Panorama Image Stitching

Yueqi Ma, Xinyu Xie

Electronic and Information Engineering
School of Computer Science and Engineering
Macau University of Science and Technology

September 2023 Academic Year

---

1 Introduction

Image stitching, the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image, has been a topic of great interest within the field of computer vision. The ability to merge images seamlessly is crucial in various applications, ranging from photography and surveillance to medical imaging and cartographic mapping.

The main difficulties in image stitching include handling image distortions, ensuring seamless blending of images, dealing with varying lighting conditions, and aligning images with different perspectives or scales. These challenges require sophisticated algorithms to accurately detect and match features across the images and to align and blend them seamlessly.

In this paper, we implement a solution for image stitching that integrates Scale-Invariant Feature Transform (SIFT) or Oriented FAST and Rotated BRIEF (ORB) for feature detection, K-Nearest Neighbors (KNN) for feature matching, and Random Sample Consensus (RANSAC) for calculating homography. In addition to these techniques, we also implement a Gaussian mixture model for blending overlapping regions, ensuring a smooth transition and minimizing visible seams, further enhancing the overall quality of the stitched panoramic images.

2 Technical Solution

Our image stitching model consists of extracting representative features, finding correspondences between features, estimating the transformation between images, and merging the images to create a seamless composite.

2.1 Feature Extraction

The first step in image stitching is to extract features from the two input images. The SIFT (Scale-Invariant Feature Transform) algorithm is used to detect and describe keypoints in the images. The SIFT algorithm is known for its robustness to changes in lighting, scale, and rotation, making it well-suited for image stitching.

2.1.1 SIFT Method

The SIFT (Scale-Invariant Feature Transform) algorithm is a commonly used method for feature detection and matching, often employed in image stitching for detecting key points and computing feature descriptors. By matching key points and estimating transformation matrices between images, the alignment and stitching of images are achieved, resulting in a larger field of view or panoramic image.

1. First, the input images undergo preprocessing. This includes resizing the images, removing image distortions, or correcting any issues to improve the results of feature detection and matching.

2. Next, the SIFT algorithm is applied to each image to detect key points. SIFT is capable of detecting key points with unique properties across different scales and rotations. These key points are typically found in corners, edges, or textured regions of the images.

3. After detecting the key points, feature descriptors are computed for each of them. SIFT extracts feature descriptors from the local regions surrounding the key points, which are invariant to their position, orientation, and scale.

4. For each pair of images, the feature descriptors are compared to match the key points. Common methods involve calculating distances or similarity measures between the feature descriptors.

5. Based on the matched key points, the RANSAC algorithm is used to estimate a transformation matrix, such as a homography matrix, between the two images. This matrix describes the translation, rotation, and scaling transformation between one image and another.

6. The estimated transformation matrix is then used to rectify and align the images for stitching. This can be achieved by applying perspective transformation or other geometric transformations.

7. The above steps are iteratively repeated for multiple images until all of them are stitched together.

2.2 Feature Matching

Once the keypoints have been extracted from the two input images, the next step is to match the features between the two images. This step involves finding pairs of keypoints that correspond to each other, and the code uses the KNN (K-Nearest Neighbors) algorithm to match the features. The KNN algorithm finds the k nearest neighbours of each keypoint, where k is a parameter set based on the desired level of accuracy.

2.2.1 BF Matcher

Brute-Force Matcher is a straightforward and simple approach to find matches between features in two sets of data, such as keypoint descriptors in images.

The matcher exhaustively compares every feature in one set with every feature in the other set to find potential matches. It computes a distance or similarity measure between each pair of features and identifies matches based on a predefined threshold.

The cv2.BFMatcher function from the OpenCV library is used to create a brute-force matcher object. This function sets up the necessary parameters and configurations for performing the matching process. The cv2.NORM˙L2 distance measurement method is used, which calculates the Euclidean distance between feature descriptors.

The advantage of a brute-force matcher is its simplicity and reliability. It guarantees finding all possible matches within the specified threshold. However, it can be computationally expensive, especially for large datasets, as it involves a pairwise comparison of all features. To optimize the matching process, additional techniques such as cross-checking or approximate nearest neighbor search can be employed.

2.2.2 Lowe's Ratio Test

Lowe's ratio test is a technique used in feature matching to filter out unreliable matches between keypoints in images. It was introduced by David Lowe as a method to improve the robustness of matching algorithms based on distance measurements between feature descriptors.

