Image Stitching

Khaled Ehab Attia	20106166
Youssef Hatem Mohamed	20108429

CS Students

Arab Academy for Science and Technology and Maritime Transport.

College of Computing and Information Technology.

December 17th, 2023

Image Stitching: Merging Multiple Views into One

Imagine capturing the breathtaking expanse of a mountain range or the bustling energy of a city street. A single photo might not do it justice. That's where image stitching comes in! It's the art of combining multiple overlapping images to create a seamless, panoramic view.

Think of it like piecing together a puzzle. Each image is a piece, and the software stitches them together to reveal the bigger picture. This opens up exciting possibilities for photography, filmmaking, and even scientific applications.

Here's a quick overview of the process:

1. Capture: Take multiple overlapping photos of your scene. Aim for at least 30% overlap between each image for smooth stitching.
2. Align: Software analyzes the images to find matching features, like edges or corners. This ensures the pieces fit together correctly.
3. Blend: The overlapping areas are carefully blended to hide the seams and create a natural-looking panorama.

Benefits of Image Stitching:

- Capture wider fields of view than a single camera can.
- Create high-resolution images by stitching together multiple smaller ones.
- Add a sense of immersion and depth to your photos and videos.

Applications:

- Panoramic photography: Capture stunning landscapes, cityscapes, and indoor spaces.
- Virtual tours: Create immersive experiences for real estate, travel, and education.
- Surveillance and security: Monitor large areas with overlapping camera views.
- Medical imaging: Stitch together X-rays or MRIs for a more complete view.

In conclusion, image stitching is a powerful technique for merging multiple viewpoints into a single, immersive image. It allows us to capture scenes beyond the limitations of a single camera lens, creating panoramic vistas, high-resolution images, and even interactive experiences. From capturing breathtaking landscapes to building virtual tours, the possibilities are endless. So, grab your camera, embrace the overlap, and start stitching together your own world!

Table of Contents

1. Introduction

1.1 Background and Context

Stitching images require more than one image with an over lapping area between two or more images. Such an operation can be done in real life by cutting images and aligning them to each other, or by using software programs like Adobe Photoshop, Microsoft Image Composite Editor, Autopano Giga, and more endless software. Image stitching can also be done by using open source software like OpenCV and Hugin. This technique has been widely used in various applications such as virtual reality, surveillance, and photography. However, traditional image stitching methods often produce visual artifacts caused by intensity discrepancy and structure misalignment, which can result in a lack of coherence and continuity in the final stitched image.

1.2 Objective

- Achieving seamless image stitching: Develop a technique that can seamlessly combine multiple input images into a single panoramic image.
- Simultaneous alignment of intensity and structure: Propose a method that can align both the intensity and structure of the input images, ensuring coherence and continuity in the final stitched image.

1.3 Scope and Limitations

1- Designing and implementing an algorithm that can simultaneously align the intensity and structure of input images to achieve seamless stitching results.
2- Providing an output for the method we used to stitch images, highlighting it’s power points in stitching complex images.
3- The method we used may have limitations in handling extremely complex images.
4- Ordering the input images is very important as the algorithm is one sided, it means that the algorithm starts by one image and accumulate the rest of the input images into one direction, it can work both ways but it require human guidance to order the left side of the view and the right side of the view.
5- The effectiveness of the algorithm varies by factors such as image quality, resolution, exposure, and the degree of misalignment in the input images.

1.4 Report Structure

- **Literature Review:** This section provides an overview of the existing literature on image stitching, covering various techniques used in image stitching, including feature detection, image alignment, and blending. It also discusses the challenges and open extensions in the field of image stitching.
- **Methodology:** This section outlines the design process, coding practices, data acquisition, and testing method used in the project. It also provides technical details on the system architecture, algorithm explanation, and code implementation.
- **Results and Discussion:** This section presents the analysis of results, performance evaluation, and challenges faced during the project. It also compares the proposed approach with existing methods and discusses the strengths and limitations of the technique.
- **Conclusion:** This section summarizes the key points, findings, and contributions of the project, emphasizing the significance of image stitching and its potential for diverse applications. It also highlights the need for innovative algorithms and methodologies to tackle the complexities of modern imaging scenarios.
- **References:** This section lists the sources cited in the paper, providing readers with a comprehensive list of relevant literature on image stitching.

2. Literature Review

Image Stitching using Structure Deformation [1] presents a new approach to image stitching that ensures seamless transitions between images without producing visual artifacts. The algorithm is based on structure deformation and propagation, and can be used for image compositing, blending, and intensity correction. The approach reduces ambiguity by deriving structure matching from 2D to 1D, making salient feature detection and matching more robust. The algorithm can handle images with significant mismatches in the overlapped area and does not assume camera motion or deformation models. There are Image Stitching Techniques [2] that discusses the process of image stitching. The paper covers various techniques used in image stitching, including feature detection, image alignment, and blending. It also discusses the challenges and open extensions in the field of image stitching. Overall, the paper provides a comprehensive overview of the basics of image stitching and its applications.

2.1 Image Stitching Sequence

Feature Detection and Matching often relies on detecting and matching features between overlapping images. The theory of feature detection involves identifying distinctive points or regions in images, while feature matching theory focuses on establishing correspondences between these features in different images.

Geometric Transformation: Image stitching involves aligning and blending multiple images to create a seamless composite. The theory of geometric transformation encompasses concepts such as homography, affine transformations, and perspective correction, which are essential for aligning images accurately during the stitching process.

Optimization and Seam Finding: Some image stitching methods involve optimizing the blending or alignment of images to minimize visual artifacts. The theory of optimization and seam finding includes techniques such as dynamic programming, graph cuts, and energy minimization, which are used to find optimal seams or blending paths in the overlapping regions of images.

Structure Deformation and Propagation: In the context of the proposed approach, the theory of structure deformation and propagation is relevant to achieving seamless image stitching by simultaneously aligning image structures and intensities. This theory involves propagating deformation cues and minimizing energy in the gradient domain to ensure consistent alignment across images.

Context Consideration: The theory of context consideration in image stitching involves taking into account the contextual information of objects in the input images during the stitching process. This includes considering the overall scene layout, object relationships, and contextual consistency to achieve natural and visually pleasing stitching results.

2.2 Gap Analysis

Traditional image stitching methods often fail to align the structures or features of input images, resulting in ghosting or blurring artifacts. Existing image stitching methods may have limited applicability in handling complex input images with significant color or intensity inconsistency. Some existing image stitching methods may require manual intervention, such as defining control points or selecting optimal seams. Image Stitching methods may have limitations in handling extremely complex or highly dynamic scenes.

