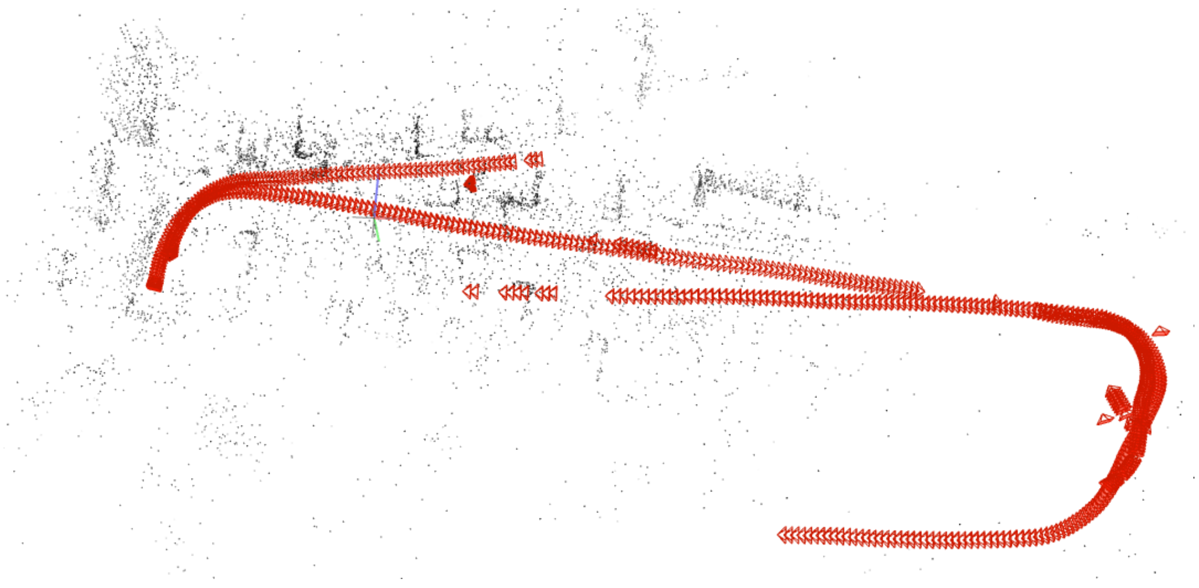
Reconstructing vehicle tracks from a fish-eye lens dataset



Henk Schaapman

Layout: typeset by the author using LaTeX.
Cover illustration: Author, made using COLMAP

Reconstructing vehicle tracks from a fish-eye lens dataset

A comparison of COLMAP and hloc pipelines

Hart Th. H. Schaapman
11676892

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. A. Visser

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

January 21st, 2022

Abstract

The task of localization is central to autonomous driving. Simultaneous Localization And Mapping (SLAM) and Structure-from-Motion (SfM) algorithms provide localization in and reconstructions of an environment from images. In this thesis, two SfM pipelines are tested on images from the IPLT dataset, a dataset containing images taken by two fish-eye lens cameras, front and rear, on a vehicle moving trough a parking area. A comparison was made between COLMAP and hloc, two pipelines capable of sparse reconstruction. SIFT, used by COLMAP for feature matching, had difficulty matching features from different perspectives, in two images taken by different cameras. SuperPoint and SuperGlue, part of the hloc pipeline for feature extraction and matching, proved valuable for reconstructions involving opposing perspectives in the IPLT dataset. The feature matches generated by SuperPoint and SuperGlue proved more reliable in producing reconstructions with geometrically correct angles. The Unified Camera Model was implemented for use in the pipelines, allowing known IPLT intrinsic camera parameters to be used in the reconstruction process. The effect of the Unified Camera Model implementation was compared with existing camera models in the pipeline.

Contents

Chapter 1

Introduction

According to a 2015 report by the US National Highway Traffic Safety Administration, the percentage of vehicle crashes caused by the driver in the US at 94% [1]; progress in research into autonomous vehicles could prevent many accidents.

One of the crucial tasks for autonomous vehicles is localization: finding the ego-position in an environment [2]. Simultaneous Localization And Mapping (SLAM) is one of the techniques used for this purpose [3]. SLAM entails creating a map or model of the environment while simultaneously placing oneself in this model. SLAM techniques have contributed significantly to the world of autonomous driving [4].

The question of whether the SLAM problem is a solved problem is sometimes asked in the AI community. In their review of the current state of SLAM, the authors of [5] say this question cannot be answered without providing information about the robot/vehicle, the environment and the performance, since questions remain open in the field: SLAM could be more robust still, for example, and currently lacks a high-level, semantic understanding [5].

Modern autonomous vehicles often have multiple sensors. Different SLAM algorithms involve different streams of data: LIDAR, for example, or RGB-D images. Visual SLAM (VSLAM) is defined as performing the SLAM task using plain images only. This will be the focus of this thesis.

Structure-from-Motion (SfM), closely related to VSLAM, entails recovering the three-dimensional structure of a scene from a set of 2D images. Recently, neural network architecture approaches have been included in SLAM and SfM. SfM-net [6], for example, is an end-to-end neural network approach to SfM. Another example is SuperGlue [7], a feature matching neural network, trained end-to-end

on image pairs. Many SLAM algorithms are split up into a front-end for visual feature extraction and a back-end for bundle adjustment or pose estimation [8]. SuperGlue is structured in such a way to be a *middle-end*: it matches keypoints, but it is trained end-to-end on images. Its creators believe that "when combined with a deep front-end, SuperGlue is a major milestone towards end-to-end deep SLAM." [7]. When combined with SuperPoint [9], a deep feature extraction neural network, SuperGlue achieves state-of-the-art results on the task of pose estimation in challenging real-world environments [7]. It can be integrated into other software to be used to perform SfM or SLAM tasks.
SuperGlue was integrated by its creators into an SfM-pipeline called hloc [10]. It uses the feature matching capabilities of SuperGlue, combined with the 3D scene reconstruction functionalities of COLMAP [**?**], an SfM pipeline created in 2016. In this thesis, the COLMAP pipeline, using SIFT [11] for feature matching, is compared to the hloc pipeline, using SuperGlue for feature matching and COLMAPs reconstruction functionalities. The testing is done on the Institut Pascal Long-Term (IPLT) dataset [12].

The Institut Pascal Long-Term dataset [12] is a dataset suitable for SLAM research purposes and contains challenging conditions, such as recording at different times of day and changing weather conditions. Two wide-angle fish-eye lenses with a field of view (FOV) of 100 degrees were used to record the images. The dataset consists of sequences of images taken by cameras on a vehicle, that followed the same path along a parking area multiple times over a period of 16 months. Besides providing images from a front and back camera, the dataset contains lidar data, GPS data and parameters of the cameras expressed in the Unified Camera Model [13]. Tags are provided to describe the circumstances in each image, e.g.: "day", "dusk", "rain" or "fog".
For this thesis, the author implemented the Unified Camera Model in the COLMAP pipeline, allowing it to be used in the COLMAP and hloc pipelines.

This thesis compares the effect of using the different feature matching processes (SIFT and SuperGlue) on the reconstruction process of the IPLT dataset. This might provide insight into the benefits of neural network approaches to feature matching in challenging conditions.

Chapter 2

Theoretical framework

This chapter provides a background and explanation of relevant topics for this thesis. Firstly, a brief background on autonomous driving is provided. Secondly, the feature extraction and matching algorithms used in this thesis (SIFT and SuperPoint+SuperGlue) are explained. After a background then on SLAM, SfM and camera projection models, the dataset used in this thesis will be introduced. Finally, the pipelines used in this paper, COLMAP and hloc, are described.