The principle behind Lowe's ratio test is that for each keypoint in an image, its closest neighbor in the feature space should be significantly closer than any other neighbor. In other words, if the distance between a keypoint and its closest neighbor is much smaller than the distance to the second closest neighbor, it is likely that the match is reliable.

1. For each keypoint in the first image, the feature matching algorithm computes a list of potential matches by comparing its feature descriptor with the descriptors of keypoints in the second image.

2. The potential matches are sorted in ascending order based on their distances to the keypoint in the first image.

3. The test is applied by comparing the distances between the keypoint and its two closest neighbors. If the ratio of the distance to the closest neighbor and the distance to the second closest neighbor is below a certain threshold, the match is considered reliable.

4. If the ratio test passes for a match, it is retained as a good match. Otherwise, it is discarded.

The purpose of the ratio test is to eliminate matches that have ambiguous or weak correspondences. By considering the ratio between the distances of the two closest neighbors, it helps to filter out matches that have similar distances, indicating potential ambiguity. The threshold used in the ratio test determines the level of strictness for accepting matches. A higher threshold allows for more matches, including potentially weaker ones, while a lower threshold results in a more conservative selection of reliable matches.

Lowe's ratio test is a simple and effective technique to improve the reliability of feature matching algorithms. It helps to reduce the number of false matches and improve the overall accuracy of the image stitching process.

2.2.3 KNN Method

The K-Nearest Neighbors (KNN) algorithm performs feature matching and computes the homography matrix. KNN is a commonly used machine learning algorithm that finds the K most similar data points in a given dataset to a target data point. In this code, the KNN algorithm is employed for feature matching, which involves finding pairs of points in two images that have similar features.

1. First, the SIFT (Scale-Invariant Feature Transform) feature detector is used to detect keypoints (kpsA and kpsB) and compute their feature descriptors (featuresA and featuresB) in image A and image B.

2. A matcher object is created, and the matcher.knnMatch() function is used to match the feature descriptors. The knnMatch() function returns the K best matches for each feature point.

3. The matches are filtered using Lowe's ratio test. For each pair of matched feature points, if the distance of the first match (m.distance) is less than the distance of the second match (n.distance) multiplied by a given ratio, the match is considered good and retained.

4. If the number of good matches is greater than 4, two sets of points (ptsA and ptsB) are constructed, which contain the coordinates of the good matches in image A and image B, respectively.

5. The cv2.findHomography() function is used to estimate the homography matrix (H) between ptsA and ptsB. The homography matrix describes the perspective transformation between two images.

6. The function returns the list of good matches (good), the homography matrix (H), and the status (status).

2.3 Homography Calculation

After matching the features, the next step(Figure:1) is to estimate the homography matrix that maps the keypoints in the second image onto those in the first image. The RANSAC algorithm is capitalized on estimating the homography matrix, which is a mathematical model that estimates the transformation between two sets of points. The RANSAC algorithm works by randomly selecting a subset of the keypoints to use in estimating the homography matrix, and then testing the resulting matrix against a validation set of keypoints to check for accuracy.

2.3.1 RANSAC Estimation Method

RANSAC (RANdom SAmple Consensus) is an iterative estimation method commonly used in computer vision and robust regression problems. It is particularly useful when dealing with datasets that contain outliers or noise. RANSAC is widely used for tasks such as image stitching, object detection, and camera calibration. The flowchart of the RANSAC algorithm is shown in Fig:2.

1. Random Sample: Randomly select a minimal subset of data points from the entire dataset. The minimal subset is typically the smallest number of data points required to estimate the model parameters.

