Yet Another Simple and Stupid Visual SLAM

Xiaoran Jiao

Department of Computer Science and Technology, Zhejiang University

Motivation

Being a hot research topic in the last two decades, SLAM techniques build a map of an unknown environment and localize the camera in the map with a strong focus on real-time operation.
As for me, I've been interested in visual SLAM for a long time and always dreamed of implementing a SLAM system from scratch to practice my programming skills.

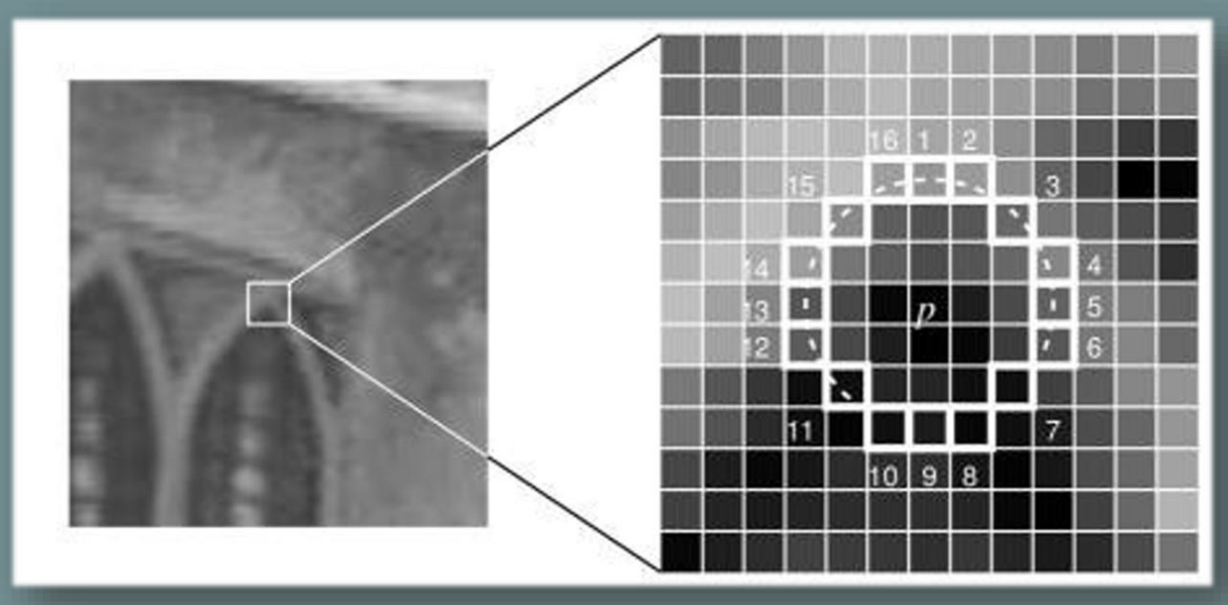
Contribution

- A real-time SLAM framework for Monocular Visual Systems, including initialization, feature-based visual odometry and local pose graph optimization
- Implemented with C++ and fully integrated with ROS
- Split the front-end, the back-end, and the viewer into different threads on CPU

Front-End: Data Association

In the Front-End, we establish **direct data association** via **ORB** features, which are robust to rotation and scale and present a good invariance to camera auto-gain and auto-exposure, and illumination changes. Moreover, they are fast to extract and match allowing for real-time operation required by the SLAM system.

ORB is basically the combination of two algorithms involved FAST and BRIEF. Firstly, **FAST** quickly selects the keypoints by comparing the intensity of distinctive regions. Then **BRIEF** takes all keypoints found by the FAST and converts it into a binary feature descriptor, which is a binary string that each bit is determined by selecting a random pair of pixels and comparing the intensity value.

