ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

Online Extrinsic Camera Calibration from Multiple Keyframes Using Map Information

Spring Term 2023

Supervised by:
Cornelius Von Einem
David Hug

Author:
Eric Tüschenbönner

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

ONLINE EXTRINSIC CAMERA CALIBRATION FROM MULTIPLE KEYFRAMES USING MAP INFORMATION

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
TÜSCHENBÖNNER	ERIC

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
MUNICH, 16.07.2023	Eric Tüschenbönner

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

Abstract

Vision-based driver assistance systems play a critical role in improving the safety of railway vehicles, especially when it comes to collision detection. In this work, we propose an online method for extrinsic calibration of a camera mounted to a track vehicle across multiple frames. The method makes use of a preinstalled GPS sensor to localize the vehicle with respect to a map and a detail camera to identify obstacles between the railway tracks. In order to determine this region of interest, local railway tracks from the map are reprojected into camera view, requiring precise knowledge of the camera pose.

Our proposed method to estimate the 6 degree-of-freedom camera pose is based on an optimization approach. The railway map and elevation data are processed in order to reproject local railway tracks into camera view for each frame, upon which the error between reprojected and detected tracks in the image is minimized, utilizing an iterative closest points (ICP) algorithm.

Evaluation across a variety of scenes shows that the proposed method is able to accurately and precisely estimate the camera pose, as compared to stereo camera calibration data. Given a decent initial height and depth estimate, it is robust to different track shapes – keyframes with depth-variability of the track actually improve the result. When optimizing across multiple frames, the accuracy of the method is limited by sensor noise, in particular the RTK-GPS rotation measurements, where small angular errors lead to reprojection inconsistencies. To overcome this limitation, sensor fusion can be applied to improve the GPS state estimate with IMU and odometry data.

Symbols

Symbols

ϕ, θ, ψ	Roll, pitch, and yaw angles
λ	Depth of image point
$\otimes$	Quaternion product

Indices

c_x, c_y	principal point coordinates in pixels
f_x, f_y	focal lengths in pixels
H	Homogeneous transformation matrix
K	Camera intrinsics matrix
q	Quaternion
R	Rotation matrix
t	Translation vector
u, v	horizontal, vertical image coordinates in pixels
x, y, z	x, y, and z coordinates

Acronyms and Abbreviations

CNN	Convolutional Neural Network
ETH	Eidgenössische Technische Hochschule
GPS	Global Positioning System
ICP	Iterative Closest Points algorithm
IMU	Inertial Measurement Unit
LROD	Long-Range Obstacle Detection
OSM	Open Street Map
ROI	Region of Interest
UTM	Universal Transverse Mercator coordinate system

Chapter 1

Introduction

Obstacle detection is crucial for safe operation of railway vehicles. A prerequisite for this is identifying a region of interest (ROI), in other words knowing where to look for obstacles, which means that the tracks ahead of the vehicle need to be correctly identified and located. This could be done by projecting a known railway map into camera view. However, this requires precise knowledge of the camera position and rotation – not typically available in the field or with existing datasets. Given the degree of accuracy required for long-range obstacle detection (LROD), this is a non-trivial task.

The goal of this semester project is to develop an extrinsic camera calibration and reprojection pipeline based on visual cues and map information. Available data includes a set of images with associated GPS poses, from a camera and GPS sensor both mounted to a vehicle driving along a track. In addition, map information includes OpenStreetMap (OSM) data with the positions and properties of railway nodes and tracks, as well as elevation data.

Related work includes approaches based on photogrammetry, convolutional neural networks (CNNs), and image geometry. The first approach [1] requires instruments such as a calibration board placed ahead of the camera, but does not use "online" visual information collected during operation. The second approach [2] trains a CNN to determine the camera extrinsics, requiring a large amount of labeled data not available in this case. Finally, [3] extracts geometric image features to compute the camera pose step-by-step. However, this approach is not generalizable as it only works on carefully-selected individual frames with straight tracks.

The solution presented here proposes to directly reproject the 3D map into camera view, and formulate an optimization problem based on iterative closest points (ICP) to determine the correct camera pose. The aim is to build a robust pipeline that works across multiple frames, thus alleviating the prior shortcomings.

This report is structured as follows. Chapter 2 provides background information about relevant technical concepts. Chapter 3 outlines the proposed method, describing all main components, processes, as well as inputs and outputs in detail, while Chapter 4 discusses the implementation details, including software architecture, objects and data types. Finally, Chapter 5 presents, analyzes, and evaluates the results, and Chapter 6 concludes the report.

Chapter 2

Background

This chapter summarizes background information that may be useful to better understand the method and implementation that will be outlined in the following two chapters. Section 2.1 defines the different coordinate systems applied in the project, while Section 2.2 outlines methods for coordinate transformations. Section 2.3 describes the camera model used for reprojection and undistortion, and Section 2.4 summarizes the Iterative Closest Points (ICP) algorithm.

2.1 Coordinate Systems

Three different coordinate systems are used for the purpose of this project: world, GPS, and camera frame. The world frame is constant and given in UTM coordinates, used by the map information. The GPS frame is the time-variant position of the GPS sensor, while the camera frame is the time-variant position of the camera on the train.

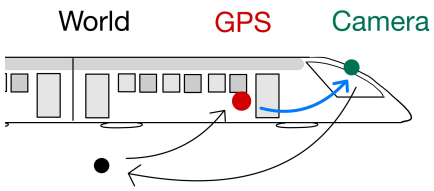

Figure 2.1: World, GPS, and camera coordinate systems illustrated, with the arrows implying their relative transformations [3].

Table 2.1 provides a comparison of the axis directions in the GPS and camera frames. In the GPS frame the X-axis points forward, the Y-axis to the right of the train, and the Z-axis upwards, whereas in the camera frame the Z-axis points forward, the X-axis to the right of the train, and the Y-axis downwards. The camera frame is defined such that the camera points in the direction of the Z-axis, and the width and height of the image are along the X- and Y- axes, respectively.

Table 2.1: Definitions of directions and associated rotations, in terms of GPS and camera coordinate systems.

Direction	**Rotation**	**GPS Frame**	**Camera Frame**
Longitudinal (forward)	Roll	$+X$	$+Z$
Lateral (sideways, right)	Pitch	$+Y$	$+X$
Vertical (upwards)	Yaw	$+Z$	$-Y$

2.2 Coordinate Transformations

In order to efficiently transform points between different coordinate systems, two different representations have been used for this project. Firstly, homogeneous transformation matrices are used most of the time since they are more intuitive and simple. Secondly, however, quaternions are used for optimization, where they are dynamically adapted, since they are not prone to numerical singularities.

