Computer Vision - EE 046746
HW3

Tomer Raz Adi Hatav

August 14, 2024

Contents

Part 1 Classic vs. Deep-learning-based Semantic Segmentation

1.1 Load images

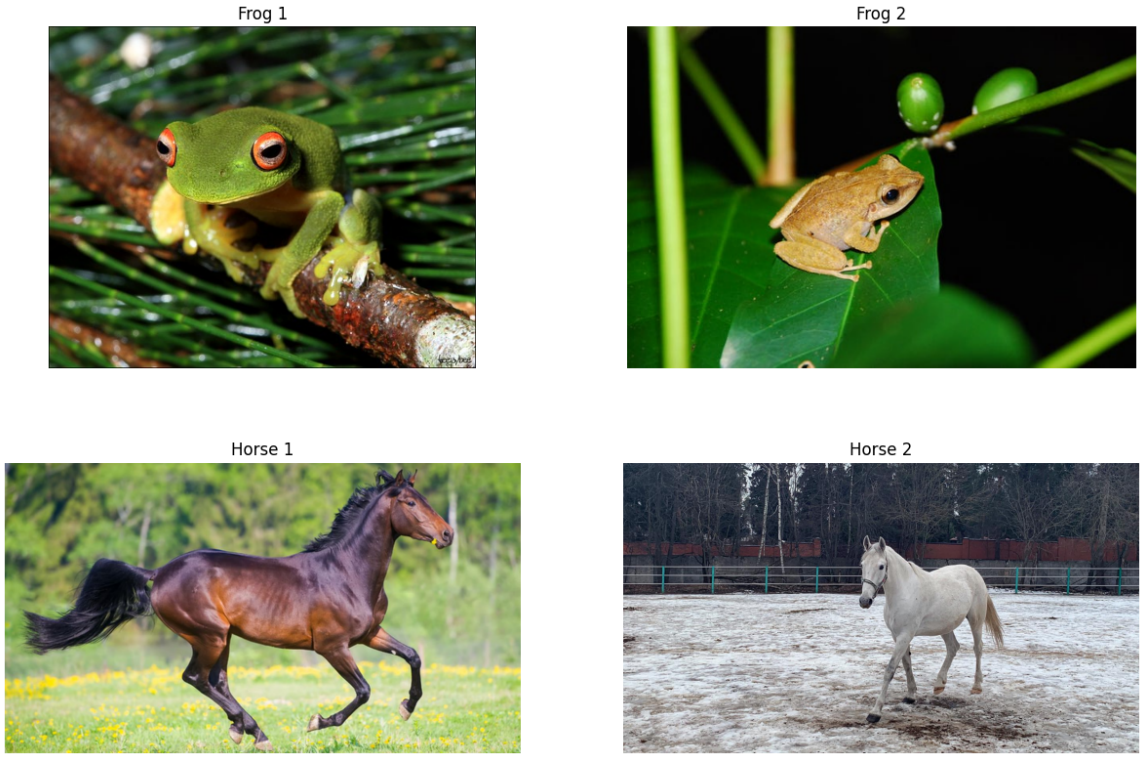

Figure 1: Images of frogs and horses.

1.2 Classic and deep segmentation methods

See fig. 10 for a full side-by-side comparison of all of the methods.

1.2.1 Classical image segmentation methods

We tried K-means clustering, Simple Linear Iterative Clustering (SLIC), and GrabCut.

K-means Clustering

K-means clustering is a popular method for image segmentation that works by partitioning the image into k clusters based on pixel intensity. In our experiments, we set $k = 2$ to distinguish the foreground from the background.

Advantages:

- Fast and easy to implement.
- Works well when there is a distinct separation in pixel intensity.

Disadvantages:

- The method includes more parts of the image than just the desired segmented item, which can lead to over-segmentation.
- Does not account for spatial information, leading to potential misclassification of adjacent pixels with similar intensities.

Simple Linear Iterative Clustering (SLIC)

SLIC is a superpixel segmentation method that groups pixels into clusters with similar color and spatial proximity, resulting in a segmented image composed of superpixels.

Advantages:

- Very fast and efficient.
- Preserves the boundaries of objects well by considering spatial information.

Disadvantages:

- Tends to divide the image into many small sections, which may not correspond to meaningful objects.
- Does not combine the correct sections together effectively, leading to fragmented segmentation results.

GrabCut

GrabCut is an interactive foreground extraction method that iteratively refines the segmentation using a Gaussian Mixture Model (GMM) and graph cuts.

Advantages:

- Produces high-quality segmentation results by iteratively refining the segmentation.
- Effectively separates the foreground from the background even in complex images.

Disadvantages:

- Computationally more expensive and takes significantly more time compared to other methods.
- GrabCut is a stochastic algorithm, meaning results vary depending on the random seed used. However, when comparing results for some of the images with those obtained in the final section of section Part 1, the difference between these arises from different image resolutions (resolution was lowered for the final plot).
- Depends on placement of initial bounding box for the foreground, which can be less convenient in a fully automated system.

From our results (Figure 2), it is evident that GrabCut provided the best segmentation quality, though it required more time to process and sometimes did not capture the main subject of the image. SLIC was extremely fast but tended to over-segment the image into many small regions. K-means was fast and worked reasonably well with $k = 2$, but it included more parts of the image than just the desired segmented item.

