University of Southern Denmark

Project in Advanced Robotics

master of science in engineering (robot systems)

Visual SLAM for autonomous driving

Anders Flemming Johannsen
anjo319@student.sdu.dk

Gian Paolo Currà
gicur22@student.sdu.dk

Lucas Anthony Thurnherr
luthu19@student.sdu.dk

Xuze Cai
xucai19@student.sdu.dk

Project period: Spring 2023

Abstract

An important field in robotics and computer vision is pose localization from visual cues. Visual Simultaneous Localization and Mapping, also known as VSLAM, is a prominent method for solving this problem. As the name suggests, this method allows a moving robot or vehicle to map its surroundings and estimate its position. This is often done in real time, leading to this being a popular solution for this problem. A method of VSLAM is developed and used on the KITTI dataset to estimate a path along a route.

This report discusses the methods used to achieve a VSLAM solution, those being Visual Odometry, Bundle Adjustment, and Loop Closure.

Visual Odometry covers keypoint generation and tracking, disparity maps from stereo vision, triangulation via the Direct Linear Transform method, pose and motion estimation with least squares optimization, and outlier removal via RANSAC.

The estimated path of the visual odometry method accumulates drift during the course of a trip, which eventually makes the estimated path unusable as is. Optimization is necessary in the form of bundle adjustment. Data sorting is done in order for the Bundle Adjustment method to read the estimations of the map and camera poses. The bundle adjustment method uses sparsity to lighten computational cost. The bundle adjustment solution does not work due to what is hypothesized to be an error in data sorting between the visual odometry step and the bundle adjustment.

Loop detection with the Bag of Words method is covered, pre-training a vocabulary of descriptors, which can detect when previously traveled paths intersect with a new frame. The loop closure method re-sorts the current estimation of the pose and map uses this new anchor point to correct the bundle adjustment method for the drift within the loop. This was not developed fully, as it would not have been implementable without a working bundle adjustment solution.

This report delves into the specifics of these methods, and discusses what can be improved upon.

Contents

Chapter 1

Introduction

In the domain of mobile autonomous robots, the requirement of being able to operate in unknown environments arises.

Witnessing a lot of issues, these robots need the integration of navigation and mapping capabilities. Visual Simultaneous Localization and Mapping (VSLAM) is a common solution for enabling robots to perceive their surroundings while gaining information about their own position within the environment.

The purpose of this paper is to document the development of a VSLAM implementation for mapping for use in autonomous driving. To perform the experiment, the KITTI dataset is being used. This dataset consists of a long series of numbered images that correspond to the motion of a car navigating with two cameras onboard. In this way, the motion of the cameras is being simulated in an environment. The goal of this study is to map the path of the subject using only the images captured from the stereo cameras and then compare the results with the ground truth GPS position available in the dataset.

To accomplish the robot's pose estimation and mapping of the environment, Visual Odometry (VO) is being performed in order to obtain translations and orientations of the subject frame-to-frame. Feature detection and tracking are obtained in order to have fixed points in the views that can be exploited to estimate the motion. 3D depth estimation is being performed via triangulation in order to get the 3D positions of the features. As VO is estimating the path, an error in the pose accumulates over time, forcing the use of a correction method, in this paper Bundle Adjustment (BA), which refines and optimizes the data by using multiple viewpoints which capture the same distinct features.

As the final part of this experiment, Loop Detection and Closure are needed to recognize previously visited points in the path, which can be used to correct for drift related to the mapping algorithm. In this way, frames which contain points of interest which are previously seen can be used as anchor points to further refine and optimize the path, eliminating error accumulation caused by the VO method in between the start and end of the loop. This implementation has been executed by using Python 3.9.13, OpenCV, and the KITTI dataset.
This report discusses the implemented algorithms, along with prospective future applications, by means of a project which aims at realizing a comprehensive Visual Simultaneous Localization and Mapping (VSLAM) implementation.

Chapter 2

Methods

2.1 Visual odometry

Visual odometry in computer vision is the process of determining the position and orientation of the cameras by analyzing the associated images recorded during the motion. Typically, the Visual Odometry algorithm is expected to be executed continuously during motion. However, this project utilizes the KITTI dataset, which is comprised of a sequence of images captured during a registration process, diminishing the priority of optimizing the algorithm to the extent of real-time execution. The visual odometry solution in this paper follows the pipeline seen in figure 2.1. An overview of the pipeline used in this paper starts by using a left image to generate keypoints. These keypoints are then tracked in a sequence of images. The left image's tracked keypoints are compared to a disparity map generated using the left and right images, by comparing the tracked keypoints generated in the left images with the disparity map, the corresponding right image keypoints are found, by using this method one assures that the tracked keypoints in both images match the same objects in the environment. By tracking the same objects in the environment using cameras from different perspective, one can determine the position of objects in environment using triangulation. By knowing how the positional changes of objects in the environment, the motion of the cameras, meaning the car, can be estimated. The following section elaborates each step of the pipeline in further details.

2.1.1 Keypoint Generation

Keypoints are spatial locations or image locations that define points of interest (features) or things that stand out in an image. Keypoints are scale and rotation invariant, meaning keypoints in an image which have been modified with rotations, translation or shrinking, can be found on the original image. This allows for comparing between images as long as identical keypoints are present, which is useful as changes happening to keypoints can be used to estimate the overall change between images. Keypoints can then be used in a variety of computer vision applications such as matching, tracking etc. This paper's implementation of keypoints generates them using a modified version of OpenCV's FAST features algorithms. The FAST features algorithms is an corner finding algorithm based on intensities of pixels in an image. So let I_p be the intensity of pixel in an image with the pixel coordinates p and let t be a threshold value. By checking the intensities of the neighboring pixels in a circle of radius 3 from pixel p see figure 2.2. The pixel I_p is determined to be a keypoint if and only if 12 (default openCV parameters) contiguous pixels are of intensities:

$$I_p \pm t \tag{2.1}$$

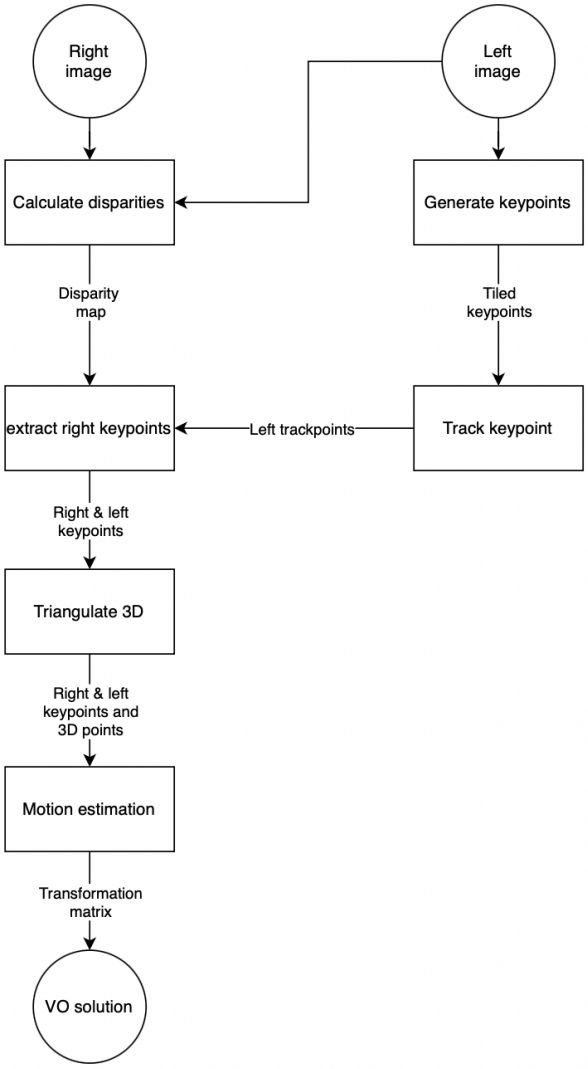


Figure 2.1: Pipeline for this papers visual odometry solution

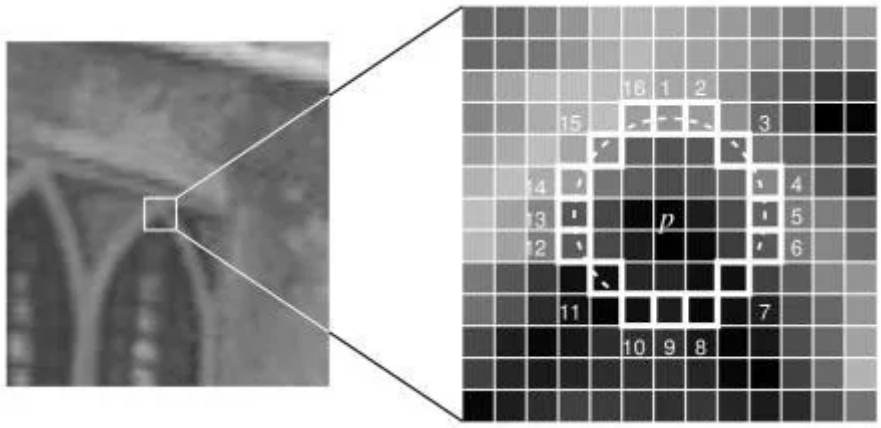


Figure 2.2: OpenCV's FAST algorithm checking neighboring pixels [1]

This implies that the pixel p is a corner, as surrounding pixels are either brighter or darker than p. The FAST algorithm is computationally optimized by first only looking at pixels 1,5,9,13 see figure 2.2. If at least 3/4 pixels do not fulfill the critiria of $I_p \pm t$, then the criteria of 12 contiguous pixels can not be fulfilled, thus reducing computational complexity as the remaining pixels are not checked. The drawback of using this method is that descriptors for keypoints are not found, which is useful for loop detection. However, this papers implementation of VO does not make use of descriptors. When compared to alternatives such as SIFT and SURF, FAST tends to be more computationally efficient and more robust [7]. Note though the comparison is between SURF, SIFT and ORB, ORB is essentially FAST fused with BREIF descriptors [8].
To better fit the use case of this papers implementation of VO, additional modifications are made in the keypoint generation process. The problem being that keypoints generated tend to group in areas where contrast is high or lots of edges occur, where as the rest of the image is ignored. This can be a problem if these high contrast areas or areas with lots of edges areas are out of frame from one frame to another, resulting in the tracking losing a massive amount of keypoints from one frame to another. To solve this problem, the implementation makes use of a technique called tiling. This technique divides the images into smaller windows. Each window then generates a set of a limited amount of keypoints. All these windows are then recombined into the final image. The resulting image's keypoint are thus more evenly distributed. Keypoint generation without tiling can be seen in figure 2.3. Keypoint generation with tiling can seen in figure 2.4. The figures show that by using tiling the quality of keypoints improve as only the best keypoints in each tile remain. Keypoints are also more evenly distributed throughout the image as each tile is limited to a set amount of keypoints. This results in less computational complexity as only the best keypoints are used further in the VO pipeline.

