EE046746: Computer Vision
HW-3
Segmentation and Homography

Daniela Boguslavsky 315720003
Shahar Rashty 312465305

Due date: 6.6.23

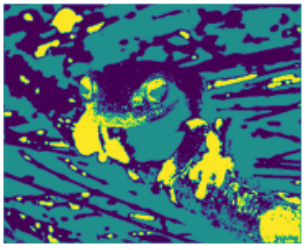
Part 1 - Classic Vs. Deep Learning-based Semantic Segmentation

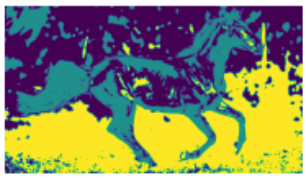
1. We wrote 3 functions , load_images_Image – takes folder path and loads as PIL Image object all jpg or png images inside this folder ,returns a list with all images.
   load_images – same but loads as cv2 and convert to RGB.
   And display_images that display all images inside a list.

   We used these to display horses and frogs images:




2. We picked KMeans as our classical method , this algorithm operates by iteratively assigning data points to the nearest cluster centroid and updating the centroids to the mean of the assigned data points.
   We wrote function segment_image_kmeans(image, num_clusters) that takes an image and desired number of clusters(hyperparameter),it first reshapes the image to 2D array each row represents a pixel and the three columns represent the RGB color values of the pixel, we performed KMeans using OpenCV to get groups and then reshaped back to the image shape .
   We used this function on the 4 images and with num_clusters=3 and got :






   For the deep learning-based method we chose to use the pretrained model DeepLabv3 that available at Pytorch .

   DeepLabv3 is a state-of-the-art deep learning model for semantic image segmentation,which utilizes a convolutional neural network (CNN) architecture, in our case ResNet, as its backbone network and employs atrous convolutions to capture multi-scale contextual information. Atrous convolution (upsampled filters), atrous spatial pyramid pooling (ASPP)

Atrous convolution - basically the dilated convolution from earlier, allows to control the resolution at which feature responses are computed within deep CNNs.
This effectively enlarges the field of view of filters to incorporate larger context without increasing the number of parameters or the amount of computation (it increases the resolution of the final output without increasing the number of weights.)

Code explanation, We loaded the pretrained model, set it to evaluation mode , Set the device , preprocess the images to fit the model ,turn to tensors and normalization , we forward each image through the model, then created a color pallette, selecting a color for each class.
We got :

Now we will discuss advantages and disadvantages of each method in general .

Advantages of DeepLabv3 for Segmentation:

1. Accurate and fine-grained segmentation: DeepLabv3 leverages dilated convolutions and atrous spatial pyramid pooling to capture both local and global contextual information, enabling precise segmentation of objects and boundaries.
2. Multi-scale feature representation: By incorporating atrous convolutions, DeepLabv3 can capture information at multiple scales without downsampling the feature maps, preserving fine details and contextual understanding.
3. Robust to object scale variations: DeepLabv3's multi-scale approach allows it to handle objects of different sizes effectively, making it suitable for segmenting images with diverse scales.
4. End-to-end trainable: DeepLabv3 can be trained end-to-end, enabling joint optimization of the backbone network and segmentation-specific layers, leading to better integration and overall performance.
5. Flexibility in backbone networks: DeepLabv3 can utilize various backbone networks, such as ResNet or Xception, allowing researchers and practitioners to choose the architecture that best fits their specific requirements.

Disadvantages of DeepLabv3 for Segmentation:

1. High computational and memory requirements: DeepLabv3, particularly when using larger backbone networks, can be computationally expensive and require substantial

memory resources, making it less suitable for deployment on resource-constrained devices.

2. Complexity and expertise required: Implementing and training DeepLabv3 requires knowledge and experience in deep learning frameworks, data preparation, and model optimization, which can be a barrier for beginners or those without prior experience in deep learning.

Advantages of K-means for Segmentation:

1. Simplicity and computational efficiency: K-means algorithm is relatively simple to implement and computationally efficient, making it suitable for quick segmentation tasks.
2. Intuitive and interpretable results: K-means produces hard cluster assignments, which can provide clear boundaries between segments, making the results easy to understand and interpret.
3. Robust to noise: K-means can handle noisy data reasonably well and can mitigate the effect of outliers by assigning them to the nearest centroid.

Disadvantages of K-means for Segmentation:

1. Requires the number of clusters (K) to be specified in advance: Determining the optimal number of clusters can be challenging and subjective, and choosing an incorrect value can lead to suboptimal segmentation results.
2. Sensitive to initialization: K-means algorithm is sensitive to the initial placement of centroids, and different initializations can result in different clustering outcomes.
3. Assumes clusters to be spherical and with equal variance: K-means assumes that clusters have a spherical shape and equal variances, which may not hold for all types of data and can limit its ability to capture complex and irregular segment boundaries.
4. Limited ability to capture spatial context: K-means treats each pixel as an independent data point and does not consider the spatial relationships between pixels, which can result in segmentation results that ignore contextual information.