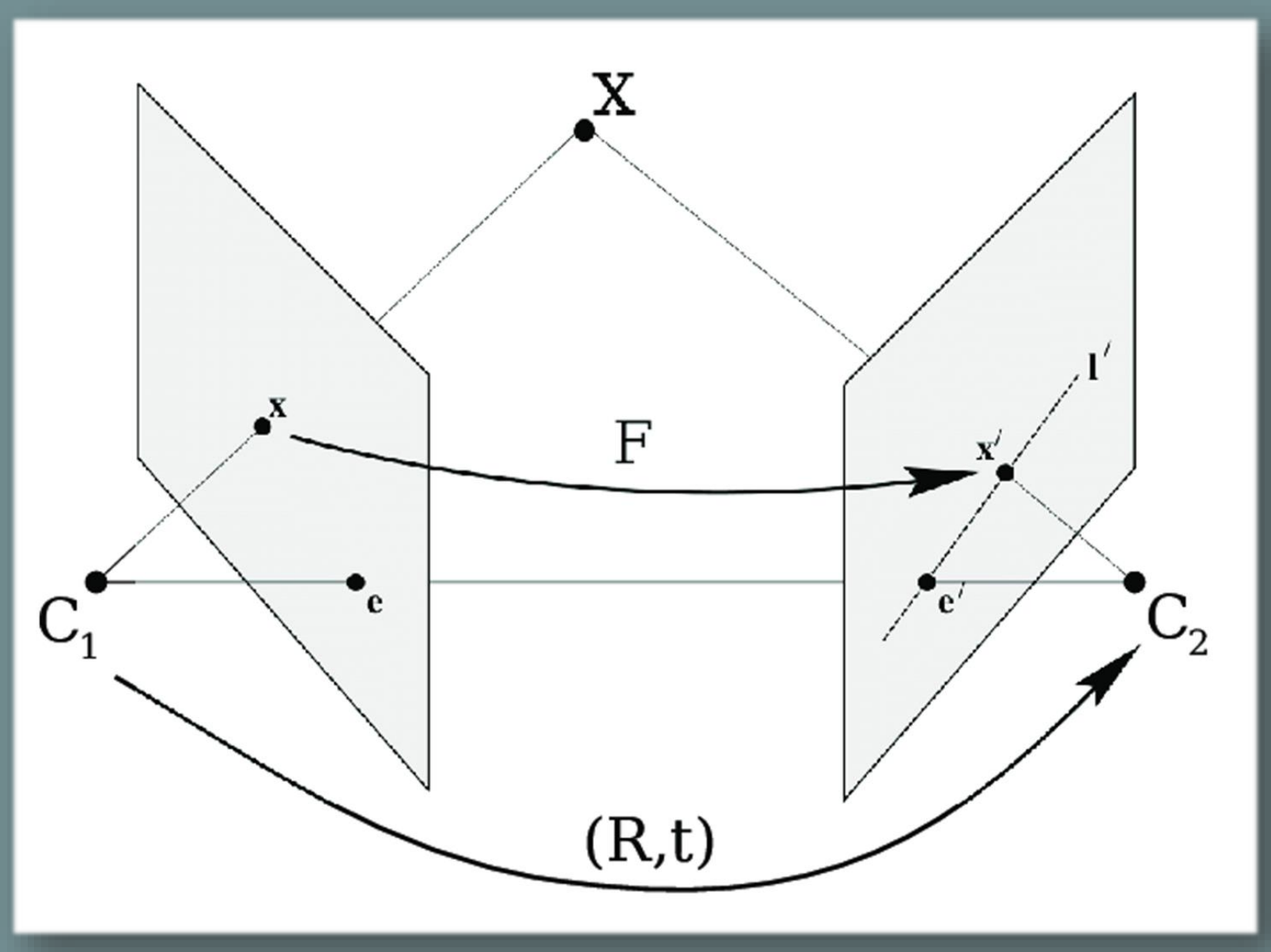


Taking **Hamming distance** as metrics, **Brute Force** Matcher is used for matching the features of one image with another.

Front-End: Geometric Verification

After obtaining at least 8 matches between 2 frames, **fundamental matrix** can be robustly estimated via **RANSAC** algorithm, which is a relationship between any two images of the same scene that constrains where the projection of points from the scene can occur in both images. Then we can **decompose the fundamental matrix to recover pose**, that is, 3D translation and rotation.

