Computer Vision：

Panorama generation from videos

Even

Chapter 1: Objectives and Key Functionalities

Objectives

The aim of this project is to develop a software system that is capable of automatically generating high quality panoramas from short videos provided. The main objectives of the system include:

Ease of use: the system should be simple and easy to operate for non-technical users.

Adaptability: it should be able to handle videos taken under different environmental conditions.

High performance: The generated panoramas should have high resolution and good visualization without visible seams or distortion.

Real-time processing capability: The system should be able to complete image processing and stitching within a reasonable time, providing fast feedback.

To achieve these goals, the project tries to use advanced computer vision techniques from the OpenCV library, including feature point detection, image matching and image fusion algorithms. SIFT and ORB algorithms are especially integrated to improve matching accuracy and stitching results. Additional features such as sharpening and Gaussian blurring are also planned to be included, as well as an interactive interface for real-time feature matching preview to optimize the ultimate utility of the project.

Key Features

Basic features

1. **Video frame extraction:**

   The system is capable of automatically extracting key frames from user-supplied video files for subsequent image processing by using OpenCV library.

2. **Feature point detection and matching:**

Key points in video frames are detected and accurately matched using Scale Invariant Feature Transform (SIFT) and Orientation Rapid and Rotation Invariant Feature (ORB) algorithms [1]. This is a critical step in ensuring that images are stitched together correctly.

3. **Image alignment and stitching:**

According to Rosebrock, use OpenCV's Stitcher class can stitch images [2]. This is done in the stitch frames method, which first creates a Stitcher instance and then passes in a list of frames to stitch. If the stitching is successful, it returns a stitched panorama.

Additional features

Image optimization aspects

During the generation of the panoramas, various image processing techniques were used to optimize the visual appearance of the final image. These techniques include:

1. **Gaussian blurring:**

In order to reduce image noise and smooth out details, we selectively apply Gaussian blurring in the image preprocessing stage. This blurring helps to improve the accuracy in the feature matching step, especially in video frames with uneven lighting or too much detail.

2. **Image Sharpening:**

By applying a sharpening filter kernel, we enhance the edges and details of the image, making the objects in the panorama clearer. Sharpening is especially effective in improving the visual quality of low-contrast video frames.

3. Black Edges Removal:

When stitching panoramas, edge mismatches can lead to black borders. We implemented an automatic black border removal function that uses a mask to remove unwanted black edges after image erosion and thresholding to ensure the visual continuity of the panorama.

The integration of these image processing techniques not only optimizes the quality of the images, but also improves the adaptability of the algorithms to different scenes, resulting in visually smoother and more detailed panoramas.

The integration of these features ensures that the system can perform well in different scenarios and meet a wide range of user needs.

User Interface Design and Implementation

In order to provide an intuitive and easy-to-use operating environment, the system allows the user to easily manage the panorama generation process through a graphical

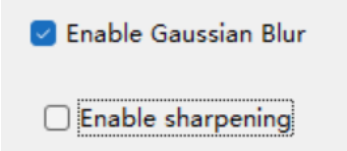
user interface (GUI). The GUI was designed with the following key aspects in mind:

1. **Algorithm selection:**

Users can select either the "SIFT" or "ORB" algorithm for feature detection and matching via a drop-down menu. This provides flexibility and allows the user to select the most appropriate algorithm based on the specific video content and the required processing accuracy.

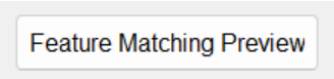
2. **Gaussian Blurring and Sharpening Processing Options:**

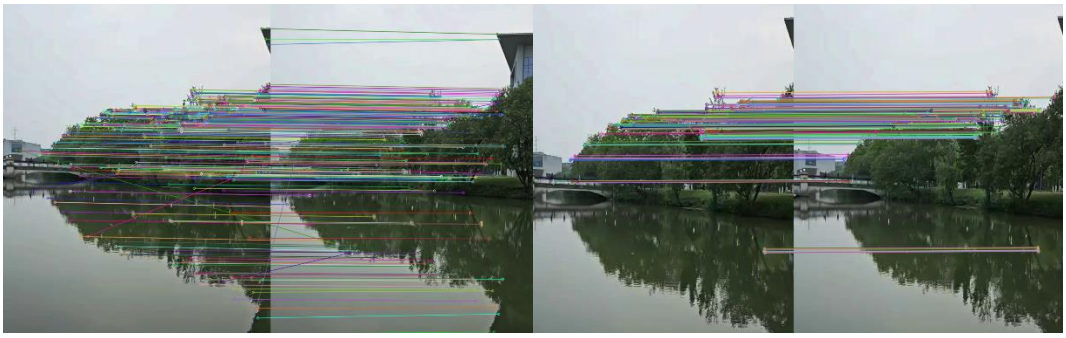
The checkboxes in the interface allow the user to choose whether to apply Gaussian Blurring to reduce image noise or sharpening to enhance image details. These processing options help users to adjust the image quality according to their needs, making the final panorama generated more in line with expectations.