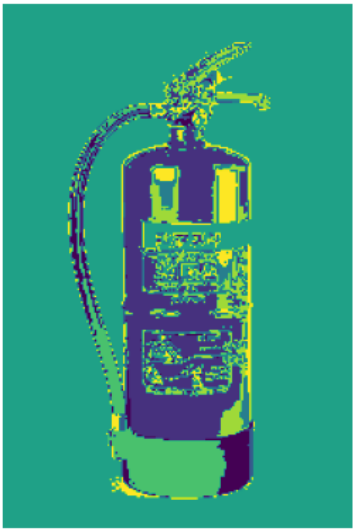
   For our images we can see that both methods got fairly good results, for the deep method we can see much better results for the horses images compared to the frogs images , it can be one possible explanation for that might be that while training the model wasn’t introduced to frogs as much as it been with horses .

3. We downloaded 3 images,loaded and display them using function we created at section 1:



4. Segmented using Kmeans:





We tried a few numbers of clusters values for each image

Person image k=3

Common used object (chair) k=2

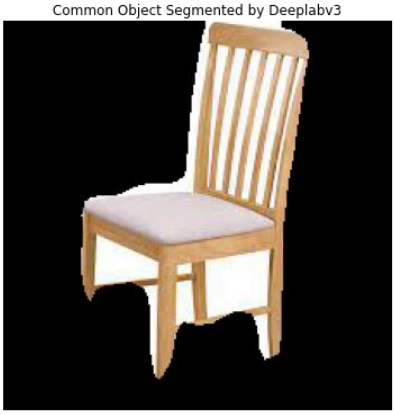
Uncommon used object (chair) k=8

Then we used the model and functions from last section to segment the images with the deep method,we musked the images to display the objects only using this set of labels :

['aeroplane', 'bicycle', 'bird', 'boat', 'bottle',
'bus', 'car', 'cat', 'chair', 'cow',
'diningtable', 'dog', 'horse', 'motorbike', 'person',
'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor']

Person Segmented by Deeplabv3

Common Object Segmented by Deeplabv3



Uncommon ObjectSegmented by Deeplabv3

5.

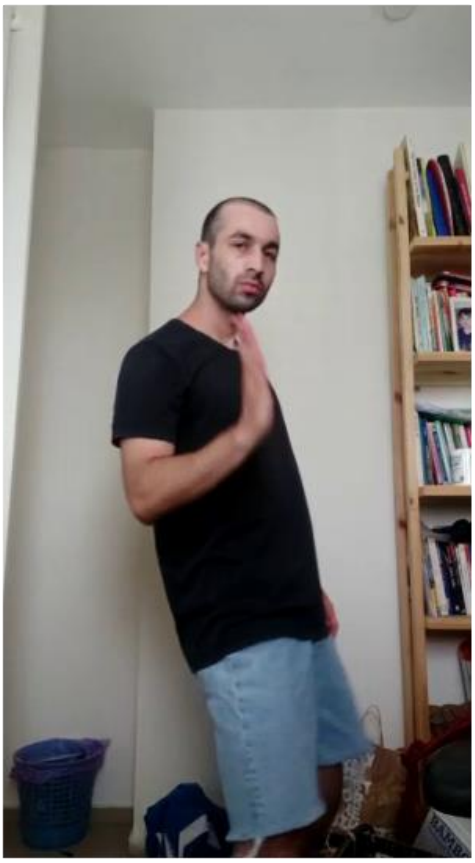
For the classical method we can try to do edges enhancement by using edge detection algorithm on the image , extract the edges and then sum the edges with the original image .

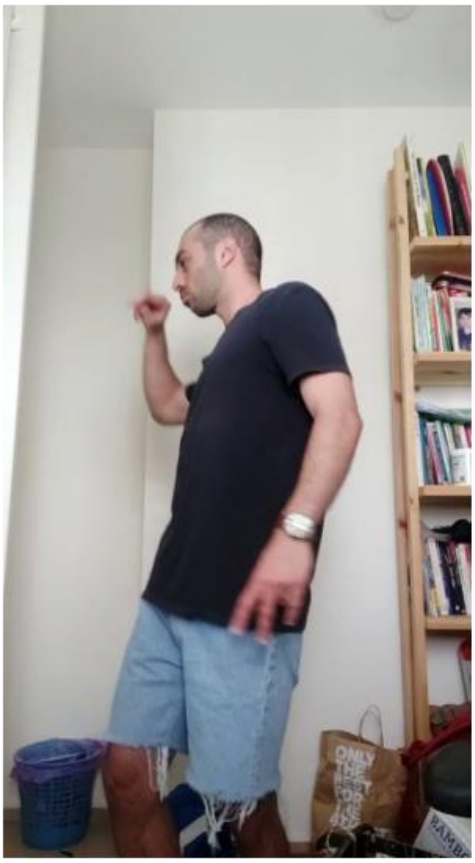
For the deep method we can try training the model better, for example training it with larger receptive fields: Increasing the receptive field of the network can help improve the model's ability to capture more contextual information.We can also try using different architectures as the CNN part of the model to try extract better edges as features .

