CSE 463 - Optional Homework

Sena Özbelen
1901042601
5 June 2024

1. **Image Loading**: Load the left and right stereo images for "sawtooth, venus, bull, poster, barn1, barn2". I used the image 2 and 6 since the given disparity maps are for these images according to the dataset webpage. I used only the disp2.pgm disparity map. The website says "a value of 100 in disp2.pgm means that the corresponding pixel in im6.ppm is 12.5 pixels to the left.". Therefore, we can use this information for the comparison part.

2. **Preprocessing**: Convert the images to grayscale for easier processing and ensure the pixel values are in float32 format for further processing.

3. **Segmentation**: Apply SLIC (Simple Linear Iterative Clustering) to segment the images into superpixels. Superpixels are the result of perceptual grouping of pixels, or seen the other way around, the results of an image oversegmentation. Oversegmentation is a process where an image is divided into many small segments or regions, often more than necessary, which can then be merged or processed further.

4. **Centroid Calculation**: Calculate the centroids of each segment in both the left and right images. Each pixel belongs to a segment. For each segment, we create a mask and we get the centroid of the segment by calculating the mean of the coordinates.

```
def segment_image(image, n_segments=200, compactness=10):
    segments = slic(img_as_float(image), n_segments=n_segments, compactness=compactness)
    return segments

def get_segment_centroids(segments):
    centroids = []
    for seg_val in np.unique(segments):
        mask = segments == seg_val
        coords = np.column_stack(np.where(mask))
        centroid = coords.mean(axis=0).astype(int)
        centroids.append((centroid[1], centroid[0]))
    return centroids
```

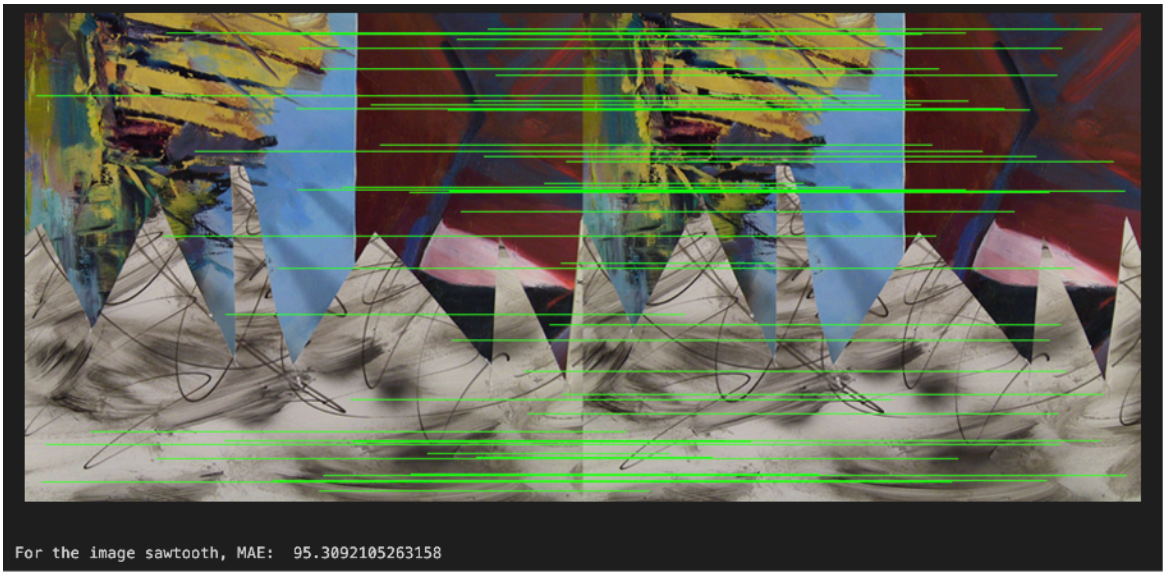
5. **Segment Matching**: Match the segments based on the centroids using the epipolar constraint and pixel value similarity. It iterates over each segment in the left image to match with the right ones. It finds the segment pairs by considering epipolar constraint. Then, it calculates the pixel value distance between the centroids. The segment that has the

minimum distance is the match for the left segment. We will do this until we iterate over all the segments in the left image.

