Image Matching Challenge

1. Overview

Problem Definition

The goal is to construct precise 3D maps using sets of images in diverse scenarios and environments. The solution is expected to generate accurate spatial representations, regardless of the source domain—images taken from drones, amidst dense forests, during nighttime, or any of the six problem categories.

Competition webpage

https://www.kaggle.com/competitions/image-matching-challenge-2024/overview

Metric

The metric is **mean Average Accuracy** (mAA) of the registered camera centers.
Detailed information is available in *Section 3. Metric* and at the completion webpage:
https://www.kaggle.com/competitions/image-matching-challenge-2024/overview/evaluation

Data

The dataset consists of several scenes, each scene containing a varying number of images. Each image is represented by camera settings: rotation matrix and translation vector.

train_labels.csv

A list of all images with ground truth information about the camera rotation and translation.

Images

All images belonging to a single scene of the dataset are stored in folders `<scene>/images`.

Strategy for solving the problem

For each scene:

1. Identify pairs of similar images.
2. Detect and match keypoints for all registered pairs of images.
3. Use colmap library to reconstruct the scene's cameras.

2. Training Data

Source code

`01_data.ipynb` – jupyter notebook with overview of the dataset and target.

Images

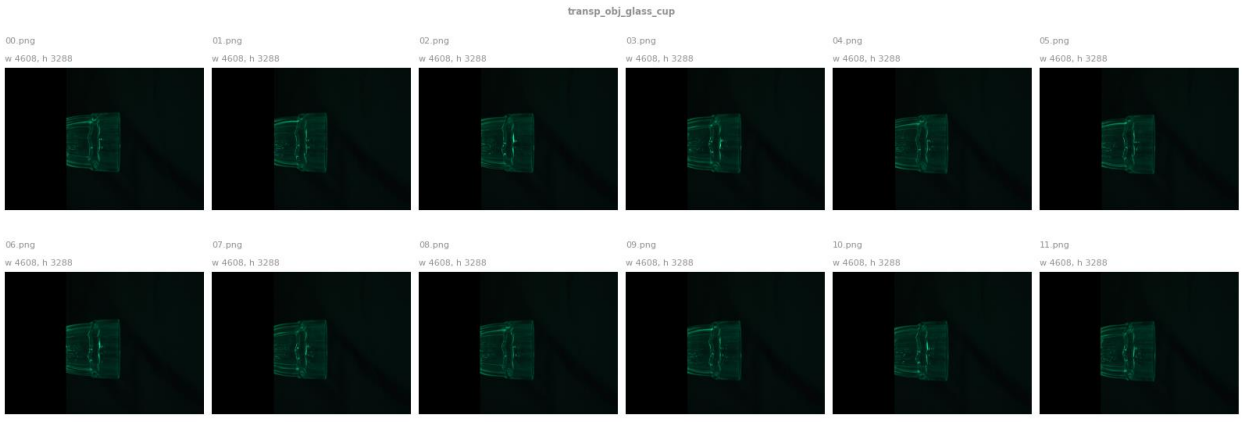
Insights

- Different categories of scenes: transparent objects, modern architecture, historical architecture, nature.
- Varying number of images in scenes:

scene	*number of images*
church	110
dioscuri	70
lizard	711
multi-temporal-temple-baalshamin	68
pond	1117
transp_obj_glass_cup	36
transp_obj_glass_cylinder	36

- Images from the same scene vary in quality, image size, orientation.
- Lighting conditions may vary (day/night).

Examples



dioscuri

img_0311.png
w 1024, h 768

img_0119.png
w 1024, h 768

3dom_fbk_img_1542.png
w 1024, h 683

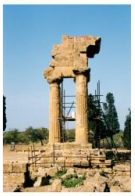
img_0391.png
w 1024, h 768

archive_0001.png
w 1024, h 700

archive_0013.png
w 700, h 1024

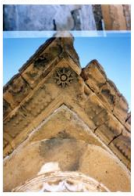
archive_0230.png
w 701, h 1024

archive_0003.png
w 700, h 1024



archive_0010.png
w 1024, h 700

archive_0025.png
w 700, h 1024



archive_0015.png
w 700, h 1024

3dom_fbk_img_1582.png
w 1024, h 683

pond

00992.png
w 576, h 1024

00891.png
w 576, h 1024

00906.png
w 576, h 1024

00242.png
w 1024, h 768

00295.png
w 1024, h 768

00252.png
w 1024, h 768

00884.png
w 576, h 1024

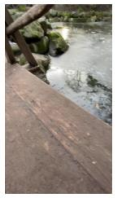
00543.png
w 576, h 1024

00184.png
w 768, h 1024

00379.png
w 576, h 1024

00428.png
w 576, h 1024

00793.png
w 576, h 1024



church

00100.png
w 768, h 1024

00108.png
w 768, h 1024

00047.png
w 768, h 1024

00105.png
w 1024, h 768

00024.png
w 768, h 1024

00111.png
w 768, h 1024

00056.png
w 768, h 1024

00073.png
w 768, h 1024

00003.png
w 768, h 1024

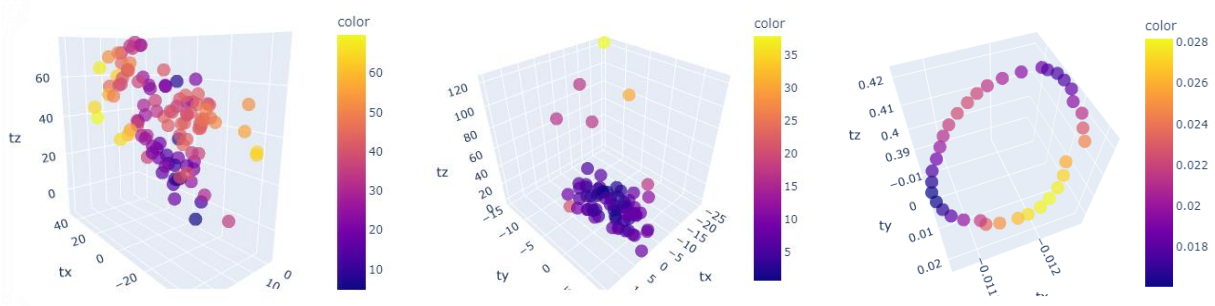
00013.png
w 768, h 1024

00053.png
w 768, h 1024

00044.png
w 768, h 1024

Target

Example of ground truth translation vectors for a scene are shown below. Colored circles represent the end of the vector (with the start of the vector at coordinate (0,0,0)); color denotes the length of the vector:



For an interactive visualization of rotation matrices (visualization of rotation vectors around each axis) refer to the `01_data.ipynb` notebook.

Note: there is some trouble with rendering plotly charts in the GitHub copy of the notebook. The notebook needs to be rerun for rendering 3D charts.

3. Metric

Source code

`imc24.py` – metric source code, copy of the public Kaggle notebook https://www.kaggle.com/code/nartaa/imc24.

