Hannah Nguyen
CS5330

Sky Pixel Identification Project Report

1. Traditional Image Processing Techniques and Rationale

- Color Space Conversion (HSV):
Sky pixels often have distinct colors. HSV (Hue, Saturation, Value) is robust in differentiating these colors, especially the hue component for blue or orange skies. Since the sky's color and brightness can vary significantly, HSV provides a more consistent separation than RGB space.
- Dynamic Thresholding:
This adapts to varying lighting conditions and sky colors. It's crucial for processing images with diverse sky appearances like clear, cloudy, or during sunrise/sunset. A portion of the image assumed to be the sky (top 25%) was used to calculate average hue, saturation, and value, which formed the basis for adaptive thresholding.
- Morphological Operations:
These operations were used to remove noise and small artifacts, refining the sky identification. They help in smoothing the resultant mask, closing gaps, and solidifying the detected sky region.
- Edge Detection (Canny):
Canny edge detection provided edge-based features for finding the horizon line, and distinguishing the sky from other regions. The Hough Line Transform was utilized to detect prominent lines that represent the horizon.

2. Implementation Process and Code Snippets

In terms of implementation, the process started with converting images to HSV. This was followed by dynamic thresholding, where I had a function in Python to calculate the thresholds based on the image's characteristics. Morphological operations were then applied to refine the mask, and Canny edge detection, along with horizon line detection, was performed using the Hough Line Transform to extract the sky from the image and black out the rest.

1. HSV Conversion:
- Converting images to HSV space for better contrast in the value channel.

```
def process_image_for_gradio(image):
    # Convert to HSV color space
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Calculate thresholds from the top part of the image
```

```
    lower_limit, upper_limit = calculate_thresholds_from_top(hsv_image,
top_fraction=0.2)
```

2. Dynamic Thresholding:

This is the function to calculate the threshold values for sky detection based on the top region (20 - 25%) of the image, which is assumed to contain the sky. Then a mask is acquired and validated based on the lower and upper limit values for a better result.

```
def calculate_thresholds_from_top(hsv_image, top_fraction=0.25):

# Define the top region of the image for sampling sky pixels
    top_region = hsv_image[:int(hsv_image.shape[0] * top_fraction), :, :]

    # Calculate the average hue, saturation, and value in the top region
    # to determine the characteristic colors of the sky in the image.
    avg_hue = np.mean(top_region[:, :, 0])
    avg_sat = np.mean(top_region[:, :, 1])
    avg_val = np.mean(top_region[:, :, 2])

    # Initialize the limits to cover a broad range of sky colors.
    # Adjust the ranges based on the averages calculated above.
    lower_limit = np.array([0, 0, min(180, avg_val - 30)])
    upper_limit = np.array([180, max(255, avg_sat + 30), 255])

    # Predefined hue ranges for typical sky colors.
    blue_hue_range = (100, 140)
    orange_hue_range = (10, 50)

    #more codes here to adjust the hue ranges
    #...
    return lower_limit, upper_limit
```

```
    # Initial mask based on the initially calculated thresholds
    initial_mask = cv2.inRange(hsv_image, lower_limit, upper_limit)

    # Validate and adjust the mask if necessary
```

```
    mask = validate_thresholds(initial_mask, hsv_image, lower_limit,
upper_limit)
```

3. Morphological Operations and Canny edge detection were used to remove small noise and find the horizon line:

```
# Apply morphological operations to remove small noise
    kernel_size = 5
    kernel = np.ones((kernel_size, kernel_size), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # Apply edge detection to help find the horizon line
    edges = cv2.Canny(mask, 50, 150)
```

4. Horizon Line Detection:

I have a function to find the horizon line and a function to apply a mask to exclude everything below the horizon. This helps to differentiate between sky and non-sky regions, particularly in cases where reflections or similar colors appear below the horizon.

```
def find_horizon_line(edges):
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=50,
minLineLength=80, maxLineGap=10)
    if lines is not None:
        lines = [l[0] for l in lines if abs(l[0][0] - l[0][2]) >
abs(l[0][1] - l[0][3])]
        # Sort lines by their midpoint y-coordinate
        lines.sort(key=lambda x: (x[1]+x[3])/2)
        # The horizon line is the one with the smallest y-coordinate
midpoint
        horizon_line = lines[0]
        return horizon_line
    return None
```

```
def mask_below_horizon(image, horizon_line):
    mask = np.zeros(image.shape[:2], dtype=np.uint8)
    cv2.line(mask, (horizon_line[0], horizon_line[1]), (horizon_line[2],
horizon_line[3]), 255, thickness=5)
```

```
    # Fill below the line to exclude anything below it
    cv2.floodFill(mask, None, seedPoint=(0, horizon_line[1] + 1),
newVal=255)
    return mask
```

3. Challenges and Solutions

- Variable Sky Conditions:

One of the most significant challenges I encountered was dealing with the variable conditions of the sky. The diverse appearances of the sky (e.g., clear, cloudy, sunrise, sunset) was also a significant challenge.
Solution: Dynamic thresholding and many trials and errors to manually adjust for the range of hues and saturations in HSV space. I also developed a function to calculate the threshold values (based on average hue, saturation, and value) for sky detection based on the top 20 - 25% part of the image, assuming that will be where the sky is. Then I also adjusted the hue range based on whether the average hue corresponds to typical blue skies or orange sunset/sunrise skies for the algorithm to better capture the sky.

- Horizon Detection:

It was a challenge to differentiate sky from non-sky regions in the image based on colors alone, especially with reflections or similar colors below the horizon.
Solution: I implemented horizon detection using edge detection and line-finding algorithms. After several tries, I found that using Hough Line Transform to focus on the most probable horizon line seemed to work best with my image dataset. After finding a horizon line, I applied a mask over anything under it. With this way, most reflection issues were solved.

- Cloud Detection:

I wanted to consider the clouds as part of the sky, but it was more difficult than I thought, because the algorithm would also detect anything white on the picture as the sky.
Solution: The thresholds were refined to include bright and less saturated regions, which are characteristic of clouds. However, this solution could be refined to be more accurate.

Reflection

1. **Effectiveness of the Approach**

Reflecting on the effectiveness of my approach, I found it quite successful in identifying sky pixels under various conditions. The use of HSV color space, horizon detection, and dynamic thresholding was particularly crucial in handling diverse sky appearances. There might be many more efficient ways to identify sky pixels, but at least my approach was efficient enough to be able to detect sky under different colors and conditions to a certain extent of correctness.

2. Limitations and Potential Improvements

However, there were limitations. The method struggled with highly complex scenes, for example: scenes where the sky meets the sea and both have the same color. Also, it was not effective for night sky or dark sky scenarios. There is also a lot of room for optimization, especially for high-resolution images. The algorithm could definitely be more sophisticated in handling the images, as there might be better functions for horizon and cloud detections that I am not aware of, and the parameters could also be more refined to best fit a wider range of images. I also see potential improvements in implementing machine learning techniques, which could enhance accuracy in more complex scenarios.

3. Learning Outcomes

Throughout this project, I gained valuable insights into traditional image processing techniques and learned the importance of adapting algorithms to diverse scenarios. There are so many more tools available out there that I did not have enough time to explore and test on. I also understood the challenges and limitations of non-ML approaches in image processing. There are a lot of trials and errors to find the best functions and then refine the parameters. I definitely recognize the value of iterative testing and optimization in algorithm development after this project.