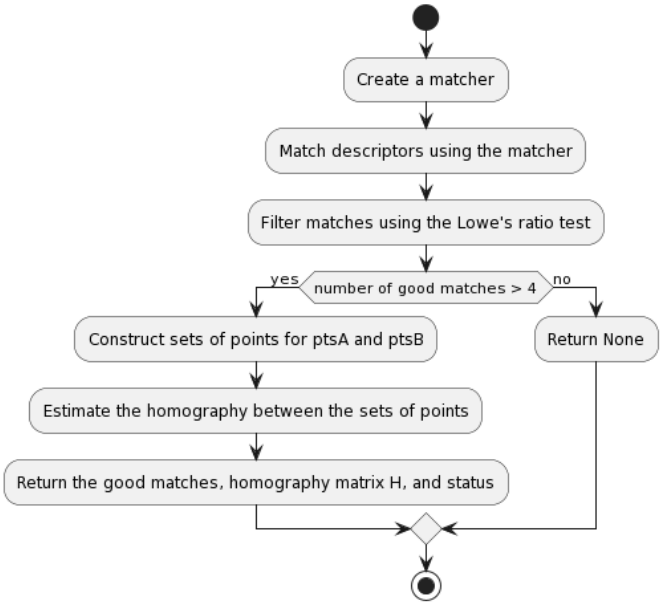

Figure 1: Homography Calculation

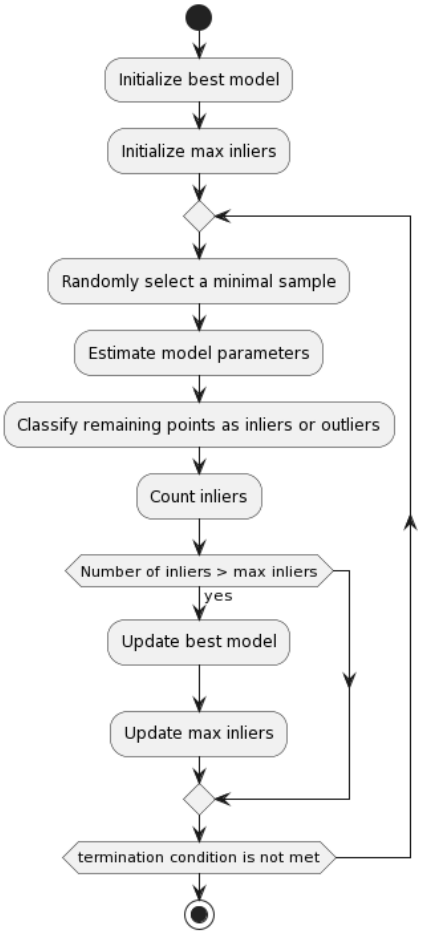

Figure 2: RANSAC

2. Model Estimation: Use the selected data points to estimate the model parameters. The specific method for estimating the model parameters depends on the problem at hand. For example, in the case of fitting a line, the model parameters are the slope and intercept.

3. Inlier Classification: Classify the remaining data points as inliers or outliers based on their compatibility with the estimated model. A data point is considered an inlier if it satisfies a predefined threshold or criterion. The threshold is typically based on a distance metric or error measure, such as the distance between a data point and the model.

4. Model Evaluation: Evaluate the quality of the estimated model by counting the number of inliers. The model with the highest number of inliers is considered the best estimate.

5. Iteration: Repeat steps 1-4 for a fixed number of iterations or until a termination condition is met. The termination condition can be a maximum number of iterations, reaching a desired number of inliers, or achieving a certain level of confidence in the model estimation.

6. Model Refinement: Optionally, after the iterations, the model can be refined using all the inliers or a larger subset of the data. This step aims to improve the accuracy of the estimated model.

RANSAC is effective in handling outliers because it only considers a subset of data points in each iteration. Outliers tend to have a significant impact on model estimation, and by randomly sampling a minimal subset, RANSAC can effectively minimize their influence. The algorithm robustly estimates the model by iteratively excluding outliers and refining the model based on the inliers.

Unfortunately, RANSAC has its limitations, such as the need to specify the minimum number of data points required to estimate the model and the threshold for classifying inliers and outliers. Setting these parameters correctly is crucial for achieving accurate results. Additionally, RANSAC may require more iterations to converge if the proportion of outliers in the dataset is high.

2.4 Image Composition

The process of image composition involves multiple steps to merge multiple images into a single panoramic image. The flowchart of the image composition is demonstrated in Fig:3.
First, input images are converted into grayscale, which simplifies the subsequent processing. Then, it detects keypoints and extracts feature descriptors using the SIFT algorithm to describe local features in the images. Next, it matches the corresponding feature points between the two images and computes a homography matrix that describes the perspective transformation between the images. The first image is then warped using this homography matrix to align its content with the second image, resulting in the stitching of the images. To achieve a smoother blending, the code performs pixel blending in the overlapping region using Gaussian-weighted blending. Afterwards, the non-overlapping portion of the second image is copied to the stitched image to ensure the complete display of both images. Finally, the stitched result is cropped to remove any blank areas and obtain the final seamless stitched image. By combining these steps, the code is able to stitch two images together into a seamless composite image, achieving the functionality of image stitching.

2.4.1 Image Blending

Once the homography matrix has been calculated, the next step is to blend the two images into a single image. Use homography transformation matrices obtained from the previous steps to progressively combine aligned images into the previous image. This stage aims to smooth any visible seams or borders between images and ensure consistent color and brightness across the panoramic image. The essential steps involved include:

Homography Transformation:
The homography transformation uses a 3x3 matrix to map points from one image to another. For simplicity, let's denote the homography matrix as $\boldsymbol{H}$ and the coordinates of a point in the source image as $(x, y, 1)$, transformed into the destination image as (x', y', ω').

