Computer Vision Homework 3

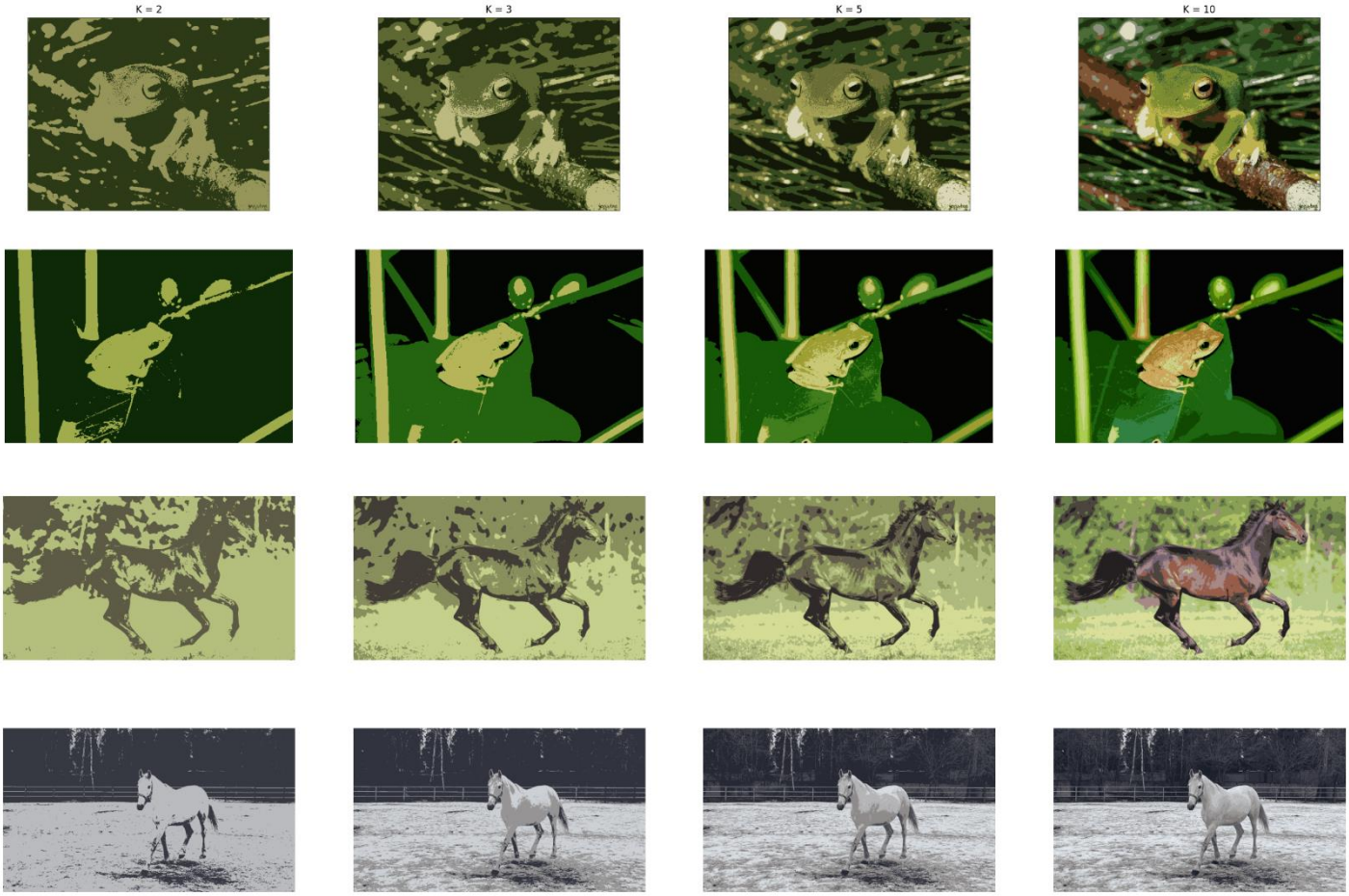
Oron Barazani 305295818 & Roee Hadar 206390098

Part 1:

1. The provided images:

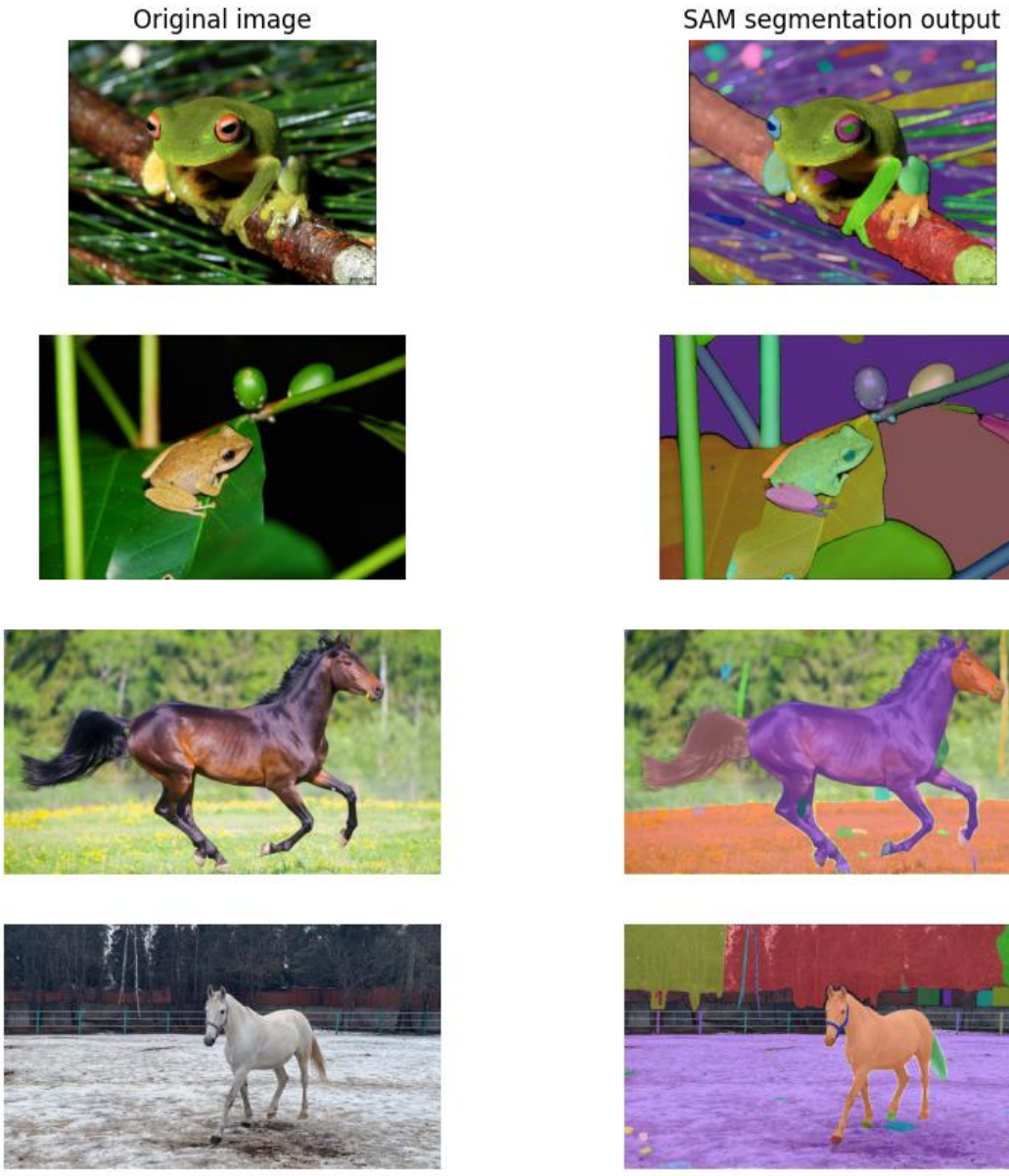



2. Classic segmentation using K-means algorithm:



K-mean algorithm offer color intensity-based segmentation. Its advantage is in segmenting object which differs from their background by color – for the second frog image only 3 clusters needed for segmentation! And for the other frog and first horse 10 clusters offered fair result. Another advantage is fast processing time compared to DL solutions. The downside of this method is demonstrated in second horse image which is white on snowy background – the horse is not properly segmented from the background.

