COMP3065 Computer Vision

Coursework
Panorama generation from videos

Junyu Liu
20216355

How to run the code

Please refer to README.md file in the code folder to configure environments, prepare corresponding data, and check panorama results. Sample outputs are shown in code blocks in execute.ipynb.

General Algorithm

This is a pseudocode for the general pipeline of my panorama generation algorithm. This is just an illustration of the general structure for ease of reading, and thus the code implementation may have some slight differences. Also, many detailed features and tricks are omitted there but are detailedly explained in the following sections.

Input: Video file.
Output: Stitched panorama image.
Open the video file with frames $\{f_1, ..., f_N\}$ and get the total number of frames N
Prompt user to decide whether deblur $isDeblur$ and choose deblur method m
if m == Non-ML Method **then**
    Initialized deblur model D by calculating PSF kernels and Wiener deconvolution.
else if m == NAFnet Method **then**
    Import, configure, and initialize pre-trained deblur model D.
end if
Prompt user to specify the reference projection plane number k
Separate frames according to N to left_frames $\{f_1, ..., f_k\}$ and right_frames $\{f_{k+1}, ..., f_N\}$
for each frame f_n in right_frames **do** ▷ // From f_{k+1} to f_N
    if $isDeblur == True$ **then**
        Feed frame f_n to D to deblur
    end if
    Match and select the keypoints $\{p_1, ..., p_m\}$ from previous f_{n-1} and current frame f_n
    Calculate homography $mask_n$ and warp current frame f_n to f_n^{warp}
    Remove outliers the warped frame f_n^{warp} to f_n^{clean}
    Blend the previous frame f_{n-1} with f_n^{clean} and update right_panorama $pano_r$
    Set the previous frame as f_n and keypoints as $\{p_1, ..., p_m\}$
end for
for each frame in left_frames **do** ▷ // From f_k to f_1
    Iteratively deblur, warp, and stitch previous f_{n+1} and current frame f_n. ▷ // Similar as above
end for
Stitch the left_panorama $pano_l$ and right_panorama $pano_r$ ▷ // Similar as above
Return final_panorama $pano$

Algorithm 1: General algorithm for stitching a panorama from video frames

Key Features of Implementation

This section introduces 5 key implementation features. Other small features and general code pipelines can be found in section or in code implementation documents.

Feature 1: User-specified Projection Plane

This feature prompts users to specify a desired Projection Plane for the panorama to customize their viewing experiences. The default setting is to project all video frames on the plane where the middle frame lies to get a balanced panorama (illustrated in Figure 1).

Figure 1: Illustration of stitching with regard to the center plane.

Users can choose different planes that they want to get different projection results. Figure 2 illustrates different projecting choices, in which the lower left subfigure, lower right subfigure, and the upper subfigure respectively illustrate the result of choosing left $1/3$, right $1/3$, and middle $1/3$ frame as projection plane.