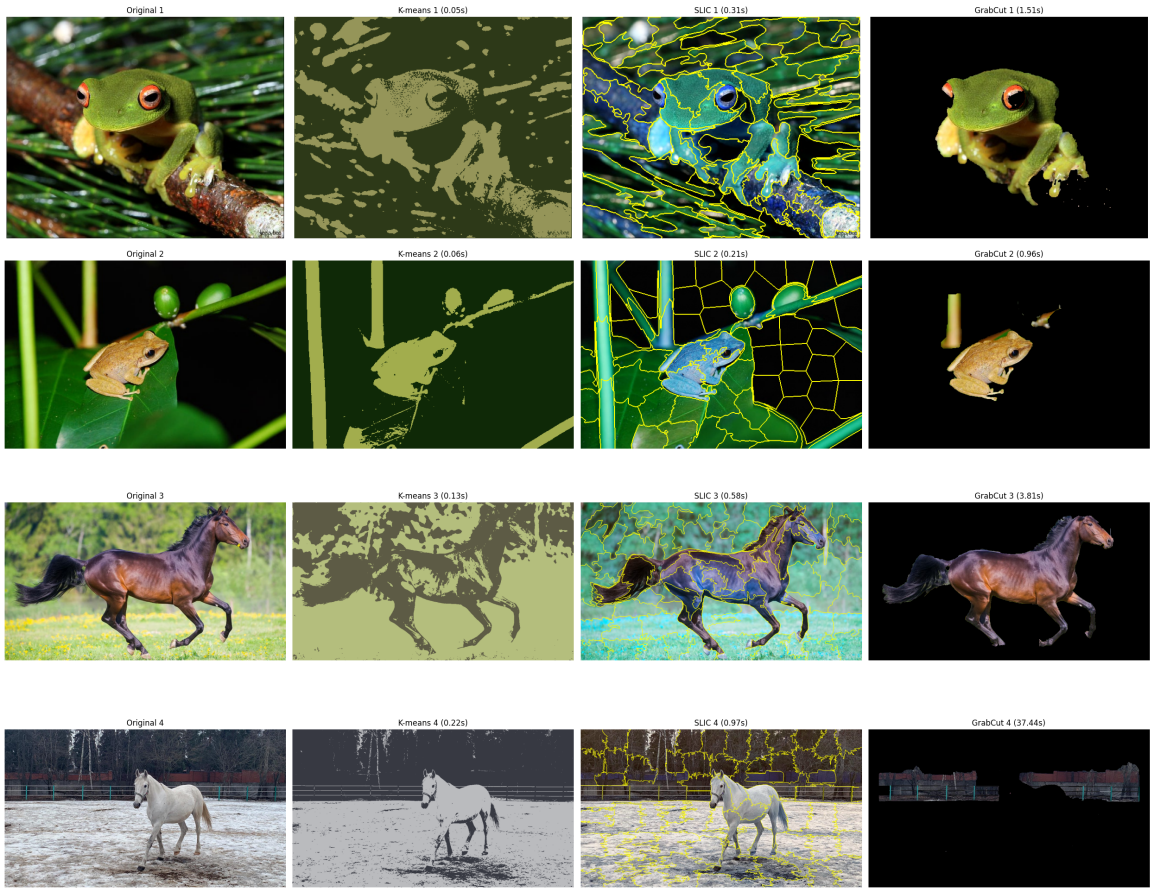

Figure 2: Results of segmentation methods applied to images. From left to right: Original image, K-means segmentation, SLIC segmentation, and GrabCut segmentation. The processing times for each method are included in the titles of the segmented images.

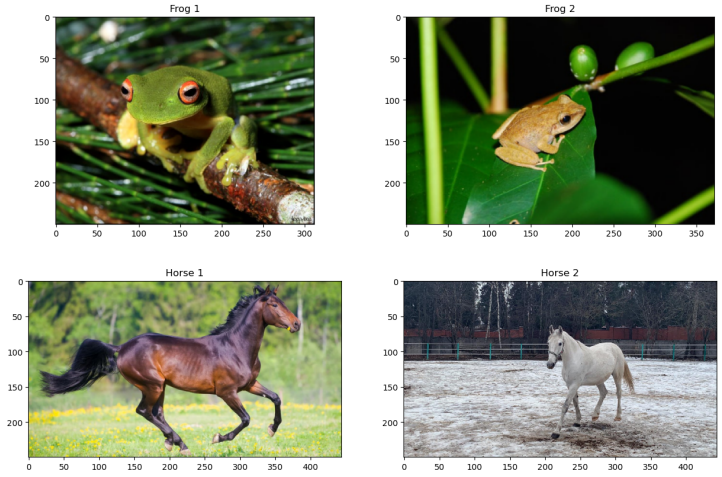

Figure 3: Resized frogs and horses.

Figure 4: SAM segmentations of resized frogs and horses. The time it took to compute the segmentations (running on the Nvidia GPU specified earlier) is displayed in each subtitle.

1.2.2 SAM segmentation

Meta's **Segment Anything Model (SAM)** (Kirillov et al., 2023) is a Vision Transformer (ViT) based architecture composed of three parts:

1. Image encoder,
2. Prompt encoder (taking as input simple masks in the form of points and boxes), and
3. Mask decoder, which returns the segments of the image given input from the other two parts of the model.

Specifically, the model used here is

- sam_checkpoint = "sam_vit_h_4b8939.pth"
- model_type = "vit_h"

Advantages:

- Powerful zero-shot capabilities: SAM can segment any image without fine-tuning, making it adaptable across various applications.
- Precision: Its deep-network architecture captures complex patterns, resulting in accurate and detailed segmentation.

Disadvantages:

- Resource Intensive: The model requires significant computational resources for feasibility. We ran the model on an NVIDIA GeForce RTX 3050 6GB Laptop GPU which did not have enough memory for high resolution images. The online demo has better infrastructure and is able to process high resolution images at a fraction of the time: https://segment-anything.com/demo#.

1.3 Display 3 images of our choice

Figure 5: Images of a romantic dog (Effi), a car and a wheel.

1.4 Apply the two methods to each of the images

See fig. 10 for a full side-by-side comparison of all of the methods.

1.4.1 Classical image segmentation methods

Figure 6: Results of segmentation methods applied to three images. From left to right: Original image, K-means segmentation, SLIC segmentation, and GrabCut segmentation. The first row shows a dog image, the second row shows a car image, and the third row shows a ship's wheel image. The processing times for each method are included in the titles of the segmented images.

1.4.2 SAM segmentation

Figure 7: Segmentation of the original, high-definition image (left), vs. our resized image (right). Visually, the segmentation does not seem to have been impacted significantly by downsampling of the image. Note the high-resolution image on the left was processed on the demo website discussed above, hence the difference in mask representation.

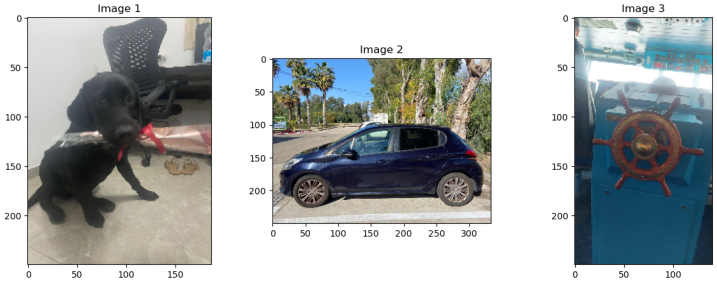

Figure 8: Resized custom images.

Figure 9: Segmented resized custom images.

1.4.3 Compare and contrast the methods

Which method performed better on each image? Our thoughts on method performances.

