Visual Algorithms for Mobile Robotics Project Report

Adarsh **Patnaik**, Abhiram **Shenoi**, Jasper **Tan**, Tianyi **Lim**

January 8, 2023

1 Introduction

Localization in robotics is a key problem which requires to be solved for most robotics applications [2]. Traditional methods use an array of sensors like GPS, IMU and Encoders for wheeled robots in order to estimate the position of a robot. These sensors may under perform in certain scenarios and the localization estimate would be incorrect. Cameras are one of the widely used sensors in all robotics applications and considerable improvements have been made in the field of visual systems and hence, a better way to perform odometry is to rely on visual systems like cameras, LiDARs etc.

Visual odometry is considered a subpart of a bigger structure from motion (SFM) problem in computer vision. Although it can be used in both outdoor and indoor environments, it is considered accurate only in feature rich scenes as opposed to a textureless environments. Within this project we implement a Visual Odometry pipeline that can provide the localization estimate of a robot using a single camera.

2 VO Pipeline Overview

The flow of data and information are described in Fig.1 and Fig.2.

In Figure 1, we track Harris features that are classified as keypoints between the first image and the manually specified bootstrapping frame using the KLT tracker. Then the relative pose between these two frames estimated. Using the SFM idea, these keypoints are then triangulated and form the inital pointcloud that will subsequently be used for localization and also expanded to in continuous operation.

For continuous VO using 3D to 2D correspondences, the VO state propagated from one frame to the next needs to track the homogenous point correspondences, the 3D world locations of these positions and also candidate keypoints along with transformation matrices. Using these tracked points, we first estimate pose using PnP-RANSAC and remove the outlier correspondences. We then try to triangulate new points based on different criteria like reprojection error, parallax and position of points infront of the camera.

However, we end up using 2D to 2D correspondences for pose estimation considering better performance, and therefore in this approach VO state simply tracks the homogenous pixel coordinates of keypoints in the currently-processed frame. As some points are lost in tracking, we regularly add new features which are not already being tracked to be used for the next frame.

For the most part we have stuck to using the Python implementations written for the exercises. However for the following algorithms we use the `OpenCV` implementation [1]:

- Harris corner detection
- PnP implementation and the Rodrigues method
- Finding the fundamental matrix between two views - normalized 5 point algorithm with RANSAC