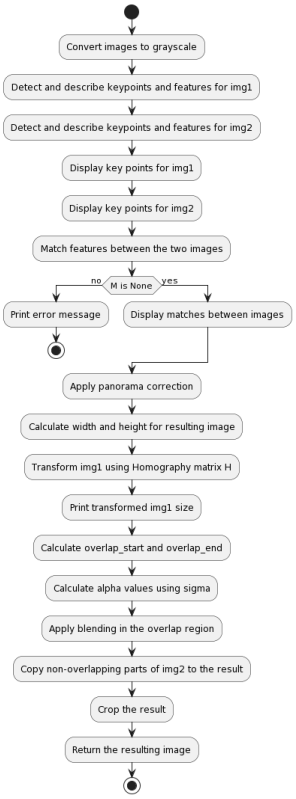

Figure 3: Image Composition

The transformation can be represented mathematically as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Blending Operation:

1. Weighted Averaging: Blending two images, img1 and img2, with weights α and β, respectively: result= $\alpha\times$img2 + $\beta\times$result, where $\alpha + \beta = 1$ (normalized weights).

2. Alpha Blending: Applying the weights α and β to the pixel values: result $(x, y) = \alpha\times$ img2 $(x, y) + \beta\times$ result (x, y).

3. Pixel-wise Blending: The blended pixel result(x, y) is computed using the weighted combination of the corresponding pixels from img1 and img2.

4. Weighted Function: The blending equation can be represented as:

$$result(x, y) = \alpha \times img2(x, y) + \beta \times result(x, y)$$

The values of α and β are determined based on the positions of the overlapping regions between the images. These weights control the contribution of each pixel from the second image (img2) to the result.

The goal is to ensure a smooth transition between the images, minimizing any visible seams or differences, to create a natural-looking composite image. The blending operation works pixel-wise, using a weighted combination of pixel values from both images to generate the final result.

2.4.2 Mosaicing

In the mosaicing stage of image composition, the objective is to seamlessly blend the overlapping regions of two images, img1 and img2, to create a panoramic view. Here's how it's done:

Blending Process:

1. **Gaussian Weighted Blending:**
   To obtain smoother transitions between the images, a Gaussian function is used to calculate blending weights (α) within the overlap area. The Gaussian distribution is defined as:

   $$\alpha = e^{-\frac{x^2}{2\sigma^2}}$$

   Here, σ controls the width of the Gaussian curve, affecting the weight distribution.

2. **Blending Pixels in the Overlap Region:**
   For each pixel in the overlap area, blending is applied using the calculated α and β weights.

   $$\text{result}(x, y) = \alpha \times \text{img2}(x, y) + \beta \times \text{result}(x, y)$$

   Here, $\text{img2}(x, y)$ represents the pixel value from img2 being blended, and $\text{result}(x, y)$ is the corresponding pixel in the existing panorama.

3. **Handling Non-overlapping Areas:**
   Pixels from non-overlapping parts of img2 are directly copied into the final panorama.

The Gaussian weighted blending technique is applied to smoothly merge the images, ensuring a seamless transition between them. Mathematical models, particularly the Gaussian distribution, are utilized to calculate blending weights, ensuring a visually pleasing and coherent panoramic output.

2.4.3 Image Cropping

Image cropping is a crucial image processing technique used to extract a specific region or area of interest from an image while discarding the rest. This process involves selecting and keeping a particular part of an image while removing the surrounding portions, allowing for a more focused view or to eliminate unwanted elements from the image.

In the final section of our mode, the cropping is used to crop an input image based on its contours.

The steps involved in the cropping process can be divided into five steps: convert to grayscale, thresholding, find contours, bounding rectangle of contour and cropping.

1. Convert to Grayscale:

   Converting an image to grayscale involves transforming a colored image into a single-channel representation that retains the luminance or brightness information while discarding color information. The principle behind converting an image to grayscale is based on the perception of human vision and the way colors are interpreted.

   In a standard color image, pixels are typically represented in a three-channel format: red, green, and blue (RGB). Each pixel's color is formed by combining different intensities of these three primary colors. When converting to grayscale, it simplifies the image by retaining only the intensity or luminance information without considering the color.