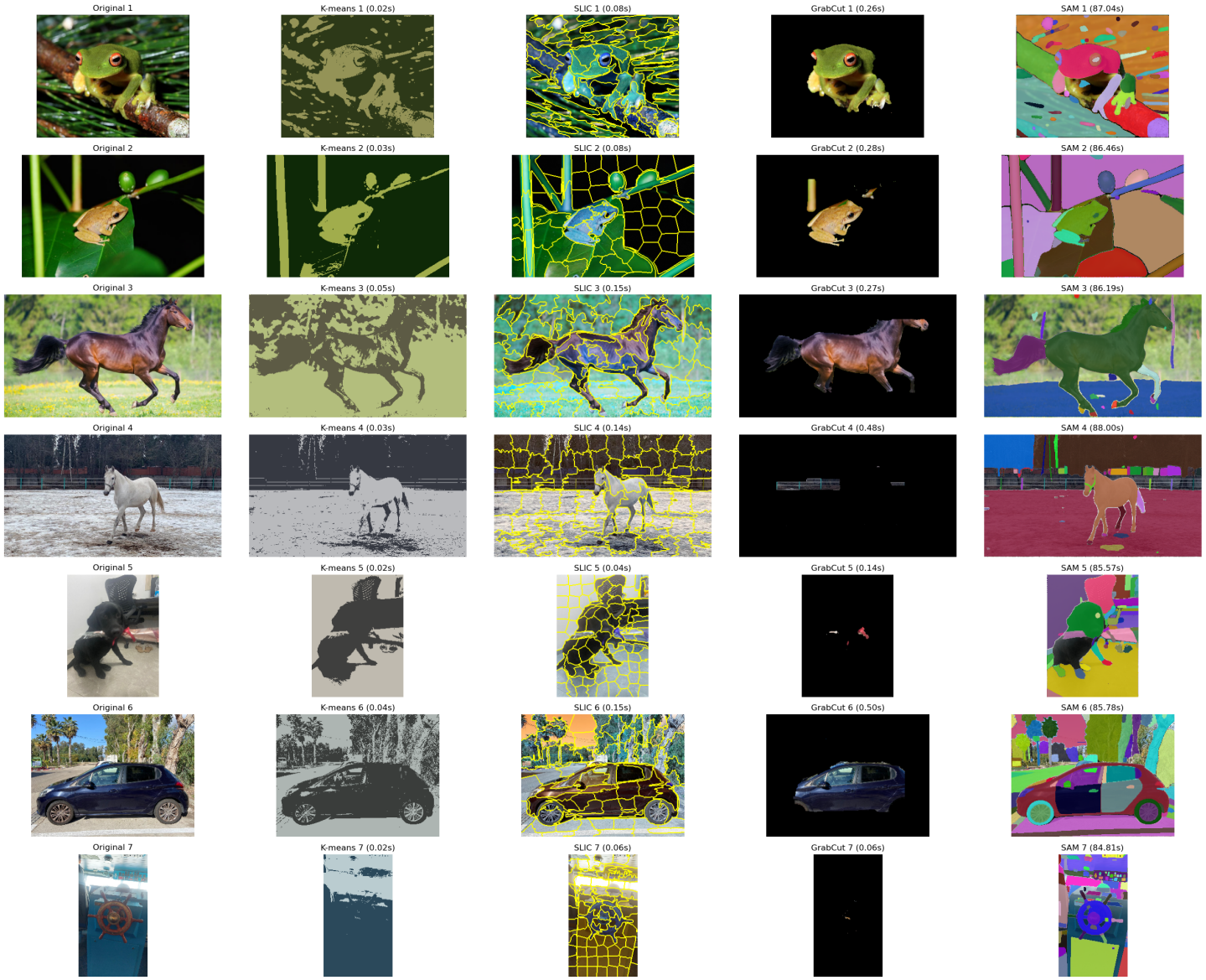

Figure 10: All segmentation methods used in section Part 1. GrabCut results are evidently different than the ones showed earlier. This may be a consequence of GrabCut's stochastic nature, but is mostly one of lowering the image resolution (a conclusion reached after running the same GrabCut code multiple times - results do not visually vary significantly). SAM captures distinct image elements better than the other methods.

- GrabCut seems to return the best segmentation of the image into foreground and background parts. This makes sense - its architecture is designed for such an operation. The only issue is that depsite some runs returning favorable results, other runs of the algorithm result in segmentations that miss the main object of the image and therefore fail to be useful.
- The Kmeans method returned splits of the images into foreground and background that are less convincing than those returned by GrabCut. This is also to be expected as the Kmeans method simply classifies based on pixel intensity in RGB space. Sometimes objects are legible in the segmented images.
- SAM has the best results, visually, for image segmentation into distinct parts of interest, i.e. a frog's limb or a car's wheel. It is the most expressive, and distinguishes the most elements of the images as separate constituents. Furthermore, the model allows prompting via points of interest or boxes. See fig. 11 for example.
- Regarding SLIC, superpixels returned by the method do seem to segment the image along key borders between parts. No effort however was put into turning these superpixels into masks like in the other methods.

Figure 11: Using a point prompt, SAM was able to predict with highest probability the mask shown above as the requested object.

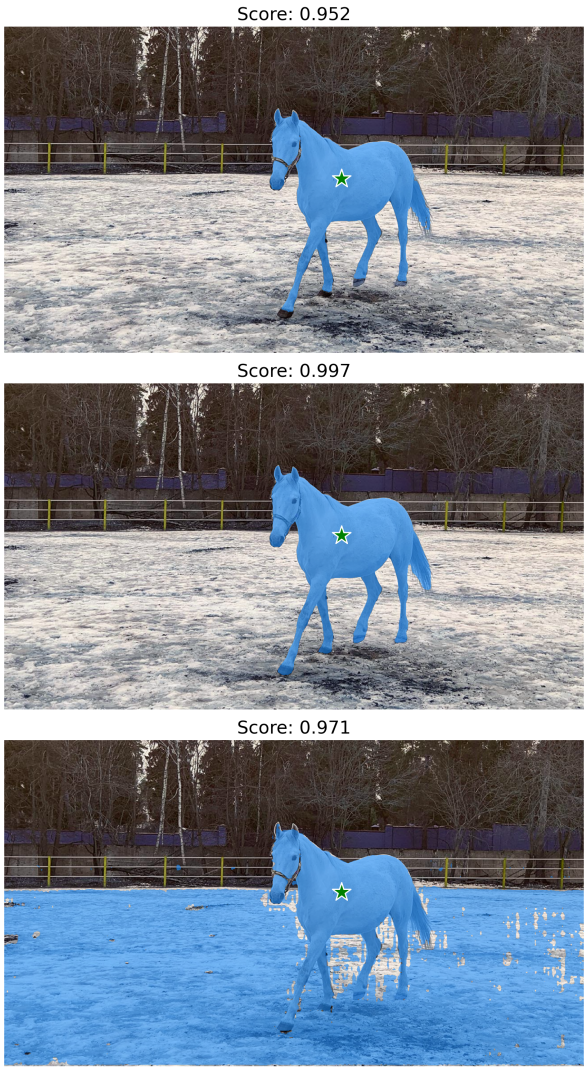

Figure 12: The three masks predicted by SAM given the point prompt, including the one with highest score that was shown above (center image).

SAM's prompt capabilities lend it an edge over the other methods, and it will therefore be the method of choice for the next section.

1.5 Problem: rough and noisy edge segmentation

Addressing the issue of rough and noisy edges in segmentation masks involves a combination of pre-processing techniques to enhance the image quality, algorithmic improvements to refine the segmentation process, and post-processing methods to clean up and smooth the boundaries. Applying these techniques can significantly improve the accuracy and visual quality of the segmentation results. We will explain on pre-processing techniques and algorithmic improvements.

1.5.1 Pre-processing techniques

- **Smoothing Filters:** Applying Gaussian or median filters to the input image can help reduce noise and smooth the edges before segmentation. This preprocessing step can make the segmentation boundaries more consistent.
- **Contrast Enhancement:** Techniques such as histogram equalization or can improve the contrast of the image, making the boundaries of objects more distinct and easier to segment accurately.
- **Edge Detection:** Applying edge detection algorithms to highlight the boundaries of objects can help in refining the initial mask generated by the segmentation algorithm.