3. **Feature Matching Preview:**
4.

The system provides a feature matching preview function that allows users to observe the feature matching between two frames before actual processing [3]. This not only increases the interactivity of the system, but also allows the user to intuitively understand the effect of feature matching and thus have more control and expectation over the panorama generation process.





With these features, the GUI not only simplifies the operation process, but also enhances the user's control over the panorama generation process, making the system more user-friendly.

Chapter 2: Detailed Steps and Techniques Employed

Basic features

1. **Video Frame Extraction**

   Use OpenCV's `cv2.VideoCapture() ` method to read the video file, the system automatically analyses the video and extracts the key frames according to the set time interval. This step is the basis of subsequent image processing, ensuring processing efficiency and quality of results.

```
import cv2
```

```python
def extract_frames(self, step=30, apply_blur=False, callback=None):
    try:
        cap = cv2.VideoCapture(self.video_path)
        if not cap.isOpened():
            raise Exception("Unable to open the video file.")
        frames = []
        idx = 0
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            if idx % step == 0:
                adjusted_frame = self.adjust_frame(frame)
                if apply_blur:  # 应用高斯模糊，根据传递的参数决定
                    adjusted_frame = self.apply_gaussian_blur(adjusted_frame)
                frames.append(adjusted_frame)
                if callback:
                    callback(len(frames), int(total_frames / step))
            idx += 1
    except Exception as e:
        messagebox.showerror("Error", f"Failed to process the video: {str(e)}")
    finally:
        cap.release()
    return frames
```

2. Feature Point Detection and Matching

The extracted key frames are detected and matched with feature points using Scale Invariant Feature Transform (SIFT) and Orientation Rapid and Rotation Invariant Feature (ORB) algorithms. The system initializes the feature detectors with `cv2.SIFT_create () ` and `cv2.ORB_create () `, computes the feature descriptors for each image frame, and accurately matches the feature points using the 'FLANNBasedMatcher' (For ORB, using the 'BFMatcher'). This is the key to achieving high quality image alignment and stitching.

```python
def detect_and_match(self, frame1, frame2):
    algo = self.algo_var.get()


    if algo == "SIFT":
        detector = cv2.SIFT_create()
        FLANN_INDEX_KDTREE = 1
        index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
        search_params = dict(checks=50)
        matcher = cv2.FlannBasedMatcher(index_params, search_params)
    elif algo == "ORB":
        detector = cv2.ORB_create()
        matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)


    keypoints1, descriptors1 = detector.detectAndCompute(frame1, None)
    keypoints2, descriptors2 = detector.detectAndCompute(frame2, None)


    if descriptors1 is None or descriptors2 is None:
        raise Exception("Descriptors cannot be None.")


    if algo == "SIFT":
        matches = matcher.knnMatch(descriptors1, descriptors2, k=2)
        # Lowe's ratio test
        good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
    elif algo == "ORB":
        matches = matcher.match(descriptors1, descriptors2)
        good_matches = sorted(matches, key=lambda x: x.distance)[:30]


    matched_image = cv2.drawMatches(frame1, keypoints1, frame2, keypoints2, good_matches, None, flags=2)
    return matched_image
```

3. Image alignment and stitching

Image stitching is implemented by the 'stitch_frames' method of the 'VideoProcessor' class, which uses OpenCV's Stitcher class. First a Stitcher instance is created based on the OpenCV version. Then, the list of frames to be stitched is passed to the stitch method of the Stitcher, which tries to stitch all the frames into a panorama. If status returns cv2.Stitcher_OK, the stitching is successful.

```python
def stitch_frames(self, frames):
    if not frames:
        raise Exception("No frames are available for stitching")
    stitcher = cv2.createStitcher() if cv2.__version__.startswith('3') else cv2.Stitcher_create()
    status, stitched = stitcher.stitch(frames)
    if status != cv2.Stitcher_OK:
        raise Exception(f'Unable to stitch image, error code:{status}')

    stitched = self.remove_black_edges(stitched)
    return stitched
```

Additional features

Image optimization techniques

1. Gaussian blurring:

Gaussian blurring is applied at the image preprocessing stage based on the user's choice, implemented as the `cv2.GaussianBlur() ` function, with the aim of reducing image noise and smoothing out details to improve the accuracy of feature matching[4].

```python
def apply_gaussian_blur(self, frame, kernel_size=(3, 3), sigmaX=0):

    return cv2.GaussianBlur(frame, kernel_size, sigmaX)
```

