University of Nottingham
UK | CHINA | MALAYSIA

Panorama Generation from Videos

Computer Vision (COMP3065) Coursework01

Submitted May 6, 2024

Ganyi Mo 20215332

Section 1: Key Feature and Implementation

Section 1.1: User Interface

- **Purpose**: The UI provides a straightforward way for users to interact with the panorama generation application. It allows users to select video files and choose blending options for generating panoramas.

- **Implementation**:
    - The UI is built using the Tkinter library in Python, which is suited for creating simple, effective graphical interfaces.
    - Key elements include a button for selecting video files and an option menu for selecting different blending methods.
    - Upon selecting a video, the application processes the footage to generate a panorama, which is then displayed in a new window. This window includes scrollable views for easy navigation of larger panoramas.

```python
def select_video_file():
    """
    Open a file dialog to select a video file, generate a panorama from it, and display the panorama in a new Tk
    """
    # Open file dialog to choose a video file
    video_path = filedialog.askopenfilename()
    if video_path:
        # Get the selected blending option
        option = option_var.get()
        # Generate the panorama using the selected video and option
        panorama = generate_panorama(video_path, option)

        # Convert the panorama to a format compatible with Tkinter
        panorama_img = cv2.cvtColor(panorama.astype(np.uint8), cv2.COLOR_BGR2RGB)
        panorama_img = Image.fromarray(panorama_img)
        panorama_img = ImageTk.PhotoImage(panorama_img)

        # Create a new window to display the panorama
        window = tk.Toplevel()
        window.title('Panorama')

        # Create a canvas widget in the window and add scrollbars
        canvas = tk.Canvas(window, width=panorama_img.width(), height=panorama_img.height())
        scrollbar_x = tk.Scrollbar(window, orient="horizontal", command=canvas.xview)
        scrollbar_y = tk.Scrollbar(window, orient="vertical", command=canvas.yview)
        canvas.configure(xscrollcommand=scrollbar_x.set, yscrollcommand=scrollbar_y.set)

        scrollbar_x.pack(side=tk.BOTTOM, fill=tk.X)
        scrollbar_y.pack(side=tk.RIGHT, fill=tk.Y)
        canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Place the image on the canvas
        canvas.create_image(0, 0, anchor=tk.NW, image=panorama_img)
        canvas.config(scrollregion=canvas.bbox(tk.ALL))

        # Display the panorama using matplotlib for comparison (optional)
        plt.figure()
        plt.imshow(cv2.cvtColor(panorama.astype(np.uint8), cv2.COLOR_BGR2RGB))
        plt.show()

        window.mainloop()
```

Section 1.2: Histogram Matching

- **Purpose**: Histogram matching is used to adjust the colors of images from different video frames to ensure uniformity across the panoramic image.

- **Implementation**:
    - The method involves converting images to the YCrCb color space, which separates image luminance from chrominance, allowing more precise adjustments.

- Histograms for each channel (Y, Cr, Cb) are computed and aligned using cumulative distribution functions (CDF), ensuring that lighting and color are consistent across all frames.

```python
def match_histograms(source, template):
    """
    Match the histogram of the source image to that of the template image.
    """
    # Convert BGR to YCrCb color space (separating luminance from chrominance)
    src = cv2.cvtColor(source, cv2.COLOR_BGR2YCrCb)
    tmpl = cv2.cvtColor(template, cv2.COLOR_BGR2YCrCb)

    # Compute and apply histogram matching
    for i in range(3):  # Process each channel separately
        # Compute the histograms
        src_hist, _ = np.histogram(src[:, :, i], 256, [0, 256])
        tmpl_hist, _ = np.histogram(tmpl[:, :, i], 256, [0, 256])

        # Compute the cumulative distribution function (CDF) for both images
        cdf_src = np.cumsum(src_hist) / np.sum(src_hist)
        cdf_tmpl = np.cumsum(tmpl_hist) / np.sum(tmpl_hist)

        # Build a mapping function from the CDFs using interpolation
        interp_map = np.interp(cdf_src, cdf_tmpl, np.arange(256))
        src[:, :, i] = interp_map[src[:, :, i]].astype('uint8')

    # Convert the modified YCrCb image back to BGR
    result = cv2.cvtColor(src, cv2.COLOR_YCrCb2BGR)
    return result
```

Section 1.3: Multiband Blending

- **Purpose**: To blend images seamlessly at their boundaries, reducing artifacts and ensuring a smooth transition between images.