   The most common method for grayscale conversion involves a weighted average of the RGB values for each pixel. This process accounts for the way humans perceive colors and their relative brightness. One common example used for grayscale conversion is:

   $$Grayscale\ Value = (0.299 * Red) + (0.587 * Green) + (0.114 * Blue)$$

These weights (0.299, 0.587, and 0.114) correspond to the human eye's sensitivity to different colors. The human eye is most sensitive to green, followed by red and then blue. Therefore, a weighted average that emphasizes these factors provides a representation that simulates grayscale perception.

By computing this weighted average for each pixel in an RGB image, it derives a single intensity value that represents the pixel's grayscale value. This process effectively removes the color information while retaining the image's luminance or brightness information.

Converting to grayscale simplifies image processing tasks by reducing the complexity of handling color data, making it easier to perform various image analyses, feature detections, and manipulations. Additionally, grayscale images consume less memory and are often preferred in scenarios where color information is unnecessary or adds complexity to the task at hand.

2. Thresholding:

   Thresholding operates on the principle of separating image regions or objects based on pixel intensity values, simplifying grayscale images into binary representations for subsequent analysis and processing tasks.

   First, Thresholding divides the grayscale image into regions based on pixel intensity levels. In this step, we should set a threshold value to classify pixels as either foreground or background, depending on their intensity compared to the threshold. Moving on, thresholding converts the grayscale image into a binary image where pixels are classified as either 0 (black) or 255 (white) based on a predefined threshold value.

   Pixels with intensities below the threshold are set to 0 (background), while those above or equal to the threshold are set to 255 (foreground). This process results in a binary image, where the foreground objects and the background have distinct pixel values.

3. Find Contours:

   In image processing, contours are outlines or boundaries of objects detected in an image. They are represented as a list of points that define the shape of an object. In this step, the boundaries of objects within the binary image can be detected. The principle involves identifying continuous curves or outlines that connect the points of the same color or intensity. It works by interpreting the image as a 2D array of pixel values and then searching for regions that share common characteristics or colors. It operates on binary images, where objects of interest are typically represented in white (255) and the background in black (0). The function scans through the image to find continuous areas of non-zero pixel values, thereby detecting and defining the contours of objects based on the contrast or differences in intensity levels.

4. Bounding Rectangle of Contour: The bounding rectangle of a contour is the smallest upright rectangle that can encapsulate the contour. It is defined by the coordinates of its top-left corner and its dimensions (width and height). The bounding rectangle is a simple and computationally efficient way to approximate the location and size of an object in an image. In this step, it finally returns the coordinates of the top-left corner of the rectangle and the dimensions of the rectangle for the final step of cropping.

5. Cropping:

   The last step is cropping the processed images. All the areas outside the bounded rectangle are removed. This is done by selecting only the pixels within the rectangle and discarding the rest. Based on the top-left corner and the dimensions (width and height), we can finally crop the image.

3 Experiments

3.1 Dataset

We conducted stitching demonstration tests using three sets of source images, each set containing two images with certain relevance and continuity between them. The second set of images are thumbnails of the first set.


Figure 4: (a) First Sets of Source Images


Figure 5: (b) Seconds Sets: thumbnail of the first sets


Figure 6: (c) Third Sets of Source Images

3.2 Code Description

In our project, we have a `main.py` file where we defined several auxiliary methods to implement specific functionalities, such as *detectAndDescribe* for keypoint and feature descriptor computation, *createMatcher* for matching these features between images, *getHomography* for calculating the homography matrix to align the images, and *crop* for

trimming the stitched image. We use these methods in a coherent workflow in the *stitchImages* function:

It starts with the conversion of images to grayscale, followed by feature extraction and alignment using the calculated homography matrix. For image blending, we then blend the overlapping regions using Gaussian blending, and finally merge the non-overlapping parts, culminating in the successful completion of the image stitching process.

We aim to evaluate the effectiveness of the two types of feature descriptors—SIFT and ORB—by comparing their performance in key tasks such as keypoint detection, feature description, and image matching within our workflow. This comparative analysis will provide insights into the trade-offs between the computational efficiency and the robustness of the matching process, guiding us in choosing the most suitable method for different stitching scenarios. The auxiliary methods in the code include:

- *detectAndDescribe*: Detects key points in the image and calculates their feature descriptors.
  - Input: An image and the chosen method for feature detection (such as SIFT or ORB).
  - Output: The key points of the image and their corresponding feature descriptors.
- *createMatcher*: Creates a matcher based on the selected method (for example, for SIFT, it creates a matcher based on the L2 norm).
  - *Input*: The chosen method for feature extraction and the option for cross-checking.
  - *Output*: A matcher object.
- *getHomography*: Finds matching points between the two sets of features, filters out good matches, and then calculates the homography matrix.
  - *Input*: method(custom),two sets of key points and feature descriptors, match ratio threshold (custom), and re-projection threshold (custom).
  - *Output*: The matching points, homography matrix, and status.