2. Image sharpening:

Apply a custom sharpening filter kernel to enhance image details, implemented as `cv2.filter2D ()`, especially to improve the visual quality of low-contrast video frames.

```python
def sharpen_frame(self, frame):
    kernel = np.array([[-1, -1, -1],
                       [-1,  9, -1],
                       [-1, -1, -1]])
    sharpened = cv2.filter2D(frame, -1, kernel)
    return sharpened
```

3. Remove black edges:

Remove mismatched black borders from panoramas using image erosion and binarization, using masking techniques to ensure visual continuity of the panorama.

```python
def remove_black_edges(self, image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

    kernel = np.ones((3,3), np.uint8)
    eroded = cv2.erode(binary, kernel, iterations=1)

    image[eroded == 0] = (0, 0, 0)
    return image
```

```python
def crop_panorama(self, pano):
    stitched = cv2.copyMakeBorder(pano, 0, 0, 0, 0, cv2.BORDER_CONSTANT, (0, 0, 0))
    gray = cv2.cvtColor(stitched, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)

    leftTopPt = None
    for i in range(1, thresh.shape[0]): ...

    rightTopPt = None
    for i in range(1, thresh.shape[0]): ...

    leftBottomPt = None
    for i in range(1, thresh.shape[0]): ...

    rightBottomPt = None
    for i in range(1, thresh.shape[0]): ...

    topMaxY = max(leftTopPt[1], rightTopPt[1])
    leftMaxX = max(leftTopPt[0], leftBottomPt[0])
    rightMinX = min(rightTopPt[0], rightBottomPt[0])
    bottomMinY = min(leftBottomPt[1], rightBottomPt[1])

    leftTopPt = (leftMaxX, topMaxY)
    rightTopPt = (rightMinX, topMaxY)
    leftBottomPt = (leftMaxX, bottomMinY)
    rightBottomPt = (rightMinX, bottomMinY)

    tempMat = pano[leftTopPt[1]:rightBottomPt[1], leftTopPt[0]:rightBottomPt[0]]

    return tempMat
```

User Interface Design and Implementation

The GUI is built using Python's Tkinter library and provides the following functionality:

```python
class PanoramaApp:
    def __init__(self, window, window_title): ...

    def setup_gui(self): ...

    def toggle_sharpen(self):
        self.sharpen_enabled = not self.sharpen_enabled

    def preview_video(self): ...

    def toggle_blur(self): ...

    def show_help(self): ...


    def stitch_video_async(self): ...

    def stitch_video(self): ...

    def enable_controls(self): ...

    def update_progress(self, frame_idx, total_frames): ...

    def open_video(self): ...


    def show_panorama(self, panorama): ...

    def create_panorama_window(self, panorama, save_path): ...


    def show_feature_matches(self): ...

    def display_image_in_gui(self, img): ...
```

1. **Algorithm selection:**

   Drop-down menus allow the user to select either the SIFT or ORB algorithm.

```python
self.algo_var.set("SIFT")
self.algo_dropdown = ttk.OptionMenu(self.frame, self.algo_var, "SIFT", "SIFT", "ORB")
self.algo_dropdown.grid(row=7, column=0, padx=10, pady=10)
```

2. **Image processing selection:**

   Checkboxes allow the user to enable or disable Gaussian blurring and sharpening.

```python
def toggle_sharpen(self):
    self.sharpen_enabled = not self.sharpen_enabled

def toggle_blur(self):
    self.blur_enabled = not self.blur_enabled
```