Although there inevitably exists mismatches, such outliers can be rejected by the above RANSAC algorithm. This step is also known as geometric verification since it rejects matches that do not satisfy the epipolar constraint.



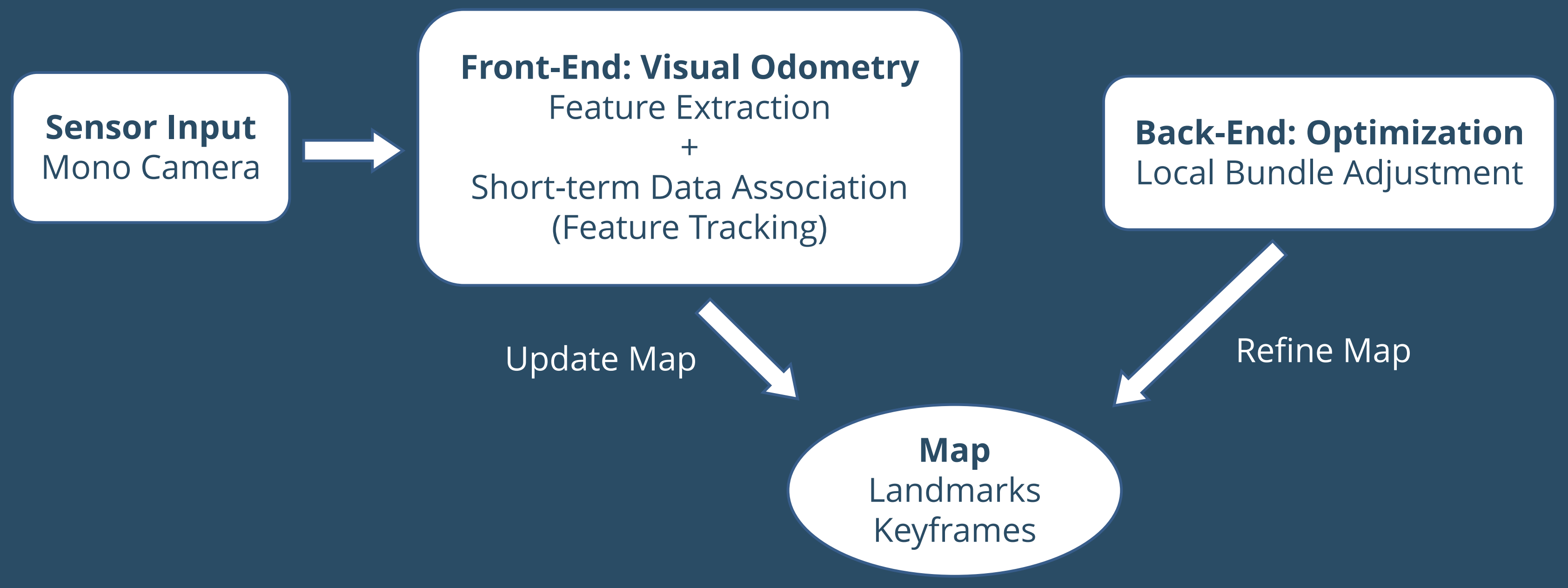
Front-End: 3D-2D PnP

Getting 2D-2D matches, with correspondences between 2D features in last frame and 3D mappoints in map, we can determine the 3D-2D correspondences in current frame. By solving PnP, we can obtain the 6-DOF camera pose which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world. To be more specific, Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image.