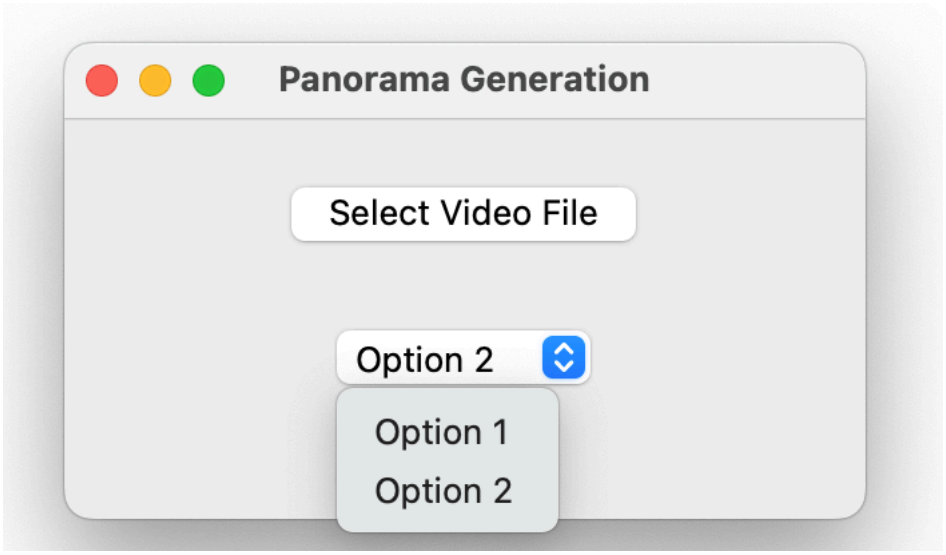
- **Implementation**:
    - The process starts by aligning images to the same size based on their levels in the Gaussian pyramid, ensuring each is divisible by a power of two.

- Gaussian pyramids are used to generate Laplacian pyramids for each image. These pyramids allow for layer-by-layer blending of images.
- A Gaussian pyramid of a mask (defining blending regions) is used to blend corresponding levels of the Laplacian pyramids of the two images.
- The final image is reconstructed from the blended pyramid, providing a high-quality seamless panoramic image.

```python
def multiband_blending(img1, img2, mask, levels=4):
    # Ensure that the dimension of the images and mask are divisible by 2**(levels-1).
    if img1.shape[0] % 2**(levels - 1) != 0:
        img1 = img1[:img1.shape[0] - img1.shape[0] % 2 ** (levels - 1), :, :]
        img2 = img2[:img2.shape[0] - img2.shape[0] % 2**(levels-1), :, :]
        mask = mask[:mask.shape[0] - mask.shape[0] % 2**(levels-1), :, :]

    if img1.shape[1] % 2**(levels - 1) != 0:
        img1 = img1[:, 0:img1.shape[1] - img1.shape[1] % 2 ** (levels - 1), :]
        img2 = img2[:, 0:img2.shape[1] - img2.shape[1] % 2**(levels-1), :]
        mask = mask[:, 0:mask.shape[1] - mask.shape[1] % 2**(levels-1), :]

    # Generate Gaussian pyramids for both images.
    gaussian_pyr1 = create_gaussian_pyramid(img1, levels)
    gaussian_pyr2 = create_gaussian_pyramid(img2, levels)

    # Generate Laplacian pyramids by subtracting each level of Gaussian pyramid from its expanded version.
    lap1 = []
    lap2 = []

    for i in range(levels-1):
        lap1.append(cv2.subtract(gaussian_pyr1[i], cv2.pyrUp(gaussian_pyr1[i+1])))
        lap2.append(cv2.subtract(gaussian_pyr2[i], cv2.pyrUp(gaussian_pyr2[i+1])))

    lap1.append(gaussian_pyr1[-1])  # Append the last level of the Gaussian pyramid.
    lap2.append(gaussian_pyr2[-1])  # Append the last level of the Gaussian pyramid.

    # Blend the Laplacian pyramids using the Gaussian pyramid of the mask.
    LS = []
    mask_pyramid = create_gaussian_pyramid(mask, levels)
    for i in range(len(lap1)):
        blended = mask_pyramid[i] * lap1[i] + (1 - mask_pyramid[i]) * lap2[i]
        LS.append(blended)

    # Reconstruct the blended image from the blended pyramid.
    LS = LS[::-1]
    img_blend = LS[0]
    for i in range(1, levels):
        img_blend = cv2.add(cv2.pyrUp(img_blend), LS[i])

    return img_blend
```