3. **Real-time feature matching preview:**

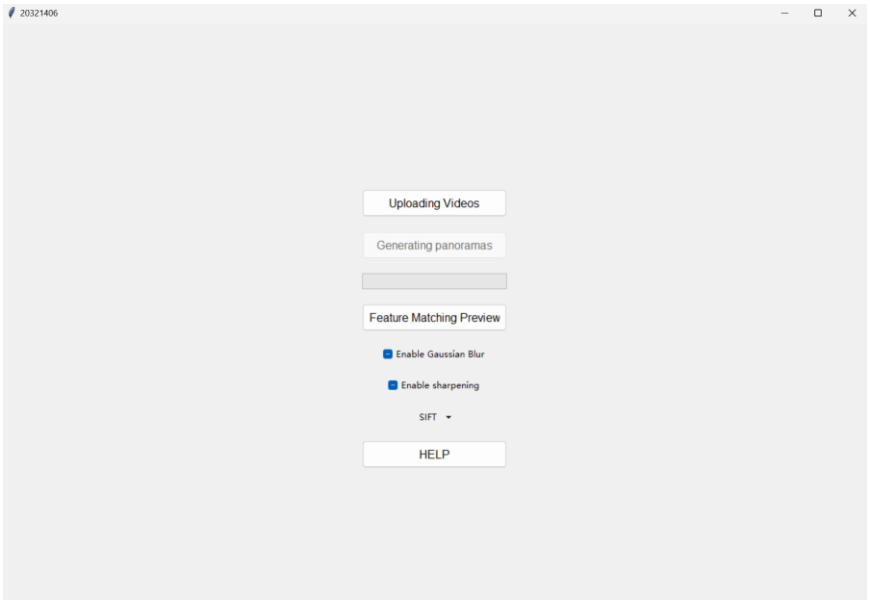
   Shows the feature matching process in real-time to help users understand the matching effect.

```python
def preview_video(self):
    if not hasattr(self, 'video_path') or not self.video_path:
        messagebox.showerror("Error", "Please upload a video file first.")
        return

    cap = cv2.VideoCapture(self.video_path)
    if not cap.isOpened():
        messagebox.showerror("Error", "Unable to open the video file.")
        return


    algo = self.algo_var.get()
    if algo == "SIFT":
        detector = cv2.SIFT_create()
        FLANN_INDEX_KDTREE = 1
        index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
        search_params = dict(checks=50)
        matcher = cv2.FlannBasedMatcher(index_params, search_params)
    elif algo == "ORB":
        detector = cv2.ORB_create()
        matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    ret, last_frame = cap.read()
    if not ret:
        messagebox.showerror("Error", "There are no frames in the video file to read.")
        return
    last_keypoints, last_descriptors = detector.detectAndCompute(last_frame, None)


    preview_window = tk.Toplevel(self.window)
    preview_window.title("Feature Matching Preview")
    label = tk.Label(preview_window)
    label.pack()
```

```python
def show_feature_matches(self):
    try:
        frame1, frame2 = self.video_processor.extract_two_frames()
        matched_image = self.video_processor.detect_and_match(frame1, frame2)
        self.display_image_in_gui(matched_image)
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

4. **Progress display and status feedback:**

   Progress bar and status label provide real-time feedback to enhance user experience.



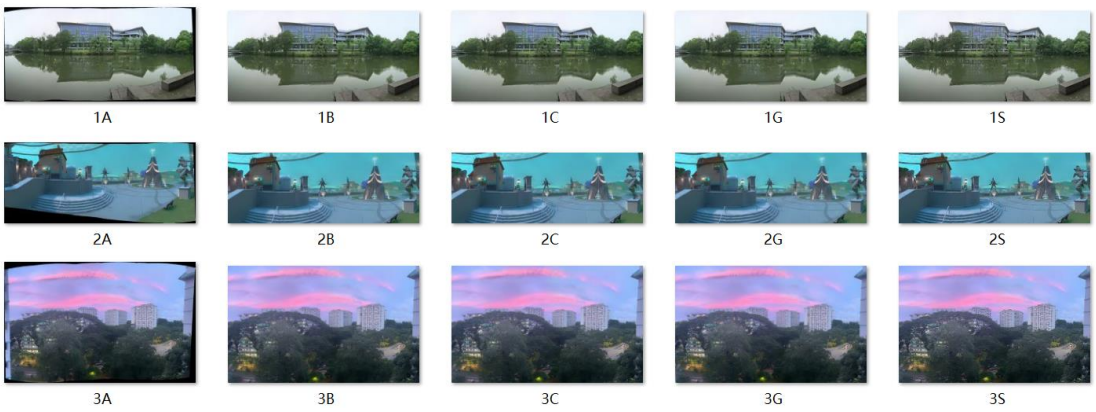
Chapter 3: Results Presentation and Analysis

Test Videos Overview

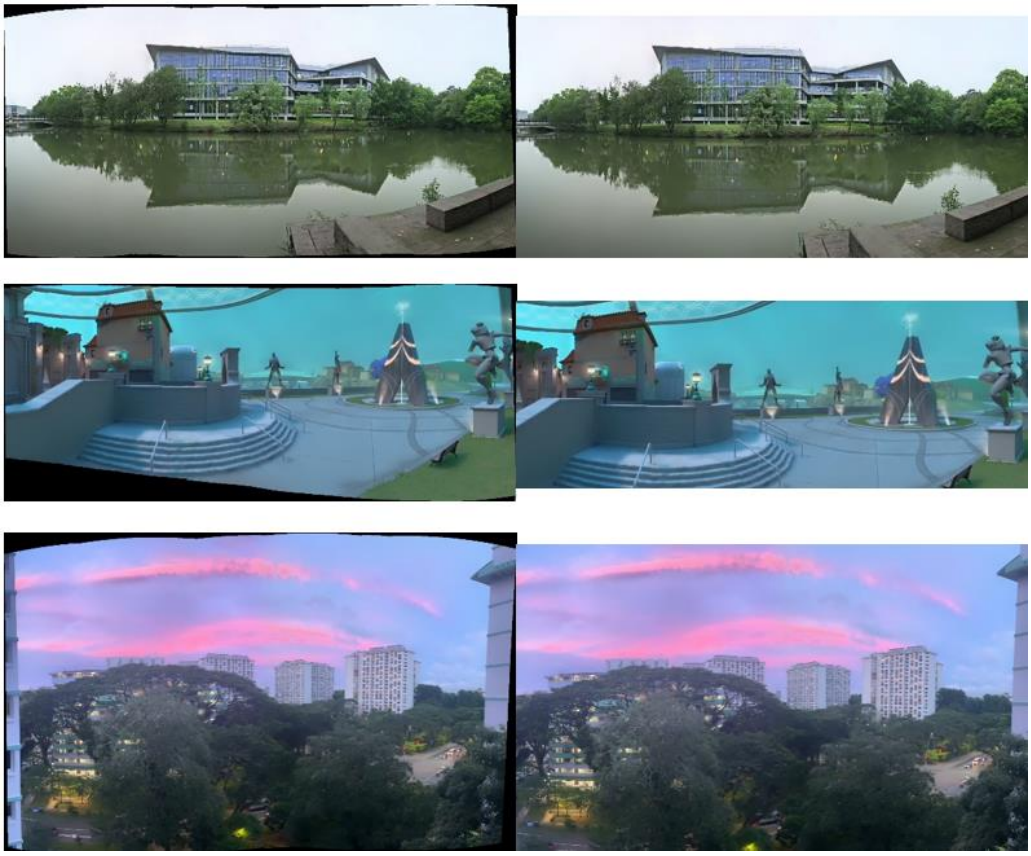
40-second video of a real scene: a video of the exterior of a library.

36-second cartoon game scene video: a shot of a game scene (‘*Valorant’*) to try out the stitching effect of a video shot in a virtual rendered environment.

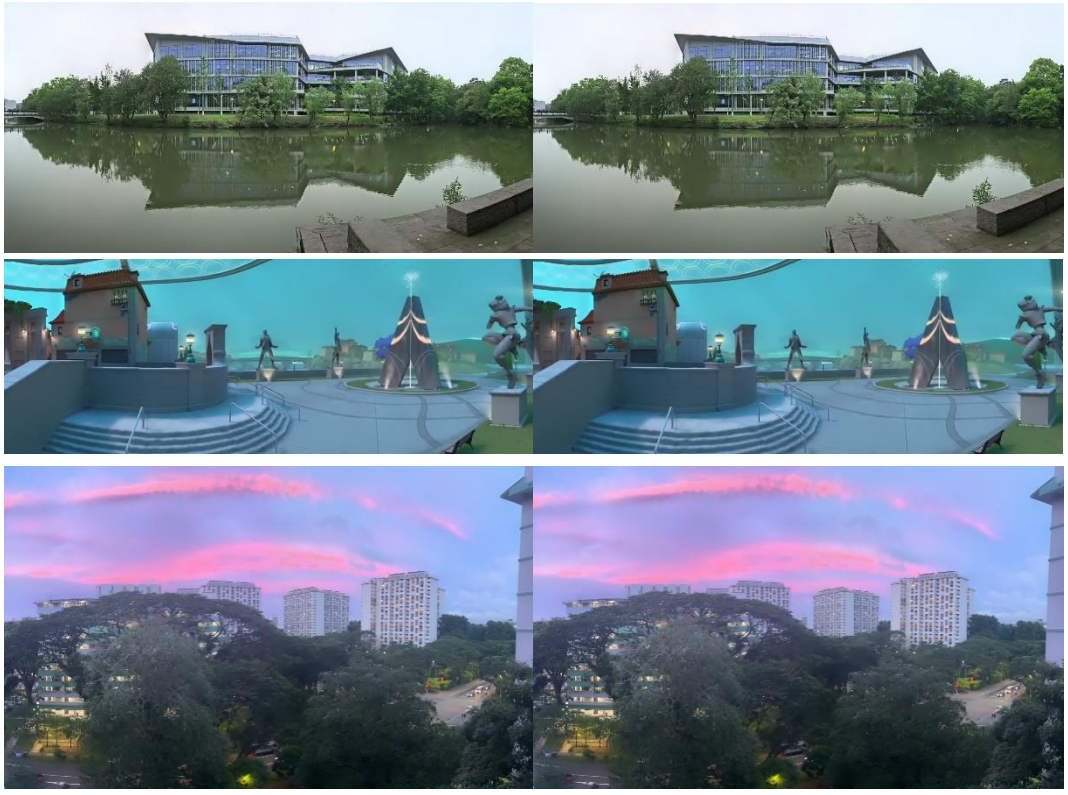