Front-End: Triangulation

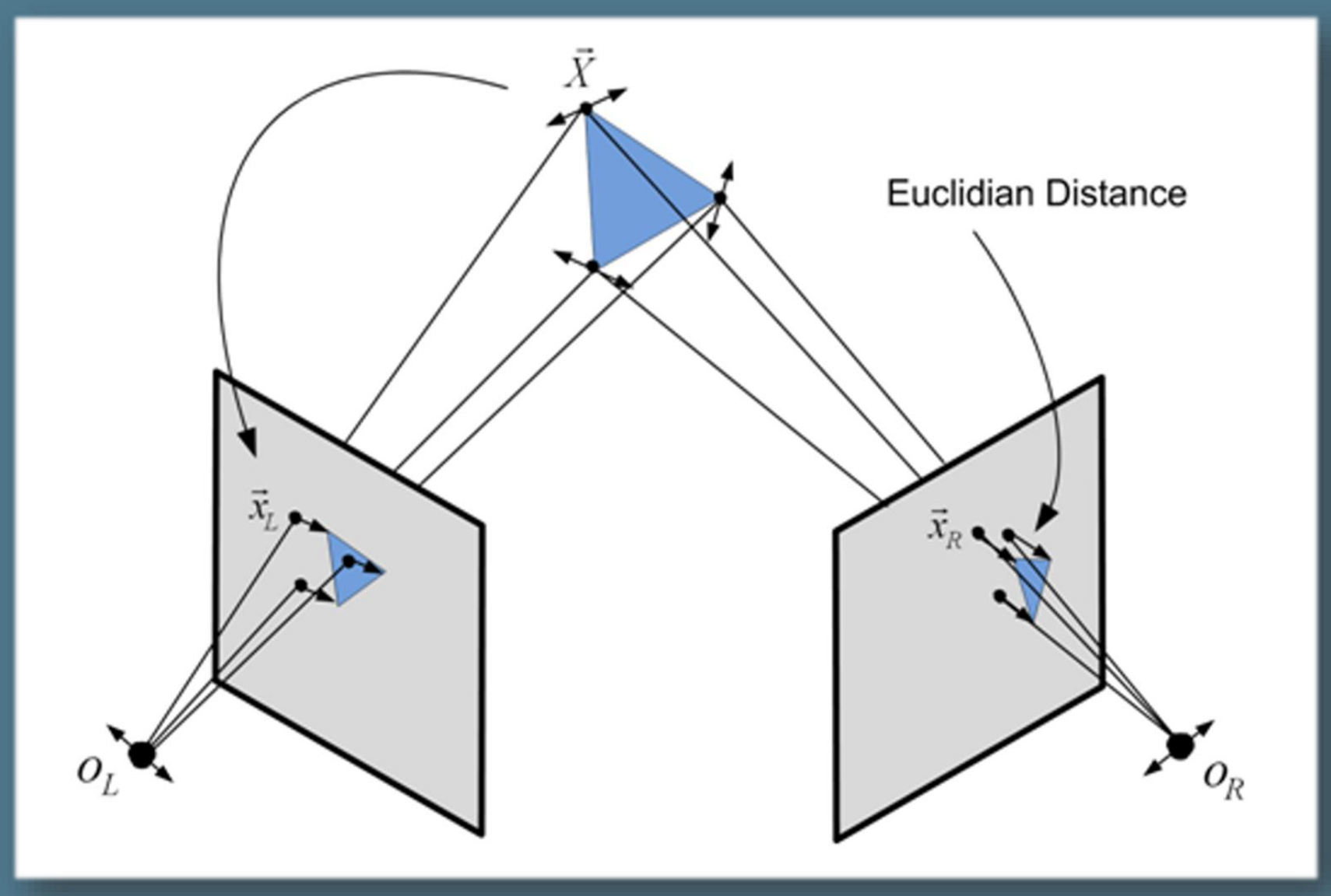
During initialization, 2-view triangulation is processing for creating a set of mappoints. Triangulation refers to the process of determining a point in 3D space given its projections onto two images. In order to solve this problem it is necessary to know the camera matrices. After obtaining the 3D location of matches points, mappoints are created and inserted into map.

