an empty list "good_matches";
For each key i and value matches in matches1 **do**
    If there are at least two matches and the distance of the first match is less than ratio times the distance of the second match **then**
        Assign the distance and index of the first match to variables "min_dist.and "j";
        If matches2 at index j has at least two matches and the second match's index is i (cross check) then
            Append the tuple (i, j, min_dist) to "good_matches";
        End
    End
End
Return "good_matches";
End

RANSAC

Data: src_points = list of source points;
dst_points = list of destination points;
samples = number of points to sample per iteration (default 4);
iterations = number of iterations to run RANSAC (default 5000);
tolerance = reprojection error threshold to count inliers (default 3);
Result: best_homography = best homography matrix found;

best_inliers = list of indices of inliers;

Initialize best_homography to None and maximum_inliers to 0;
For each iteration **do**
Randomly select samples indices from src_points;
Extract corresponding points src_samples and dst_samples based on selected indices;
Compute homography matrix current_homography using compute_homography_matrix function with src_samples and dst_samples;
Count inliers current_inliers where reprojection error is below tolerance using count_inliers function;
If current_inliers is greater than maximum_inliers **then**
Update maximum_inliers to current_inliers;
Update best_homography to current_homography;
Update best_inliers to list of indices where reprojection error is below tolerance;
End
End
Return best_homography, best_inliers;
End

Computing Homography Matrix

Data: source_points, destination_points = lists of corresponding points;
Result: homography = 3x3 homography matrix;

Assert that the number of source_points equals the number of destination_points;
Initialize matrix matrix_A with dimensions (2number of points, 9) filled with zeros;
For each point pair (src, dst) in zipped source_points and destination_points **do**
Fill matrix_A with values derived from src and dst to set up the homography equations;
End
Perform Singular Value Decomposition (SVD) on matrix_A;
Extract the last row of V matrix from SVD, reshape it to 3x3, and **assign** to homography;
Normalize homography by dividing by the element at position (2, 2);
Return homography;
End

Note: The function transform_point() applies a homography transformation to a point, and the function count_inliers() counts how many points from the source match the destination within a given error threshold using the computed homography.

6. Results evaluation

6.1. Results

The project has successfully developed an image-stitching algorithm that combines multiple overlapping images into a single panoramic picture. To illustrate the capabilities of the

algorithm, the resulting panoramic images will be presented. Firstly, the authors applied the algorithm to a set of images of Roman bridge to demonstrate how it can stitch images together horizontally.

Figure 6: Image 1 and Image 2

Here is a result of identifying distinct features in each image that can be matched between them. The circles represent key points, with their sizes corresponding to the scale of the feature detected.

Figure 7: Image 1 and Image 2 after SIFT

To find the best matches between those key points the KNN algorithm with Cross Check validation was used.

Figure 8: Image after KNN and Cross Check

With these matches, the authors applied the RANSAC algorithm in order to find the best subset of matches-inliers.

Figure 9: Image after RANSAC

As a result, the both images are stitched and the panorama is presented below:

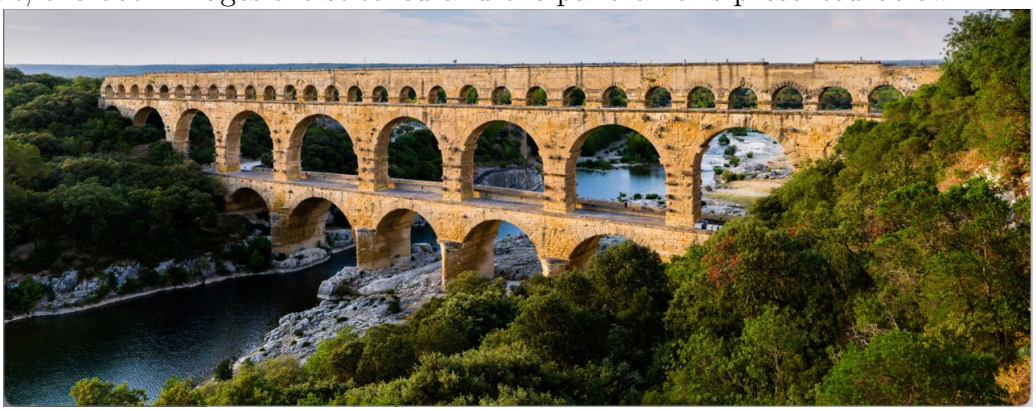

Figure 10: Result

Let's consider one more example of the multiple image stitching horizontally.