7-second Real Scene short video: a video of a landscape with a low frame rate.



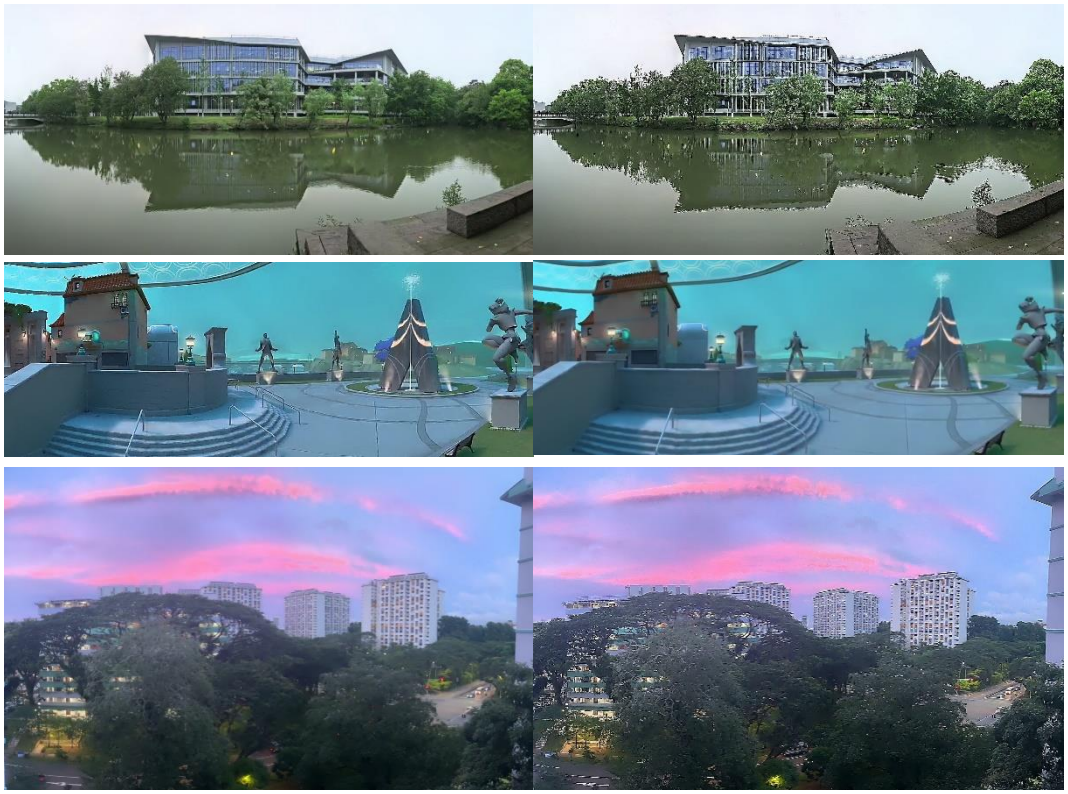
Test Results



Non-functional trimming / Remove Black edges



SIFT / ORB



Sharpen / Gaussian Blur

Quality analysis of results

1.Unprocessed version (left)

Black borders visible at image edges:

Imperfect alignment during image stitching or discontinuities in the field of view result in areas at the edges of the image that are not covered by the image content, which appear as black borders.

Visual continuity is affected:

The presence of black borders visually interrupts the continuity of the scene and may affect the viewing experience, especially in panoramic images or wide fields of view.

Black border removal version (right)

Black edges are effectively removed:

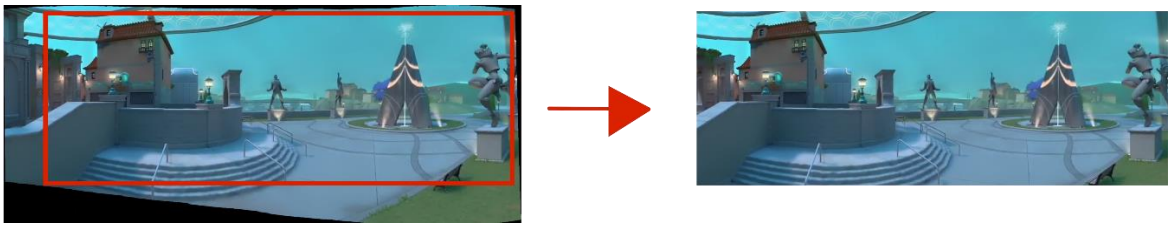
Image processing techniques, such as image erosion and masking, are applied to remove the black edges created during the stitching process.

Improved visual continuity:

After removing the black edges, the visual continuity of the panorama is restored, making the overall image neater and the visual experience smoother.

Improved overall image aesthetics:

By removing the visually distracting elements (black edges), the overall aesthetics of the image is improved, especially in the more detailed treatment of the edge areas.



2.SIFT algorithm and ORB algorithm stitching results

Visual coherence:

In both images, whether using the SIFT or ORB algorithm, the spliced panoramas show good visual coherence, with no obvious seams or distortions observed by the naked eye.

Feature matching performance:

Since both algorithms can effectively identify and match key feature points in the image, the panoramas generated by them are visually similar in these two examples, with no obvious difference.

Image processing performance:

Both algorithms successfully achieve smooth transitions between images without obvious color or brightness inconsistencies, indicating that the SIFT and ORB algorithms perform comparably in terms of processing quality under these specific test conditions.

3.Without Gaussian blur (left)

Image sharpness:

The image maintains its original sharpness, with details such as architectural lines and reflections of leaves retained intact.