1.5.2 Segmentation algorithm improvements

- **Parameter Tuning:** Tuning the parameters of the segmentation algorithm (e.g., the number of clusters in K-means, compactness in SLIC) can lead to better-defined boundaries.
- **Multi-scale Segmentation:** Using multi-scale approaches that segment the image at different resolutions and then combine the results can capture both fine and coarse details, leading to smoother boundaries.

1.5.3 Post-processing techniques

- The main idea we have for post processing that may generally work is using an ensemble of generated masks to generate a new and more refined mask (one option could be averaging the masks out, weighting them according to their scores). This could smoothen the final mask.

Part 2 Jurassic Fishbach

2.1 Video of ourselves

Let there be juggling (see fig. 13)! In our video, a (suspicious-looking) man jumps into the frame and starts juggling an assortment of round items: a baseball, a lemon, a tennis ball and a juggling ball. Maybe one of these items will be able to stop the ferocious dinosaur around the corner!

Figure 13: Two frames from the video.

2.2 Segment ourselves

We chose to segment the suspicious human out of the video using SAM with a combined point and box prompt in the center of the image (fig. 14).

Figure 14: The two frames from earlier, now segmented using SAM with a combined point and box prompt in the center of the image.

2.3 Segmenting a video model

We chose to segment the dinosaur model. Since the background is a homogeneous green, to extract the dinosaur we initially suspected nothing more is needed other than selecting the pixels that are not this green color, i.e. no need for classic segmentation algorithms and deep algorithms. However, after attempting this naive approach it was evident that the green color of the green screen is not constant enough for a successful segmentation. We therefore turned to kmeans. Results for the initial attempt and the successful attempt are shown below (fig. 16, fig. 17).

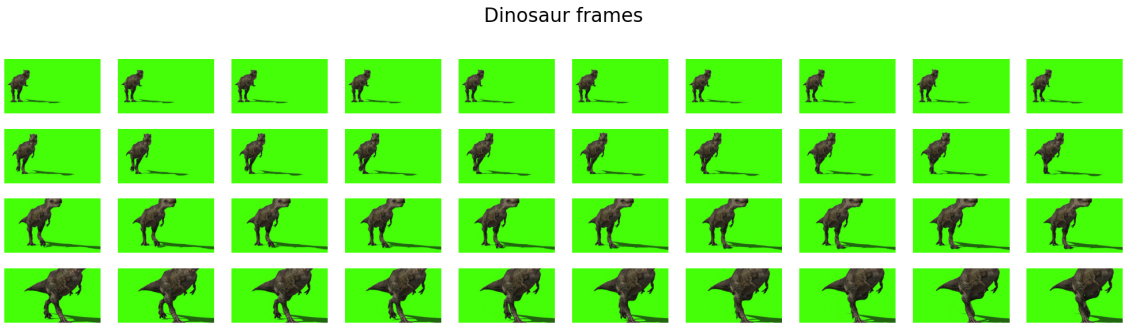

Figure 15: Dinosaur frames. Yikes!

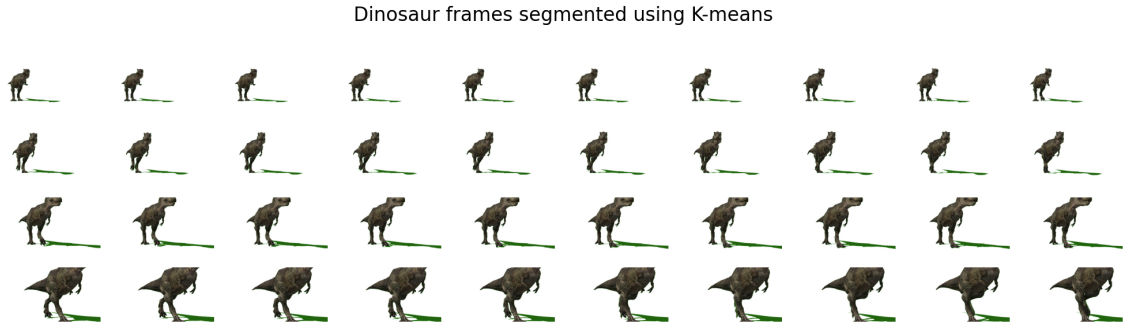

Figure 16: Successful green-screen segmentation using kmeans.

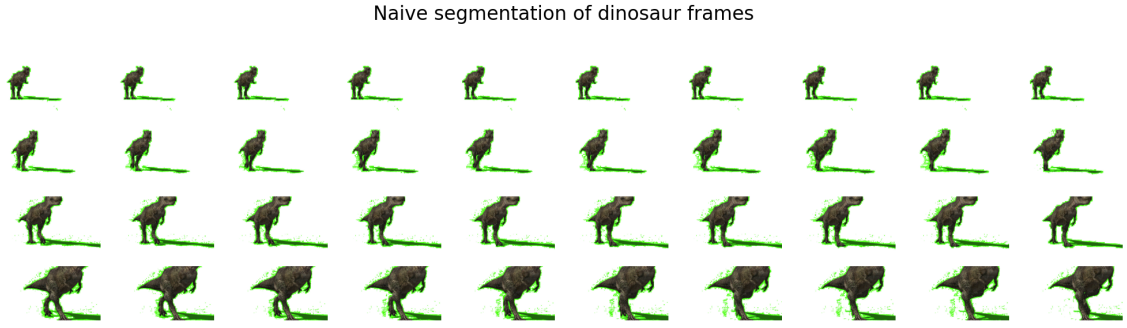

Figure 17: Unsuccessful segmentation using the naive approach of comparing pixel color to the solid green of most of the background. Some of the background made it to what is supposed to be a clean cut of the dinosaur.