Part 3 - Jurrasic Fishbach

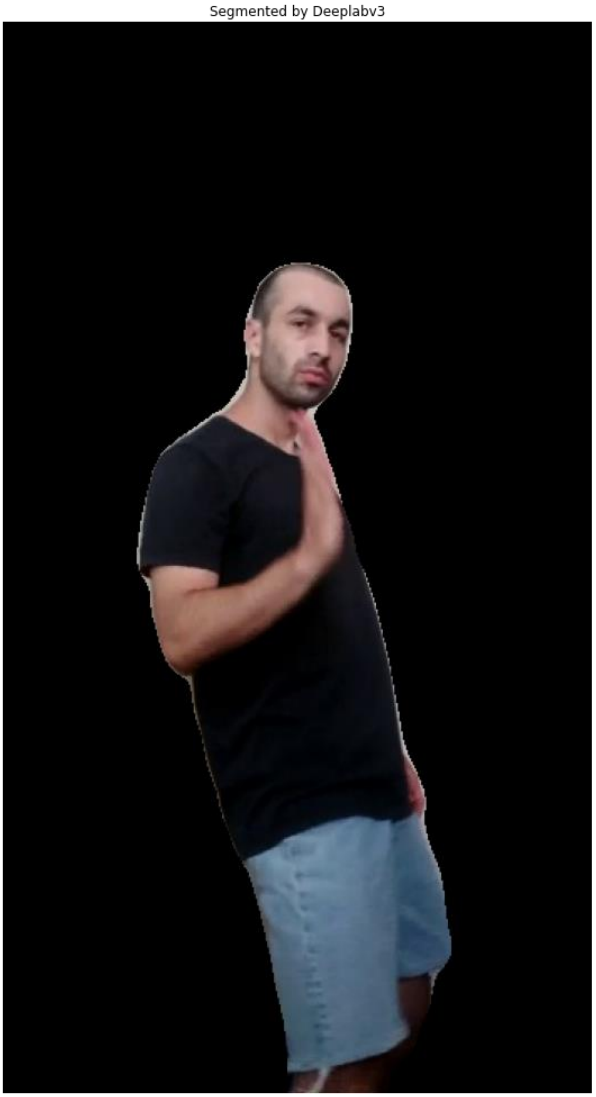
1. We film the video and convert it to frames using video_to_image_seq

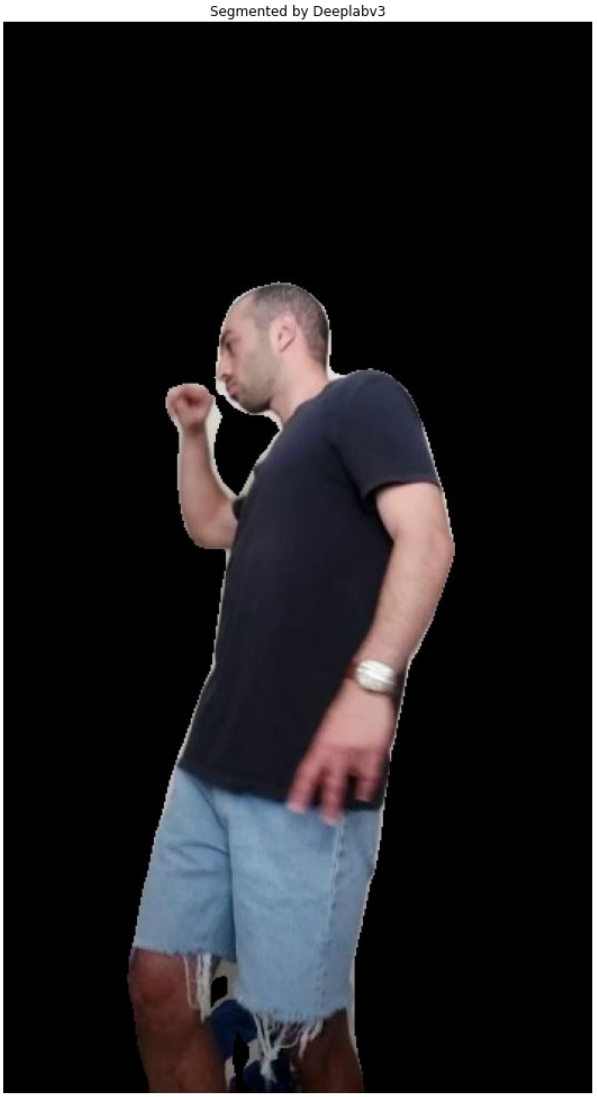
2.Loaded the frames and display 2 of them:





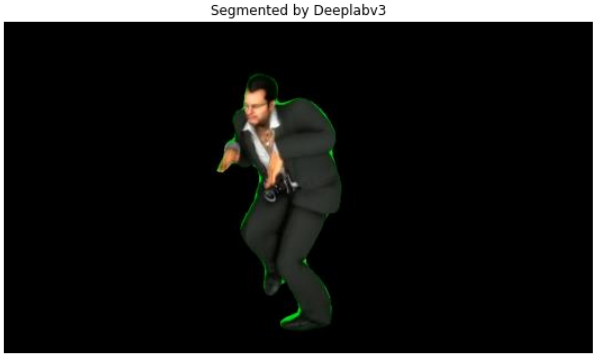
We preprocessed the frames and forward them to deeplab3 then masked them using the “person” label only .

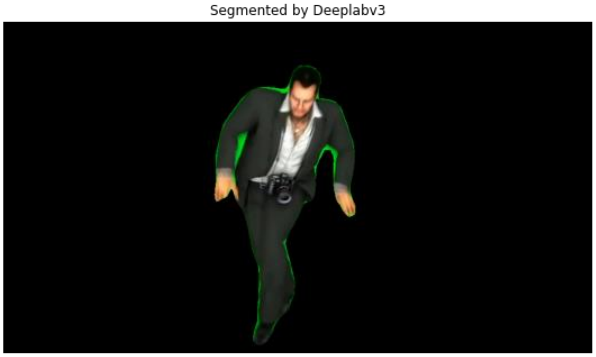




3.

