# VAMR Mini-Project Report

- Visual Odometry Pipeline -

Bowei Liu (23-941-495)
Hanyu Wu (23-956-352)
Kehan Wen (22-945-117)
Shi Chen (22-943-005)

# Contents

# Chapter 1

# Introduction

In this mini-project, a monocular visual odometry pipeline is to be implemented. As the lectures and exercises explain, visual odometry consists of several sequential procedures, as shown in Figure 1.1. After receiving raw images, features should be efficiently detected and extracted from the images. Then, the extracted features are compared and matched, so that the motion of features can be tracked. With the information gathered from the features, the camera can be further localized, and its motion can be estimated.

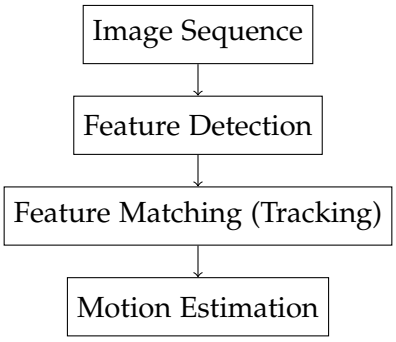


**Figure 1.1:** Visual Odometry Flow Chart

This report is divided into the following parts:

- In the 2nd chapter, the initialization module will be presented as the first main component. The initial set of 2D-3D correspondences from the first frames of the sequence is extracted. The initial camera poses and the module bootstrap landmarks.
- A continuous visual odometry module will be introduced in the 3rd chapter of this report. In this module, the camera's current pose is estimated using the existing set of landmarks, while the new landmarks are triangulated regularly.
- Finally, this pipeline will be summarized in the last part of this report.

# Chapter 2

# Pipeline

In this chapter, we will introduce the overall pipeline of our approach.

## 2.1 Module 1: Initialization

The general initialization module is illustrated as below.

**Algorithm 1** Initialization module

**Require:** Dataset $\mathcal{D}$ and corresponding hyperparameters.
1: Setect two frames from $\mathcal{D}$ used for bootstrapping.
2: **while** `keypoints_num < cfg.init_min_keypoints` or `mean_projection_error > 1` **do**
3: Detect initial keypoints in the first frame.
4: Initialize the KLT tracker and track the keypoints in the second frame.
5: Estimate the fundamental matrix using matched keypoints.
6: Extract extrinsic matrix from fundamental matrix given Intrinsic matrix.
7: Triangulate the landmarks and return the metric `mean_projection_error`.
8: **end while**
9: **Output:** Initial state *S_init*.

**Bootstrapping frames selection** Firstly, two frames are chosen at the beginning of the dataset. To make sure that the baseline between the two initialization frames is appropriate, we don't choose the second frame as the second bootstrapping frame but a few frames later (tuned depending on the dataset), so that the baseline between them is large enough, but also not too distant from each other. As suggested in the project statement, we tailored the initialization images of `KITTI` case to be the 1st and the 3rd frames.

**Keypoint detector selection** The keypoints of the first frame is extracted by the feature detector. We have faced much difficulty while choosing an appropriate feature detector and its parameters. After testing Harris, Shi-Tomasi, ORB, SIFT detector and so on, we

decided to use SURF detector because of its faster computation compared to SIFT and its invariance to scale, rotation and viewpoint.

**Keypoint Association** We compared the feature descriptor matching and KLT tracking to find the keypoint correspondence. Using the same keypoints detector, KLT find more corresponding matches. We speculate that the results can be attributed to the dataset fitting the assumptions of the KLT tracker well. These assumptions specifically include brightness constancy, temporal consistency, and spatial coherency.

**Relative Pose Extraction and Evaluation Metrics** Based on these matched keypoints, we employ RANSAC to estimate the fundamental matrix and find the inlier points which can be done via MATLAB function of `estimating_fundamental_matrix`. Given camera intrinsic parameters, the relative pose can be determined using MATLAB function `estrelpose`. After validating this relative pose in terms of reprojection error, this correspondence can be confirmed. Thus, we can triangulate the a point cloud of 3D landmarks.