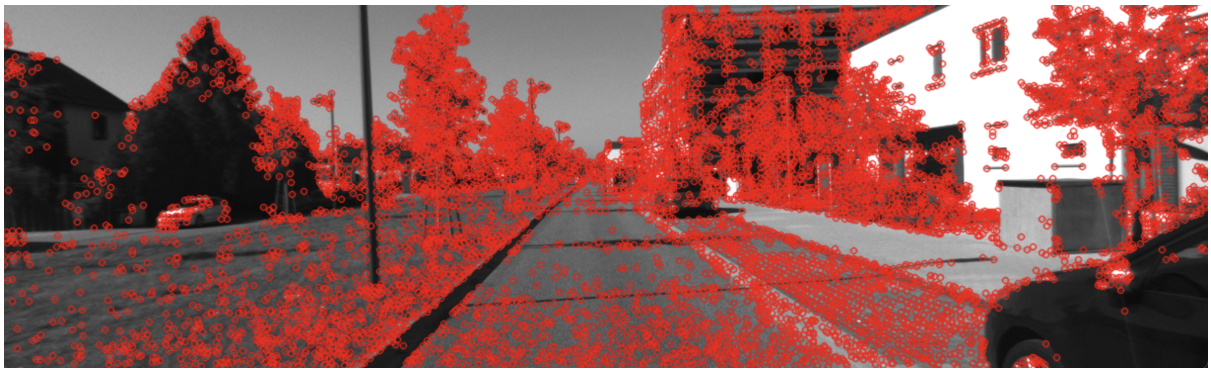


Figure 2.3: Keypoint generation without tiles

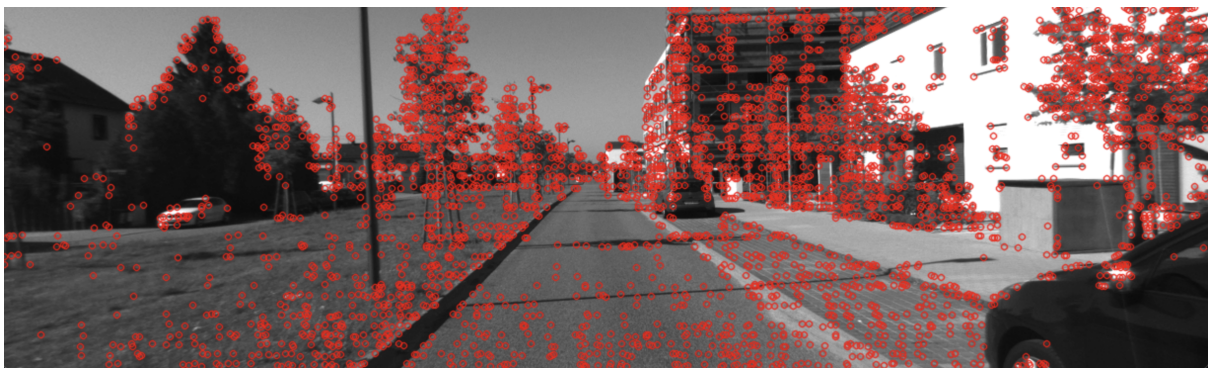


Figure 2.4: Keypoint generation with tiles

2.1.2 Tracking

The tracking method in this paper recieves the keypoints generated during keypoint generation. These keypoints are tracked using a method by the name of optical flow. Optical flow is based on the optical flow equations which is derived under certain assumptions:

Assumption 1. *The pixel intensities of an object do not change between consecutive frames.*

Assumption 2. *Changes in images are small*

Assumption 3. *Neighboring pixels have similar motions.*

Let $I(x, y, t)$ be a pixel in the first image, where x, y are pixel coordinates and t is time. It moves a distance $(\delta x, \delta y)$ in the time δt. Under assumption 1 the following is true

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{2.2}$$

By using the property of Taylor series expansion $f(x + \delta x) = f(x) + \frac{\delta f}{\delta x}\delta x + \frac{\delta^2 f}{\delta x^2}\frac{\delta x^2}{2!} + ... + \frac{\delta^n f}{\delta x^n}\frac{\delta x^n}{n!}$ and assumption 2. The higher order terms of the Taylor series expansion can be neglected as the higher order $\frac{\delta x^n}{n!}$ terms become almost zero. Thus the equation can be written as:

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\delta I}{\delta x}\delta x + \frac{\delta I}{\delta y}\delta y + \frac{\delta I}{\delta t}\delta t \tag{2.3}$$

By subtracting (2.2) from (2.3):

$$I_x\delta x + I_y\delta y + I_t\delta t = 0 \tag{2.4}$$

Where $I_x = \frac{\delta I}{\delta x}$ and I_y, I_t being their respective subscripts. Then by dividing by δt and taking the limit as $\delta -> 0$:

$$I_x\frac{\delta x}{\delta t} + I_y\frac{\delta y}{\delta t} + I_t = 0 \tag{2.5}$$

$$I_x u + I_y v + I_t = 0 \tag{2.6}$$

Where $u = \frac{\delta x}{\delta t}$ and $v = \frac{\delta y}{\delta t}$. Equation 2.6 is the optical flow equation, in which u, v are unknowns.

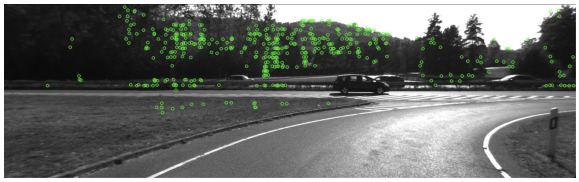
To solve the optical flow equation this paper uses OpenCV's sparse optical flow implementation of "calcOpticalFlowPyrLK" which is based Lucas-Kanade's algorithm. Lucas Kanake's algorithm assumes that the 3x3 neighboring pixels about the keypoint have similar motion. This results in the optical flow equation 2.6 becoming a optimization problem with 9 equation 2 unknowns problem.

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ ... & ... \\ I_x(n,n) & I_y(n,n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ ... \\ I_t(n,n) \end{bmatrix} \tag{2.7}$$

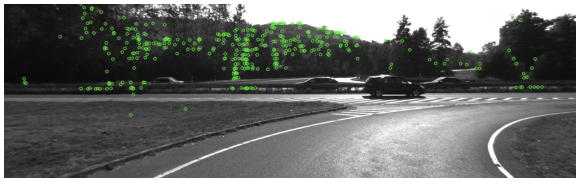
where k, l denotes the positions of neighboring pixels about the keypoint. This problem is solved using least squares, finally arriving at:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \Sigma_i I_x^2 & \Sigma_i I_x I_y \\ \Sigma_i I_x I_y & \Sigma I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\Sigma_i I_x I_t \\ -\Sigma_i I_y I_t \end{bmatrix} \tag{2.8}$$

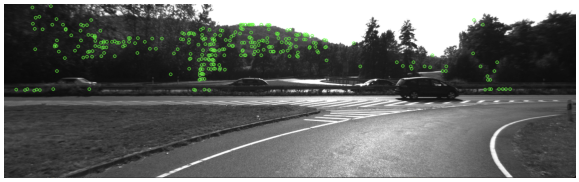
Having attained the motion vector $\begin{bmatrix} u \\ v \end{bmatrix}$ The motion of keypoints can be tracked from one frame to another, or in a sequence by rerunning the algorithm using tracked keypoints from previous run. The shortcomings of this method is poor handling of abrupt movement, sudden changes in lighting, tracking occluded objects and accumulation of error in longer runs. However for the use case in this paper, abrupt movement changes and sudden changes in lighting are rare due to the KITTI dataset. The drawback of tracking occluded objects and tracking objects in longer runs is insignificant as objects in the camera frame fade out of view quickly due to the motion of the car. Thus this implementation is chosen. A sequence tracked using the KITTI sequence 1 dataset can be seen in figure 2.5.



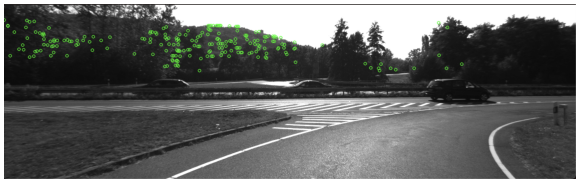
(a) Time sequence 0



(b) Time sequence 1



(c) Time sequence 3



(d) Time sequence 4

Figure 2.5: A sequence of tracked points using optical flow

2.2 Stereo Vision

Stereo Vision is a study field that uses methods to extract 3D information while having as a main structure two different points of view of the same scene. The procedures can be divided into two categories: Dense Stereo and Sparse Stereo.

The main difference between these methods is the way the information is compared between the right and left camera views. Dense Stereo searches for a matching pixel/sub-pixel in the other view, while the Sparse case works on extracting features and looking for a match between them [2].

This project presents the main aspect of a Sparse Stereo approach because, as can be seen in this section 2.1, the first nodes of the pipeline are characterized by feature detection, and then tracking. By this, and the next steps, it will be possible to map the cars movements in the environment.

Sparse Stereo

The necessary step of a Sparse Stereo approach is to determine a 3D point from a matching object between the two Stereo images. To accomplish that, a couple of methods were brought under examination: Descriptor Matching and Disparity Maps. These two algorithms would be applied to the i-stereo frames which portray the same scene at the same moment and used on the keypoints detected in the first steps of the pipeline.

Disparity Map

A disparity map, the method chosen in this study, represents the pixel-wise difference in a position between corresponding points in a pair of stereo images taken from slightly different viewpoints. In this project, the KITTY dataset is used, which contains arrays of images from right and left cameras which share the same plane and are distanced by a constant horizontal displacement. For this reason, the

hypothesis of a rectified system can be assumed, and a disparity is computed for every pair of images. The applied method, which is necessary to find the disparity maps, is Semi-Global Block Matching (SGBM). This disparity can be used to determine the right camera position of keypoints in the left camera, which have been tracked from a previous frame. This gives 2D points in 2 views, which can be used for triangulation.

Disparity estimation

The algorithms for disparity estimation are classified into two main categories: Local and Global methods.