3. Methodology

3.1 Design Process

3.1.1 Algorithm Design

Algorithms used in this project are:

1. cvtColor function to convert images to (RGB, BGR, gray scale)
2. Oriented FAST and Rotated BRIEF (ORB) algorithm for feature detection and description.
3. Brute Force matcher algorithm to find the best matches between the two images.
4. Sorting Algorithm to sort matches (built-in python).
5. RANSAC algorithm for finding perspective transformation.
6. Warp perspective algorithm to apply the transformation to generate the result image.
7. Bounding Rectangle algorithm to find the boundaries of the stitched image.

3.1.2 Coding Practices

We used python programming language to develop this program that stitches two different related images into one image. Inside the code 3 libraries were used to reach the goal of stitching images.

Libraries used:

- **OpenCV:** used for image manipulation and it's an open-source project that is free and available to the public. It offers a rich set of functions for image and video processing, computer vision, and machine learning.
- **NumPy:** is short for Numerical Python, a library that provides support for large or multi-dimensional arrays like images. It provides numerical operations, linear algebra operations, statistical operations, and more. It also provides utility to easily control and manage arrays.
- **Matplotlib:** a comprehensive 2D plotting library for python Programming Language. It enables the creation of a wide variety of static, animated, and interactive plots in Python.

3.2 Data Acquisition

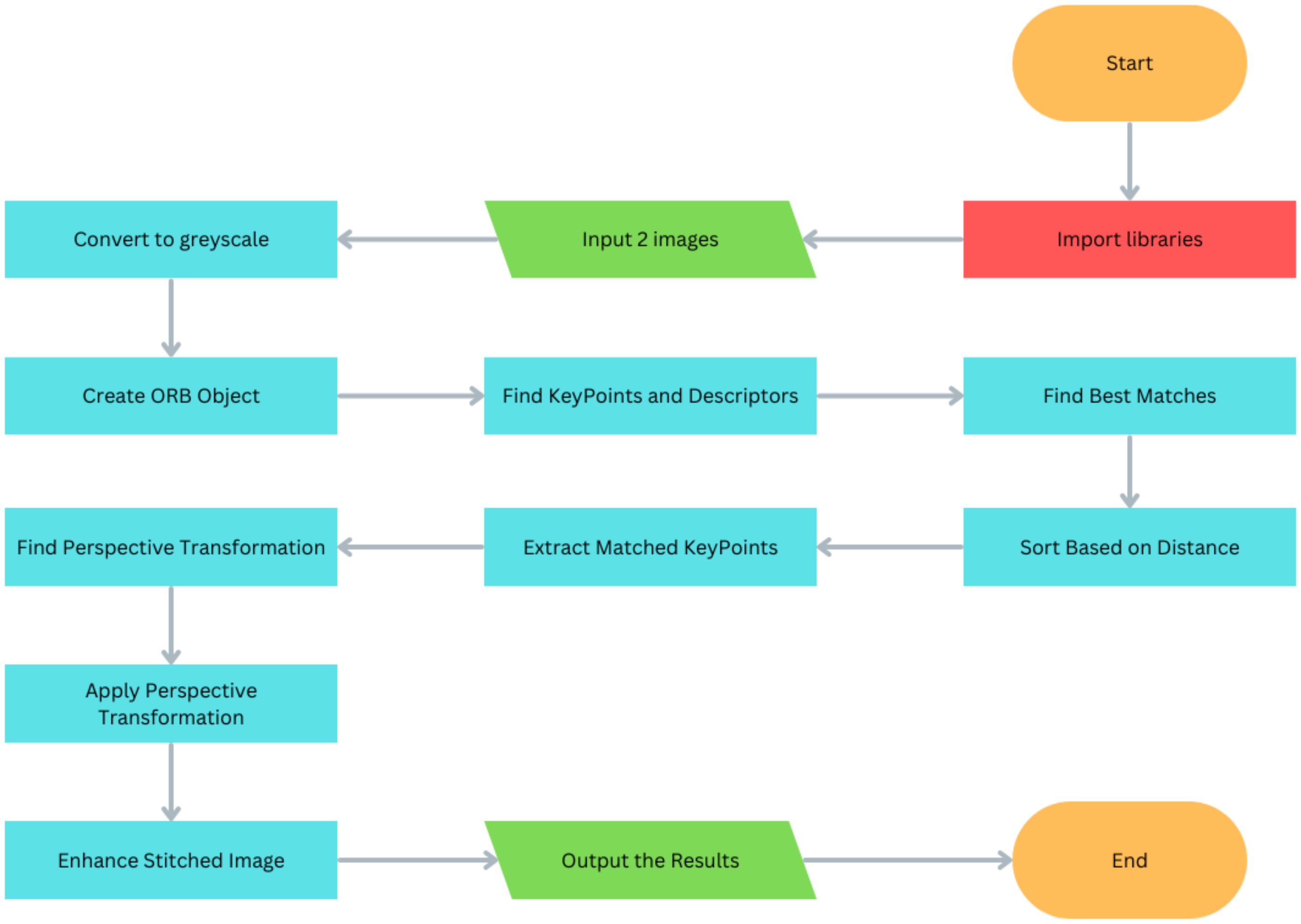
Our program doesn't require a data-set or big number of images to achieve it's goal, instead we used some landscape images to test the code, we also tested the code on freshly taken images in real time. Only some of the images needed some preprocessing to match the colors and the exposure of the second image, and turning the images into greyscale to enhance feature detection (key points and descriptors) efficiency.

3.3 Testing Method

Testing the validity of stitched images demands human surveillance to the results of the program. There is a flaw in the output image that leads to a black area in the image that needs to be removed, it's treated with a function that finds the non-zero pixels and bound the image to provide the coordinates of the real image without the black area, long story short, the image is cropped to show only the real image.

4. Technical Details

4.1 System Architecture:



The provided code implements image stitching using the OpenCV library in Python. The system can be divided into several key components:

4.1.1 Image Loading and Display

Two images, 'image1' and 'image2,' are loaded using OpenCV.

The images are then converted from BGR to RGB color space.

The horizontally concatenated images are displayed using Matplotlib.

4.1.2 Image Grayscale Conversion

Grayscale versions of the original images are created.

Both grayscale images are displayed side by side using Matplotlib.