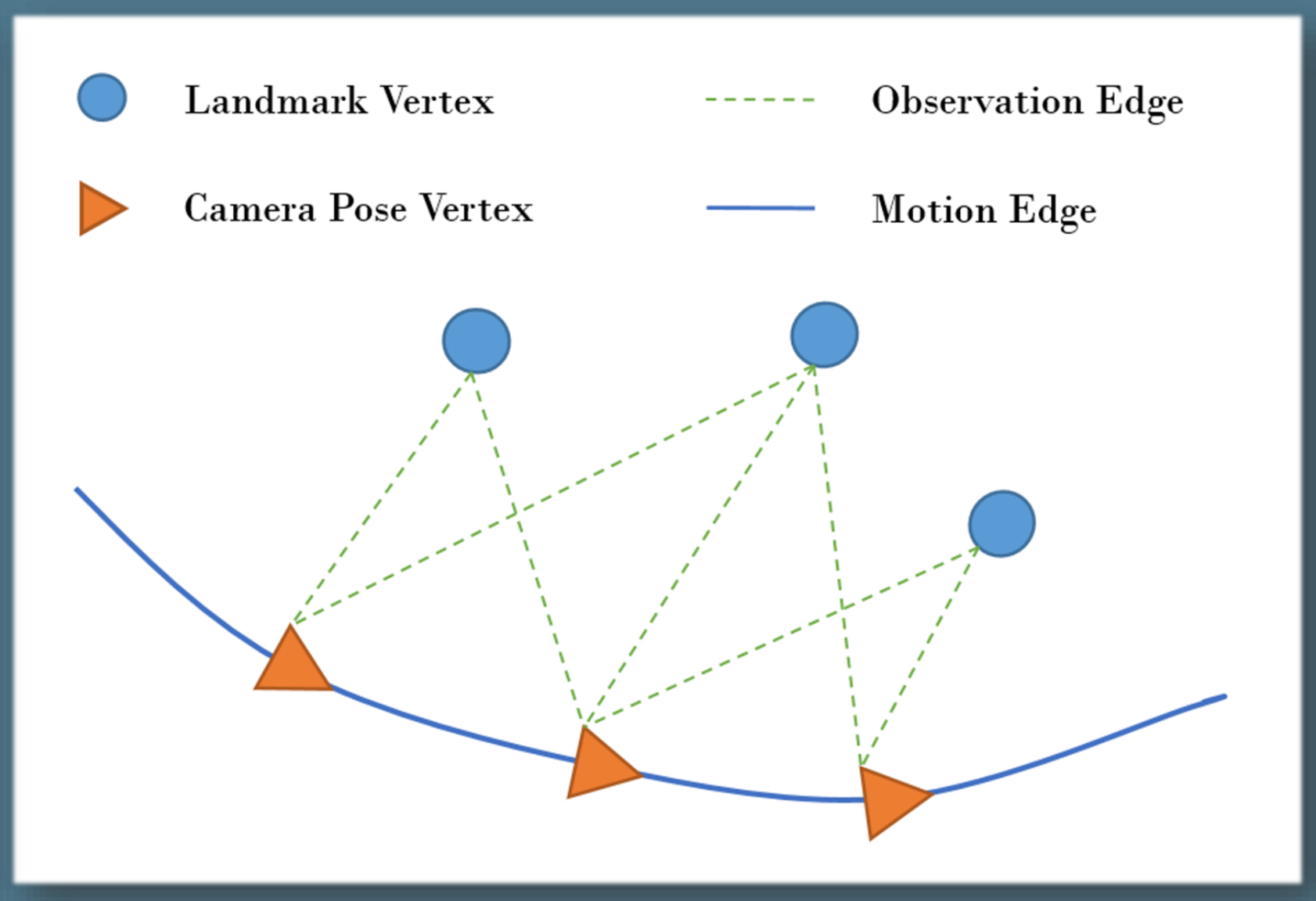
Software Architecture



Back-End: Local Bundle Adjustment

Bundle adjustment is the problem of simultaneously refining the 3D coordinates of mappoints and the parameters of the camera motion, given a set of images depicting a number of 3D points from different viewpoints. Its name refers to the geometrical bundles of light rays originating from each 3D feature and converging on each camera's optical center, which are adjusted optimally according to an optimality criterion involving the corresponding image projections of all points.

