Mini-project: A Visual Odometry Pipeline

Autors: Xiao'ao Song (xiaoao.song@uzh.ch); Kexin Shi (kexin.shi@uzh.ch)
Date: Jan. 10, 2021

1 Overview

This report includes building blocks of a monocular visual odometry pipeline. The pipeline contains two main parts: initialization and continuous visual odometry operations. Following this pipeline, we analyzed the results on the three given datasets. At last, we demonstrated how do we collect our own dataset and adopted the pipeline onto it with some necessary steps. Meanwhile, we proposed some alternative methods to solve the cause of failure in the visual odometry process. Some visual outcomes of this pipeline can be seen here at our youtube channel: *https://www.youtube.com/playlist?list=PLNDicWz0WmKx9FvRC04ntaGnnbVQeVNTH*.

2 Visual Odometry Pipeline

2.1 Initialization Algorithm

In this part, two-view geometry is used to estimate the relative pose between two selected frames and triangulate a point cloud of landmarks. This step is described in Algorithm 1:

Algorithm 1: Initialization

```
1: Detect keypoints in the first frame by Harris detector
2: found_next_keyframe ← false
3: while ~found_next_keyframe do
4:     Track keypoints in the candidate frame using KLT
5:     Estimate fundamental matrix by Matlab function estimateFundamentalMatrix
6:     Estimate the relative pose by decomposing Fundamental Matrix
7:     Triangulate Landmarks
8:     Landmark Sanity Check
9:     if keyframe_distance / average_depth > threshold then
10:        found_next_keyframe ← true
11:    else
12:        candidate frame ← candidate frame + 1
13:    end if
14: end while
```

2.1.1 Automatic Keyframes Selection

Instead of manually selecting two initialization keyframes, we automatically select keyframes according to the following formula:

$$\frac{keyframe\ distance}{average\ depth} > threshold$$

The threshold is set to 10% in our setting. This selection mechanism can make sure the baseline is large enough but also not too distant. Therefore, the bootstrapping step will be robust for different types of datasets.

2.1.2 Different Feature Detectors

To detect keypoints from each frame, there are many options, e.g. *Harris, BRISK, FAST, KAZE, ORB, MinEigen* and other feature detectors. To decide which feature detector to use, we compared those feature detectors on the first 10% frames in each dataset and calculate the average reprojection errors. The below table and Figure 1 show numerical and visual comparison of how each feature detectors performed on the first 50 frames in parking dataset. Based on the results, we chose Harris detector in our pipeline for all datasets.

Feature Detector	Harris	BRISK	FAST	KAZE	ORB	MinEigen
Average Reprojection error	148.0096	257.9047	196.5903	339.6837	207.142	185.3214

