Coursework for COMP3065 Computer Vision

This report shows the work of the computer vision coursework. The selected project is a) panorama generation from videos.
This project needs to develop of a program aimed at generating panorama images from provided short videos. There are three sets of videos in the folder for testing the program's functionality.

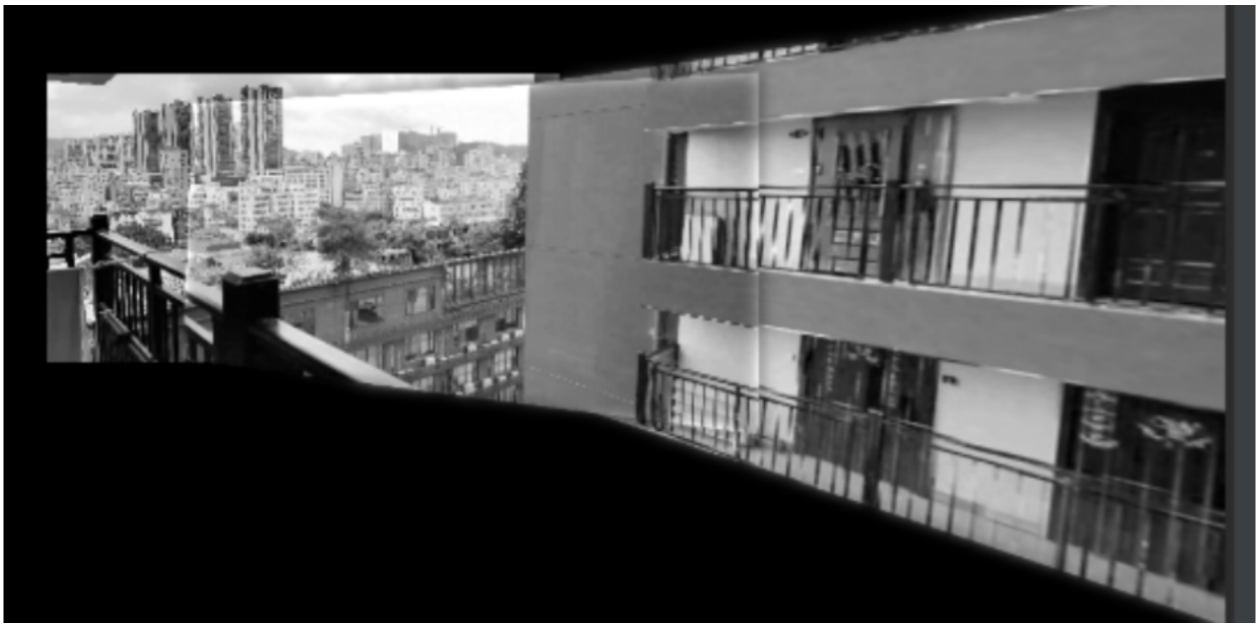
Introduction

This project explores innovative solutions to create high-quality panoramic images from video sequences, focusing primarily on overcoming common obstacles in feature point extraction, precise transformation calculations, and seamless frame stitching. The cylindrical warp technique stands at the core of the approach, chosen for its ability to correct image distortions inherently associated with simpler stitching methods.

Methodology

Feature Extraction and Transformation

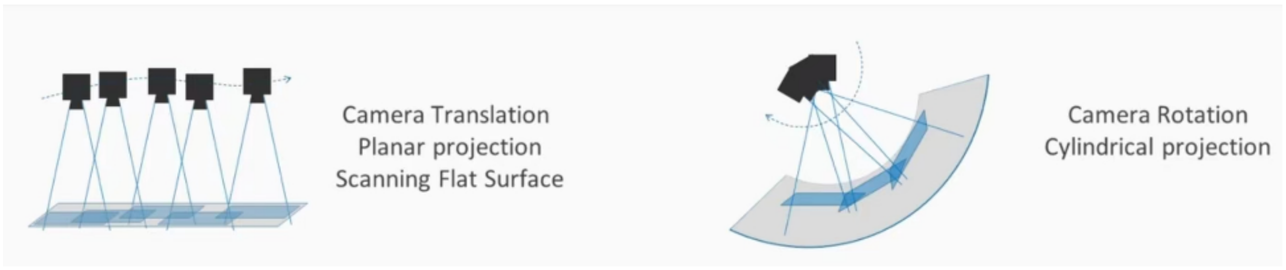
Initially, standard projective transformations (`warpPerspective`) were employed but often led to unsatisfactory results with noticeable distortions and misalignments in the final stitched panorama. This was particularly evident in scenarios involving rapid camera movements or varied scene depths. Below is an example.