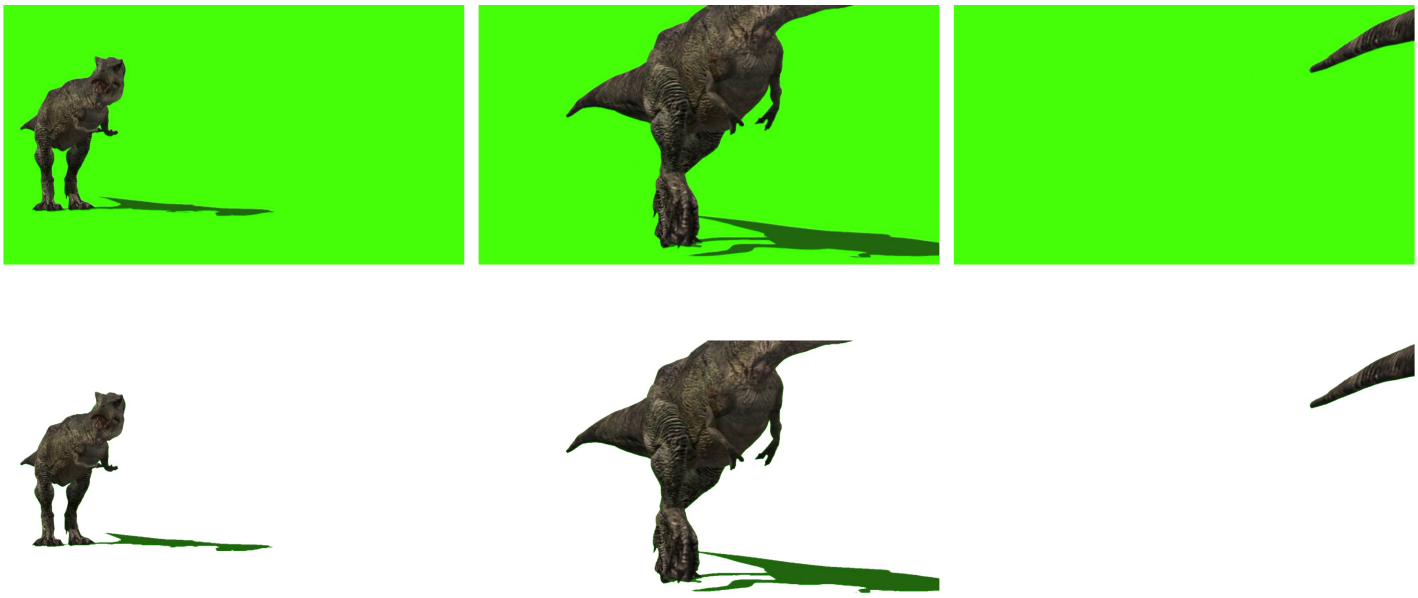

Figure 18: A closer look.

2.4 Putting it together: combined video from segmented images

Figure 19: An array of frames from the video.

Figure 20: How the juggler came to fend off the hungry dinosaur that was on the attack.

Part 3 Planar Homographies: Image Stitching Algorithm

Meaning, i.e., sequence of images to panorama.

3.0 SIFT implementation

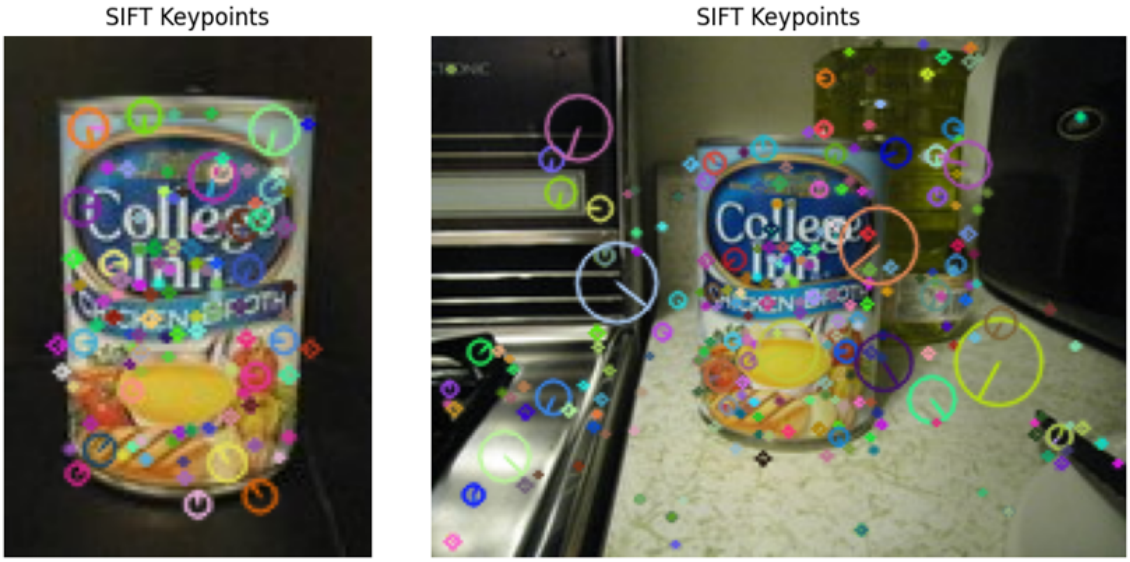

Figure 21: SIFT Keypoints detected in two images of the same object (a can of chicken broth). The left image shows the keypoints detected in the first image, while the right image shows the keypoints detected in the second image. The size and orientation of the circles represent the scale and orientation of the detected SIFT keypoints, respectively.

3.1 Finding corresponding points using SIFT

In this section, we employed SIFT to identify keypoints in two images and match corresponding points between them.

The process of finding corresponding points using SIFT involves the following steps:

1. **Keypoint Detection**: SIFT detects keypoints in both images independently. These keypoints are areas in the images that have distinctive patterns, such as corners, blobs, or edges.
2. **Descriptor Computation**: For each detected keypoint, a descriptor vector is computed. This vector represents the local image gradient around the keypoint and is invariant to scaling, rotation, and illumination changes.
3. **Keypoint Matching**: The descriptors from the first image are matched with those from the second image using a nearest-neighbor search. Each keypoint in the first image is matched with its closest counterpart in the second image based on the descriptor's Euclidean distance.
4. **RANSAC Filtering**: To improve the robustness of the matching, as thought in the lecture, we used RANSAC algorithm. RANSAC iteratively estimates the homography between the images and filters out outliers, retaining only the keypoints that agree with the computed transformation.

Figure 22 shows the detected keypoints and the corresponding matches between two images of the same object (a can of chicken broth). The images were padded to have the same height before being placed side by side for visualization. The green circles represent the detected keypoints, while the blue lines connect the corresponding points between the two images.

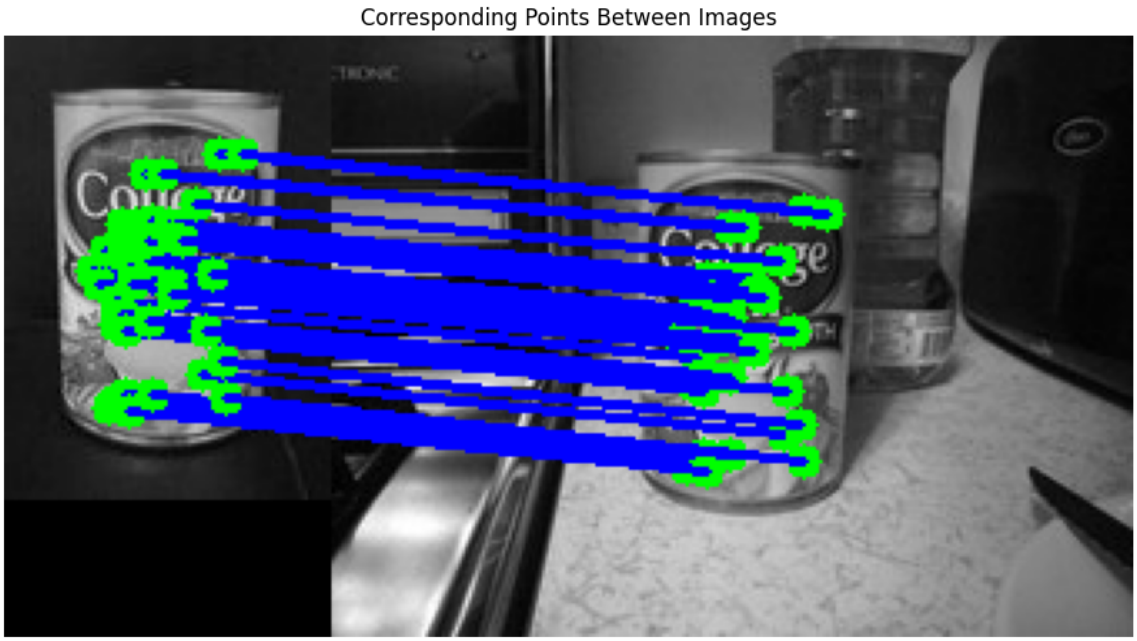

Figure 22: Corresponding points between two images using SIFT. The images have been padded to ensure the same height. Green circles indicate detected keypoints, and blue lines represent the corresponding matches between the images.