2.1 Autonomous driving

The 2019 autonomous driving (AD) survey by Yurtsever et al. [3] shows that clear gaps in the AD research remain. The optimal sensor for use in localization, mapping and perception is still disagreed upon, and algorithms still lack efficiency and accuracy, to name some examples [3]. The survey also lists multiple techniques employed in autonomous driving, both modular and end-to-end ones [3]. As mentioned before, SLAM techniques have contributed significantly to the world of AD [4]. SLAM is mentioned in the survey as having a medium-high robustness, a high accuracy, but a very high computational cost [3].

2.2 Features

In this section, relevant feature extraction and matching techniques will be described.

2.2.1 Features/keypoints

The terms *features* and *keypoints* will be used interchangeably in this thesis. Features are points of interest in an image, to be matched with features in other images. Feature extraction entails finding suitable features in an image. Feature matching is the process of finding the same point in two or more images.

A keypoint *descriptor* describes characteristics of a keypoint and is used as a metric to match keypoints (as will be showed in Section 2.2.2).

2.2.2 Scale Invariant Feature Transform

First introduced by David Lowe [11], the well-known Scale Invariant Feature Transform is a method for finding features in images and comparing them, to find the same features in different images.

Finding keypoints

The first step taken by SIFT involves convolution with a Gaussian kernel. A scale space $L(x, y, \sigma)$, first introduced by Witkin [14], is produced by the convolution of a Gaussian $G(x, y, \sigma)$ with an input image $I(x, y)$ [11]:

$$l(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.1}$$

with convolution sign $*$ and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.2}$$

This $L(x, y, \sigma)$ represents a "smoothed out" version of the image.
A difference-of-Gaussian (DoG) $D(x, y, \sigma)$ can be calculated, which finds so-called *scale space extrema*. Two Gaussians with different scales σ are subtracted from one another, and then convolved with the image:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned} \tag{2.3}$$

The result of this operation is equal to performing the convolution on both images and subtracting the resulting scale spaces.

In this DoG, local extrema are found, by comparing pixels to their spatial neighbours (8) and their neighbours in scale (16).

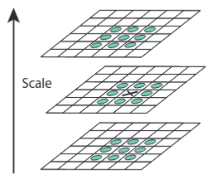


Figure 2.1: Neighbours in space and scale [11]

The local extrema are selected based on their suitability, meaning some will need to be rejected. This rejection process involves the Taylor expansion up to quadratic terms of the DoG $D(x, y, z)$, and then setting its derivative with respect to $\mathbf{x} = (x, y, z)$ to zero. This process removes keypoints with low contrast.

Matching keypoints

The keypoint descriptor creation process is described in this paragraph, as it works in SIFT [11]. A descriptor is generated for each keypoint. In the SIFT algorithm, this happens by first using the scale of the keypoint (the magnitude of the gradient at the location of the keypoint) to select an L, one of the smoothed out Gaussians (one of the layers in figure 2.1). This makes sure the descriptors are created in a scale-invariant manner. Using this image $L(x, y)$, a gradient magnitude, $m(x, y)$, and a gradient orientation, $\theta(x, y)$, for this scale can be computed, using the difference between neighbouring pixels:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}, \quad (2.4)$$

$$\theta(x, y) = tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y))). \quad (2.5)$$

This computation is done for a number of sample points around the keypoint location, as can be seen in figure 2.2. The orientations $\theta(x, y)$ are entered into a histogram, divided into 36 bins for the 360 degrees. Each entry into this histogram is weighted by its magnitude $m(x, y)$ and by "a Gaussian-weighted circular window with a that is 1.5 times that of the scale of the keypoint" [11]. The histogram is illustrated on the right in figure 2.2. This image, however, shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in the SIFT paper use 4x4 descriptors computed from an 16x16 sample array [11].

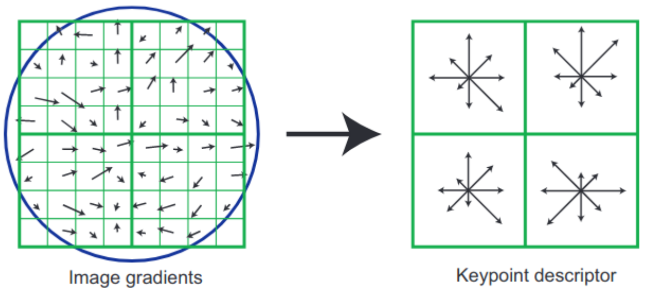


Figure 2.2: SIFT keypoint descriptor creation [11]

Keypoint descriptors are matched as follows: the best candidate match is found by finding its nearest neighbour in the database of training image keypoints, the nearest neighbour being the keypoint with minimum Euclidean distance in the space of descriptors [11].

2.2.3 Superpoint

Superpoint [9] is a framework for training detectors of interest points in images, and creating descriptors to go along with them. Superpoint is a fully convolutional network: it takes in full images, and in one pass computes both the interest points and the descriptors [9]. In this thesis, these points and descriptors are then passed on to serve as input for the SuperGlue algorithm.

2.2.4 SuperGlue

SuperGlue is the feature-matching neural network used in the hloc pipeline. It makes use of an Attentional Graph Neural Network. This architecture enables it to disambiguate similar keypoints or repeating patterns in an image, enabling SuperGlue to perform better in some challenging conditions [7]. Before describing the network architecture, two concepts will be elaborated upon below.

Multi-Layer Perceptron
A Multi-Layer Perceptron is used as a blanket term to describe feed-forward neural networks. Two separate MLP's are defined in SuperGlue's architecture [7].

Graph Neural Networks
A graph is simply a set of nodes with edges between them. These edges can either be directional, meaning one-way, or undirected, meaning two-way. In the case of the SuperGlue network, the keypoint descriptors of two images are the graph nodes of the first layer in the GNN. The edges are undirected, and are split into

self edges ε_{self}, connecting keypoints from the same image, and inter-image edges ε_{cross}, connecting keypoints to keypoints from another image.

Network Architecture

SuperGlue consists of two main parts: an Attentional Graph Neural Network and an Optimal Matching Layer. Its structure can be seen in the figure below:

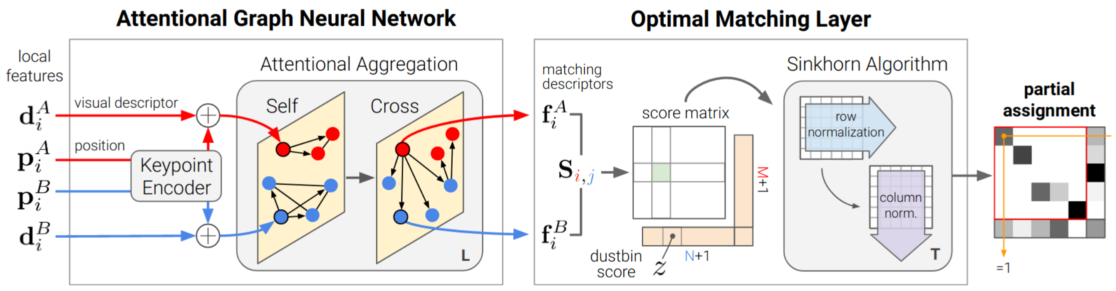


Figure 2.3: **The SuperGlue Architecture** [7].

Attentional Graph Neural Network

Keypoints (extracted by a feature extraction algorithm, e.g. SIFT/SuperPoint) $\mathbf{p}_i := (x, y, c)_i$ (coordinates x and y as well as detection confidence c) and visual descriptors $\mathbf{d}_i$ are combined into high-dimensional vector $\mathbf{x}_i$, with $\mathbf{p}_i$ being encoded into a multilayer perceptron: ${}^{(0)}\mathbf{x}_i = \mathbf{d}_i + \mathrm{MLP}_{\mathrm{enc}}(\mathbf{p}_i)$.

