Computer Vision CS-GY 6643 - Project 2

Sumedh Parvatikar, sp7479@nyu.edu

1 Image Transformation and Stitching

(a) Manual Feature Identification:

As part of Manual identification, we need to identify points which are similar in both the images. These acts as features for stitching the image further. These points help us calculate the homography matrix for the image which can then be used to warp and as mentioned earlier stitch the images. There are a lot of points which are common among the images uploaded as we will see in the Automated feature detection part. We showcase some sample points which are manually marked to show the features in both the images. The manual features can be noticed in Figure 1.1(a), Figure 1.1(b), Figure 1.1(c). Figure 1.1(a) contains the features from Image 1 and Image 2. Figure 1.1(b) contains the features from Image 2 and Image 3. Figure 1.1(c) contains the features from Image 1 and Image 3

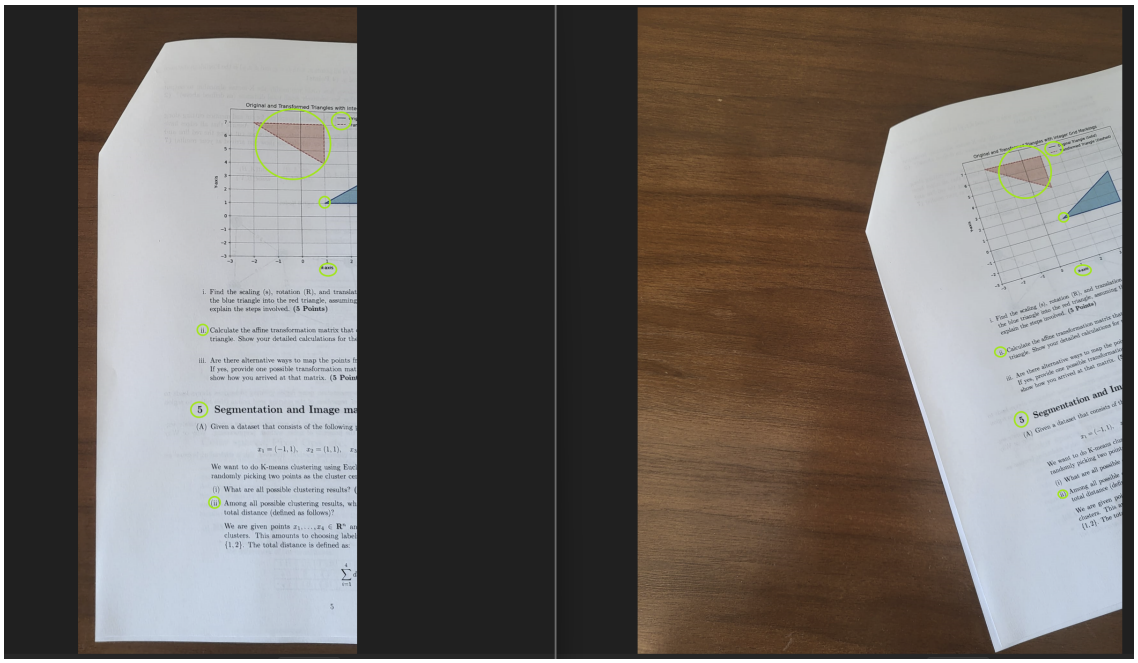

Figure 1.1(a)

(b) Automated Feature Identification:

To automatically detect features we use the **SIFT** feature descriptor, which automatically detects features from two features. These features can then further be used to perform more complex operations like stitching, warping and more. We use the openCV library BFMatcher method to draw the matches on the images between two images. For our ultimate objective to stitch the images, we initially start with finding the descriptor of two images, and stitch these images and later on perform the same set of operations on the intermediate image and the final image, thus effectively stitching all the images. The SIFT features can be observed in Figure 1.2(a) which contains features marked in fluorescent colors. From the title of the figure, we can note that there are around at least 4k key-points in these images.

As part of Figure 1.2(b) and Figure 1.2(c), we can see mapping between the images showing that there are common keypoints which can be leveraged in our downstream pipeline to stitch the images.

(c) Image Stitching: Image stitching is a complex process and requires two images and a **homography** matrix which basically contains the transformation details. We perform image warping before returning a blend of both the images. Image warping is a process which transforms the source image

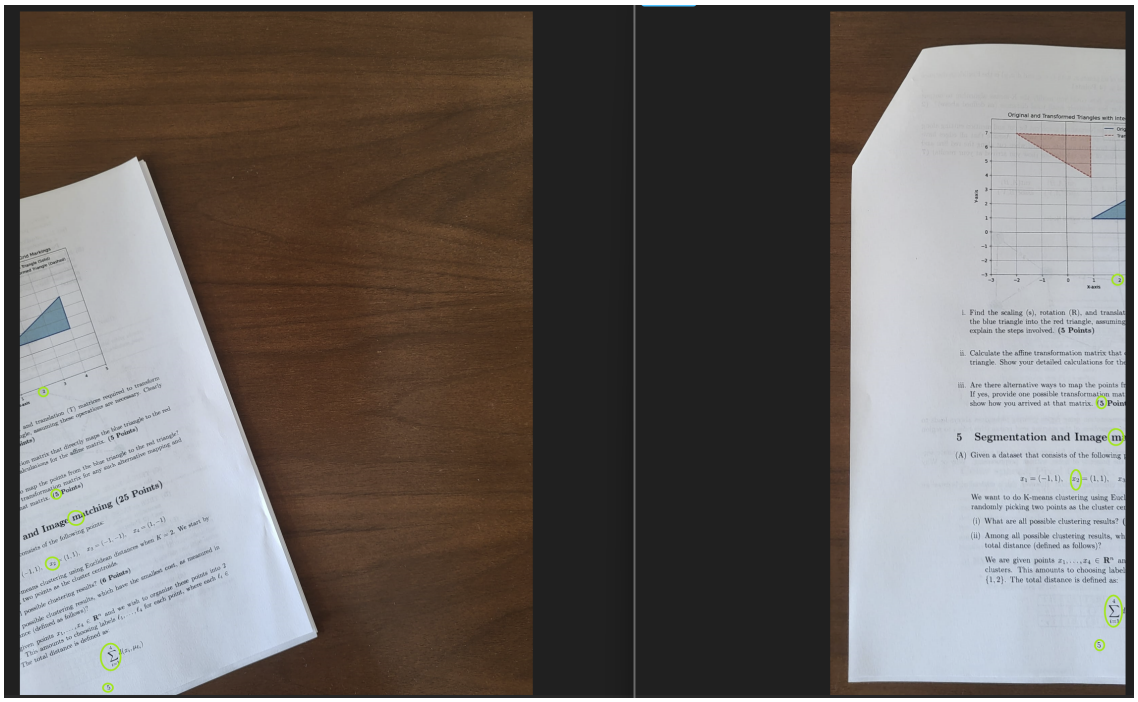

Figure 1.1(b)

into a different shape or orientation using homography matrix. In the process of warping, we use bilinear interpolation in case the transformed points don't align with the pixel grid points.

Now, one of the most important parameters required to stitch the image is the homography matrix. We calculate the homography matrix by initially detecting kepoints (using SIFT detector) and then further extract descriptors and match as done previously in the Automated Feature Identification phase. After the completion, we proceed to sampling points and randomly select a subset of 4 point pairs for our homography matrix (since homography matrix contains 8 variables [degrees of freedom] and requires at least 4 correspondences to solve the equation and get values for the variables). After the above process, we compute the homography using the SVD method by getting the eigenvectors and eigenvalues. We then proceed to apply the estimated homography on all the matches and compute the reprojection error. We then proceed to count the number of inliers below a threshold. If we get a new best score for the model, we update our values accordingly storing the best inlier count as well as the respective homography matrix. We repeat the process for a predefined number of iterations or until the best consensus set is found. In our case, we use the concept of adaptive iteration limit where based on the inlier ratio adjust the number of iterations. This is a more refined approach allowing us to reach the convergence faster with efficient use of resources. From the notebook, we can validate that the number of inlier statistics we observe is less than 40 iterations even though we initially define the iterations as 3000. Finally, we can see the inlier matches by the ransac model as part of Figure 1.3(a). We've found around 195 inlier matches from the model above. A visual graph can be observed containing the matches between the two images as part of Figure 1.3(b). We can also see the homography matrix of image 1 and image 2 as part of Figure 1.3(c)

In our final phase, we use the homography matrix calculated above in our final method to stitch both the images. We can see final stitched version of the image 1 and image 2 as part of Figure 1.3(d). This particular stitched image acts as in intermediate image for our stitching with the third image.

We follow the same process as mentioned for stitching newly stitched intermediate image with image 3. The SIFT keypoints can be noticed in Figure 1.3(e) We do this by initially using SIFT to detect keypoints and then extract the descriptors and match them in the two images. We then proceed towards sampling and compute the homography matrix. The homography matrix is attached as part of Figure 1.3(f). We also visualize the inliers captured as part of the process in Figure 1.3(g) and Figure 1.3(h).

Our final stitched image can be viewed as part of Figure 1.3(i).

Figure 1.1(c)

(d) Rectification: As the stitching of the document is reasonable enough allowing the user to read, we didn't perform any rectification separately in this section.

(e) Comparative Analysis: Although we did not perform manual feature extraction and stitching, but based on understanding from class and some knowledge from online sources, here's high-level overview:

Manual feature selection is time consuming and involves expertise in carefully choosing points with high relevancy for the homography matrix creation. On the other hand, it is also useful in some cases where the automated feature detector fails. Another advantage of manual feature selection would be avoiding the complex computations and usage of resources to retrieve the homography matrix.