2.2.1 Homogeneous Transformation Matrix

Transformation, including both translation and rotation, of a vector to point P, from initial frame $\mathcal{A}$ to frame $\mathcal{B}$. This is achieved using the rotation matrix $R_{\mathcal{BA}}$ (notation: frame $\mathcal{A}$ to frame $\mathcal{B}$) and translation vector ${}_{\mathcal{B}}\boldsymbol{t}_{BA}$ (notation: from point A to point B, expressed in frame $\mathcal{B}$). To avoid computation issues, it is crucial to remember which frames the vectors are expressed in.

The homogeneous transformation can be written as a simple Equation (Equation (2.1)) or combined in the form of a matrix $H_{\mathcal{BA}}$ ()Equation (2.3)).

$$ {}_{\mathcal{B}}\boldsymbol{r}_{BP} = {}_{\mathcal{B}}\boldsymbol{t}_{BA} + R_{\mathcal{BA}} \cdot {}_{\mathcal{A}}\boldsymbol{r}_{AP} \tag{2.1} $$

$$ \tag{2.2} $$

$$ \begin{bmatrix} {}_{\mathcal{B}}\boldsymbol{r}_{BP} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} R_{\mathcal{BA}} & {}_{\mathcal{B}}\boldsymbol{t}_{BA} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}}_{H_{\mathcal{BA}}} \cdot \begin{bmatrix} {}_{\mathcal{A}}\boldsymbol{r}_{AP} \\ 1 \end{bmatrix} \tag{2.3} $$

To determine the inverse of a homogeneous transformation matrix, the translation vector need not only be reversed but also rotated to the new frame, while the rotation matrix is simply transposed. This results in the composition as in Equation (2.4).

$$ H_{\mathcal{AB}} = \begin{bmatrix} R_{\mathcal{AB}} & {}_{\mathcal{A}}\boldsymbol{t}_{AB} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} R^T_{\mathcal{BA}} & -R^T_{\mathcal{BA}} \cdot {}_{\mathcal{B}}\boldsymbol{t}_{BA} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{2.4} $$

2.2.2 Quaternion Rotation

An effective mathematical tool that can be used to express rotations are quaternions. A quaternion $\boldsymbol{q}$ is a 4D vector with the elements q_w, q_x, q_y, and q_z (Equation (2.5)). Its conjugate $\boldsymbol{q}^*$ is the same rotation in the opposite direction.

$$\boldsymbol{q} = q_w + q_x \cdot \boldsymbol{i} + q_y \cdot \boldsymbol{j} + q_z \cdot \boldsymbol{k} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \qquad \boldsymbol{q}^* = \begin{bmatrix} q_w \\ -q_x \\ -q_y \\ -q_z \end{bmatrix} \tag{2.5}$$

In order to rotate a vector $\boldsymbol{r}$ using a quaternion $\boldsymbol{q}$, the quaternion product $\otimes$ is used (equal to the cross-product minus the dot-product), see Equation (2.6). Note that the quaternion $\boldsymbol{q}$ must be a unit quaternion, i.e. scaled to unit norm.

$$\begin{bmatrix} 0 \\ {}_{\mathcal{B}}\boldsymbol{r} \end{bmatrix} = \boldsymbol{q}_{\mathcal{BA}} \otimes \begin{bmatrix} 0 \\ {}_{\mathcal{A}}\boldsymbol{r} \end{bmatrix} \otimes \boldsymbol{q}^*_{\mathcal{BA}} \tag{2.6}$$

2.3 Camera Model

This section describes the camera model used in this project. Firstly, the reprojection of 3D points to 2D pixel coordinates is described, followed by the undistortion of the image.

2.3.1 Reprojection

Following the pinhole camera model [4], coordinates (x, y, z) in the camera frame are reprojected to pixel coordinates (u, v) in the image plane using Equation (2.7). The variables f_x and f_y are the focal lengths in pixels, while c_x and c_y are the principal point coordinates in pixels.

$$u = f_x \cdot \left(\frac{x}{z}\right) + c_x \qquad v = f_y \cdot \left(\frac{y}{z}\right) + c_y \tag{2.7}$$

This can also be written in matrix form (Equation (2.8)), using the camera instrinsics matrix K. The variable λ is the depth scaling factor, since infinitely many 3D points along a line would project to the same 2D point.

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.8}$$

2.3.2 Undistortion

Image distortion means that the camera does not follow the pinhole camera model perfectly. The image is distorted by radial distortion and tangential distortion, as shown in Figure 2.2.

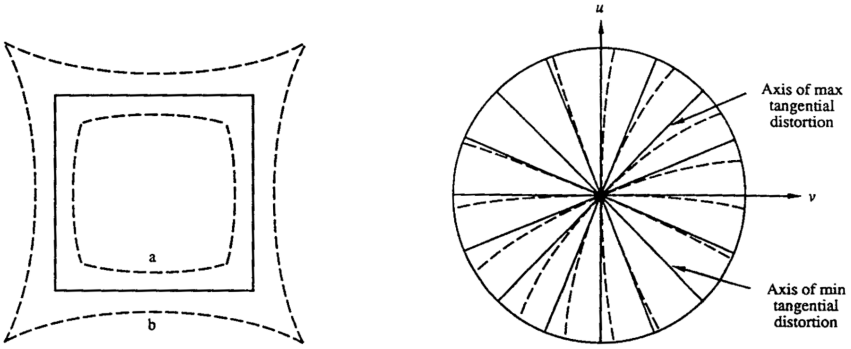

Figure 2.2: Radial distortion (left) and tangential distortion (right) [5].

Since the initial images are distorted, they first need to be undistorted given the camera intrinsics that include the distortion coefficients. The distortion is equidistant and addressed using the OpenCV fisheye model [6].

2.4 Iterative Closest Point (ICP) Algorithm

The ICP algorithm [7] [8] is an iterative method for aligning two point clouds. It is commonly used in robotics for estimating the pose of a robot with respect to a known map, or for estimating the relative pose between two point clouds.

The algorithm iteratively minimizes the sum of squared distances between the points in the two point clouds. Initialized with an initial guess of the relative pose between the two point clouds, at each iteration, the points in the second point cloud are transformed using the current estimate of the relative pose. The closest points in the first point cloud are then found for each point in the second point cloud. The relative pose is then updated using the closest points. This process is repeated until the relative pose converges.

