Computer Vision - Project 3

Multi-view Image Sequence Estimation

Lorenzo Schiavone

January 1, 2026

1 Assignments

Given an unordered set of images captured sequentially, estimate the correct sequence order by detecting similarities between pairs. Write a Python code including:

1. Load and preprocess the unordered set of images,
2. For each image extract meaningful features with SIFT or ORB,
3. Compute matches between different images pairs using the features above,
4. Based on the feature matches estimate the sequence order.

Do it for the *easy* RGB upper loop and then for the *challenging* set. As optional tasks,

- Repeat it for the Near-Infrared (NIR) images,
- Repeat the task include the lower loop without knowing which frames belong to each loop.

2 Main Task

Load and preprocess: After loading the images, we equalize the luminosity component in the LAB color space. This serves as a normalization for effectively compare the images in the next stages.

Extract SIFT Features: Afterwards we extract the SIFT features from each equalized image. We didn't modify the openCV SIFT default parameters as they were already good enough.

Compute matches: For computing the matches between different images we prefer to use the Flann based Matcher to the Brute Force Matcher as shown in openCV documentation. Instead of comparing every pair of features, it divides the space into regions in order to avoid comparing points clearly too different. This choice was necessary when computing matches for both upper and lower loop together since the brute force matcher took too long. We decide a match is good if the ratio between the distance of the first and second closest match is less than 0.6, that is low to keep only true strong matches and reduce the amount of false positive. For each pair we count how many good matches are there and fill a similarity matrix `good_matches` in the corresponding position. Close images in the loop have an higher number of good matches as the feature matching with the SIFT features is good with small change in the viewpoint angle but then it degrades increasingly. Figure shows this both for the *easy* and *challenging* scenario: there is a large difference in the amount of good matches found between close and far pictures in the loops.

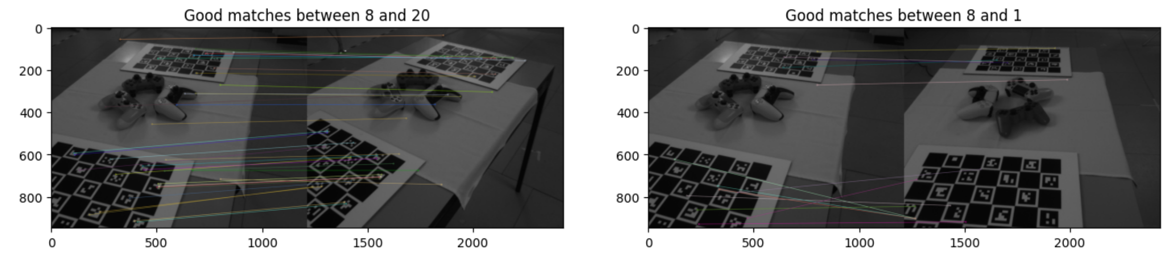

Figure 1: Amount of good matches for near (left) and far pictures for the *challenging nir* upper loop.

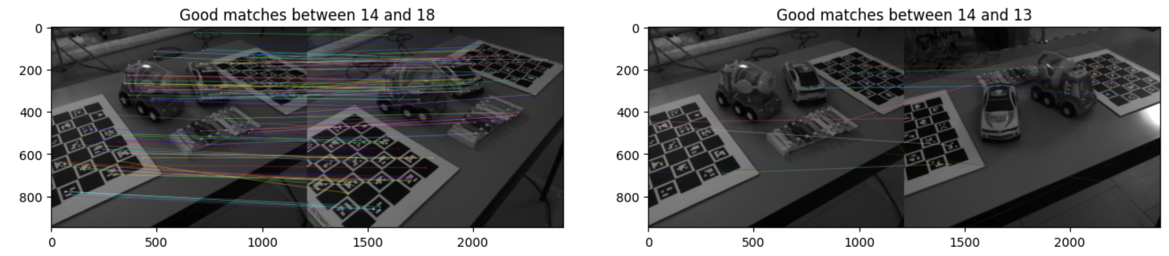
Good matches between 14 and 18
Good matches between 14 and 13
0
200
400
600
800
500
1000
1500
2000

Figure 2: Amount of good matches for near (left) and far pictures for the *easy nir* upper loop.

Sequence Order Estimation: Finally, to estimate the sequence order we aim at making a tour that maximize the similarity for neighbour images. This problem may be cast in a Travel Salesman Problem (TSP), where we have to find a maximum cost hamiltonian tour instead of a minimum cost one. The equivalent minimization problem is obtained by considering $\texttt{cost\_matrix} = -\texttt{good\_matches}$.

The solution for TSP problems may rely on exact linear programming strategies, that for problem with size around 30 already takes minutes to be solved, or on heuristics, that have no performance guarantee but are able to provide good solutions in short time. I adapt some C code already developed for another exam, i.e. methods and models for combinatorial optimization, by adding the python binding. It performs a local search with 2opt moves, starting from the initial tour obtained with the nearest neighbour heuristic.