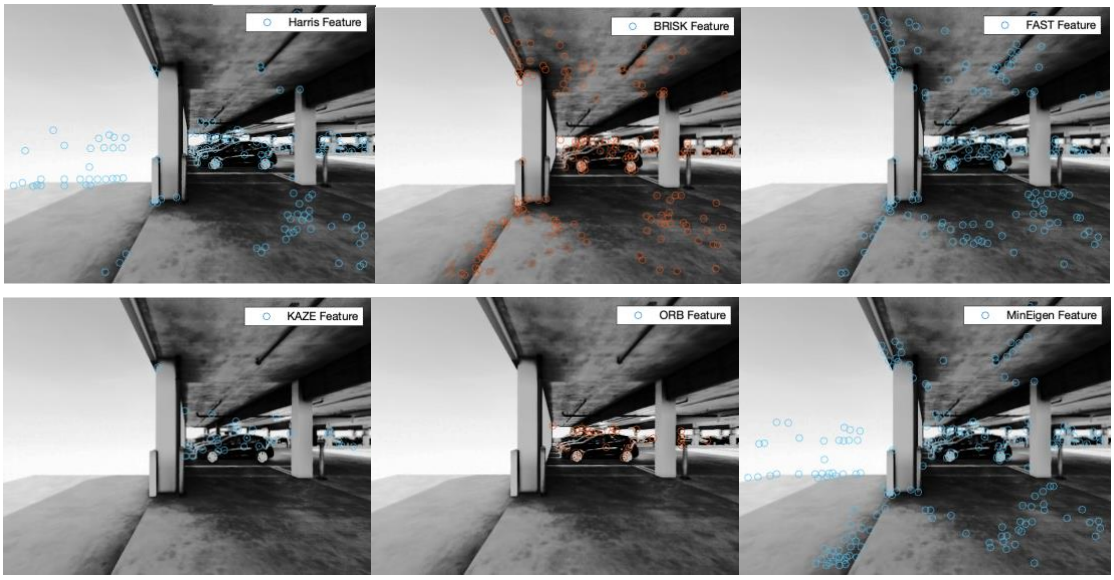

Figure 1: Different Feature detectors in frame 1 of Parking dataset

2.1.3 Feature Detection with Non-Maxima Suppression

Before feature detection, we enhanced the image intensify contrast by Matlab function ***histeq*** first. After the enhancement, the image is divided into several sub-regions. Then, we conducted Harris detector with a Non-Maxima Suppression on each sub-region. This process will ensure the keypoints are extracted evenly from the images and also representative.

Figure 2: Comparison of Harris detector without (top) and with Non-Maxima Suppression(bottom) on Kitti dataset

2.1.4 Track and Triangulate

Once we got the keypoints from the previous frame, we can track them to the candidate frame using KLT algorithm. Then the MATLAB function ***estimateFundamentalMatrix*** is used to get Fundamental Matrix. The current camera pose can be estimated by decomposing it. With the tracked keypoints and camera pose, we can then triangulate 3D landmarks.

2.1.5 Sanity Check of Landmarks

In order to improve the quality and rationality of landmarks, a sanity check is conducted subsequently. The basic logic is: we only retain the points that are located in front of the camera lens and within a certain range. Without these geometric constraints, the subsequent process will produce a lot of uncertainty.

2.2 Visual Odometry Continuous Operation

In this part, new frame is added into pipeline. We associate tracked keypoints in the new frame to previously triangulated landmarks, estimate the current camera pose and triangulate new landmarks. The algorithm is described below:

Algorithm 2: Visual Odometry Continuous Operation

```
for current frame in all frames do
    Track keypoints in the current frame using KLT
    if number(keypoints) < threshold then
        Extract SIFT features in the previous frame and current frame using SIFT toolbox
        Match keypoints in these two frames using SIFT toolbox
        Estimate fundamental matrix by Matlab function estimateFundamentalMatrix
        Estimate the current camera pose by decomposing Fundamental Matrix
        Triangulate Landmarks
    else
        Estimate the current camera pose using P3P/DLT with pose refinement
    end if
    Triangulate new Landmarks from candidate points
    Landmark Validity Check
    Landmark Sanity Check
    Bundle Adjustment
end for
```

2.2.1 SIFT Matching

When the number of keypoints tracked by KLT is lower than a threshold (say 3 points) in the current frame, the pipeline will fail due to insufficient points for subsequent P3P algorithm. To alleviate this, we use SIFT to re-detect features and match the previous frame with the current frame. SIFT matching improves the robustness of the tracking pipeline particularly when the images sequence has illumination changes, big viewpoint variations and even for small motion blurring. In our ***indoor_ros_img*** dataset, SIFT matching has better results than KLT tracking (see Figure 3). SIFT algorithm is implemented by Matlab open source package: ***VLFeat toolbox*** from *https://www.vlfeat.org/index.html*.