As we did not perform stitching using manual feature selection, I don't think I'm in a good position to comment on the smoothness of blending, accuracy in-detail.

The automated feature selection is complex but is highly scalable and performs stitching faster compared to manual feature selection. It generally works well in all scenarios except for some niche sections where we would have to focus on specific parts of the image, which the automated feature selection won't be able to adhere. The accuracy of automated feature is also very decent.

2 Hough Transform

(a) Implementing Hough transform for runway: Hough transform is a feature extraction technique to find imperfect instances of objects within a certain class of shapes by a voting procedure. In our case, we first have to identify the edges of the runway after which we can apply the hough transform technique to detect straight lines. We first start with a Sobel operator to detect edges in our runway image. But the Sobel operator would detect all the edges, and we are interested in detecting the edges only along the runway so that the shuttle can land properly. We know from the runway image, that the runway is usually marked in white color highlighting it perfectly. We leverage this part, and apply a white color mask on the image by converting the image to hsv (which is better when distinguishing color intensities). We can see the edges of the runway clearly as part of the Figure 2.1(a).

After detecting the edges, we then apply hough transform on these edges and perform voting. We use the standard equation of the line

$$xcos\theta + ysin\theta + c \tag{1}$$

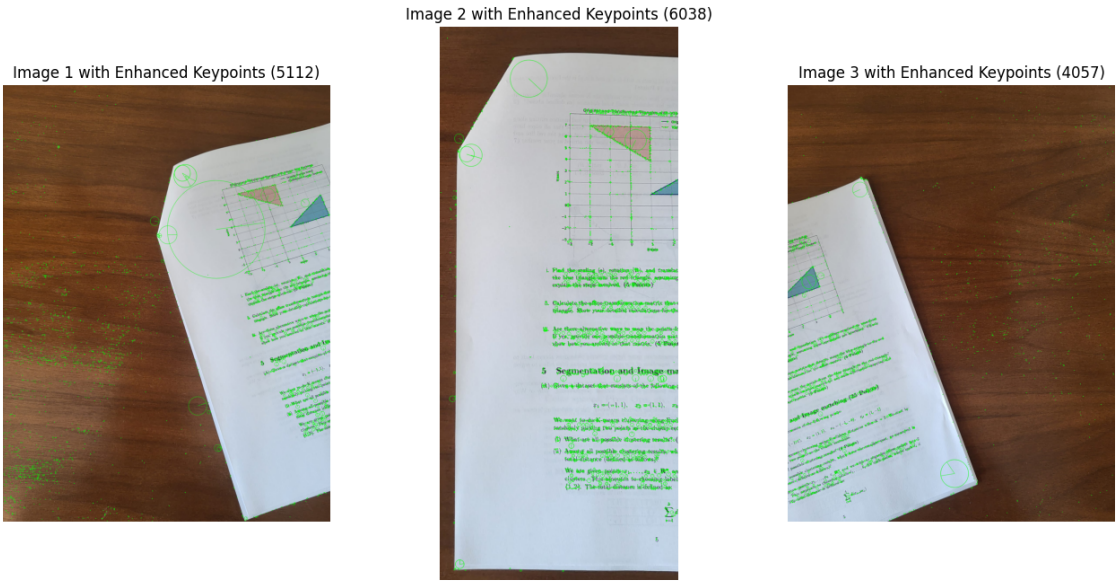

Figure 1.2(a)

where c is maximum distance possible. We perform this from -90 to 90 degrees in theta and store the values in an accumulator which would contain the votes for the lines. After calculating the values for the accumulator, we provide a value for the peaks and threshold which will be used to identify peaks. Using these peaks, we draw the hough lines. We can observe the final image where we've successfully detected the runway lines as part of Figure 2.1(b)

(b) Implementing Hough transform for landing pad For the landing pad, the concept of detecting the circles on the landing pad is a little different than for a runway. As we can see here, we have to detect two circles, inner and outer circle which will allow for safe landing by the SpaceX booster. As we can see from the image, that the landing pad is white in color and we can use this to our advantage by applying a white mask and focusing on detecting circles only in this segmented image. We can see the edges as part of Figure 2.2(a) where we can observe the edges only in white areas.

After finishing the edge detection, we now proceed to the voting process similar to the one we performed for runway lines. In this particular scenario, we perform for two circles, one larger and one smaller. As we already know the radius ranges of the bigger and the smaller circles, we pass these ranges to our hough transform function. In our hough transform function, we define the accumulator comprising of a, b and r variables. The accumulator would contain votes for most probable points with circles falling the user-defined range. We use the same method for our smaller circle detection too. After getting the accumulators comprising of votes for both the circles, we then set a threshold for votes and extract parameters from the accumulator greater than that threshold. These parameters would be used to draw a circle in the image. We can see the circles drawn in Figure 2.2(b).

(c) Overlaying of Images From Figure 2.1(b), we can see the overlaying of the hough transform edges on the original image.

From figure 2.2(b), we can see the overlaying of the hough transform circles on the original image. In our particular case, we have an image of overlaying only the large circle, only the small circle and finally both the circles in the bottom right.

(d) Improvements

The detection of the lines and the circles in case of the landing pad was satisfactory as seen from the overlayed images. With these results, the spaceship and the SpaceX booster should be successfully able to navigate and detect their respective runway and landing pads. That being said, there is a lot of scope to improve the processing time for our results (especially for circles) which takes a lot of time. This is the case as we compute the parameter for each theta does increasing our search space. We can reduce our search space by utilizing advanced techniques in finding optimal parameters to get high voting numbers.

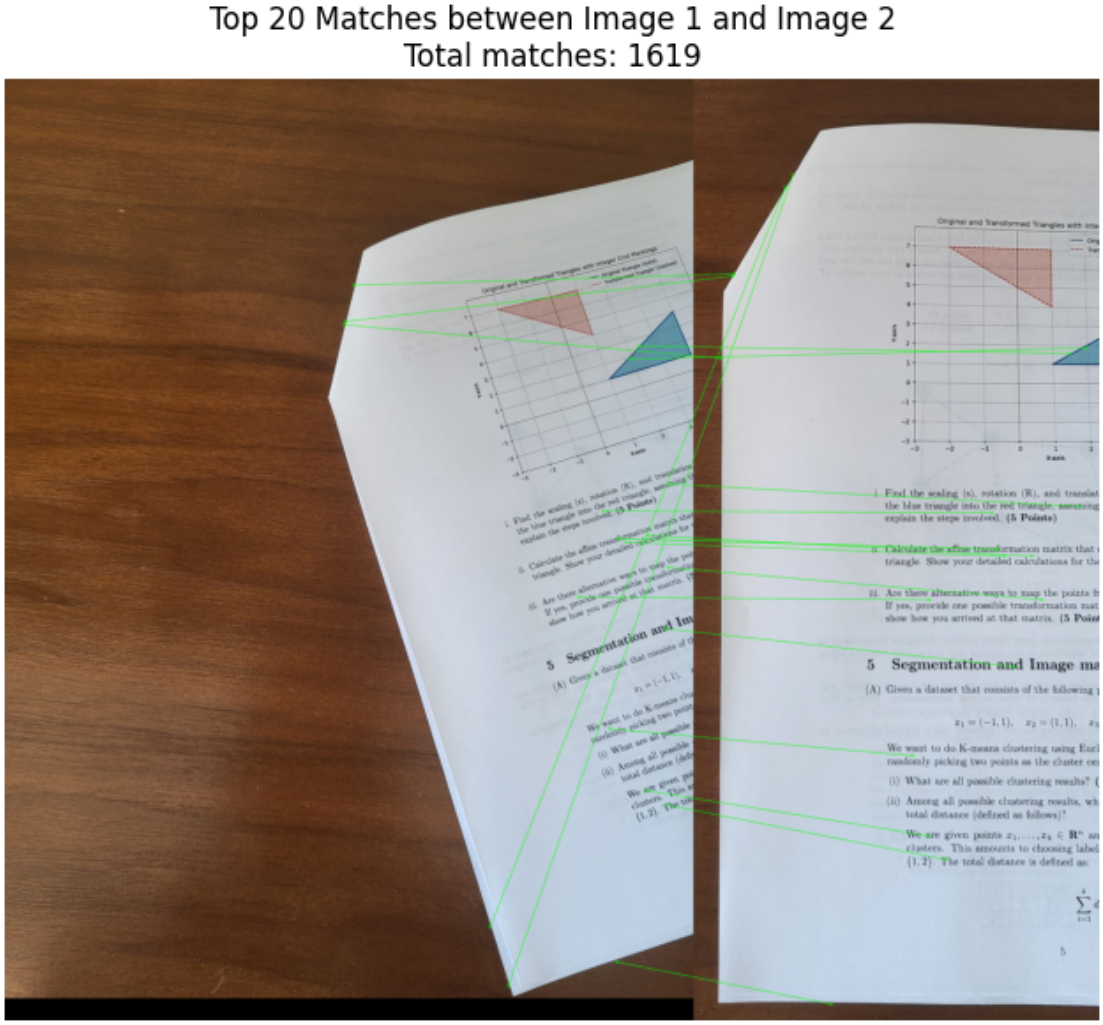

Figure 1.2(b)

3 Segmentation

(a) Mean Shift Segmentation

Brief Description : The mean shift segmentation algorithm is a non-parametric clustering method that segments an image by iteratively shifting data points toward the densest regions in the feature space. It's very promising and effective for segmenting regions with similar features like color and texture. It operates by shifting a kernel or window iteratively toward regions of higher density in a combined spatial and feature space, eventually converging to cluster centers. Each pixel is associated with the mean of its neighborhood, and this process helps grouping similar pixels into segments.