This visualization confirms that the SIFT algorithm successfully identified and matched several keypoints between the two images.

3.2 Calculate transformation

To compute the transformation between two images, we need to determine the homography matrix H that maps points from the first image to corresponding points in the second image.

As a sanity check, we applied the SIFT-based keypoint matching algorithm to the same image (`im1` with itself) and computed the homography matrix $H2to1$. Since the images are identical, the computed homography should ideally be the identity matrix, indicating no transformation is needed between the two images. The resulting homography matrix $H2to1$ is shown in Figure 23. The matrix is very close to the identity matrix, with slight deviations due to numerical precision errors inherent in floating-point calculations.

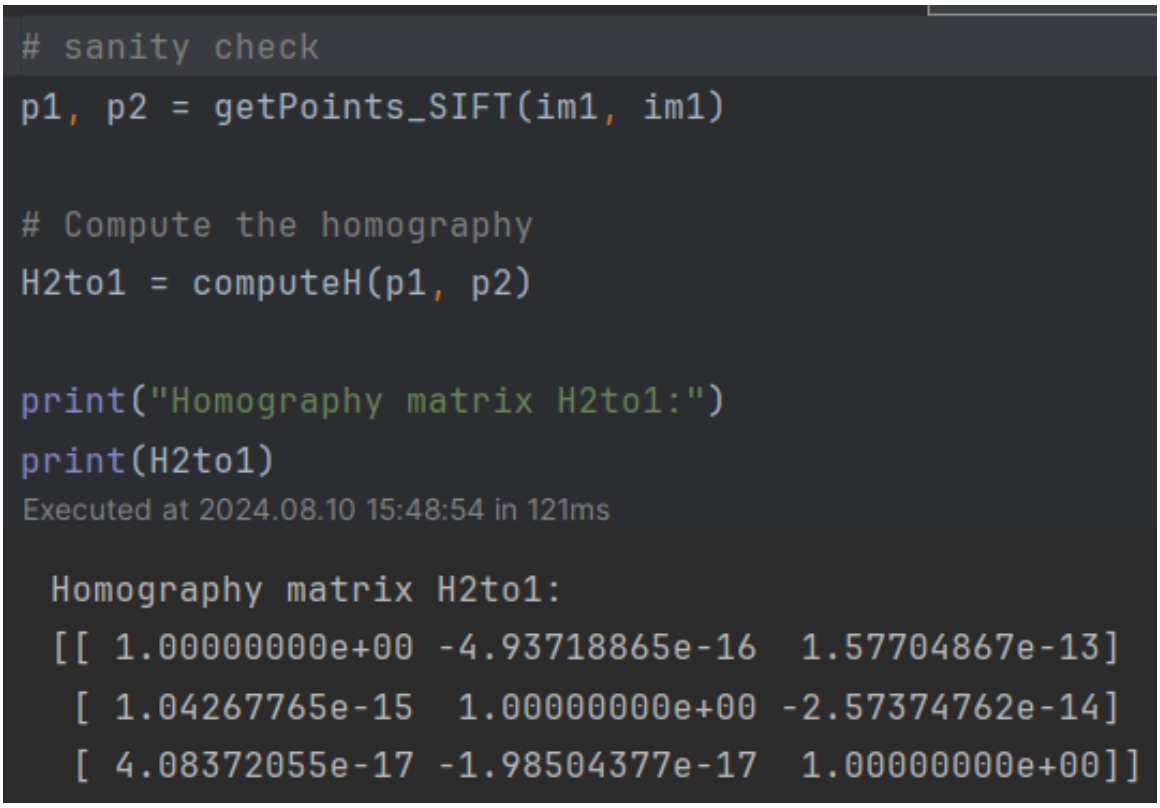

Figure 23: Computed homography matrix $H2to1$ when matching the same image with itself. The matrix is nearly the identity matrix, as expected.

3.3 Image warping

In this experiment, we applied a homography transformation to warp an image using two different interpolation methods: linear and cubic. The original images and the resulting warped images are shown in Figure 24.

Both interpolation methods produced similar results, which is expected when the image does not contain high-frequency details or sharp edges that require more complex interpolation.

Figure 24: Comparison of warping using different interpolation methods. Top-left: Original Image. Top-right: Second Original Image. Bottom-left: Warped Image using Linear Interpolation. Bottom-right: Warped Image using Cubic Interpolation.

As shown in the results, the differences between the two methods are minimal in this specific case. Both methods successfully aligned the images.

3.4 Panorama stitching

In this section, we implemented a function to create a panorama by stitching two images together after aligning them using a homography matrix. The panorama is created by warping one image to align with the other and then overlaying the images on a black canvas. The resulting image is a continuous view that combines the two original images into a single, seamless panorama.

The resulting panorama is shown in Figure 25. The panorama successfully combines the two images, creating a broader view of the scene.

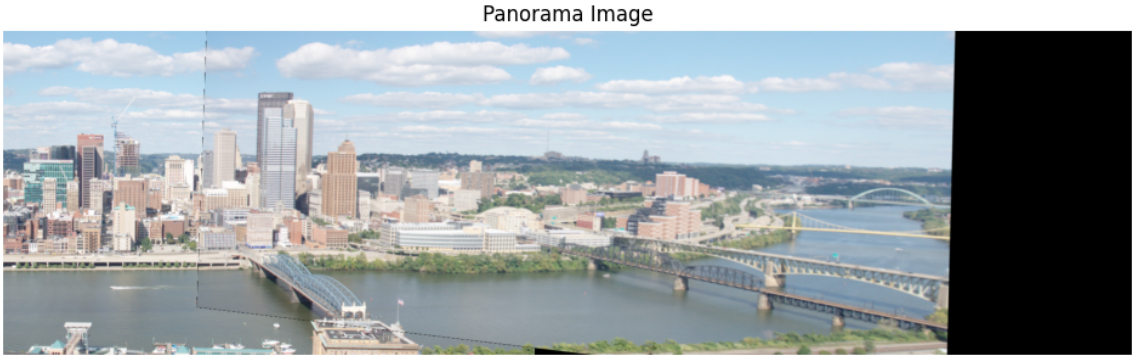

Figure 25: Panorama created by stitching two images from the ‘incline‘ dataset. The panorama is formed by aligning and overlaying the right image onto the left image.