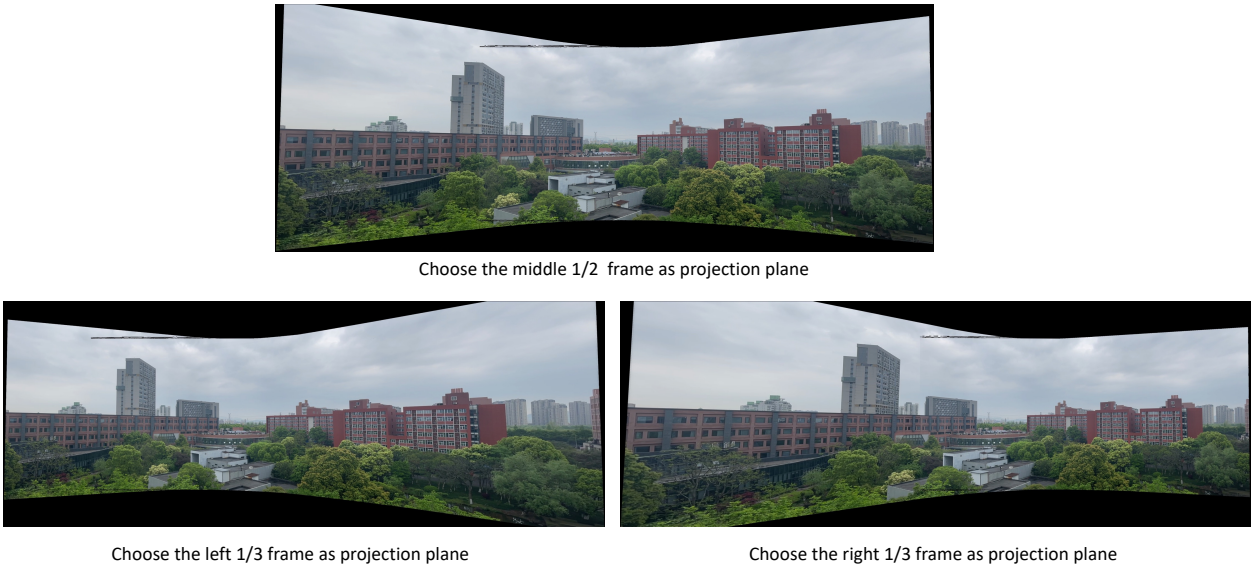

Figure 2: Illustration of choosing different projection planes.

Feature 2: Warpped image outlier removal

When warping an image, the outermost pixels of the transformed content could result as an outlier (especially when the image contains pure black areas, such as the case in panorama). I have confirmed that this is not a problem of unalignment of two images. I implement a function to detect the outliers of content areas and remove the detected pixels, which is explained in Algorithm 2. Figure 3 illustrates the comparison of outlier removal and some zoomed in look of the warp outlier.

Input: Warped image with outlier $warp_{raw}$
Output: Cleaned-up warped image without outlier $warp_{clean}$
Convert $warp_{raw}$ to grayscale and create a mask $mask_{all}$ for the content area
Find the contours $cnts$ in the masked content area
Create a mask $mask_{outermost}$ for the outermost pixels from the largest contours $cnts_{max}$
Set the outermost pixels to pure black to get $warp_{raw}$
Return cleaned-up warped image $warp_{raw}$

Algorithm 2: Automatic content locating and loose/tight cropping

Feature 3: Automatic content locating and loose/tight cropping

The raw output of iterative stitching from video frames often results in large areas of pure black areas and I implemented a function to automatically detect the content location and crop out the black areas. A pseudo-code for this function is provided in Algorithm 3: This function first locates

Figure 3: Illustration of outlier removal of the wrapped image.

Input: Raw panorama $pano_{raw}$
Output: Loose crop $pano_{loose}$ and tight crop $pano_{tight}$
Convert $pano_{raw}$ to grayscale and threshold it to create a binary image $pano_{binary}$
Find the contours $cnts$ in the thresholded image $pano_{binary}$
Find the bounding box $x_{min}, x_{max}, y_{min}, y_{max}$ of the largest contour $cnts_{max}$
Crop according to $x_{min}, x_{max}, y_{min}, y_{max}$ to get the loose crop $pano_{loose}$
Search for the tightest crop location $x'_{min}, x'_{max}, y'_{min}, y'_{max}$ respectively in the left half, right half upper half, and lower half of the content area
Crop according to $x'_{min}, x'_{max}, y'_{min}, y'_{max}$ to get the tight crop $pano_{tight}$
Return $pano_{loose}$ and $pano_{tight}$

Algorithm 3: Automatic content locating and loose/tight cropping

the content range and then performs to crops: a loose crop to keep all the content, and a tight crop to remove black areas (illustrates in Figure 4).

Figure 4: Illustration content detection and loose/tight crop of panorama.

Feature 4: Automatic perspective rectification

When taking photos of buildings or scenes that contain large amount of vertical lines, photography aesthetics requires that the lines should all be vertical and parallel. However, this requires photographers to point their cameras exactly horizontal when taking photos or videos, which is usually not the case in the real world (especially when panning cameras to take videos). Thus, automatically rectifying the perspective of the panorama is important for photography aesthetics usage. I implement an algorithm to automatically correct the perspective of the panorama. The pseudo code is provided in Algorithm 4.
Figure 5 illustrates a comparison of an un-rectified panorama and a corresponding rectified version.