In our implementation, we convert the image to LAB color space for uniform color comparison and each pixel is transformed to a point in a 5D space (spatial:x, y, color:L, A, B).

For each pixel, we determine the neighbors based on the user-defined parameters spatial radius and color radius. We then compute the mean of spatial and color values of the neighbors. Finally we shift the mean toward the calculated mean until convergence or a maximum number of iterations is reached.

To smoothen the region, we replace the pixel's value with the mean of its cluster which allows us to group similar regions. In the end, we convert our vectors back into image space from 5D space.

In our implementation, we finally apply a red mask, to only allow for the STOP sign related colors to be visible thus effectively providing us with a technique to detect the STOP sign using this method. We can see the results as part of Figure 3.1(a)

Strengths :

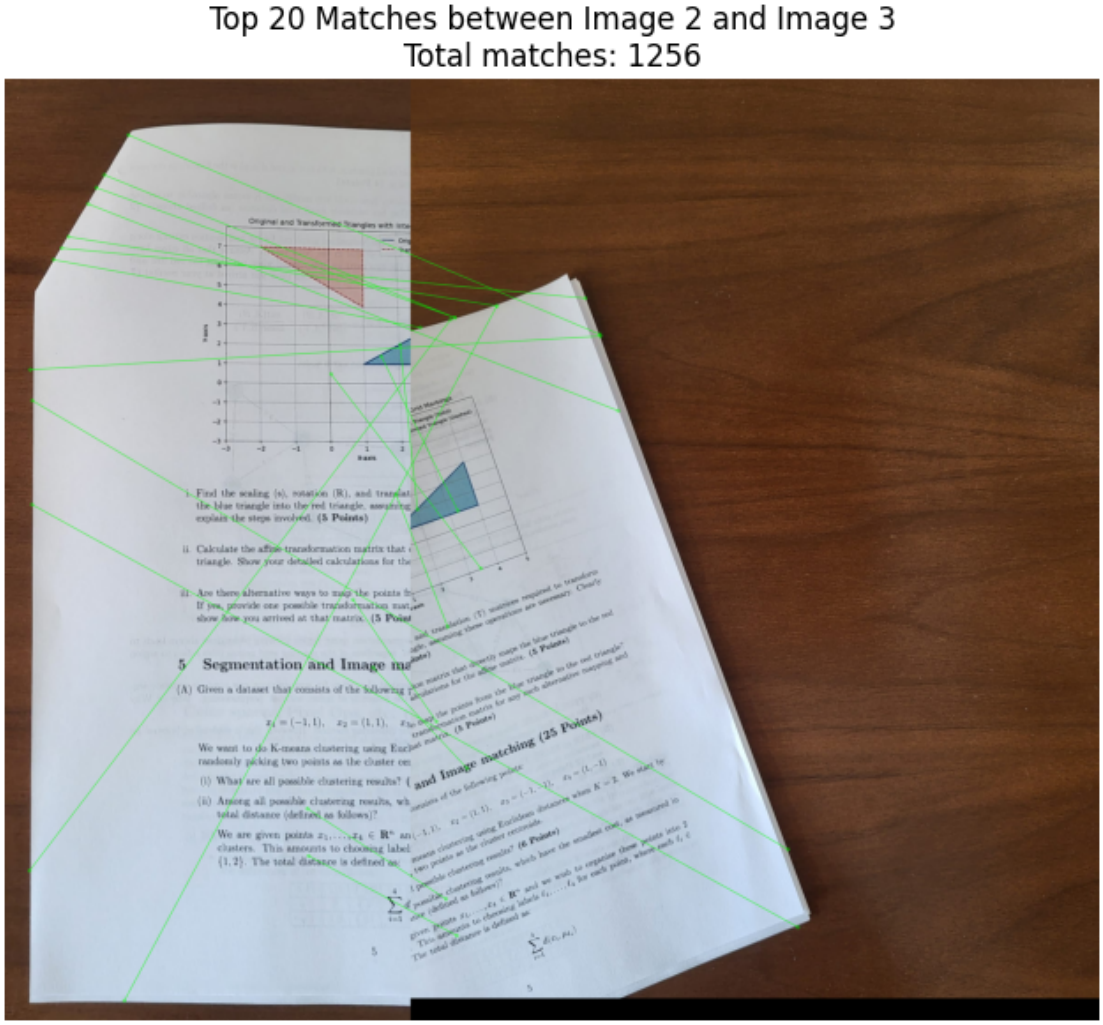

Figure 1.2(c)

(a) The boundary detection approach of Mean shift segmentation can be very effective considering it's algorithm's edge-preserving nature.

(b) The Red color regions of the STOP sign will be grouped effectively when using LAB color space.

(c) The Mean shift segmentation algorithm uses non-parametric approach and does not require prior knowledge of the number of clusters or segments in the image unlike K-means.

(d) MSS works well for arbitrary shapes as it doesn't assume the clusters needing to be spherical.

(e) This algorithm has the capability to work with different feature spaces by defining appropriate kernels.

(f) MSS is robust to noise due to the kernel density estimation

Weaknesses :

(a) The background objects or noise with similar colors may merge with the sign if the parameters are not tuned correctly.

(b) The iterative nature of the algorithm along with kernel density estimation can be slow especially for large images or high-dimensional data.

(c) MSS performance is heavily depended on bandwidth parameter viz. kernel size. Incorrect bandwidth can lead to over-segmentation or under-segmentation

(d) High-resolution images require significant memory and computation due to large number of feature points.

(e) MSS does not inherently recognize shapes, so the octagonal shape of STOP signs needs additional processing.

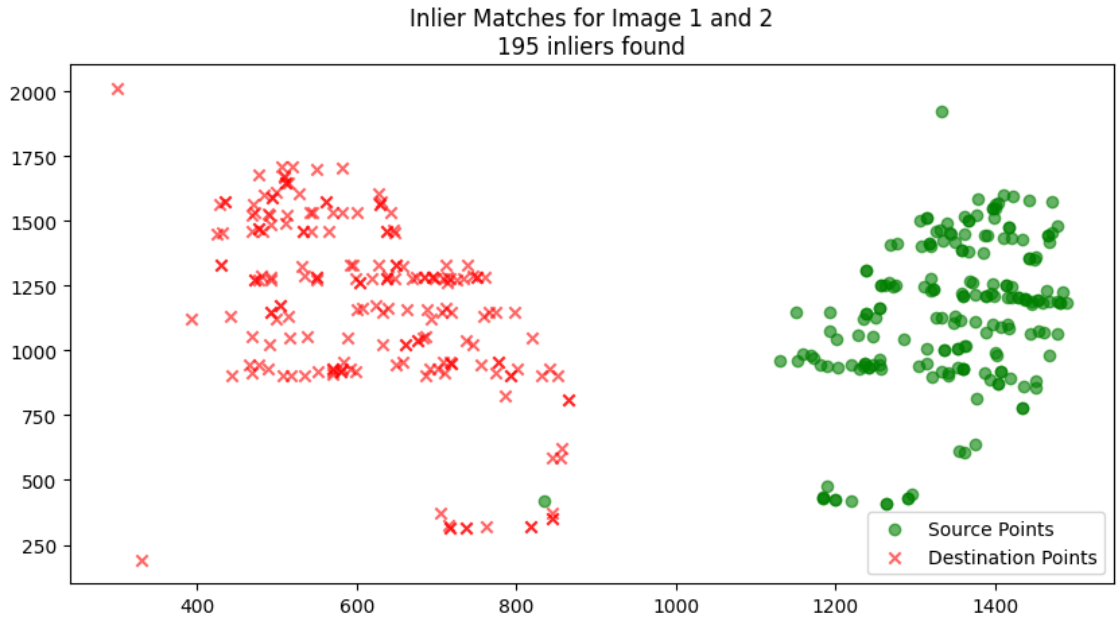

Figure 1.3(a)

(b) Normalized Graph Cut Segmentation

Brief Description : Normalized graph cut (N-Cut) is a graph based image segmentation technique that partitions an image into regions by minimizing a criterion that balances the similarity within regions and the dissimilarity between regions. It can do this by solving an eigenvalue problem on a graph constructed from the image's super-pixels.

In our first step we compute super-pixels which basically reduces our memory consumption and also allows us to effectively find segments. Super-pixels are groups of pixels with similar spatial and color characteristics, reducing computational complexity. We use the SLIC (simple linear iterative clusterin) algorithm to compute our super-pixels. Once we have our super-pixels, we construct a graph where each node is a super-pixel and the edges would have weights which would be computed based on color similarity. Once the graph is constructed we compute the normalized laplacican matrix by following the approach L = I - D$\hat{}$(-1/2)WD$\hat{}$(1/2), where D is the degree matrix of W and I is the identity matrix.

We then proceed to compute the second smallest eigenvector of L, which provides the optimal portioning criterion for segmentation. We partition the super-pixels based on the sign of the eigenvector values, dividing the image into foreground and background.

In the end, we apply the mask on the segmented regions and copy the same color scheme as present in the original image thus providing an effective view of the segmented image.

We can view the results as part of Figure 3.2(a).

Strengths :

(a) The Normalized graph cut segmentation (NGCS) provides a global optimization framework that minimizes the normalized cut, ensuring coherent segmentation

(b) The super-pixel based approach reduces the computational cost effectively compared to other graph-based segmentation methods.

(c) It preserves important STOP sign boundaries by grouping pixels with similar properties.