Noise and details:

Some noise or slight distortion may be observed in high-contrast areas, such as the junction of trees and sky since no blur was applied to smooth out these details.

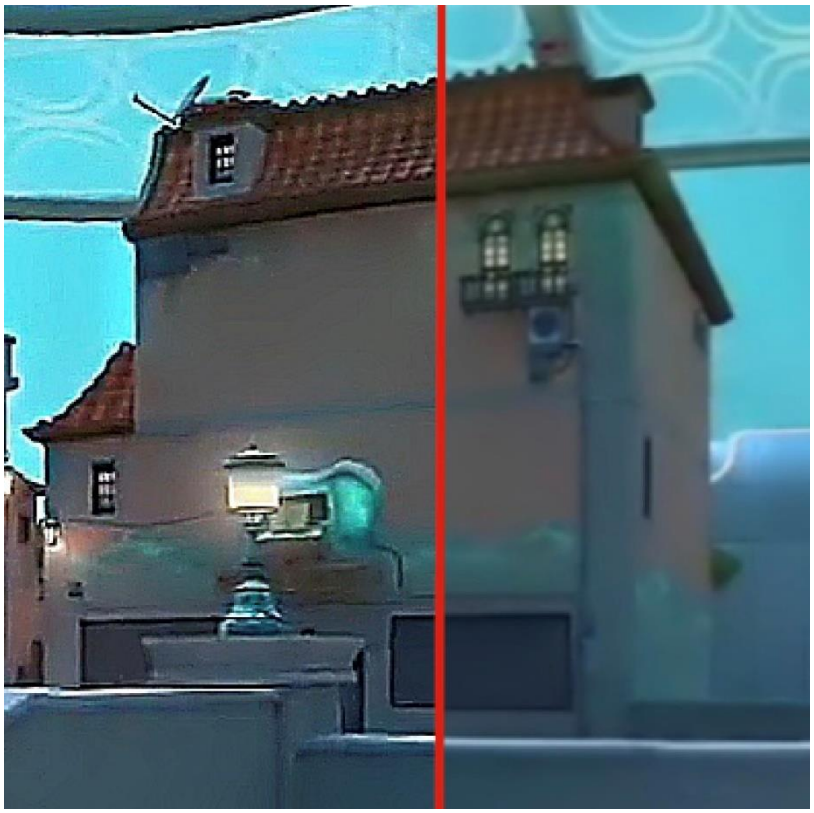
Using Gaussian Blur (right)

Image Softening:

After applying Gaussian Blur, the overall sharpness of the image is reduced, giving a softer, smoother visual effect. This is very effective in reducing image noise and over sharpening of details.

Noise Reduction:

At the edges of tree and building silhouettes, Gaussian Blur helps to reduce noise, making these areas visually more uniform.



While Gaussian Blur improves noise issues, it also makes some important details, such as window borders and small objects in the distance, less visible.

Not using Gaussian blur is better suited to scenes where the original detail and clarity of the image needs to be maintained, such as in applications that require precise analysis or post-processing.

The use of Gaussian blur is better suited to reducing distracting elements in visual

presentations and enhancing the overall look and feel of the image, especially when it is to be viewed over a long period of time or used as a background image.

4.Unsharpened (first image)

Image softness:

An unsharpened image is visually softer, and details such as building edges and sculpture outlines may appear less prominent.

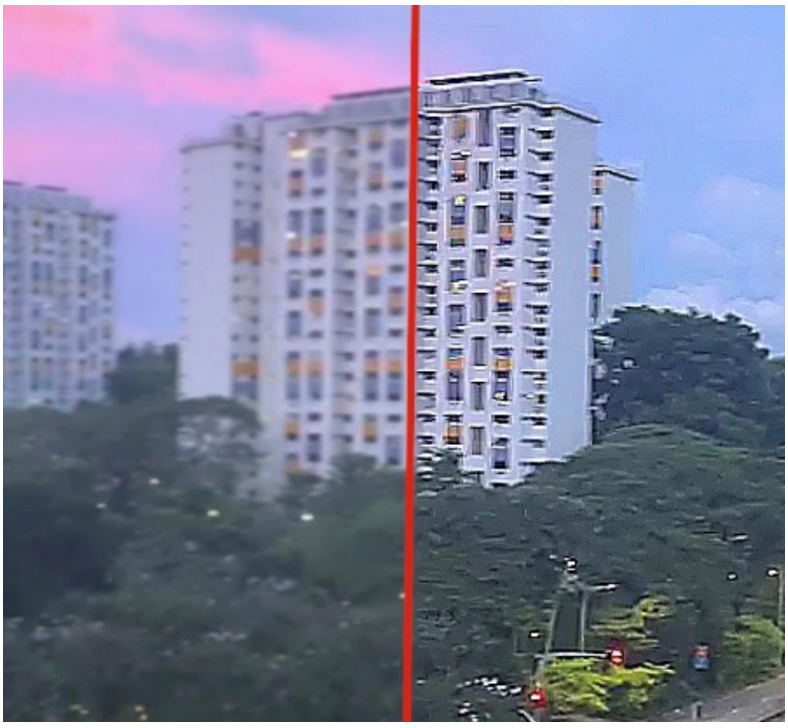
Noise and texture:

No sharpening may make noise and texture more visible in an image, especially in low-light or long-distance scenes.

Sharpening (second image)

Detail Enhancement:

Sharpening enhances the detail of the image, making the window lines of the building, the contours of the sculpture and the texture of the ground clearer and more prominent.



Visual Contrast:

Sharpening improves the contrast of an image, especially at the edges of objects, and helps to enhance the overall visual appearance of the image.

Possible increase in noise:

While sharpening enhances detail, it may also make noise in the image more noticeable, especially in areas of even color, such as the sky or water.