Feature 5: Devonvolution-based deblur between frames

The non-ml-based deblur method mainly consists of motion estimation (pseudo code given in Algorithm 6) and Wiener deconvolution (pseudo code given in Algorithm 5). This method is inspired by image deblurring methods introduced in [1].
The key observation for designing this deblurring method is that a frame's magnitude and direction of motion blur could be approximated motions between adjacent frames. Thus this method first estimates the motion magnitude and direction by matching key points on adjacent frames. The estimated magnitude and direction are used for creating an estimated motion kernel for inversely convolving blurred frames in subsequent steps. Then, Weiner Deconvolution is used to deconvolute the estimated motion kernel to minimize the blur in the frequency domain. It utilizes the power density of the targeted deblurred frame and the motion noise to recover the deblurred frame. A

Input: Horizontally tight crop image $pano_{crop}$ from Algorithm 3
Output: Perspective rectified panorama $pano_{rectify}$
Convert $pano_{crop}$ to grayscale and threshold it to create a binary image $pano_{binary}$
Find the contours in the binary image and sort out the largest one cnt_{max}
Find the left edge line $edge_l$ and right edge line $edge_r$ from the cnt_{max}
Sort the points in $edge_l$ and find 2 end points end_l^1 and end_l^2
if y value in end_l^1 ¿ end_l^2 **then**
    Set left top point src_l^t to be end_l^2 and left bottom point src_l^b to be end_l^1
else
    Set left top point src_l^t to be end_l^1 and left bottom point src_l^b to be end_l^2
end if
Similar step to find and set right top point src_r^t and right bottom point src_r^b
Calculate the perspective transform matrix M from $\{src_r^t, src_r^b, src_l^t, src_l^b\}$ to four corners of the rectangle image.
Warp the image according to M.
Return the rectified image $pano_{rectify}$

Algorithm 4: Automatic perspective rectification

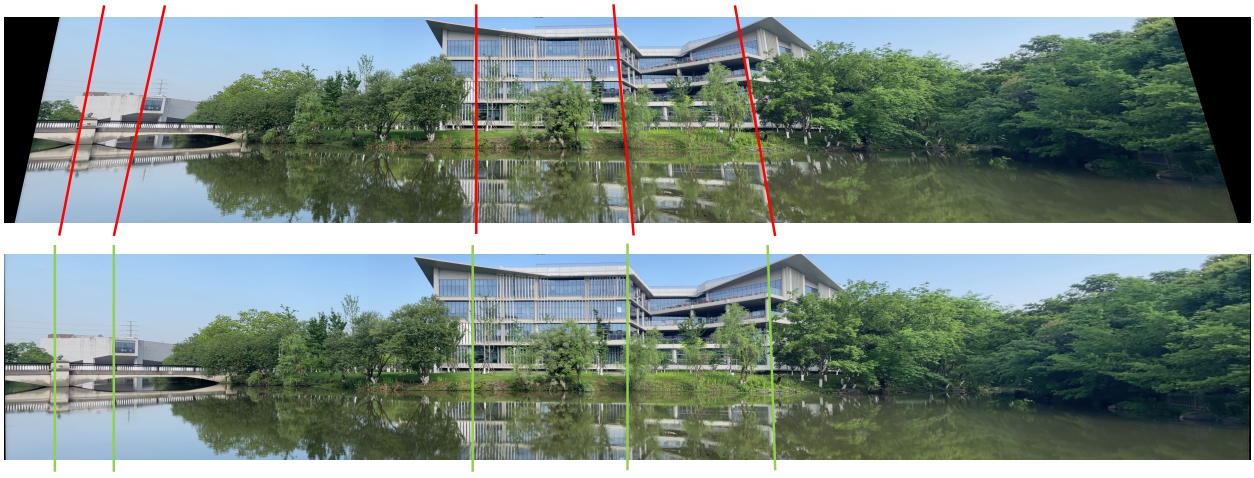

Figure 5: Illustration of automatic perspective rectification. The upper image is the raw panorama where the building looks like ”falling down” due to the un-horizontal pointing of the camera. The lower one is the automatically rectified version where all lines that should be vertical are straight and vertical