(d) It can effectively handle segmenting regions with complex or irregular shapes, as it does not assume predefined cluster geometry

(e) The NGCS focuses on the overall graph structure and hence it is less sensitive to noise compared to purely local segmentation methods.

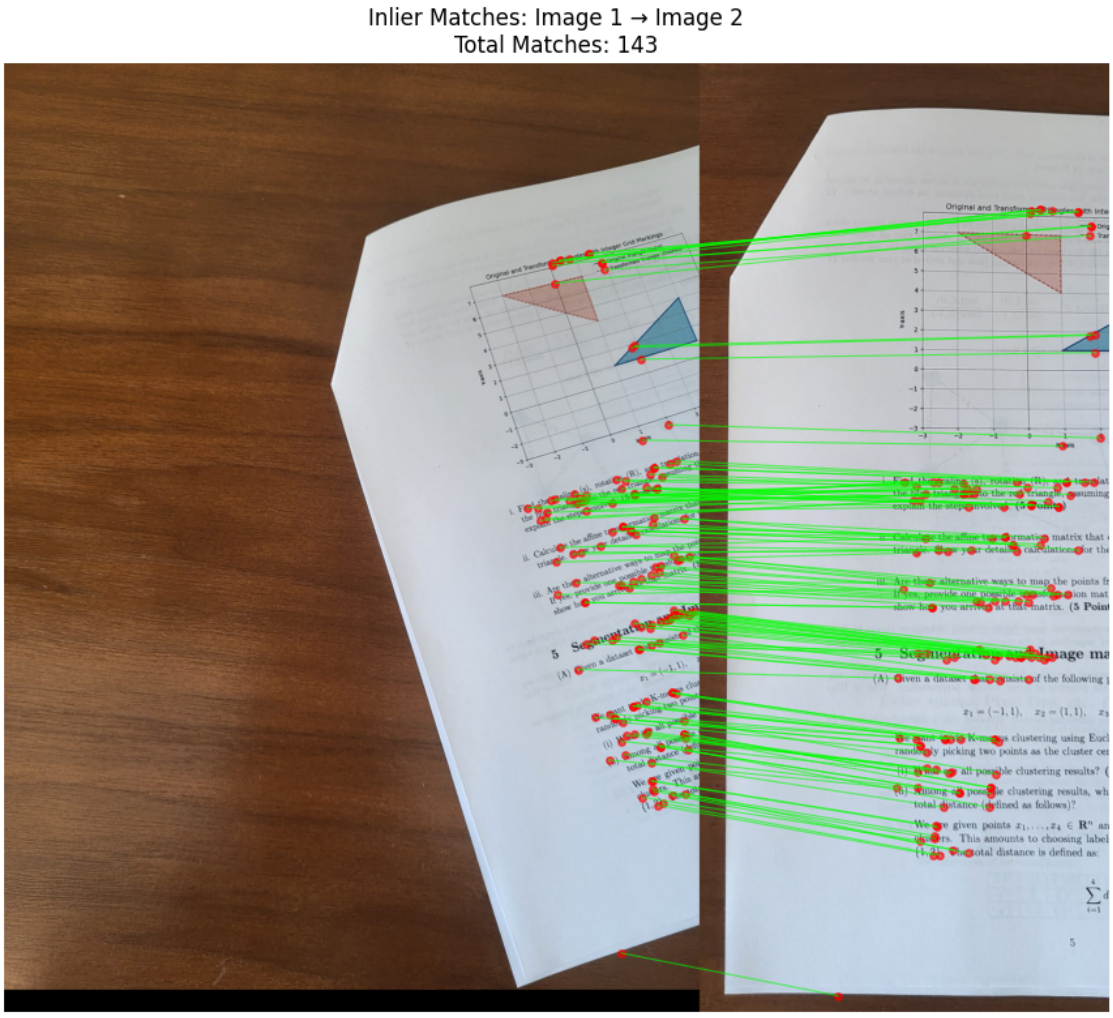

Figure 1.3(b)

(f) The NGCS is reliable and interpretable because of the backing of solid mathematical principles. The eigenvector-based partitioning can effectively separate the STOP sign from the background even in cluttered environments.

Weaknesses :

(a) The Normalized graph cut segmentation (NGCS) is computationally expensive due to the need to compute eigenvalues and eigenvectors of large matrices, especially for high-resolution images (if super-pixel concept is not used)

(b) It requires careful tuning of parameters like the number of super-pixels, color sensitivity and compactness.

(c) The method primarily focuses on separating the image into two segments (foreground vs background)

(d) Eigenvalue computation can be slow for large graphs, especially with high-resolution images.

(e) It lacks any explicit shape constraints relying entirely on color and superpixel similarity

(f) If the STOP sign's color closely matches the background or other objects, segmentation quality may degrade.

4 Creative Section

(a) Segmentation Algorithm:

```
print ("Homography matrix")
print (H12)

Homography matrix
[[ 1.07205814e+00 -3.64868356e-01 -4.25513714e+02]
 [ 2.95906808e-01  1.13299682e+00 -5.35200083e+02]
 [-5.37155557e-05  4.28259167e-05  1.00000000e+00]]
```

Figure 1.3(c)

Felzenszwalb's graph-based segmentation algorithm is a computationally efficient method for segmenting images into regions based on color and texture similarity. It is particularly useful for applications requiring fast and over-segmented outputs, such as object detection pipelines.

The algorithm starts with the construction of the graph. Each pixel is a node and the edges connect neighboring pixels based on the connectivity defined (4-connectivity or 8-connectivity). The weights are calculated based on differences in pixel intensities.

Every pixel starts as its own segment and iteratively the segments are merged if the difference between them is smaller than a threshold based on internal differences and external differences. We continue merging the segments until no more merges are possible under the threshold condition.

The final segments are regions of the image that are internally homogeneous and distinct from their neighbors. For our implementation, we use the skimage.segmentation.felzenszwalb library to segment the image and test them. As part of this algorithm we have 3 parameters: scale, sigma and minimum size. The scale parameter balances the importance of color differences (higher scale = larger segments). The sigma parameter represents the amount of Gaussian smoothing applied before segmentation to reduce noise. The minimum size corresponds to the minimum size of the segments. The smaller values produce more segments.

Compared to Mean shift and Normalized Graph cut algorithms, Felzenszwalb's algorithm has unique strengths. Firstly the algorithm is computationally efficient due to its greedy, bottom-up merging strategy. It handles high-resolution images very well because of this. Secondly, the algorithm is intuitive and has flexible parameters which make it easy to control segment size and granularity. And thirdly, the Felzenszwalb algorithm naturally produces over-segmented results, making it ideal for downstream processing like object detection or region merging. Additionally, the edge preservation in this algorithm is good, which is critical for maintaining object boundaries.

We can observe the results of this algorithm as part of Figure 4.1(a). This algorithm when compared with the ground-truth was able to get decent IOU values in comparison to other models.

We can notice the segmentation results with the ground truth in Figure 4.1(b), Figure 4.1(c), Figure 4.1(d) and Figure 4.1(e).

As part of the results i.e., IOUs with the ground truth. We can see the results in Figure 4.1(f)

We can observe that the algorithm has done a very good job in identifying the books cluster and below average result in identifying the cup segment. One of the reasons for poor segmenting results especially for the cup would be because of over-segmenting and creating multiple labels in that cup. With respect to the cat, the black color of the cat overshadows it's complement color white thus making it very difficult to merge these segments.

When we checked the results of this algorithm on another image, it was able to clearly segment the results, but was not able to merge the results as mentioned in the ground truth. We can observe the segmented results as part of Figure 4.1(g)

The IOU results of the algorithm on this image can be observed as part of Figure 4.1(h). We can conclude that the algorithm had performed decently well to identify all the segments with at least 30% considering the multiple segments were labeled to a single class ID which is difficult to achieve