4.1.3 Feature Detection and Description

The ORB (Oriented FAST and Rotated BRIEF) feature detector and descriptor are created using cv2.ORB_create().

4.1.4 Homography Estimation

The homography matrix (M) is calculated using the cv2.findHomography function, employing the RANSAC algorithm.

4.1.5 Image Warping

The warped image (result) is obtained by applying the homography matrix to one of the images.

The second image is then overlaid on the first to create a stitched result.

4.1.6 Image Cropping

A function crop_image is defined to crop the black borders from the stitched image.

The function converts the stitched image to grayscale, finds non-zero pixels, and determines the bounding rectangle.

The bounding rectangle is adjusted to exclude black borders, and the image is cropped accordingly.

4.1.7 Result Display

The original images and the cropped stitched image are displayed side by side using Matplotlib.

4.2 Algorithm Explanation:

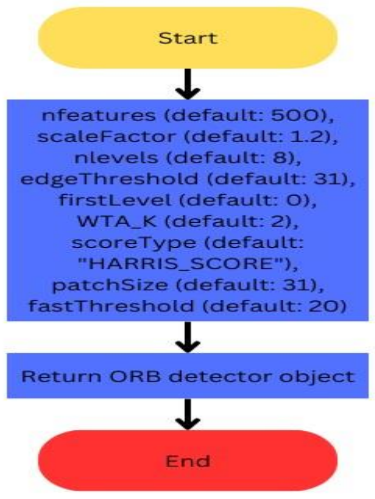
1-ORB Feature Detection:

The ORB algorithm is used to detect and describe key points in the grayscale images.

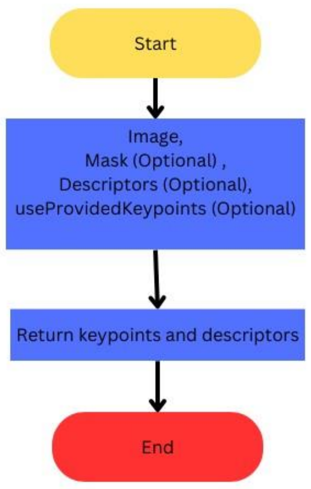
Key points and descriptors are computed for both grayscale images.

- **keypoints** The detected keypoints. A 1-by-N structure array with the following fields:
  - **pt** coordinates of the keypoint `[x,y]`
  - **size** diameter of the meaningful keypoint neighborhood
  - **angle** computed orientation of the keypoint (-1 if not applicable); it's in [0,360) degrees and measured relative to image coordinate system (y-axis is directed downward), i.e in clockwise.
  - **response** the response by which the most strong keypoints have been selected. Can be used for further sorting or subsampling.
  - **octave** octave (pyramid layer) from which the keypoint has been extracted.
  - **class_id** object class (if the keypoints need to be clustered by an object they belong to).
- **descriptors** Computed descriptors. Output concatenated vectors of descriptors. Each descriptor is a 32-element vector, as returned by cv.ORB.descriptorSize, so the total size of descriptors will be `numel(keypoints) * obj.descriptorSize()`, i.e a matrix of size `N-by-32` of class `uint8`, one row per keypoint.

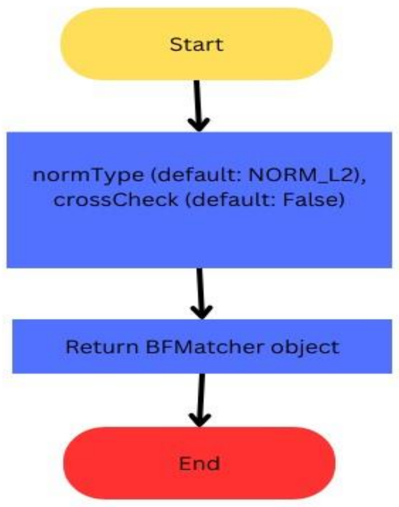
cv2.ORB_create()


Start
nfeatures (default: 500),
scaleFactor (default: 1.2),
nlevels (default: 8),
edgeThreshold (default: 31),
firstLevel (default: 0),
WTA_K (default: 2),
scoreType (default: "HARRIS_SCORE"),
patchSize (default: 31),
fastThreshold (default: 20)
Return ORB detector object
End

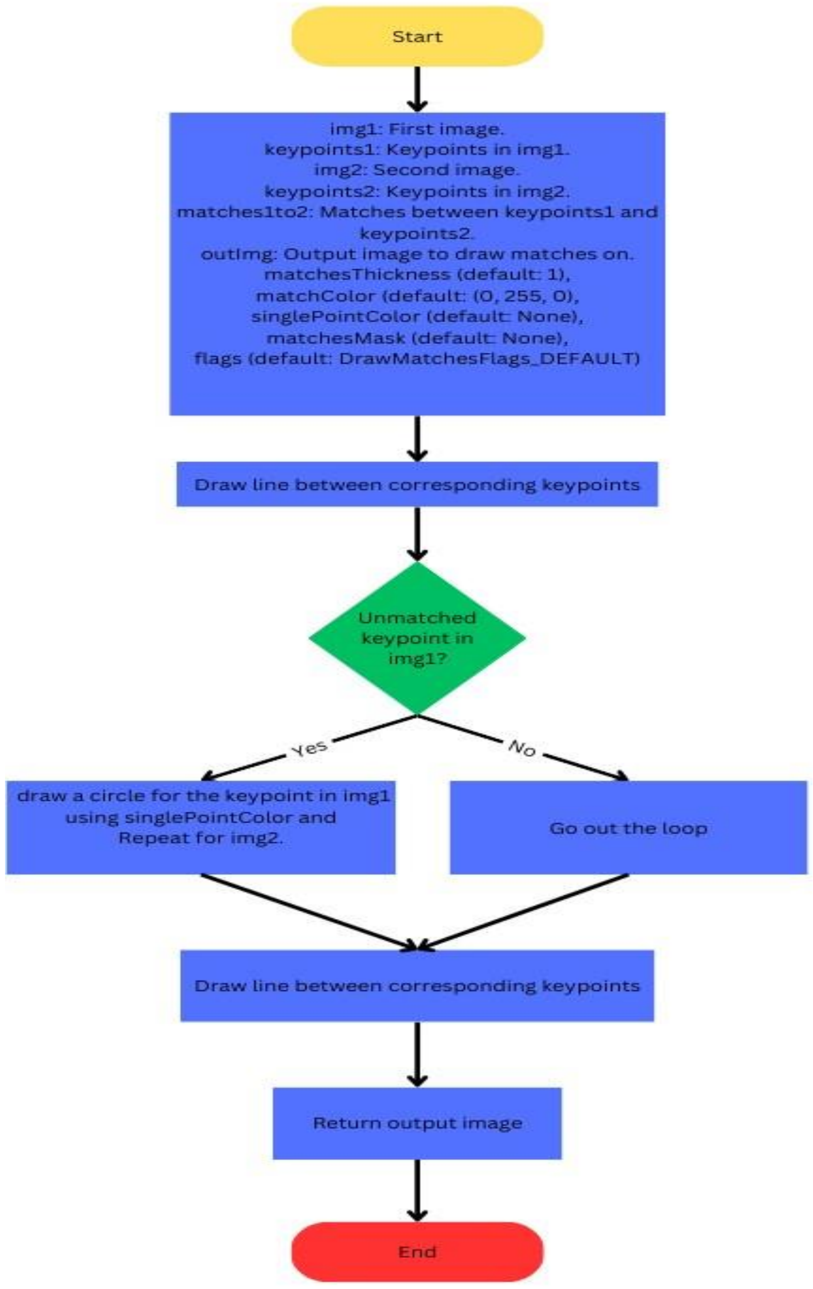
orb.detectAndCompute()