Section 2: Results

Section 2.1: User Interface



Section 2.2: Histogram Matching

The results show that histogram matching effectively standardizes the color and brightness of consecutive frames and significantly improves the visual

continuity of the panorama. However, there is also a set of images that show over-exposure of the stitched area.

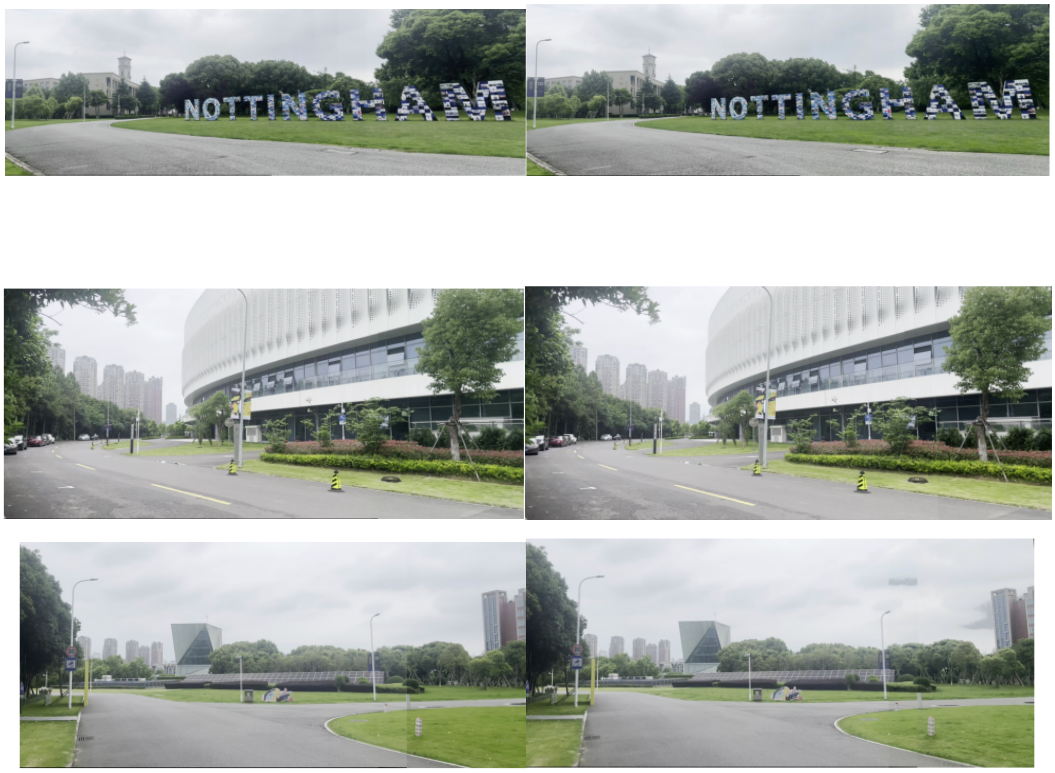


Figure: Non-Histogram Matching vs Histogram Matching

Section 2.3: Multiband Blending

The results of employing multiband blending in the panorama generation were somewhat underwhelming. Expected improvements in the seamless integration of images were not distinctly noticeable, and the technique introduced a general blurriness to the panorama. Comparatively, simple blending, while less sophisticated, did not result in a significant degradation in

image quality, calling into question the effectiveness of the multiband approach for this application.