  Initially, it matches descriptors between these images using the matcher. Then, it applies Lowe's ratio test to filter out unreliable matches, ensuring only the best matches are retained. If there are more than four good matches, the function constructs two sets of points based on these matches and estimates the homography matrix with the RANSAC algorithm.
- *crop*: Crops out the excess border areas, retaining only the valid image content.
  - *Input*: The stitched image.
  - *Output*: The cropped image.

  It begins by converting the input image to grayscale, which simplifies the process of identifying the relevant contours of the image. Then, it applies a binary threshold to create a binary image where the non-zero pixels represent the areas of interest. Next, the function finds contours in this binary image, essentially detecting the edges or boundaries of the significant portions of the image. It then extracts the bounding rectangle of the first (and typically the largest) contour, which effectively represents the area of the stitched image that needs to be retained. Finally, the function crops the original image to this bounding rectangle, effectively removing any unnecessary or empty peripheral regions, and returns this cropped image.

3.3 Result

The final stitching results of the three sets of images are as follows:

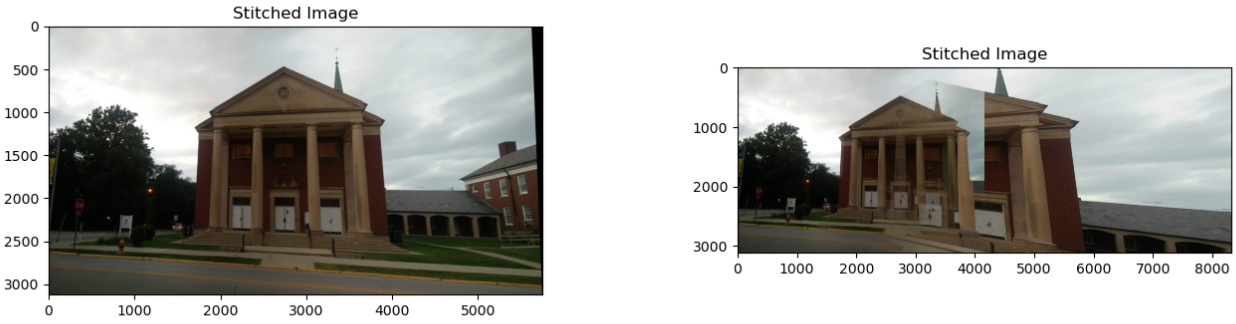

Figure 7: (a) SIFT and ORB

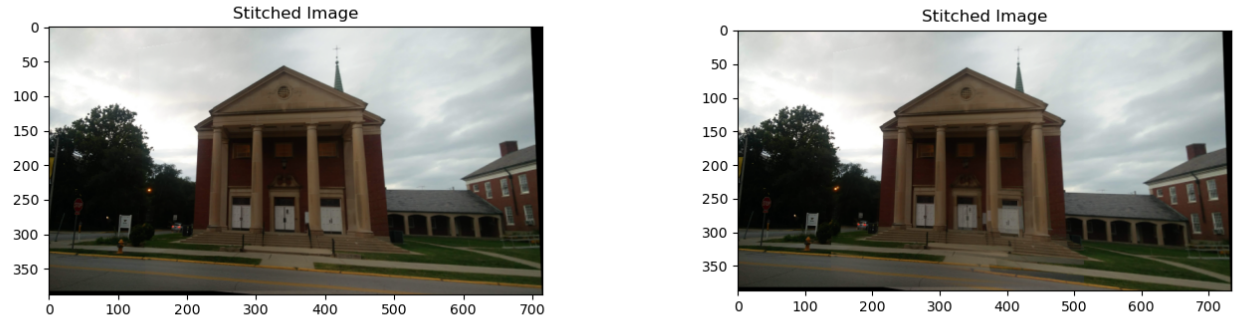

Figure 8: (b) SIFT and ORB

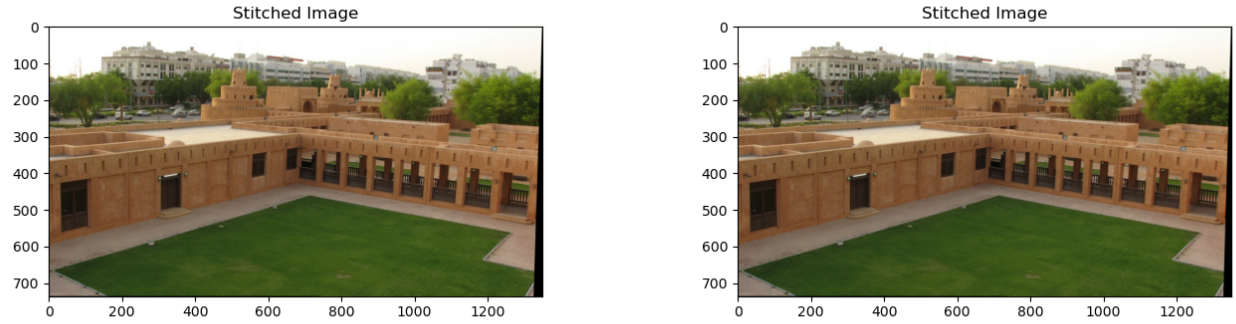

Figure 9: (c) SIFT and ORB

4 Discussion

In this project, we employed two popular feature matching algorithms: Scale-Invariant Feature Transform (SIFT) and Oriented FAST and Rotated BRIEF (ORB), to explore and compare their efficacy in the application of image

stitching. As illustrated in the figure below, the left image utilized the SIFT method, while the right image applied the ORB method.

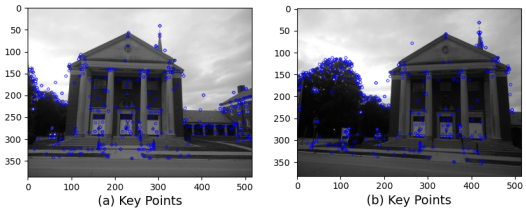

Figure 10: picture use SIFT

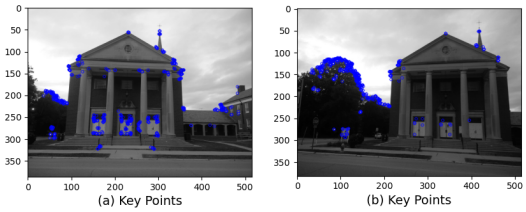

Figure 11: picture use ORB