All vectors ${}^{(0)}\mathbf{x}_i$ are fed into a message passing graph neural network in the following way:

$$ {}^{(l+1)}\mathbf{x}_i^A = {}^{(l)}\mathbf{x}_i^A + \mathrm{MLP}\left(\left[{}^{(l)}\mathbf{x}_i^A \,\|\, \mathbf{m}_{\varepsilon\to i}\right]\right). \tag{2.6} $$

The vectors $\mathbf{x}_i$ get updated by way of a message: the concatenation of the vector $\underline{\mathbf{x}_i^A}$ and the message $\underline{\mathbf{m}_{\varepsilon\to i}}$ it receives are fed into an MLP, the outcome of which forms the next iteration of $\mathbf{x}_i$.

The message is calculated using aggregation, a weighted average of all the messages received from the representations of every other keypoint: "Akin to database retrieval, a representation of i, the query $\mathbf{q}_i$, retrieves the values $\mathbf{v}_j$ of some elements based on their attributes, the keys $\mathbf{k}_j$" [7]. These keys are used to calculate the weights in this average, called *attention weights*, by using their similarity to the query: $\alpha_{ij} = \mathrm{Softmax}_j(\mathbf{q}_i^{\mathrm{T}}\mathbf{k}_j)$. These weights determine how much the value from each other keypoint will contribute to the aggregated message, in the following way:

$$\mathbf{m}_{\varepsilon \to i} = \sum_{j:(i,j)\in\varepsilon} \alpha_{ij}\mathbf{v}_j. \tag{2.7}$$

The queries, keys and values are all linear projections of deep features in the neural network. The projection parameters "are learned and shared for all keypoints of both images" [7].

This attention structure allows for information flow between the keypoints representations in the layers. This, combined with the fact that the keypoint's position is encoded in its representation as well, enables SuperGlue to, to some degree, disambiguate keypoints that look alike or are part of a repeating pattern in an image.

The final matching descriptors are linear projections as well: they are projections of the parameters in the last layer of the network.

$$\mathbf{f}_i^A = \mathbf{W} \cdot {}^{(L)}\mathbf{x}_i^A + \mathbf{b}, \qquad \forall i \in A. \tag{2.8}$$

Multihead attention [15] is used to improve expressivity.

Optimal Matching Layer
In the second part of the SuperGlue algorithm, a score matrix is computed. Entry $\mathbf{S}_{i,j}$ in this matrix represents the score between keypoint i and j from image $\mathcal{A}$ and $\mathcal{B}$ respectively. The score matrix is populated in the following way:

$$\mathbf{S}_{i,j} =< \mathbf{f}_i^A, \mathbf{f}_j^B >, \forall (i,j) \in \mathcal{A} \times \mathcal{B}, \tag{2.9}$$

,
where <·,·> is the inner product.

After using dustbins to allow the network to suppress unmatched keypoints, the final assignment **P** is obtained by calculating the optimal transport between distributions **a** and **b**.

2.3 SLAM

Simultaneous Localization And Mapping (SLAM) entails creating a map or model of the environment and simultaneously placing oneself in this model. While a robot moves through an environment, it updates its environment model and its estimation of its location and orientation. Cadena et al. [5] describes that the model is used twofold: one, for other tasks, such as enabling the robot to perform tasks in its environment and creating a visualization for humans, and two, to help update its estimation of its state (position and orientation). Some SLAM tasks involve a-priori knowledge, for example: a robot navigating a pre-mapped space, or a space that has beacons installed for the robot to use as guidance [5]. In other problems, the map is being built from scratch using the robots sensors, as the robot moves through the environment for the first time. Different SLAM algorithms run by using different kinds of sensor data. ORB-SLAM3 [16], for example, is a SLAM method that is able to use either monocular, stereo or RGB-D cameras. The more detailed data the algorithm is allowed to use, the better its performance.

Some SLAM algorithms, like COLMAP [17], extract features from images and then attempt to match the features to find the same point in different images. The feature matches can then be used to create the 3D model of the environment. Other SLAM algorithms for example use the images directly, and track the camera orientation from image alignment, like in Engel et al. [18]. Different SLAM methods use different streams of data. Like mentioned before, SLAM methods using images are called Visual SLAM (VSLAM) methods.

2.4 Structure from Motion

The problem of Structure from Motion (SfM) is closely related to the SLAM problem: [19] describes the SLAM problem as a specific case of the SfM problem. One distinction between SLAM and SfM lies in the fact that SfM is mostly an offline process, while SLAM can also be performed in real-time. Another distinction is that images are the main input for an SfM algorithm, possibly with GPS or odometry data, while SLAM sometimes involves many other streams of data, as seen for example in ORB-SLAM3 [16].

The process of SfM can, according to [19], be broken down into three steps: (1) the extraction of features in images and the matching of features between these images, (2) camera motion estimation and (3) calculating the 3D structure from the features and camera motions. The SfM-net end-to-end approach does not adhere to these steps. Its Convolutional Neural Network (CNN) has multiple components chained together, but not exactly in the order displayed above. It does not use a feature extraction step: it employs a separate, parallel motion network and structure network, that take in a video frame pair and a video frame respectively. The point cloud resulting from the structure network is used together with the output of the motion network to create the reconstruction [6].

2.5 Camera Models

When performing a SLAM or SfM task, cameras with a wide angle or large field of view can be beneficial [20]. An example of these is the fish-eye camera: a camera with a large field-of-view lens. A system using only lenses is called a dioptric system. Another example of a large field-of-view system is called a catadioptric system: a camera system consisting of both refractive and reflective components, of both lenses and mirrors. To calculate the projection process in these systems, many models have been proposed.

COLMAP comes with eleven built-in camera models: *simple_pinhole, pinhole, simple_radial, radial, opencv, full_opencv, simple_radial_fisheye, radial_fisheye, opencv_fisheye, fov* and *thin_prism_fisheye* [21]. Some information about their use in COLMAP can be found in the COLMAP tutorial [21]. The Unified Camera Model, one of the models used in this thesis, is not one of them, and was implemented in COLMAP (further explained in Section 3.2.1). Several of the models will be elaborated upon below.

2.5.1 Pinhole model

The pinhole camera model is the simplest of the camera models. It can be visualized in the following way (the *scene* being that which is photographed): the light from a scene passes through a small hole, and is projected onto a screen behind this hole, thereby forming a projection from the 3D scene to the 2D screen (a side effect is that the image is mirrored along both axes). It is described as having four projection parameters [22]: $\mathbf{i} = [f_x, f_y, c_x, c_y]^T$, with f being the focal length and c the principal point parameters. The projection function is as follows:

$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x \frac{x}{z} \\ f_y \frac{y}{z} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}. \tag{2.10}$$

COLMAP has a simple pinhole camera model, which uses only three parameters $[f, c_x, c_y]^T$. This means that in the projection process, the image can be "stretched" or "compressed" only at the same rate in the x direction as in the y direction.

2.5.2 Field-Of-View model

The Field-Of-View (FOV) camera model was created to be a new way to model fish-eye lenses (wide-angle cameras that use only lenses for the projection). Its design is meant to mimic the non-linear radial distortion effects that a fish-eye lens created [23]. Earlier ways to address these non-linear distortions were a logarithmic or a polynomial model, as seen for example in the Fish-Eye Transform (FET) or the Polynomial FET (PFET) [24]. The FOV models this distortion using one parameter w, which is the field of view in degrees of the ideal fish-eye lens that the modeled lens corresponds to [23]. This means it will not be exactly equal to the actual field of view of the lens being modelled. With the extra parameter as an extension of the pinhole model, FOV's parameters are $\mathbf{i} = [f_x, f_y, c_x, c_y, w]^T$.