Unsharpened images are more appropriate in scenes where a natural and raw feel needs to be maintained, while sharpening is appropriate for scenes where detail and contrast need to be emphasized.

In Bhatt's opinion sharpening can significantly improve the visual quality of an image, giving it a more attractive and professional feel, but care needs to be taken with regard to the possible negative effects of over-sharpening, such as excessive noise and unnatural edges [5].

Chapter 4: Evaluation

Advantages of the methodology

Algorithms can be selected:

SIFT and ORB algorithms are used in this project for feature detection and matching, these algorithms perform well in terms of stability and robustness of image features. In particular, they are able to efficiently identify the same feature points when dealing with images with scale and rotation variations, which is crucial in panorama stitching.

The choice of these algorithms allows the system to handle a wide range of different video inputs, maintaining good performance even in dynamic scenes or with changing lighting.

Full functionality:

Enough features are added to make the desired settings to get the best panorama for the requirements.

User Interface Convenience:

The system provides an intuitive graphical user interface that allows users to easily upload videos, view processing progress and preview the generated panoramas. This enhances the usability of the system and enables non-technical users to operate and utilize the technology.

Limitations of the methodology

Adaptability to complex scenarios:

Although the system performed well in most of the test scenarios, it was still possible to experience splicing errors or inaccurate feature matching (sometimes with noticeable bottom bending) in extreme conditions (e.g., very low light, highly dynamic scenes). This

suggests the need to further optimize the algorithm or introduce new techniques to improve the system's adaptability.

Insufficient preprocessing details:

The video was read without any stabilization. Toxtli argues that the addition of stabilisation operations (deep learning) can make the feature matching and stitching process smoother and produce better images [6].

Comparison with expected results

