Lab01 Report

Techniques

To segment the sky area in a landscape image, I explored four traditional computer vision techniques:

- **Color thresholding in HSV space**: HSV provides better separation between sky and non-sky regions compared to RGB space. A threshold is applied to extract blue/cyan regions in clear scenarios and red/orange regions in sunset/sunrise scenarios as potential sky.
- **Canny/Sobel edge detection**: Edges help exclude erroneous regions not connected to the top edge. Only edges connected to the top or left/right borders are kept as sky boundaries.
- **Morphological operations**: Opening and closing refine the sky mask by removing small noise and connecting gap regions.
- **Floodfill operation**: Some larger gap areas in the sky mask can not be filled by morphological operations. Using flood-fill technique can sometimes help further refine the sky mask.
- **Convex hull operation**: Sometimes larger gap areas in the sky mask are not closed areas, so it's hard to use morphological operations/flood-fill technique to refine the mask in these situations. Applying Convex hull to the mask contours can help overcome these problems.

These techniques are chosen because they allow extraction of the sky region by utilizing color, edge, and morphological information in a simple and effective way.

Implementations

I built two pipelines utilizing the techniques I mentioned above, because I found combining all of them can not produce satisfying results in most scenarios.

- **Pipeline1**: HSV thresholding based segmentation
  - convert image from BRG space to HSV space

    ```
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    ```

  - get blue and red areas in the HSV image

```
mask_blue = np.int8(img_hsv[:,:,0] >= 105) & np.int8(img_hsv[:,:,0] <= 135)
mask_red = np.int8(img_hsv[:,:,0] <= 30) | (np.int8(img_hsv[:,:,0] >= 165) & np.i
mask = mask_red | mask_blue
```

- use close morpho operation to denoise mask

```
kernel = np.ones((5, 5), np.uint8)
mask = cv2.morphologyEx(mask * 255.0, cv2.MORPH_CLOSE, kernel)
```

- extract the contours from the mask. I used `RETE_EXTERNAL` mode here to further exclude the inner gap areas.

```
mask = np.uint8(mask)
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

- filter contours that only exist on the top part of the image

```
filtered_contours = []
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    if y < height // 4:
        filtered_contours.append(contour)
```

- apply convex hull to all the contours. I did not end up using this techniques because it led to wrong results in pictures with mountain peaks.

```
use_convex = False
if use_convex:
    convex_hulls = [cv2.convexHull(contour) for contour in filtered_contours]
else:
    convex_hulls = filtered_contours
```

- fill the filtered contours

```
filled_mask = np.zeros_like(mask)
cv2.fillPoly(filled_mask, convex_hulls, 255)
```

- **Pipeline2**: Edge detection based segmentation
  - convert the image from BGR to gray scale

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- use edge detection operators(Canny/Sobel) to extract edge map. I tried both Canny and Sobel and found y-direction Sobel
  worked better in most scenarios, because the only edges I'd like to extract are the sky boundaries, most of which are in x-direction.

```
dx = 0  # dx
dy = 1  # dy
ksize = 3  # kernel size
edges_sobel = cv2.Sobel(img_gray, cv2.CV_64F, dx, dy, ksize)
edge_map = cv2.convertScaleAbs(edges_sobel)
```

- fill the top/left/right borders of the edge map with 255 to construct a closed contour with the sky boundary.

```
edge_map[0,:] = 255
edge_map[:, 0] = 255
edge_map[:,-1] = 255
```

- dilate edge map and find the inner level contours in the edge map in mode `RETR_CCOMP` .

```
kernel = np.ones((3, 3), np.uint8)
edge_map = cv2.dilate(edge_map, kernel, iterations=6)
ret, thresh = cv2.threshold(edge_map, 80, 255, cv2.THRESH_BINARY)
edge_contours, hierarchy = cv2.findContours(np.uint8(thresh), cv2.RETR_CCOMP, cv2
```

- filter the edge contours based on 2 conditions: 1. only exist on the top; 2. only the second level contour(inner contour).

```
edge_filtered_contours = []
for i, contour in enumerate(edge_contours):
    x, y, w, h = cv2.boundingRect(contour)
    if 0 <= y <= 30 and hierarchy[0][i][3] != -1:
        edge_filtered_contours.append(contour)
```

- fill the edge contours and apply close operation

```
edge_filled_mask = np.zeros_like(img_gray)
cv2.fillPoly(edge_filled_mask, edge_filtered_contours, 255)
# morpho filled mask
edge_filled_mask = cv2.dilate(edge_filled_mask, kernel, iterations=7)

# use close operation to fill inner holes
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
for _ in range(5):
    edge_filled_mask = cv2.morphologyEx(edge_filled_mask, cv2.MORPH_CLOSE, kernel
```

- use flood fill technique to fill larger gaps in the mask. I did not use `cv2.floodfill` to implement this because I
  found it's inconvenient it will sometimes fill the unexpected areas. I simply applied `cv2.findContours` to the mask and
  assumed that contours with area smaller than 20000 are gaps areas in the sky mask.

```
all_contours, _ = cv2.findContours(edge_filled_mask, cv2.RETR_LIST, cv2.CHAIN_APP
hole_contours = []
for cnt in all_contours:
    if cv2.contourArea(cnt) < 20000:
        hole_contours.append(cnt)
final_mask = np.zeros_like(edge_filled_mask)
cv2.fillPoly(final_mask, hole_contours, 255)
final_mask = cv2.bitwise_or(final_mask, edge_filled_mask)
```

Challenges

1. Thresholding based method is hard to separate sky and foreground area when sky and foreground have similar color, e.g.
   sky and sea/lake.
   - solution: filtering the contours based on their location in the image can solve some cases, but when the sky contour
     and foreground contour connect to each other, this solution fails.
2. Large gap areas in the generated sky mask are hard to be filled using morphological operations.
   - solution: using `cv2.findContours` and `cv2.fillPoly` technique can help locate the large hole and fill the inner hold
     with 255.
3. Advanced edge detection techniques, like Canny, incline to be affected by the subtle textures, like cloud, in the sky.

- solution: use simpler edge detection techniques, like Sobel, and only calculate the gradient in y-direction will be
  much more helpful when trying to detect more obvious edges and ignore the details, like the sky boundaries.