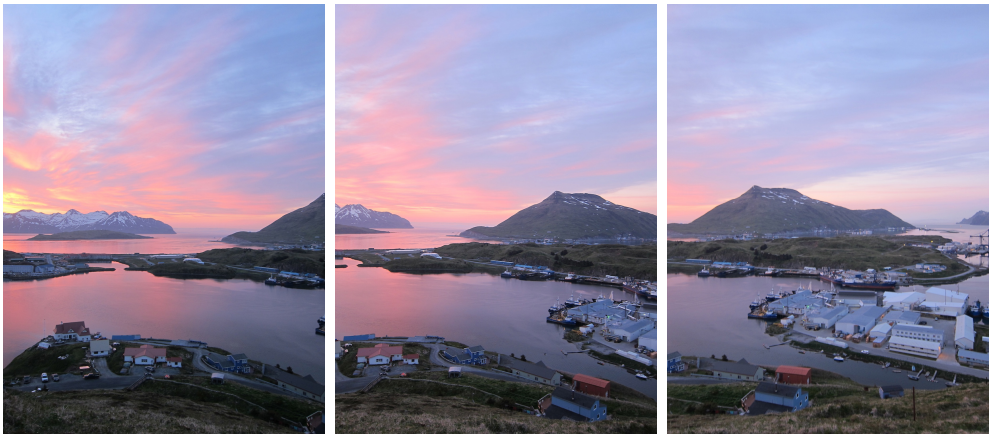

Figure 11: Image 1, Image 2, Image 3

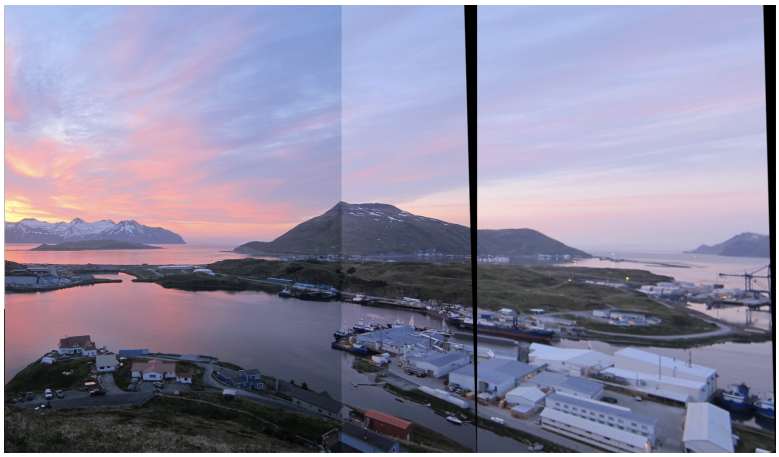

Figure 12: Result

Here is an example of multiple vertical stitching.

Figure 13: Image 1, Image 2, Image 3

Figure 14: Result

6.2. Analysis and Comparison

A frequent problem faced by authors is black fields in photos. They occur when images with varying exposures or perspectives are merged. Although the stitching algorithm aims to align the images smoothly, this issue still occurs in both cases - vertical and horizontal panorama stitching.

The authors also aim to compare two algorithms for panorama stitching: one implemented using the OpenCV library (CV2) and the other developed by themselves.

First of all, let's compare feature matching using KNN. After computing the number of matches of both algorithms, we got the following results:

- **built-in KNN:** 3847 matches
- **custom implementation of KNN:** 3901 matches

As we can see, there is no significant difference between the numbers of matches of those 2 algorithms, meaning that the custom-implemented KNN performs really well.