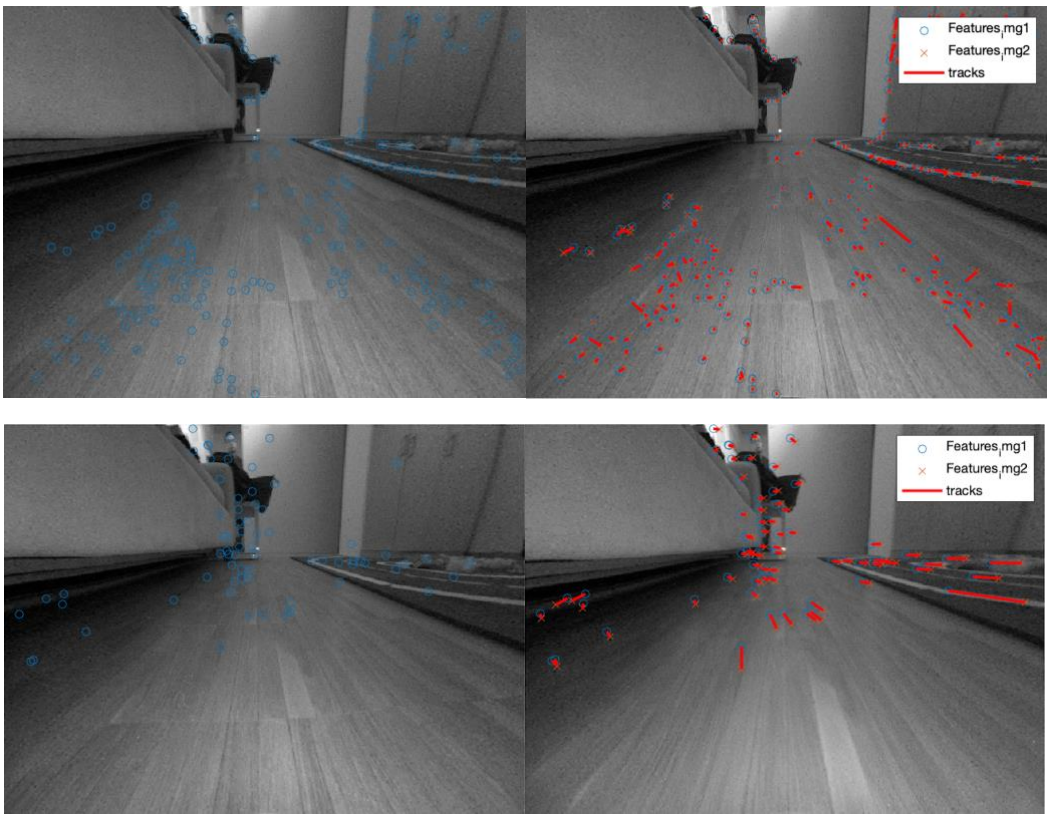

Figure 3: Comparison of KLT Tracking(top) and SIFT Matching(bottom) on ***indoor_ros_img*** dataset

2.2.2 Pose Refinement

When localizing a new frame based on the tracked keypoints and the current landmarks using P3P and subsequently DLT on the best inlier set, a small pose refinement is conducted. We have also tried using DLT only for localization but found it performed worse than P3P + DLT. A Comparison of these two methods is shown below in Figure 4. Therefore, in all of our experiments, we use P3P and then followed with DLT refining all inliers in the best P3P guess. After that, we minimize the reprojection error of all found inliers by optimizing the pose. Only very few iterations are necessary to reduce the reprojection error quiet a bit.