The main difference between them is the number of pixels considered at a time; if look at the correspondences pixel by pixel with their neighboring patches (Local methods) or look at the information contained in the whole image or sub-images (Global methods).

SGBM belongs to the Global Methods because the main function is to use blocks of the stereo images and compare them, calculating a cost function. The cost itself represents how similar the two blocks are. It is obtained by comparing the brightness or colors of the pixels in the blocks. With this, it finds the block in the right image that looks the most like the block in the left image.

As the second step, thanks to the cost function, the algorithm pursues the matching with the blocks that have the lowest cost function (most similar block). By computing the difference in position between the block and the ones, it outputs the disparity value for that block.

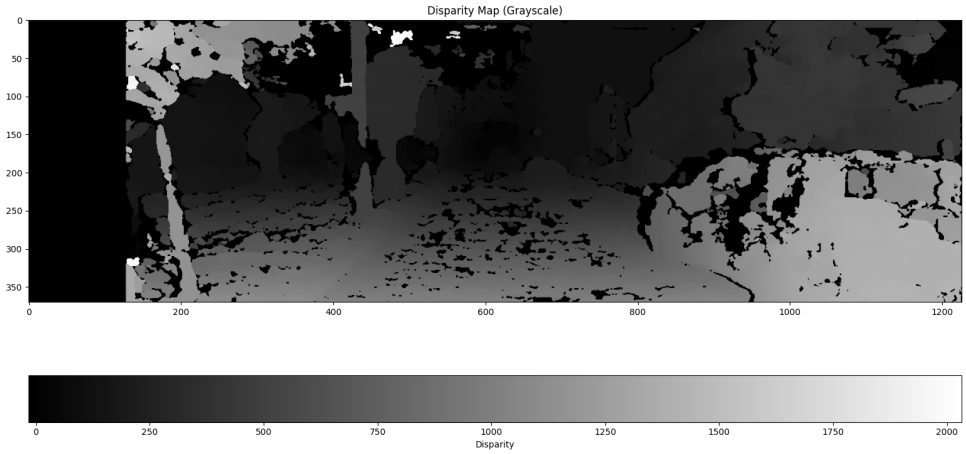


Figure 2.6: SGBM Disparity Map

Descriptor Matching

The other method that is descriptor matching. Driven by the weakness of 2D keypoints to withstand rotation or deformation, descriptors are objects which maintain the same amount of information as keypoints, but are robust against geometric transformations. Descriptors, are compact representations of keypoints that contain the local information around the keypoint. They encode the appearance and the arrangement of the neighbor pixels around each keypoint, making it possible to match keypoints across different non-consecutive images. Descriptor encoding can be obtained with many algorithms that are divided into two categories:

- **Gradient-Based Descriptors**, which capture the image's local intensity changes and gradient directions (SIFT, SURF, DAISY);

- **Binary Descriptors**, which represent local image patches as binary strings (BRIEF, ORB, BRISK, FREAK, LATCH).

	Scale	Rotation	Viewpoint	Lightness
SIFT	x	x	x	x
SURF	x	x	x	x
DAISY			x	x
BRIEF				x
ORB		x		x
BRISK	x	x		x
FREAK	x	x	x	x
LATCH		x		x

Table 2.1: Invariance table [3]

In table 2.1 it is stated how the descriptors algorithms are listed and filtered by their invariances.

Oriented FAST and Rotated BRIEF (**ORB**) was the first approach in this study. As a matter of fact, the descriptor principle is the same as BRIEF but related to a keypoint generated by the FAST feature.

Although due to the binary nature of BRIEF, it is normally rotation sensitive[1], but the ORB algorithm takes care of this aspect by adding orientation information after FAST.

This step is made by searching the most significant direction of local intensity changes around the keypoint and by taking into account the gradient information: magnitude and direction[2]. Once the dominant orientation is determined for each keypoint, every patch around the keypoint is aligned along with the calculated orientation. By doing this, every aligned patch is consistent with the dominant orientation so the invariance to rotation for the descriptor is obtained. [9]

2.2.1 Triangulation

The triangulation procedure is done by using the calculations necessary to find the position of a point in space, given its position in two different images taken with cameras with known calibrations and poses [5].

Assuming that, for every 3D point analyzed by two points of view, there are two different rays that connect the 3D point and the center of the camera (back-projected optical rays). Theoretically, as it can be seen in figure 2.7 the 3D point lies at the intersection between the two optical rays, but in reality the two rays usually never intersect. For this reason, an optimal 3D point estimation is needed.

2.2.2 Direct Linear Transform (DLT)

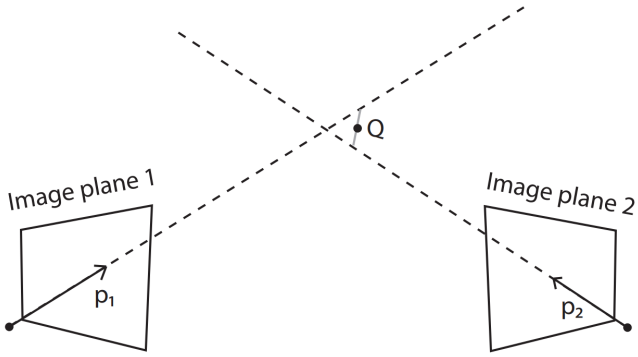
The Direct Linear Transform algorithm is a method that uses the corresponding image points from two or more cameras. In this way, it can set up an over-determined linear system (two equations per camera) based on the camera projection model that can be solved using methods like the Singular Value Decomposition (SVD) or least-squares minimization.

After rewriting the 2D and 3D points in homogeneous coordinates, the triangulation problem can be resumed with equations 2.9 and 2.10, where:

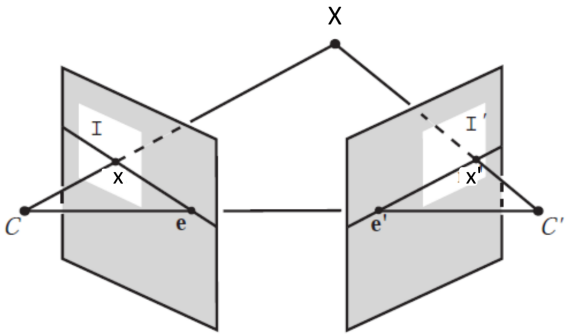
$$\vec{q_{left}} = P_{left}\vec{Q} \tag{2.9}$$

[1] Binary strings are generated by comparing pixel intensities within the patch using a set of predefined pairs of pixel locations.

[2] The gradient magnitude indicates the strength of the intensity change, while the gradient direction gives the angle at which the change occurs.



(a) Midpoint Triangulation



(b) DLT Solution [10]

Figure 2.7: Triangulation Methods

$$q_{r\vec{i}ght} = P_{right}\vec{Q} \tag{2.10}$$

where q is the 2D pixel coordinates in the given frame, P is the projection matrix, and Q the unknown 3D coordinates. Given this, it is possible to set up the overdetermined system by using the cross product between q and $P\dot{Q}$, which is zero for the parallelism between them:

$$\begin{bmatrix} q_{x,left} \\ q_{y,left} \\ 1 \end{bmatrix} \times \begin{bmatrix} \vec{p}_1\vec{X} \\ \vec{p}_2\vec{X} \\ \vec{p}_3\vec{X} \end{bmatrix} = \begin{bmatrix} q_{y,left}\vec{p}_3\vec{X} - \vec{p}_2\vec{X} \\ \vec{p}_1\vec{X} - q_{x,left}\vec{p}_3\vec{X} \\ q_{x,left}\vec{p}_2\vec{X} - q_{y,left}\vec{p}_1\vec{X} \end{bmatrix} = \begin{bmatrix} v_1\vec{p}_3 - \vec{p}_2 \\ \vec{p}_1 - q_{xleft}\vec{p}_3 \\ q_{x,left}\vec{p}_2 - q_{y,left}\vec{p}_1 \end{bmatrix} \vec{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} q_{x,right} \\ q_{y,right} \\ 1 \end{bmatrix} \times \begin{bmatrix} \vec{p}_1\vec{X} \\ \vec{p}_2\vec{X} \\ \vec{p}_3\vec{X} \end{bmatrix} = \begin{bmatrix} q_{y,right}\vec{p}_3\vec{X} - \vec{p}_2\vec{X} \\ \vec{p}_1\vec{X} - q_{x,right}\vec{p}_3\vec{X} \\ q_{x,right}\vec{p}_2\vec{X} - q_{y,right}\vec{p}_1\vec{X} \end{bmatrix} = \begin{bmatrix} v_1\vec{p}_3 - \vec{p}_2 \\ \vec{p}_1 - q_{xright}\vec{p}_3 \\ q_{x,right}\vec{p}_2 - q_{y,right}\vec{p}_1 \end{bmatrix} \vec{X} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Then the system can be rewritten as:

$$A\vec{X} = \begin{bmatrix} q_{y,left}\vec{p}_3 - \vec{p}_2 \\ \vec{p}_1 - q_{x,left}\vec{p}_3 \\ q_{y,right}\vec{p_3} - \vec{p}_2 \\ \vec{p}_1 - q_{x,right}\vec{p}_3 \end{bmatrix} \vec{X} = 0$$

And then decomposed as Singular Value Decomposition which factorizes the matrix A into three

matrices:

$$A = USV^T$$

where U, V^T are orthogonal and S diagonal. To find the solution of $A \cdot P = 0$ with error minimized, since it is a homogeneous system, any scalar multiple of P is valid. The eigenvalues of S are in descending positional and value order. The last eigenvalue, corresponding to the last column of V is the one that minimizes the error.

2.3 Motion estimation

Motion estimation aims to find the transformation matrix from the camera pose in image I_{k-1} to image I_k. There are three different methods to compute motion estimation, depending on whether the features are given as 3D or 2D points. Motion estimation that uses 2D-to-2D and 3D-to-2D minimizes reprojection error and 3D-to-3D minimizes feature position error. Based on this, 3D-to-2D is preferable because reprojection errors are more reliable than 3D-to-3D [4]. The features f_{k-1} are given in 3D, and f_k is in 2D. 3D-to-2D is a PnP problem, and the minimal case needs three point-correspondences. In a monocular case, correspondences for at least three images are needed because the 3D point for f_{K-1} needs to be triangulated using f_{K-2}. In a stereo case, f_{k-1} can be found by triangulating from the other camera in the pair. The points needed to find the Transformation T_k is computed from Q_{k-1}, q_{k-1}, which is the 3D and 2D points from time-step k-1, and Q_k, q_k from time step k. The ideal T_k is one with the least reprojection residuals calculated for both time steps. The formulation for computing the transformation T_k that minimized the reprojection residuals is:

$$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = \arg\min_{T_k} \sum_i \| p_k^i - \hat{p}_{k-1}^i \|^2$$

Where $\hat{p}_{k-1}^i$ is the reprojection of the 3D point Q_{k-1}^i into image I_k according to the transformation T_k, and p_k^i is the original 2D point [4]. Feature matching is prone to errors that create point outliers in the dataset, which can impair successful motion estimation. Outliers distort the camera pose when present, leading to a significant compounding error over time.