3.5 Stitching several images

In this section, we attempted to create a panorama using multiple images of the Sintra castle. Due to the high resolution of the images, the processing time became significantly lengthy. To address this, we reduced the quality of the images by resizing them to 30% of their original size. This resizing step was necessary to make the processing more manageable and to ensure that the algorithm could complete the task in a reasonable amount of time.

Moreover, during the stitching process, we encountered difficulties in finding consistent matches when attempting to concatenate all the images sequentially, one at a time. The algorithm struggled to find reliable corresponding points across all images when processed individually in sequence.

To mitigate this issue, we create two separate panoramas: one from the first three images and another from the last two images. These two smaller panoramas were then stitched together to form the final panorama. This approach of creating smaller panoramas first, and then combining them, proved to be more effective in achieving a successful and visually coherent final panorama.

The final stitched panorama, shown in Figure 26, illustrates the result of this approach, demonstrating the effectiveness of breaking down the process into smaller, more manageable steps.

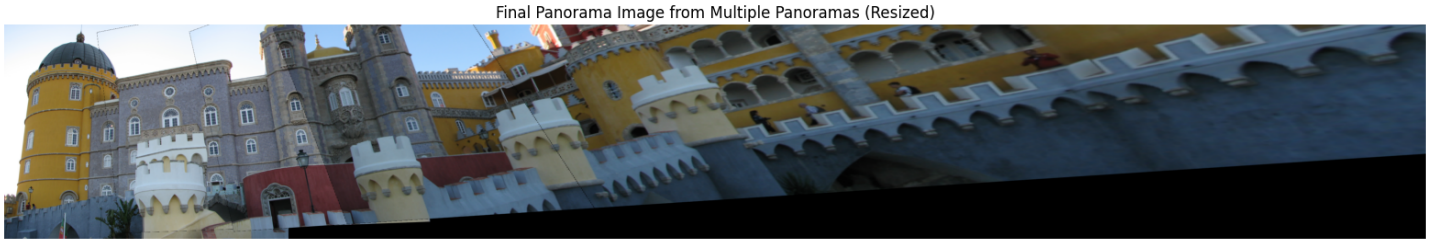

Figure 26: Final Panorama Image from Multiple Panoramas

3.6 RANSAC in panorama stitching

What is RANSAC and When is it Needed?

RANSAC is an iterative method used to estimate parameters of a mathematical model from a set of observed data points that contain outliers. In the context of image stitching, RANSAC is particularly useful for computing the homography between two sets of corresponding points, where some matches may be incorrect due to noise, errors in feature detection, or other anomalies.

RANSAC is needed when the dataset (in this case, the matched keypoints between images) is likely to contain outliers that can negatively affect the model's accuracy. By iteratively selecting random subsets of the data and computing the homography, RANSAC identifies the best model that fits the largest number of inliers, effectively filtering out outliers.

Comparison: Using RANSAC vs. Not Using RANSAC

In our experiments, we used RANSAC to create panorama images for SINTRA images. The results were better when RANSAC was applied compared to when it was not.

- **Using RANSAC:** As discussed in the previous sections, we initially implemented RANSAC to improve the panorama stitching results. The rationale behind this choice was based on the discussion in class, where we concluded that RANSAC is generally superior for such tasks due to its robustness against outliers. The panoramas created using RANSAC were coherent, with minimal distortions and smooth transitions between the images.

- **Not Using RANSAC:** When RANSAC was omitted, the panorama stitching process became highly sensitive to outliers. The resulting panoramas were heavily distorted, especially when attempting to stitch more than two images. For instance, when trying to create a panorama of all five SINTRA images without RANSAC, the algorithm failed entirely, producing unrecognizable and warped images. Even with just two or three images, the results were far from satisfactory, with clear misalignments and visual artifacts.

The drastic difference in the quality of the results highlights the importance of RANSAC in image stitching, particularly when dealing with complex scenes or multiple images.

Improving the Results

While RANSAC significantly improved the quality of the panoramas, there are additional steps that could be taken to further enhance the results:

1. Better Keypoint Detection and Matching: - Using more sophisticated feature detectors and descriptors (other than sift) could improve the accuracy of the keypoints and reduce the number of outliers, thereby making the panorama stitching more robust even without RANSAC.

2. Multi-Scale Processing: - Implementing a multi-scale approach where images are processed at different resolutions could help in better aligning images with varying scales or perspectives, reducing the chances of misalignment.

3. Refining the RANSAC Threshold: - Fine-tuning the RANSAC parameters, such as the inlier threshold and the maximum number of iterations, could lead to even better results.

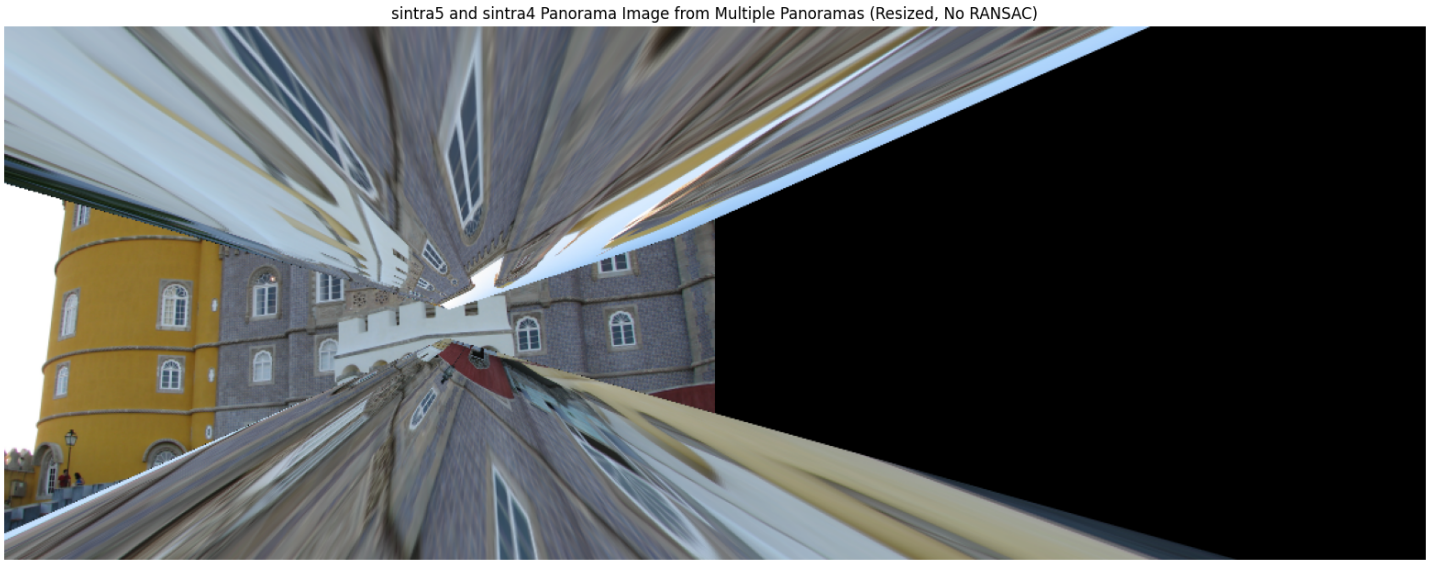

Figure 27: sintra5 and sintra4 Panorama Image from Multiple Panoramas (Resized, No RANSAC)