2.5.3 Unified Camera Model

The Unified Camera Model (UCM) was first proposed in 2001 to be a unifying model for all central catadioptric cameras [25]. Catadioptric camera systems utilize a system of mirrors and lenses; a central catadioptric camera is one with a single central viewpoint [25]. The model has been shown to work with fish-eye cameras [26], although the fit with any given fish-eye lens is not necessarily perfect, which means an additional distortion model is sometimes added [22]. The image is projected onto a unit sphere in this model, then onto the image plane of a pinhole camera [22]. The model works with five parameters $\mathbf{i} = [f_x, f_y, c_x, c_y, \xi]^T$, with the projection function:

$$\begin{aligned} \pi(\mathbf{x}, \mathbf{i}) &= \begin{bmatrix} \gamma_x \frac{x}{\xi d + z} \\ \gamma_y \frac{y}{\xi d + z} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}, \\ d &= \sqrt{x^2 + y^2 + z^2}. \end{aligned} \tag{2.11}$$

The parameters of cameras used for creating the images in the IPLT dataset [12] were included in the dataset, and provided in the UCM format. For the COLMAP UCM implementation in this thesis, the rewritten projection and unprojection functions as proposed in [22] were used:

$$\pi(\mathbf{x}, \mathbf{i}) = \begin{bmatrix} f_x \frac{x}{\alpha d + (1-\alpha) z} \\ f_y \frac{y}{\alpha d + (1-\alpha) z} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}. \tag{2.12}$$

This rewritten version has parameters $\mathbf{i} = [f_x, f_y, c_x, c_y, \alpha]^T$, with $\alpha \in [0, 1]$. Since the model was simply rewritten, the parameter can be converted as follows: $\xi = \frac{\alpha}{1-\alpha}$, $\gamma_x = \frac{f_x}{1-\alpha}$, $\gamma_y = \frac{f_y}{1-\alpha}$.

The unprojection function of the rewritten model is as follows:

$$
\begin{aligned}
\pi^{-1}(\mathbf{u},\mathbf{i}) &= \frac{\xi + \sqrt{1+(1-\xi^2)r^2}}{1+r^2}\begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \xi \end{bmatrix}, \\
m_x &= \frac{u-c_x}{f_x}(1-\alpha), \\
m_y &= \frac{u-c_y}{f_y}(1-\alpha), \\
r^2 &= m_x^2 + m_y^2, \\
\xi &= \frac{\alpha}{1-\alpha}.
\end{aligned} \tag{2.13}
$$

The conversion of the Unified Camera Model IPLT dataset camera parameters to the rewritten form can be found in Appendix C.

Extended Unified Camera Model

Extensions or expansions of the UCM have been proposed. One example is the Extended Unified Camera Model (EUCM), a model with one extra parameter β, thus with six in total [27]. This EUCM can be described with the tuple $\mathbf{i} = [f_x, f_y, c_x, c_y, \alpha, \beta]^T$. In the "regular", non-rewritten UCM, ξ denotes the distance from the pinhole camera to the center of the unit sphere (and thus $\frac{\alpha}{1-\alpha}$ denotes this distance for the rewritten version and for the EUCM) [22]. The EUCM does not have a unit sphere, but uses the parameter β to transform the unit sphere into an ellips for the projection.
The EUCM works well for both catadioptric and fish-eye cameras, without an additional distortion model; the further improvement in reprojection error from EUCM to UCM-D (a UCM with an additional distortion model, totalling 10 parameters) is negligible, but the EUCM is significantly less computationally intensive [27].

2.6 Institut Pascal Long-Term dataset

The Institut Pascal Long-Term dataset [12] is used for localization algorithms, and was created with challenging conditions in mind. The dataset consists of sequences of images taken by cameras on a vehicle, driving along the same path in a parking area multiple times over a period of 16 months. Providing images from a front and rear camera, the dataset contains other streams of data: intrinsic and extrinsic camera parameters, wheel odometry and GPS data and lidar readings.
The shape of the track driven by the vehicle looks like this:

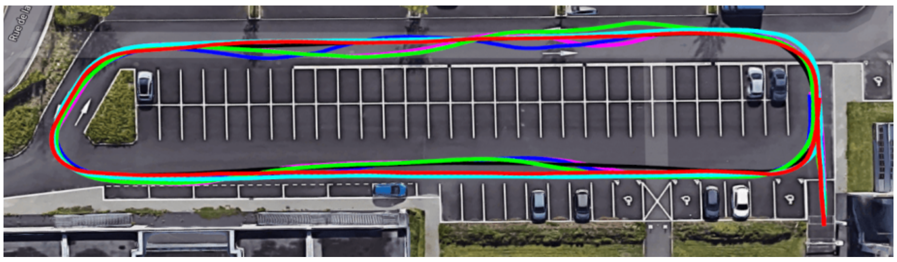


Figure 2.4: Shape of the tracks driven by IPLT dataset vehicle [12]

The challenging aspect of this dataset is twofold. Firstly, data recording is done in a parking area, which means that some days the cars are in different spots than other days. The parked cars can thus not always be used in the localization process, and a localization algorithm would have to treat them as noise to be looked past. Secondly, the weather and the time of day can differ every time the data was recorded. Each data recording (one lap around the parking area) is assigned a tag: "sun", "cloudy", "dusk", "rain" or "fog". The data can thus be grouped into different groups, each with similar environmental circumstances. Appendix A provides a table of these groups and some examples of images from the groups.

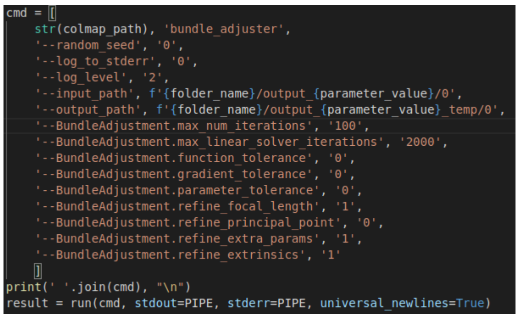
2.6.1 ROS

Every data recording (one lap around the parking area: one front_camera folder, one back_camera folder and the other data) is stored in a rosbag [28]. A rosbag, with file format extension `.bag`, combines several sensor streams with their timestamp, which makes the data easily accessible for further experimentation.

2.7 COLMAP

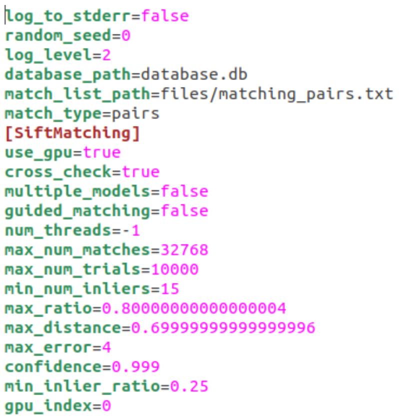
2.7.1 COLMAP architecture

COLMAP has both a graphical and a command-line interface. Both exhibit the full functionality of COLMAP, aside from one aspect: the graphical interface allows the user to view a generated reconstruction, which is not possible from the command line. The benefit of the command-line interface is that it can be called from a script, meaning one could write a script that performs an experiment several times, incrementally changing a parameter, for example. COLMAPs command-line interface has 43 available commands. Excluding "help" and "gui", COLMAP has 41 different functionalities. Each of these functionalities comes with some required arguments, for example: when running the command `colmap feature_extractor`, among others, the argument `--image_path` (the location of the images) is required. Each function has many optional arguments as well. All arguments can be passed to a function in the command line, or in a file. In the case of the latter, the only argument passed to the function is the location of said file.



```
cmd = [
    str(colmap_path), 'bundle_adjuster',
    '--random_seed', '0',
    '--log_to_stderr', '0',
    '--log_level', '2',
    '--input_path', f'{folder_name}/output_{parameter_value}/0',
    '--output_path', f'{folder_name}/output_{parameter_value}_temp/0',
    '--BundleAdjustment.max_num_iterations', '100',
    '--BundleAdjustment.max_linear_solver_iterations', '2000',
    '--BundleAdjustment.function_tolerance', '0',
    '--BundleAdjustment.gradient_tolerance', '0',
    '--BundleAdjustment.parameter_tolerance', '0',
    '--BundleAdjustment.refine_focal_length', '1',
    '--BundleAdjustment.refine_principal_point', '0',
    '--BundleAdjustment.refine_extra_params', '1',
    '--BundleAdjustment.refine_extrinsics', '1'
    ]
print(' '.join(cmd), "\n")
result = run(cmd, stdout=PIPE, stderr=PIPE, universal_newlines=True)
```