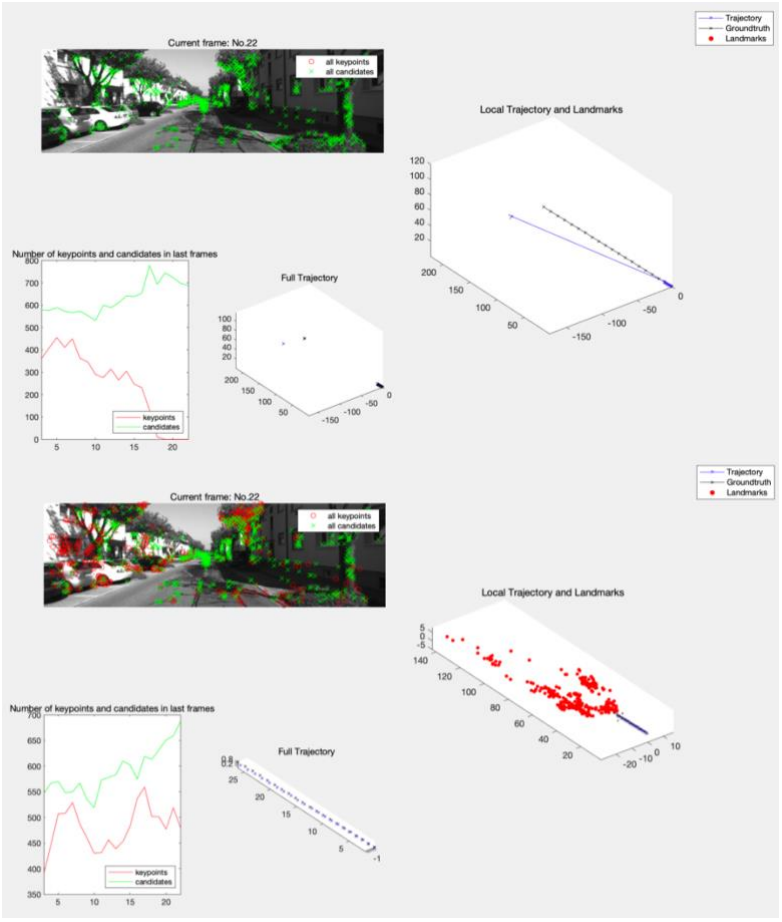

Figure 4: Comparison of standard DLT (top) and P3P with DLT refinement(bottom) on Kitti dataset

2.2.3 Validity Check of New Landmarks

Before adding newly triangulated landmarks, a validity check needs to be conducted. In addition to enforcing angle constraint, the reprojection error of the candidate landmarks should be constrained as well. Candidates with large reprojection error are consequently removed from the state.

2.2.4 Sliding Window Bundle Adjustment

To refine the trajectory, we also implemented bundle adjustment. The implementation contains the pre-calculation of non-zero elements in jacobian matrix to accelerate the process. The code was adopted from exercise 9. However, to further reduce the computation overhead, we perodically used a sliding window to optimize the past frames inside of the window range.

A comparison of global trajectory with and without bundle adjustment is shown below in Figure 5. As we can see that, after applying bundle adjustment, the global trajectory is closer to the ground truth trajectory.

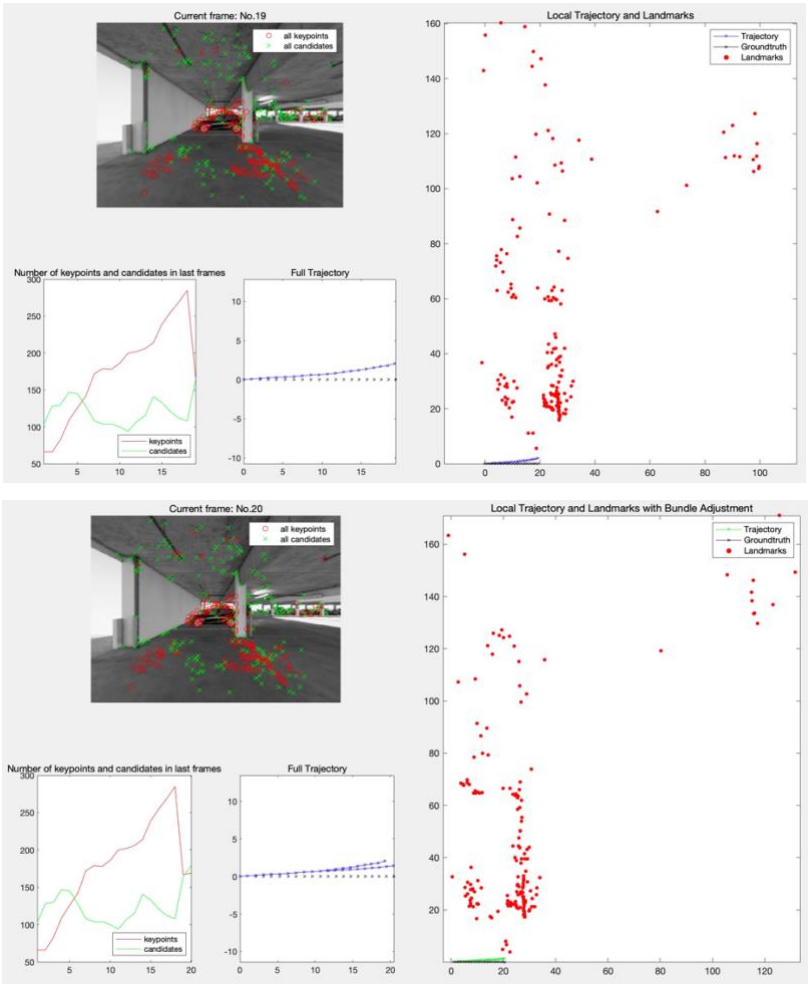

Figure 5: Comparison of trajectory without BA (top) and with BA (bottom) on parking dataset