Start
Image,
Mask (Optional) ,
Descriptors (Optional),
useProvidedKeypoints (Optional)
Return keypoints and descriptors
End

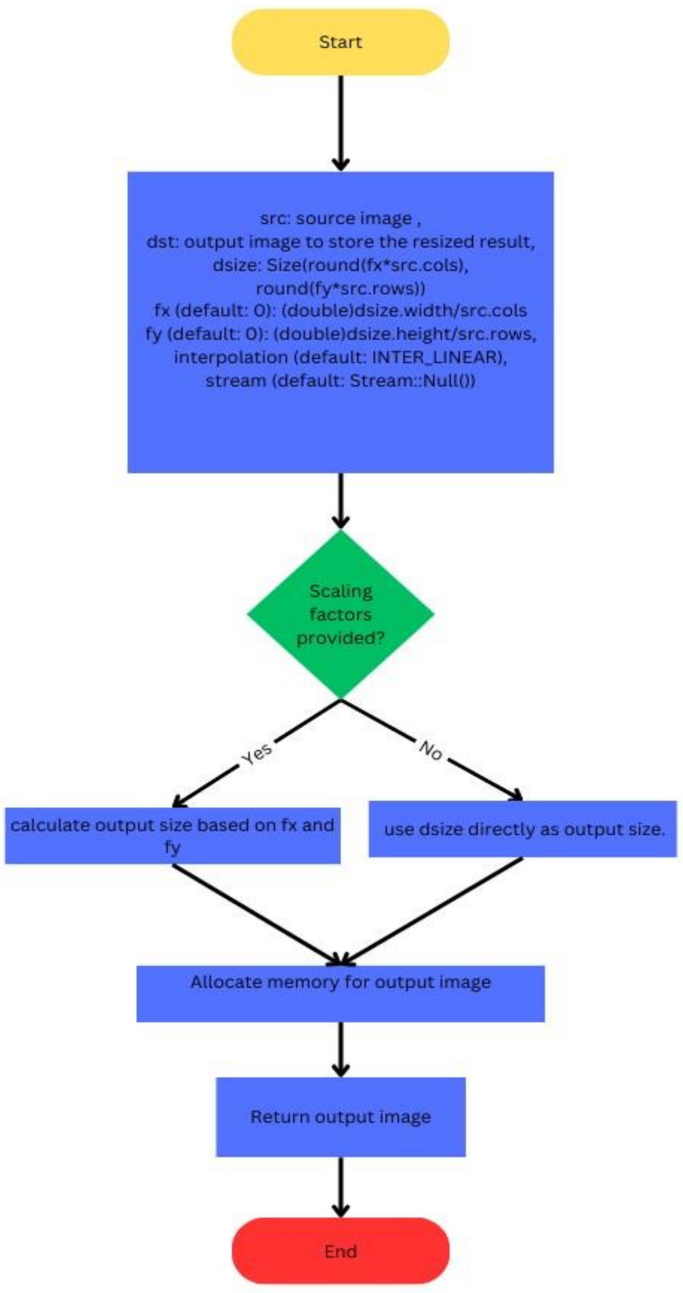
BFMatcher()



drawMatches()

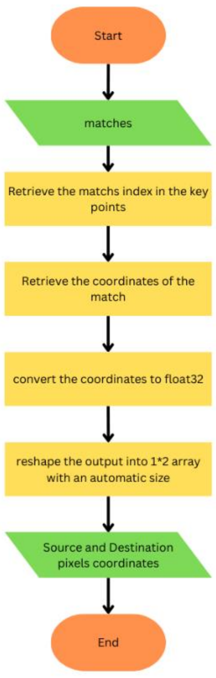

Start
img1: First image.
keypoints1: Keypoints in img1.
img2: Second image.
keypoints2: Keypoints in img2.
matches1to2: Matches between keypoints1 and keypoints2.
outImg: Output image to draw matches on.
matchesThickness (default: 1),
matchColor (default: (0, 255, 0),
singlePointColor (default: None),
matchesMask (default: None),
flags (default: DrawMatchesFlags_DEFAULT)
Draw line between corresponding keypoints
Unmatched keypoint in img1?
Yes
No
draw a circle for the keypoint in img1 using singlePointColor and Repeat for img2.
Go out the loop
Draw line between corresponding keypoints
Return output image
End

resize ()


Start
src: source image ,
dst: output image to store the resized result,
dsize: Size(round(fx*src.cols), round(fy*src.rows))
fx (default: 0): (double)dsize.width/src.cols
fy (default: 0): (double)dsize.height/src.rows,
interpolation (default: INTER_LINEAR),
stream (default: Stream::Null())
Scaling factors provided?
Yes
No
calculate output size based on fx and fy
use dsize directly as output size.
Allocate memory for output image
Return output image
End

2-Extracting Matches:

1. **matches**: This presumably contains the matches between keypoints in two images. Each match object (**m**) likely has information about the corresponding keypoints in the two images.
2. **keypoints1[m.queryIdx]**: For each match (**m**), **queryIdx** is the index of the keypoint in the first image (**keypoints1**). This retrieves the keypoint from the first image.
3. **keypoints1[m.queryIdx].pt**: This extracts the (x, y) coordinates of the keypoint.
4. **[keypoints1[m.queryIdx].pt for m in matches]**: This is a list comprehension that iterates through all matches and creates a list of (x, y) coordinates of keypoints from the first image.
5. **np.float32(...)**: This converts the list of coordinates to a NumPy array with data type float32.
6. **.reshape(-1, 1, 2)**: This reshapes the array to have three dimensions: the first dimension is determined automatically based on the size of the array, the second dimension is set to 1, and the third dimension is set to 2. This is often done to prepare the array for further processing or use in functions that expect a specific shape.