Input: Frames with blur $\{f_1, ...f_n\}$
Output: Frames that are deblurred $\{f_1^{deblur}, ...f_n^{deblur}\}$
for frame f_k in blurred frames $f_2, ...f_n$ **do**
  Get the previous frame f_{k-1} and the next frame f_{k+1}
  for channel f^c in $\{f^r, f^g, f^b\}$ of f_{k-1}, f_k, f_{k+1} **do**
    Feed $\{f_{k-1}^c, f_k^c, f_{k+1}^c\}$ into Algorithm 5 to get estimated angle ang and amount d
    Create a motion kernel sf using the ang and d and pad with zero
    Normalize the kernel by dividing it by its sum to get psf
    Compute the Discrete Fourier Transform of the padded kernel to get $PSF1$
    Compute the squared magnitude of the complex DFT to get $PSF2$
    Compute the inverse filter by dividing $PSF1$ by $PSF2 + noise$
    Multiply the Fourier Transform of the image with the inverse filter to get $iPSF$
    Compute the inverse Discrete Fourier Transform to get the result $f_k^{c\prime}$
    Shift the result by half of the kernel size in both dimensions
  end for
  Combine deblurred $\{f_k^{r\prime}, f_k^{g\prime}, f_k^{b\prime}\}$ into f_k'
  Normalize histogram for f_k' to get f_k^{norm}
  Add the f_k^{norm} to the deblurred list $\{f_1^{deblur}, ...f_n^{deblur}\}$
end for
Return deblurred list $\{f_1^{deblur}, ...f_n^{deblur}\}$

Algorithm 5: Devonvolution-based deblur for frames using PSF estimation and Wiener Deconvolution

Input: Frames with blur $\{f_1, ...f_n\}$
Output: Estimated motion angle ang and motion magnitude d
for frames f_k in frame list $\{f_1, ...f_{n-1}\}$ **do**
  Detect, match and select the keypoints $\{p_1, ..., p_m\}$ from current f_k and next frame f_{k+1}
  Accumulate the motion estimates $flow_x$ and $flow_y$
end for
Calculate the average $avgflow_x$ and $avgflow_x$
Calculate estimated angle ang by $arctan(avgflow_x/avgflow_x)$
Calculate d by $\sqrt{avgflow_x^2 + avgflow_y^2}$
Return ang and d

Algorithm 6: Motion kernel estimation

comparison of a motion-blurred frame, a deblurred frame, and a gound-truth motion-free frame is provided in Figure 6

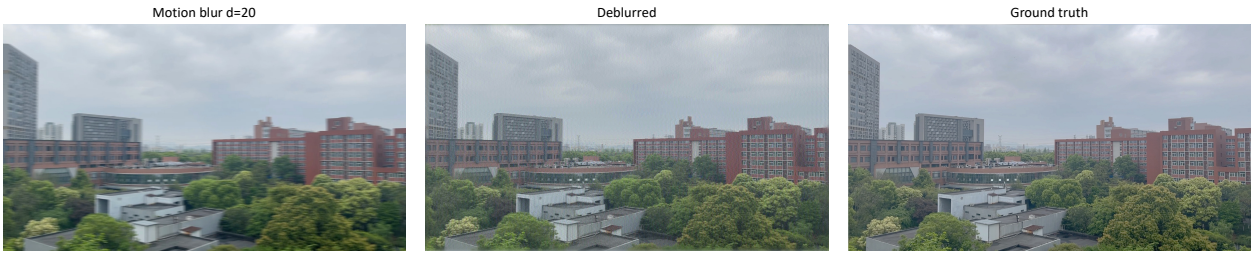

Figure 6: A comparison of a motion-blurred frame, a deblurred frame, and a gound-truth motion-free frame.

Feature 6: NAFNet-based low-quality video frame rectification

Nonlinear Activation-Free Neural Network (NAFNet) is originally introduced in [2], which decomposes and extracts essential components in existing state-of-the-art methods with no traditional activation functions (e.g., ReLU, Sigmoid, Softmax, GELU, etc.) to achieve superior Computer Vision related tasks with lower system complexity. I adapt the original NAFNet with a customized network structure and dataset to train a video frame deblurring model. The customized network and dataset are introduced as follows.

The backbone of NAFNet is a traditional single-stage U-Net [3] with skip connections. I modified the width of NAFNet to 32. The encode block number is set to $[1, 1, 1, 28]$ and the decoder block is $[1, 1, 1, 6]$. The middle block number remains the original setting as 1.

I create a dataset with a mixture of my own videos and an open-source video dataset GoPro [2]. My own video dataset contains 9 videos pairs (one pair contains a blurred video and a corresponding sharp video as network) covering 3 different magnitudes of blur for 3 different scenarios, with a total of 1067 pairs of frames. The GoPro dataset contains 2103 pairs of frames. The final training dataset is prepared by segmenting the frames in my own videos and GoPro dataset into sub-images with the size of $256 * 256$ (illustrated in Figure 7)

Some selected inference results of my customized model are presented in Figure 8. I test the performance of the deblurring model on both a video with artificially added blur and a video with authentic camera blur. The deblurring model can generate relatively sharp and clean deblur results.