2.3.1 RANSAC

Random sample consensus (RANSAC) is an iterative method for removing outliers from a dataset. Each iteration takes a subset of data, creates a model based on that data, and tests the model with the entire dataset. The model with the best score can then be used to remove outliers from the dataset. The number of iterations needed for RANSAC is dependent on the expected number of outliers in the dataset, and early termination conditions can be implemented to improve its runtime. This paper uses RANSAC to find a transformation T_k with minimal reprojection residuals. Calculating the camera pose requires at least three point correspondences. During each iteration of the RANSAC loop, the transformation is found by setting it up as a least squares problem and solving it with the Levenberg-Marquard method, as described in section 2.3.2. The RANSAC loop runs for 100 iterations, with an early termination condition that triggers if a transformation with smaller reprojection residuals than the current best estimation is not found for five iterations. Once a pose has been estimated at the end of the RANSAC loop, all points are reprojected onto the new camera pose, and all points which have a reprojection error above a fixed threshold can be discarded as outliers, which was not added to the final solution.

2.3.2 Levenberg-Marquardt Least Squares Optimization

In this project, the implementation to find the transformation matrix is treated as a least squares problem, formulated as seen in equation 2.11.

$$x^* = \arg\min_x \frac{1}{2}\sum_{i=1}^{m}(f_i(x))^2 \rightarrow x^* = \arg\min_x \frac{1}{2}f(x)^T f(x) \tag{2.11}$$

The minimization method used to solve the pose estimation is the Levenberg-Marquardt method, which is a combination of the gradient descent method, and the Gauss-Newton method. For gradient descent, a Taylor expansion can be done if the gradient in local or global minima is differentiable, as seen in equation 2.12.

$$F(x+h) = F(x) + h^T g + \frac{1}{2}h^T H h + \mathcal{O}(\|h\|^3) \tag{2.12}$$

h is the step direction, which should be $h = -F^{'(x)}$ for gradient descent. g and H are defined in equations 2.13 and 2.14.

$$g = \nabla F(x) = \begin{bmatrix} \frac{\partial F}{\partial x_1}(x) \\ \vdots \\ \frac{\partial F}{\partial x_n}(x) \end{bmatrix} \tag{2.13}$$

$$H = \nabla^2 F(x) = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1 \partial x_1}(x) & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial^2 F}{\partial x_n \partial x_n}(x) \end{bmatrix} \tag{2.14}$$

The resulting calculated step is seen in equation 2.15.

$$x_k = x_{k-1} - \lambda \nabla F(x_{k-1}) \tag{2.15}$$

λ is a step size scalar, and $k-1$ is the last step taken. The problem with gradient descent is that the hessian matrix H is expensive to compute, which is why it is combined with the Gauss-Newton method in the LM method.

The Gauss-Newton method is expressed in equation 2.16, and defined as a least squares problem in equation 2.17.

$$F(x) = \frac{1}{2}(x+1)^2 + \frac{1}{2}(0.2x^2 + x - 1)^2 \tag{2.16}$$

$$f(x) = \begin{bmatrix} x - 1 \\ 0.2x^2 + x - 1 \end{bmatrix} \tag{2.17}$$

This yields equation 2.18.

$$F(x) = \frac{1}{2}f(x)^T f(x) \tag{2.18}$$

For the Gauss-Newton method, the second derivatives are not needed because it estimates a step with equation 2.19.

$$f(x + \Delta x) \approx f(x) + \nabla F(x)\Delta x \tag{2.19}$$

This yields the step, which is seen in equation 2.20.

$$x_k = x_{k-1} - \lambda(\nabla f(x_{k-1})^T \nabla f(x_{k-1}))^{-1} \nabla f(x_{k-1})^T f(x_{k-1}) \tag{2.20}$$

As a least squares problem, it can be extrapolated as seen in equation 2.21 to determine the step direction, which is defined in equation 2.22.

$$\begin{aligned}\frac{1}{2}f(x+\Delta x)^T f(x+\Delta x) &\approx \frac{1}{2}(f(x)+\nabla F(x)\Delta x)^T(f(x)+\nabla F(x)\Delta x)\\ &\rightarrow \nabla F(x)^T(f(x)+\nabla F(x)\Delta x)=0\\ &\rightarrow \nabla F(x)^T f(x) = -\nabla F(x)^T\nabla F(x)\Delta x\\ &\rightarrow \Delta x = (\nabla F(x)^T\nabla F(x))^{-1}\nabla F(x)^T f(x)\end{aligned} \tag{2.21}$$

$$x_k = x_{k-1} - \lambda(\nabla f(x_{k-1})^T\nabla f(x_{k-1}))^{-1}\nabla f(x_{k-1})^T f(x_{k-1}) \tag{2.22}$$

Because the Gauss-Newton method does not require the second derivatives, the calculation cost is reduced, but this method is more sensitive to the initial guess, which can lead to convergence at a local minimum.

The Levenberg-Marquardt method is a combination of gradient descent optimization and Gauss-Newton optimization. It uses the same assumption, as seen in equation 2.19, which yields equation 2.23.

$$\frac{1}{2}f(x+\Delta x)^T f(x+\Delta x)+\alpha\|\Delta x\|^2 \approx \frac{1}{2}(f(x)+\nabla F(x)\Delta x)^T(f(x)+\nabla F(x)\Delta x)+\alpha I\Delta x \tag{2.23}$$

This yields equation 2.24, which defines the optimal step, as seen in equation 2.25.

$$\nabla F(x)^T(f(x)+\nabla F(x)\Delta x)+\alpha I\Delta x = 0 \rightarrow \Delta x = (\nabla F(x)^T\nabla F(x)+\alpha I)^{-1}\nabla F(x)^T f(x) \tag{2.24}$$

If α is 0, the Gauss-Newton method is in use, and if α is 1, the Gradient Descent method is in use. anything in between is an interpolation of the two. This is done to utilize the fast convergence of the Gauss-Newton method early on, and use the stability of the gradient descent method to avoid overshooting near the optimal solution.

$$x_k = x_{k-1} - \lambda(\nabla f(x_{k-1})^T\nabla f(x_{k-1})+\alpha I)^{-1}\nabla f(x_{k-1})^T f(x_{k-1}) \tag{2.25}$$

The optimizer is given a transformation matrix with zeros as an initial guess, and the reprojection error for that pose is found. The pose is then estimated with the Levenberg-Marquardt method. The estimated pose is the least squares estimated function, and the reprojection error is used to determine the gradient. The optimizer then returns the optimal pose after a fixed amount of iterations or the reprojection error is below a fixed threshold. An alternative to the Levenberg-Marquardt method is to use a closed form solution like P3P or DLT to calculate the transformation.

2.3.3 Compounding Error

Since the calculated poses are estimates, the transformations T_k will have minor errors. The uncertainty for the transformation T_k is based on the previous transformation T_{k-1}, which means there will be a compounding error for the vehicle's pose. An example of this compounding error can be seen in Figure 2.8. Techniques like bundle adjustment can help mitigate this compounding error.

2.4 3D Local to Global

When the 3D-points are calculated, their location is local and based in the frame in which they were estimated. Bundle adjustment takes the 3D-points in the world frame as an input, so they need to be

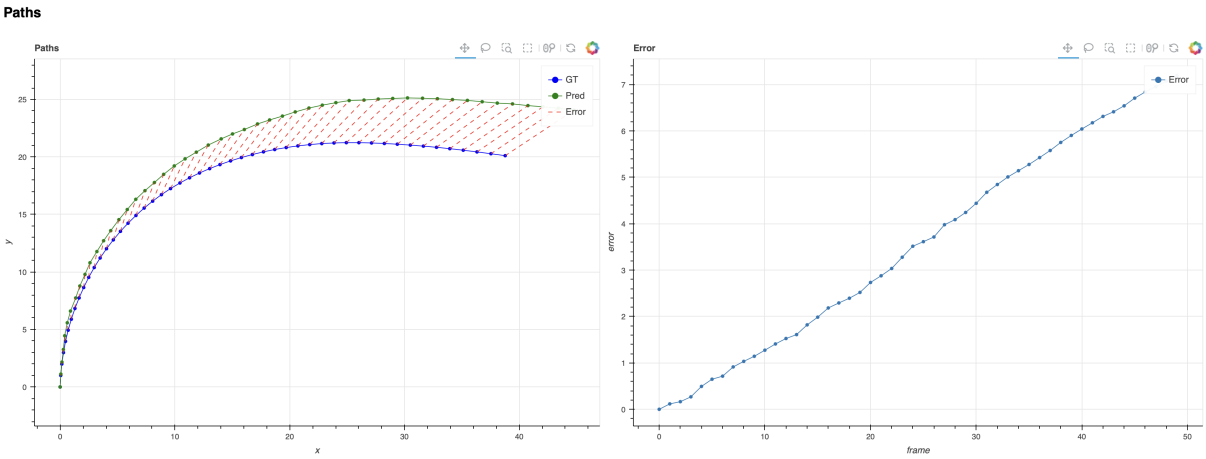


Figure 2.8: Increasing error

converted. Before these values are globalized, they are sorted for outliers, as described in section 2.5.1. This is done at this point because the outlier removal requires the distance of the 3D point from the camera, which is given by the z-axis of the local 3D point. Once the outlier removal is complete, the remaining 3D points can be converted into global points by multiplying the homogenised coordinates with the transfer function of the frames they where produced.

2.5 Data Sorting

In order for the data provided from visual odometry to be efficient and robust, it must be cleaned, as the bundle adjustment is very sensitive to outliers. This is what makes it good for loop closure, as only the pointset of a single frame in the closing of a loop is needed to pull the entire path back on track. Oppositely, this means that it only takes a few outliers to severely decrease the accuracy of the bundle adjustment, if not outright ruin the estimate of the path.
A larger dataset will result in the bundle adjustment running slower, so culling of redundant and less meaningful data is preferred.