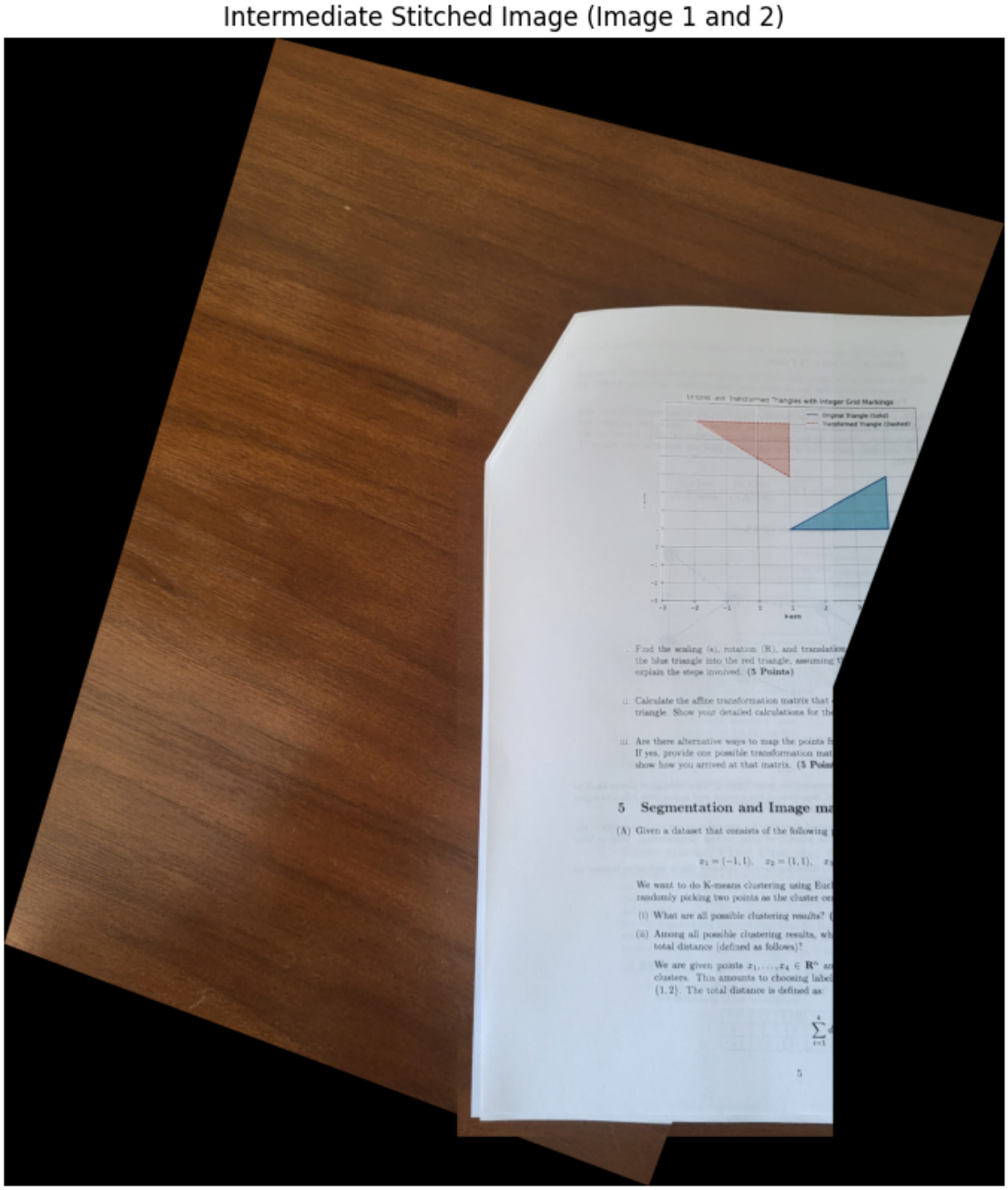

Figure 1.3(d)

in a classical segmentation technique with good accuracy. More analysis like the segment images are present in the colab notebook.

(b) Comparing results with other classical segmentation techniques and present empirical proof of method's performance.

The algorithm performed well when compared to standard graph algorithms Mean Shift and Normalized Graph cut algorithm. We were able to achieve higher IOU in our algorithm in comparison to these algorithm by a big margin. Moreover the above algorithm also mostly divided the image into two segments (foreground and background) thus making it difficult to capture nuances like the cat segment, bookshelf segment and the cup segment individually. The results for these algorithms are present as part of Figure 4.2(a) and Figure 4.2(b).

Figure 4.2(a) consists the results from Mean shift algorithm and Figure 4.2(b) consists the results from Normalized graph cut algorithm.

As we can see from the above results, that the algorithm proposed in this section was able to beat conventional graph-based algorithms.

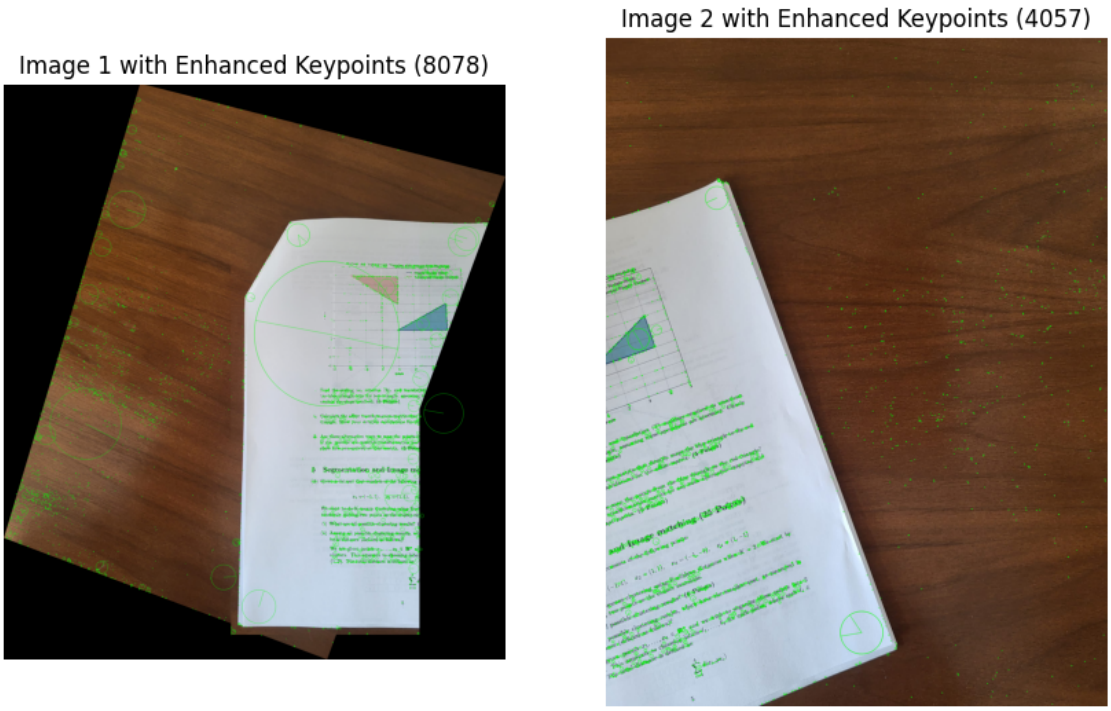

Figure 1.3(e)

```
[134] print ("Homography matrix between intermediate and final image")
      print (H_intermediate)

      Homography matrix between intermediate and final image
      [[ 9.33561725e-01  2.55782914e-01 -2.06423898e+03]
       [-2.39537796e-01  9.12003944e-01  5.30400386e+02]
       [ 7.79793263e-05  7.60998765e-06  1.00000000e+00]]
```

Figure 1.3(f)

NOTE: For the calculation of the IOUs we retrieve the segments from the algorithm. We also retrieve the segments from the ground truth image (annotation file) and then perform masking on the each segments. We find the best match for the respective segment in the algorithm labels and then assign that IOU value for the segment. Because we are directly taking the values from the algorithm and not performing additional steps like merging some similar labels, we end up with a slightly lesser IOU.

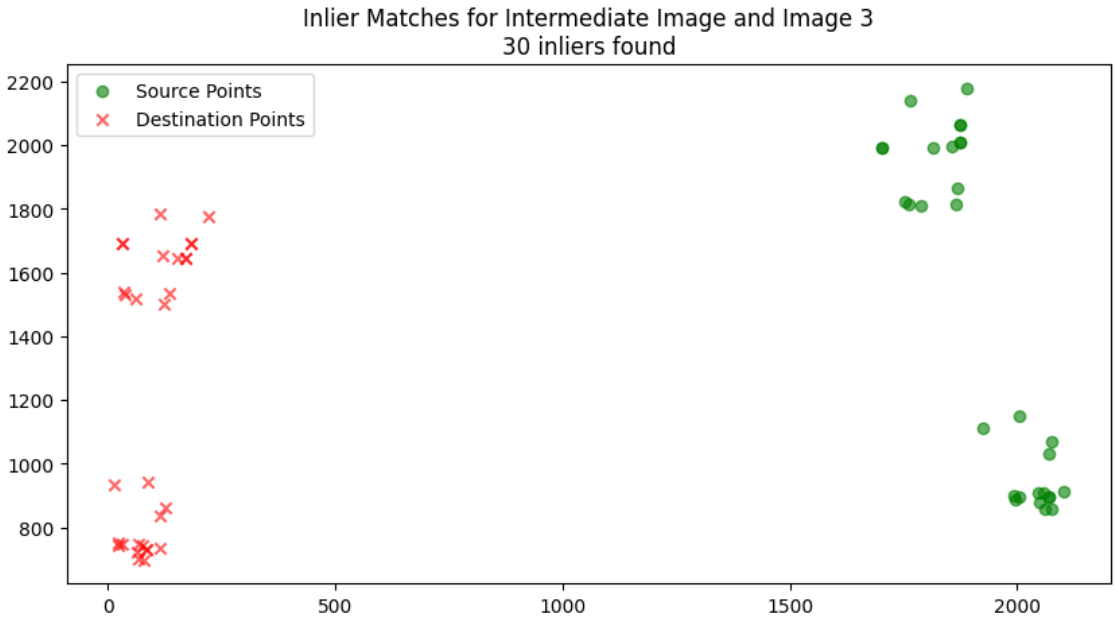

Figure 1.3(g)