Start
matches
Retrieve the matchs index in the key points
Retrieve the coordinates of the match
convert the coordinates to float32
reshape the output into 1*2 array with an automatic size
Source and Destination pixels coordinates
End

2-Homography Estimation:

The homography matrix is estimated using the RANSAC algorithm, which is robust to outliers in matching points.

This line of code is using the `cv2.findHomography` function from the OpenCV library to find a perspective transformation matrix (`M`) that maps points from the source image (`src_points`) to points in the destination image (`dst_points`). The RANSAC algorithm is employed to robustly estimate the transformation matrix, and a threshold of 5.0 is specified.

1. `cv2.findHomography`: This function is used to find the perspective transformation matrix (`M`) between two sets of points.
2. `src_points`: The source points are the coordinates of keypoints in the first image, typically obtained through a feature matching algorithm as in your previous code.
3. `dst_points`: The destination points are the corresponding coordinates of keypoints in the second image.
4. `cv2.RANSAC`: This is the robust method used for estimating the transformation matrix. RANSAC stands for Random Sample Consensus, and it is used to deal with outliers in the matching points. It randomly samples a subset of points, estimates a model (transformation matrix in this case), and then evaluates how well the model fits the remaining points. This process is repeated to find the best model.
5. `‘5.0’`: This is the reprojection threshold used by RANSAC. Points that are further away than this threshold from the projected points will be considered as outliers and will not contribute to the estimation of the transformation matrix.
6. `M, mask`: The function returns two values:
   - M: The estimated perspective transformation matrix.
   - mask: A binary mask indicating which points were inliers (1) and outliers (0) according to the RANSAC algorithm.

3-Image Warping:

The cv2.warpPerspective function is employed to warp one of the images based on the calculated homography matrix.

1. **cv2.warpPerspective**: This function applies a perspective transformation to an image.
2. **image1**: This is the source image that will be transformed.
3. **M**: This is the perspective transformation matrix obtained from the previous step using **cv2.findHomography**.
4. **(image1.shape[1] + image2.shape[1], image1.shape[0])**: This is the size of the output image. The width is set to the sum of the widths of **image1** and **image2**, and the height is set to the height of **image1**. This ensures that the output image is wide enough to contain both images side by side.
5. **result**: The transformed image is stored in this variable.

4-Image Cropping:

The crop_image function crops the stitched image to remove black borders, ensuring a clean result.

1. cvtColor(img, cv2.COLOR_BGR2GRAY): turning the image into grayscale to prepare it.
2. findNonZero(gray_img): find all pixels that are not black to define the image boundaries
3. boundingRect(coords): calculating the image boundaries xmin, xmax, ymin, ymax.

4.3 Code Implementation:

The code utilizes OpenCV for image processing tasks, Matplotlib for visualization, and NumPy for array manipulation.

ORB is used for feature detection and description.

Homography is estimated using the RANSAC algorithm.

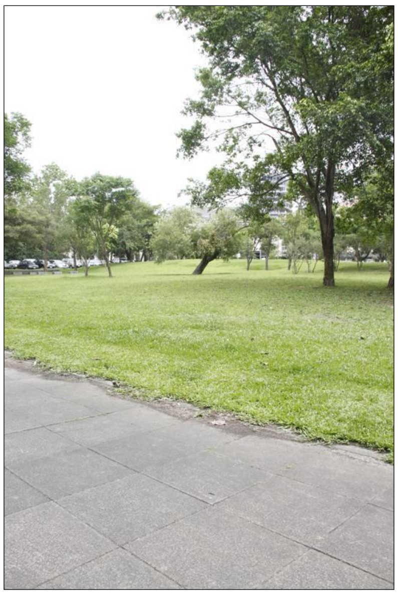
The final stitched image is obtained by warping and overlaying the original images.

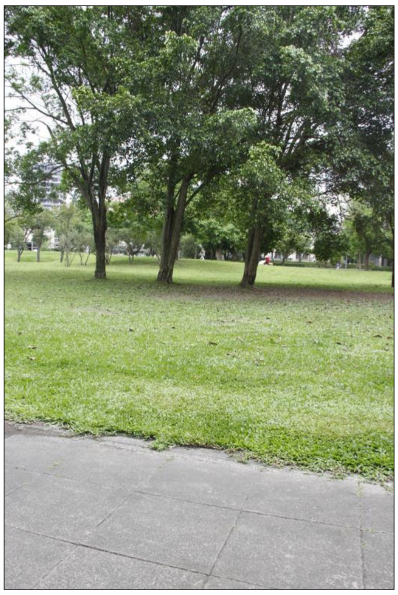
The crop_image function ensures the resulting image is free of black borders.

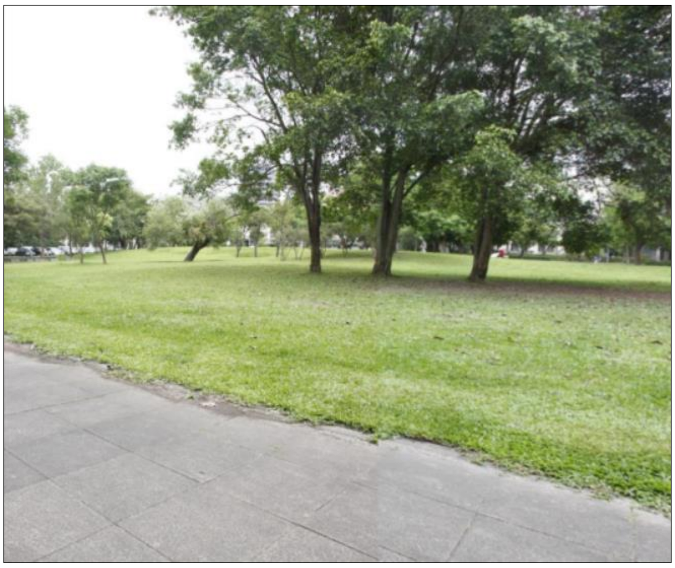
5. Results and Discussion

5.1 Analysis of Results

As shown in figure 1, 2, 3. The program takes 2 images and stitch them together to create panoramic views.