`02_metric.ipynb` – a jupyter notebook with exploration of the metric.

Overview

The metric is **mean Average Accuracy** (mAA) of the registered camera centers across all images of all scenes of the dataset.

Camera centers are characterized by a matrix $C = -R^T T$, where R is a 3×3 rotation matrix and T is a translation vector.

For each image of the scene, a camera with computed center C is registered if

$$\|C_g - \mathcal{T}(C)\| < t,$$

where C_g is the ground-truth camera center, t is a given threshold, and $\mathcal{T}$ is the best similarity transformation (scale, rotation and translation) that is able to register onto the ground-truth the highest number of cameras for the scene.

The best similarity transformation $\mathcal{T}$ is obtained by a 2-step process:

1. Using a RANSAC-like approach, exhaustively verify all feasible similarity transformations $\mathcal{T}'$ that can be derived by the Horn's method on triplets of corresponding camera centers (C, C_g);

2. Each transformation $\mathcal{T}'$ is further refined into $\mathcal{T}''$ by registering again the camera centers using the Horn's method, but this time including the previously registered cameras together with the initial triplets. The transformation with the highest number of registered cameras is considered the best.

Insights

For each scene:

- Invariance to translation vectors' scale – only relative measurements matter;
- Invariance to rotation of rotation matrices.

Smallest noise in registered cameras affects the score: for ground truth predictions with additive noise level 1% in either rotation matrices or translation vectors → the average mAA score is ~0.4.

4. Solution

Solution pipeline

Source code

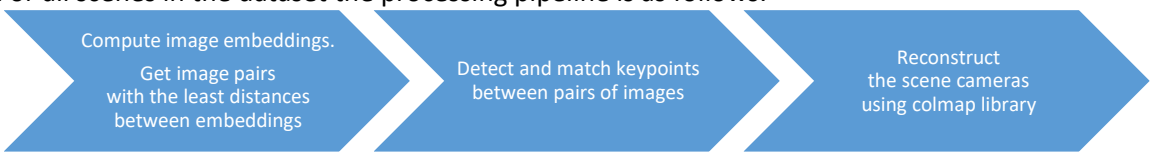
`IMC_solution.py` – solution class for processing all scenes in the dataset and logging the results using Weights&Biases.

`IMC_pipeline.py` – class for the processing pipeline of a single scene.

`IMC_utils.py` – utility functions for the solution.

Overview

For all scenes in the dataset the processing pipeline is as follows:



Each stage is represented in code by a class.

Image embedding and pair matching

Source code

`03_image_pairs.ipynb` – jupyter notebook with examples of matched image pairs.

`image_pairs_matcher.py` – class for computing and ranking distances between image embeddings.

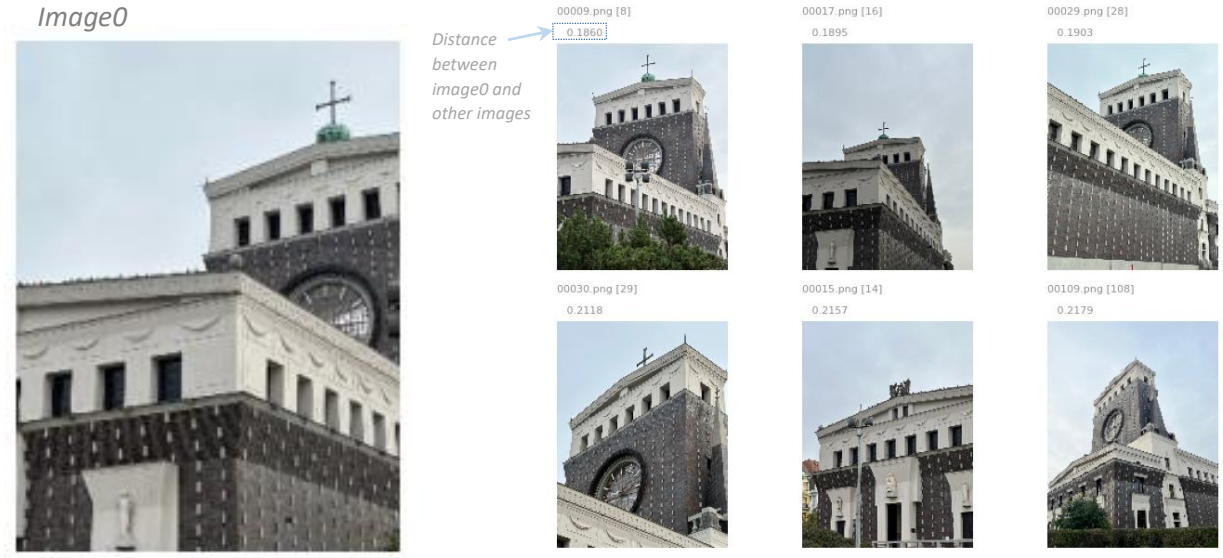
Overview

Images of a scene cannot be compared directly. Pretrained [DINOv2] model is used to obtain normalized image embeddings, then the Euclidean distances between the embeddings are computed for all possible pairs. Image pairs are then sorted according to the computed distance. Pairs with the least distances are then passed to the next stage of the processing pipeline.

Note: Image embeddings are precomputed outside of the solution pipeline and saved in an `h5` file. The pipeline operates under the assumption that the embedding file exists; otherwise, the scene is not processed.

Results

Example of computed distances between images for the `'church'` scene:



Example of computed distances between images for the `'dioscuri'` scene:

Keypoint detection and matching

Source code

`04_display_keypoint_matches.ipynb` – jupyter notebook with examples of matched image keypoints.

`keypoints_matcher_base.py` – base abstract class `KeypointsMatcherBase` for all keypoints matching methods.

`keypoints_LightGlue.py` – wrapper class `Keypoints_LightGlue` for matching keypoints using the LightGlue model.

`keypoints_cv2.py` – wrapper class `Keypoints_cv2` for brute force keypoint matching method from the OpenCV library.

`keypoints_LoFTR.py` – wrapper class `Keypoints_LoFTR` for the LoFTR keypoint matching method.

`parser_matched_keypoints.py` – class for preparing registered keypoint matches for the scene reconstruction step of the pipeline.

KeypointsMatcherBase

`KeypointsMatcherBase` is an abstract class that provides a uniform interface for all keypoint matching methods.

The main method `run` takes in an array of scene image paths and an array of image index pairs from the previous step of the pipeline. For all image pairs the matching is performed using the method `_match_image_pair_with_rotation`.

The abstract method `_match_image_pair` returns an array of matched coordinates of the keypoints of the image pair and is different for all implemented methods.