2.5.1 Outlier Removal

The removal of outliers is done on the triangulated 3D points in the relative frame of the pose in which they were estimated. The xyz-coordinates in this frame is horizontal position, vertical position, and depth.
As it is not possible to detect points behind the camera view, 3D-coordinates with a negative depth value. Therefore the z-coordinate is just compared to the median of the z-coordinates in a given frame, and removed if it was above that value. The effects of this outlier removal is visualized in figure 2.9b. The average means for each axis is given in table 2.2.

Average Median in dataset	**z**
00	20.815113
07	22.662796

Table 2.2: Average median depth on z-axis across 1001 frames

2.5.2 Simple Sorting Data Treatment for Bundle Adjustment

Bundle adjustment works better the more correspondences there are in the data. It can work if you only have two images to where a given 3D-point is tracked, but it works better if you have more frames where the point is seen. The pose estimation implementation only uses 2 frames per pose in order to do pose tracking. Since the pose estimation creates new key-points for each frame, the points between the current and the other frames cannot be associated as they are. However, strong features generate key-points in the same relative areas in new frames compared to the old ones. This can be exploited in order to fit the data better for bundle adjustment.
The data is compared to see if there is any overlap between the 2D-coordinates tracked from the points in frame i-1 and frame i, and the the 2D-coordinates tracked from the points in frame i and frame i+1. If a match is found, then the track-points from i+1, are added to the data-set of i-1 to i, changing the unique point index and 3D-point to match. An illustration of this matching can be seen in figure 2.12. This means that the same feature ends up being tracked across at least 3 frames. Before this concatenation, the feature could have two different 3D points associated with it due to differences in triangulation and pose estimation. If BA has fewer input parameters covering the same area which it can manipulate, then it will find a better fitting solution.

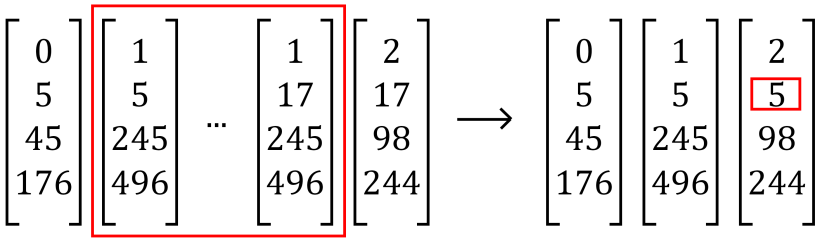


Figure 2.12: Combining two coordinate point sets to a single one by frame and pixel matching

2.5.3 Multi-Frame Sorting Data Treatment for Bundle Adjustment

A way to achieve even more point correspondences between frames is to create an entirely new data-set for bundle adjustment, where key-points are not only generated from image i-1 and tracked to i, but tracked from i-4 all the way through to i. With 2 frames of consecutive tracking, only around 10 percent of the points tracked from i-1 could be tracked to i+1. With this alternative method there should be way more overlap of points between frames. Not all points are able to be tracked though 5 frames, so only the ones that can be tracked in 3 or more are kept. As this method looks back 5 frames, the tracking starts in i=4

The dataset produced by this method is illustrated in figure 2.13.

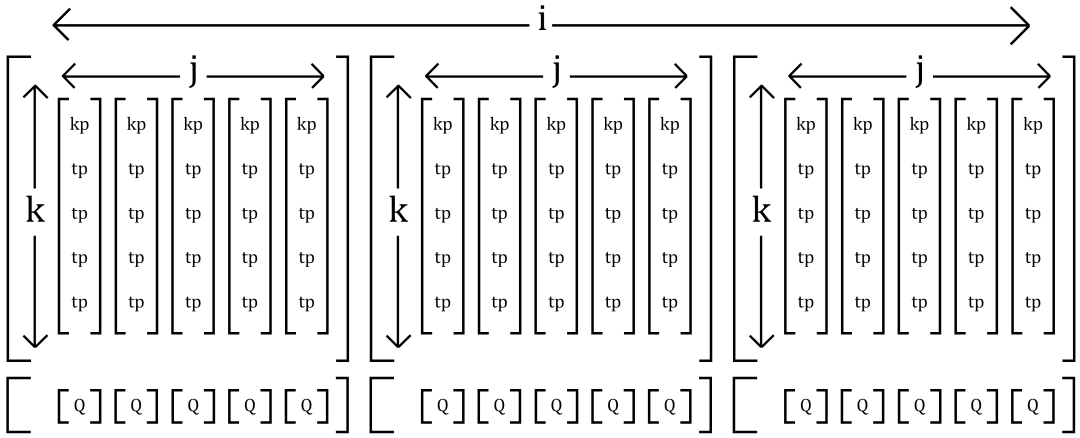


Figure 2.13: Multi-Frame Sorting Illustration. i is current frame index, j is trackpoint sequence in i, k is trackpoint from sequence j in frame $i-4+k$

i denotes the current frame being tracked back from. j denotes the j'th trackpoint sourced from frame i-4. k denotes the frame of the trackpoint in frame *i-4+k*, sourced from frame i-4. This way, when $k = 5$, the trackpoint is at the i'th frame.

With this method, each element of the i'th index can be matched up with the previous and future i indexes by a skew of their offset. An example is illustrated in figure 2.14, where consecutive sequences have trackpoints which match in both frame and pixel coordinates. With this, they can have their unique point index relabelled, and thereby creating a larger sample pool for each unique point for the bundle adjustment to use. The oldest estimated 3D point is then used for all the concatenated trackpoints.

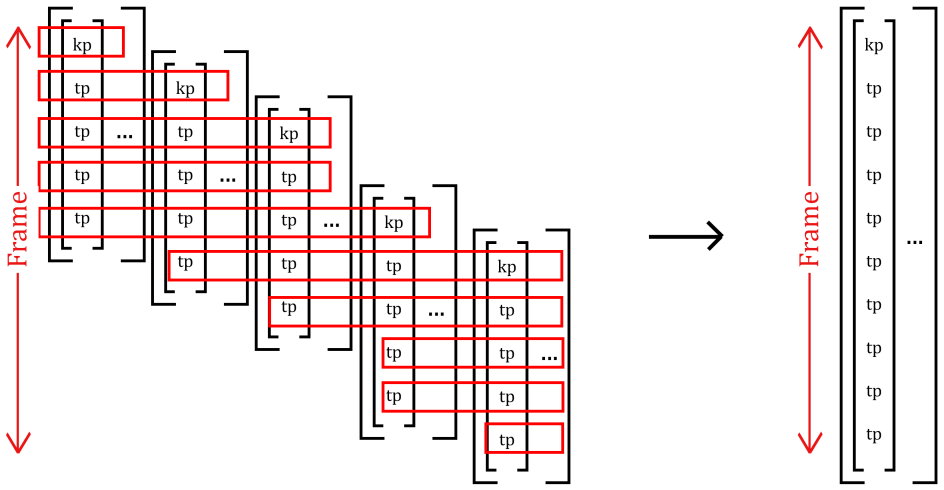
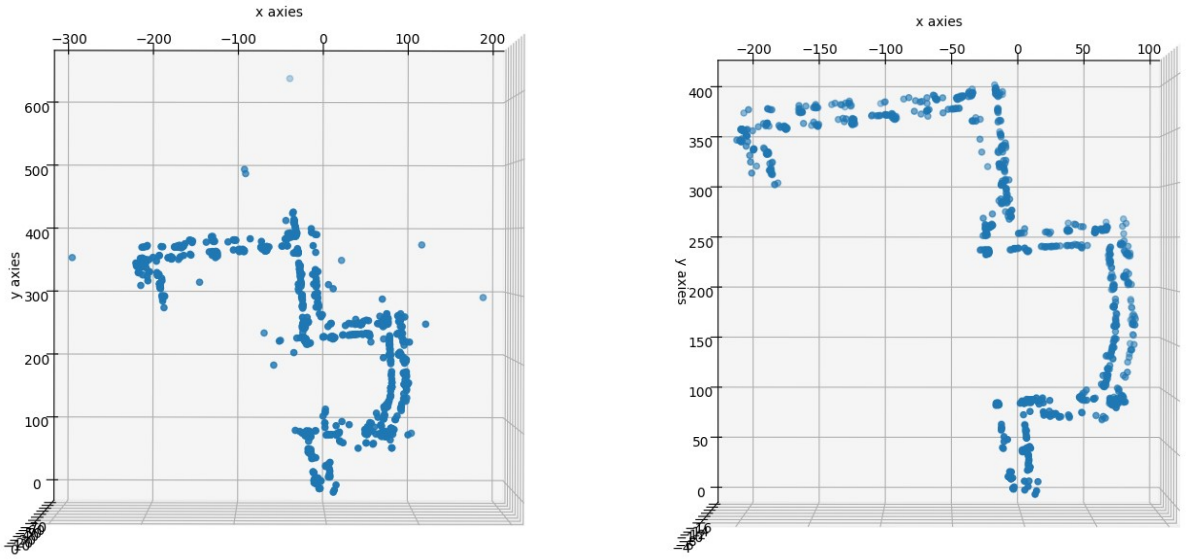
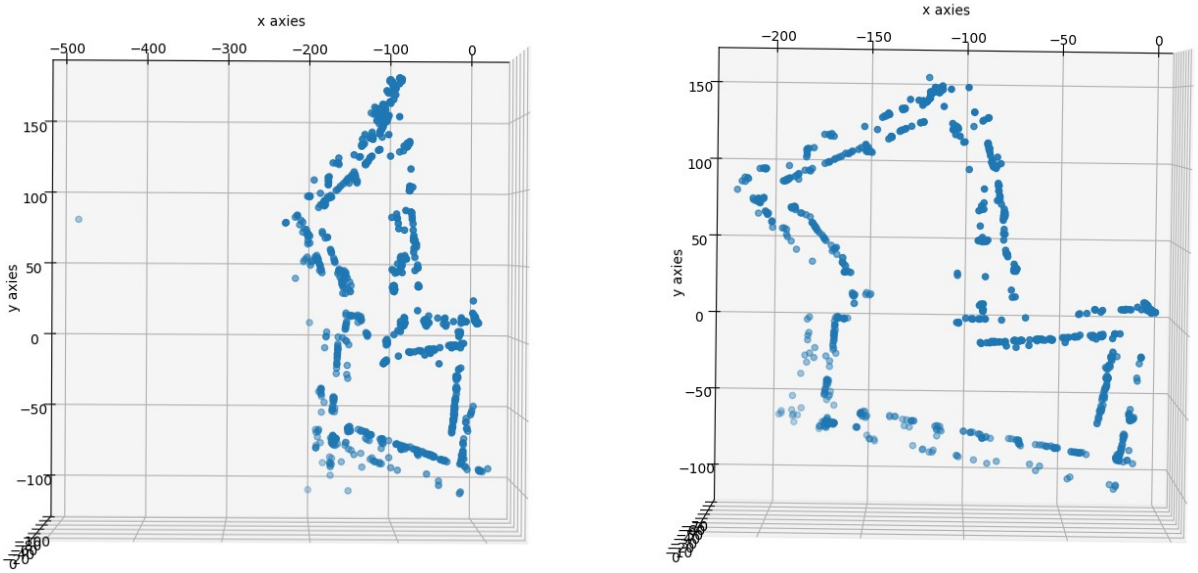