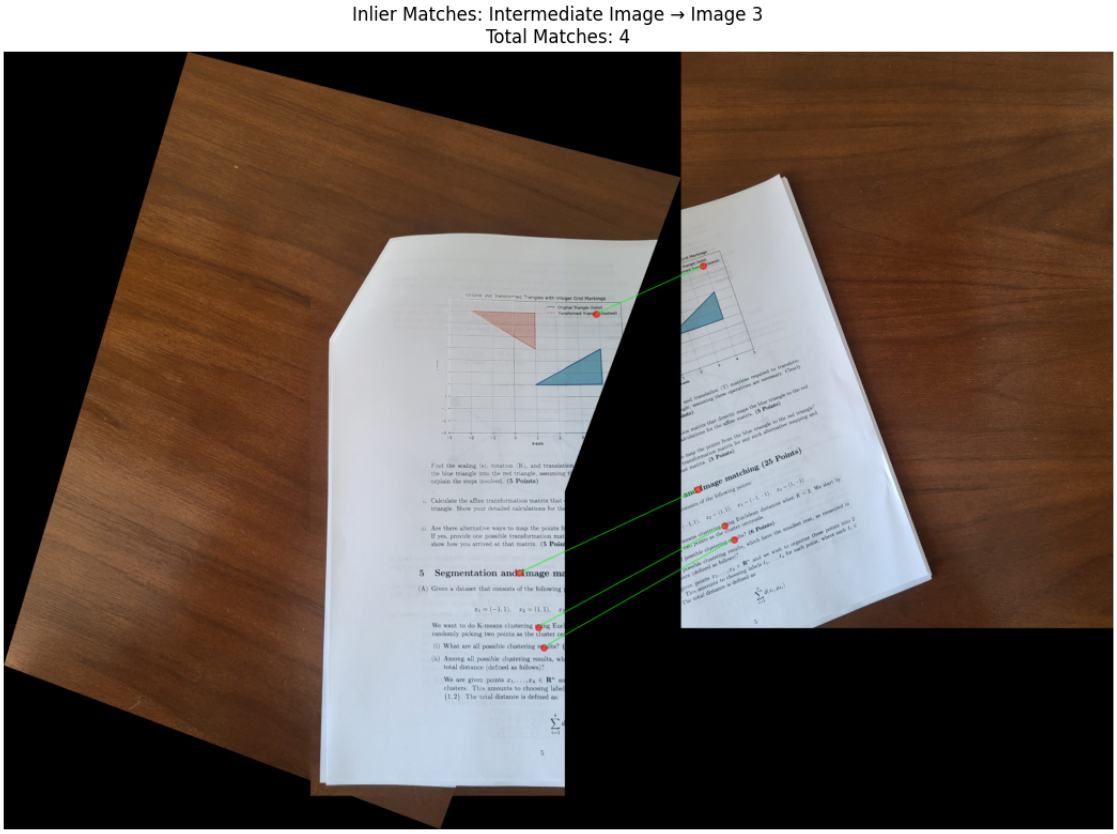

Figure 1.3(h)

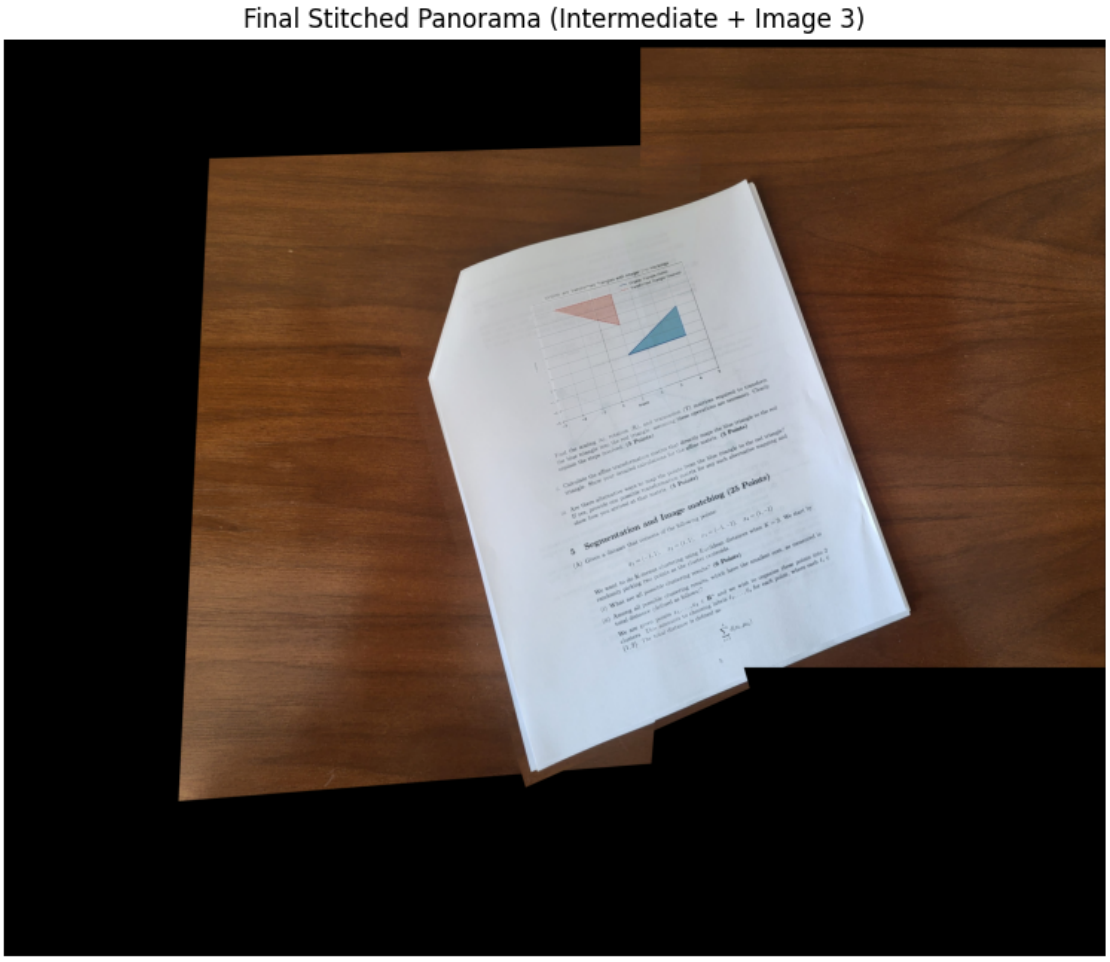

Figure 1.3(i)

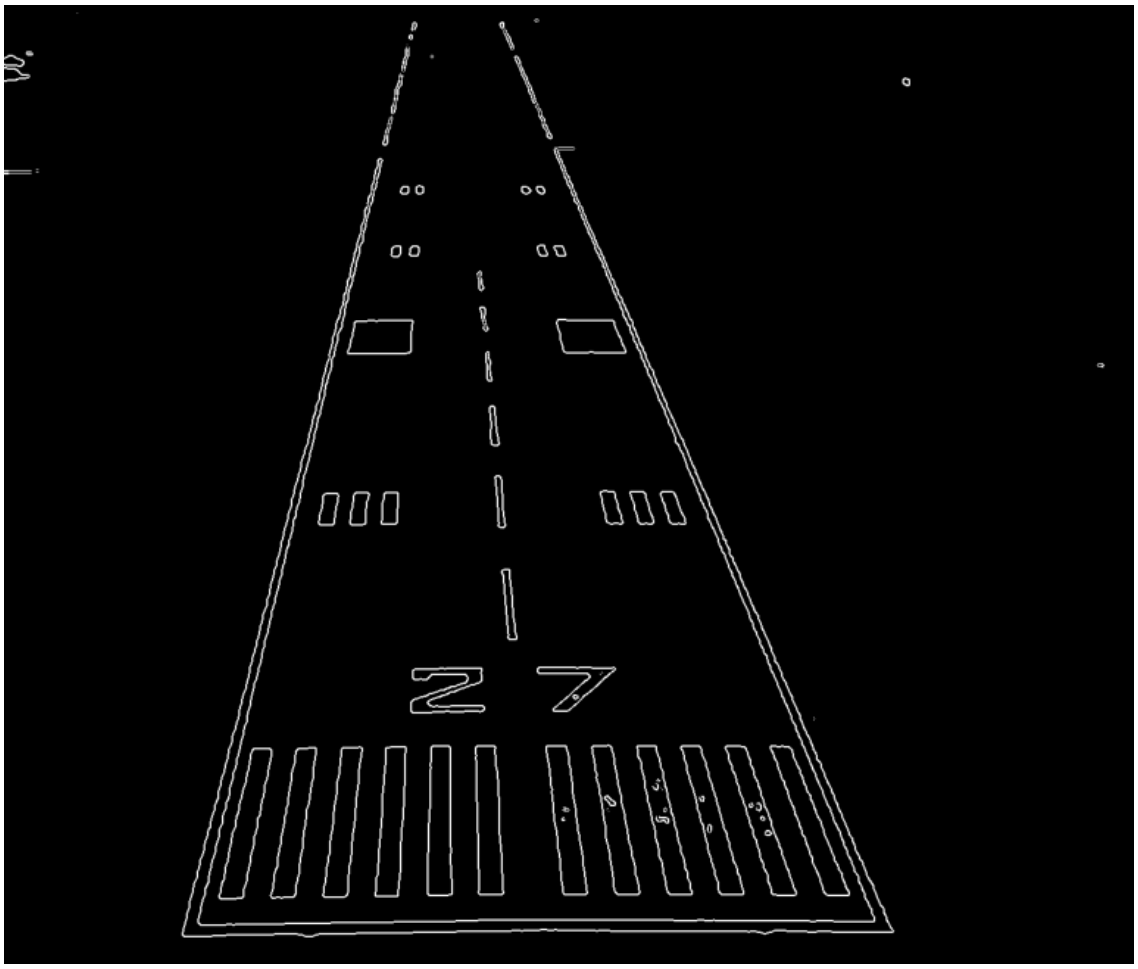

Figure 2.1(a)