Image rotation

Keypoint matching methods are sensitive to orientations of the images, which impacts the reconstruction of the scene's cameras. In this project, the following approach is used for dealing with varying orientations of the images:

1. Assume the rotation of the first image `image1` to be 0°.
2. For all four rotations 0°, 90°, 180°, 270° of the second image `image2`, match keypoints of `image1` and `image2` (downsized x4 for speed). If one of the four pairs produces significantly more matches than the other 3, this orientation `R` is considered correct for the `image2`; otherwise assume the correct orientation is 0°.
3. Keypoints are matched for full-sized `image1` and correctly rotated full-sized `image2`. The coordinates of detected image2 keypoints are then inversely rotated to match the initial orientation of `image2`.

One of the scenes in the training dataset contains images with varying orientations. By using correctly rotated `image2` the score for the `'dioscuri'` scene changes from 0.1 to 0.4 without changing other parameters of the pipeline.

Keypoints_LightGlue

`Keypoints_LightGlue` class performs keypoint matching using LightGlue method with a choice of keypoint extraction methods:

1. ALIKED,
2. SuperPoint,
3. DoGHardNet,
4. DISK,
5. SIFT.

Keypoints_LoFTR

`Keypoints_LoFTR` class performs keypoint matching using LoFTR method.

Keypoints_cv2

`Keypoints_cv2` class performs keypoint matching using OpenCV library's `BFMatcher` method with a choice of keypoint extraction methods:

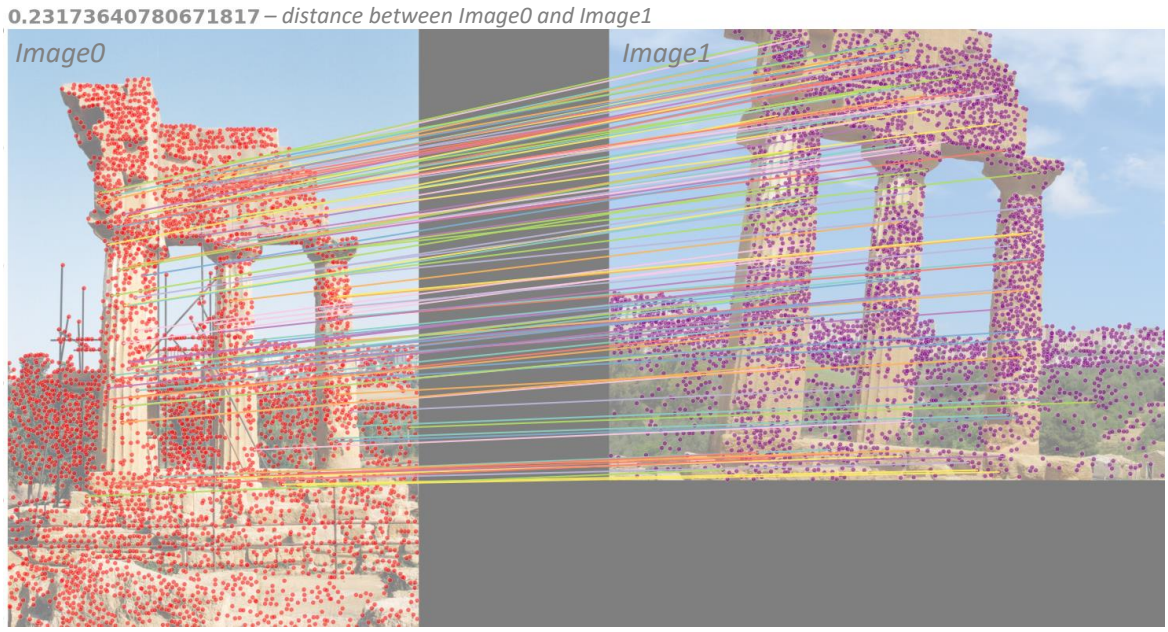
1. ORB,
2. AKAZE.

Results

Overall, LightGlue detects significantly more keypoints than LoFTR and cv2's BFMatcher. Examples of image pairs with matched keypoints are presented below. For a detailed comparison of different methods refer to the section 5. Results.

Example of matched keypoints for the `'church'` scene:

Example of matched keypoints for the `'dioscuri'` scene:



Scene reconstruction

Source code

`scene_reconstructor.py` – a class for reconstructing scene's cameras from matched keypoints.

`database.py`, `h5_to_db.py` – utility functions for importing keypoints into COLMAP-compatible database (provided by the competition organizers).

Overview

`SceneReconstructor` class provides functionality for importing files `'keypoints.h5'` and `'matches.h5'` from the previous step of the pipeline into a COLMAP database and processing the scene using the `incremental_mapping` method from the pycolmap library.

Reproducibility of the reconstruction is ensured by explicitly setting the number of processing threads to 1.

The result of this step is considered to be the reconstruction with the highest number of registered images.

5. Results

Source code

`05_run_solution.ipynb` – a jupyter notebook for running the main solution loop.
`06_results.ipynb` – a jupyter notebook for comparing results of all experiments.

Summary

Scenes 'lizard' and 'pond' were not processed due to the high number of images in these scenes, which is too computationally heavy for the available hardware.

Top scores for each processed scene:

scene	*score*	*pairs threshold*	*pairs min number*	*extractor*	*matcher*	*resize to*	*match threshold*
transp_obj_glass_cylinder	**0.005051**	0.3	5	aliked	LightGlue	1024	0.01
	0.000000	0.1	30	disk	LightGlue	1024	0.01
transp_obj_glass_cup	**0.035354**	0.1	30	LoFTR	LoFTR	512	0.4
	0.025253	0.1	30	disk	LightGlue	1024	0.01
multi-temporal-temple-baalshamin	**0.241026**	0.1	30	disk	LightGlue	768	0.3
	0.235897	0.1	30	disk	LightGlue	1024	0.01
dioscuri	**0.405473**	0.1	30	disk	LightGlue	768	0.3
	0.398010	0.1	30	disk	LightGlue	1024	0.01
church	**0.285047**	0.1	30	disk	LightGlue	1024	0.01
	0.285047	0.1	30	disk	LightGlue	768	0.01

Insights

- The best keypoint matching method is LightGlue with keypoint extraction method DISK.
- All of the methods failed to correctly match keypoints of transparent objects.
- Detecting relative rotation of image pairs is very important for successfully matching keypoints.

6. Approaches that didn't work

Automatic image orientation detection

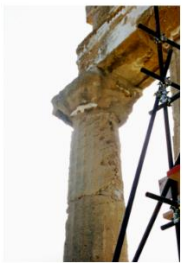
Source code

`07_rotations.ipynb` – a jupyter notebook for automatic rotation detection.