Figure 2.14: Illustration example of overlap in features when sequencing multiple frames



(a) Without Outlier Removal (b) With Outlier Removal

Figure 2.9: Effects of Outlier Removal Over 1001 Frames in sequence 00. Only one point plotted for each frame to increase visual clarity



(a) Without Outlier Removal (b) With Outlier Removal

Figure 2.10: Effects of Outlier Removal Over 1001 Frames in sequence 07

Figure 2.11: Results of a random trip with Visual Odometry for KITTI Sequence 00 over 1001 frames. Only one point plotted for each frame to increase visual clarity

2.6 Bundle Adjustment Optimization

Optimization is necessary due to noise inherent in the pose estimation solution. Small errors occur due to inaccuracies in feature tracking, triangulation of the 3D image points and the numerical nature of the pose estimation method. Those small errors add up over the run-time of the method, because pose estimation relies on knowing the previous frame pose. This results in the final estimate being very different from the ground truth. The overlap of corresponding points in non-sequential frames can be exploited in order to drastically lessen this drift using bundle adjustment. This section explains the approach behind the implementation of this bundle adjustment method.

Bundle adjustment is a technique which optimizes the camera poses and 3D point locations generated from the pose estimation, by minimizing the re-projection error between the observed image points and their corresponding 3D points in the scene. It does this by using the least Levenberg-Marquardt squares method, as described in section 2.3.2, where it starts with an initial guess of poses and 3D points, as well as intrinsics of the camera views. These initial guesses are input as the estimated poses and 2D points, as well as the fixed intrinsics of the camera. Using this sparse decomposition, the amount of necessary calculations to perform bundle adjustment is greatly reduced.

Bundle adjustment is formulated through equation 2.26:

$$ {}^{a}\mathrm{x}'_{ij} + {}^{a}\hat{\mathrm{v}}_{x'_{ij}} = \hat{\lambda}_{ij}\,{}^{a}\hat{\mathrm{P}}_j(x_{ij}, p, q)\hat{\mathrm{X}}_i \tag{2.26} $$

${}^{a}\mathrm{x}'_{ij}$ is the pixel coordinate of 3D point $\hat{\mathrm{X}}_i$ in frame a. ${}^{a}\hat{\mathrm{v}}_{x'_{ij}}$ is the pixel-wise correction necessary in order to have a correct reprojection. i denotes the 3D point being projected, and j denotes the image frame being projected to. $\hat{\lambda}_{ij}$ is a scaling factor. ${}^{a}\hat{\mathrm{P}}_j(x_{ij}, p, q)$ is the projection matrix necessary in order to correctly project the 3D point to 2D observation i in frame a. $\hat{\mathrm{X}}_i$ is the current estimate of 3D point j.

As the desired estimate is a 2D path, there is no reason for a 3D pose estimation, and the entire equation can be simplified to equation 2.27. The scaling factor in equation 2.26 can be eliminated as well, as it is there to adjust for different cameras being used for each frame, and in this case the same camera is capturing all frames:

$$ {}^{a}x'_{ij} + {}^{a}\hat{v}_{x'_{ij}} = \frac{{}^{a}\widehat{\mathrm{P}_{1:2j}}(x_{ij}, p, q)\hat{\mathrm{X}}_i}{{}^{a}\widehat{\mathrm{P}_{3j}}(x_{ij}, p, q)\hat{\mathrm{X}}_i} \tag{2.27} $$

${}^{a}x'_{ij}$ and ${}^{a}v'_{ij}$ are still the projection estimate and necessary correction.

Bundle adjustment iterates over this, minimizing the reprojection error using Levenberg-Marquardt least squares optimization. The iteration stops either once the reprojection error reaches below a fixed threshold, or a maximum number of iterations are met. The threshold is set to 1e-4, and the maximum iterations are set to 50. These values are set in the provided bundle adjustment method.

2.6.1 Sparsity

The least-squares method functions by trying to reduce residuals by changing the parameters it is given. However not every data-point is going to have an effect on the residuals, as many 3D points are only seen in a relatively small percentage of frames. Normal bundle adjustment iterates over every combination of 3D point and view. The resulting wasted operations can be reduced by using a sparsity matrix, which is a matrix of ones and zeroes which informs the least squares method about which 3D points impact which residuals. The sparsity matrix is derived from the Jacobian matrix of the problem. The Jacobian matrix represents the partial derivatives of the reprojection error in regards to the parameters being optimized. These are the camera poses and the 3D points. If an element is zero, there is no correlation between

a 3D point and a camera pose, and there is no need to make any calculations regarding reprojection during the optimization. If an element is non-zero, there is, and these values in the are set to 1, so that reprojection calculations are only done when certain 3D points have an effect on pose estimation of certain frames.

2.6.2 Implemetation

In order for the bundle adjustment to have a proper form which can be read properly. This is done by making a list with each index having the form seen in figure 2.15:

[Frame # of obs, 3D Point # of obs, x-coordinate of obs, y-coordinate of obs]

⋮

[Frame # of obs, 3D Point # of obs, x-coordinate of obs, y-coordinate of obs]

Figure 2.15: Form of each index of the list of information given to bundle adjustment

Another list of the same size is created with the coordinates of the 3D point with the corresponding index of the observation. Both lists are sorted firstly by the value of the 3D Point of the observation, then by the frame # of the observation. An example of sorting of the list seen in figure 2.15 can be seen in figure 2.16.

Frame #	3D Point #	x-coord	y-coord
0	0	405	201
1	0	265	285
0	0	481	246
1	1	554	302
1	1	399	189
2	2	244	154
1	2	352	221
2	3	457	262

x-coord	y-coord	z-coord
X1	Y1	Z1
X2	Y2	Z2
X3	Y3	Z3
X4	Y4	Z4
X5	Y5	Z5
X6	Y6	Z6
X7	Y7	Z7
X8	Y8	Z8

Figure 2.16: Example of sorted list with associated list of 3D point coordinates

A corresponding list of transformation matrices for each frame is provided, so that the bundle adjustment solution can extract the positions of the camera frames. The known intrinsics of the camera are also given to the bundle adjustment solution for projection.

To perform the bundle adjustment with sparsity, a sparsity matrix is generated using `sparsity_matrix()` to generate a matrix with 1's in positions where 3D points have an effect on residuals, and 0 elsewhere. The non-linear optimization is then done using `bundle_adjustment_with_sparsity()` to minimize the reprojection error. Since the square of the error is being used as a metric, the method tends towards smaller errors everywhere, rather than very high errors in very few places. This means it returns a statistically optimal set of 3D points and poses for the given 2D image points.

`bundle_adjustment_with_sparsity()` returns a list of the initial residuals, the residuals at the solution, and the solution itself. The solution is a list of 3D coordinates of the corrected camera poses and corrected 3D points in the form of transformation matrices. The corrected camera poses are at the start of the list, and the corrected 3D points come afterwards in the list. Knowing the amount of camera poses, the corrected x- and y- coordinates can then be read from the list, and plotted.

2.7 Loop Closure with Bag-of-Words

During a trip along a route, the estimate for the global position of the car will accumulate drift. Past a certain distance traveled, this drift will result in a skewed map of the route. To account for this, the Bag-of-Words method for loop detection was planned to be implemented. This, along with the bundle adjustment, would be sufficient for loop closure, in order to account for this drift. Loop closure works given the condition that the route intersects, or passes close by, itself. If a loop is detected, it can be assumed that some of the points seen in the camera image are the same as points seen previously at the start of the loop.
Unfortunately, for this project, it was not possible to implement this method in time, so going forward, the methods discussed for loop detection and closure will be theoretical.

2.7.1 Bag-of-Words

The Bag-of-Words method for loop detection is based on histograms of feature descriptors of an image instead of pixel coordinates. By not comparing the individual features of the i-th image to the individual features of every previous image, but instead comparing the amount of each type of feature seen, the comparison for loop detection in each frame becomes much faster.
As visual odometry is performed on a given frame, its features are extracted and saved for later comparison. When histograms are generated for this comparison, they must have an overlap in what features they might see. If they note any and all features in the image, the set of features that do not overlap will dwarf the set of features that do. In order to work around this, a vocabulary is generated before the route is trip is started. The histogram generator will only look for the features in the vocabulary, and then tabulate how many of each are seen. The pre-training of the vocabulary can be seen in section 2.7.2.

2.7.2 Vocabulary Pre-Training

To generate the vocabulary, features from a set of images are used which are similar (ex. building types, nature, lighting, gray/colour-scale cameras). It would be bad practice to sample these images from the dataset of the route which shall be run, as it cannot be assumed that a vehicle mapping with VSLAM already knows what a future route looks like. As the route used in this project is one from the KITTI dataset, images for pre-training are randomly sampled from the other KITTI routes excluding the one being run. This will ensure that the features in the sample images are somewhat similar, as all the routes in the KITTI dataset are from the same general area, and shot with the same cameras.
When generating the vocabulary, a fixed amount of "words" are used, as to not have a too large vocabulary. Once the feature descriptors are collected, they are grouped with k-means clustering. It should be tested to determine the smallest pool of clusters possible which still detects loops robustly. This gives the vocabulary used in loop detection.

2.7.3 Loop Detection

Histogram Matching

Once the trip starts, whenever the car reaches a new frame, a histogram of the feature descriptors in the image is generated by finding what vocabulary clusters each feature descriptor falls into. It is then normalized, so it sums to 1, and appended to an array of all histograms. It is then sequentially compared to the histograms of all previous images using k-nearest-neighbors. If the distance between the histograms of 2 frames is below a set threshold, it means that they are looking at the same features from a different perspective, and a loop is detected.