**Initialization of the Data Structure** `S` After gaining the initialization keypoints and landmarks, they are assigned to be the parameters of the first "previous" state of the continuous VO pipeline. In order to perform the visual odometry, further parameters, such as candidates of keypoints, first observation and poses of candidates, are also added.

## 2.2 Module 2: Visual Odometry

In this part, the visual odometry module will be introduced. In general, this module is implemented with three main sequential functions:

- Associating keypoints in the current frame to previously triangulated landmarks
- Estimating the current pose
- Triangulating new landmarks regularly by keypoints which are not associated to previously triangulated landmarks

As recommended in the statement, the data will be processed in a Markov way, which means that the function inputs depend merely on the output the previous function call and thus is easier to maintain. The data flow is shown in Figure 2.1,


**Figure 2.1:** Function Design for Visual Odometry Module

**Algorithm 2** VO module

**Require:** Current and previous frames, previous state `S_prev`, and corresponding hyper-parameters.

Initialize KLT Tracker and track the keypoints in current frame.
Estimate the current camera pose.
Relax the metric threshold if there are not enough keypoints.
**if** `S_prev.candidate_keypoints` is not empty **then**
    Delete the keypoints which do not meet the triangulation condition or reprojection condition for too long.
    Initialize KLT Tracker and track the candidate points in current frame.
    Calculate the bearing vector angle.
    Triangulate the candidate points and calculate the reprojection errors.
    Add the landmarks if both bearing vector and reprojection error meet the requirements.
**end if**
Detect the features and add it to candidate list if they are not redundent.
Visualization.
**Output:** Current state $S$.

**Keypoint Association** We can associate keypoints like what we have done in the initialization part. By using KLT tracker, we try to track the keypoints from the previous frame and remove the keypoints that can not be tracked.

**Estimation of Current Camera Pose** The camera poses can be further determined by using P3P and MSAC which is used to remove outliers.

**Triangulation of the new landmarks** As the camera moves far away, fewer landmarks obtained in initialization step can be tracked. Thus, we need to continuously update the landmarks by triangulating new keypoints. To this end, we create 3 lists, namely candidate keypoints, the location of the candidates in image when the first observed, and the corresponding camera pose.

Firstly, if the candidate list is not empty, we try to track these candidate keypoints in current image using KLT tracker and remove the candidates that can not be tracked. Then, we triangulate these candidate keypoints and check whether they meet the following conditions:

1 the angle between two bearing vector is larger than a threshold.

2 the reprojection error is not too large.

Only if the candidate keypoints meet both of the conditions, they could be added to the keypoints list and we obtain therefore new landmarks. If the candidates cannot meet both conditions for too long (i.e. consecutively for over a certain number of frames), they will be removed from the candidate list.

In the next step, we expand the candidate list. First, we detect new features in current image and compare these features with the features in keypoints list. Here we compute the distance between them to estimate whether they represent an identical feature. If not, we add these new keypoints to the candidate list and record their location in image and current camera pose.

# Chapter 3

# Results

The estimated trajectory of the camera are plotted and analyzed in the following part.