3 Results

3.1 Primary Tunable Parameters

The primary tunable parameters in our pipeline and their description are listed in the table below:

Functionality	Parameter Name	Description
Harris	rows	Number of rows of the sub-region.
	cols	Number of columns of the sub-region.
	radius	Radius used for the Non-Maxima Suppression.
	npoints	Desired number of features per sub-region.
	bins	Bins count for image enhancement.
Matching	maxBidirectionalError	Forward-backward error threshold for KLT.
	block size	Size of neighborhood around each tracked point.
Triangulation	sanity check factor	The upper limitation of sanity landmarks
	angle threshold	Angle threshold for new landmarks.
	reprojection error threshold	Reprojection error threshold for new landmarks.
BA	periode	BA period
	window size	Window size for windowed BA.

3.2 Result Analysis on Given Datasets (Parking, Kitti, Malaga)

All experiments were run on a machine with the following specifications: i5-3733 MHz CPU, 8 cores, 16GB RAM and using a maximum of 121 threads. The Matlab version is R2020b.

Our VO pipeline runs on the parking very nicely. The feature tracking and the localization work very reliably. The result on Kitti dataset looks quiet promising. The below Figure 6 shows an example frame in tracking process. The black line is the ground truth trajectory and the blue line is our result. A bit of scale drift is noticeable because we enable 3D plotting, while 2D plotting result looks better. Malaga is the most difficult dataset to handle because of large scale drift and lack of ground truth. Nevertheless, our VO pipeline runs successfully on both of the dataset.

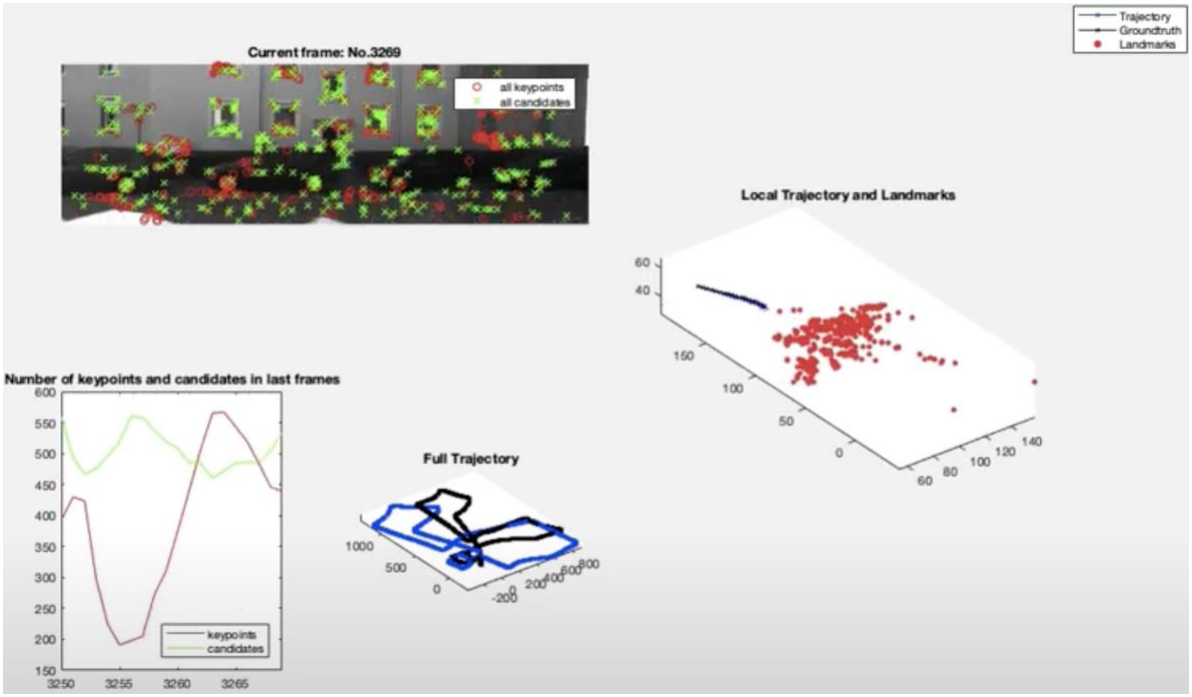

Figure 6: An example frame on Kitti dataset: black line refers to ground truth, blue line refers to our result.