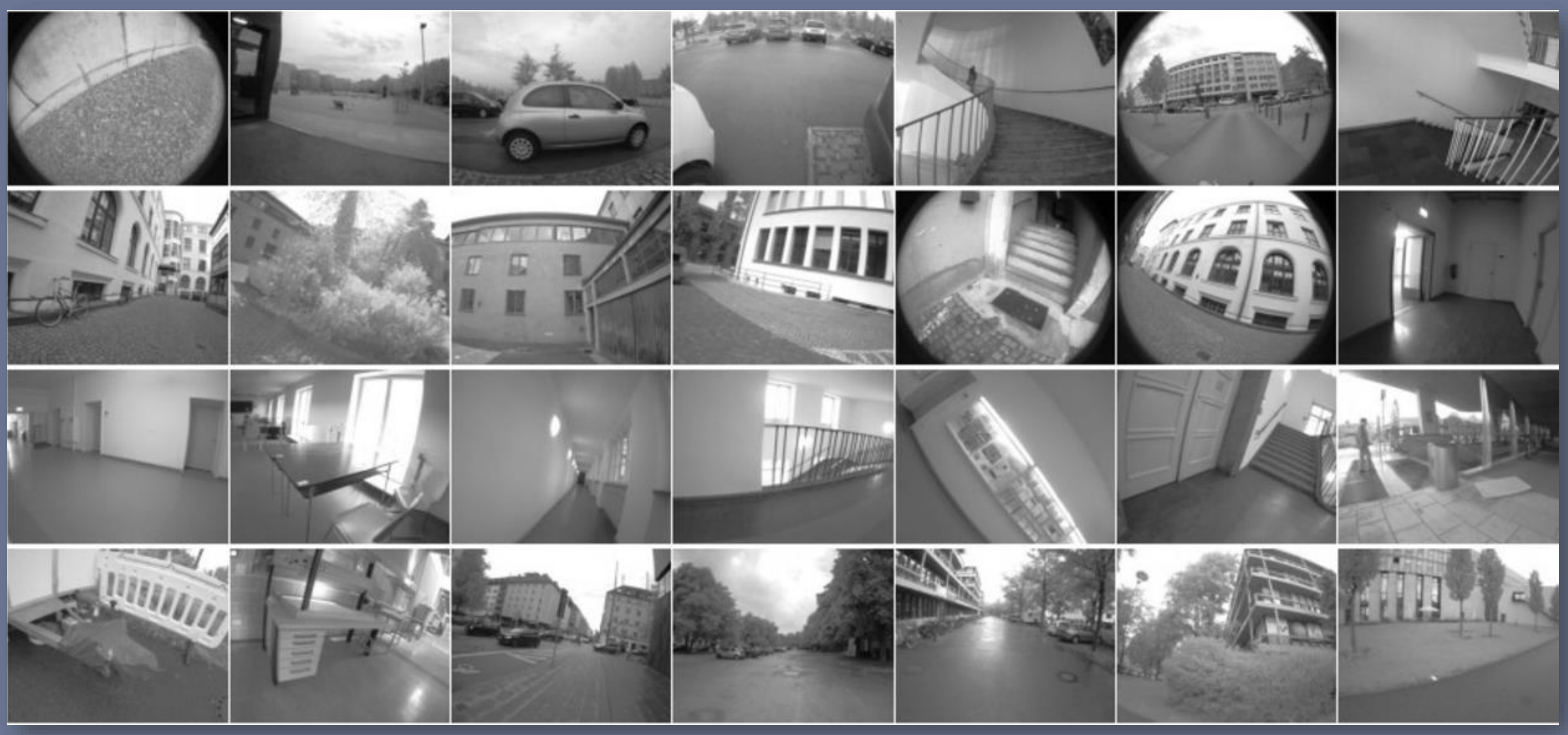




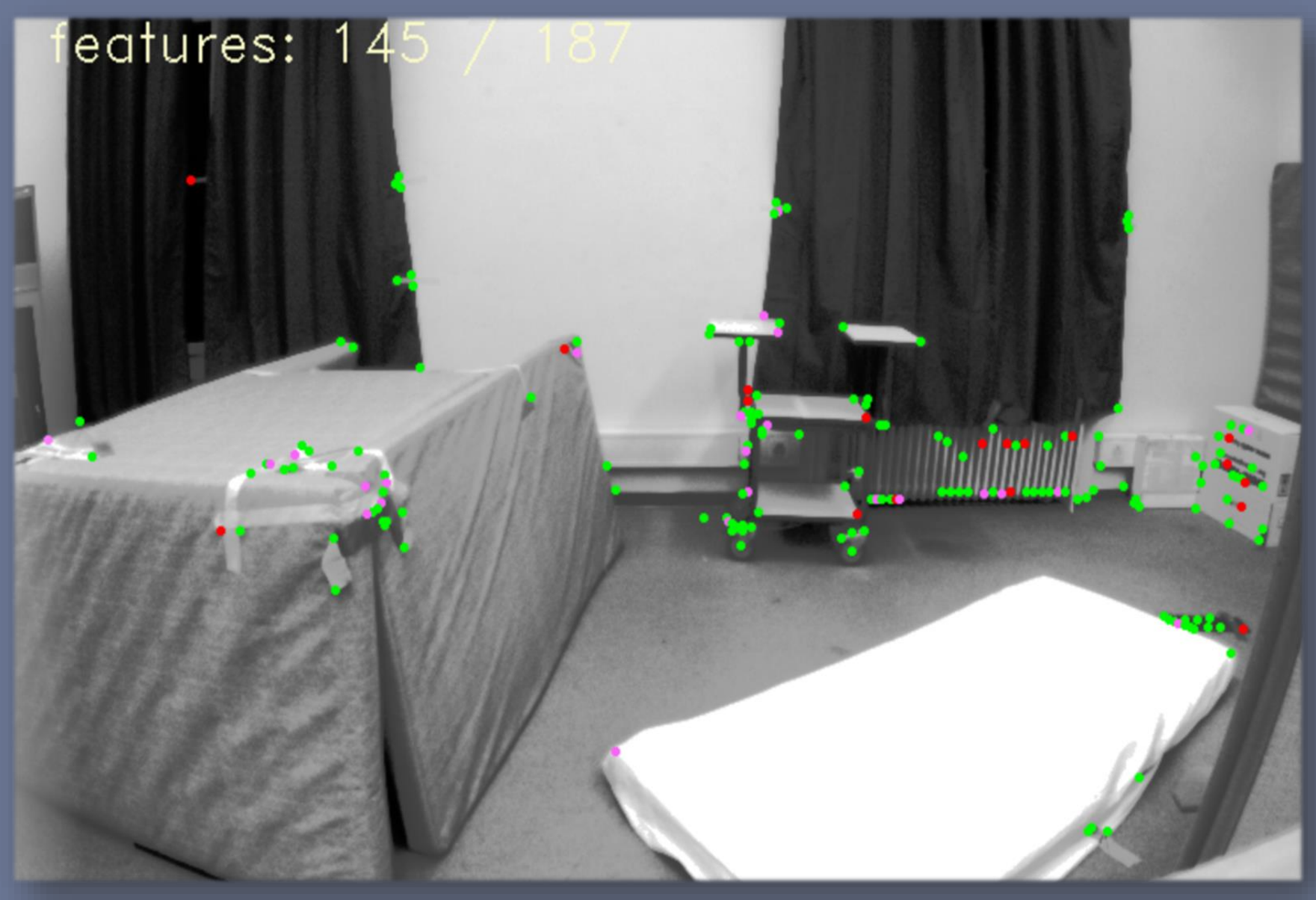
During this step in back-end, we use g2o for optimization. Camera poses and mappoints locations are set as vertices, while the observations are regarded as edges.