Observing the performance of these two methods in keypoint detection, we can identify some critical differences. Firstly, the number of keypoints detected by the SIFT method in the left image is significantly greater than that by the ORB method in the right image. The distribution of SIFT features is also more uniform, covering the main parts of the building and the surrounding environment, which aids in establishing a more stable and precise correspondence between the two images.

In contrast, the ORB algorithm extracted fewer keypoints in the right image, with most concentrated in certain areas of the building. Although the computation speed of the ORB algorithm is usually faster than that of SIFT and it consumes less memory, in this example, the distribution of its keypoints may not be as uniform as that of the SIFT method, which could affect the quality of the final stitched image.

4.1 Challenges and Future Optimizations

In the results of image stitching, we observe some prominent issues, with color discrepancy being the most noticeable. This discrepancy often arises from inconsistencies in lighting, exposure, or camera settings among source images. In our stitched images, the color discrepancy manifests as incoherent colors in the joining regions, potentially distracting viewers and impacting the visual experience.

Moving forward, we plan to optimize several key aspects of the image stitching process to enhance the quality and visual appeal of the final panoramic images. We will focus on the following areas:

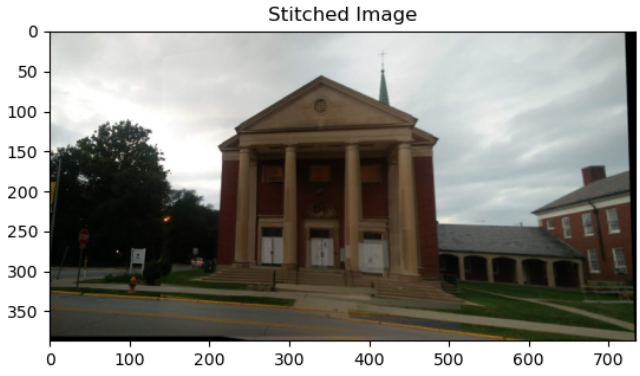

Figure 12: result with color discrepancy

Optimization of Image Fusion Techniques:

- **Seam Carving:** A content-aware image editing method that adjusts image sizes without significantly altering the content. It reduces visual discrepancies in the seam areas by finding and removing or inserting inconspicuous seams within the image.
- **Multi-band Blending:** This technique involves processing images at different spatial frequency layers to achieve a smooth and natural transition in the stitching region, particularly effective for resolving issues caused by significant contrast or color differences.
- **Laplacian Pyramid Blending:** A fusion method based on image pyramids that blends images at different scale levels, preserving detail while achieving smooth transitions in lower frequency levels.