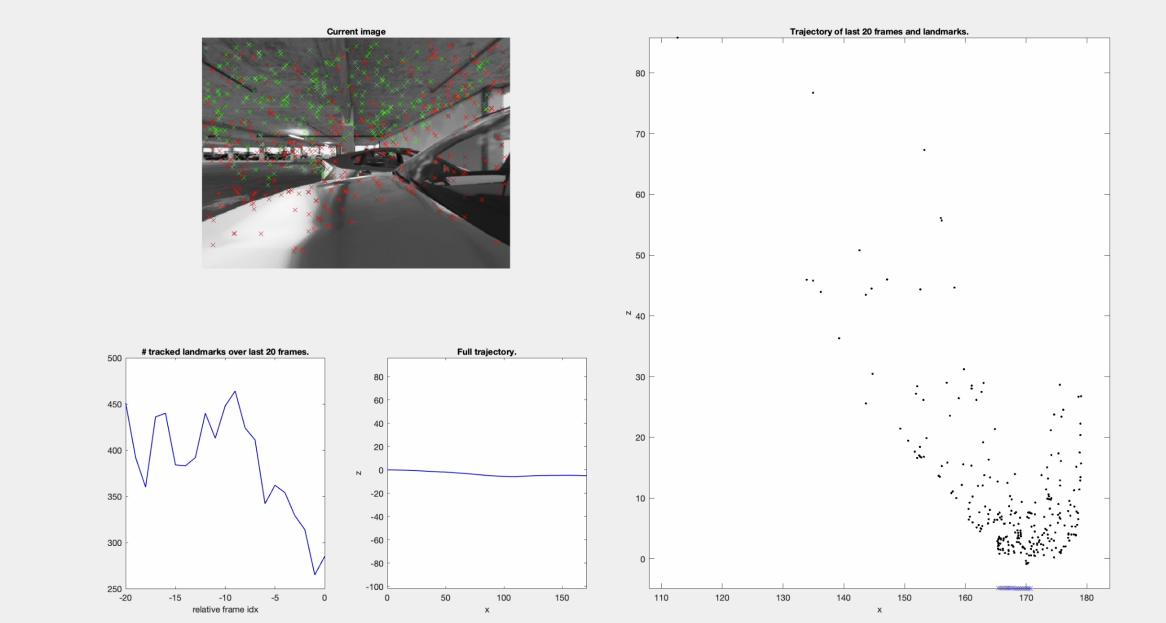


**Figure 3.1:** Estimated Trajectory from Parking Dataset

As shown in Figure 3.1, this code implementation is capable of estimating the trajectory of the vehicle in general, as the output of estimated trajectory corresponds to the visual images and the ground truth trajectory in shape.