3 Bootstrapping

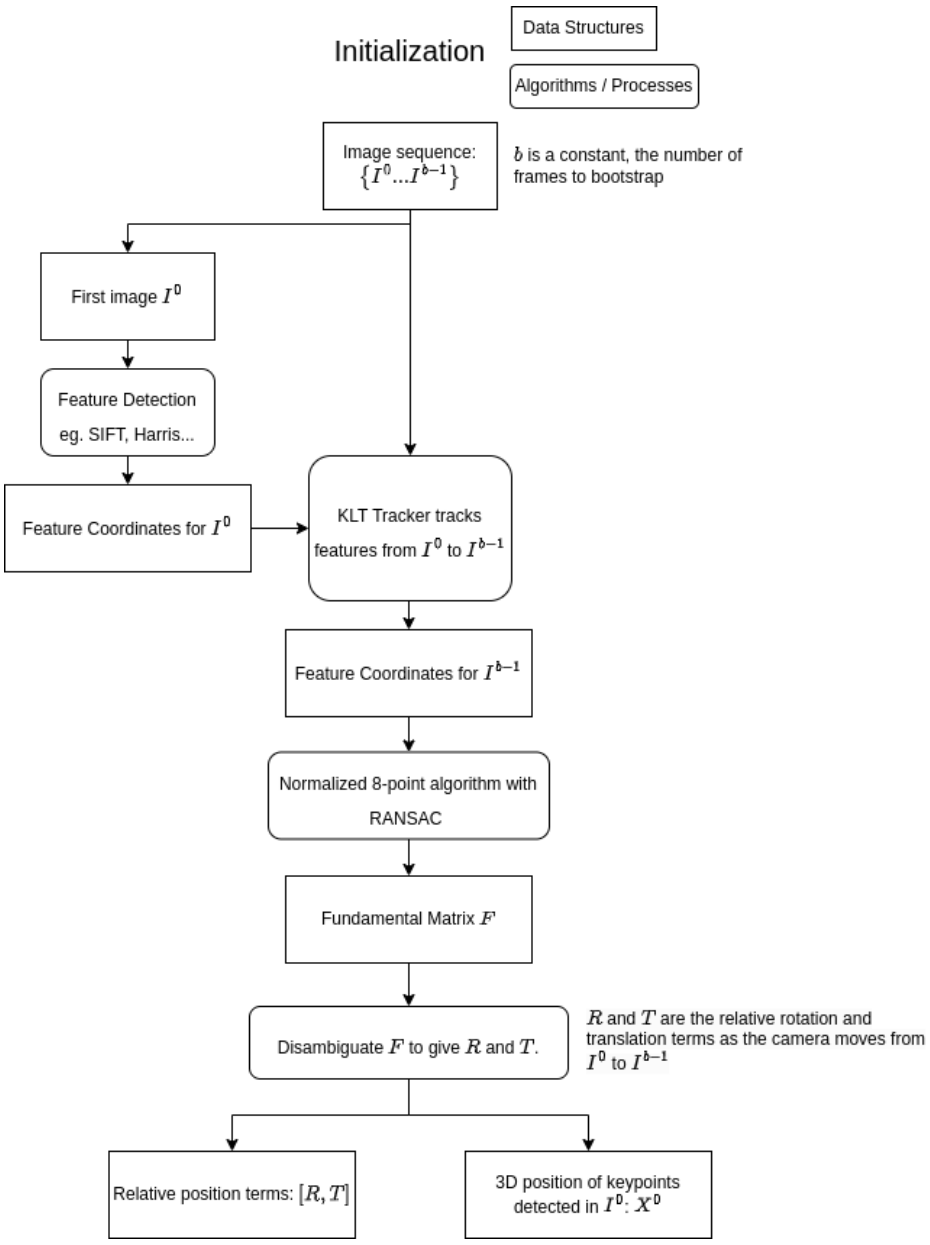

Figure 1: Flowchart of bootstrapping

3.1 Feature detection

To detect features for bootstrapping, we decided to use the Harris detector over the SIFT detector as the Harris detector is significantly faster than SIFT. Harris process frames about 5 times faster than SIFT on a Apple M1 ARM-based systems on a chip (MacBook Air).

Harris is faster as it does not check a patch through different scales and directions. While this means that Harris is not scale and rotation invariant, these properties are only relevant when doing brute force matching to track features between one frame and another. Instead, this pipeline uses the Kanade Lucas Tracker (KLT) to perform optical flow, tracking pixels from a template (a patch around a detected feature) between frames.

To implement SIFT, we used solutions provided for Exercise 04. To implement Harris, we used OpenCV's Harris detector function.

Finally, the feature detector returns an array of (x,y) pixel coordinates describing keypoints from the first image.

3.2 Feature tracking - KLT

As mentioned, feature tracking can be done using brute force matching or KLT. We chose to use KLT over brute force matching as it is more efficient. Apart from computation saved using Harris over SIFT, KLT is faster than brute force as the number of features to match increases.

Secondly, brute force matching requires a distance-checking function, to remove ambiguous points. Doing so would potentially remove points available for structure from motion, lowering accuracy. KLT optimises the template warping in the template's surrounding region, and since consecutive frames are passing into the function, we have spatial coherence.

Finally, the KLT function for bootstrapping returns two arrays of (x,y) values, one describing the locations of keypoints detected from the original image that can still be tracked in the final image for bootstrapping, and the locations of corresponding keypoints in the final image.

3.3 Structure from Motion

Structure from Motion (SfM) is the process of obtaining A and B from a set of images.

We solve the Structure from Motion problem using the 8-point Algorithm with RANSAC, using the OpenCV `findFundementalMat` function. This function returns the Fundemental Matrix F, which is converted into the Essential Matrix E as we know the calibration matrix K for the dataset. This is done with the relation $F = K^T * E * K$.