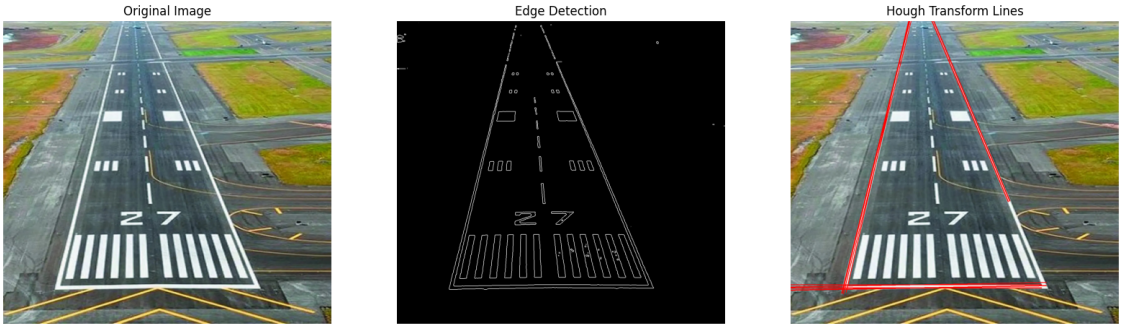

Figure 2.1(b)

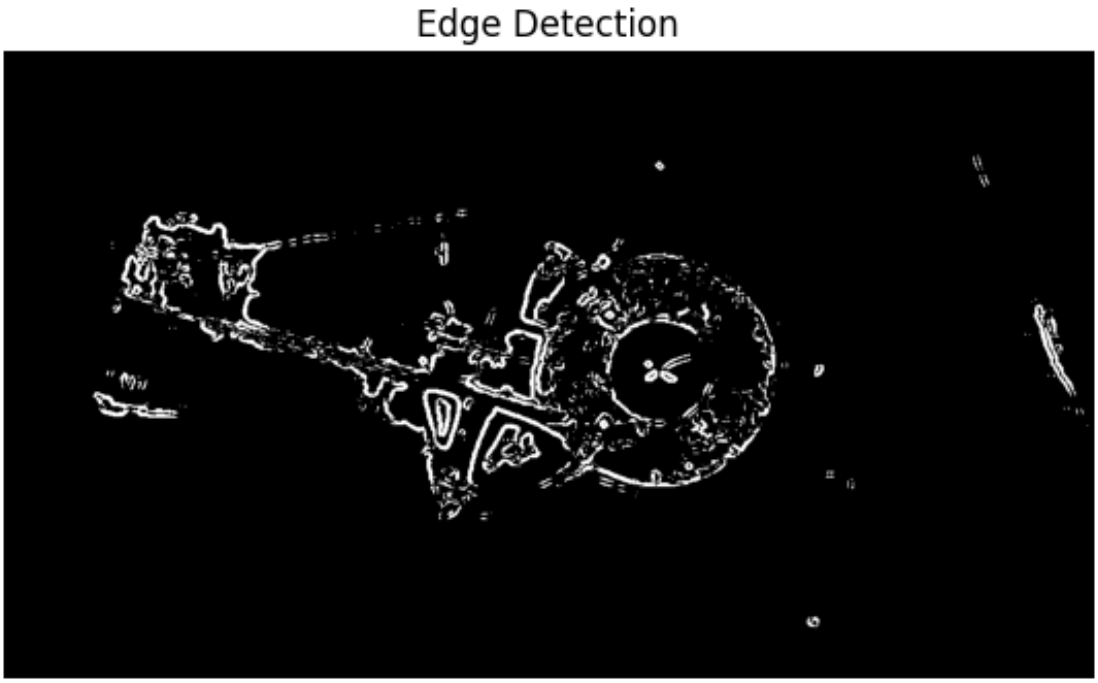

Figure 2.2(a)

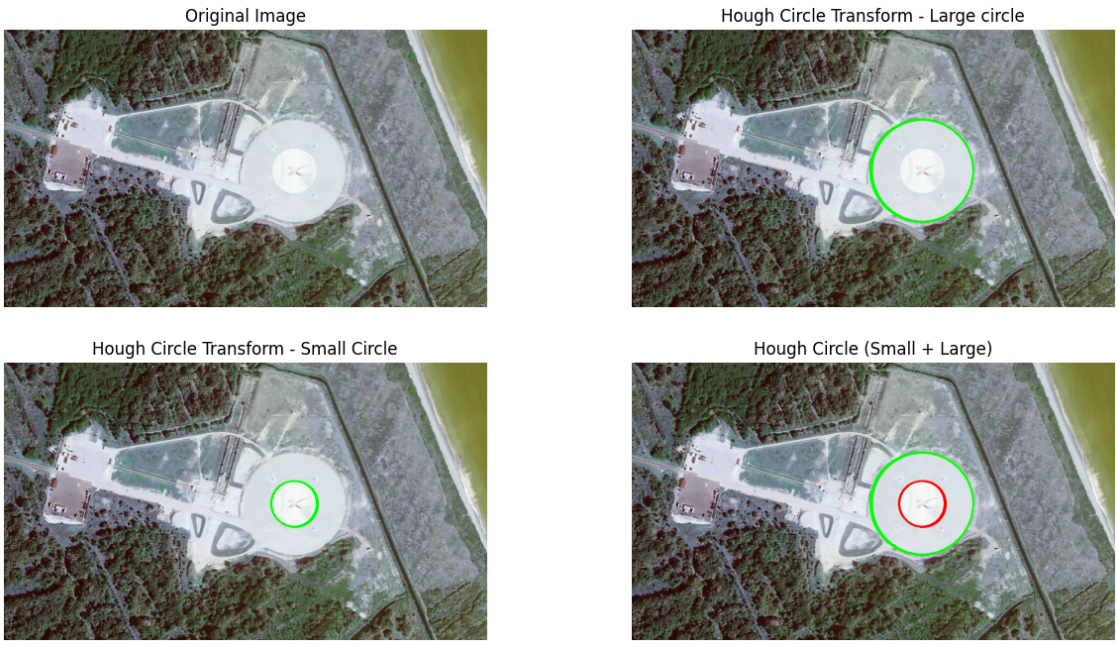

Figure 2.2(b))

Figure 3.1(a))

Figure 3.2(a))

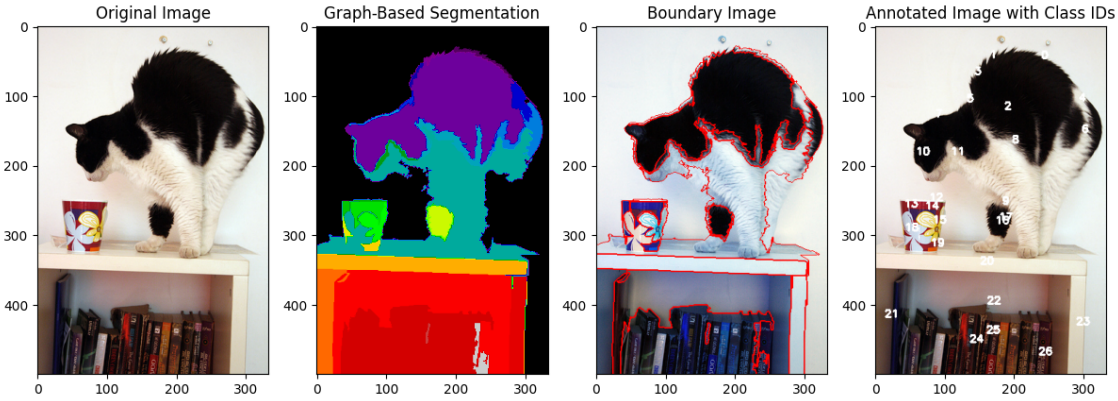

Figure 4.1(a))

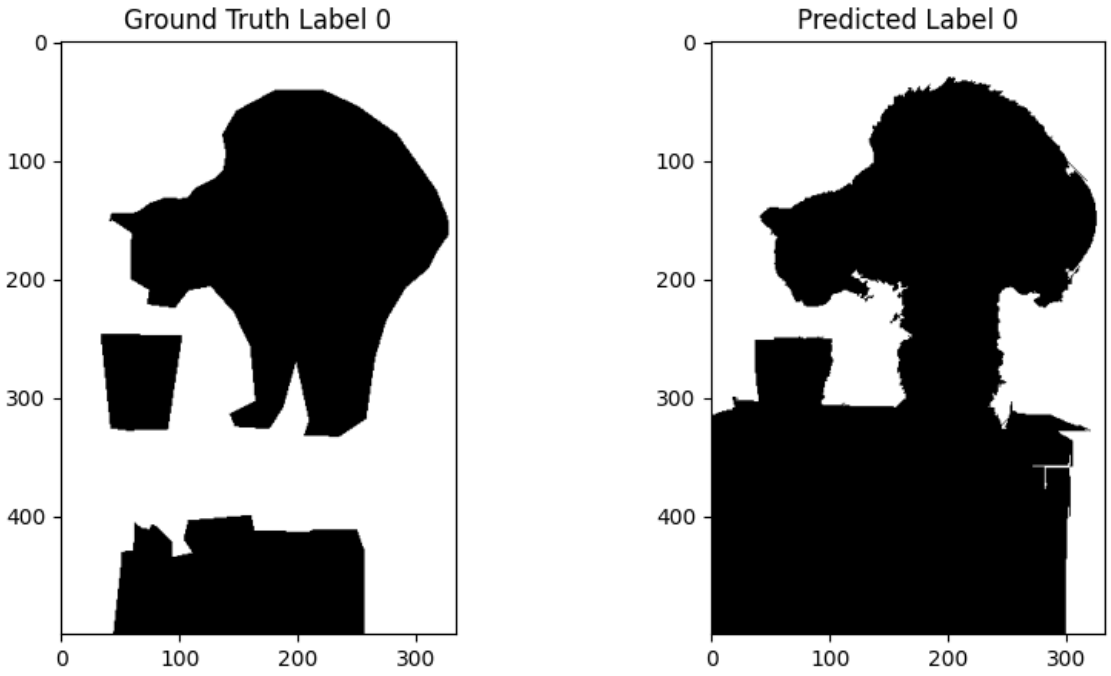

Figure 4.1(b))