Experimental Result

Test Results for 4 videos

I have conducted experiments with 4 test videos taken from different places in different weather conditions. The scenario complexity in these videos is also different as some videoes contain a large

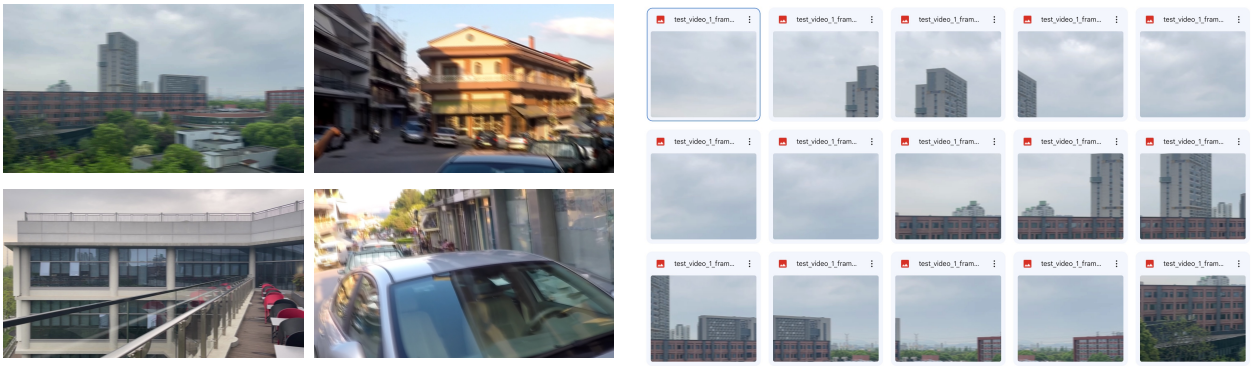

Figure 7: Illustration of sample frames in training data and segmentation operation for preparing the training dataset.

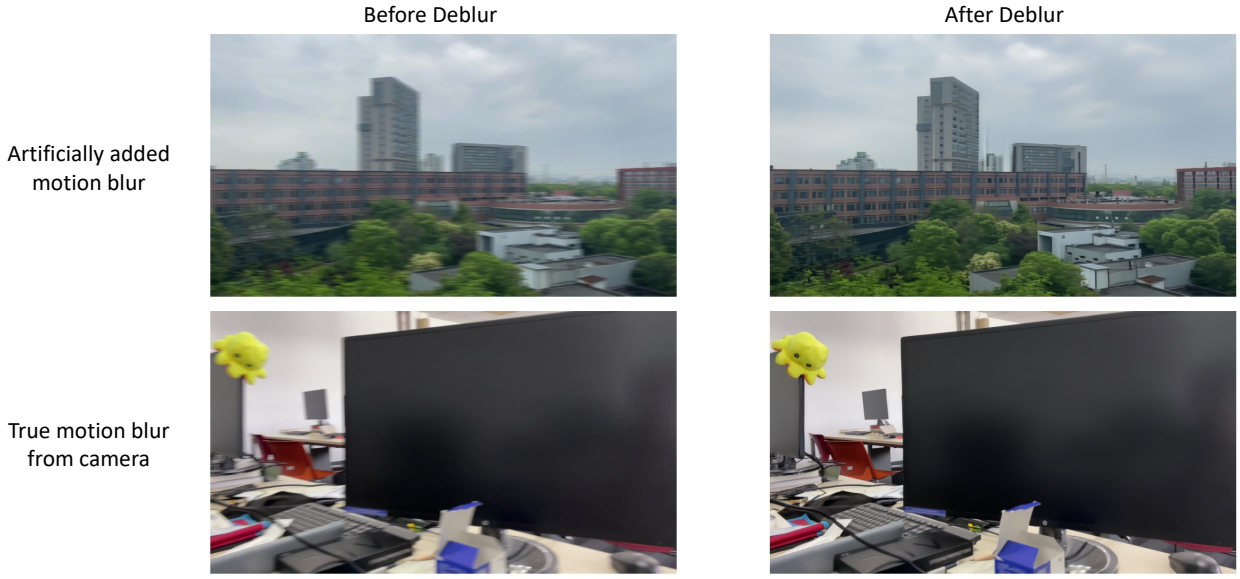

Figure 8: Illustration of modified NAFnet deblurring for single frames, one with artificially added blur and one with authentic camera blur.

portion of fine-detailed objects like trees while others contain more portion of buildings and water. The first 3 videos are taken by an iPhone 12 Pro Max main camera (35mm-full-frame-equivalent focal length of 26mm), at $1920 * 1080$ (full HD) resolution with 25 frames per second. The fourth video is taken by an iPhone 12 Pro Max short-telephoto camera (35mm-full-frame-equivalent focal length of 64mm), at $1920 * 1080$ (full HD) resolution with 25 frames per second.
The first video contains an almost equal amount of regularly-shaped buildings (mostly box-like), trees, and a cloudy sky. There are total 274 frames in this video. The panorama result (one with full content and one with tight crop) is shown in Figure 9.