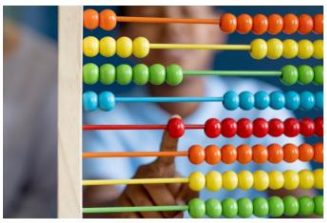
Using DL model – SAM:



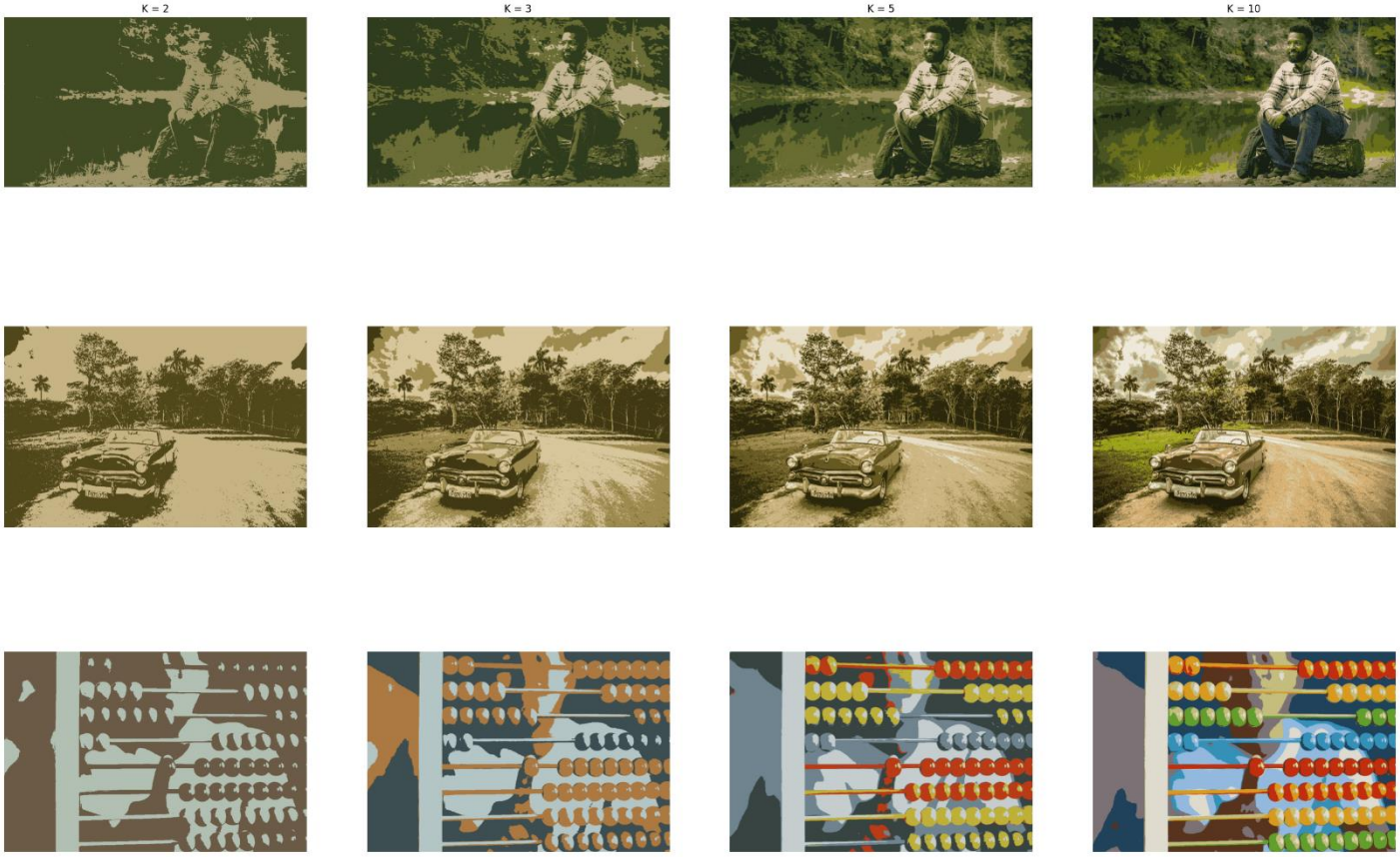
We chose SAM (Segment Anything Model) as our DL solution. This model is based on vision transformer (ViT) which capturing local as well as global features in the image resulting in better segmentation – we chose *sam_vit_h_4b8939.pth* . The model is well generalized for different objects as it was trained on huge and diverse dataset. On the other hand, the model is extremely large and slow in computation time compared to classic methods.
We can see that using SAM we got good segmentation result, and in some of the cases as in the first frog, the model segmented the image in excessive manner. Few blobs, that aren't really an object are detected, and the same can be observed in the first horse image.
The strength of this model is demonstrated in second horse image, where k-Means struggled to segment correctly object from background. The horse is segmented completely off the background