In the context of this project, the algorithm is used to minimize the 2D residuals between reprojected points and observed image points. The reprojected points are obtained by transforming 3D points using a relative pose estimate, and then reprojecting them onto the image plane using the known camera model. After finding the closest reprojected point for each observed image point, the relative pose is updated using these closest points. This process is repeated until the relative pose converges. The method is described in more detail in Section 3.5, while the implementation is described in Section 4.2.

Chapter 3

Method

This chapter outlines the method of the reprojection and optimization pipeline. Following an overview of all components in Section 3.1, the process of each component is described in detail. Implementation details will be covered in the next chapter.

3.1 Overview

Figure 3.1 shows an overview of the four main components: railway processing, track detection, track reprojection, and pose optimization, as well as the global inputs and outputs, color-coded by the type of data. The global map data (green) includes *railway network* and *elevation data*, the frame-specific data (red) includes *image* and *GPS pose*, and the camera-specific data (blue) includes *camera intrinsics* and *camera pose*. All of these are inputs to the pipeline, and the output that is iterated over is the *camera pose*. The processes in each component are described in the following sections.

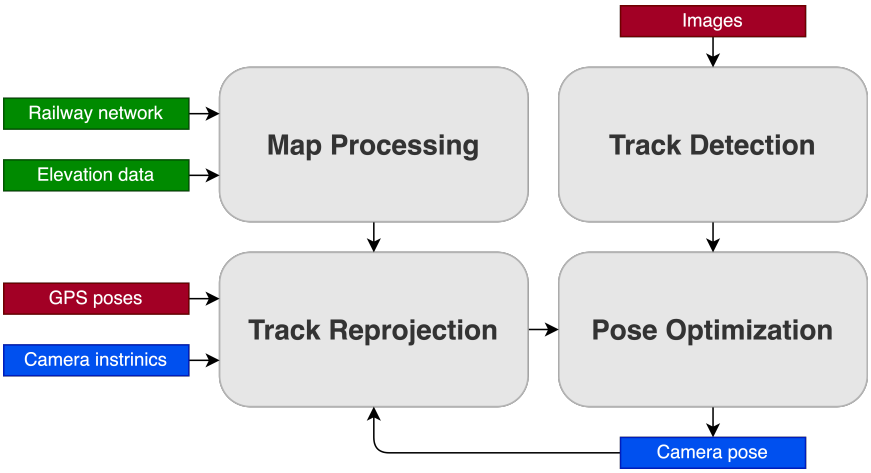

Figure 3.1: Overview of the components, interactions, and inputs/outputs color-coded by type (green: global map, red: frame-specific, and blue: camera-specific).

3.2 Map Processing

In this component, raw map data is processed into a 3D point cloud for each railway track that is more usable for downstream components, particularly Track Reprojection. The raw map data includes a railway network in the form of an OpenStreetMap (OSM) file (illustrated in Figure 3.2), as well as elevation data. The output is a set of 3D points for each railway track.

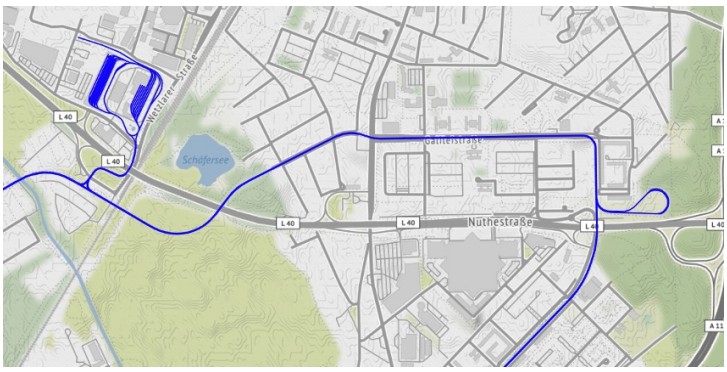

Figure 3.2: Railway network from OpenStreetMap (OSM) file.

The process includes extracting the nodes and tracks from the railway map, converting them to 2D splines sorted by tracks, then filling the gaps between the points to achieve more regular spacing, and finally adding elevation data to obtain a 3D point cloud for each track. The last step also includes adapting the elevation to the start/end points if the current track segment is a bridge or tunnel. An overview of the input, process, and output is shown in Table 3.1.

Table 3.1: Map processing component: input, process, and output.

Input:	• Railway network (OSM file) • Elevation data
Process:	1. Extract nodes and tracks from railway network 2. Convert nodes to 2D splines per track 3. Fill track gaps by 2D interpolation 4. Add elevation data to get 3D points
Output:	• Railway tracks as 3D point clouds

Overall, the initial data is enhanced, while also being reduced to what is actually needed for downstream components – only in the relevant area that is a combined map of all specified frames. Even though processing a combined map of all keyframes takes more time once, it is much faster than processing the railway surrounding each frame location individually, since points may overlap. Moreover, this is the only way to ensure that large railway gaps do not lead to a problem of missing data in the reprojection of any single frame, as would be the case otherwise.

3.3 Track Reprojection

In this component, local 3D railway points are reprojected onto each image, given the GPS pose of the current frame, the camera intrinsics, and camera pose estimate. The output is a set of dense, regularly-spaced 2D points on the image.

This is done by first finding local railway tracks, by searching within a radius of interest around the current frame GPS pose, then increasing the point density using 3D interpolation, transforming these points into the camera frame, filtering the points by angle to the camera (to obtain a more regular spacing in image space), and finally reprojecting these points into the image space. An overview of the input, process, and output is shown in Table 3.2. The reprojected points for an initial camera pose estimate are shown in Figure 3.3.

Table 3.2: Track reprojection component: input, process, and output.

Input:	• Railway tracks as 3D point clouds • GPS pose • Camera pose estimate • Camera intrinsics
Process:	1. Find local railway tracks – search radius of interest 2. Increase point density by 3D interpolation of tracks 3. Transform points into camera frame 4. Filter number of points by angle to camera 5. Reproject onto image
Output:	• Reprojected local tracks

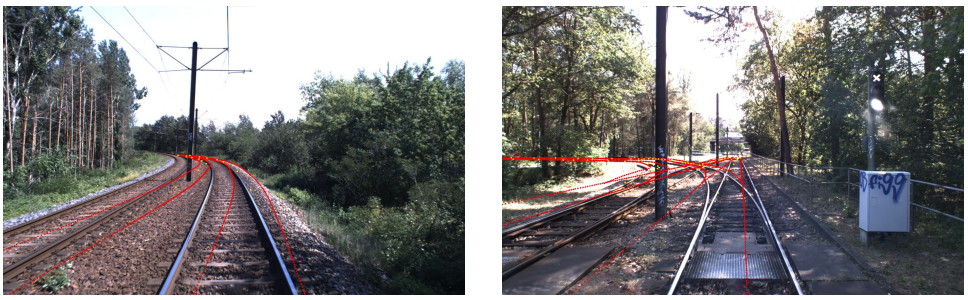

Figure 3.3: Reprojected enhanced railway tracks (red) and original nodes (yellow).