Source Image 1


Source Image 2


Output Stitched Image

5.2 Performance Evaluation

The image stitching technique is very fast relatively to stitching multiple images in the same view or even frames taken from video source.

There is no metrics to describe the performance of the algorithm for now.

5.3 Challenges and Resolution

5.3.1 Alignment

The output stitched images may have a not aligned part in it like a line or a rectangular vertical line, it may be noticeable in some images and not very clear in other images. This problem is addressed by blurring the stitched image.

5.3.2 Histogram Equalization

Another problem here is the difference in colors between the 2 source images. This problem is addressed with histogram equalization or histogram matching.

5.3.3 Image Direction

The images should be entered from the user in a specific direction as follows:

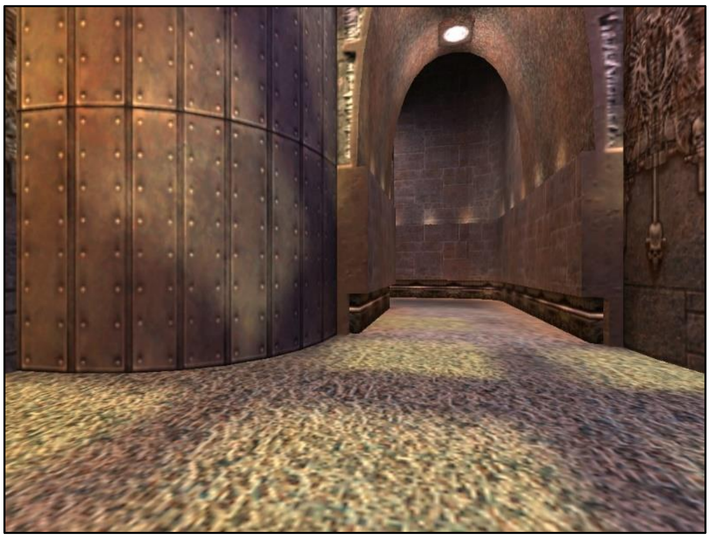
Image 1 : the right part

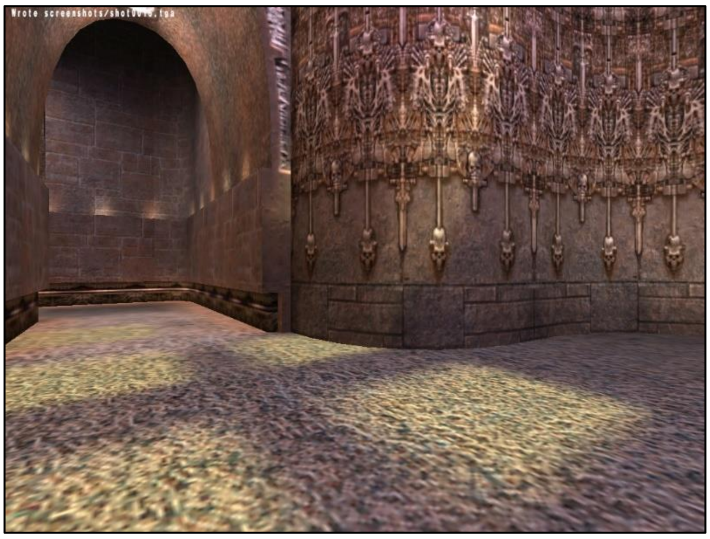
Image 2 : the left part

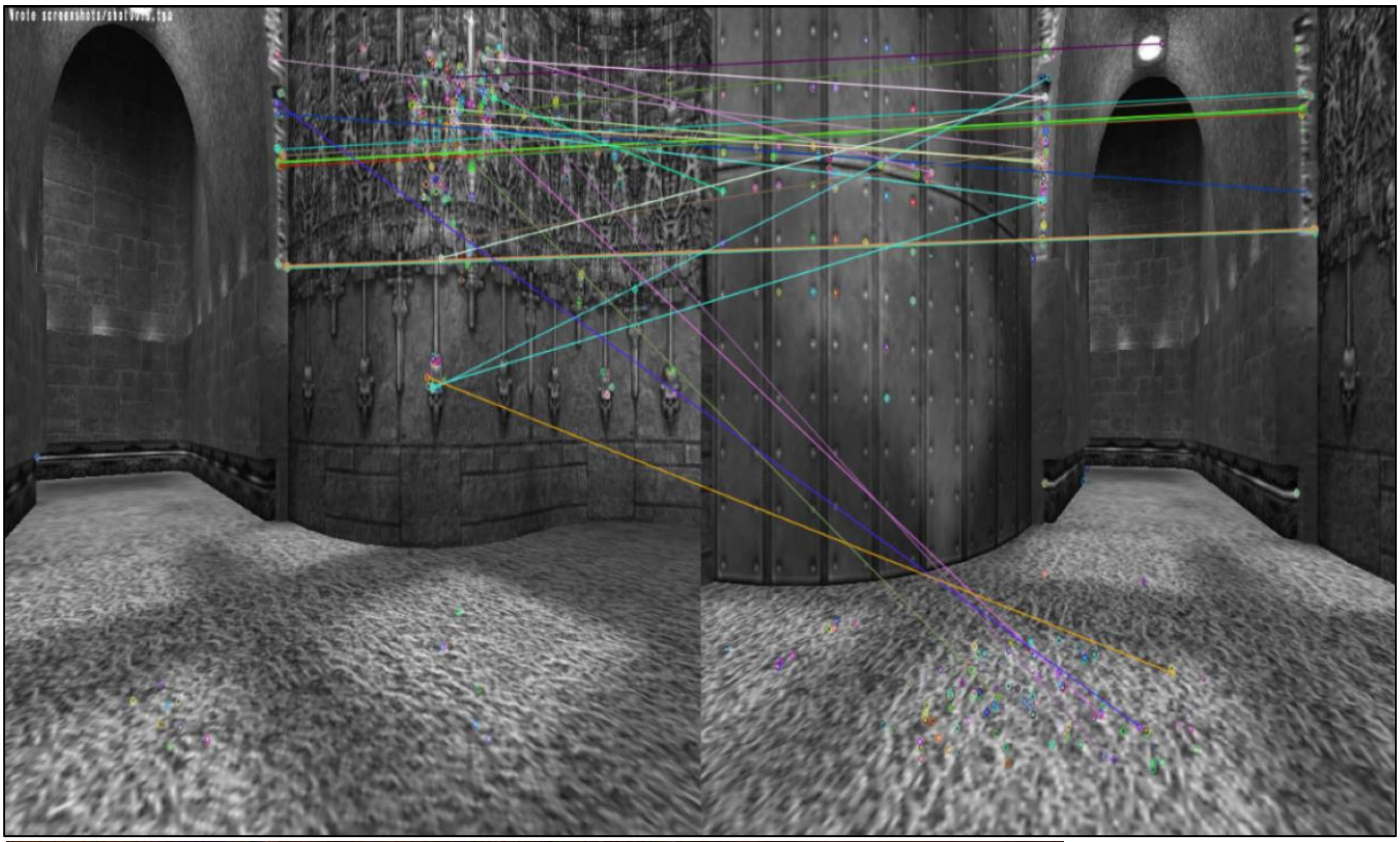
We couldn't handle this program from our side, so we require the end user to input the images correctly as prompted.