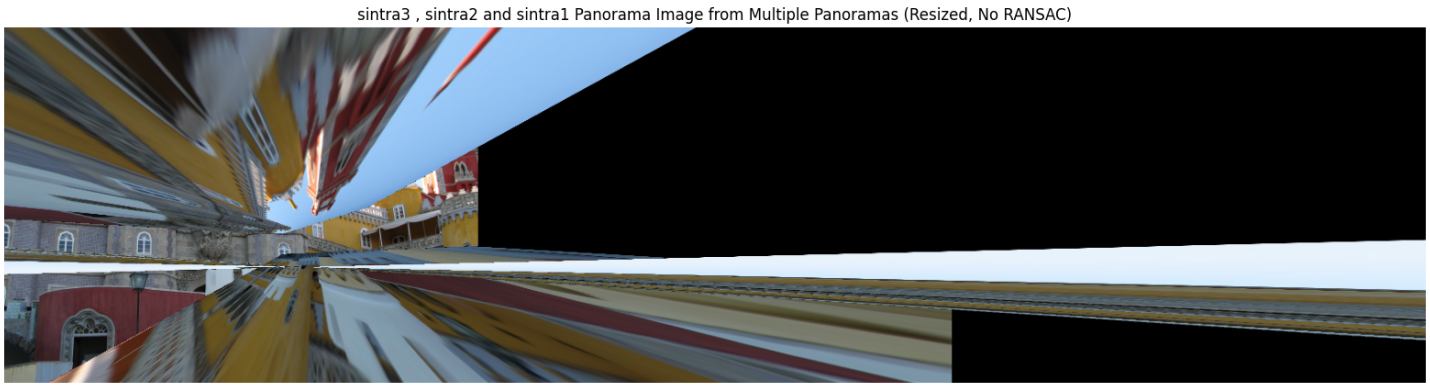

Figure 28: sintra3, sintra2, and sintra1 Panorama Image from Multiple Panoramas (Resized, No RANSAC)

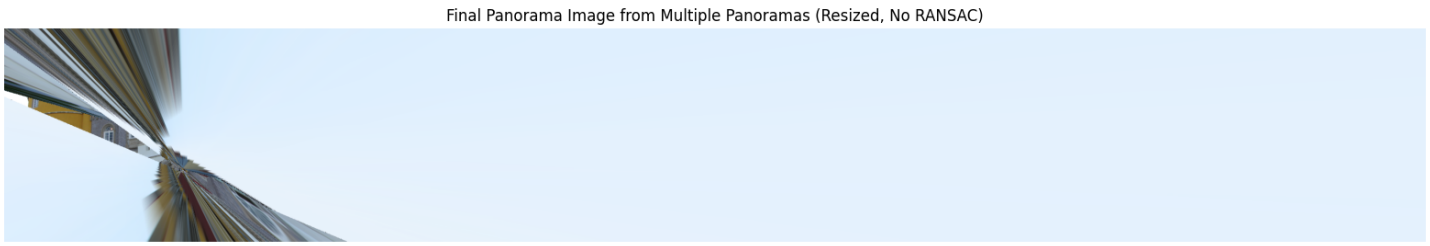

Figure 29: Final Panorama Image from Multiple Panoramas (Resized, No RANSAC)

3.7 Be creative

In this section, We went out and captured a series of photos of a building in the street. The goal was to apply the techniques learned throughout this project to create a seamless panorama from these images. The resulting panorama is shown in Figure 30.

Figure 30: Final street Panorama Image

To create this panorama, we used the same SIFT-based approach with RANSAC as in the previous sections. The images were taken with the intention of capturing different parts of the scene with some overlap, ensuring that there were enough common features for accurate keypoint matching.

The panorama was created by stitching together the individual images, aligning them using the computed homographies, and then blending them to produce the final seamless result.

Part 4 Dry Questions

4.1 a) Translation $x' = x + u$, $y' = y + v$

The homography matrix for translation is:

$$H_{\text{translation}} = \begin{bmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{bmatrix}$$

Given a general point (x, y), the new coordinates after translation are calculated as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & u \\ 0 & 1 & v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + u \\ y + v \\ 1 \end{bmatrix}$$

So, the point (x, y) is translated to $(x + u, y + v)$.

4.2 b) 90 degree rotation in the clockwise direction

The homography matrix for a 90-degree clockwise rotation is:

$$H_{\text{rotation}} = \begin{bmatrix} 0 & -1 & \text{image\_height} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given a general point (x, y), the new coordinates after rotation are calculated as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & \text{image\_height} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -y + \text{image\_height} \\ x \\ 1 \end{bmatrix}$$

So, the point (x, y) is rotated to $(-y + \text{image\_height}, x)$.

4.3 c) Mirroring around the line $x = a$

To mirror an image around the line $x = a$, the homography matrix is:

$$H_{\text{mirror}} = \begin{bmatrix} -1 & 0 & 2a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given a general point (x, y), the new coordinates after mirroring are calculated as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 2a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -x + 2a \\ y \\ 1 \end{bmatrix}$$

So, the point (x, y) is mirrored to $(-x + 2a, y)$.

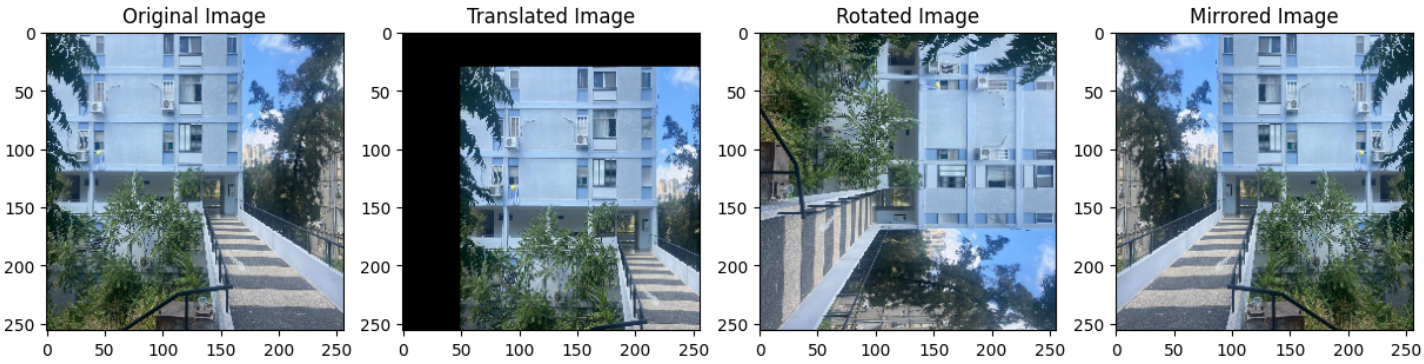

Figure 31: Image transformations: (a) Original Image, (b) Translated Image (shifted right and down), (c) Rotated Image (90 degrees clockwise), (d) Mirrored Image (flipped across a vertical axis).

References

A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick. Segment anything. *arXiv:2304.02643*, 2023.