Figure 2.5: Passing arguments in the command line via a python script



```
log_to_stderr=false
random_seed=0
log_level=2
database_path=database.db
match_list_path=files/matching_pairs.txt
match_type=pairs
[SiftMatching]
use_gpu=true
cross_check=true
multiple_models=false
guided_matching=false
num_threads=-1
max_num_matches=32768
max_num_trials=10000
min_num_inliers=15
max_ratio=0.80000000000000004
max_distance=0.69999999999999996
max_error=4
confidence=0.999
min_inlier_ratio=0.25
gpu_index=0
```

Figure 2.6: Passing arguments in a file

The COLMAP pipeline creates an sqlite database, where it stores the image paths, features, feature matches and camera info/parameters. The reconstruction process then takes this database and creates a 3D reconstruction, which it stores in a designated output folder in the form of three binary files: one for the cameras, one for the images and one for the 3D points. These files can then be opened by the COLMAP GUI to view the reconstruction.

The following subsections outline COLMAP's structure: it adheres to the three steps mentioned in Section 2.4 (feature extraction, camera motion estimation, calculating 3D structure) quite well.

2.7.2 Feature extraction

COLMAP utilizes the SIFT algorithm for extraction of the features.

2.7.3 COLMAP camera models

The COLMAP software has built-in support for camera intrinsic parameters in eleven different models, mentioned in Section 2.5. As mentioned in the COLMAP tutorial, not all camera models are suitable for use with fisheye lens cameras. Only *simple_ radial_fisheye, radial_fisheye, opencv_fisheye, fov* and *thin_ prism_fisheye* are suitable for this [21].

2.7.4 Feature Matching

COLMAP has several modes for feature matching [29]:

Exhaustive: Every image is matched with every other image, using SIFT.
Custom: A file containing image pairs is provided to the algorithm. These pairs are matched using SIFT.
Sequential: Each image is matched to a certain amount of its neighbours in name: this means that the images have to be named sequentially. This mode has built-in loop detection using a pre-built vocabulary tree.
Vocabulary Tree: Images are matched [30] using a pre-trained visual vocabulary tree.
Spatial: This image mode requires GPS or odometry data to be passed to the algorithm: the images will be matched with their spatial neighbours, using SIFT.
Transitive: This image mode matches image A with C, if A had many matches with B, and B with C, using SIFT.

2.7.5 Reconstruction

Two of the 41 COLMAP functions mentioned earlier (in Section 2.7.1) are the `mapper` and the `hierarchical_mapper`. These are the tools that COLMAP provides for creating a 3D reconstruction from feature matches. Many different arguments can be passed to the mapper functions to change parameters in the reconstruction process.

During the reconstruction process, COLMAP also calculates and refines the camera parameters of the cameras used in a reconstruction and saves them to a database, unless they are provided from the start.

2.8 hloc - the hierarchical localization toolbox

The winner of CVPR 2020's Indoor/Outdoor localization challenges [31] was a Hierarchical Localization algorithm `hloc` working with SuperPoint, a fully convolutional feature extraction algorithm [9], and SuperGlue, a Graph Neural Network for feature matching [10].
The hloc toolbox include multiple pipelines: one pipeline called *pipeline_ Aachen.ipynb*, applied to the Aachen Day-Night dataset [32] and one called *pipeline_ InLoc.ipynb*, applied to the InLoc dataset [33]. Both serve as examples of an application of the hloc pipeline, and to show its performance. The third pipeline is called *SfM_pipeline*. In the example, it is applied to the South_Building dataset [34], but it can easily be applied to another set of images. This third pipeline was used in this thesis.

All three pipelines share some "backend" scripts, but the first two utilize some scripts specific to their respective datasets.

2.8.1 Pipeline structure

The hloc pipelines are structured as follows: SuperPoint and SuperGlue do the feature extraction and matching, then the COLMAP `mapper` is used for the reconstruction. In this thesis, this point onwards, the third pipeline mentioned in the paragraph above, *SfM_pipeline*, will be referred to as the hloc pipeline.
The hloc pipeline extracts and matches features with SuperPoint+SuperGlue, and then, after creating an sqlite database in COLMAP format, imports the matches into said database. The COLMAP (hierarchical) mapper can then be run from this database to create the reconstruction, as mentioned in Section 2.7.1.

Chapter 3

Method

In this chapter, the data and the software used in this thesis will be described in further detail, as well as the way they were utilized and the changes made to them.

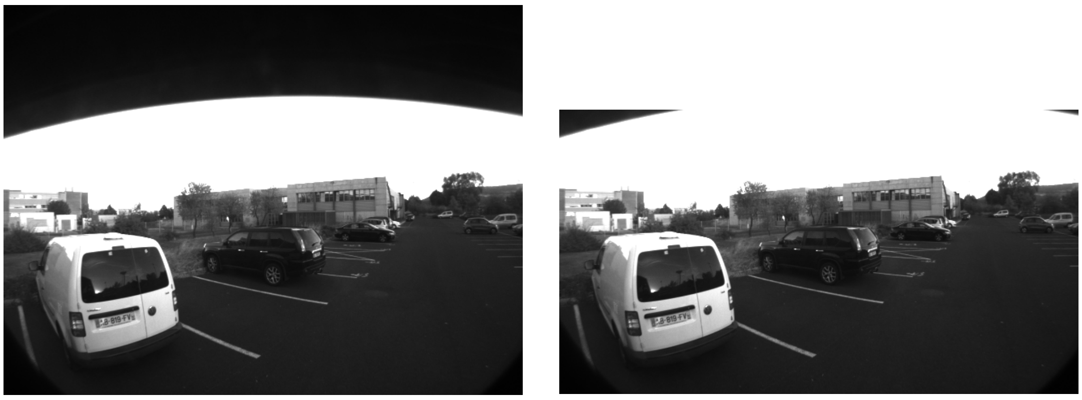
3.1 Institut Pascal Long-Term dataset

3.1.1 Extraction

For this thesis, a script was written for convenient extraction of multiple rosbags at once. The script extracts the images, extracts GPS and odometry data, crops images if desired (see Section 3.1.2), mirrors images from either the front camera or the rear camera folder if desired (see Section 3.4.3), or takes a subset of the images and deletes the rest if desired (mentioned in Section 4.1.2).

3.1.2 Modifications

In the first part of the dataset, the rear camera captures a black bar at the top of each image, which makes up approximately 1/4th of the image. The camera configuration on the vehicle taking the images was changed a little after halfway through the data collection, so in the images taken by the rear camera from the 10th of January 2019 onward the black part is somewhat smaller. The images taken by the front camera do not have this black part present.