3. Display our images (a person, a car, an abacus):



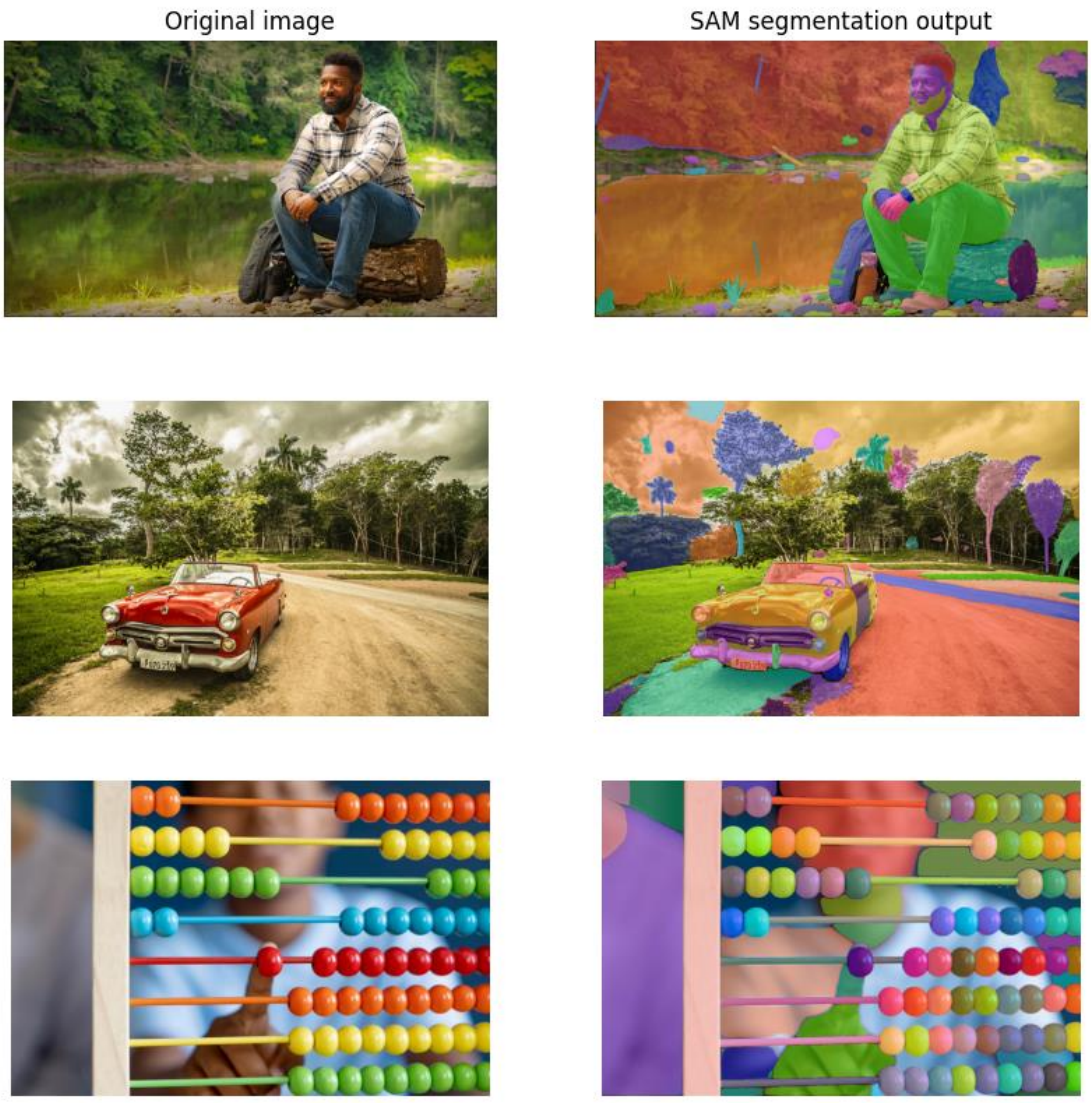


4. K-means segmentation:



For the first two images the algorithm failed to segment the object correctly. In the first image, the person's shirt is similar in its color to the background and it hard to tell where the edge of the shirt is. In the second image (car), we expected for a good segmentation – red car on green-brown background, however the algorithm failed to create the car-reds cluster. Maybe higher number of clusters would have helped. For the third image (abacus) – we got excellent result for merely 5 clusters. The abacus beads have unique color that differ from the background, leading for good results – the problem is that each abacus row is segmented as a whole, and not separated beads.

SAM segmentation:

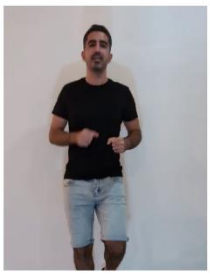
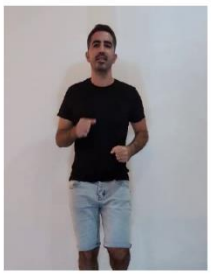


in SAM segmentation we got excellent overall results. In the person image we got different masks for the shirt and its background. The car image is also segmented correctly. In the last image the results are even better. The beads are segmented separately as opposed to k-means segmentation as mentioned above.

Part 2:

We filmed a short video of Oron running on a white background and chose the dinosaur video. We cut the videos frame to appropriate size which only include the object. Here are 2 frames from each video:



The algorithm for segmentation we chose is same for the two videos: k-means based segmentation. We chose this algorithm because it is far more computationally efficient than DL methods, and it is very accurate when segmenting object out of monotonic background.

For the dinosaur video it was straight-forward since it has green background, only 2 clusters needed to almost perfectly segment the frames. Then we chose the cluster which its centroid has green element higher than 250.

For Oron's video it was a bit trickier – first we tried with 2 clusters, but the results weren't that good, so we chose 3 clusters and the result improved. To chose the white background cluster, we combined 2 conditions:

- Whiteness measure (or no color measure) – std of the pixel lower std means pixel who is not colorful, rather black, gray or white.
- Darkness of pixel – mean of the pixel to separate whites from blacks (especially the black shirt from the background).

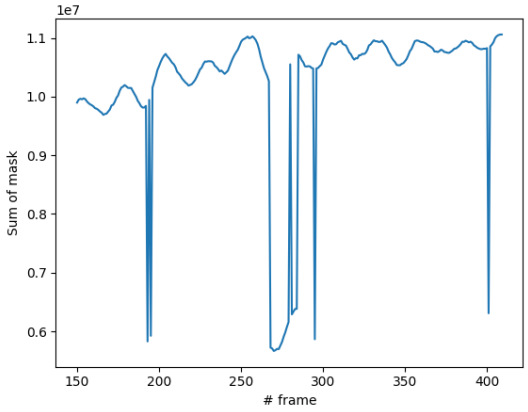
We chose empirically $\sigma < 2.2$ and $\mu > 80$ to be background pixel.