Point Reassignment

Once a loop is detected, it needs to be determined which feature points in the i-th image correspond to which feature points in the matched image. The features previously generated in the matched image are used again for comparison with the i-th image. To determine which points match where, a matcher should be used. A brute force matcher such as SIFT can be used, but since the features of each image already are available, a feature based matcher such as ORB is more appropriate.

Once the coordinates of feature points in the i-th image which correspond to the predetermined feature points in the matched image, the 3d-point labels of the matched image feature points, as well as the corresponding 3d-points, are found and assigned to the matches, along with the rest of the relevant data needed for bundle adjustment:

`[frame, 3D-label, x-pixel, y-pixel], [x-coordinate, y-coordinate, z-coordinate]`.

2.7.4 Proof of Concept

As a proof of concept, in figure 2.17, there is used an ORB feature matcher (follow by tutorial here[3]) to match features between frame 0 and 1054 in the KITTI dataset 07, as those frames are in a loop. This was read manually from the ground truth poses.

If a loop was detected via the histogram comparison, this feature matching would take place to determine which points in frame 0 correspond to which in frame 1054. They would then be appended appropriately as discussed, whereafter the loop closure would occur. Notably, there would also be matches between frame 0 and the frames before and after frame 1054. This would only increase the robustness of the estimate, but it does stand to question whether it is necessary to call the bundle adjustment method in quick succession or only after larger intervals.

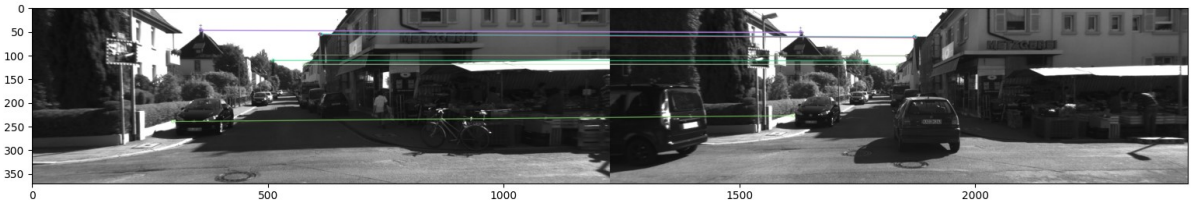


Figure 2.17: Example of an ORB matched set of images, meant for loop closure.

2.7.5 Loop Closure

Once a loop is detected, and the coordinates and labels of the 3D points in the i-th image are reassigned and the data from the frame is appended to the list of bundle adjustment input data, bundle adjustment can be run. With this, the drift between the start and end of the loop will be corrected.

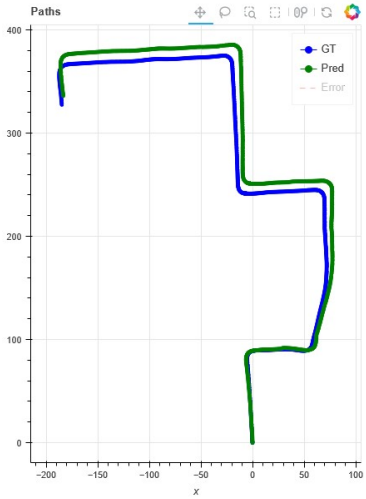
[3] `https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html`

Chapter 3

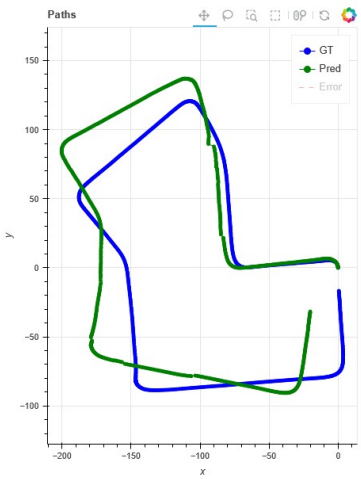
Results

The paths estimated by visual odometry compared to the ground truth can be seen in figures 3.1a and 3.1b. Figure 3.1b shows an example of aggressive drift, which shows that visual odometry is not enough by itself to accurately estimate a path. The resulting shape of the path does resemble the ground truth path to a fair point, so it can be concluded that the VO works correctly.

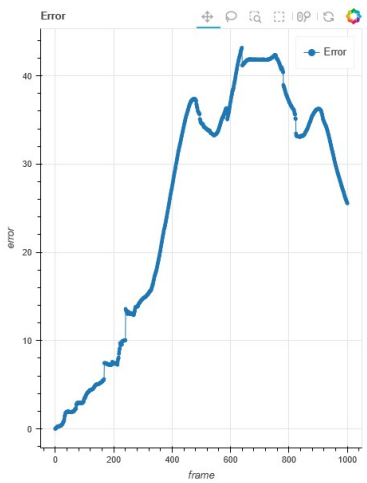
The resulting path estimation across 1001 frames from bundle adjustment, can be seen in figures 3.2 and 3.3. This path is significantly worse than the path estimated with visual odometry alone, which is seen in figure 3.1a. This has been narrowed down to a few likely faults, which is most likely a wrong sorting and passing of data between visual odometry and bundle adjustment. This is discussed further in section 4.3.



(a) Results of a random trip with Visual Odometry for KITTI Sequence 00 over 1001 frames. Blue gt pose gained from gps tracking of car. Green pose from VO pose tracking.

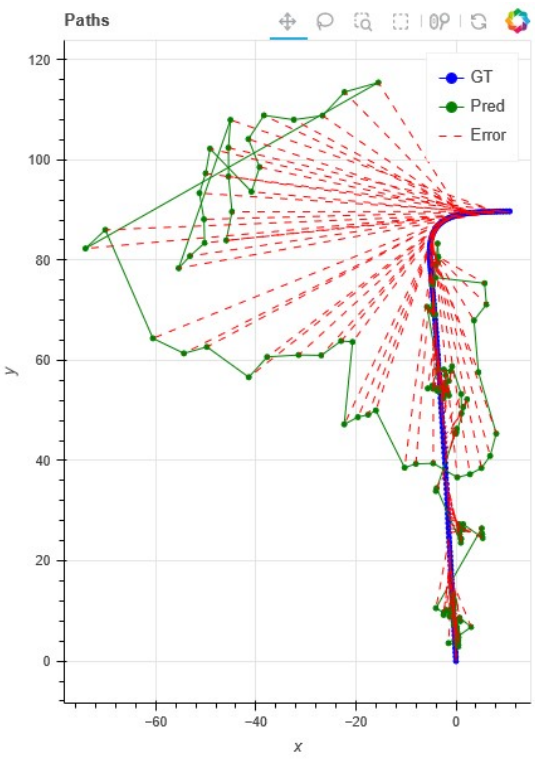


(b) Results of a random trip with Visual Odometry for KITTI Sequence 07 over 1001 frames. Blue ground truth pose gained from GPS tracking of car. Green pose from VO pose tracking.

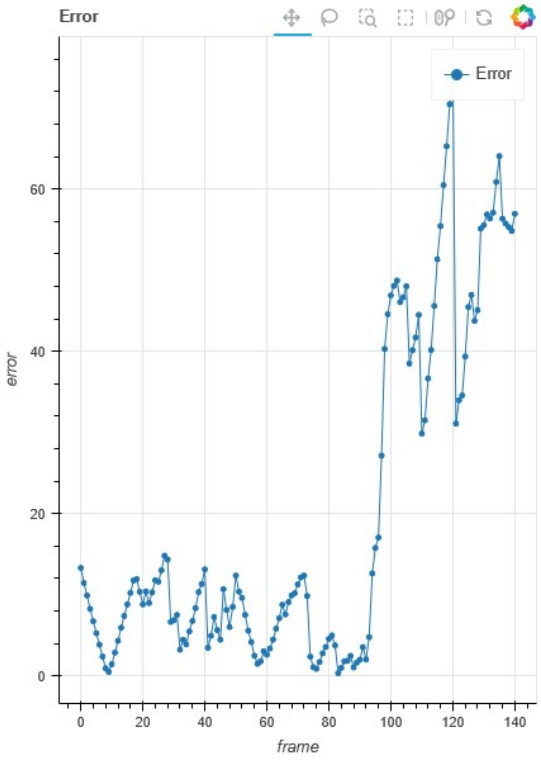


(c) Graph of the error between ground truth pose and VO estimated pose for figure 3.1b

Figure 3.1: Examples of varying consequences of drift during a VO run.

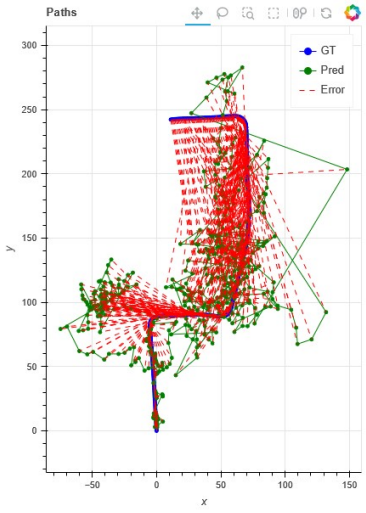


(a) Resultant path

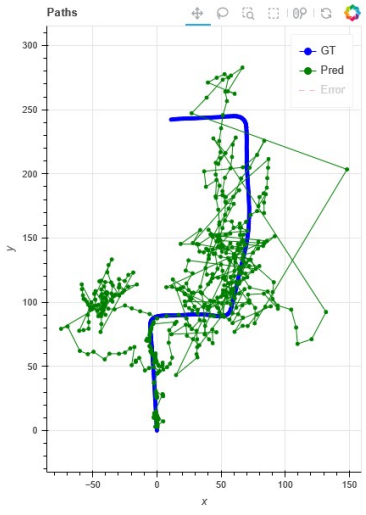


(b) Error of pose to ground truth pose

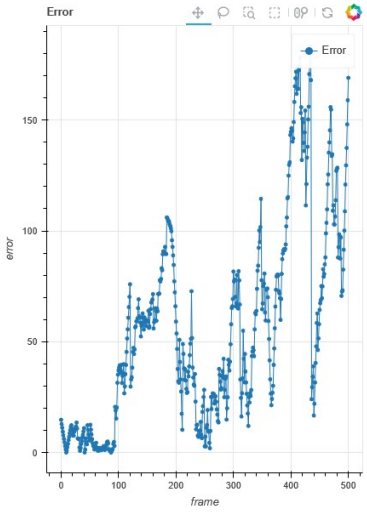
Figure 3.2: Path estimation with bundle adjustment and outlier removal over 140 frames



(a) Resultant Path With Error Lines



(b) Resultant Path



(c) Error of pose to ground truth pose

Figure 3.3: Path estimation with bundle adjustment and outlier removal over 1001 frames

Chapter 4

Discussion

4.1 Visual Odometry

The implementation of this papers' visual odometry method, per requirement, is succesful. The requirements are estimating the vehicles orientation and translation using stereo imaging. However, obstacles occurred when passing information to the bundle adjustment solution.