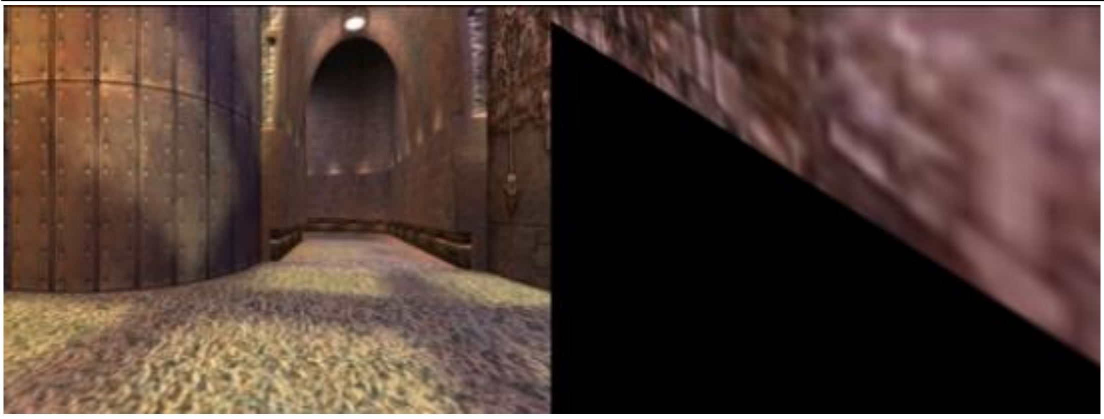
5.3.4 Complex Patterns

The code fails to determine where to stitch an image with a lot of similar patterns in it.









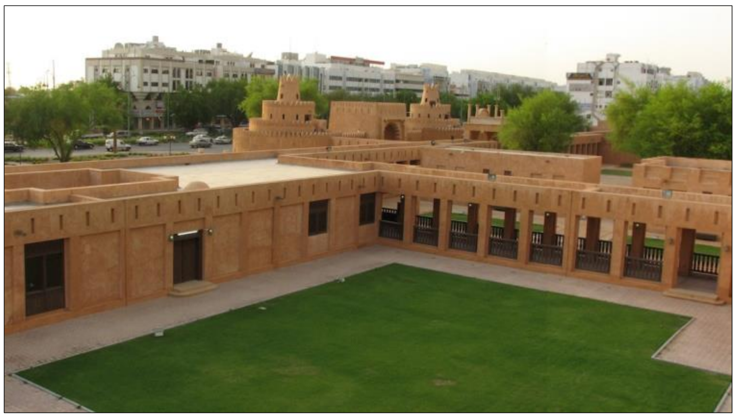
The program fails to stitch the two images together as there were similar patterns in different area in the image.

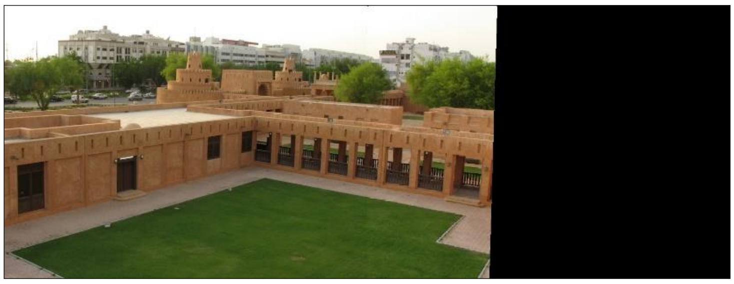
5.4 Comparison with Existing Methods

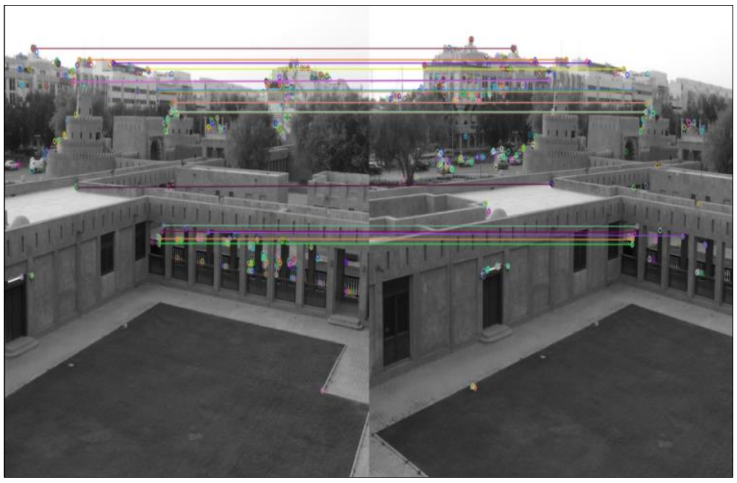
<u>Comparison 1</u> with a model that uses SURF to detect and describe features