Summary

Random examples of using a pretrained model from the library `check_orientation` show that the detected orientation can be faulty:

✗ *Detected orientation 0° (correct 270°)*



✗ *Detected orientation 90° (correct 0°)*

✓ *Detected orientation 0°*

Appendix I: Keypoint extraction methods

SIFT

SIFT (Scale-Invariant Feature Transform) is a widely used algorithm in computer vision for detecting and describing local features in images. Developed by David Lowe in 1999, SIFT has been extensively used in various applications like object recognition, image stitching, and 3D reconstruction due to its robustness to changes in scale, rotation, and illumination. Here is a detailed explanation of the SIFT keypoint extraction method:

Key Steps in SIFT Keypoint Extraction

1. **Scale-Space Extrema Detection**:
    - The first step is to identify potential keypoints by searching for local extrema (minima and maxima) in the scale-space of the image. This is done by constructing a scale-space representation using Gaussian filters.
    - An image is convolved with Gaussian filters at different scales to produce a series of images called octaves.
    - The Difference of Gaussian (DoG) is calculated between these blurred images, which helps in efficiently detecting stable keypoints in scale-space.
2. **Keypoint Localization**:
    - Once potential keypoints are identified, the next step is to accurately locate them. This involves fitting a 3D quadratic function to the local sample points to determine the precise location and scale of each keypoint.
    - Low contrast keypoints and keypoints located along edges are discarded to improve stability and reduce the number of false matches.
3. **Orientation Assignment**:
    - For each keypoint, one or more orientations are assigned based on local image gradient directions. The gradient magnitude and orientation are calculated around the keypoint within a region.
    - An orientation histogram is created, typically with 36 bins covering 360 degrees. Peaks in this histogram correspond to the dominant orientations of the keypoint.
    - This step ensures that the keypoint descriptor is rotation-invariant.
4. **Keypoint Descriptor**:
    - A descriptor is created for each keypoint that is highly distinctive and robust to variations in illumination and viewpoint.
    - The gradient magnitudes and orientations are sampled around the keypoint in a region (typically 16x16 pixels) and organized into smaller subregions (e.g., 4x4 grids).
    - Each subregion's gradient information is used to create histograms, and these histograms are concatenated to form a vector, which is the keypoint descriptor.
    - The resulting descriptor is normalized to enhance robustness to illumination changes.

Characteristics of SIFT Keypoints

- **Scale Invariance**: By detecting keypoints in different scales and using the scale-space representation, SIFT ensures that the keypoints are invariant to image scaling.

- **Rotation Invariance**: The orientation assignment step makes keypoints invariant to image rotation.
- **Robustness to Illumination Changes**: The use of gradient information and descriptor normalization makes SIFT keypoints robust to changes in lighting conditions.
- **Distinctiveness**: The detailed descriptor allows for accurate matching of keypoints between different images, making SIFT very effective for tasks like object recognition and image matching.

ORB

ORB (Oriented FAST and Rotated BRIEF) is a keypoint detection and description method that is designed to be efficient and fast, while still providing robust results. It was developed to overcome some of the computational drawbacks of SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features) by being less computationally expensive and still achieving good performance.

ORB Keypoint Extraction Method

ORB combines the FAST (Features from Accelerated Segment Test) keypoint detector and the BRIEF (Binary Robust Independent Elementary Features) descriptor with several modifications to enhance performance and provide orientation information. Here's a breakdown of the ORB keypoint extraction method:

1. **Keypoint Detection using FAST**:
    - **FAST Detector**: ORB uses the FAST detector to find keypoints. FAST is very efficient and works by comparing the intensity of a central pixel to a circle of surrounding pixels to quickly determine if a point is a keypoint.
    - **Adaptive Non-Maximal Suppression**: To ensure keypoints are evenly distributed, ORB applies an adaptive non-maximal suppression technique. This step keeps only the most prominent keypoints within a local region.
2. **Orientation Assignment**:
    - **Intensity Centroid**: ORB computes the orientation of keypoints using the intensity centroid. The vector from the center of the keypoint to the center of mass of the intensities is used to determine the dominant orientation.
3. **Descriptor Computation using BRIEF**:
    - **BRIEF Descriptor**: ORB uses the BRIEF descriptor, which is a binary descriptor that compares the intensities of pairs of points around each keypoint. These comparisons create a binary string that serves as the descriptor.
    - **Rotation-Invariant BRIEF**: To make the BRIEF descriptor rotation-invariant, ORB rotates the BRIEF descriptor according to the orientation of the keypoint.
4. **Descriptor Matching**:
    - **Hamming Distance**: For matching keypoints, ORB uses the Hamming distance, which is computationally efficient for binary strings.

Differences between ORB and SIFT

1. **Detection Method**:
    - **ORB**: Uses the FAST detector, which is designed for speed. FAST identifies keypoints by comparing pixel intensities and is much faster than SIFT's Difference of Gaussian (DoG) approach.

- **SIFT**: Uses the Difference of Gaussian (DoG) method to detect keypoints, which involves constructing a scale-space and identifying extrema, a more computationally intensive process.

2. **Descriptor Type**:
   - **ORB**: Utilizes the BRIEF descriptor, which is a binary string based on intensity comparisons. This makes it faster and more memory-efficient.
   - **SIFT**: Uses a 128-dimensional floating-point vector, which captures gradient information around the keypoint, leading to more precise but slower matching.
3. **Rotation Invariance**:
   - **ORB**: Achieves rotation invariance by calculating the orientation using the intensity centroid and rotating the BRIEF descriptor accordingly.
   - **SIFT**: Assigns orientation based on local gradient directions and constructs the descriptor relative to this orientation, inherently providing rotation invariance.
4. **Scale Invariance**:
   - **ORB**: ORB is not inherently scale-invariant because the FAST keypoint detector is not scale-invariant. However, ORB achieves scale invariance by constructing an image pyramid and detecting keypoints at multiple scales. This multi-scale approach allows ORB to handle different scales, but it is not as inherently robust to scale changes as SIFT.
   - **SIFT**: Is inherently scale-invariant due to its scale-space construction using Gaussian blurs.
5. **Computational Efficiency**:
   - **ORB**: Designed to be faster and more efficient, making it suitable for real-time applications.
   - **SIFT**: More computationally expensive due to its detailed keypoint detection and descriptor computation process.
6. **Performance**:
   - **ORB**: Generally performs well in terms of speed and efficiency, but may not be as robust as SIFT in handling complex scenes or very large changes in viewpoint.
   - **SIFT**: Known for its robustness and accuracy, especially in challenging conditions, but is slower and more resource-intensive.
7. **Applications**
   - **ORB:** Preferred in real-time applications like mobile vision, robotics, and augmented reality due to its speed and efficiency.
   - **SIFT:** Used in applications where robustness and accuracy are critical, such as 3D reconstruction, object recognition, and image stitching.

In summary, ORB is a more computationally efficient alternative to SIFT, trading off some of the robustness and precision for speed and efficiency, making it suitable for real-time applications.