Results: With this procedure we are able to correctly estimate the ordered sequence for the RGB lower and upper loop, separately, either for the *easy* scene and for the *challenging* scene.

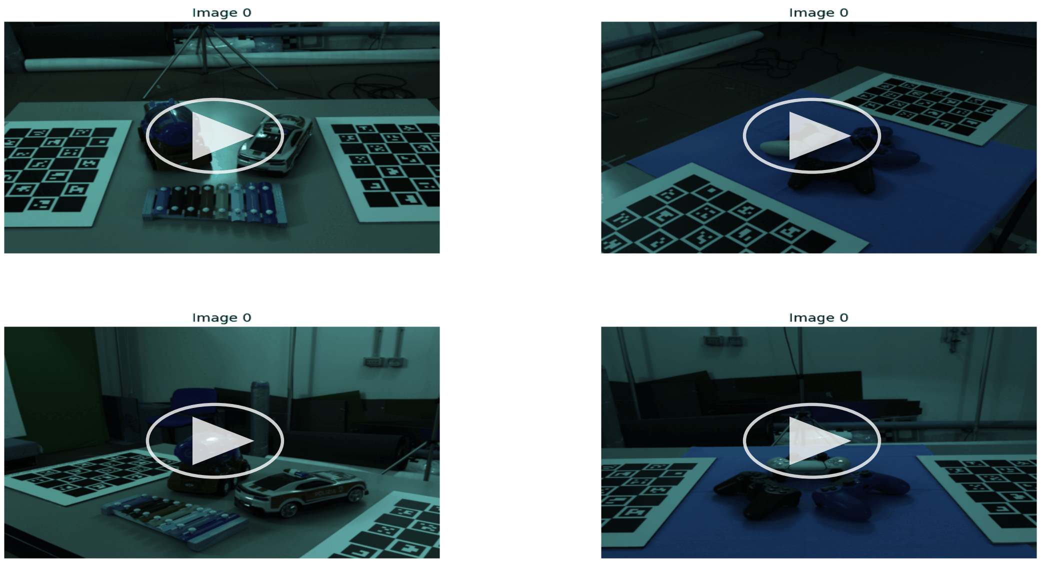

Figure 3: Sorted Estimated Loops. In the left the *easy* scenario, in the right the *challenging* scenario. In the top the *upper* loop, at the bottm the *lower* loop.

3 Near Infrared images

Surprisingly, these steps result in the correct ordered sequence also for the NIR images without any change.

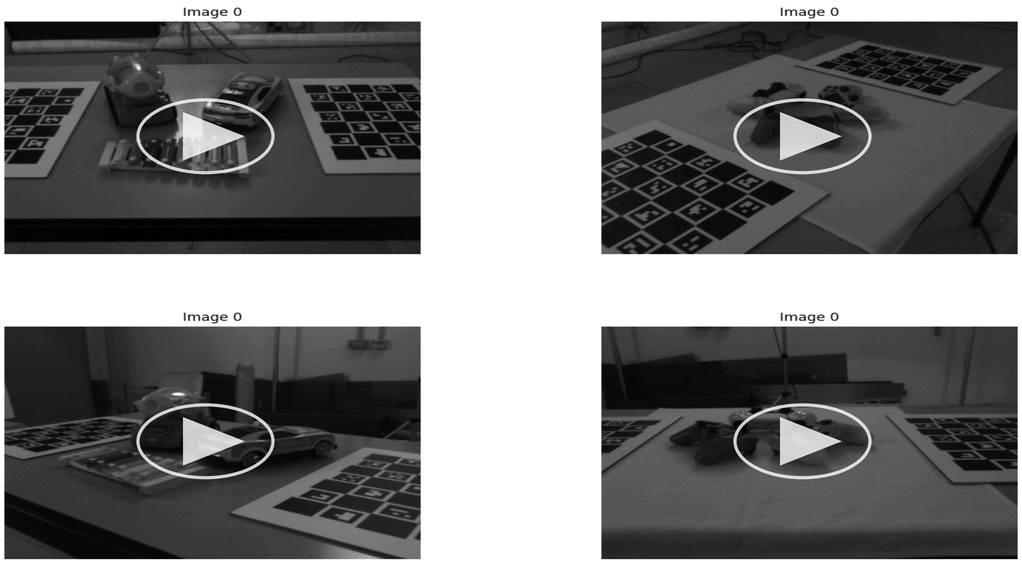

Figure 4: Sorted Estimated Loops for NIR images. In the left the *easy* scenario, in the right the *challenging* scenario. In the top the *upper* loop, at the bottm the *lower* loop.

4 Including the Lower Loop

We now have to estimate the correct tour starting from the images including both the lower and the upper loop. For this, we split the problem in three main steps:

1. Classify the entire set of images in *upper* and *lower* loop,
2. Estimate separately the ordered tours for the two classes (as in the Main Task),
3. Find the closest inter class pair and link the two ordered tours.