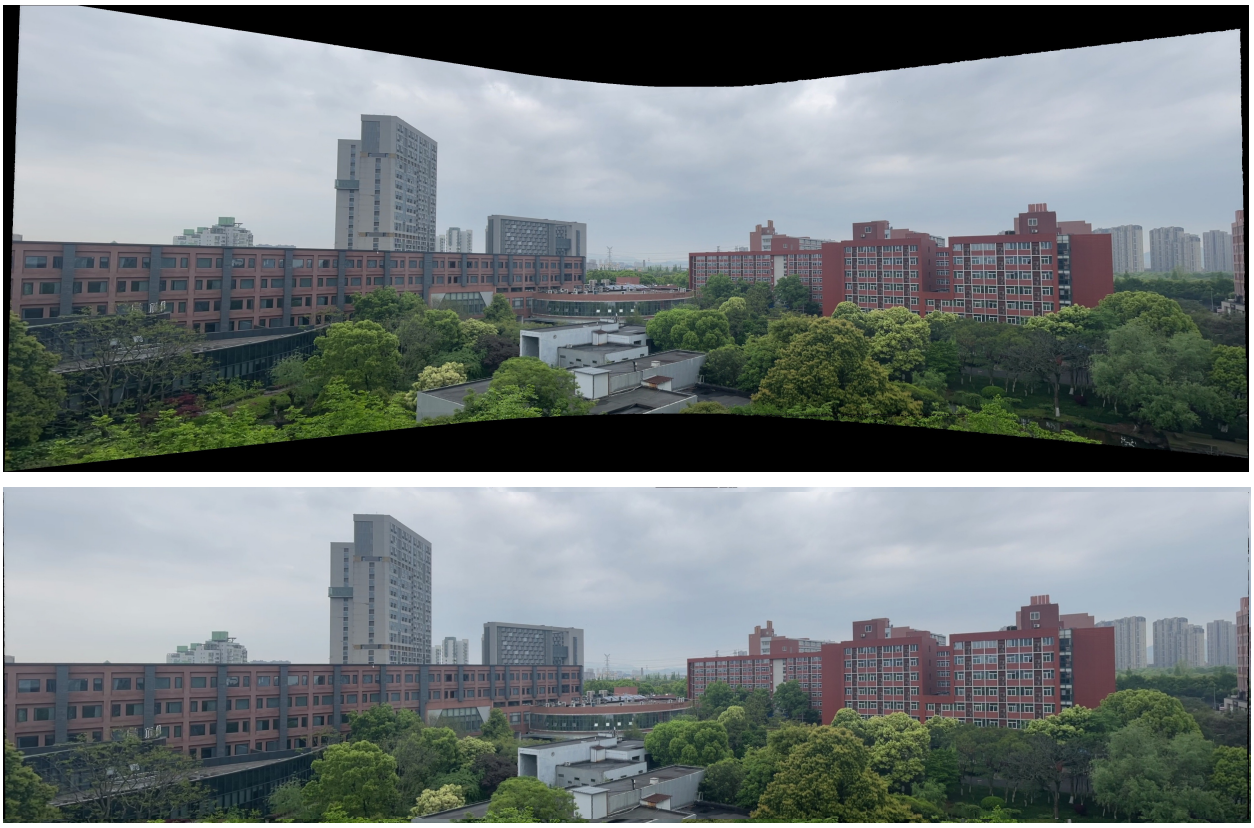

Figure 9: Generated panorama from test video 1 (the upper is full content version and the lower is perspective rectified version). Note that the width/length of panorama can be customized by users.