AKAZE

AKAZE (Accelerated-KAZE) is a keypoint detection and description method that builds on the KAZE (which stands for K-Adaptive Scale Invariant Feature Transform) algorithm, aiming to provide efficient computation while maintaining high performance. It leverages a nonlinear scale-space and is particularly designed for real-time applications. Here's an explanation of the AKAZE keypoint extraction method and how it differs from SIFT (Scale-Invariant Feature Transform):

AKAZE Keypoint Extraction Method

1. **Nonlinear Scale-Space Construction**:
    - **Nonlinear Diffusion Filtering**: Unlike SIFT, which uses Gaussian blurs to construct a linear scale-space, AKAZE uses nonlinear diffusion filtering. This approach helps preserve edges and fine details better than Gaussian blurring.
    - **Efficient Computation**: AKAZE achieves efficient computation of the nonlinear scale-space using the Fast Explicit Diffusion (FED) solver, making it faster and more suitable for real-time applications.
2. **Feature Detection**:
    - **Multiscale Detector**: AKAZE uses a multiscale Hessian matrix-based detector to identify keypoints. This helps in detecting keypoints across different scales.
    - **Keypoint Localization**: The keypoints are accurately localized by detecting the local extrema in the nonlinear scale-space.
3. **Orientation Assignment**:
    - **Gradient-Based Orientation**: Similar to SIFT, AKAZE assigns an orientation to each keypoint based on the local image gradients. This ensures that the keypoints are rotation-invariant.
4. **Descriptor Computation**:
    - **MLDB (Modified-Local Difference Binary) Descriptor**: AKAZE uses the MLDB descriptor, which is a binary descriptor similar to BRIEF but modified to work efficiently with the nonlinear scale-space. This descriptor is constructed by comparing the intensities of pairs of pixels around the keypoint.
    - **Compact and Efficient**: The binary nature of the descriptor makes it compact and efficient to compute and match.

Differences between AKAZE and SIFT

1. **Scale-Space Construction**:
    - **AKAZE**: Uses a nonlinear scale-space created through nonlinear diffusion filtering. This method preserves image details and edges better than linear approaches.
    - **SIFT**: Uses a linear scale-space constructed using Gaussian blurring, which is more computationally intensive and can smooth out fine details.
2. **Feature Detection**:
    - **AKAZE**: Employs a multiscale Hessian matrix-based detector in the nonlinear scale-space, allowing for better edge preservation and detail retention.
    - **SIFT**: Detects keypoints by identifying extrema in the Difference of Gaussian (DoG) space, which is less efficient in terms of computational cost.
3. **Descriptor Type**:
    - **AKAZE**: Uses the MLDB descriptor, a binary descriptor that is efficient to compute and compare. This makes AKAZE faster and suitable for real-time applications.
    - **SIFT**: Utilizes a 128-dimensional floating-point vector that captures detailed gradient information around the keypoint, providing robust matching at the cost of higher computational complexity.
4. **Efficiency and Speed**:
    - **AKAZE**: Designed for real-time performance with a focus on efficiency, leveraging the Fast Explicit Diffusion (FED) solver for quick scale-space computation.

- **SIFT**: Known for robustness and accuracy but is computationally expensive and slower due to its detailed and resource-intensive processes.

Summary

- **AKAZE**: Aims to provide efficient keypoint detection and description by using a nonlinear scale-space and binary descriptors. It is well-suited for real-time applications due to its speed and efficiency while maintaining good performance.
- **SIFT**: Focuses on robustness and accuracy, using a linear scale-space and detailed descriptors. It is more computationally expensive and slower, making it better for applications where accuracy is critical, and real-time performance is not essential.

AKAZE differs from SIFT primarily in its use of a nonlinear scale-space and binary descriptors, which enhance its efficiency and suitability for real-time applications. SIFT, on the other hand, provides more detailed and robust feature detection and description but at the cost of higher computational resources and slower performance.

ALIKED

ALIKED stands for Adaptive Learned Invariant Keypoints with Efficient Descriptors. It is designed to leverage deep learning techniques to provide robust keypoint detection and description, optimizing both accuracy and computational efficiency.

Key Components of ALIKED

1. **Adaptive Keypoint Detection**:
   - ALIKED uses a neural network to adaptively detect keypoints in an image. The network is trained to identify points that are robust to various transformations such as scaling, rotation, and illumination changes.
2. **Learned Descriptors**:
   - Descriptors are also learned through a neural network, which captures the local image structure around each keypoint. These descriptors are designed to be highly distinctive and robust.
3. **Efficiency**:
   - The model is optimized for efficiency, making it suitable for real-time applications. The neural networks used in ALIKED are designed to run efficiently on modern hardware, leveraging GPU acceleration.
4. **Invariance**:
   - ALIKED aims to provide invariance to common image transformations through its learned features. The adaptability of the neural network allows it to generalize well across different conditions.

Differences between ALIKED and SIFT

1. **Detection Method**:
   - **ALIKED**: Uses a deep learning approach for adaptive keypoint detection, which can dynamically adjust to various conditions and transformations.
   - **SIFT**: Uses the Difference of Gaussian (DoG) approach, which is a more traditional method based on image gradients.
2. **Descriptor Type**:
   - **ALIKED**: Employs learned descriptors from a neural network, capturing complex local structures in a more nuanced manner than traditional methods.
   - **SIFT**: Uses a 128-dimensional floating-point descriptor based on local gradient orientations, which is handcrafted and fixed.
3. **Scale Invariance**:
   - **ALIKED**: Achieves scale invariance through its adaptive learning process, which inherently captures scale variations.
   - **SIFT**: Inherently scale-invariant due to its scale-space construction using Gaussian blurs.
4. **Efficiency and Speed**:
   - **ALIKED**: Optimized for efficiency using modern deep learning frameworks, capable of running in real-time with appropriate hardware.
   - **SIFT**: More computationally intensive and slower due to its detailed keypoint detection and descriptor computation process.
5. **Robustness and Performance**:
   - **ALIKED**: Leverages the power of deep learning to provide high robustness to various image transformations and conditions.
   - **SIFT**: Known for its robustness and accuracy, particularly in handling scale and rotation, but it might not be as adaptable as learned methods.

Summary

- **ALIKED**: A modern, deep learning-based method for keypoint detection and description. It provides adaptive and efficient keypoint extraction suitable for real-time applications, leveraging neural networks for robustness and invariance.
- **SIFT**: A classical method known for its robustness and accuracy, particularly in challenging conditions. It uses traditional techniques for keypoint detection and description but is computationally more demanding.

In conclusion, ALIKED represents a shift towards leveraging deep learning for feature extraction in images, providing significant advantages in terms of adaptability, efficiency, and robustness compared to traditional methods like SIFT.

DISK

The DISK method (Deep Image Structure Keypoints) is another modern approach to keypoint detection and description, utilizing deep learning to achieve robust and efficient performance.

Key Components of DISK

1. **Deep Learning-Based Detection**:
    - DISK employs a convolutional neural network (CNN) trained to detect keypoints that are invariant to transformations such as scaling, rotation, and illumination changes. The training process involves learning to identify features that are consistent across different views and conditions.
2. **Learned Descriptors**:
    - The descriptors in DISK are also derived from a neural network. These descriptors are learned to be highly distinctive, capturing the local structure around each keypoint in a robust manner. The learning process involves matching keypoints across different images, enhancing the descriptor's discriminative power.
3. **Training with Synthetic Data**:
    - DISK often uses synthetic data for training, which can include a wide variety of transformations and noise patterns. This enhances the generalizability of the model to real-world scenarios.
4. **End-to-End Learning**:
    - Both the detection and description processes are integrated into an end-to-end learning framework. This allows the model to jointly optimize the keypoint detection and description, leading to better overall performance.
5. **Real-Time Performance**:
    - The architecture of DISK is designed to be efficient, enabling real-time performance on modern hardware with GPU acceleration. This makes it suitable for applications requiring quick and reliable feature extraction.