We noticed a problem:

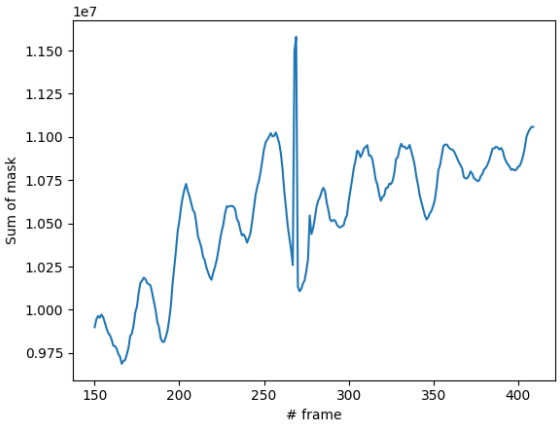
- small holes in the mask every few frames – to deal with it we used morphological close operation (dilation and then erosion) to eliminate those small artifacts. This indeed improved the masks quality




The masks were good at that point, but we noticed another problem which is flickering every few frames (video seen in $combined\_video\_flickering.mp4$)
We sum the mask in each frame to find out when and where this is happening:

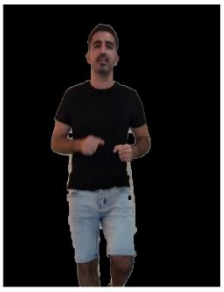


We can see few drops in the mask sum. We introduced improvement to the algorithm by applying another iteration of k-means, but with 5 clusters whenever the mask sum is lower than 0.75*median of the 50 last frames mask sum. This solution improved drastically the results and almost eliminated the flickering:



Next, we used the masks to segment out each object frame by frame:






Final step was to stich all object together. We chose the moon background. To properly place the objects in the frames we preformed:

- scaling – when loading the raw videos. It also saved us computational time.
- Expanding object's frame to match the moon frame shape – we added zeros (black pixels) in object surrounding so it will be located in the desired location and also match the desired frame shape.

Finally, we inverted the masks and zeroed out the object pixels in the moon frame and added the objects frames:

$$frame = bgd * (1 - obj1\_mask) * (1 - obj2\_mask) + obj1 + obj2$$

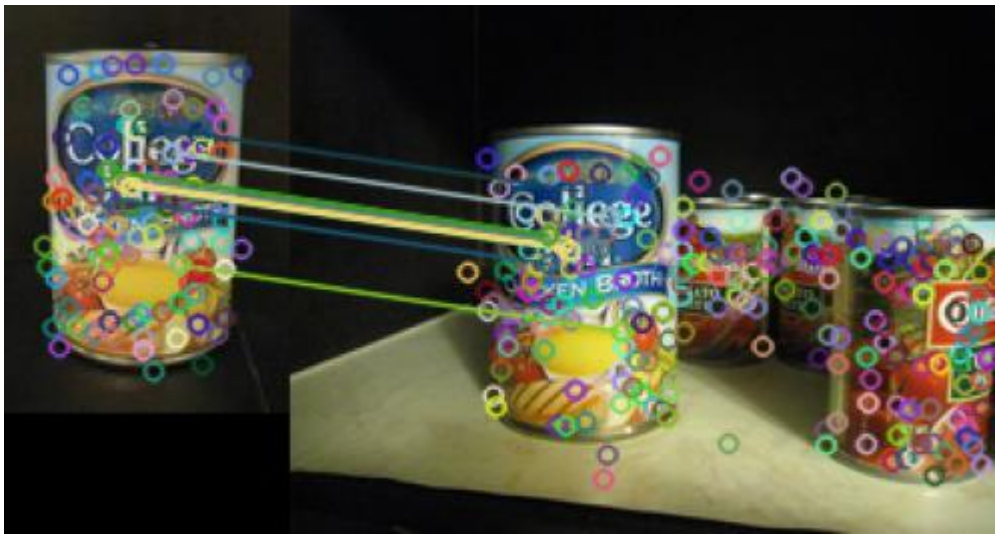

The final video is $combined\_video.mp4$

Part 3 – Planar Homographies

3.0.1 – As required from us by the question, the implementation of the SIFT descriptor on the chicken broth pictures is presented below:

Our implementation provided us with the points of interest which, as we can see above, have some outliers – most of the points are a good match! Hence, when going forward in this exercise, we will take this result into account.