4 Additional Features: Own Datasets

Except for applying the pipeline onto the given datasets, we also recorded our own data set and tuned the program to make it work on the new dataset. We recorded our own dataset based on indoor environment. Unlike the outdoor environment, indoor environment is more challenging because it contains lots of textureless objects such as door, wall, floor and etc.

4.1 Collect Data using ROS by Duckiebot

Benefit from another course, ***Autonomous Mobility on Demand: From Car to Fleet***, I had this semester, I can use a duckiebot to drive around my apartment and record images with its on board camera sensor. We wrote the ROS code to collect data and the scripts are also provided along with our project submission. The below Figure 7 shows how the duckiebot looks like. The benefit of using duckiebot to collect data is that we do not need to calibrate the camera; the default intrinsic matrix can be found from a yaml file of its system.

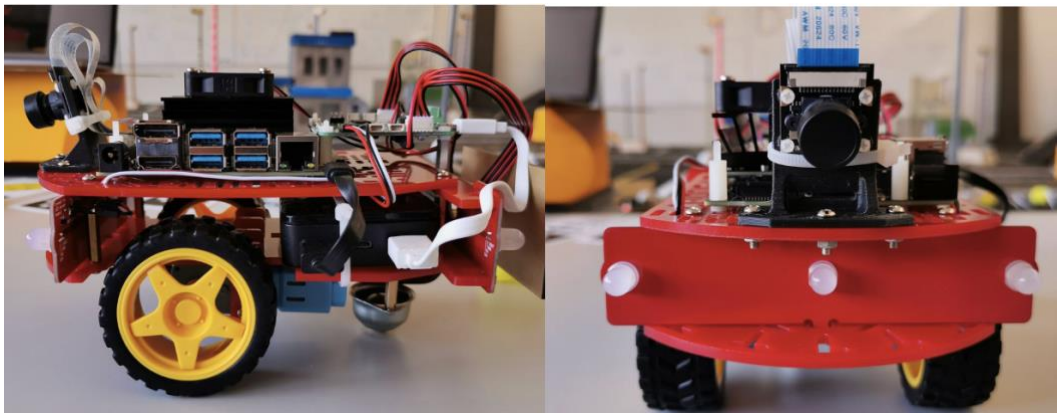

Figure 7: Appearance of duckiebot. Left: side view. Right: front view

However, the datasets collected by duckiebot are very challenging because of these issues: low image quality, motion blurring, image distortion and etc. Without image preprocessing, it is hard to adopt our pipeline onto this data set. Two sample images are shown in Figure 8.

Figure 8: Image with motion blurring (left) and Image with distortion (right)

4.2 Collect Data using iPhone8 Plus

Then we decided to record the dataset by iPhone8 Plus. To get the instrinsic parameters of its camera. We applied Zhang's Camera Calibration Algorithm: taking 36 images of a checkerboard from different view points and utilize the Matlab calibration toolbox from http://www.vision.caltech.edu/bouguetj/calib_doc. The calibration process is shown in Fig. 9.