E is then split into the relative translation and rotation T, R. This is done using the SVD in our function `get_relative_pose()`.

$$E = U\Sigma V^T \tag{1}$$

$$\hat{T} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Sigma V^T;\ \hat{R} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{2}$$

$$T = K_2\hat{T};\ R = K_2\hat{R}K_1^{-1} \tag{3}$$

As there are multiple solutions (4 solutions) we just try them all and retain the one that gives the most number of points in front of both the cameras as the best solution.

We therefore are able to obtain the relative transformation between the two frames R, T, and also the 3-D position of the feature correspondences, X.

4 Continuous Visual Odometry

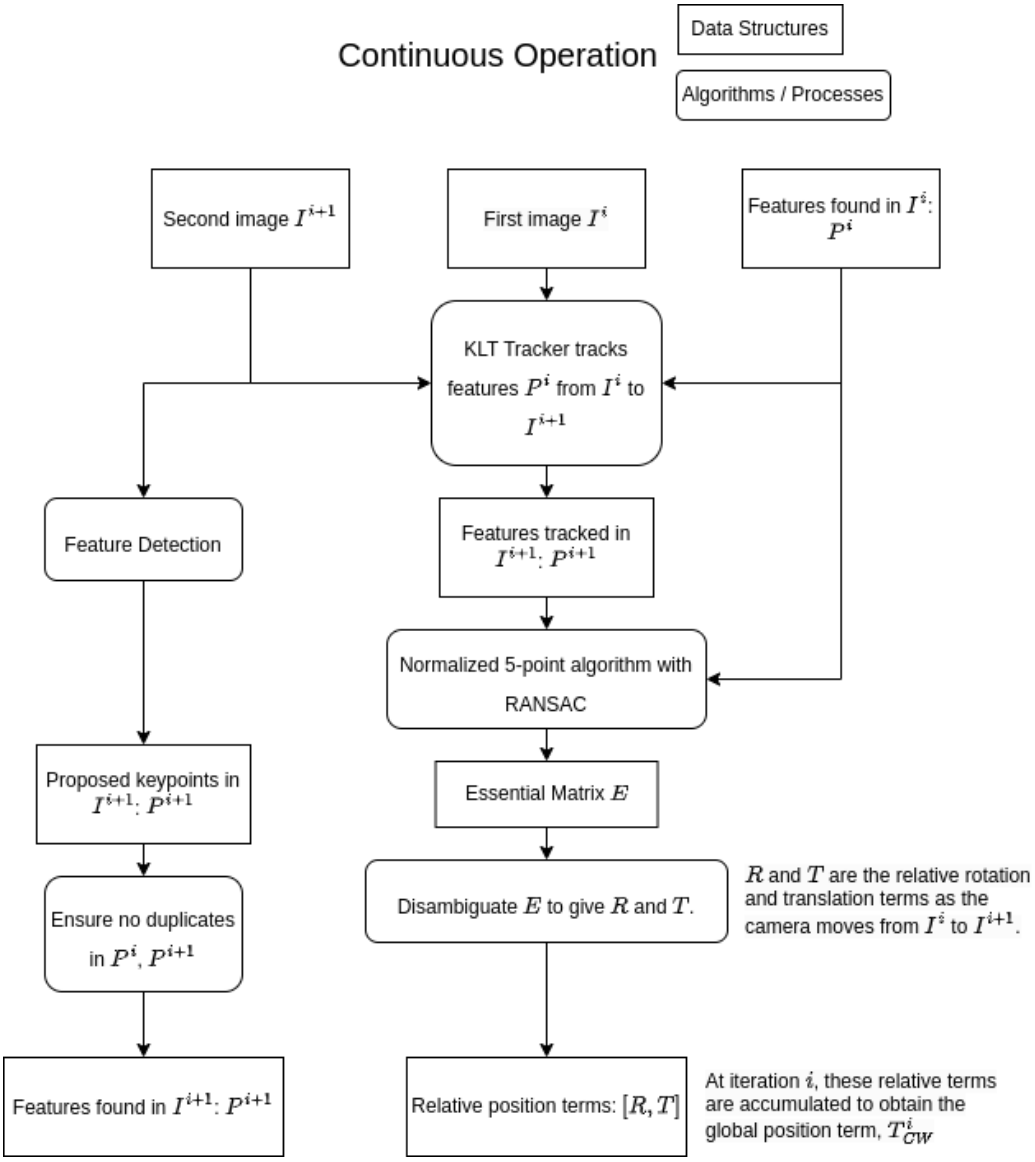

Figure 2: Flowchart of continuous operation

4.1 Markovian design

The continuous VO module for the pipeline works in a Markovian way where processing the current frame only requires the information from the current state and returns the state for the next iteration. The continuous VO accepts the incoming frame and updates the current state and current pose as follows,

$$[S^i, T^i_{WC}] = processFrame(I^i, I^{i-1}, S^{i-1}) \tag{4}$$

By doing this we do not need to store the entire history of past frames in order to handle incoming frames.

4.2 Feature Tracking

KLT is used in continuous VO to track the keypoints detected in the previous image into the current image. KLT in continuous VO is performed twice for each new frame, tracking points of state P and

C respectively. KLT is chosen due to its ability to overcome ambiguity in brute force matching and sub-pixel positions, as mentioned before. Furthermore, the implementation is similar to bootstrapping, except that we return only one array of (x,y) values, describing the locations of keypoints tracked in the new image. The KLT for continuous VO also removes points which are outside the given image by considering the (x,y) values with respect to the image size.

4.3 Pose Estimation

Pose estimation can be done by using 2D-3D correspondences by applying the PnD algorithm. PnP along with RANSAC is used to localize the camera with respect to the existing point cloud that has been generated. RANSAC is used to increase the robustness of the pose estimation. The function returns the pose of the camera and the inlier points information which is used to filter out the outliers from the set of keypoints. Our function `PnPRANSAC()` essentially acts as a wrapper for the `OpenCV` function `solvePnPRansac()`. The rotation matrix is converted to a rotation vector post this and all inputs and outputs to the PnP RANSAC module are done to conform with the datatypes defined in our Markovian design.