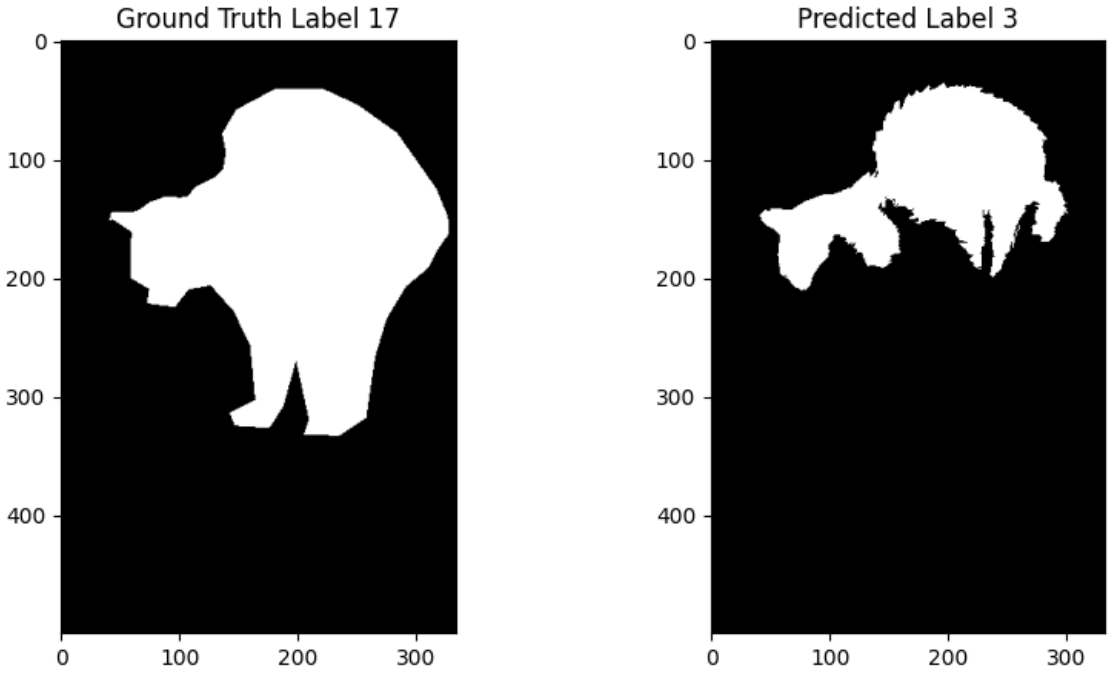

Figure 4.1(c))

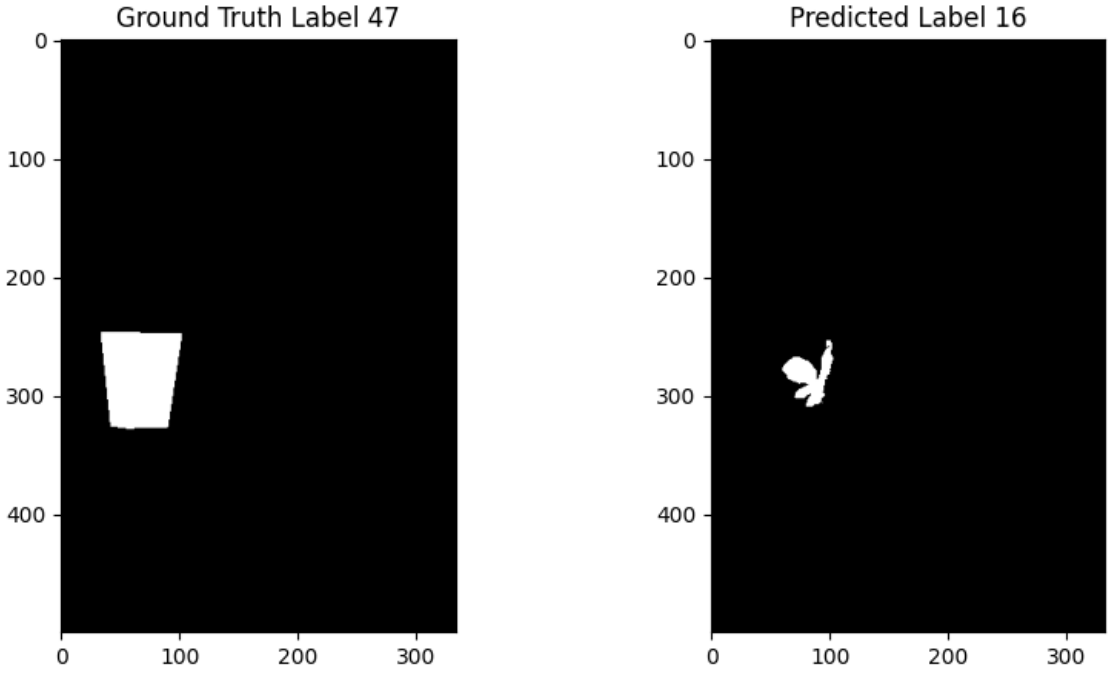

Figure 4.1(d))

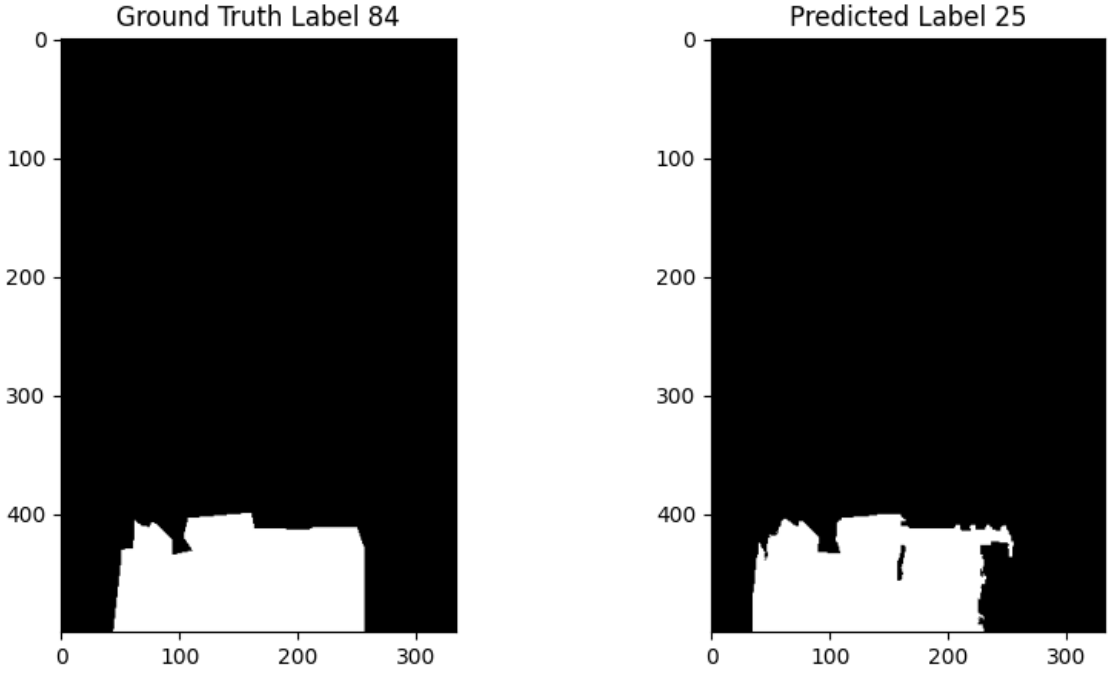

Figure 4.1(e))

```
IOU based on ground truth image
Label 0: IoU = 0.5757
Label 17: IoU = 0.4575
Label 47: IoU = 0.2083
Label 84: IoU = 0.8197
```

Figure 4.1(f))

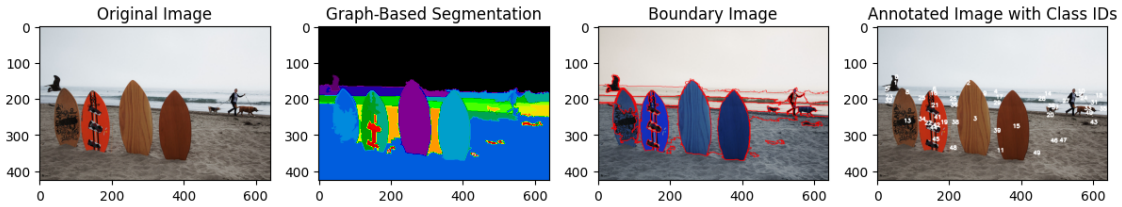

Figure 4.1(g))

```
IOU based on ground truth image
Label 0: IoU = 0.5027
Label 1: IoU = 0.3243
Label 16: IoU = 0.7271
Label 18: IoU = 0.4943
Label 42: IoU = 0.2987
```

Figure 4.1(h))

```
IOU based on ground truth image
Label 0: IoU = 0.6726
Label 17: IoU = 0.3252
Label 47: IoU = 0.0292
Label 84: IoU = 0.2756
```

Figure 4.2(a))

```
IOU based on ground truth image
Label 0: IoU = 0.6212
Label 17: IoU = 0.3074
Label 47: IoU = 0.0380
Label 84: IoU = 0.2640
```

Figure 4.2(b))