Comparison between DISK and ALIKED

1. **Detection Method**:
    - **DISK**: Uses a convolutional neural network specifically trained to detect keypoints invariant to various transformations. The training process focuses on robustness and consistency across different views.
    - **ALIKED**: Similarly employs a deep learning approach for adaptive keypoint detection, using neural networks to dynamically adjust to various conditions and transformations.
2. **Descriptor Type**:
    - **DISK**: Utilizes learned descriptors from a neural network, optimized during the same training process as the keypoint detection. These descriptors are designed to be highly distinctive and robust.
    - **ALIKED**: Also uses learned descriptors, capturing complex local structures in a nuanced manner. The descriptors are designed to be efficient and adaptable.
3. **Training Data**:
    - **DISK**: Often involves training with synthetic data, allowing the model to learn a wide variety of transformations and noise patterns.
    - **ALIKED**: The training process details for ALIKED might vary, but it similarly focuses on learning invariances to transformations through deep learning.

4. **Architecture and Efficiency**:

- **DISK**: The model architecture is streamlined for efficiency, supporting real-time performance with GPU acceleration. The end-to-end learning framework ensures that both detection and description are optimized together.
- **ALIKED**: Also optimized for efficiency, making use of modern deep learning frameworks to achieve real-time performance. The adaptive learning process ensures robustness to various transformations.

5. **Generalizability and Robustness**:
   - **DISK**: Emphasizes robustness and generalizability through its use of synthetic training data and end-to-end learning. This approach ensures that the model can handle a wide range of real-world scenarios effectively.
   - **ALIKED**: Focuses on adaptive learning to achieve robustness and invariance, leveraging neural networks to generalize well across different conditions.

Summary

Both DISK and ALIKED represent advanced approaches in the field of computer vision, leveraging deep learning to improve upon traditional methods like SIFT in terms of robustness, efficiency, and adaptability. The key differences lie in their specific training processes, data used for training, and the architectural choices made to optimize performance.

SuperPoint

SuperPoint is a self-supervised deep learning approach for keypoint detection and description. It combines a convolutional neural network (CNN) for detecting keypoints and computing descriptors in a unified architecture.

Key Components of SuperPoint

1. **Self-Supervised Learning**:
   - SuperPoint is trained in a self-supervised manner using synthetic data. This involves generating training pairs where one image is a perturbed version of another. The network learns to detect consistent keypoints and descriptors between these pairs.
2. **Architecture**:
   - **Shared Encoder**: The SuperPoint model uses a shared encoder (a CNN) that processes the input image to produce a dense feature map.
   - **Detector Head**: A detector head processes the feature map to predict keypoint locations. This is done by creating a probability map where each pixel represents the likelihood of being a keypoint.
   - **Descriptor Head**: A descriptor head computes local descriptors for each keypoint by sampling the feature map at the detected keypoints. The descriptors are typically 256-dimensional vectors.
3. **Interest Point Detection**:
   - SuperPoint uses a heatmap to detect interest points. The heatmap is produced by the detector head and peaks in the heatmap correspond to keypoint locations.

4. **Descriptor Extraction**:

    - Descriptors are extracted from the feature map at the keypoint locations. These descriptors capture the local image structure around each keypoint and are designed to be robust to various transformations.
5. **Training Process**:
    - **Homographic Adaptation**: The training involves generating synthetic homographies (planar transformations) to create multiple views of the same scene. The network is trained to produce consistent keypoints and descriptors across these views.
    - **Self-Improvement**: An iterative process is used where the initial weak labels (pseudo ground-truth) are improved over time through self-supervised learning.

Differences between SuperPoint and ALIKED

1. **Learning Approach**:
    - **SuperPoint**: Trained in a self-supervised manner using synthetic homographies. This process creates training data by applying random transformations to images and training the network to produce consistent keypoints and descriptors across these transformations.
    - **ALIKED**: Typically employs supervised or semi-supervised learning approaches where the network is trained with labeled data to learn invariant keypoints and descriptors.
2. **Architecture**:
    - **SuperPoint**: Utilizes a shared encoder with separate detector and descriptor heads. The encoder extracts dense feature maps, while the heads detect keypoints and extract descriptors.
    - **ALIKED**: The specific architectural details might vary, but it similarly uses a neural network to detect keypoints and compute descriptors. It emphasizes adaptability and invariance through deep learning.
3. **Training Data**:
    - **SuperPoint**: Relies heavily on synthetic data generated through homographic adaptation. This self-supervised approach allows the model to learn without extensive labeled datasets.
    - **ALIKED**: Might use a combination of synthetic and real data, with an emphasis on supervised learning to capture invariances and generalize well across different conditions.
4. **Efficiency and Real-Time Performance**:
    - **SuperPoint**: Designed for efficiency with a streamlined architecture. It performs well in real-time applications, leveraging the power of CNNs for fast keypoint detection and description.
    - **ALIKED**: Also optimized for real-time performance, focusing on efficient computation of keypoints and descriptors through modern deep learning techniques.
5. **Invariance and Robustness**:
    - **SuperPoint**: Achieves robustness through self-supervised learning and homographic adaptation, ensuring that keypoints and descriptors are consistent across various transformations.
    - **ALIKED**: Aims for robustness and adaptability through supervised learning, capturing complex invariances in the data.

Summary

Both SuperPoint and ALIKED are modern approaches leveraging deep learning for keypoint extraction, but they differ in their training methodologies and specific architectural designs. SuperPoint emphasizes self-supervised learning with synthetic data, while ALIKED uses supervised learning to achieve robustness and adaptability.

DoGHardNet

DoGHardNet is a method that combines the Difference of Gaussian (DoG) keypoint detection with HardNet descriptors. This approach leverages traditional and deep learning techniques to create a robust and efficient keypoint detection and description system.

Key Components of DoGHardNet

1. **DoG Keypoint Detection**:
    - **Difference of Gaussian (DoG)**: This method involves convolving the image with Gaussian filters at different scales and subtracting the resulting images to obtain the DoG images. Keypoints are detected as local extrema in these DoG images. This step is similar to the keypoint detection method used in SIFT.
2. **Keypoint Refinement**:
    - **Localization and Orientation**: After detecting keypoints, their positions are refined to sub-pixel accuracy, and their orientations are assigned based on the local image gradients. This ensures that the keypoints are invariant to rotation and scale changes.
3. **HardNet Descriptors**:
    - **Deep Learning-Based Descriptor**: HardNet is a deep learning-based descriptor that provides robust and distinctive local descriptors for the detected keypoints. It is trained using a triplet margin loss to ensure that the descriptors are discriminative and can effectively distinguish between different keypoints.
    - **Training**: HardNet is trained on a large dataset of image patches, learning to produce descriptors that are invariant to common transformations such as rotation, scaling, and viewpoint changes.