In similar way we got frames of provided dancing men video segment and masked them:





We saved all masked frames images to new folders called “dancer” and “mine”

4.

We created array with background images , then we looped over a zip of 3 arrays,

my_masked_all, dancing_man_masked_all, background_images

my images masked , dancing men masked and all the background images .

At each iteration we created zeros template, resized the images to the background image shape , first we pasted the dancing_img to an appropriate pixels range on the template , then did the same for my image at different pixels ,

As final step the places where the templates still contains zeros were filled with the background values . we write each frame to a file using cv2.imwrite.

Then we used all files with image_seq_to_video to get the final video , here are 2 frames :

Part 3 - Planar Homographies:

SIFT_descriptor

Returns the SIFT descriptor and keypoints of an image,and draw its detected keypoints.

We used cv2.xfeatures2d.SIFT_create() to Instantiate the SIFT detector,Then

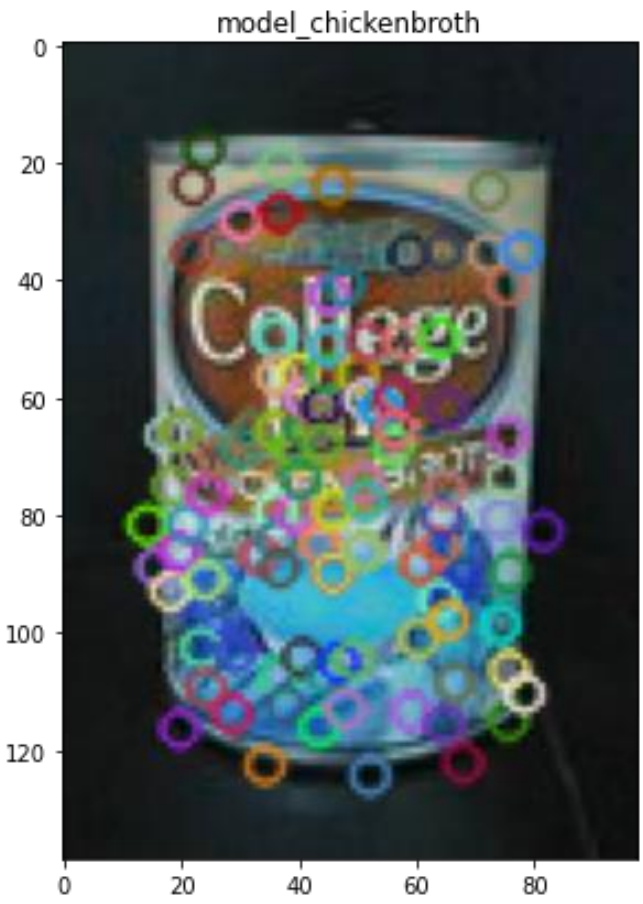
Detected and computed SIFT keypoints and descriptors with detectAndCompute

Method, finally used cv2.drawKeypoints to draw key points on the image.

We loaded model_chickenbroth

And chickenbroth_04

Found key points and paint them:


model_chickenbroth
0
20
40
60
80
100
120
0
20
40
60
80


model_chickenbroth_4

3.1

We wrote getPoints_SIFT,first computes SIFT descriptors and keypoints for both input images using the SIFT_descriptor function. It then matches keypoints between the two images using a brute-force matcher, we wrote helper function for that find_k_best_matches.It then extracts corresponding points from the matches using ".pt" method, with another help function extract_matched_points we did it for all matches. Finally, it returns the corresponding point matrices p1 and p2.

3.2

The computeH function takes as input two matrices p1 and p2 that contain corresponding (x,y) coordinates between two images. It constructs matrix A that encodes the linear system of equations derived from the homography equation p=Hq. It then solves for the homography matrix H using the singular value decomposition (SVD) of A. Specifically, it extracts the last row of the V matrix from the SVD and reshapes it into a 3X3 matrix H. Finally, it normalizes H by dividing by its bottom-right element to ensure that it satisfies the homogeneity constraint.

To test the computeH function, we calculate H of image with itself

```
# sanity check
p1,p2 = getPoints_SIFT(model_chickenbroth,model_chickenbroth)
computeH(p1,p2)
# We got the identity matrix,Nice!
```

	0	1	2
0	1.000000e+00	-5.480839e-16	-9.298440e-14
1	-3.656679e-16	1.000000e+00	2.438661e-14
2	-1.338149e-18	1.142324e-18	1.000000e+00