The actual performance of the system was broadly in line with expectations, being able to generate high quality panoramas from different videos.

Chapter 5: Conclusion and Prospect

This project successfully developed a system to automatically generate panoramas from video, which employs advanced computer vision techniques such as SIFT and ORB for feature detection and matching, as well as efficient image fusion methods to ensure the quality of the panoramas. The system simplifies the operation process through a user-friendly interface, making it easy for non-technical users to generate high-quality panoramas. Despite the challenges in processing speed and adaptability to complex scenes, future work will focus on algorithm optimization and functionality extensions to improve the performance and applicability of the system.

Algorithm optimization: future work could focus on optimizing existing algorithms or exploring new ones to improve processing speed and accuracy. For example, more advanced feature detection algorithms or machine learning methods can be investigated to predict and optimize feature matching.

Extended functionality: new features such as support for 360-degree panorama generation or improving the system's ability to track objects in dynamic videos could be developed to enhance the system's application scope and appeal.

References

[1] Image matching using SIFT, surf, brief and Orb. (n.d.). https://arxiv.org/pdf/1710.02726.pdf

[2] Rosebrock, A. (2023, June 8). *Image stitching with opencv and python*. PyImageSearch. https://pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python/

[3] Feature matching. OpenCV. (n.d.). https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html

[4] Flores, T. (2019, August 19). *Gaussian blurring with python and opencv*. Medium. https://medium.com/analytics-vidhya/gaussian-blurring-with-python-and-opencv-ba8429eb879b

[5] Bhatt, K. (2022, December 19). *How to sharpen an image in python using opencv*. CodeSpeedy. https://www.codespeedy.com/how-to-sharpen-an-image-in-python-using-opencv/

[6] Toxtli, C. (2018, June 7). *Deepstab: Real-time Video Object Stabilization Tool by using Deep Learning*. Medium. https://medium.com/hci-wvu/face-stabilization-in-videos-using-deep-learning-features-dcfd4be365