
# Computer Vision

# COMP3065 UNNC

## Coursework Report

**Hao Yu**

**20320510**

**Git link:**
**https://github.com/fishotato/Computer_Vision_Coursework.git**

**Date: 2024/5/5**

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Project objectives and functional implementation

The key objective of this project is to develop a robust program that is capable of extracting key frames from video files in order to generate higher resolution panoramic images. The innovation of this program is that it can automatically analyse video files and generate panoramic images of high quality based on the distinctive characteristics of different video file. Despite the fact that the results of panoramic images can not meet a professional level, they are already applicable to the majority of commonplace situations. The program can overcome most of the challenges associated with video quality, complexity, and environmental conditions through the utilisation of advanced computer vision techniques that were covered in class. To achieve this objective, I implemented the three core functions listed below:

1.Generate panoramas from video files of varying sizes

The algorithm uses a dynamic strategy to extract key frames from video files of varying sizes. To be more specific, for video files that are less than 10MB, 20% of the total number of video frames are extracted as key frames to ensure that important information is captured without loss. With this strategy, we are able to guarantee superior performance and produce high-quality panoramas regardless of the video size.

2.Generate panoramas from videos containing scene transitions

Continuity and visual coherence of the panorama image are affected during the conversion from video to panorama due to frequent scene changes. In other words, it can result in mistakes during the process of stitching the image, which allows discontinuous frames to be spliced together. In this project, the extracted key frames are analysed and filtered using the SIFT algorithm and we filter the valid key frames based on the number of feature points. In particular, we set up a minimum and maximum threshold for the number of feature points. For panorama stitching, we use only key frames with the specified number of feature points between these two thresholds, while key frames outside this range are filtered out. This functionality maximises the use of visual information in the video. Therefore, it is especially suitable for videos that contain frequent scene transitions or varying content.

3.Generate panoramas from blurred videos

The overall quality of the panoramas produced from blurred videos is enhanced via a series of image enhancement measures and Gaussian blurring techniques. Firstly, we dynamically adjust the size of the Gaussian blurring kernel during the filtering of key frames based on the video's blurring level and the density of detected feature points. We can remove a large amount of noise and effectively use the feature points of the key frames for stitching by applying Gaussian blurring to the key frames. In addition, following the stitching process, a series of image enhancement procedures are executed on the panorama to guarantee that the ultimate panorama maintains its high quality while retaining as much detail and clarity as possible.

# Chapter 2

# Detailed steps and computer vision techniques

In this section, a detailed description of the process and specific computer vision techniques used in the program for converting video to panorama images will be outlined.

## 2.1 Key frames extraction

The algorithm used for key frame extraction is able to dynamically extract key frames from the input video according to the file size. As can be seen in Figure 2.1, it first uses 'os.path.getsize' to get the file size of the input video. It then calculates the total number of frames for the video using the VideoCapture class of OpenCV. For videos that are smaller than 10MB, we extract 20% of the total number of frames as key frames. For videos that are between 10MB and 20MB, we extract 10% of total number of frames. For videos that are larger than 20MB, we extract a fixed number of 40 frames. Next, the algorithm calculates the interval between each key frame to be extracted and extracts key frames by directly accessing frames using their indexes. It is more efficient to jump directly to specific frames without processing the entire video, which significantly speeds up the extraction process. Furthermore, to maintain the integrity and coherence of the video information, we ensure that the first and the last frames of the video are extracted.

```python
def extract_key_frames(video_path):
    # Get the size of the input video in MB
    file_size = os.path.getsize(video_path) / (1024 * 1024)
    # Initialise the VideoCapture object
    video_capture = cv2.VideoCapture(video_path)
    # Calculate the total number of frames in the video
    total_frames = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))

    # Use different keyframe extraction strategies based on the video size
    # 1.For videos under 10MB, extract 20% of the total number of frames as keyframes
    if file_size <= 10:
        key_frames_count = int(total_frames * 0.20)
    # 2.For videos between 10MB and 20MB, extract 10% of the total number of frames as keyframes
    elif file_size <= 20:
        key_frames_count = int(total_frames * 0.10)
    # 3.For videos over 20MB, extract a fixed number of 40 frames
    else:
        key_frames_count = 40
```

Figure 2.1: Python code for key frames extraction

## 2.2 Filtering valid key frames

During the process of filtering valid key frames, SIFT algorithm of OpenCV is applied to each extracted key frame to detect feature points. Then we calculate the number of detected feature points for each key frame and use it to filter the valid key frame for stitching. Figure 2.2 shows that we set a minimum threshold and a maximum threshold for the number of feature points. Only those with 200 to 17000 features are selected for later alignment and stitching to ensure sufficient details. Besides, a Gaussian blur (GaussianBlur class of OpenCV) is applied to these valid key frames to effectively reduce noise and preserve important visual information. The kernel size of Gaussian blur is determined according to the number of features. The kernel of Gaussian blur is set to 3 when the number of feature points is less than 2000, and 5 for feature points more than 2000. It can not only preserve key information in the frame, but also removes as much as noise as possible.