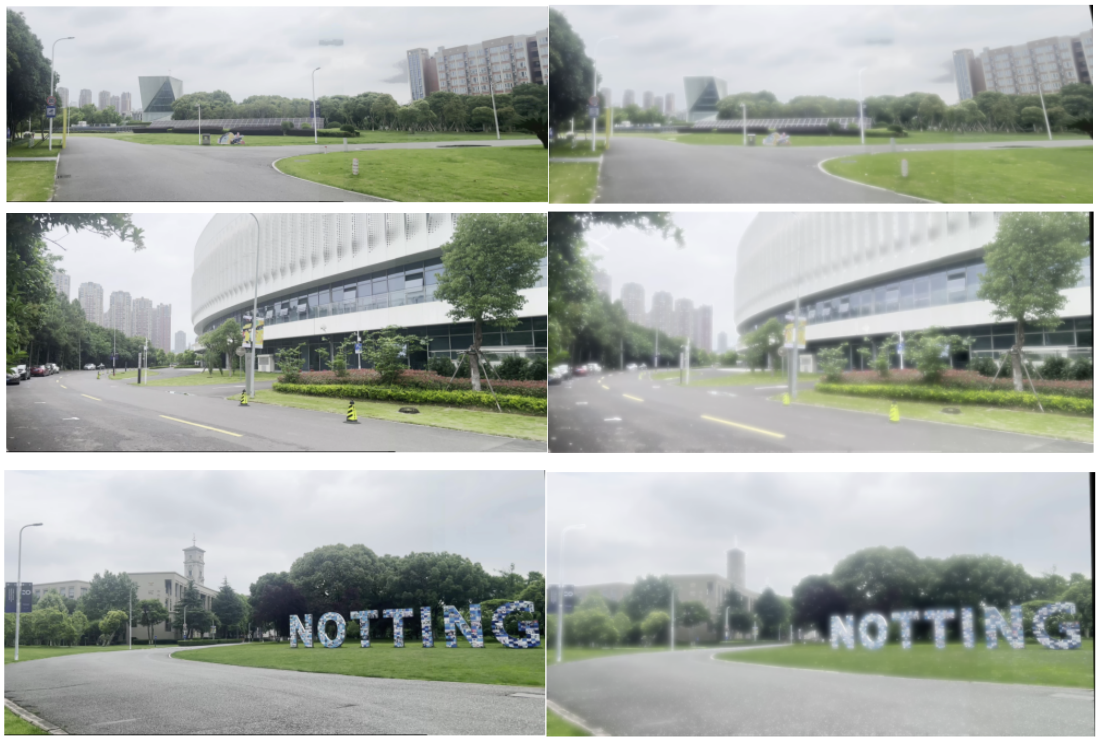


Figure: Simple Blending vs MultiBand Blending

Section 3: Discussion

Section 3.1: Histogram Matching

Strengths: This technique's primary strength lies in its ability to unify the appearance of video frames that would otherwise display inconsistent coloring and exposure. By aligning histograms across the frames, the panorama presents a more cohesive image, which is crucial for applications requiring high visual quality.

Weaknesses: Despite its advantages, histogram matching sometimes fails when the overlap area between frames is too small to lack enough information for accurate adjustment. This can lead to discrepancies in color and brightness, making some areas of the panorama look out of place. An improvement could be to incorporate a more adaptive approach that analyzes the differences in the overlapping regions more thoroughly and adjusts the target frame's colors and brightness based on a more complex model of the local image content.

Section 3.2: Multiband Blending

Strengths: The technique excels in creating seamless transitions by effectively handling differences in image content at the borders. However, I tried various combinations of parameters and it still didn't work as well as simple blend.

Weaknesses: The primary issue observed with multiband blending was an overall loss of image clarity, characterized by a blurring effect, particularly noticeable in high-detail areas. This outcome suggests a potential misalignment in the implementation of the Gaussian and Laplacian pyramids or an inappropriate choice of the number of levels used in these pyramids. Additionally, the blending did not significantly enhance the transitions between images, which was the primary objective of using this technique.

Potential Causes:

- **Pyramid Level Misconfiguration**: The chosen number of levels in the Gaussian and Laplacian pyramids might have been too high or too low, affecting the detail preservation during the blending process.