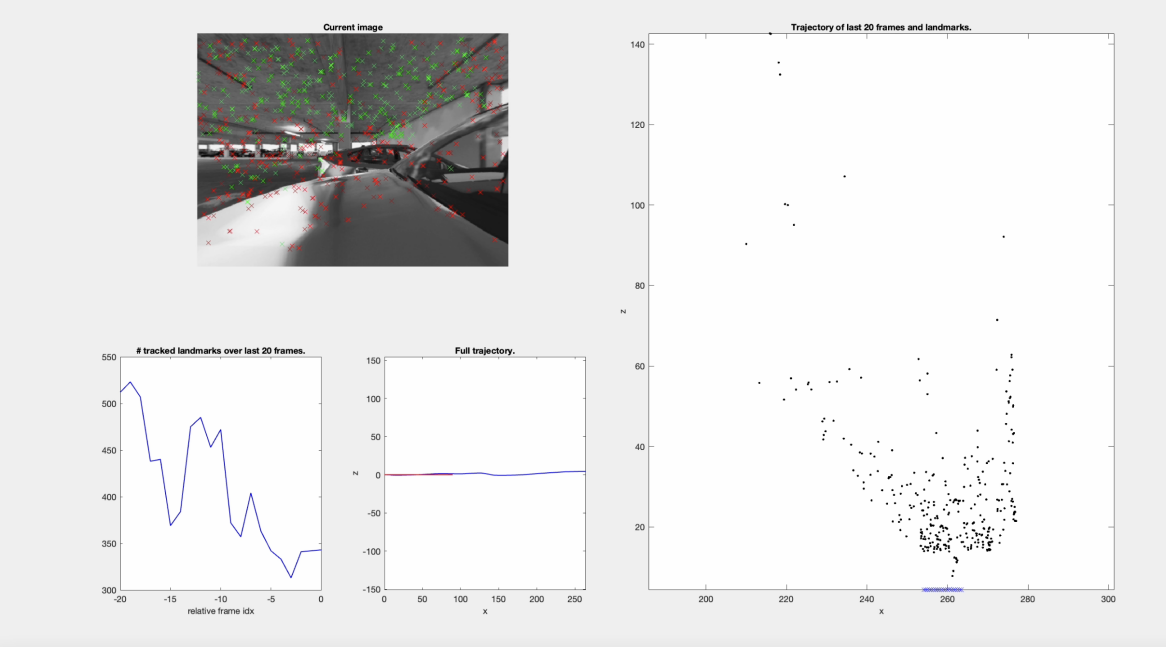


**Figure 3.2:** Estimated Trajectory with Ground Truth from Parking Dataset

Furthermore, by comparing the estimated trajectory to the ground truth, we can easily observe the scale ambiguity of monocular vision odometry, as shown in Figure 3.2 where the red line in true trajectory denotes the ground truth trajectory. Furthermore, due to the randomness of the RANSAC, we can encounter different scale every time we run the code.

As can be seen in our screencasts for KITTI and Malaga datasets, the scale may change significantly when the vehicle experiences rotation. This is not surprising since the landmarks are not consecutive if the rotation is relatively fast (i.e. The VO pipeline needs to track the camera with almost a brand new set of landmarks after the rotation). Therefore, if the current scale cannot hold anymore, essentially another translation scale will be assigned after the rotation. Sometimes, the scale gets too small that the estimated speed is reduced enormously, which leads to almost unobservable motions in the scale of global trajectory.

# Chapter 4

# Conclusion

In this project, a pipeline is constructed in order to implement a visual odometry algorithm for monocular camera. And this pipeline is tested in three different data sets. Within this pipeline, two main modules are implemented.

In the first module, the initialization process is implemented in order to obtain the initial 3D point model and relative pose from some beginning frames. The second module can further be initialized based on the information gained in this module.

In the second module, the continuous VO is implemented, which consists mainly of three parts. In the first part, keypoints of the current previous frame are associated by KLT tracker. The landmarks, which are bound to keypoints in the previous frame, are simultaneously chosen so that the keypoint-landmark association is achieved. In the second part, we use this keypoint-landmark association to estimate the relative pose transformation. In the last part, because of the loss of old landmarks during moving, new landmarks are triangulated. Valid landmarks are chosen by checking the performance of corresponding candidate keypoints. In our design, the bearing angle and the reprojection error of the candidate keypoint are the two metrics to determine the valid new landmarks.

Apart from these two data processing module, a visualization module is also implemented for illustration strictly according to the requirements in the project statement.

As we test our VO pipeline on KITTI, Malaga and parking datasets, we found that it shows acceptable local consistency up to a scale. However, we also observe that it experiences significant scale drifts almost inevitably after a few fast rotations. We expect it can be alleviated by elaborate bundle adjustments, but cannot be perfectly tackled due to the monocular nature of the pipeline.

# Appendix A

# Appendix

## A.1 Result Screencasts

| Dataset | Link |
|---|---|
| Kitti | https://youtu.be/Vox3zYr6nsY |
| Malaga | https://youtu.be/7Sj3SLY8QuU |
| Parking | https://youtu.be/MsBfpxU7oRM |

**Table A.1:** Links to Results

## A.2 Author Contributions

The following is the contribution of all the team members.

Bowei Liu mainly works on the code implementation of continuous VO module.

Hanyu Wu mainly works on the code implementation of VO module.

Kehan Wen works on the code implementation of initialization module. He is also responsible for recording the screencasts of all data sets.

Shi Chen works on the code implementation of the visualization module. He is also responsible for refactoring and merging the initialization, VO and visualization modules into the same framework.

In the end, all four of us write the report together on overleaf.