## 2.3 Stitching key frames into a panorama

As can be shown in Figure 2.3, to create a complete panorama from key frames, we use the OpenCV Stitcher class with the parameter set to panorama mode. The filtered and processed key frames are stitched into a continuous panorama image in this step. After the process of stitching, we are to check the output status of the stitcher to ensure a

```python
# Filter the frames based on the number of feature points per frame and
# apply Gaussian blur to the selected frames.
1 usage
def filter_frames(frames):
    # Create SIFT feature detector
    sift = cv2.SIFT_create()
    filtered_frames = []
    min_threshold = 200
    max_threshold = 17000
    for frame in frames:
        # Compute SIFT feature points on each frame
        keypoints, descriptors = sift.detectAndCompute(frame, None)
        num_keypoints = len(keypoints)
        # Filter out keyframes based on number of feature points
        if min_threshold <= num_keypoints <= max_threshold:
            # Adjust the kernel size of the Gaussian blur dynamically
            if num_keypoints < 2000:
                ksize = (3, 3)  # Fewer feature points, use smaller kernel
            else:
                ksize = (5, 5)  # More feature points, use larger kernel
            blurred_frame = cv2.GaussianBlur(frame, ksize, 0)
            filtered_frames.append(blurred_frame)
    return filtered_frames
```

Figure 2.2: Python code for filtering valid key frames

successful creation of the panorama or an error status if the stitching fails. By using the OpenCV stitcher class, the output panorama image can reach a high-quality level.

```python
# Stitch keyframes into a panorama by using the OpenCV Stitcher class
1 usage
def stitch_frames(key_frames):
    # Create a Stitcher instance
    stitcher = cv2.Stitcher.create(cv2.Stitcher_PANORAMA)
    # Perform stitching operation
    status, pano = stitcher.stitch(key_frames)
    # Check the stitching status
    if status == cv2.Stitcher_OK:
        return pano
    else:
        print("Stitching failed: ", status)
        return None
```

Figure 2.3: Python code for stitching key frames into a panorama

## 2.4 Enhancing the panorama image

As can be seen in Figure 2.4, to implement the functionality of generating panoramas from blurred videos, we add a series of image enhancement techniques to improve the visual appeal and clarity. Firstly, we apply a second Gaussian blur with a kernel size of (3,3) on the panorama image. Through using a small kernel size, it helps reduce noise effectively without blurring the image details. In addition to this, gamma correction technique is used to perform a dynamic range adjustment. The gamma value is set to 1.1 and it is used for calculating an inverse gamma correction table. The table can adjust the luminance values of the panorama image, enhancing the contrast and the brightness. By using this gamma value, the program fine-tunes the output of the panorama image, which makes the dark and light areas of the image are well represented. The last enhancement technique is sharpening the image. We apply a Gaussian blur to blur the image slightly and use a weighted sum approach to blend the blurred version with the original panorama image. The weight for the original image is 1.5 and -0.5 for the blurred image, which aims to enhance the details and edges of the image. This step is significant to generate high-quality panoramas as it enhances the textures and details.

```python
# Perform a series of image enhancement processes to cope with
# the generation of panoramas under conditions of blurred video
1 usage
def enhance_image(image):
    # Apply Gaussian blur to suppress noise
    def suppress_noise(img):
        return cv2.GaussianBlur(img, (3, 3), 0)

    # Apply dynamic range adjustment to improve the brightness
    # and contrast of panorama
    def adjust_dynamic_range(img, gamma=1.1):
        invGamma = 1.0 / gamma
        table = np.array([((i / 255.0) ** invGamma) * 255 for i in np.arange(0, 256)]).astype("uint8")
        return cv2.LUT(img, table)

    # Sharpen the image
    def sharpen_image(img):
        blurred = cv2.GaussianBlur(img, (0, 0), 3)
        return cv2.addWeighted(img, alpha: 1.5, blurred, -0.5, gamma: 0)
```

Figure 2.4: Python code for enhancing the panorama image

## 2.5 Resizing the panorama image

The Figure 2.5 illustrates the last step in the program. We set the height to 1500 and calculate the desired height-to-width ratio. Then, we resize the image and get the ultimate panorama.

```
# Resize the panorama
1 usage
def resize_panorama(pano, height=1500):
    # Calculate the new size ratio
    height_ratio = height / pano.shape[0]
    new_width = int(pano.shape[1] * height_ratio)
    # Resize the panorama
    pano_resized = cv2.resize(pano, dsize: (new_width, height))
    return pano_resized
```

Figure 2.5: Python code for resizing the panorama image

# Chapter 3

# Results presentation and explanation

We use three video files of different difficulty levels to test our panorama generation program. The purpose of testing videos with different complexities is to verify that our program can maintain the high-quality panoramas when dealing with videos with varying resolutions and containing complex scenes. The videos are named 'video_1.mp4', 'video_2.mp4', and 'video_3.mp4'. Each video is processed to generate a corresponding panorama image named 'video_1_panorama.jpg', 'video_2_panorama.jpg', and 'video_3_panorama.jpg'. These panorama images are generated automatically from the videos in the current directory by running our program.