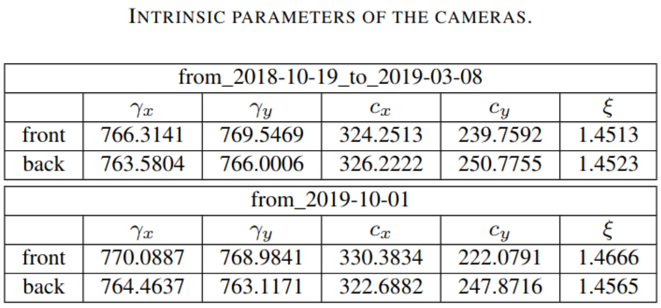
(a) original rear camera image (b) cropped rear camera image

Figure 3.1: images from 2018-10-22-18-24-58.bag, with the larger black part

The decision was made to crop part of the rear camera images, to get rid of the black bars at the top, before using them (thus, images from before the 10th of January 2019 were cut somewhat shorter than those from after). This was done only to the images made by the rear camera, since the front camera does not have these black parts.

3.1.3 Camera model

The images in the Institut Pascal Long-Term Dataset are taken by 100° field-of-view (FOV) lenses. The intrinsic parameters for these cameras are provided by the creators of the dataset. They are expressed in the Unified Camera Model, and are as follows:



INTRINSIC PARAMETERS OF THE CAMERAS.

from_2018-10-19_to_2019-03-08					
	γ_x	γ_y	c_x	c_y	ξ
front	766.3141	769.5469	324.2513	239.7592	1.4513
back	763.5804	766.0006	326.2222	250.7755	1.4523

from_2019-10-01					
	γ_x	γ_y	c_x	c_y	ξ
front	770.0887	768.9841	330.3834	222.0791	1.4666
back	764.4637	763.1171	322.6882	247.8716	1.4565

Figure 3.2: IPLT dataset intrinsic camera parameters [12]

The parameters provided with the IPLT dataset are in the form $[\gamma_x, \gamma_y, c_x, c_y, \xi]$. For use of the parameters in COLMAP, the parameters were converted to a rewritten version of the model, further elaborated upon in Section 2.5.3.

3.2 COLMAP

COLMAP is described by its creators as a "general-purpose Structure-from-Motion (SfM) and Multi-View Stereo (MVS) pipeline" [17], and extensively described in Section 2.7. The implementation details are described in this Section. The original code is publicly available at [35].

3.2.1 COLMAP UCM implementation

The Unified Camera Model [22], which was used to express the intrinsic camera parameters in the IPLT dataset, is not one of the eleven mentioned in 2.5. COLMAP allows for the possibility to implement a custom camera model, following the steps laid out in the source code file `camera_models.h` [36]. These steps were taken for the UCM, in the rewritten form mentioned in Section 2.5.3. The code can be found in Appendix B. The IPLT dataset camera parameters could thus be used in the COLMAP software in the reconstruction process.

3.2.2 Feature Matching

Of the COLMAP feature matching modes mentioned in 2.7.4, the exhaustive, sequential and custom matching modes were used for this thesis. The exhaustive matching mode worked well for smaller image collections, and will be featured in the results. For using larger image collections, however, the amount of image pairs increases quickly.

The sequential feature matching mode seems a good choice for this dataset, since it automatically generates image pairs of images taken closely in time to each other, and because it has built-in loop detection. However, because images have to be named sequentially, only the images from one rosbag at a time can be used, since images from other rosbags have a different timestamp. The matching mode was used for some experiments, but results were not included in this thesis.

The spatial feature matching mode was explored as well. The automatic image pair generation based on spatial location seemed like a convenient option to find suitable image pairs. The extraction script mentioned in Section 3.1.1 was modified to generate one `.json` file containing timestamped wheel odometry data and one file containing timestamped GPS data for the images from all the desired rosbags involved. The odometry data did not prove very useful: the axes of the data were not aligned between rosbags, as can be seen in in figure A.1 in Appendix A.1.

The GPS data was more suitable for use in the spatial matching mode, but ultimately the custom image pair generation proved more useful in non-exhaustive image pair generation.

The custom matcher (named matches_importer in the COLMAP software) was used in several tests. The main reason for this choice is that the custom matching feature allows the user to pass an image pair list. To generate an image pair list, a python script was written. This will be elaborated upon in Section 3.4.2.

All results included in this thesis were generated using the exhaustive or the custom feature matching modes.

3.2.3 Reconstruction

In the experiments for this thesis, both the `mapper` and the `hierarchical_mapper` mentioned in 2.7.5 were used. The hierarchical mapper parallelizes the reconstruction process, by first generating overlapping submodels, then merging them into one [37]. The creators of COLMAP recommend running the functions `point_triangulator` and `bundle_adjuster` a few times after the hierarchical mapper [37]. This was done as well: four times for both.

After reconstruction, the mapper sometimes yields multiple models. These can sometimes be merged using the COLMAP function `model_merger`. This functionality, however, is best suited to merge models generated in different reconstruction attempts. When multiple models are produced in a single reconstruction, however, one can assume that the merging failed, resulting in multiple proposals, with only one resembling the actual situation. The `model_merger` often cannot add any value in this situation.

3.3 hloc toolbox

The `hloc` toolbox is structured into three python notebooks (described in Section 2.8). It comes with a folder of python3 scripts that are used in these notebooks, named, for example, match_features.py or reconstruction.py. These scripts contain the working code of the pipeline, and they were used in these experiments: they were called from python scripts written for this thesis.

3.3.1 Feature extraction and matching

The `hloc` scripts folder containing the SuperPoint+SuperGlue scripts were used for feature extraction and matching. For image pairs, the scripts have an `exhaustive` option, but custom image pairs can also be passed in a file. For this, the same script used for COLMAP (further explained in Section 3.4.2), was used. In this thesis, both the exhaustive and custom image pairs options were used to generate reconstructions with the `hloc` pipeline.

3.3.2 Reconstruction

The `hloc` pipeline then uses the COLMAP `mapper` or `hiermapper` to generate a 3D sparse model.

3.4 Feature matching

3.4.1 Cameras

The IPLT dataset contains images from two cameras: the front and the rear camera. A few reconstructions were first made using only images from the front camera and using only images from the rear camera. This worked with varying degrees of success, as can be seen in the Results chapter. Using images from only one camera has one flaw: objects in the surroundings of the vehicle taking the pictures are only viewed from one side, meaning that only features from that side of the object will be extracted and used in the matching. When another image is entered into the model (say, in the hypothetical scenario that a robot drives around in the IPLT dataset parking lot and is trying to localize), it would be difficult to find matching keypoints if the camera is oriented in the opposite direction from the camera used in the IPLT dataset. For this reason, the front and rear camera images were also paired to be matched. It (naturally) proved much more difficult for both COLMAP and SuperPoint+SuperGlue to find and match features in a pair of one front camera image and one rear camera image. The performance of feature matching of these image pairs was tested for both pipelines to find an optimal configuration for creating a sparse 3D model using both front camera and rear camera images. One parameter in this configuration is the image pairs used in the matching, discussed in the next paragraph.

3.4.2 Image pairs

The script mentioned in Section 3.2.2 can generate three types of image pairs: front-camera front-camera (front-front) pairs, front-camera rear-camera (front-rear) pairs and rear-camera rear-camera (rear-rear) pairs. The script takes as input a time threshold for each of these types of pairs: an image a only gets matched with an image b if their timestamps are within a certain range. Since the images in the IPLT dataset are named after their Unix timestamp with nanosecond resolution, the script that generates the image pairs takes the aforementioned thresholds in nanoseconds. For example, if for the front-front image pairs a threshold of 5×10^9 nanoseconds was chosen, a front camera image would be paired with all front camera images that were taken five seconds before and five seconds after it was taken.

For the front-rear image pairs, a different approach was chosen. There is a time delay between the front camera taking a picture and the same features coming into view for the rear camera. This means that front camera images get paired only with rear camera images taken, say, 5 to 15 seconds later, instead of the aforementioned 5 seconds earlier to 5 seconds later window used for front-front pairs. In front-rear pairs, the 5 second later would be the lower threshold for the window, and the 15 seconds the upper threshold.