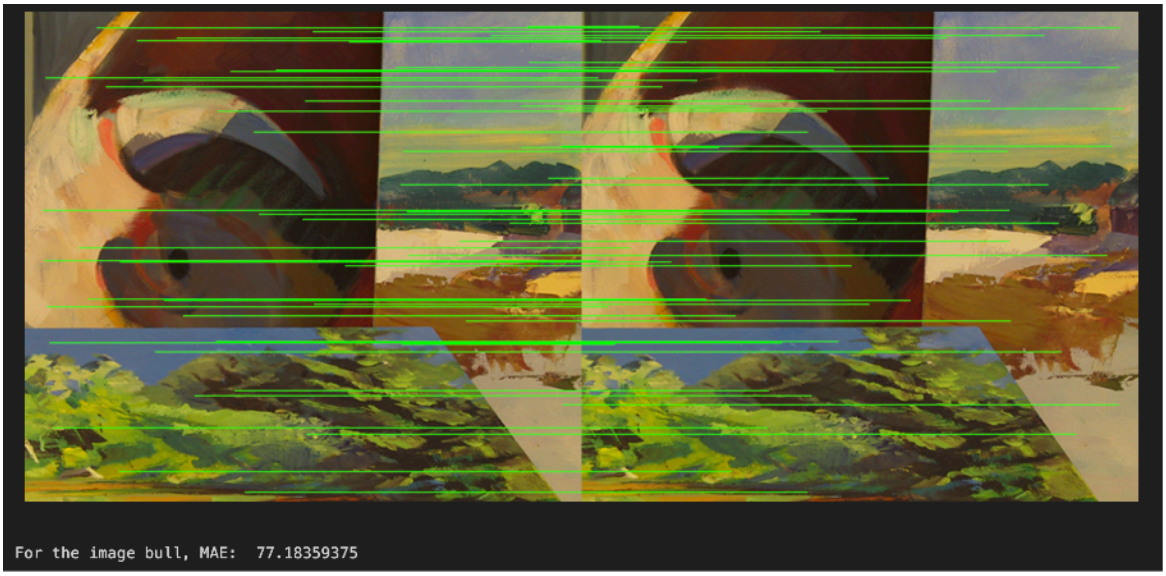
```
def match_segments(left_image, right_image, left_segments, right_segments):
    left_centroids = get_segment_centroids(left_segments)
    right_centroids = get_segment_centroids(right_segments)

    matches = []
    for left_c in left_centroids:
        min_distance = float('inf')
        best_match = None
        for right_c in right_centroids:
            if left_c[1] == right_c[1]:  # Epipolar constraint
                distance = np.linalg.norm(left_image[left_c[1], left_c[0]] - right_image[right_c[1], right_c[0]])
                if distance < min_distance:
                    min_distance = distance
                    best_match = right_c
        if best_match:
            matches.append((left_c, best_match))

    return matches
```

6. **Error Calculation**: Compare the matched segments' shifts with the ground-truth disparity values to compute the Mean Absolute Error (MAE). It basically loads the disparity map and compare the results. We know that "a value of 100 in disp2.pgm means that the corresponding pixel in im6.ppm is 12.5 pixels to the left.” Therefore, the difference between the matched centroids’ coordinates should be close to this value. We will calculate the error by using Mean Absolute Error.

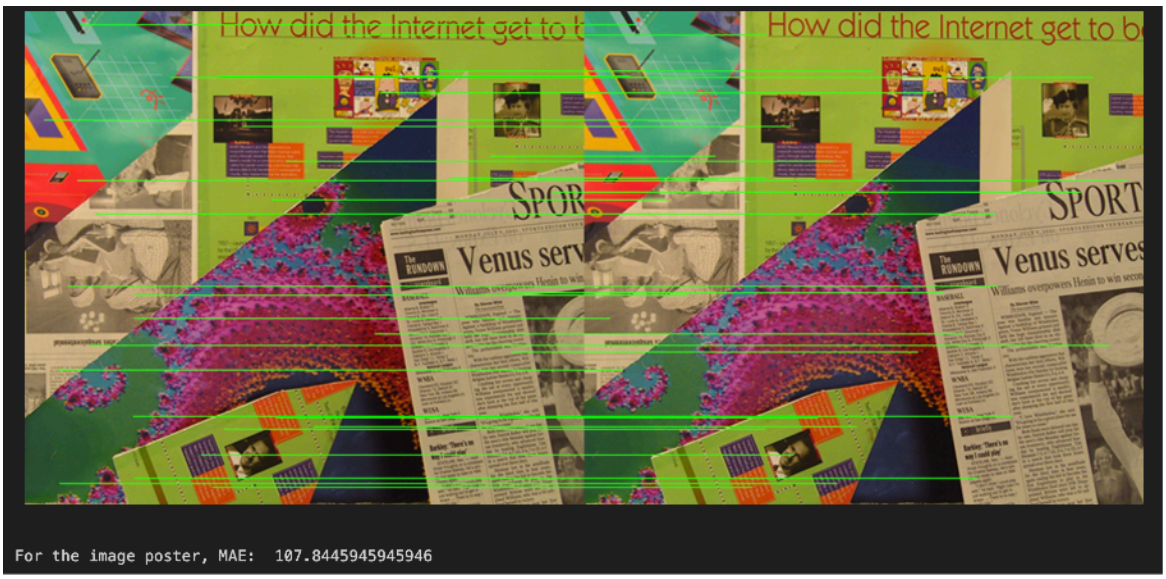
Results



For the image venus, MAE: 91.44132653061224



For the image bull, MAE: 77.18359375



For the image poster, MAE: 107.8445945945946

These errors are mostly because of the matching algorithm. We can see that some matches are not correct because we basically match those segments by considering the pixel value difference of the centroids. If two segments' centroids have the minimum distance in terms of pixel value, they are matched. Therefore, this increases the error value.