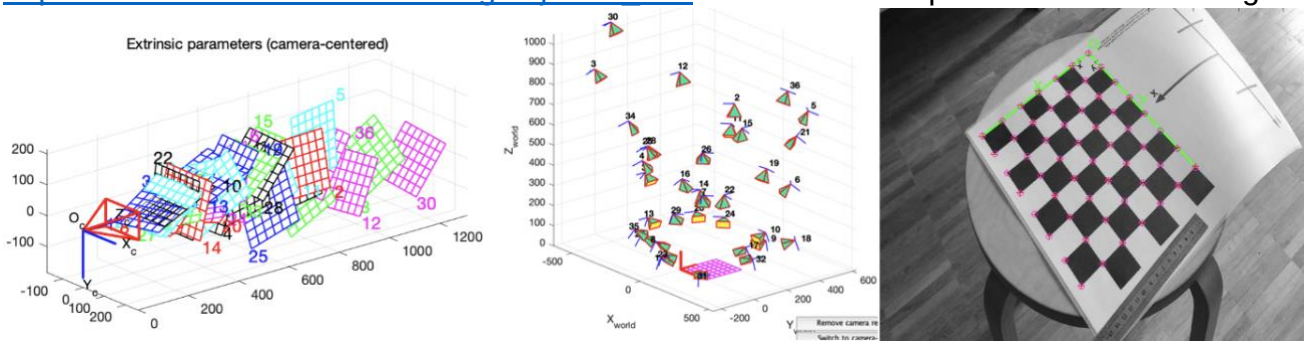

Figure 9: Camera calibration

The frame rate of iPhone8 Plus is 30 fps and we downsampled it to 3 fps. Some extracted sample images from this dataset are shown below. Except for the kitchen and bedroom,

walls, doors, and aisles are almost without texture. Therefore, our visual odometry pipeline failed to map the whole trajectory. Some failed video snippets are also uploaded onto our Youtube channel.

Figure 10: Sample images: kitchen, living room, bed room, aisle and etc.

4.2.1 Alternative Remedies

4.2.1.a SIFT

As discussed in section 2.2.1, SIFT matching can improve the robustness of the tracking pipeline; therefore, one of the remedies is we switch to SIFT matching when we lose all the tracked keypoints. However, SIFT is not as good at localizaton as Harris detector, the trajectory totally messed up in this dataset (see Figure 11).

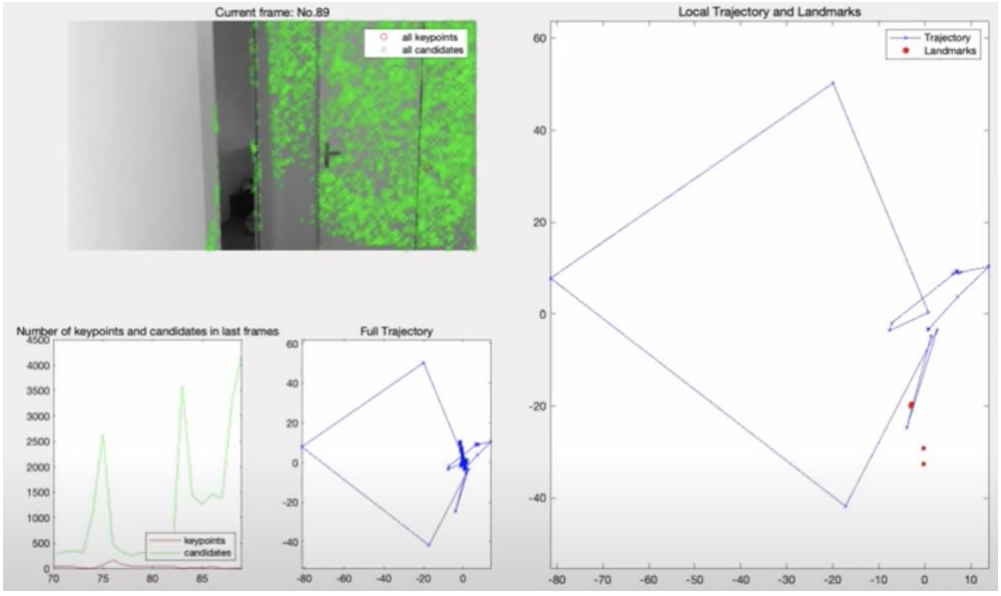

Figure 11: Alternative remedy: adding SIFT matching to VO pipeline.

4.2.1.b Adding Visual aids and Increasing RANSAC Iterations

Another way to remedy this is adding visual aids at where the images are lacking textures. The visual adds we used include duckies and April tags. We placed these visual aids on the wall, aisles and corners (see Figure 12). Meanwhile, when the tracked keypoints are lower

than a certain threshold (say 50), we increased the iteration numbers of RANSAC greatly. This will help us to get a better model and more inliers as keypoints.