In contrast to the previous component, this process is run for each frame individually, and the reprojection step is repeated at each iteration of the camera pose, which is iterated over in the Pose Optimization component. This means that the filtered, dense points in 3D space are constant throughout the iterations for a single frame, yet reprojected differently for each camera pose estimate.

3.4 Track Detection

In this component, visible railway tracks are observed in each image, currently by manual annotation but expandable to automated detection. The output is a set of dense 2D points for each observed track.

The process consists of manually annotating points along the tracks in each image, converting these points to 2D splines, and increasing the point density by 2D interpolation. An overview of the input, process, and output is shown in Table 3.3, while the annotated points and interpolated output is shown in Figure 3.4.

Table 3.3: Track detection component: input, process, and output.

Input:	• Image
Process:	1. Manually annotate 2D points in images 2. Convert to 2D splines 3. Increase point density by 2D interpolation
Output:	• Observed tracks as 2D points

Figure 3.4: Annotated points (light blue) and interpolated splines (dark blue).

This component is not yet implemented with machine learning since the focus of this project was on the optimization algorithm and reprojection pipeline. As such, manual annotation is used to obtain ground truth data for the evaluation of the Pose Optimization component. Nevertheless, track detection is a crucial part of the pipeline, and should be integrated in future work in order to obtain a fully automated and generalizable pipeline.

3.5 Pose Optimization

In this component, the camera pose is optimized by minimizing the error between the observed tracks and the reprojected local tracks. The output is an updated camera pose estimate.

This is done by using an iterative closest point (ICP) algorithm. At first, one-to-one correspondences are found between the observed and reprojected points from which residuals are computed, equal to the distance between corresponding points. Second, these residuals are added to the optimization problem and it is solved to obtain a new camera pose estimate. This process is repeated until convergence, with new correspondences found at each iteration. An overview of the input, process, and output is shown in Table 3.4, while the correspondences are illustrated in Figure 3.5.

Table 3.4: Pose optimization component: input, process, and output.

Input:	• Observed tracks • Reprojected local tracks
Process:	1. Find one-to-one correspondences for each observed point 2. Compute residuals 3. Add to optimization problem 4. Solve optimization problem
Output:	• New camera pose estimate

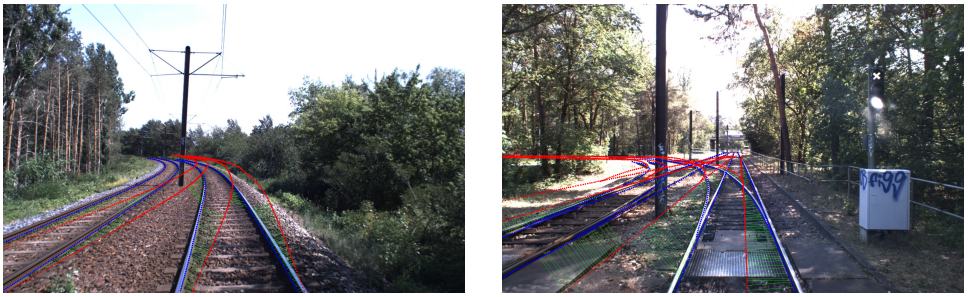

Figure 3.5: Initial correspondences (green) between reprojected points (red) and observed points (blue).

Specifically, one-to-one correspondences are found as follows: for each observed point, the closest reprojected point is found. If a reprojected point happens to be associated to multiple observed points, only the closest observed point remains associated to the reprojected point, while all others are removed and left without correspondence. This is done to ensure that each observed point is associated to only one reprojected point, and vice versa.

The optimization problem can be solved as single-frame or multi-frame, where the latter implies that multiple frames are optimized in parallel. More details regarding these approaches are provided below.

3.5.1 Single-frame vs. Multi-frame

In the single-frame approach, the optimization problem is solved for an individual frame. At first, the initial camera pose estimate is used to reproject the 3D points onto the image plane. Then, correspondences are found between the reprojected points and the observed points. These are used to compute the residuals and add them to the optimization problem, which is then solved to obtain a new camera pose estimate. This process is repeated until convergence, with new correspondences found at each iteration given the updated camera pose. This is equivalent to only the middle column in Figure 3.6.

In the multi-frame approach, the optimization problem is solved for multiple frames in parallel. Here, the computed residuals are added to the same combined optimization problem, which is then solved to obtain a new camera pose estimate. This process is repeated until convergence, with new correspondences found for all frames at each iteration given the updated camera pose. An overview of this process is shown in Figure 3.6.

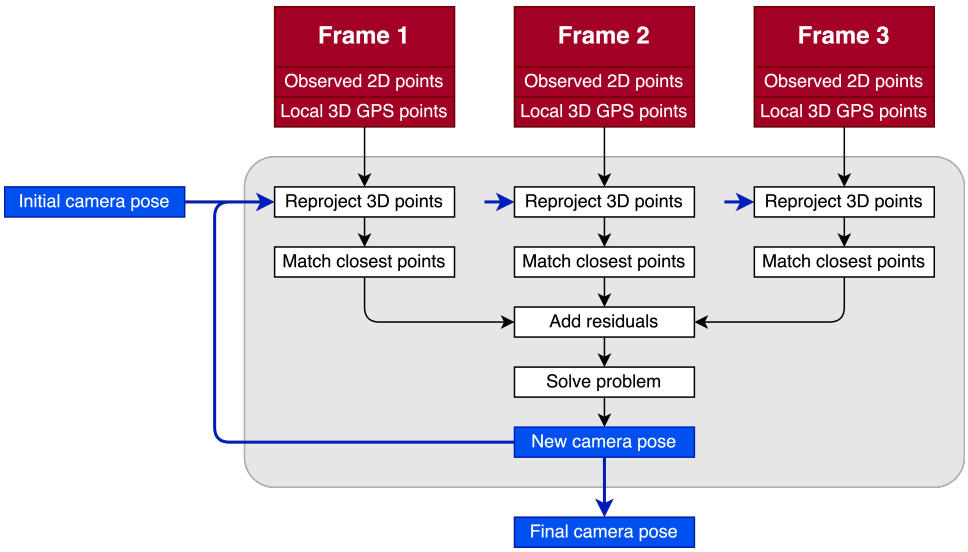

Figure 3.6: Multi-frame optimization process combining residuals.