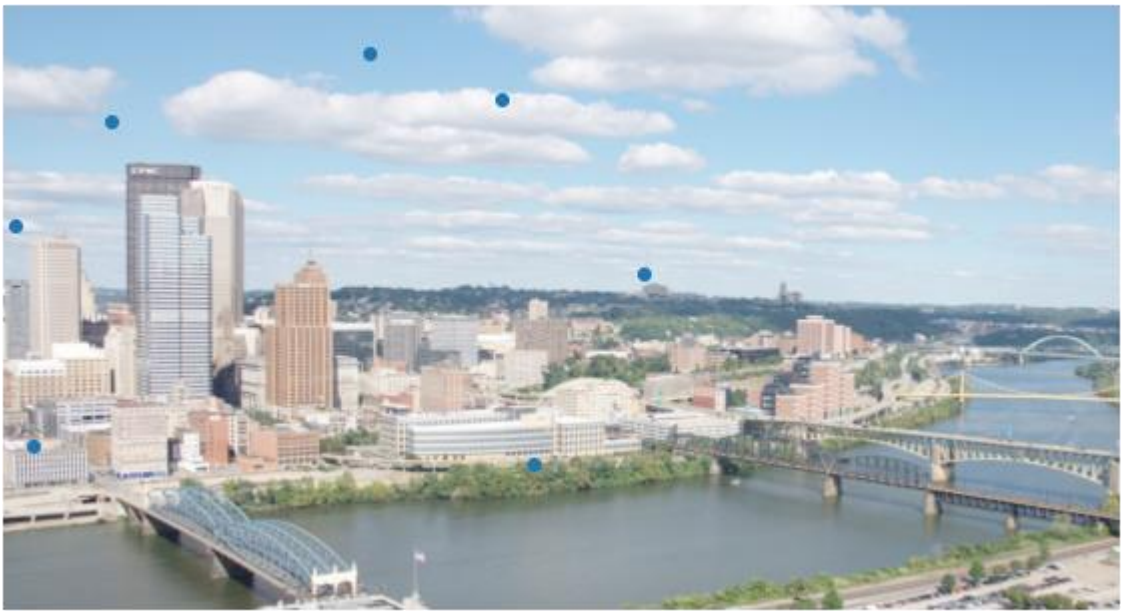
3 rows × 3 columns Open in new tab

And received the Identity matrix as expected.

Show that the transformation is correct by selecting arbitrary points in the first image and projecting them to the second image:

We calculated SIFT points for the two images, computed a transformation matrix between them, selects random points on one image, projects them onto the other image using the transformation matrix, and visualizes the results.

Points from image_R



Points from image_R on image_L

3.3

The function `warpH` performs a geometric transformation of an input image `im1` using a given homography matrix `H`. It warps the image to a specified output size `out_size` and returns the warped image.

create an zeros template `warp_im1` with dimensions specified by `out_size`. The image has three color channels (R, G, B) and is initially filled with.

Then computes the inverse of the input homography matrix `H` using `np.linalg.inv`. The inverse matrix `H_inverse` is required to map coordinates from the output image back to the input image.We used `scipy.interpolate.interp2d` and created Three interpolation (`R_interpolate`, `G_interpolate`, `B_interpolate`) one for each color channel then used to interpolate the pixel values at non-integer coordinates during the warping process. The `kind` parameter specifies the type of interpolation (defaulting to cubic interpolation).

And can be changed to linear.

Finally, Perform image warping, loop over each pixel in the output image. For each output pixel, performs the following steps:

- Calculates the homogeneous coordinates of the corresponding pixel in the input image by applying the inverse homography matrix to the output pixel coordinates.

- Normalizes the homogeneous coordinates by dividing by the last coordinate value.

- Checks if the normalized input coordinates are within the bounds of the input image.

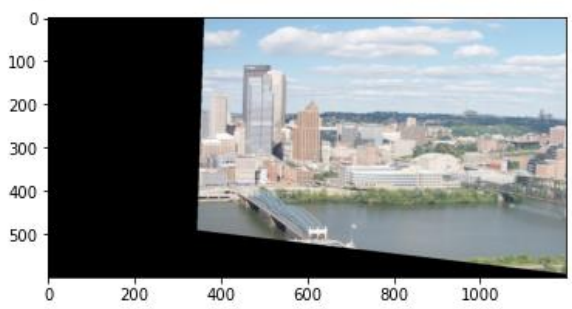
- If the input coordinates are valid, the interpolation functions are used to obtain the interpolated pixel values at the input coordinates. These pixel values are then assigned to the corresponding pixel in the warped image (`warp_im1`).

We did tried to perform more efficient implementation using meshgrid instead of nested for loops,It failed due to memery limitations.

We checked all function up to this point by warping incline_R image to incline_L:

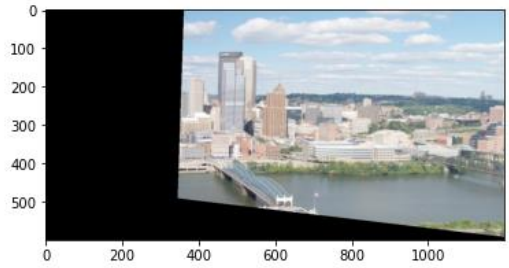
In [150]:
```
p1,p2 = getPoints_SIFT(incline_r,incline_l)
H_R_2_L = computeH(p1,p2)
warped = warpH(incline_r,H_R_2_L,(600,1200),'cubic')
```

In [151]:
```
warped = cv2.convertScaleAbs(warped)  # Convert to CV_8U depth
warped1 = cv2.cvtColor(warped, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(warped1)
plt.show()
```



And we also used the function with “linear” interpolation :

In [152]:
```
warped_linear = warpH(incline_r,H_R_2_L,(600,1200),'linear')
warped_linear = cv2.convertScaleAbs(warped_linear)  # Convert to CV_8U depth
warped_linear = cv2.cvtColor(warped_linear, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(warped_linear)
plt.show()
```



'linear' interpolation is computationally efficient and suitable for images with relatively smooth pixel variations, while 'cubic' interpolation is more accurate, captures fine details, and preserves sharp edges but is computationally more expensive.