## 3.1 Analysis of 'video_1.mp4'

The file size of the 'video_1.mp4' is 6.74MB which meets the criteria for the video under 10MB. As we mentioned before, we extract 20% of the total number of frames as key frames. The length of the video is 7 seconds and is shot in a library environment. The video includes scenes of ceiling, lights, floor, students studying and some other library facilities. Due to the low resolution and small size of the video, the number of feature points in each key frame is very small and the generation of the panorama for this video is fast. As shown in the Figure 3.1, the generated panorama from this video is of high quality and captures the details and features of the objects in the scene. The result proves that our program can efficiently process small videos files with low complexity and also

verifies that the functionality of dynamically extract key frames according to the size of the video file.


Figure 3.1: panorama1

## 3.2 Analysis of 'video_2.mp4'

The 'video_2.mp4' is a large file reaching 24.4MB. Since its size is more than 20MB, the program extracts a fixed number of 40 frames as key frames. The scene of the video is a blurred campus night scene with dark skies, trees, streetlights and dark green lawns. We select this video to test the functionality of generating panorama from blurred videos. Generating panorama image from this video is difficult due to its low resolution, large file size and varying brightness. As can be seen in Figure 3.2, the program successfully generates a panorama with higher resolution compared with the scene in the blurred video. We improve the sharpness while accurately reproducing the details and contours of the scene. By using dynamic range adjustment, the brightness and overall contrast of the right side of the scene are optimized and prevent distortion of the panorama image due to overexposure. The result demonstrates we have implemented the functionality of generating panoramas from blurred videos and the ability to handle large videos files.

Figure 3.2: panorama2

## 3.3 Analysis of 'video_3.mp4'

The file size of 'video_3.mp4' is 22.6 MB, which also resulted in the extraction of 40 key frames because of its size. Although the video is not as large as 'video_2.mp4', the scene of the video contains complex scene transitions such as blue sky, green grass, buildings and colorful logos, which indicates a high density of feature points in each frame. From our calculations, the average number of feature points contained in each frame is 15,000. Therefore, it is significantly challenging in maintaining a high-quality panorama output. Figure 3.3 shows that the generated panorama is not only coherent, but also excellently preserve the features and details of the video with a high degree of resolution. The result conforms the ability to generate high-quality panoramas when coping with complex scene transitions and rich visual content.


Figure 3.3: panorama3

# Chapter 4

# Critical Evaluation

## 4.1 Advantages

1.Processing video files of various sizes dynamically

One significant strength of our program is to dynamically adjust the key frame extraction based on the video file size. This method is able to process the small video files quickly and efficiently by extracting a higher percentage of frames to retain essential visual information. For larger files, it reduces the number of key frames extracted to speed up the panorama generation time and optimizes the resource usage.

2.Enhancing the panorama image from blurred videos

Another notable advantage is the program's ability to handle blurred videos. By enhancing the feature detection process and apply Gaussian blur according to the number of features in each key frame, we have successfully captured and enhanced the details and features of the blurred videos. The method is particularly useful when the quality of the source videos is low.

3.Capturing the details and features comprehensively

The panorama image generated by our program thoroughly reproduce the visual information of the original video without any noticeable loss of details. It is especially effective when coping with complex scene transitions by preserving these details and features of

the videos.

## 4.2 Disadvantages

1.Irregular shape of the panorama

One of the obvious disadvantages of the program is that the output panorama image is irregular and contains black shadows around the edges. We also considered cropped the output panorama using a maximum internal rectangle, However, this would lose a lot of information from the video. Therefore, we decide to present the panorama without performing any cropping operation. To improve the overall visual appearance of the panorama, the program still need additional processing to clean the edges.

2.Loss of information in high-resolution video

Although the program generally performs well, it still has difficulties in processing high-resolution videos when the sharpness and details requirements are very strict. The problem is happened in the process of stitching the key frames. The incorrect matching of feature points can lead to obvious seams in the panorama image and the error of key frames stitching. To deal with the high-resolution videos, we still need to improve the feature detection and matching algorithms.

# Bibliography