Considering the poor performance of PnP due to imperfect triangulations and presence of many outliers, we tried to use the five-point algorithm for determining the pose of the camera using 2D-2D correspondences determined by KLT. Within the algorithm we first estimate the Essential Matrix using the `OpenCV` function `findEssentialMatrix()` from the correspondences. Then we get the relative pose of the camera using the method followed in Bootstrapping as discussed in Section 3.3. This method showed considerably better performance than using PnP with RANSAC and hence was considered for the final submission.

4.4 Triangulation of New Landmarks

The continuous VO pipeline removes those 2D-3D correspondences where the 2D keypoints are not tracked from previous image to current image or the PnP considers the correspondences as outliers. Because of this the number of correspondences shrink. In order to ensure enough number of correspondences for PnP to work properly, we regularly triangulate new points and add their respective 2D image points from the image they are triangulated from.

For triangulation, we first track candidate keypoints C^{i-1} to form an initial guess for C^{i}. We then remove the points which lost tracking from previous frames. We also filter out the points which are already triangulated and are used in 2D-3D correspondences. Once we have the set of points which are tracked and not triangulated before, we try to triangulate these points. Once all points are triangulated, we evaluate the baseline and the reprojection errors for each triangulation and only consider the points where the baseline is big enough and reprojection errors are below a particular threshold. We also filter out points which are triangulated behind the camera. We then add these triangulated points and their respective 2D keypoints in the current image to the set of 2D-3D correspondences and remove them from the list of candidate keypoints.

On using 2D-2D correspondences for pose estimation as discussed in Section 4.3, we do not need the 3D points for pose estimation but we do triangulate points for creating the pointcloud.