The second video contains irregularly-shaped buildings (freely-designed library), trees, water, water reflections, and a clear sky. There are total 325 frames in this video. The panorama result (one with full content and one with tight crop) is shown in Figure 10.
The third video contains very large portions of fine-detailed trees, some irregularly-shaped buildings (the Auditorium), water, water reflections, and a sunny sky with some clouds. This is a more challenging scenario since there are large amount of details to be matched and aligned. There are total 241 frames in this video. The panorama result (one with full content and one with tight crop) is shown in Figure 11.
The fourth video has similar scenarios as the third video, but it is taken with a short-telephoto lens.

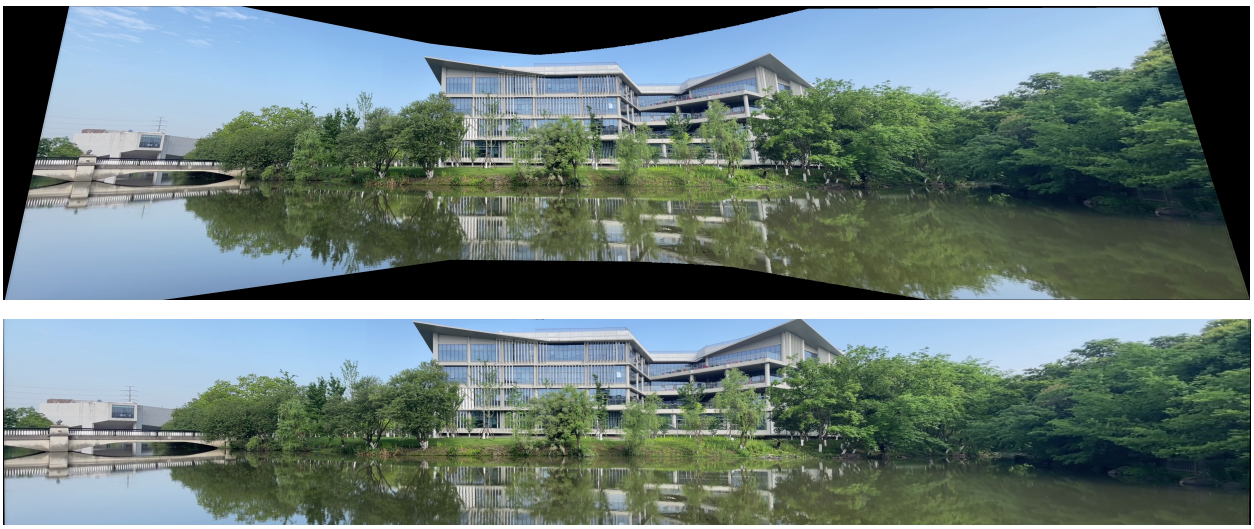

Figure 10: Generated panorama from test video 2 (the upper is full content version and the lower is perspective rectified version). Note that the width/length of panorama can be customized by users.

Figure 11: Generated panorama from test video 3 (the upper is full content version and the lower is perspective rectified version). Note that the width/length of panorama can be customized by users.

This is designed to test whether the implemented panorama stitching algorithm can handle videos taken at different focal lengths. There are total 225 frames in this video. The panorama result (one with full content and one with tight crop) is shown in Figure 12.

Figure 12: Generated panorama from test video 4 (the upper is full content version and the lower is perspective rectified version). Note that the width/length of panorama can be customized by users.

Test Results for Deconvolution-based deblur

To test the deblurring effect of the Deconvolution-based methods introduced above, I have conducted experiments on 3 motion-blurred videos with different amounts of blur ($d = \{10, 20, 30\}$). Figure 13 shows the deblur results for a selected single frame and Figure 14 shows the deblur results for the final panorama.

Test Results for NAFNet-based deblur

To test the deblurring effect of the NAFNet-based methods introduced above, I have conducted experiments on 3 motion-blurred videos with different amounts of blur ($d = \{10, 20, 30\}$). Figure 13 shows the deblur results for a selected single frame and Figure 14 shows the deblur results for the final panorama.

Figure 13: Illustration of deblur results (Deconvolution-based methods) for each frame at 3 blur levels.

Figure 14: Illustration of deblur results (Deconvolution-based methods) for the panorama at 3 blur levels.

Critical Analysis

Strength

Overall decent results in different scenarios and focal lens My implementation achieve visually natural, rich-detailed, and almost outlier-free results when testing on the 4 test video sets provided above. Even when the scenario is complex (e.g., Figure 10 and Figure 11), the stitched panorama can successfully match points and results in detail-preserved and clean images.

