In this solution for the image matching task, I used basic keypoint detection algorithms: SIFT, ORB, and BRISK. Before that, I conducted some preliminary research to understand which algorithm is best suited for this task. I didn't have enough time to make comprehensive conclusions about the algorithms, so I selected the three that performed the best (in terms of the number of detected keypoints and search time) and discarded those that performed poorly for this specific task. In the future, it's important to explore the quality evaluation of keypoint detection algorithms in more depth.

The next step was to set parameters for the algorithms, such as the standard deviation for applying the Gaussian filter, the scale, the grid size used to describe keypoints, and the distance ratio between the nearest neighbors during descriptor matching. These parameters need to be fine-tuned for each specific task. I used the recommended values from the article **Optimizing SIFT algorithm parameters for better matching UAV and satellite images** (https://www.researchgate.net/publication/376158891_Optimizing_SIFT_algorithm_parameters_for_better_matching_UAV_and_satellite_images). The adjusted parameters helped detect 15% more keypoints with minimal time cost. However, I only managed to study and fine-tune parameters for the SIFT algorithm and apply similar settings to the others. It would be beneficial to learn more about optimizing parameters for the BRISK (threshold, neighbor search radius, interpolation method) and ORB (number of pyramid levels, scale factor, number of orientations) algorithms to improve their performance for this task.

Regarding the keypoint matching algorithm, I considered both **Brute-Force** and **FLANN** matchers. In the Brute-Force Matcher, for each descriptor from the first image, the closest descriptor in the second image is found by comparing each descriptor with all others, which makes it accurate but very slow. **FLANN Matcher**, on the other hand, uses specialized data structures and algorithms to perform fast approximate nearest-neighbor searches, which, based on my experiments, is about seven times faster than Brute-Force. FLANN adapts its nearest-neighbor search algorithm depending on the input data type. For example, it uses KD-tree or the LSH (Locality Sensitive Hashing) algorithm for high-dimensional data. While FLANN is much faster than Brute-Force, it sacrifices some accuracy since it searches for **approximate** nearest neighbors.

If accuracy is critical, one could try **false match filtering**, such as using **NNDR (Nearest Neighbor Distance Ratio)**, which helps to discard incorrect matches. Additionally, FLANN requires parameter tuning, such as selecting the index type and algorithm, to achieve the best results in specific cases. I believe that the results obtained with the FLANN algorithm are good enough, but it's worth exploring its parameter optimization further.

The best solution for this task would be to train a model on labeled data. Since my input data wasn't labeled, and I didn't have much time, I couldn't build a custom model for the task. However, I'm aware of the **RoMa** method and the **SuperGlue** model, which I later imported and used in my solution. **RoMa** is an approach that improves keypoint matching between images, designed to be robust against changes in scale, rotation, and scene variations (such as lighting or noise). **SuperGlue** uses a neural network to learn keypoint correspondences, allowing the algorithm to improve matching accuracy by learning from various datasets and leveraging contextual and spatial relationships. I don't fully understand how these methods

work yet, so, for example, the **SuperGlue** model would need to be trained on custom datasets.

The results of the matching are difficult to evaluate. I did this intuitively. I know that the **correct match rate (CMR)** is used for evaluation, followed by calculating **Precision** and **Recall** for correct and incorrect matches, but I didn't have time to fully understand how to compute these metrics. I believe that correct matches can be determined intuitively. The length of all the lines connecting the matched keypoints and their angles should be consistent in the best results. Therefore, one could measure the line lengths, select the most common value, and calculate the percentage of pairs whose lengths deviate from the majority. The same can be done for angles. This would provide a rough estimate of matching quality. I am almost certain that better evaluation solutions already exist, but I didn't search for them. In the future, it will be important to evaluate the results properly.