Cylindrical Warp Implementation

To resolve the stitching issues, cylindrical warp was implemented (https://stackoverflow.com/questions/12017790/warp-image-to-appear-in-cylindrical-projection). This method maps the image onto a cylindrical surface, allowing for more flexible and forgiving image stitching, particularly in handling rotations and alignments.

This image demonstrates the idea behind cylindrical warp:



Video Frame Processing for Panorama Generation

The video processing involved handling unpredictable rotation speeds, necessitating an automatic algorithm to manage frame fusion only when overlaps were minimal. This approach minimized image degradation due to excessive fusion:

```python
def estimate_overlap_from_left_border(img1, img2, points1, points2, good_matches):
        overlap2 = max(points2[:, 0, 0])
        overlap_percentage = overlap2 / img2.shape[1] * 100
        return overlap_percentage

def estimate_overlap_from_right_border(img1, img2, points1, points2, good_matches):
        overlap2 = min(points2[:, 0, 0])
        overlap_percentage = (img2.shape[1] - overlap2) / img2.shape[1] * 100
        return overlap_percentage
```

Only when the overlap percent is below 50%, shall it start the stitching process.

Bidirectional Stitching

The video sequences typically exhibit continuous rightward rotation. To harness this, the strategy involved stitching images up to 180 degrees to the right first, storing these frames, and subsequently processing them in reverse to complete the leftward stitching, thereby forming a full 360-degree panorama.

```python
while True:
        success, frame = cap.read()
        if not success:
                break

        # print(current_frame, int(fps * time_interval))
        if current_frame % int(fps * time_interval) == 0:
                frame = cv2.resize(frame, None, frame, resize_scale, resize_scale)

                if new_width / standard_width < 4.1 and total_percent < 460: # stitch to the
right
                        img2 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                        img2, mask2 = cylindricalWarpImage(img2, f) #
cylindrical_warp_with_focal(img2, f) #
                        img2 = cv2.copyMakeBorder(img2, VERTICAL_BORDER_LEN,
VERTICAL_BORDER_LEN, 2, 2, cv2.BORDER_CONSTANT)
                        img1copy = img1.copy()
                        img1, colored_bound, overlap_percent, match_cnt, affine_matrix =
stitch_images(img1, img2, sift, flann,
                        estimate_overlap_from_left_border, True)

                        if match_cnt <= 10:
                                total_percent = 1460
                                continue

                        if overlap_percent > 0:
                                cv2.waitKey(5000)

                                stitch_list.append(frame)
                                n_h, m_h, n_w, m_w = colored_bound
                                new_width = m_w - n_w + 20

                                total_percent += overlap_percent
                                print("width changes to ", new_width, " img width=",
img2.shape[1], " total percent=", total_percent, " key_index=", key_index)
                                cv2.imwrite(f"temp/{video_path}_img2_{key_index}.jpg",
frame)

                                cv2.imwrite(f"temp/{video_path}_img1_{key_index}.jpg",
img1copy)


                                key_index += 1
                                if new_width + img2.shape[1] * 2 // 3 > img1.shape[1]:
                                img1 = cv2.copyMakeBorder(img1, 0, 0, 0, new_width +
img2.shape[1] * 2 // 3 - img1.shape[1], cv2.BORDER_CONSTANT)
                else:
                        left_frame_arr.append(frame)

        current_frame += 1

img1 = cv2.copyMakeBorder(img1, 0, 0, standard_width * 2 // 3, 0, cv2.BORDER_CONSTANT)
# start to handle left
good_stitch_index = 0
for frame in left_frame_arr[::-1]:
        img2 = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        img2, mask2 = cylindricalWarpImage(img2, f)
        img2 = cv2.copyMakeBorder(img2, VERTICAL_BORDER_LEN, VERTICAL_BORDER_LEN, 2, 2,
cv2.BORDER_CONSTANT)
        img1copy = img1.copy()
        img1, colored_bound, overlap_percent, match_cnt, affine_matrix = stitch_images(img1,
img2, sift, flann, estimate_overlap_from_right_border, False)

        if match_cnt <= 10:
                break
        if overlap_percent > 0:
                stitch_list.append(frame)
                n_h, m_h, n_w, m_w = colored_bound
                new_width = m_w - n_w + 20
                good_stitch_index += 1
                if good_stitch_index % 4 == 1 or good_stitch_index > 7:
                        debug_print(img1copy, img2, img1)
                total_percent += overlap_percent
                print("handling left images: width changes to ", new_width, "img width=",
img2.shape[1], "total percent=", total_percent, " key_index=", key_index)
                cv2.imwrite(f"temp/{video_path}_img2_{key_index}.jpg", frame)
                cv2.imwrite(f"temp/{video_path}_img1_{key_index}.jpg", img1copy)
                key_index += 1
                if new_width + img2.shape[1] * 4 // 3 > img1.shape[1]:
                        img1 = cv2.copyMakeBorder(img1, 0, 0, new_width + img2.shape[1] * 4
// 3 - img1.shape[1], 0, cv2.BORDER_CONSTANT)
```