Chapter 4

Implementation

This chapter gives an insight into the implementation of the proposed method, demonstrating the modularity and adaptability of the code. The pipeline is implemented in Python, with C++ integrated for optimization using Ceres Solver [9]. Section 4.1 describes the Python code structure and classes, while Section 4.2 outlines the integration of C++ code for optimization.

4.1 Python Structure & Classes

The "main.py" file is the core of the pipeline where all other functions and classes are called. Here, one can interact with the code and adjust specific parameters. The sequence of calls is as follows:

1. Define Camera objects with intrinsics and initial pose
2. Create Railway object using frames along track
3. Visualize Railway or frame data (GPS position, rotation, elevation)
4. Visualize initial reprojections
5. Create keyframes as list of Keyframe object
6. Optimize pose of each camera across keyframes
7. Compute stereo camera transformation & accuracy w.r.t. calibration

Data sources are specified in the "data.py" file. Along with the system paths where map and frame data is stored, known parameters such as the railway width and tram data [10] are defined.

Camera

Each camera is initialized as its own Camera object (file: "camera.py"), containing camera-specific data including intrinsics, pose (updated during optimization), as well as methods for reprojection and undistortion. Undistortion maps are automatically created and the camera model is updated accordingly. Other objects make use of the Camera object's methods for reprojection and undistortion.

Railway & MapInfo

The Railway object (file: "railway.py") is the output of processing the railway map and elevation data into relevant 3D point clouds. It uses a sequence of frames as input, whose GPS data are used to create a combined map. Additional inputs are the maximum interpolation gap between points ("max_gap") and the search radius ahead / behind each frame that is used to find railway nodes ("r_ahead", "r_behind").

All outputs are automatically created at class initialization. Moreover, the Railway object is stored as a pickle file for later use with the same map, so that it does not need to be reprocessed each time.

The MapInfo class (file: "map_info.py") contains methods to obtain non-railway map data, such as elevation and pole locations (currently unused). The elevation data is taken from [11].

Keyframe, GPS & Annotation

The Keyframe object (file: "keyframe.py") contains all frame-specific data of the selected keyframes, including their ID, GPS data, images, associated cameras, and annotations. All information processing, such as obtaining the local 3D GPS points, is done automatically at initialization.

Frame is a simplified and more lightweight version of Keyframe that the latter it is based upon. It only contains ID and GPS data (without elevation) and is used for creation of the Railway object. The idea is to quickly create a dense sequence of frames to create the relevant Railway object, while only creating a sparse selection of the more sophisticated keyframes that take longer to initialize.

The GPS object (file: "gps.py") is part of each Frame/Keyframe and contains the processed GPS sensor pose readings. This includes the heading, different representations of the position and rotation, as well as local elevation computed via the MapInfo class.

The Annotation object is also part of each Keyframe (for each camera) and outsources the processing of the image annotations. It creates a 2D spline for each annotated pixel sequence read via a CSV file.

Transformation

The Transformation class (file: "transformation.py") includes several static methods. These include operations on homogeneous transformation matrices, conversions between different rotation parametrizations, transformations of points between different coordinate frames, and spline interpolation. Computations from several other parts of the code are outsourced to this file.

4.2 C++ Integration

Ceres solver has been selected for optimization due to its flexibility, efficiency, and documentation, however, the library is written in C++. Integration into the existing Python pipeline has been achieved using Pybind11 [12], which creates Python bindings for C++ code. The following sections describe the optimization process and data transfer.

Optimization Process

Optimization is done using Ceres Solver [9], for each camera separately and a specified number of iterations. At each iteration, a new Ceres problem is initialized, then for each keyframe a residual block is computed using the reprojection and correspondence search functions, the residual blocks are added to the problem using the cost function, and the problem is solved to update the camera pose.

Reprojection function: reprojects 3D GPS points to 2D image coordinates using the current camera pose and camera intrinsics. At first, the 3D GPS points are transformed to the camera frame using the current camera pose, and then reprojected to 2D image coordinates using the camera intrinsics.

Correspondence search function: finds the closest reprojected point to each observed point. For each observed point, the closest reprojected point is determined by finding the smallest Euclidean distance between the observed point and the reprojected points. If there are now multiple correspondences for any reprojected point, only the closest observed point is kept as its correspondence. These one-to-one correspondences make sure that the problem is better conditioned and that the optimization converges faster.

Cost function: calculated as the sum of all residuals, which are the distance between corresponding reprojected and observed points. The Ceres cost functor keeps track of the residuals and their derivatives, which are used to update the camera pose. Minimizing the cost function is the goal of the optimization.

Python Bindings & Data Transfer

Data is passed between Python and C++ using Pybind11 [12] as a Git submodule. Pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa. It is used to create Python bindings for C++ code, which can then be called from Python. Other integration methods (e.g. PyCeres) have also been attempted but did not succeed or function as expected without errors.

Three functions are made available to Python that directly call their associated C++ functions: "add_keyframe", "reset_keyframes", and "update_camera_pose". The first two are used to create a new list of keyframes for a new camera, while the last one is used to optimize the pose of a camera over a fixed number of iterations.

To make relevant data available to C++, the "add_keyframe" function is called, which creates a list of C++ Keyframe structs for the current camera. Inputs are

the filename, camera ID, image, observed 2D points, and GPS 3D points. The first three inputs are for visualization purposes and storing the results with suitable filenames, while the last two are essential for the optimization. See Table 4.1 for an overview of the types before and after conversion.

To optimize a new camera, the function "reset_keyframes" is called, which resets the list of keyframes, after which the keyframe data of the new camera can be added again using the "add_keyframe" function. The "reset_keyframes" function takes no inputs.

Once the keyframe data has been added for a specific camera, the function "update_camera_pose" is called, which takes the camera pose, camera intrinsics, and number of iterations as inputs. The camera pose is a vector with 7 elements, containing the translation vector and quaternion in scalar-last representation (x, y, z, q_x, q_y, q_z q_w). The camera intrinsics are a vector with 4 elements, containing the focal length and principal point coordinates (f_x, f_y, c_x, c_y). The number of iterations is an integer value. The output is the final camera pose, which is also a vector with 7 elements. An overview of the types is given in Table 4.2.