Differences between DoGHardNet and ALIKED

1. **Detection Method**:
    - **DoGHardNet**: Uses the traditional Difference of Gaussian (DoG) method for keypoint detection. This method is well-established and known for its robustness and accuracy in detecting scale-invariant keypoints.
    - **ALIKED**: Uses a deep learning-based approach for adaptive keypoint detection. The neural network dynamically adjusts to various conditions and transformations, providing robust keypoint detection.

2. **Descriptor Type**:

- **DoGHardNet**: Employs HardNet, a deep learning-based descriptor that is trained to be highly discriminative. HardNet focuses on creating descriptors that are robust to various transformations.
- **ALIKED**: Uses learned descriptors from a neural network, capturing complex local structures in a nuanced manner. The descriptors are designed to be efficient and adaptable, optimized through supervised or semi-supervised learning.

3. **Training Process**:
   - **DoGHardNet**: Involves training the HardNet descriptors using a triplet margin loss on a large dataset of image patches. The keypoint detection step is based on traditional methods and does not require learning.
   - **ALIKED**: The entire process, including keypoint detection and descriptor extraction, is trained using supervised or semi-supervised learning techniques, focusing on adapting to various conditions.
4. **Efficiency and Real-Time Performance**:
   - **DoGHardNet**: Efficient due to the use of the DoG method for keypoint detection, which is less computationally intensive than some deep learning approaches. HardNet descriptors, while deep learning-based, are designed to be computationally efficient.
   - **ALIKED**: Optimized for real-time performance using modern deep learning frameworks, with both detection and description designed to be efficient.

Summary

In conclusion, DoGHardNet and ALIKED represent different approaches to keypoint extraction and description. DoGHardNet combines the strengths of traditional and modern methods, while ALIKED fully leverages deep learning for robust and adaptive feature extraction. The choice between these methods depends on the specific requirements of the application, including the need for real-time performance, robustness to transformations, and computational efficiency.

Appendix II: Keypoint matching methods

LightGlue

LightGlue is a modern method for keypoint matching that leverages deep learning techniques to provide accurate and efficient matching between keypoints in different images. LightGlue is designed to work with existing keypoint detectors and descriptors, enhancing the matching process with advanced neural network techniques.

Key Components of LightGlue

1. **Deep Learning-Based Matching**:
   - LightGlue utilizes a neural network to perform the matching of keypoints. This network is trained to learn correspondences between keypoints in different images, leveraging the rich contextual information present in the feature descriptors.

2. **Integration with Keypoint Detectors and Descriptors**:

- LightGlue is designed to work with various keypoint detectors and descriptors, such as SIFT, ORB, or newer deep learning-based methods like SuperPoint or DoGHardNet. It enhances the matching process regardless of the initial feature extraction method.

3. **Contextual Matching**:
   - The neural network in LightGlue takes into account the spatial and contextual information of keypoints, improving the robustness and accuracy of the matching. This means it doesn't just match keypoints based on their local descriptors but also considers their relative positions and the global structure of the image.

How LightGlue Works

1. **Feature Extraction**:
   - First, keypoints and their descriptors are extracted from the images using any chosen method, such as SIFT, ORB, SuperPoint, or DoGHardNet. Each keypoint is represented by its location and a feature descriptor.
2. **Neural Network Matching**:
   - The extracted keypoints and descriptors are fed into LightGlue's neural network. The network processes these features, learning to identify corresponding keypoints between the images.
   - The network considers the descriptors and the spatial arrangement of the keypoints, using this information to produce a set of likely matches.
3. **Training the Model**:
   - LightGlue is trained on large datasets of image pairs with known correspondences. During training, the network learns to optimize the matching process, minimizing the distance between matched keypoints in the training data.
   - The training process involves a loss function that penalizes incorrect matches and rewards correct ones, effectively teaching the network to distinguish between true correspondences and false matches.
4. **Output Matches**:
   - The network outputs a set of matched keypoints, providing not just the matched pairs but also a confidence score for each match. This confidence score helps in filtering out less reliable matches.

Differences from Traditional Matching Methods

1. **Deep Learning Integration**:
   - **Traditional Methods**: Rely on handcrafted algorithms for matching, such as nearest-neighbor search in descriptor space. These methods typically use brute-force matching or approximate nearest neighbor algorithms, which can be computationally intensive and less accurate.
   - **LightGlue**: Uses a neural network to learn the matching process, leveraging deep learning to improve accuracy and robustness. The network can capture complex patterns and relationships between keypoints that traditional methods might miss.
2. **Contextual Information**:
   - **Traditional Methods**: Typically match keypoints based on local descriptors alone, without considering the broader context of the keypoints' spatial arrangement.
   - **LightGlue**: Incorporates contextual information, considering both the local descriptors and the spatial relationships between keypoints, leading to more accurate matches.

3. **Efficiency**:
    - **Traditional Methods**: Can be computationally intensive, especially for large sets of keypoints, as they often involve pairwise comparisons of all keypoints.
    - **LightGlue**: Designed to be efficient by leveraging the parallel processing capabilities of neural networks, potentially reducing the computational load and speeding up the matching process.

Summary

LightGlue represents a significant advancement in the field of keypoint matching, offering a modern solution that leverages the strengths of deep learning to address the limitations of traditional matching methods.

LoFTR

LoFTR (Local Feature TRansformer) is a deep learning-based method designed for direct feature matching without requiring explicit keypoint detection. This approach uses transformers to establish dense correspondences between images, providing a robust and efficient alternative to traditional keypoint matching techniques.

Key Components of LoFTR

1. **Dense Feature Extraction**:
    - Instead of detecting sparse keypoints, LoFTR extracts dense features from images using convolutional neural networks (CNNs). These features capture rich local information at every pixel.
2. **Transformer-Based Matching**:
    - LoFTR uses a transformer architecture to perform feature matching. Transformers excel at capturing long-range dependencies and contextual information, making them well-suited for matching tasks.
    - The transformer architecture processes the dense feature maps to compute a correlation matrix, which helps in establishing initial correspondences between image regions.
3. **Coarse-to-Fine Matching**:
    - LoFTR employs a coarse-to-fine approach to improve efficiency and accuracy. Initially, coarse matches are established at a lower resolution, capturing the global context. Then, these matches are refined at a higher resolution to improve precision.

How LoFTR Works

1. **Feature Extraction**:
    - Dense feature maps are extracted from the input images using a CNN. These feature maps represent the local image structure at every pixel.
2. **Coarse Matching with Transformers**:
    - The feature maps are downsampled to a lower resolution.
    - A transformer model processes the coarse feature maps from both images. It computes a correlation matrix to establish initial matches by identifying high-correspondence regions.