Linear Interpolation for Image Fusion

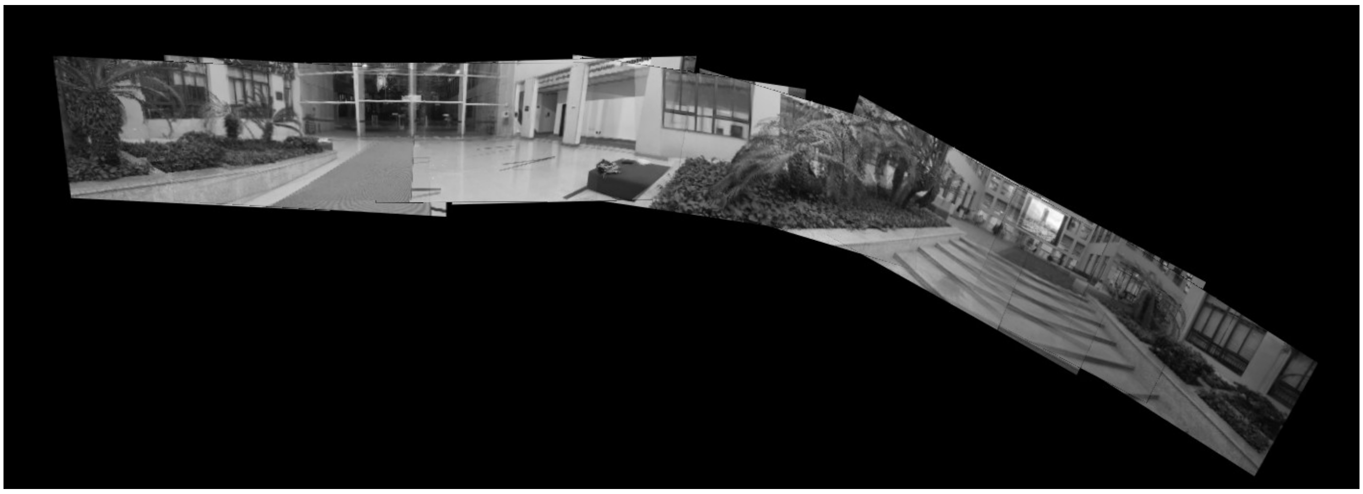
To ensure visually seamless transitions at image seams, linear interpolation was employed. This technique adjusts the pixel values along the overlap region to blend images smoothly, enhancing the visual continuity across the stitched panorama.

```
def blend_images(img1, img2, img1_pts, img2_pts, left_to_right = True):

        w = img1.shape[1]
        h = img1.shape[0]

        (MM, mask) = getTransform(img2_pts, img1_pts)
        img_final = cv2.warpAffine(img2, MM, (img1.shape[1], img1.shape[0]))

        if left_to_right:
                img2_start, w1_, w2_, w3_ = find_color_bound(img_final)
                w1_, img1_end, w2_, w3_ = find_color_bound(img1)

                gap_len = img1_end - img2_start

                for i in range(h):
                        for j in range(w):
                                if img_final[i][j] == 0:
                                        img_final[i][j] = img1[i][j]
                                elif img1[i][j] != 0:
                                        alpha = 1.0 * (j - img2_start)
                                        alpha /= gap_len
                                        img_final[i][j] = img_final[i][j] * alpha + img1[i]
[j] * (1-alpha)
        else:
                w1_, img2_end, w2_, w3_ = find_color_bound(img_final)
                img1_start, w1_, w2_, w3_ = find_color_bound(img1)

                gap_len = img2_end - img1_start

                for i in range(h):
                        for j in range(w):
                                if img_final[i][j] == 0:
```

```
                    img_final[i][j] = img1[i][j]
                elif img1[i][j] != 0:
                    alpha = 1.0 * (j - img1_start)
                    alpha /= gap_len
                    img_final[i][j] = img1[i][j] * alpha + img_final[i]
[j] * (1 - alpha)

    n_w, m_w, n_h, m_h = find_color_bound(img_final)
    return img_final, (n_h, m_h, n_w, m_w), MM
```

Flattening the Field of View

Ensuring a flat perspective was crucial to prevent the panorama from tilting upwards, a common issue with naively stitched panoramas. Below is an example.