Adjustable projection plane Users can customize the plane that they want for the panorama to lie on, which gives them the freedom to decide which part of the video they want to highlight. For example, if a user wants to highlight more left regions of the panorama, choosing the left 1/3 index would be preferable (shown in Figure 2), and vise versa.

Automatic content detection and cropping My implementation automatically detects, locates, and crops out image content from the raw panorama which contains a large amount of black space. The loose crop preserves all content information and the tight crop removes all black space, which gives users abundant useful results.

Optimized implementation for fast computational speed The implemented code for processing the panorama stitching has been optimized by replacing all explicit for loop into numpy embedded

Figure 15: Illustration of deblur results (NAFNet-based methods) for each frame at 3 blur levels.

Figure 16: Illustration of deblur results (NAFNet-based methods) for the panorama at 3 blur levels.

index operations. Such optimization achieves 8.5x speed up when processing each frame in the code on the same computer.

Decent visual results and low-overhead using wiener deconvolution deblurring The deconvolution-based methods provide visually sharp and clean results, especially at low and medium motion magnitudes ($d = \{10, 20\}$ in Figure 13 and Figure 14). At higher blur magnitude, this method could still provide sharp results but there exists some alias. One of the most significant advantages of this method is its low overhead, since the average deblurring time for one frame is only $0.48seconds$.

Decent visual results and flexibility using NAFNet-based deblurring The NAFNet-based deblurring method achieves visually sharp and clean results at different blur magnitudes. The key advantage of this method is the flexibility in handling different magnitudes and different directions of blur, since it doesn't rely on the pre-estimation of motion kernels. Thus, this method could still provide relatively sharp and clean results in challenging or unexpected scenarios, while the deconvolution-based method might fail due to inaccurate estimation of motion kernels.

Weakness

Could have more automation support in the whole pipeline Current pipeline still requires users to define certain parameters including the length and width of the expected panorama. However, one possible improvement is to automatically estimate the range of the content in the raw panorama and

prompt the user with recommended values. Other additional adjustments could also be performed to improve the user experience.

Could have further improvement in computational speed of stitching panorama Although the code implementation is highly optimized (discussed above), some other tricks could be carried out to improve the computation speed of stitching panorama. One possible way is to not iteratively go through every frame but jump between frames to reduce the length of iteration. However, this possible improvement needs to balance the jump rate and the key point matching quality between the jumped frames.

Need more real motion-blurred videos Most deblurred videos tested are manually additive blurs, but real motion-blurred videos are lacking. The main reason I haven't include them here is that it's challenging to capture a motion-blurred videos with modern phone cameras that heavily rely on computational photography. My phone (iPhone 12 Pro Max) has advanced built-in deblur algorithms that cannot be explicitly turned off, and thus in most conditions it automatically rectifies videos. I've tried to pan the camera extremely fast, but the resulting videos contain consecutive frames that don't even have single matching points, which is useless.

Lack of flexibility of wiener deconvolution deblurring The deconvolution-based approach has limited flexibility since it relies on the estimation of motion magnitudes and directions. Although the estimation could be generally accurate in normal conditions, it could fail in extreme conditions (e.g., panning the camera while rapidly jittering the camera in different directions). Even when the motion kernel is accurately estimated, this method could generate some alias when the motion blur is very strong ($d = \{30\}$ in Figure 13).

Not enough very fine-grained detail of NAFNet-based deblurring The NAFNet-based deblurring could generate reasonably sharp and clean results in many scenarios but the deblurred frames still cannot achieve the same level of fine-grained detail as the ground truth. One possible method is to further finetune the kernel sizes and layers in the model to get preserve better details.

Could have further improvement in NAFNet-based deblurring speed Although the design of NAFNet is to reduce network structure complexity and improve computational speed, the inference time (average of 2.38 seconds per frame) is still 5x slower than non-ML based methods like the deconvolution-based deblurring (average of 0.48 seconds per frame). Thus, there could still be space for improvement by using techniques like pruning.

Reference

[1] Biswas, P., Sarkar, A. S., Mynuddin, M. (2015). Deblurring images using a Wiener filter. International Journal of Computer Applications, 109(7), 36-38.
[2] Chen, L., Chu, X., Zhang, X., Sun, J. (2022, November). Simple baselines for image restoration. In Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VII (pp. 17-33). Cham: Springer Nature Switzerland.
[3] Ronneberger, O., Fischer, P., Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18 (pp. 234-241). Springer International Publishing.