4.1 Classification

Feature Extraction For the classification problem, we exploit the pretrained ResNet50 neural network available in PyTorch. We remove the last fully connected layer to access the feature descriptor that the net computes. Indeed, we are not interested in the image classification task the resnet was designed for. Rather, the obtained global descriptor carries useful information for understanding the content of the images.

Cluster Computation: Then, we cluster the images in two classes using the KMeans implemented in openCV with the features above. Even though KMeans is not the finest method because it only looks for spherical clsuster in the feature space, this method is enough to correctly discern the upper and lower loops. This doesn't necessarely means that in the 2048 size resnet global descriptor there are meaningful geometric information of the processed image. Indeed, the amount of wall and floor changes with different azimuth angles and the KMeans is able to discriminate it.

Sequence Order Estimation: The tour estimation on the two separate classes is carried out as before: we isolate the intraclass `good_matches` matrix using the SIFT descriptors, the Flann based Matcher and the same ratio threshold. Then, the tour is obtained with the TSP heuristics applied on `-good_matches`.

Closest Link: The link between the two tours is computed as the pair, one for the loop with class 0 and one for class 1, with the maximum cosine similarity for the resnet feature. The cosine similarity betweeen two 2048 vectors is the cosine in the euclidean space between the two vectors, i.e.,

$$\cos(v_1, v_2) = \frac{v_1 \cdot v_2}{\|v_1\| \|v_2\|}.$$

At most the cosine similarity is one if the vectors are collinear with the same direction and zero if they are orthogonal.

For the *challenging* scenario the link is found when the white joystick controllor is align centrally, that corresponds to the human natural guess.

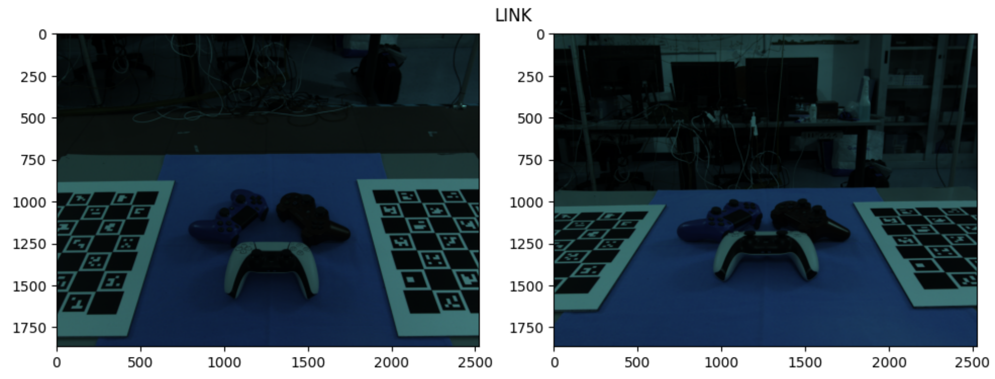

Figure 5: *Challenging* link between upper and lower tour.

On the other hand, for the *easy* scenario the inter class cosine similarity is maximal not when the setting is centrally aligned but when it is slightly tilted. This results in a smoother transition than the human guess.

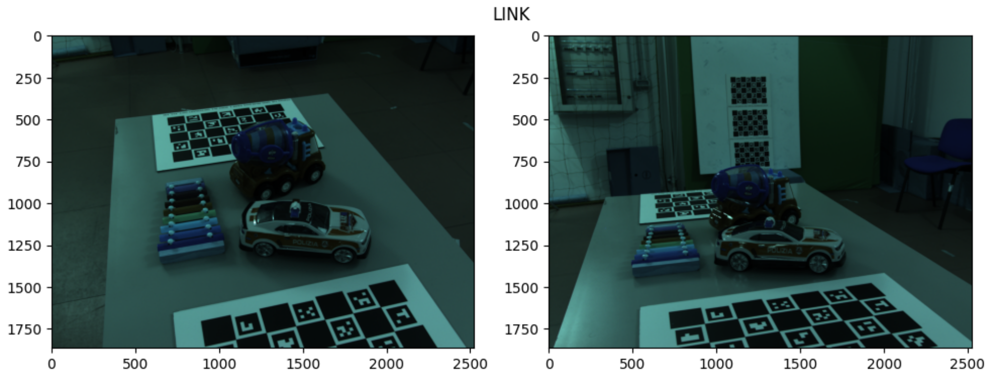

Figure 6: *Easy* link between upper and lower tour.

Total Loop Stitched: The resulting stitched total loops for the challenging and easy scenario are shown in the figure below.

Figure 7: Total Loop Stitched. On the left for the *easy* scenario on the right for the *challenging* scenario.

Remark: We experiment to repeat the same procedure also for the NIR images. However, the classification upper and lower loops fails. Probably, the extracted Resnet global feature descriptors are not informative as the Resnet was trained on RGB images and it is not able to adapt and extrapolate well its capabilities also for NIR images.