5 Video Recordings & Code

The link to the video recording is here: https://polybox.ethz.ch/index.php/s/089LXzUeORKMswT. The link to the GitHub repo is here: https://github.com/JasperTan97/VAMR_Lost_in_the_jungle/tree/main. There are instructions on how to setup the environment and run the code in the readme.

5.0.1 Parking Dataset

Our VO tracks the pose of the camera locally, with a slight drift in the positive y-axis direction (positive Z in the world frame). Throughout the video, the VO tracks about 4000-5000 keypoints between frames. The constant drift begins when cars appear in the frame, and ceases near the end when there were no more cars. There are multiple sources of errors; the cars are reflective, are curved and look similar from different angles. While KLT tracking is able to warp points that have been transformed (in an

affine way), the cars are very close to the camera and do not have spatial coherence. This may confuse the KLT tracker, and drop or create errant matches, leading to drift.

5.0.2 Malaga Dataset

The VO pipeline handles the Malaga dataset well, tracking the camera pose on the straight roads and curves locally. However, during frame 1145, the rotation estimated was 90 degrees on a straight road. We are unable to discern the cause of this error, as the frame is similar as any other feature. This error occurs after multiple reruns of the Malaga dataset, and hence does not come about from the non-determinism of RANSAC. One possible reason might be that the camera moved through the featureless intersection too quickly, and the keypoints tracked are in far away objects that do not change much between frame. This may cause large errors in estimated rotation when computing small differences in matched feature positions. (Similar u1, u2 and v1, v2 values can cause the Q matrix to lose more ranks).

Other than the above problem, the rest of the estimated poses follow similar trajectories to the ground truth. Throughout the VO, about 4000 keypoints were tracked between frames.

5.0.3 KITTI Dataset

Similar results are found in the KITTI dataset, and good local pose estimation trajectories are found. Globally, the trajectory is also very close to the ground truth, and the repeated road from frames 1330-1600 are almost overlapping without using loop closure. From frame 2370, the camera stopped moving, allowing for cars to pass by the intersection. This caused some errors in the pose estimation, as cars close to the camera are moving perpendicular to it. These large occlusions violate the basic assumption of VO, that there is a dominance of static scenes over moving objects. The consequence of the violation is the pose estimation confusion, creating a spaghetti like trajectory on the location the camera was stationary. Fortunately, the pose estimation carried on well after the camera started to move again, when there are no more dynamic objects dominating the image.

5.0.4 Parking dataset with 3D to 2D correspondence

As our approach does not generate a pointcloud, we also implement some code for one to be generated.

The approach takes a buffer of the past few images and the keypoints detected at each image. The keypoints corresponding to the last image are tracked using KLT to the most recent image, and the relevant keypoints are then triangulated, similar to the process in bootstrapping.

Shown in the bottom right is a visualization of this process.

The past poses are blue arrows and the current pose is a large yellow arrow.

The pose of the last image, the "base pose", is a red arrow, and the pose calculated from triangulating poses from the current pose and base pose as a large purple arrow. Points in the pointcloud are green crosses.

We see that the pose error from 3D triangulation is not too far off from the base pose, so long there are sufficient keypoints for triangulation.

Keypoints for triangulation are filtered based on two factors: if they are projected in front of the camera and if there is a large enough baseline between the two poses where they are observed. We observe that as time goes on, the point cloud starts to drift to the right. This results in the baseline for each point shrinking, reducing the number of valid points for triangulation, which thus degrades the quality of the point cloud, and subsequently the poses from PnPransac.

6 Author Contributions

Team Member	Contribution
Abhiram	OpenCV implementations of exercises, PnP RANSAC, bootstrapping pseudocode
Adarsh	Triangulating new landmarks, Pose detection using 2D-2D correspondences
Jasper	Feature detection(SIFT), Feature tracking(KLT), data visualisation
Tian Yi	Bootstrapping implementation, Pose disambiguation, Feature detection(Harris)

References

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] Davide Scaramuzza. *Vision Algorithms for Mobile Robotics*. URL: https://rpg.ifi.uzh.ch/teaching.html.