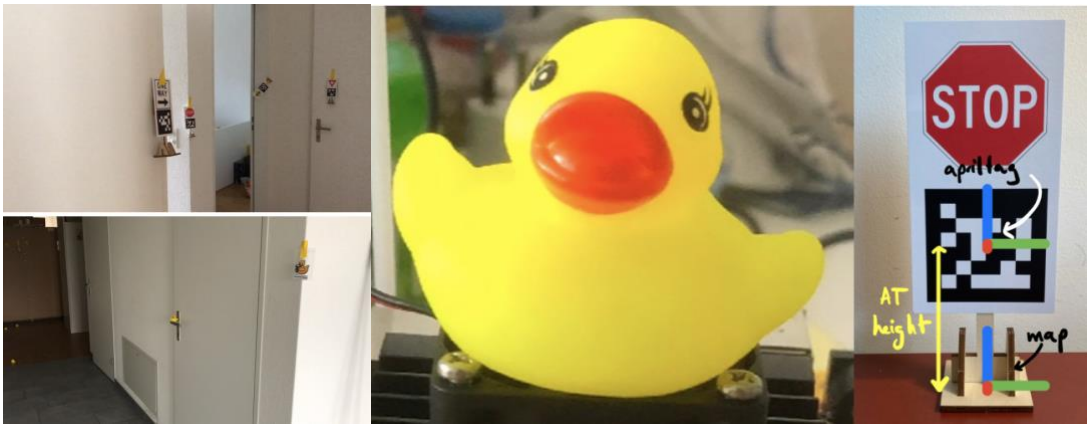

Figure 12: Adding visual aids

By this remedy, our VO pipeline runs nicely on this indoor dataset. The full trajectory is close to the ground truth we drawn (see Figure 13).

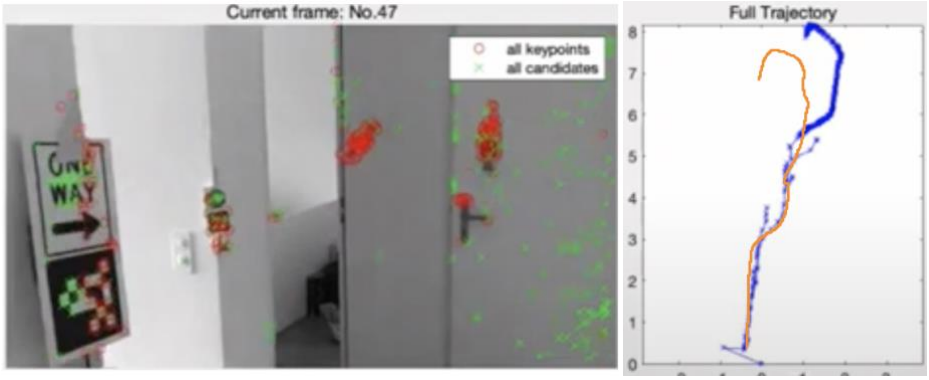

Figure 13: Left: Detecting April tag. Right: full trajectory mapping by our VO pipeline (blue) and ground truth trajectory (orange).

4.2.2 Indoor Dataset: Stairs

At last, we collected another indoor dataset: ***stairs*** and applied our VO pipeline. The full trajectory can be seen on Figure 14.

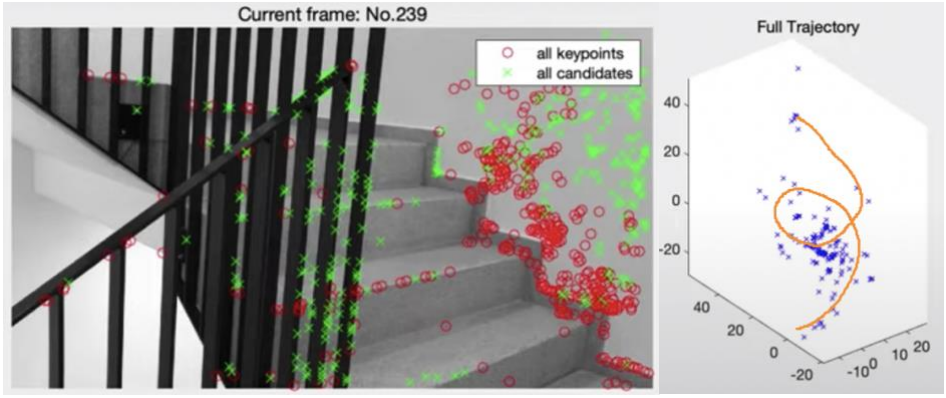

Figure 14: Left: Detecting stairs. Right: full trajectory mapping by our VO pipeline (blue) and ground truth trajectory (orange).