In our case we don’t see to much of a difference between the two.

3.4

imageStitching(img1, warp_img2)

Very similar operation to what we did at section 2.4 when creating the frames for the video.

We implemented imageStitching function that takes in two images, image1 and warped_image2, and stitches them together to create a panorama image. first create a copy of warped_image2 and then finds the non-zero pixels in image1. Then replace the corresponding pixels in panorama_image with the corresponding pixels in image1. Finally, it returns the stitched panorama image.

We used all functions and set the output shape to be big enough by summing the width of the 2 images and taking the max value of the heights of the 2 images and adding to it 300 to make sure output size is large enough , here is the result panoramic image for the incline images:

3.5

After setting the width and the height of the output images manually in the last section for this section we implemented 2 help function to get the warped images .

get_warped_info(image, homography_matrix)

This function takes in an image and a homography matrix and returns information about the warped image. It first defines four original points in the image and calculates their corresponding warped points using the homography matrix. It then normalizes the warped points and extracts their x and y coordinates. It calculates the bounding box of the warped image using the minimum and maximum x and y coordinates. It also calculates the size of the warped image. Finally, it creates a translation matrix to adjust the homography matrix and returns the warped size, adjusted homography matrix, and bounding box.

resize_for_image_stitching

This function takes in two images, `image2` and `warped_image1`, and a bounding box. It resizes the images to prepare them for image stitching. It first extracts the top and left coordinates of the bounding box. It then calculates the maximum y and x coordinates of the resized images based on the bounding box and the original sizes of the images. It creates two new arrays of zeros with the maximum y and x coordinates and 3 channels. It then copies the original images into the new arrays at the appropriate locations based on the bounding box. Finally, it returns the resized `warped_image1` and `image2`.

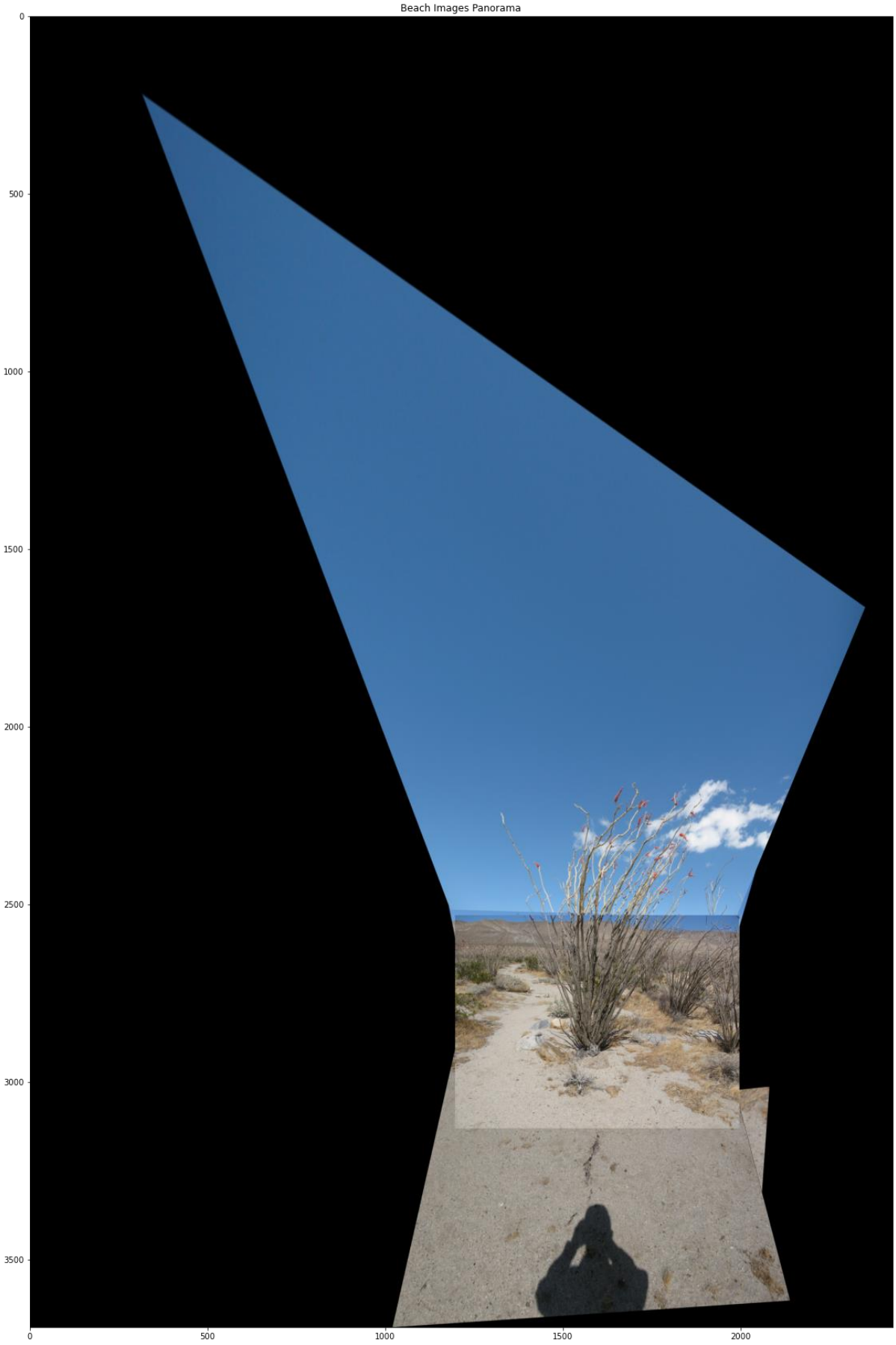
Then we wrote the final function that will combine all functions