Table 4.1: Keyframe struct attributes and their types.

Variable	**Python Type**	**Converted C++ Type**
filename	str	std::string
camera_id	str	std::string
image	np.ndarray	cv::Mat
observed_2D_points	np.ndarray	std::vector<Eigen::Vector2d>
gps_3D_points	np.ndarray	std::vector<Eigen::Vector3d>

Table 4.2: Pose update function inputs and their types.

Variable	**Python Type**	**Converted C++ Type**
camera_pose	np.ndarray	double[7]
camera_intrinsics	np.ndarray	double[4]
iterations	int	int

Chapter 5

Results & Evaluation

This chapter presents the results of the proposed method, and evaluates its performance in terms of accuracy, robustness, and generalization. Focus is put onto the single-frame optimization in Section 5.1, but the adaptation to multi-frame optimization is also discussed in Section 5.2. In both cases, the results are evaluated regarding different aspects and recommendations are given to optimize performance.

5.1 Single-frame

This section evaluates single-frame optimization with respect to convergence, robustness, and accuracy on a variety of different scenes, using the same initial camera pose. The converged results are shown in Figure 5.1.

Figure 5.1: Converged correspondences after single-frame optimization.

It can be seen that the final alignment between reprojected and observed tracks is quite good in all cases. Even if the tracks are not perfectly aligned, as is the case with some intersections (e.g. bottom right image) which is likely due to inaccuracies in the railway map, the track directions on all sides of the intersection is still very well aligned.

5.1.1 Convergence & Initial Estimates

The method converges predictably and reliably, given a decent initial height and depth estimate. During analysis, the method always converged after 10-50 ICP iterations (i.e. number of correspondence updates), depending on the scene complexity and initial pose estimate. In this context, convergence implies that the alignment does no longer noticeable improve with further iterations.

However, it is important to have a close initial height and depth position estimate, which are both hard to optimize for. Height is difficult to optimize for because a slight change ($\pm$ 1m) leads to a very different reprojection that may not converge to the global minimum. Depth, on the other hand, is difficult to optimize for because the reprojection error is very flat in this dimension, i.e. a small change in depth hardly leads to any change in the reprojection, since the track does not change as quickly with depth. Nevertheless, the height and depth positions are easy to approximate beforehand.

When it comes to all other initial guesses, including the lateral position and all rotations, they were simply set to zero in the camera frame – implying that the camera faces perfectly forward at the same lateral position as the GPS sensor – without any convergence issues. Overall, the method is resilient to incorrect initial guesses, as long as the height and depth estimates are decent.

5.1.2 Robustness & Generalization

The method is also robust to scene changes and can be generalized to new data without problems. Even if the initial ICP correspondences are predominantly incorrect, i.e. the wrong tracks are associated with each other, they still adjust themselves during optimization. An example of this is shown in Figure 5.2. This self-correcting of correspondences to the correct tracks ocurrs relatively quickly within the first few iterations, after which the pose is only refined.

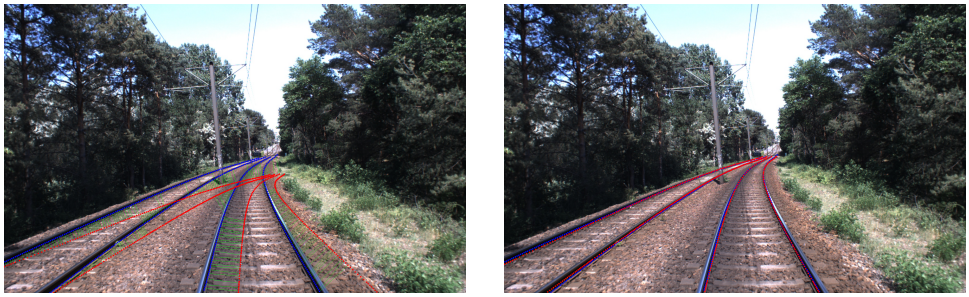

Figure 5.2: Initially incorrect correspondences (left) adjusting themselves (right).

Generalization to different scenes and new datasets is also possible, yet keyframes with higher track complexity should be selected to achieve better results. Higher track complexity means that the scenes should have a depth-variability of the track, as is the case in curves or intersections, as opposed to simple straight tracks. Examples of such scenes are the chosen frames in this chapter. In general, more complex tracks also lead to better results because more information is available to optimize for in different dimensions. With these prerequisites, the method is robust and generalizes well.

5.1.3 Accuracy & Precision

The accuracy of the pose estimation can be evaluated using the stereo camera calibration data, which provides a ground truth for the relative pose between the stereo cameras. At first, the stereo cameras are optimized separately, using the same optimization method as for the single-frame optimization, to get their poses relative to the GPS sensor. Then, the relative pose between the stereo cameras is computed and compared to the calibration ground truth.

Table 5.1: Stereo calibration data compared to relative camera transformation obtained from independent optimizations of different keyframes.

	ΔX [m]	ΔY [m]	ΔZ [m]
Calibration	0.307	0.002	0.010
Frame 1	0.289	-0.006	0.163
Frame 2	0.261	-0.047	0.128
Frame 3	0.300	0.010	0.133

As can be seen from Table 5.1, the accuracy of the calculated stereo camera transformations is relatively high. The horizontal offset ΔX and vertical offset ΔY are both within less than 0.05 m of the calibration value, while there's a slightly higher error of up to 0.16 m in the depth offset ΔZ. The horizontal and vertical offset accuracies reflect the accuracy of the reprojection and optimization pipeline. Moreover, the results are very consistent, with all three frames producing similar results, which indicates that the pose estimation is very precise as well.

The lack of accuracy in the depth estimate can be explained by the fact that the track does not change very quickly with depth (at least not on a centimeter-scale), which makes it hard to accurately estimate and sensitive to the initial guess. Nevertheless, for the same reason this is also the least critical dimension to estimate accurately, since small errors don't lead to visual reprojection errors.

When it comes to rotation, the results are all quite accurate and close to zero, as expected. If this were not the case, the reprojections would look very different since small angular errors can lead to large reprojection errors.

5.2 Multi-frame

The multi-frame optimization is evaluated on two sets of 3 frames, one with consecutive frames and another with keyframes of different scenes. All frames use the same initial camera pose estimate, as outlined in single-frame optimization above.

5.2.1 Multiple Keyframes

The first set of frames consists of three keyframes with different scenes that are optimized simultaneously. The initial estimates are shown in Figure 5.3, while the converged outputs are shown in Figure 5.4. A comparison to the equivalent single-frame optimization outputs is shown in Figure 5.5.