4.1.1 Feature Based ORB Tracker

The solution for this paper uses optical flow for tracking and estimating the path of the car. Though this method is convenient, as it was provided as a finished solution for path estimation, it is a no-traditional method for use in Visual SLAM. This is because optical flow is a pixel-based tracking method, whereas traditional methods tend to be feature- or object-based. These methods improve robustness and enables tracking of occluded objects across non-sequential frames, which allows for tracking of objects which are hidden in some frames or scenes.
This is briefly touched upon in section 2.7.4, but as this was a last-minute addition, it was not developed upon any further.

4.1.2 Combination of different trackers

For added robustness, multiple trackers could run concurrently. A pixel-based tracker could add detail to tracking between concurrent frames, and a feature-based tracker, such as ORB, could compensate for loss of point-tracking due to both occlusion and eventual loss, as happens with pixel-based trackers. This would, of course, decrease the processing speed of the solution, so it must be evaluated whether or not this is necessary, and if so, what mix is optimal.

4.1.3 3D Point Estimation

As highlighted in section 2.2, multiple methods to accomplish the goal of 3D point estimation were considered. The goal of this step in the pipeline was to provide the triangulation algorithm with the necessary input to process a 3D point from two 2D points, depicting the same feature, i.e. same keypoint in the left and right image.

The method that was used calculates a disparity map for each pair of frames. The disparity map links the corresponding keypoints in the left and right images through their pixel offset. This connection is very important within the pipeline because it lightens the computational load of the algorithm responsible for keypoint triangulation. This happens because upstream of the DLT triangulation algorithm, the inputs are already connected by the disparity value without resorting to the estimations present in the ORB procedure. At the expense of greater precision given by the ORB descriptor matching approach, a more systematic and computationally light method was chosen.

Only during the design of the Loop Detecting and Closing part, as described in section 2.7.1, would the descriptors become useful again and thus a system capable of storing descriptors would be more optimized. However, a proper implementation of the Loop Closing part was not completed, due to it not being valuable to the final quality of the pose estimation, because the loop closing procedure needed a working bundle adjustment method.

4.1.4 Keypoint Permanence

Once keypoints are generated, and a number of them are tracked to the next frame, instead of generating all new keypoints to track to the next-next frame, the keypoints that were tracked over could be kept, and the rest of the image filled in with new keypoints. This way, the triangulated 3D points could be kept across a longer tracking cycle, and the need for more data sorting and duplicate removal would be eliminated.

4.2 Outlier Removal

The outlier removal works fairly well, but a downside of it is that it can take long amounts of time to do, as it is done after the path is already estimated, having only noted which 3D-points to mark for deletion later. If it could be done in runtime, as the poses are being estimated, it would eliminate the need for a deletion loop entirely, but this is simply a quality-of-life improvement. An error occurred in the implementation of runtime outlier removal, which resulted in the estimate of visual odometry having an even larger drift as time went on. It was decided to focus on higher priority objectives than to implement this. Current outlier removal runs after pose estimation.

4.3 Bundle Adjustment

The implementation of bundle adjustment as a corrector for drift in visual odometry was unsuccessful. There are a few things that could have gone wrong in the information pipeline. The current theories for why BA fails are:

- Errors in data refining
- Defining the wrong points relationships
- Incorrect definition of sparsity matrix
- Errors in reprojection / residual calculation
- Parameter formatted passed to BA function incorrect

Since BA returns something that does not mach our path at all, the current theory is that BA gets the wrong data associations.

However, this assumption could be wrong, as there was found discrepancies between the functions in the provided bundle adjustment template and the theoretical functions and equations of bundle adjustment. The dataset which accompanied the provided template worked fine, so by method of elimination, it is deemed most likely that the cause of fault of this project is in the passing of the bundle adjustment inputs from the visual odometry.

4.4 Loop Closure

Had the bundle adjustment worked correctly, a method for Bag-of-Words loop detection could have been implemented, as discussed in section 2.7. If the loop detector could robustly detect loops, it could reassign 3D-points seen late in the trip, so that bundle adjustment could have corrected for drift. While

possible to develop and test this method by itself, it would have been impossible to implement without a working bundle adjustment solution. As the bundle adjustment was the higher priority for this project, the loop detector unfortunately had to be relegated to a hypothetical which could be added later, given more time.

4.5 Future Work

4.5.1 Real-Time Runtime

Had all gone ideally, and loop closure had been implemented, the next step would have been to optimize the VSLAM program to run in real-time. That being, that the cameras showed the trip, and estimated the path and closed loops before loading entire datasets into memory. With this, the VSLAM solution would have been implementable into autonomous vehicles.

4.5.2 Comparison With Other Solutions

Once the VSLAM solution was running in real-time, it would be necessary to optimize the code. As an example, the developed solution could be compared to the KITTI Benchmark Suite for Visual Odometry and SLAM[1].

Since the KITTI Benchmark Suite is from 2012, should the developed solution be satisfactory by that comparison, search could begin for other sources of more modern solutions for comparison.

[1] https://www.cvlibs.net/datasets/kitti/eval$_o$*dometry.php*

Chapter 5

Conclusion

Though this papers' implementation of a stereo VSLAM solution was unsuccessful, the individual components of the solution seem mostly successful. The fault is caused by the interplay in between components of the solution. The various techniques used in this paper tend towards achieving the desired solution, though the goal has not been reached.
Bundle Adjustment itself is an algorithm with fixed input and output and it can successfully perform the requested task. The problem is likely with to be the sorting and filtering of the output from Visual Odometry. A single 3D feature can be referenced into multiple pairs of frames from VO, but have different 3D coordinates, which may not be corrected for appropriately in the implementation. For this reason BA can generate errors instead of correct them. Every observation of a 3D point comes with errors added by the triangulation process and the statistical determination of the transformation matrix 2D-3D in two different frames/rows of our dataset.
The data collected and estimated from to Visual Odometry, before the bundle adjustment, are promising. The 3D structure in figure 1 before the outlier removal is proof of this. Given more time to solve the integration of visual odometry with bundle adjustment, the implementation of loop detection and closure would have been doable in relatively short order. If this is a reasonable estimation of ability and knowledge, it can be concluded that there were only a few obstacles hindering the completion of this project.

.1 Appendix

The code for this projected were based on solutions for class assignment and from the github project VisualSLAM made by the user niconielsen32: https://github.com/niconielsen32/VisualSLAM.

- Paper Authorship:
  - Abstract: Lucas
  - 1.Introduction: Gian, Xuze
  - 2.1 Visual odometry: Xuze
  - 2.1.1 Keypoints: Xuze
  - 2.1.2 Tracking: Xuze
  - 2.2 Stereo Vision: Gian
  - 2.2.1 Triangulation: Gian
  - 2.2.2 Direct Linear Transform:
  - 2.3 Motion estimation: Anders
  - 2.3.1 RANSAC: Anders, Lucas
  - 2.3.2 Levenbert-Marquardt Least Squares Optimization: Lucas
  - 2.3.3 Compounding Error: Anders
  - 2.4 3D Local to Global: Anders
  - 2.5 Data sorting: Lucas
  - 2.5.1 Outlier Removal: Lucas
  - 2.5.2 Simple sorting: Lucas, Anders
  - 2.5.3 MultiFrame Sorting: Lucas
  - 2.6 Bundle Adjustment Optimization: Lucas
  - 2.6.1 Sparsity: Lucas
  - 2.6.2 Implementation: Lucas
  - 2.7 Loop Closure with Bag-of-Words: Lucas
  - 2.7.1 Bag-of-Words : Lucas
  - 2.7.2 Vocabulary Pre-Training : Lucas
  - 2.7.3 Loop Detection: Lucas
  - 2.7.4 Proof of Concept: Lucas
  - 2.7.5 Loop Closure: Lucas
  - 3 Results: Lucas
  - 4 Discussion: Lucas, Gian, Xuze
  - 5 Conclusion: Xuze, Lucas
- Code Implementation:
  - Visual Odometry:
    - Keypoint generation and tracking: Xuze , Gian Paolo
    - Pose Estimation before BA: Anders, Gian Paolo

- Bundle Adjustement:
  - Data Processing: Lucas, Gian Paolo, Anders
  - Reprojection: Anders, Gian Paolo
  - Bundle Ajustment: Anders
- Loop Closure:
  - Loop detection: Lucas
  - Loop Closure: Lucas

Bibliography

[1] Fast algorithm for corner detection. URL https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html.

[2] H. Aanæs. Lecture notes on computer vision. 2015.

[3] H. Chatoux, F. Lecellier, and C. Fernandez-Maloigne. Comparative study of descriptors with dense key points. pages 1988–1993, 2016. doi: 10.1109/ICPR.2016.7899928.

[4] F. F. Davide Scaramuzza. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 2011. URL https://ieeexplore.ieee.org/abstract/document/6096039.

[5] R. I. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2): 146–157, 1997. ISSN 1077-3142. doi: https://doi.org/10.1006/cviu.1997.0547. URL https://www.sciencedirect.com/science/article/pii/S1077314297905476.

[6] A. Jafari Malekabadi, M. Khojastehpour, and B. Emadi. Comparison of block-based stereo and semi-global algorithm and effects of pre-processing and imaging parameters on tree disparity map. *Scientia Horticulturae*, 247:264–274, 2019. ISSN 0304-4238. doi: https://doi.org/10.1016/j.scienta.2018.12.033. URL https://www.sciencedirect.com/science/article/pii/S0304423818309154.

[7] M. Kennerley. A comparison of sift, surf and orb on opencv. 2021. URL https://mikhail-kennerley.medium.com/a-comparison-of-sift-surf-and-orb-on-opencv-59119b9ec3d0.

[8] OPENCV. Orb (oriented fast and rotated brief). URL https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. pages 2564–2571, 2011. doi: 10.1109/ICCV.2011.6126544.

[10] J. W. . C. Stachniss. Triangulation slides. *Optimization Slides of ART Master Course*, pages 16–17, 2023.