panorama(im1, im2,k):

This function takes in two images, `im1` and `im2`, and an integer `k`. It first uses the `getPoints_SIFT` function to extract SIFT features and match them between the two images- we added 'K' to take only the best K matches. It then uses the matched points to compute a homography matrix using the `computeH` function. It then uses the homography matrix to warp `im2` using the `cv2.warpPerspective` function. It resizes the warped image and `im1` using the `resize_for_image_stitching` function. Finally, it stitches the two images together using the `imageStitching` function and returns the resulting panorama image.

Next we loaded the beach and Sintra images , since they were pretty large we downgraded each picture by factor of 2 , then we used the panorama function to create panoramic image of 2 images at a time , we tried different set ups and found the best order for combining all images was stich images 4 and 3 , the result stitched to image 2 then to 1 and then to 0 .

We tried different values for k , since the beach images do not contains many "good" key points for the SIFT detection we used k=10 :


Beach Images Panorama
0
500
1000
1500
2000
2500
3000
3500
0
500
1000
1500
2000

Similar process was done for Sintra , here is the result with K=40 since this images contains more “interesting” points:

We checked the affect of K parameter which determined how many matches will be return from the getPoints function , has we said earlier for the beach images there are limited key points for the SIFT algorithm , as we expected for larger K=40 we got bad results :

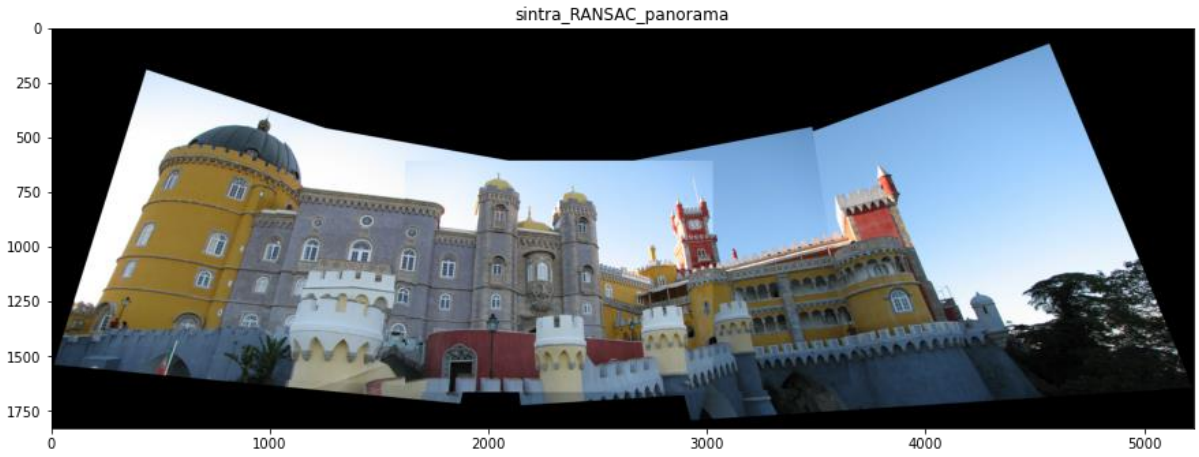
If we don't take only the top K matches for estimating the homography when using SIFT without RANSAC, we may end up with a poor quality homography matrix. This is because SIFT can produce a large number of feature matches, but not all of them are necessarily good matches. By taking only the top K matches, we are selecting the most reliable matches and discarding the unreliable ones. This helps to ensure that the homography matrix is based on the most accurate matches, which in turn leads to better image stitching results. If we don't limit the number of matches, we may end up with a homography matrix that is based on a large number of unreliable matches, which can result in poor quality image stitching.