Figure 5.3: Initial correspondences between reprojected and observed points.

Figure 5.4: Converged correspondences after multi-frame optimization.

Figure 5.5: Equivalent converged correspondences after single-frame optimization.

When looking at Figure 5.4, the multi-frame results don't have a very good alignment. It seems that there are conflicts in the data, since the same camera pose estimate leads to inconsistent reprojections across frames. This becomes especially aparent when comparing to the equivalent single-frame optimization results in Figure 5.5, which are much more consistent.

5.2.2 Consecutive Frames

To test if these issues persist with more similar frames, a second set of frames is used that consists of three nearby frames with a similar scene that are optimized simultaneously. The initial estimates are shown in Figure 5.6, while the converged outputs are shown in Figure 5.7.

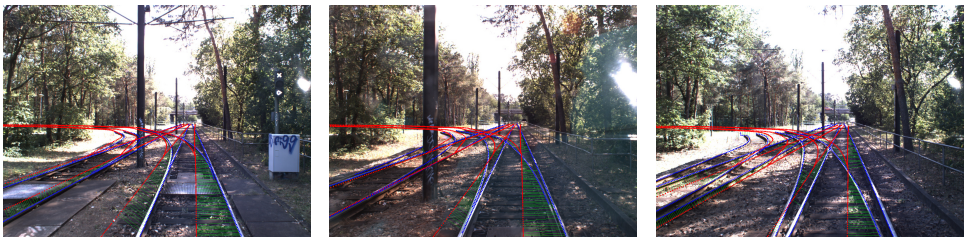

Figure 5.6: Initial correspondences between reprojected and observed points.

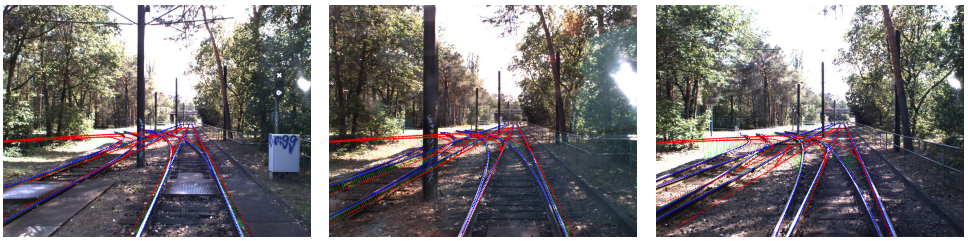

Figure 5.7: Converged correspondences after nearby multi-frame optimization.

While the alignment in Figure 5.7 seems slightly better than in Figure 5.4, it is also not very precise. The same issues with inconsistent reprojections across frames are still present. The fact that even nearby frames lead to inconsistent reprojections is a strong indicator of the data being too imprecise.

5.2.3 Analysis

From the results presented in this section, it has become clear that the data is not very precise across frames to be optimized together for consistent reprojections. The measurements tend to be somewhat noisy, which leads to inconsistent reprojections across frames. This is likely due to a combination of different factors, including RTK-GPS sensor noise, elevation data, and the railway nodes.

While the RTK-GPS sensor is very accurate globally, it is not always precise locally. Figure 5.8 shows RTK-GPS position measurements, while Figure 5.9 shows RTK-GPS rotation measurements.

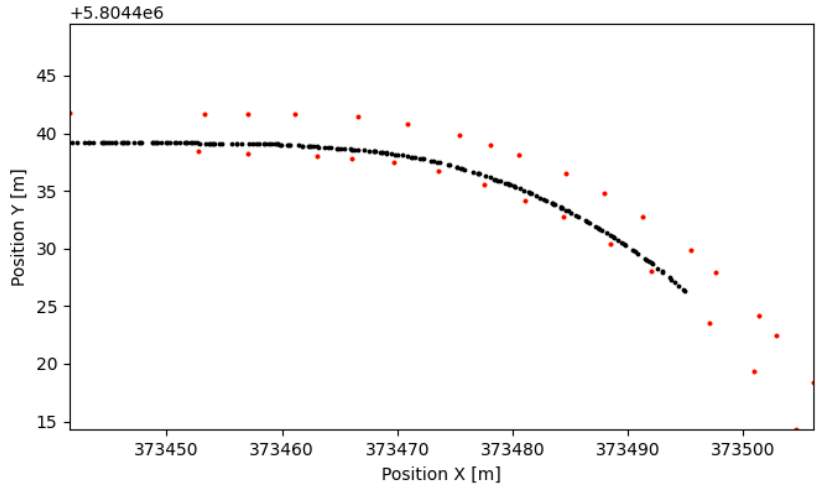

Figure 5.8: GPS position measurements (black) compared to railway nodes (red).

The position measurements seem very precise, since they all lie along a continuous line that follows the railway track. However, it could be that there are longitudinal jumps between frames, meaning that the data might not be as precise in measuring the position along the track. This could explain some inconsistencies in Figure 5.7, where the position does not seem to be consistent with the reprojected tracks.

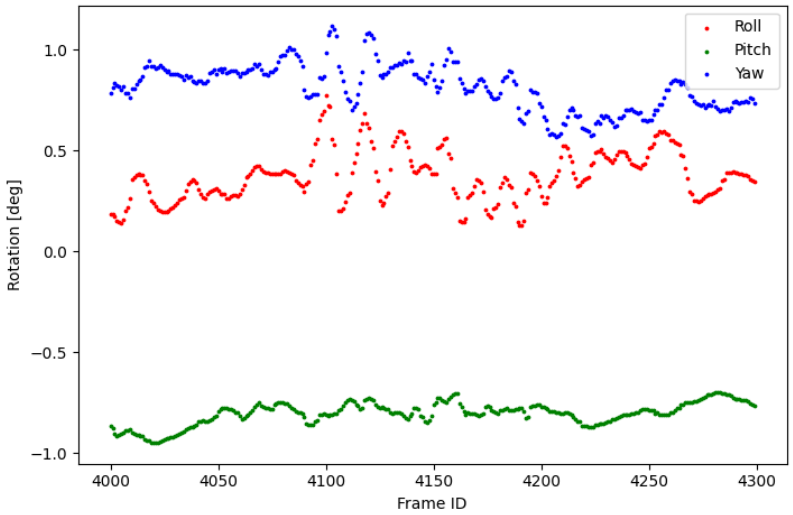

Figure 5.9: GPS rotation measurements for roll (red), pitch (green), and yaw (blue).