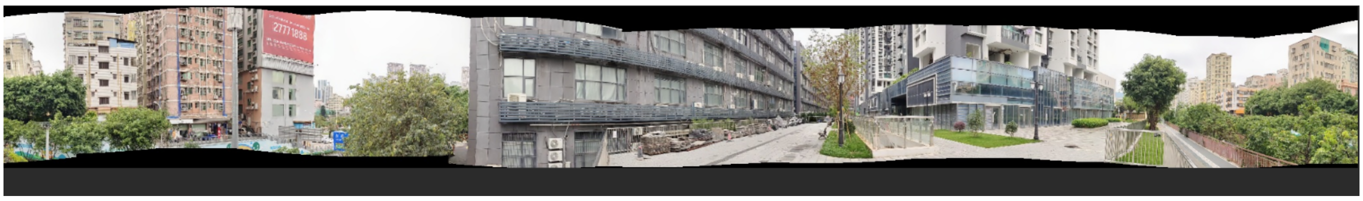


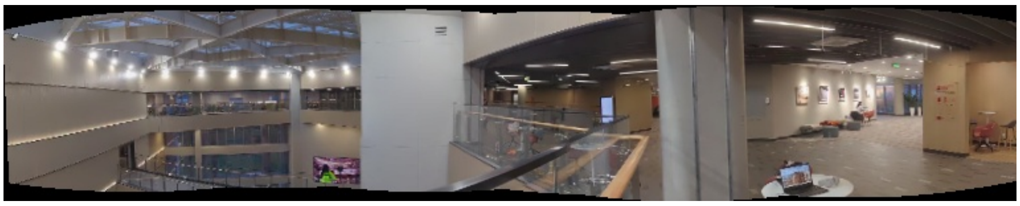
Maintaining a flat field of view is crucial to avoid unwanted tilting effects that can arise from improper alignment or perspective distortions during stitching. Manual adjustments and algorithmic corrections were periodically necessary to achieve the desired flatness in the resulting panorama.

For cases of severe tilting, the built-in OpenCV `Stitcher` class was utilized as a temporary fix until a more robust solution could be integrated into the workflow.

One might question the rationale behind not utilizing the built-in Stitcher class at the outset of the process. However, it is imperative to consider that our objective involves generating video panoramas. Given the abundance of images within each frame, the preceding steps serve to curate a suitable selection of frames for stitching—sufficient in quantity yet not excessive. Consequently, the preparatory steps furnish the Stitcher class with optimal material for seamless integration.

Allow me to present several aesthetically pleasing outputs:





Discussion

The cylindrical warp approach introduced significant improvements over traditional planar stitching methods, particularly in handling complex video inputs with varied rotation speeds and alignment challenges. While the transformation calculations became more complex, the resulting panoramas exhibited superior quality and continuity.

The adoption of the cylindrical warp approach presents notable strengths in comparison to conventional planar stitching methods. Specifically, it excels in addressing the intricacies of video inputs characterized by diverse rotation speeds and alignment discrepancies. By utilizing cylindrical warping, the method enhances the ability to seamlessly merge frames, thereby mitigating distortions and inconsistencies commonly encountered in planar stitching. Moreover, the resultant panoramas demonstrate enhanced quality and continuity, offering a more immersive viewing experience.

However, it's essential to acknowledge certain weaknesses associated with this approach. Primarily, the computational demands escalate due to the increased complexity of transformation calculations required for cylindrical warping. This heightened computational burden may translate into longer processing times and resource-intensive operations, potentially posing challenges in real-time applications or scenarios with constrained computational resources. Additionally, while the cylindrical warp approach addresses many challenges inherent in planar stitching, it may still encounter limitations in scenarios with extreme rotational variations or irregular frame alignments, necessitating further refinement or alternative strategies for optimal results.

Moving forward, the project can endeavor to improve in several key areas. This includes the implementation of more advanced fusion techniques, utilizing complex algorithms to minimize visible seams further. To improve the project, the future work can be aimed at developing more sophisticated automated adjustment mechanisms, enabling dynamic alignment and tilt correction for enhanced precision. Furthermore, there is a focus on adapting the algorithm to support real-time video input, facilitating live event streaming through real-time panorama creation. These enhancements collectively aim to elevate the project's capabilities and address emerging demands in panoramic video processing.

Conclusion

The implementation of cylindrical warp techniques has proven to be a game-changer in the field of panoramic video processing. This method has not only resolved many of the traditional stitching challenges but has also opened up new possibilities for real-time and dynamic video panorama creation.