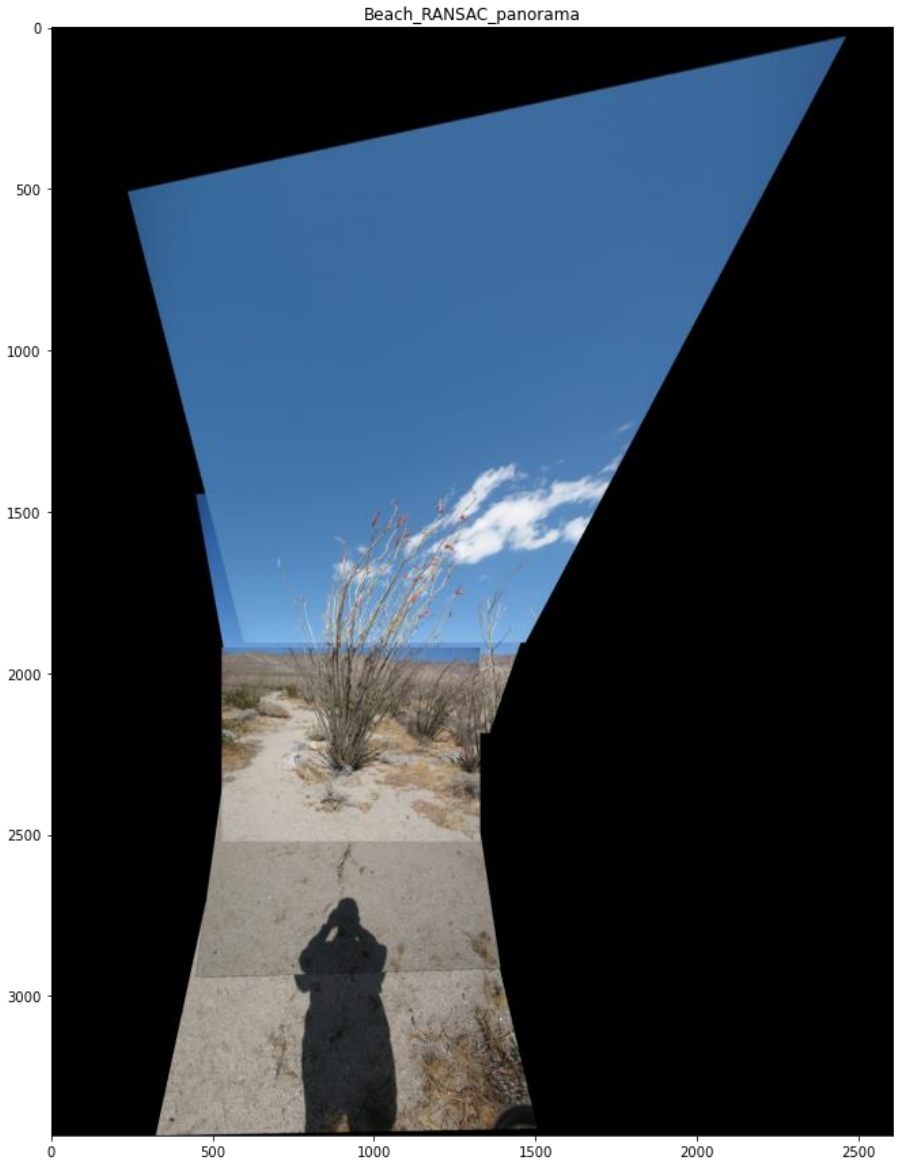
3.6

We set nIter=15000, tol=0.8

Created panorama_ransac function that uses the same steps as the panorama function but uses RANSAC instead of computH to get the homography matrix .

Results :





RANSAC (Random Sample Consensus) algorithm is a robust method for estimating parameters of a mathematical model from a set of observed data that contains outliers.

When creating a panorama image, RANSAC is needed to estimate the homography matrix that maps the pixels in one image to the corresponding pixels in the other image. This is because the feature matching process can produce a large number of incorrect matches, which can lead to a poor quality homography matrix. RANSAC helps to identify and discard these incorrect matches, resulting in a more accurate homography matrix.

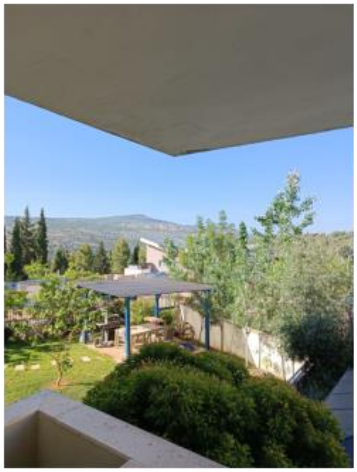
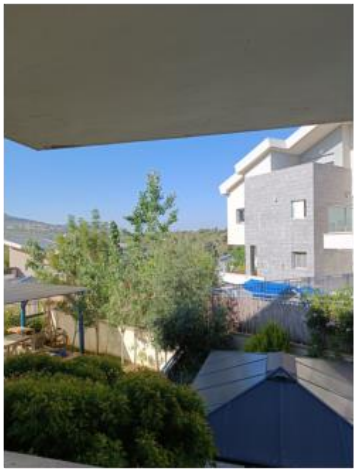
When comparing using RANSAC vs. not using it for creating the panorama images of the beach and SINTRA, we can expect that using RANSAC will result in better quality panorama images. This is because RANSAC helps to identify and discard incorrect matches, which can lead to a more accurate homography matrix and better image stitching results. Without RANSAC, we may end up with a homography matrix that is based on a large number of

incorrect matches, resulting in poor quality image stitching as happened when we didn’t use RANSAC or K to limit the matches.

To get better results, we could have used a more robust feature matching algorithm, such as SURF or ORB, which are less sensitive to changes in lighting and viewpoint. We could also have used a more sophisticated image blending technique, such as multi-band blending, which can produce smoother and more seamless transitions between the images. Additionally, we could have used a larger number of images to create the panorama, which can help to reduce distortion and improve the overall quality of the panorama.

3.7

We took 3 images :

And used the panorama_ransac to create a beautiful panoramic image :