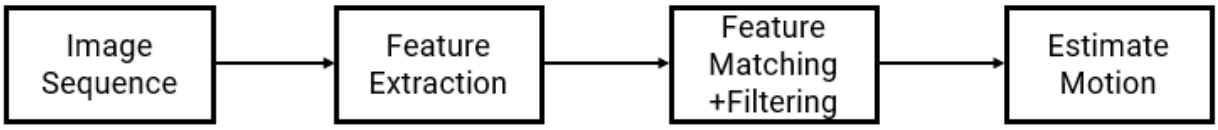
CS585 Challenge Report

Zhengqi Dong

The entire visual odometry (VO) pipeline can be roughly abstracted to five steps: preparing image sequence, feature extraction, feature matching, filtering good features, and pose estimation.



Step 1: Image Sequence

At the first beginning, we have an initializer will load the training images, calibration parameter(e.g., focal length, principal points), ground truth sequences(the actual camera pose).

In this step, it's helpful to convert the image data type into tensor, which will be more efficient to perform computation later. Also, if the data set is too big, we can use data generator instead of loading the whole dataset into memory.

Step 2: Feature Extraction

There are many features extractor available, e.g., SIFT, SURF, BRIEF, ORB, BRISK, and etc. In this challenge, only two feature extractors have tested: ORB and SIFT. ORB is faster, but less accurate and robust than SIFT. However, since the OpenCV version installed in Gradescope doesn't support SIFT, so ORB is used through all the experiments.

Note: The default number of features will be extracted is 500, but it's often not enough. So, we can ask OpenCV for more features point with parameter nfeatures, e.g., cv2.ORB_create(nfeatures=3000); Second, we can lower distance threshold during feature filtering, and this will give us more feature points and thus increase the performance in a great scale.

Step3: Feature Matching + Filtering

Two feature machers has tested: BFMatcher(brute-force macher), and FLANN based matcher.

In term of final quality of matches, there is not much difference between two, but FLANN optimized performance by using nearest neighbor search and can be faster in larger datasets and high dimensional features, so FLANN is used throughout the experiment. (Reference: OpenCV, Feature Matching)

Besides, there are some other operations has performed to improve the performance after FLANN macher: 1) use knnMatch to extract the top 2 matches and perform Lowe's distance ratio test, which was proposed in Prof. Lowe's 2004 SIFT paper. 2) Sort the distance in increasing order.

Note: Both techniques showed certain amounts of improvement.

Step4: Estimate Motion:

Estimating fundamental matrix: Eight-point algorithm is used to estimate the fundamental matrix, and RANSAC is used to select the fundamental matrix that has the maximum number of inliers that is consistent with F within certain amount of error_threshold. A pseudo algorithm is shown below (credit to CMSC426 Computer Vision, Structure from Motion).

```
n=0;
for i = 1:M do
    // Choose 8 correspondences, x̂1 and x̂2 randomly
    F = EstimateFundmentalMatrix(x̂1, x̂2);
    S = ∅;
    for j = 1:N do
        if | x2j^T F x1j | < ε then
            S = S ∪ {j}
        end
    end
    if n <| S | then
        n =| S |;
        S_in = S
    end
end
```

Note: An inlier_ratio and max_no_change_time is used to speed up the algorithm. The inlier_ratio defind the number of inlier data points that we think is enough to define a good model, and max_no_change_time will stop the RANSAC algorithm if there is no any improvement in error after running max_no_change_time times.

PoseRecovery is implemented by first decomposing the essential matrix to get the rotation matrix and translation vector. By following the theory described here (CMSC426 Computer Vision, Structure from Motion), we can see that there are four possible pose configurations. Based on the Cheirality condition defined in this paper, we need to remove the disambiguity by selecting the configuration that provides the maximum number of points that has positive value along Z direction for all those 3D points in the camera coordinate system, and those 3D points can be constructed by triangulating two set of camera points with cv2.triangulatePoints function.

The result of this implementation shown an significant improvement on the native OpenCV version, cv2.recoverPose().