Improvements in Image Preprocessing:

- **Exposure Compensation:** Adjusting brightness differences between images by analyzing image data, maintaining consistency in the luminance of the stitched panoramic image.
- **Color Correction:** Implementing color correction algorithms to balance colors across different images, ensuring natural color transitions and reducing discrepancies.
- **White Balance Adjustment:** Applying white balance to the images to eliminate color temperature variations, especially between images taken at different times.

Extensions to Panoramic Stitching Algorithms:

- **Cylindrical Projection:** For images captured by professional cameras, particularly those with a high focal length, cylindrical projection can be used to reduce projection distortion. This method maintains the linearity of features such as buildings, which is especially suitable for scenes rich in linear characteristics.
- **Projection and Perspective Adjustment of Panoramic Images:** Further research into different projection methods, such as spherical projection, and how changing the viewpoint and perspective can optimize the visual effects of panoramic images.

The slow processing speed and suboptimal results when dealing with large images constitute a challenge in our project. This issue is particularly evident when employing the SIFT and ORB algorithms, as demonstrated by the two figures (which are the original, larger-sized versions of the previously discussed images). The left image, utilizing SIFT, detects a higher number of keypoints but at the cost of longer processing time and higher computational expense. The right image, using ORB, processes quickly but performs substantially worse on large images, barely detecting effective keypoints.

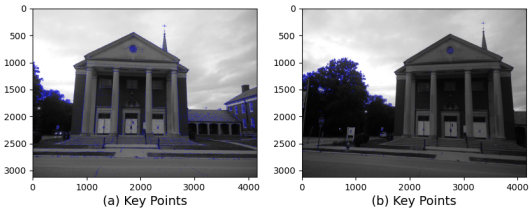

Figure 13: big picture use SIFT

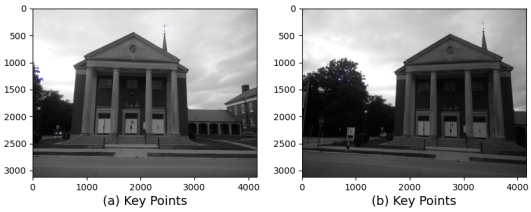

Figure 14: big picture use ORB

Inspecting the result images, the use of ORB directly failed. To address the issues of slow processing speeds and unsatisfactory results with large images, we will explore different feature extraction algorithms and performance optimization techniques for improvement. The following are some potential strategies for optimization:

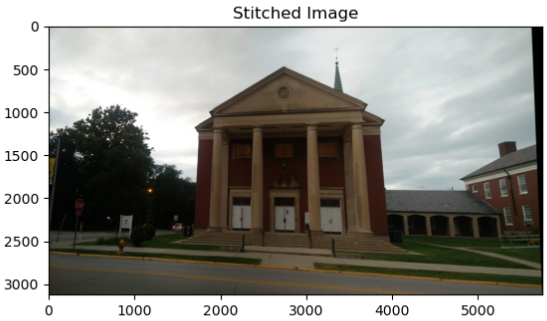

Figure 15: result picture use SIFT

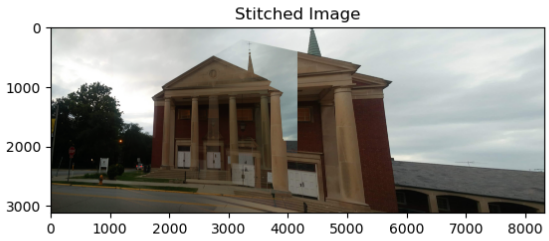

Figure 16: result picture use ORB(failed)

Feature Extraction Algorithms:

- **SURF (Speeded Up Robust Features):** An accelerated version of SIFT, which may be more effective for quickly processing large images while maintaining invariance to rotation and scale.
- **AKAZE (Accelerated KAZE Features):** This algorithm optimizes the non-linear scale space of images, enhancing computational efficiency while maintaining a high detection rate.

Performance Optimization:

- **Image Pyramids:** Processing images at different resolution levels allows for rapid detection of features at lower resolutions, followed by precise matching at higher resolutions.
- **Intelligent Keypoint Selection:** After feature extraction, keypoints are filtered by evaluating their quality (such as their response magnitude) to select those most likely to provide good matches, thus reducing the computational load during the matching process.