When it comes to RTK-GPS rotation measurements in Figure 5.9, the data is not as precise. In all three dimensions it varies by up to 0.5 degrees between frames while it should effectively stay the same on this straight track segment. Combining this small error in all three dimensions and considering the fact that small rotations can lead to large reprojection errors, it is likely a big factor in the inconsistent reprojections. This would explain the inconsistencies in Figure 5.4, where the rotation does not match the reprojections.

Comparing the RTK-GPS height measurements (blue) with the local elevation data (red) in Figure 5.10, it appears that the former are very precise, while the latter varies by up to 0.2 m across frames. Yet, small errors in the elevation data (used for creating the 3D point clouds per railway track) are unlikely to be a big factor in the inconsistent reprojections, since all frames are affected equally. Also note that, for the RTK-GPS height measurements, there have been some outliers in the data (not visible in the graph), which are likely due to bad signal or else. Avoiding these outliers is important, but otherwise the GPS height measurements do not seem to be the issue.

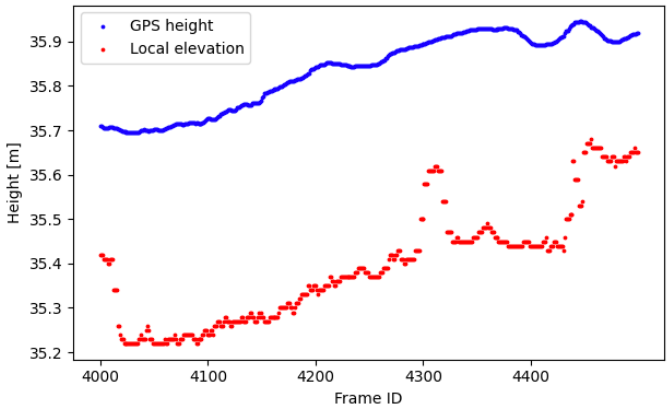

Figure 5.10: GPS height measurements (blue) vs. local elevation data (red).

Finally, the railway nodes from the OpenStreetMap data are also a possible source of error. However, similar to the elevation data they are unlikely to be a big factor in the inconsistent reprojections, since all frames are affected equally. Figure 5.8 also shows that the railway nodes look relatively precise, in addition to being globally accurate.

In conclusion, if the above data sources were more precise, the reprojections would be more consistent across frames. This would lead to better results when optimizing multiple frames together.

5.2.4 Recommendations

In order to overcome the limitations of the data and obtain more precise state estimates, it would be very useful to implement a sensor fusion algorithm, such as a Kalman Filter. This means combining the RTK-GPS sensor with available IMU and odometry data. The RTK-GPS sensor is very accurate globally, but not very precise locally. The IMU and odometry are precise locally, but drift over time. By combining all three, it is possible to obtain a more precise state estimate that is both precise locally and globally. This would lead to reduced reprojection errors across frames.

A low number of only 3 frames has purely been chosen for evaluation purposes, since this made it easier to visualize the results. In practice, the method can be applied to a much larger number of frames, which would merely take longer to optimize but also not necessarily lead to better results. Using a selection of about 5-20 different keyframes is likely a good balance between accuracy and optimization time.

Chapter 6

Conclusion

This report has presented a complete extrinsic camera calibration and reprojection pipeline for a camera attached to a railway vehicle, given a set of images with associated GPS measurements as well as global map information. The 6 degree-of-freedom camera pose is estimated using an optimization formulation, where the railway map and elevation data are processed to reproject local railway tracks into camera view for each frame, upon which the error between reprojected and detected tracks in the image is minimized, utilizing an iterative closest points (ICP) algorithm.

The proposed method has been implemented in a modular and adaptable pipeline. While the overall structure and interface of the code has been programmed in Python using an object-oriented approach, the optimization is performed in C++ using the Ceres Solver library. This allows for intuitive interaction with the code, but also provides the computational efficiency for the optimization algorithm.

Evaluation on a variety of scenes has shown that the method predictably converges within 10-50 ICP iterations, depending on scene complexity and initial pose estimate. Given a decent initial height and depth estimate, the method is robust to different track shapes and still manages to converge if the initial ICP correspondences are incorrect. More complex keyframes with depth-variability of the track actually improve the result and should be given preference as input for the optimization. It has been demonstrated that the proposed method is able to both accurately and precisely estimate the camera pose, as compared to stereo camera calibration data. Especially the horizontal and vertical positions are very accurate, while the depth error is slightly larger yet without noticeable effect. Extending to multi-frame optimization, the accuracy of the method is limited by sensor noise, in particular the RTK-GPS rotation measurements, where small angular errors lead to reprojection inconsistencies. Beyond this limitation, everything is in place to perform consistent and accurate multi-frame optimization.

An extension to this project to address the limitation of multi-frame optimization is sensor fusion of GPS measurements with IMU and odometry data. This would improve the accuracy of the state estimate and thus the reprojection error in multi-frame optimization. Another extension would be automated track detection using machine learning, which would allow the method to be applied to arbitrary scenes without manual annotation.

Bibliography

[1] Z. Wang, X. Wu, Y. Yan, C. Jia, B. Cai, Z. Huang, G. Wang, and T. Zhang, "An inverse projective mapping-based approach for robust rail track extraction," in *2015 8th International Congress on Image and Signal Processing (CISP)*. IEEE, 2015, pp. 888–893.

[2] H.-Y. Lin *et al.*, "3d object detection and 6d pose estimation using rgb-d images and mask r-cnn," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2020, pp. 1–6.

[3] N. Spiegelhalter, C. Von Einem, and D. Hug, "Online estimation of camera extrinsics using map information," 2023.

[4] K. M. Dawson-Howe and D. Vernon, "Simple pinhole camera calibration," *International Journal of Imaging Systems and Technology*, vol. 5, no. 1, pp. 1–6, 1994.

[5] J. Weng, P. Cohen, M. Herniou *et al.*, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[6] OpenCV. Camera calibration and 3d reconstruction - fisheye camera model.

[7] Z. Zhang and P. J. Besl, "Iterative point matching for registration of free-form curves and surfaces," *International Journal of Computer Vision*, vol. 13, no. 2, pp. 119–152, 1994.

[8] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[9] S. Agarwal, K. Mierle, and T. C. S. Team, "Ceres Solver," 3 2022.

[10] M. Dittrich. Strassenbahnen in potsdam.

[11] L. und Geobasisinformation Brandenburg. Digitales geländemodell.

[12] W. Jakob and other contributors, "pybind11 — seamless operability between c++11 and python," 2023.