3.1 – Using the 10 best points matching from both chicken broth images, we got the following result:



As discussed earlier (in the previous section), using only the 10 best points provides us with the ability of "throwing away" the outliers which provide harmful information for the calculation of the homography matrix, which will come in the next section.

3.2 – The computation function which we used in order to calculate the homography matrix is, as provided to us in the background information, $Ah = 0$, where:

$$A = \begin{pmatrix} -x & -y & -1 & 0 & 0 & 0 & u \cdot x & u \cdot y & u \\ 0 & 0 & 0 & -x & -y & -1 & v \cdot x & v \cdot y & v \end{pmatrix} \text{ and } h = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}_{CS}$$

where CS represents Column Stack, and the matrix A is of size 2Nx9 where N is the number of corresponding points which we used from the previous section (need minimum of 4, since the matrix H has 8 unknown elements, considering we can normalize the last element to be 1, as we have done in the code with normalization of the vector with respect to h_9).

Looking at three arbitrary points we get the following:

First Image with Selected Points

Second Image with Projected Points

As we can see, the homography matrix
indeed provided us with good transformation for the arbitrary points chosen – hence we can infer that the matrix we got is the correct one.

3.3 – The implementation of the function was created using the following:

- Get input image dimensions using the 'shape' attribute of Image 1.
- Creating a meshgrid of the output coordinates using np.mgrid. the grid spans from 0 to 'out_size[0]' in the y axis and from 0 to 'out_size[1]' in the x axis.
- Convert output coordinates to homogenous coordinates – we created a 3D array by stacking the 'x_out' and 'y_out' arrays along with an array of ones. This represents the output image coordinates in homogenous form (x, y, 1)
- Compute inverse transformation of the output coordinates using H – computation by multiplying the homogenous coordinates with the inverse of the homography matrix 'H'. The output coordinates are reshaped to a 2D array 'coords.in' by transposing the result and taking the transpose again.
- Converting homogenous coordinates to Cartesian coordinates
- Reshape the input coordinates to match the output grid.
- Initialize warped image to zeros and warp each color channel separately – the function performs channel-wise interpolation for each color channel of the input image. It uses linear/cubic interpolation method and allows for values to be extrapolated beyond the defined grid bounds and sets the fill value to 0 for points outside the grid.

- Assign interpolated values to the corresponding locations in the warped image's channel – the resulting interpolated values are assigned to the corresponding locations in the warped image array, specifically in the current channel.
- The wrapped image is returned.

As for the different interpolation methods – they can have an impact on the quality and characteristics of the image warping result.

Linear Interpolation:

- Estimates pixel values based on linear combination of the surrounding pixels. It assumes that pixel's intensity changes linearly.
- Advantages –
    1. Simplicity and efficiency – Linear Interpolation is computationally less expensive compared to more complex interpolation methods.
    2. Smoothness – It produces relatively smooth results and can work well for images with gradual changes in intensity or color.
- Disadvantages –
    1. Lack of accuracy for sharp transitions – Linear Interpolation may introduce blurring or artifacts around edges and sharp transitions, as it assumes a constant rate of change between neighboring pixels.
    2. Inability to capture fine details – it may not accurately represent intricate details or high-frequency variations in the image.

Cubic Interpolation:

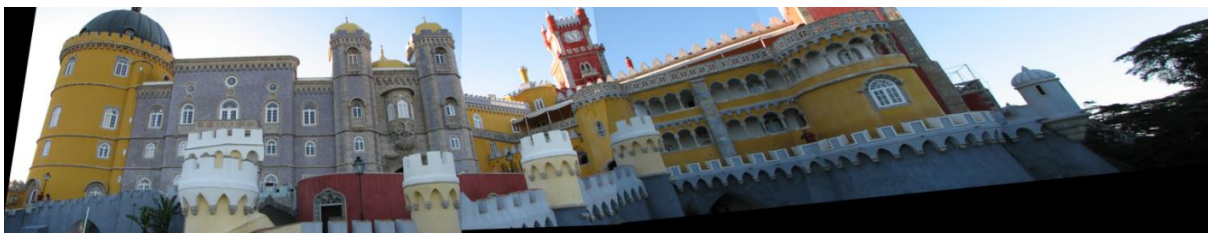
- Cubic Interpolation estimates pixel values using a cubic polynomial function fitted to the surrounding pixels. It provides a smoother and more accurate estimation compared to the linear interpolation.
- Advantages:
    1. Improved accuracy – Cubic Interpolation can capture more complex variations in pixel intensities and represent finer details, leading to a more faithful reproduction of the image.
    2. Better preservation of edges – it can mitigate the blurring effect around edges and sharp transitions compared to linear interpolation.
- Disadvantages:
    1. Increased computational complexity – Cubic interpolation involves more calculations compared to linear interpolation, resulting in slower performance.
    2. Possibility of overshooting artifacts – in some cases, cubic interpolation may introduce overshooting artifacts, where the estimated pixel values exceed the actual intensity values, leading to exaggerated features or noise amplifications.

Using the linear interpolation, we arrived at the following result (after stitching both pictures together):

Whereas, when we used the cubic interpolation the results we as follow (this took 20 minutes of running time compared to the former 1 minute...):

3.4 – After using all the functions we've implemented above, we arrived at the above results for the panoramic pictures of both 'incline' pictures – the first one with the linear interpolator, and the latter with the cubic interpolator.

3.5 – Added below are the panorama pictures of the beach and sintra:



As described above (in section 3.1) the SIFT descriptor provides us with a lot of matches, where not all of them are necessarily "good" matches – hence they take away from the accuracy of the homography matrix H which we compute using those points. Since the computation of the matrix may be executed from only 4 points, we would like

to take the minimal number of points, where we will not deter from the quality of the matrix provided from the function on one hand, but on other hand not ignore too many points.

When we use too many points in the SIFT descriptor in order to calculate the homography matrix, we have received a horrible homography matrix H, which provided us with results that does not make sense for the human eye. Added below are the results of the panorama using all the points of SIFT descriptor on the 'incline' pictures:

3.6 – Added below are the pictures with RANSAC activated instead of out computeH function using SIFT descriptor:

RANSAC is needed when there are outliers or erroneous data points present in the dataset. These outliers can severely affect the accuracy and reliability of parameter estimation algorithms. RANSAC helps overcome this issue by robustly estimating parameters by iteratively selecting random subsets of data points, fitting a model, and evaluating the quality of the model based on a predefined threshold. It is particularly useful when the percentage of outliers is relatively high.

As for the pictures received with RANSAC – without RANSAC, there is a higher probability (as we have gotten when we used all the points in the SIFT descriptor) that outliers would affect the result. This can lead to misalignment, ghosting, or blurring in the stitched image, especially in areas with inconsistent or incorrect matches.

With RANSAC, there is an improvement in the robustness of the stitching process. It helps identify and discard outliers during the estimation of the transformation matrix. By iteratively selecting random subset of correspondences and fitting a model, RANSAC can filter incorrect matches and produce a more accurate transformation.

To achieve a better result in creating the panorama images, some potential improvements could be considered:

1. Increasing the number of RANSAC iterations – increasing the number of iterations allows for a more exhaustive search and increase the likelihood of finding a better model fit. However, this comes at a cost of increased computational time.
2. Adjusting the tolerance parameter – the tolerance threshold determines the maximum allowed distance for some correspondence to be considered an inlier. Fine tuning this threshold can help strike a balance between removing outliers and preserving valid correspondences

3.7 – For our panorama of choice, we chose a few pictures I took from a trip to Georgia of the Russia – Georgia friendship monumental, which are added here below:





Added below is the received Panoramic picture:

As we can see, although the alignment of the pictures came out well, a few details of the pictures didn't show up at the output (such as the sky which arrive in some of the sub-pictures) – this is due to fact that we chose an output size for the picture which matched the length of a single picture. Had we chosen a broader length for the output size, we could've gotten more information from the end result.