3. **Fine Matching with Transformers**:
    - The initial coarse matches are upsampled to a higher resolution.
    - A second stage of the transformer processes the fine feature maps to refine the matches, improving their accuracy.
4. **Match Filtering**:
    - The final matches are filtered based on confidence scores provided by the transformer. This step ensures that only the most reliable matches are retained.

Summary

LoFTR represents a significant advancement in keypoint matching, leveraging modern deep learning techniques to overcome the limitations of traditional methods, providing a powerful tool for computer vision applications.

OpenCV brute force matching algorithm

The brute force keypoint matching method in OpenCV is a straightforward and widely-used technique for matching keypoints between images. It involves comparing each descriptor from one set of keypoints with every descriptor in another set to find the best matches. Here's a step-by-step explanation of how this method works:

Key Components of Brute Force Matching

1. **Matching Descriptors**:
    - The brute force matcher compares each descriptor from the first image with every descriptor from the second image, computing a distance between them. Common distance metrics include:
        - **Euclidean Distance** for SIFT and other float-based descriptors.
        - **Hamming Distance** for binary descriptors like ORB.
2. **Finding the Best Matches**:
    - For each descriptor in the first image, the brute force matcher finds the descriptor in the second image with the smallest distance. This forms a pair of matched keypoints.
3. **Filtering Matches**:
    - To improve the quality of matches, filtering techniques can be applied:
        - **Ratio Test (Lowe's Ratio Test)**: Compares the closest match to the second closest match and only accepts the match if the ratio is below a certain threshold, reducing the number of ambiguous matches.
        - **Cross-Check**: Ensures that a match is mutual, i.e., the best match from the first set to the second set should also be the best match from the second set to the first set.

Summary

Advantages:

- **Simplicity**: Easy to understand and implement.
- **Flexibility**: Works with different types of descriptors and distance metrics.

Disadvantages:

- **Computationally Intensive**: Comparing each descriptor with every other descriptor is computationally expensive, especially for large sets of keypoints.
- **Not Scalable**: Less efficient for real-time applications or large-scale image matching tasks.

The brute force matching method in OpenCV provides a simple yet powerful way to match keypoints between images, making it a valuable tool for various computer vision applications, despite its computational cost.

Appendix III: RANSAC

RANSAC (Random Sample Consensus) is a robust algorithm used to estimate the parameters of a mathematical model from a dataset that contains outliers. It is widely used in computer vision and image processing tasks, such as fitting a line to a set of points, estimating the fundamental matrix in stereo vision, or finding the homography matrix between images.

Workflow of RANSAC

Input:

- A set of observed data points.
- A model that can be fitted to the data.
- A method to estimate the parameters of the model using a subset of the data points.
- A threshold value to determine when a data point fits the model (i.e., an inlier).

Algorithm Steps:

Initialize:

- Set the number of iterations N.
- Set the inlier threshold t.
- Set the minimum number of data points required to fit the model k.
- Initialize the best model parameters and the highest number of inliers found to zero.

Iterate N times:

- Randomly select a subset of k data points from the input dataset.
- Estimate the model parameters using the selected subset.
- Determine which points in the entire dataset are inliers by checking if they fit the estimated model within the threshold t.
- If the number of inliers is greater than the highest number found so far, update the best model parameters and the highest number of inliers.

Output:

- The best model parameters found after NNN iterations.
- The set of inliers associated with the best model.

Summary

Advantages:

- **Robust to Outliers**: Can handle datasets with a significant percentage of outliers.
- **Simple and Effective**: Easy to implement and often provides good results in practice.

Disadvantages:

- **Computationally Expensive**: May require many iterations to find a good model, especially with large datasets or high-dimensional models.
- **Parameter Sensitivity**: Performance depends on the choice of parameters like the number of iterations, the inlier threshold, and the minimum number of points to fit the model.
- **Not Deterministic**: Results can vary between runs due to its random nature.

RANSAC is a powerful algorithm for fitting models to data that contains outliers. By iteratively sampling subsets of the data, estimating model parameters, and refining the model using inliers, RANSAC can robustly estimate the correct model despite the presence of noise and outliers. Its simplicity and effectiveness make it a popular choice in various computer vision and image processing applications.

Appendix IV: Horn's Method

Horn's method, also known as the Horn's relative orientation method, is a widely used algorithm for solving the problem of finding the relative orientation between two sets of 3D points. This method is particularly useful in computer vision and robotics for tasks like pose estimation, where one needs to determine the rotation and translation that align one set of 3D points with another.

Problem Statement

Given two sets of corresponding 3D points $\{A_i\}$ and $\{B_i\}$, where $i = 1,2,\ldots,N$, the goal is to find the rotation matrix R and the translation vector t that minimize the sum of squared distances between the transformed points in one set and their corresponding points in the other set.

Steps in Horn's Method

1. **Compute Centroids**: Calculate the centroids (mean positions) of the two point sets $\{A_i\}$ and $\{B_i\}$:
$$\bar{A} = \frac{1}{N}\sum_{i=1}^{N} A_i, \quad \bar{B} = \frac{1}{N}\sum_{i=1}^{N} B_i$$
2. **Center the Points**: Subtract the centroids from the respective point sets to center them at the origin:
$$\hat{A}_i = A_i - \bar{A}, \qquad \hat{B}_i = B_i - \bar{B}$$
3. **Compute the Cross-Covariance Matrix**: Calculate the cross-covariance matrix H from the centered points:
$$H = \sum_{i=1}^{N} \hat{A}_i \hat{B}_i^T$$
4. **Singular Value Decomposition (SVD)**: Perform SVD on the matrix H:
$$H = U\Sigma V^T$$
5. **Compute Rotation Matrix**: The optimal rotation matrix R is given by:
$$R = VU^T$$
If $det(R) < 0$, adjust V by changing the sign of the third column to ensure a proper rotation matrix (i.e., to prevent reflection): $V[:,2] *= -1$

6. **Compute Translation Vector**: The translation vector t is given by:
$$t = \bar{B} - R\bar{A}$$

Advantages of Horn's Method

- **Accuracy**: Provides an optimal solution in the least-squares sense.
- **Efficiency**: Computationally efficient with a clear, structured approach.
- **Robustness**: Handles noisy data well if outliers are minimal.

Applications

- **Pose Estimation**: Determining the position and orientation of an object in 3D space.
- **Robot Navigation**: Aligning point clouds from different scans for mapping and localization.
- **3D Reconstruction**: Aligning different views of a scene to create a coherent 3D model.

Summary

Horn's method is a robust and efficient algorithm for finding the relative orientation between two sets of 3D points. It leverages centroids, cross-covariance matrices, and singular value decomposition to compute the optimal rotation and translation that align the point sets in a least-squares sense. This method is fundamental in various fields, including computer vision, robotics, and 3D modeling.