Result

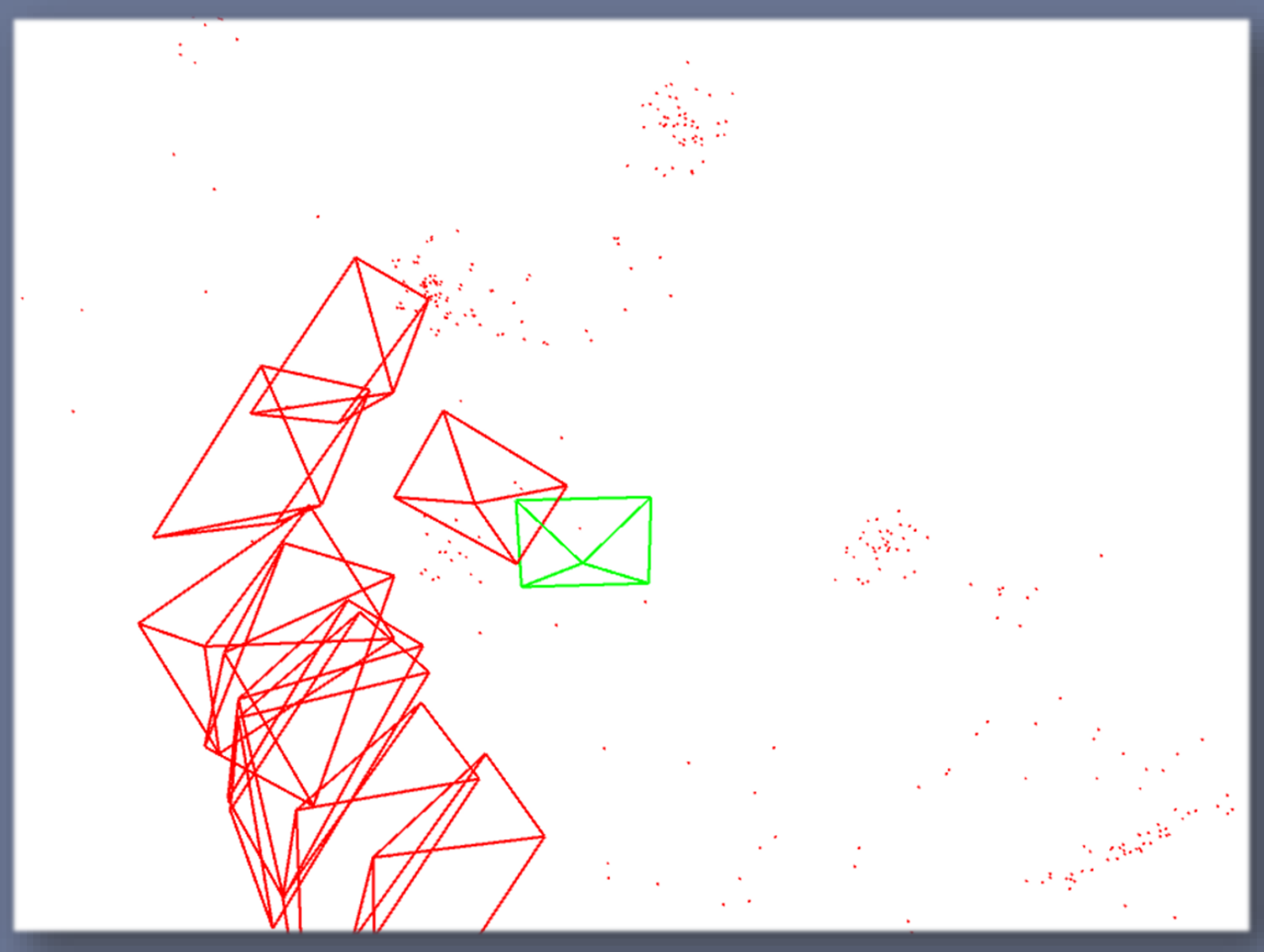
TUM Mono VO dataset



Feature Track Visualization



Camera Poses and Mappoints



Analysis

This naïve system suffers severe drift problem. After initialization, when the camera shakes, the backend optimizer will cull a large number of map points by marking them as outliers. When the next frame of image comes, it cannot match enough mappoints, so it is impossible to solve the PnP problem to estimate the pose. However, if I disable the back-end optimizer, the front-end will work fine until it drifts due to accumulated error.

In my view, there may be two possible reasons for this phenomenon: one is that the front-end feature tracking introduces a large number of mismatches, and the other is the improper setting of the optimizer during back-end optimization.

Unfortunately, I haven't fixed the problem so far.

Acknowledgement

I would like to give my heartfelt thanks to all the authors of VINS-Mono, ORB-SLAM2, and SLAMBook2. It is my great honor to benefit from their outstanding work and selfless open-source spirits.

My Project Website Here: github.com/kamzero/ECT_SLAM