Figure 15: Matches with built-in KNN

Figure 16: Matches with custom implementation of KNN

Let's also compare the built-in and custom RANSAC implementation.

CV2 Homography Matrix:

$$\begin{bmatrix} 9{,}99710066 \times 10^{-1} & -3{,}73661847 \times 10^{-5} & 4{,}29003797 \times 10^{2} \\ 2{,}25877089 \times 10^{-5} & 9{,}99978689 \times 10^{-1} & -7{,}18038822 \times 10^{-3} \\ 4{,}80565469 \times 10^{-8} & -5{,}66742353 \times 10^{-8} & 1{,}00000000 \times 10^{0} \end{bmatrix}$$

Custom RANSAC Homography Matrix:

$$\begin{bmatrix} 9{,}99083830 \times 10^{-1} & -6{,}67069995 \times 10^{-5} & 4{,}29017362 \times 10^{2} \\ -1{,}63304126 \times 10^{-4} & 9{,}99781541 \times 10^{-1} & 4{,}69897127 \times 10^{-2} \\ -4{,}67768557 \times 10^{-7} & -1{,}75755995 \times 10^{-7} & 1{,}00000000 \times 10^{0} \end{bmatrix}$$

As we can see, the values in homography matrix are approximately the same, meaning that the custom RANSAC algorithm also performes pretty good.

Figure 17: Inliers with built-in RANSAC

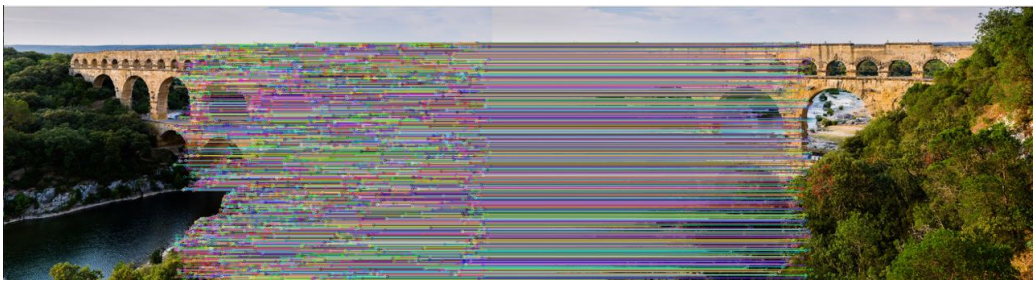

Figure 18: Inliers with custom RANSAC

Now we'll compare the final panoramas generated by both CV2 and the custom algorithms. Despite some minor differences in the algorithms and intermediate results, the final panoramas appear visually similar to the human eye, implying that the custom algorithm works well.

Figure 19: The resulting panorama developed by built-in methods

Figure 20: The resulting panoramas developed by custom-implemented methods

7. Conclusion

In this project, the authors worked on joining several photos into one big panorama picture. They chose the best tools for each part of the job, picking KNN to match up points across photos and RANSAC to line them up just right. The stitching works well when it takes photos with different levels of sharpness and blends them into panoramas. When the original photos have lots of clear details, the algorithm takes its time to do a thorough job, which makes the final panorama look great. When the photos are a bit blurrier, it finishes the job faster.

The issue of black areas showing up where the photos meet was tricky, but the authors improved the way the algorithm puts photos together to deal with it.The algorithm now minimizes these gaps, enhancing the continuity of the panoramic scene.

8. Literature

https://www.semanticscholar.org/paper/Feature-based-Automatic-Image-Stitching-Using-SIF
2C-Caparas/5383bbd594bf6e5873c38b617eb1d5b39f6c445a

https://www.researchgate.net/publication/339740410_Featurebased_Automatic_Image_
Stitching_Using_SIFT_KNN_and_RANSAC

https://towardsdatascience.com/sift-scale-invariant-feature-transform-c7233dc60f37