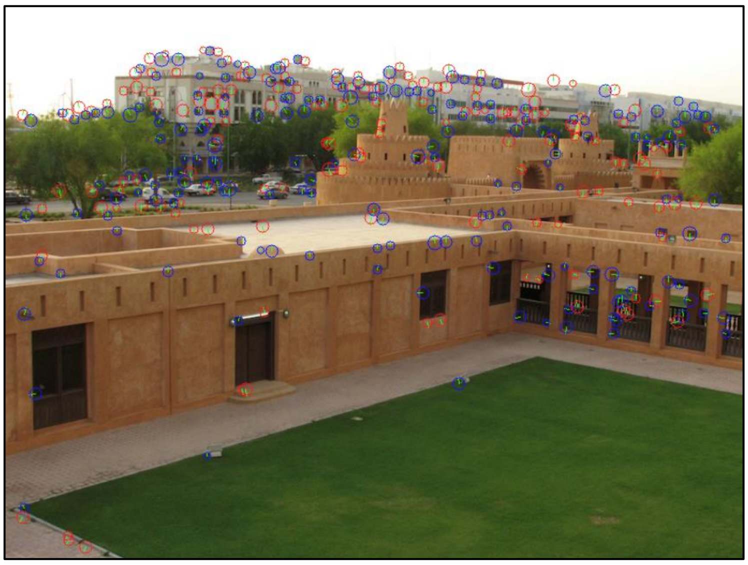
Source Image 1

Source Image 2


Stitched Image : our model


Stitched Image : using xfeatures2d

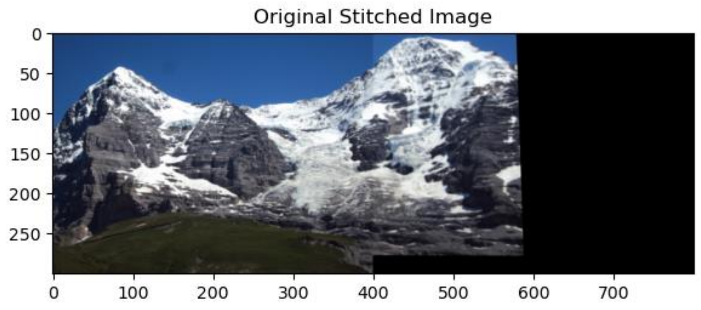

Matches : our model

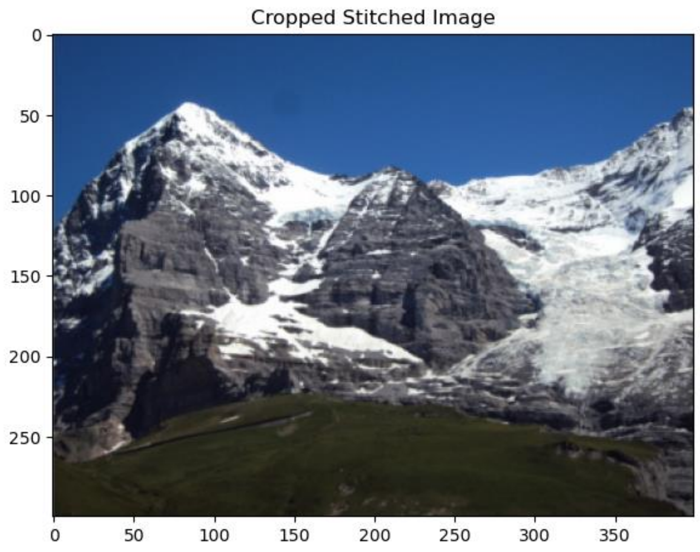

Matches : xfeatures2d

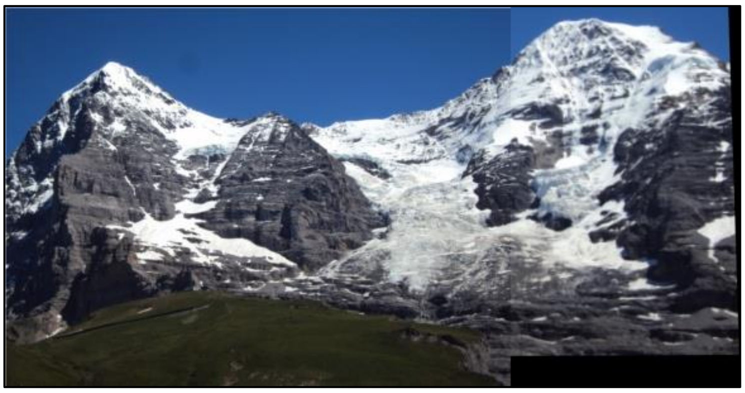
Comparison 2 with a model made with c++ that uses AGAST and FREAK feature detector

Source Image 1

Source Image 2






Stitched Image using AGAST and FREAK

The stitched image is almost the same in both programs but our enhanced cropping technique shows no black pixels that are not part of the image.

6. Conclusion

6.1 Summary of Work

This project has delved into the realm of image stitching, presenting a comprehensive exploration of the process, challenges, and solutions. Key points include the significance of capturing overlapping images with careful attention to exposure, stability, and consistent visual elements, as well as the limitations and scope of traditional image stitching methods. The findings underscore the importance of achieving seamless image stitching through the simultaneous alignment of intensity and structure, addressing issues of coherence and continuity in the final stitched image. The project's contributions lie in shedding light on the power of image stitching to create panoramic vistas, high-resolution images, and immersive visual experiences, while also highlighting the need for innovative algorithms and methodologies to tackle the complexities of modern imaging scenarios.

6.2 Achievements and Objectives

The project aimed to achieve seamless image stitching by developing a technique that could seamlessly combine multiple input images into a single panoramic image while aligning both the intensity and structure of the input images. The objectives also included providing an output highlighting the strengths of the method used in stitching complex images, despite potential limitations in handling extremely complex images. The effectiveness of the algorithm was acknowledged to vary based on factors such as image quality, resolution, exposure, and the degree of misalignment in the input images. In evaluating the project's achievements against these objectives, it can be observed that the technique successfully addressed the simultaneous alignment of intensity and structure, contributing to the creation of visually coherent and immersive panoramic images. However, the project also recognized the potential limitations in handling extremely complex images, thereby acknowledging the need for further advancements in algorithmic approaches to address such challenges.

6.3 Future Directions

We are looking forward to work on videos, and multiple images to form a single panoramic view next. Enhancing the cropping technique and automatically adjusting the alignment of the stitched images. Making an algorithm that works on all four directions (up – down – right – left) instead of one (right to left).

7. References

[1] M. I. Jiaya Jia and S. M. I. C. S. Chi-Keung Tang, "Image Stitching using Structure Deformation," IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, 2008.

[2] J. D. Mehta and S. G. Bhirud, "Image Stitching Techniques," Department of Computer Technology, VJTI, Mumbai, India.