3.4.3 Front-rear matching test process

To test the feature matching of COLMAP and of SuperGlue, one rosbag from the IPLT dataset was used: 2018-10-22-18-55-41. This one has the tag "cloudy", meaning there is less solar glare for the algorithms to be confused by (COLMAP's creators recommend using images without high dynamic ranges and shiny surfaces [38]).

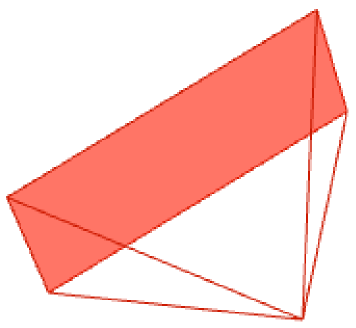
The matching windows (image pair generation thresholds) used were varied, so as to find an optimal point with the highest mean of matched features per image. Early experiments showed that both pipelines had very little trouble matching a large number of features in front-front and rear-rear images pairs; a larger challenge lay in matching features in front-rear image pairs. Thus, tests were run varying the lower window threshold and the upper window threshold for the front-rear pairs. With the COLMAP pipeline, this process was repeated for multiple camera models: `OPENCV`, `THIN_PRISM_FISHEYE`, `UCM`, `FOV`, `RADIAL` and `RADIAL_FISHEYE`. Also, the matching parameter `guided_matching` was tested on both true and false, to explore the effects of these changes.

Mirroring images

The above-mentioned tests were performed twice for both pipelines: once with the rear camera images mirrored horizontally, once without any mirroring. The mirroring of the images of one of the folders can be beneficial to the feature matching process: features extracted in a front camera image would appear flipped in a rear camera image. Mirroring the images from either the front or rear folder improves keypoint matching performance. However, the parking lot, the environment of the vehicle, seen in the mirrored rear camera images looks different from the one in the front camera images: a mirrored environment appears. This poses a problem for the reconstruction process, since the mapper would be trying to reconstruct two mirrored environments into one. Thus, no reconstruction was made involving any mirrored images.

3.5 Reconstruction

The reconstruction process happens exclusively with functions from the COLMAP algorithm. The COLMAP software has two functions, `mapper` and `hiermapper`, which produce a 3D reconstruction based on matched features (which are provided by either SIFT, called COLMAP pipeline in this thesis, or by SuperPoint+SuperGlue, called hloc in this thesis). This 3D reconstruction can be viewed in the COLMAP GUI, and is represented by a point cloud of black dots. Each image used for to create the reconstruction is represented in the reconstruction as well, by a red figure (see Fig. 3.3). The reconstructed location and orientation of the camera can thus be seen for each image.


(a) camera representation

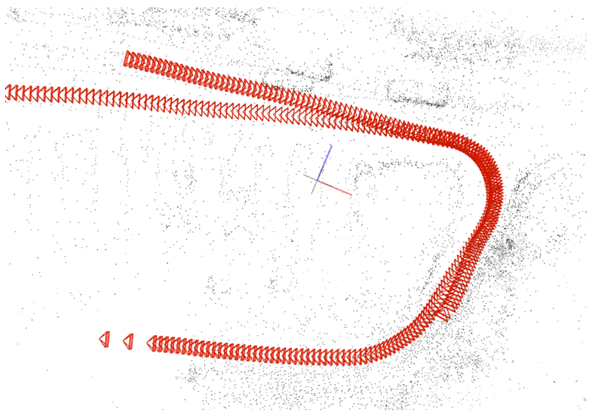

(b) reconstruction example

Figure 3.3: COLMAP GUI reconstruction

In figure 3.3b an example of a COLMAP reconstruction can be seen. The camera representation shown in figure 3.3a can be seen many times, forming something of a track. If the reconstruction goes well, this track thus takes the shape of the path driven by a vehicle taking pictures (in the case of a dataset like IPLT, where a driving vehicle is taking the pictures used in the dataset).

In the next chapter, the results of all the methods

Chapter 4

Results

The results of the methods presented above are laid out in this chapter. In the first section, results (of feature matching and reconstruction), obtained using only front-front and rear-rear image pairs, are presented. In the second section, results obtained using front-rear image pairs are presented.

4.1 Front-front/rear-rear image pairs

The COLMAP and `hloc` pipeline were first tested on images from one of the cameras in the dataset only, as described in Section 3.4.1, so only front-camera front-camera image pairs and rear-camera rear-camera image pairs. Results of both pipelines will now be presented separately. For both, statistics about the feature extraction and matching will be presented first, followed by reconstruction results.

4.1.1 COLMAP

Feature matching

COLMAPs SIFT feature matching is tested here. The images used here are from rosbag 2018-10-22-18-55-41, which has the tag "cloudy".

The time window matching front_cam images to rear_cam images is of size 5×10^8, which is equivalent to 0.5 seconds. The amount of feature matches between two images decreases with the time between taking both images. This short time window of 0.5 seconds pairs an image with only a few other images taken in its temporal vicinity. Since images taken right after each other will be similar, these image pairs provide a good view of the maximum amount of feature matches generated by either pipeline.

The mean amount of keypoints extracted from images from this rosbag is 965.42; the highest is 2462, the lowest 534. The mean amount of feature matches per image are shown below:

front-front pairs mean feature matches	rear-rear pairs mean feature matches
555.150	499.9312

Reconstruction

As shown in Section 4.1.1, the COLMAP pipeline did not struggle to find feature matches in pairs of images that were made by the same camera. This allowed for a good number of matches that the COLMAP `mapper` can then use for creating the sparse models. A couple of reconstructions will now be shown.

One experiment involved the images from the front camera folder from one rosbag. This experiment was done with the camera model set on OPENCV and a custom image pair generation window of 5 seconds. The result was as follows:

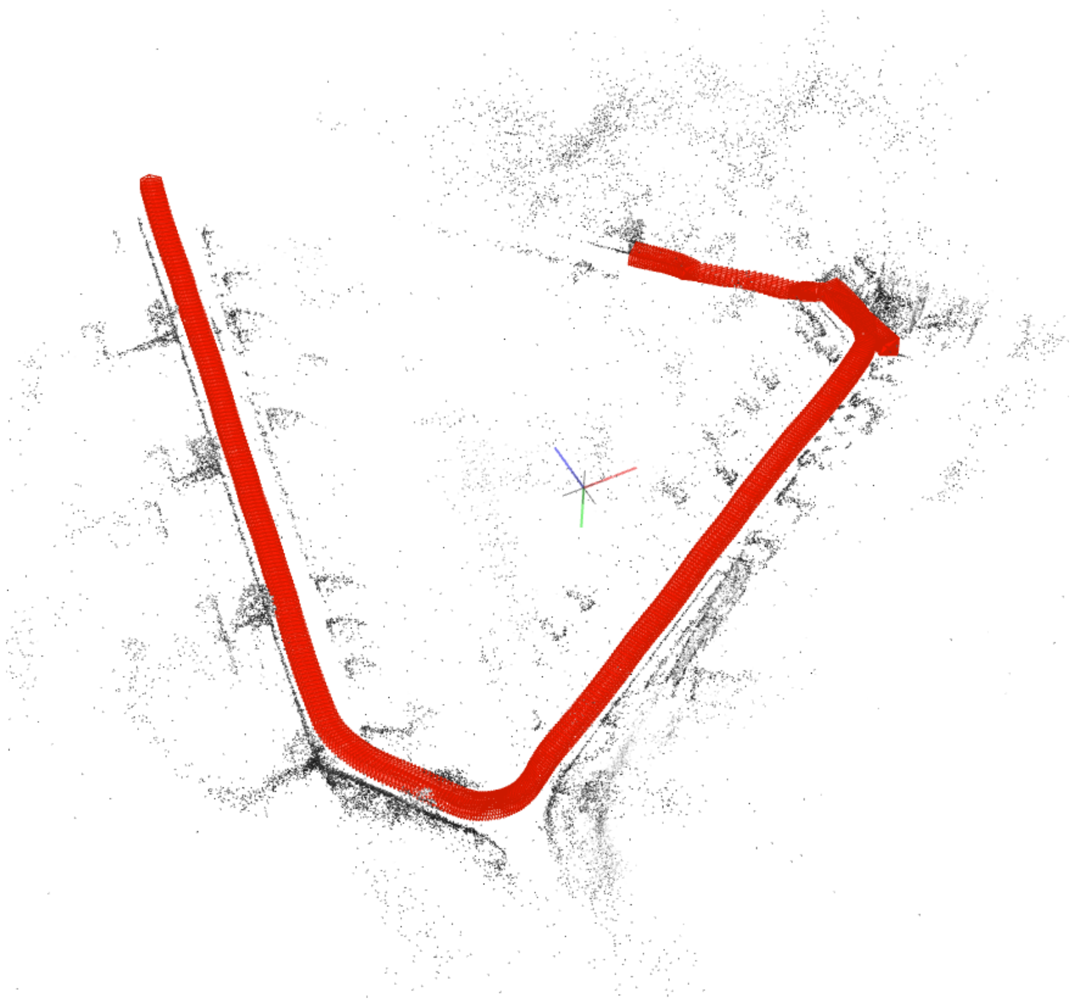


Figure 4.1: COLMAP pipeline: custom match mode

The COLMAP pipeline managed to bring all the images together into a track. The shape of this track, however, is incorrect (see figure 2.4 for reference). The top right part of the image, where the track seems to be overlapping, corresponds to the right of figure 2.4, the part where the vehicle start and ends its loop. On the left side of the image, one can see trees: three blobs of points in a row next to the track. The trees (five) can be seen at the top of figure 2.4. All corners in the reconstruction are too "shallow", not sharp enough, resulting in a cut in the track and an open loop.

In the dataset images, trees appear like in the iamge below:

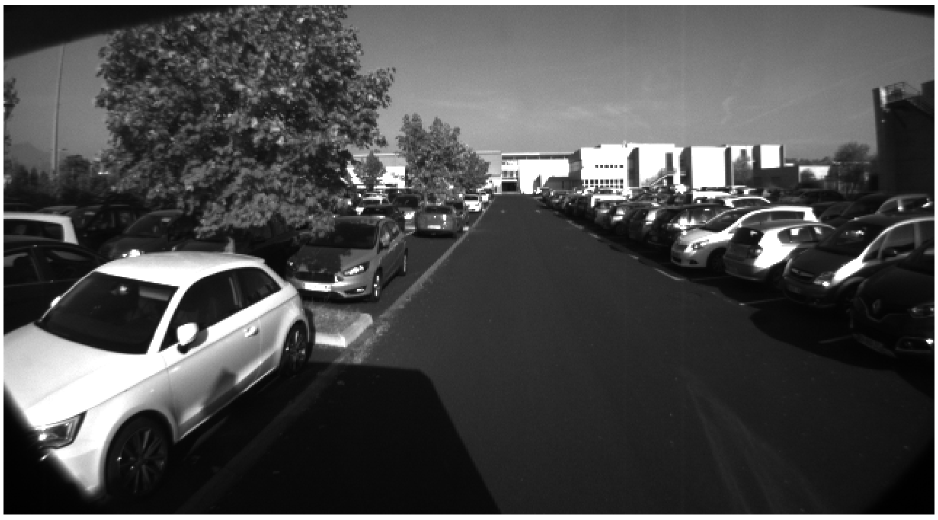


Figure 4.2: Trees next to the road [12]

Another experiment involves the images from the front camera folders from two of rosbags. The result can be seen in the figure below:

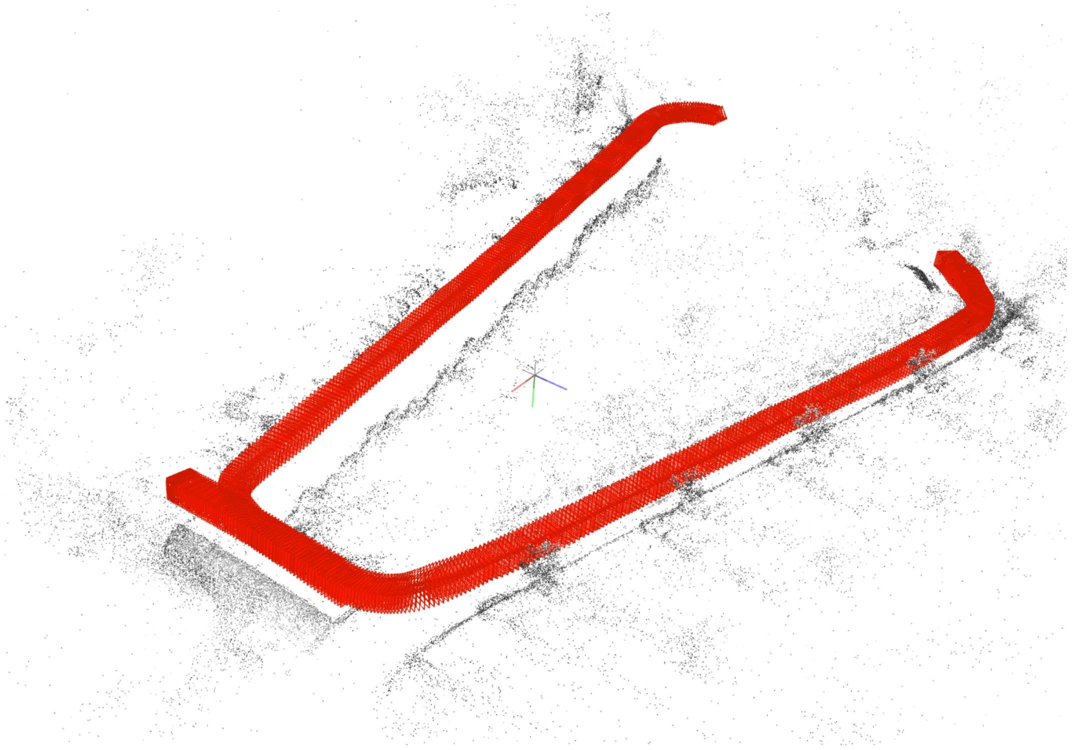


Figure 4.3: COLMAP pipeline: two front_cam folders

This experiment utilized the exhaustive feature matching mode. Although the corners in the reconstruction are still not sharp enough, the shape of the track resembles the true parking lot more closely and the cut in the loop is smaller. The trees seen in figure 4.2 are clearly visible on the bottom right side of the reconstruction.

4.1.2 hloc

Feature matching

`hloc` 's SuperPoint+SuperGlue feature matching is tested here. The configuration for this test was the same as for COLMAP, described in Section 4.1.1: images from rosbag 2018-10-22-18-55-41, a time window of 0.5 seconds.
The mean amount of keypoints found in images from this rosbag is 571.70. The highest amount of keypoints found is 925, the lowest 359.

front-front pairs mean feature matches	rear-rear pairs mean feature matches
419.733	419.555

Reconstruction

The following reconstruction is one performed with the `hloc` pipeline (SuperPoint feature extraction, SuperGlue feature matching, COLMAP `mapper` (see Section 2.8.1). The images were paired and matched exhaustively. It was performed on the images from the front cameras from two rosbags, 2018-10-22-18-24-58.bag and 2018-10-22-18-29-07.bag, both with the tag "cloudy".

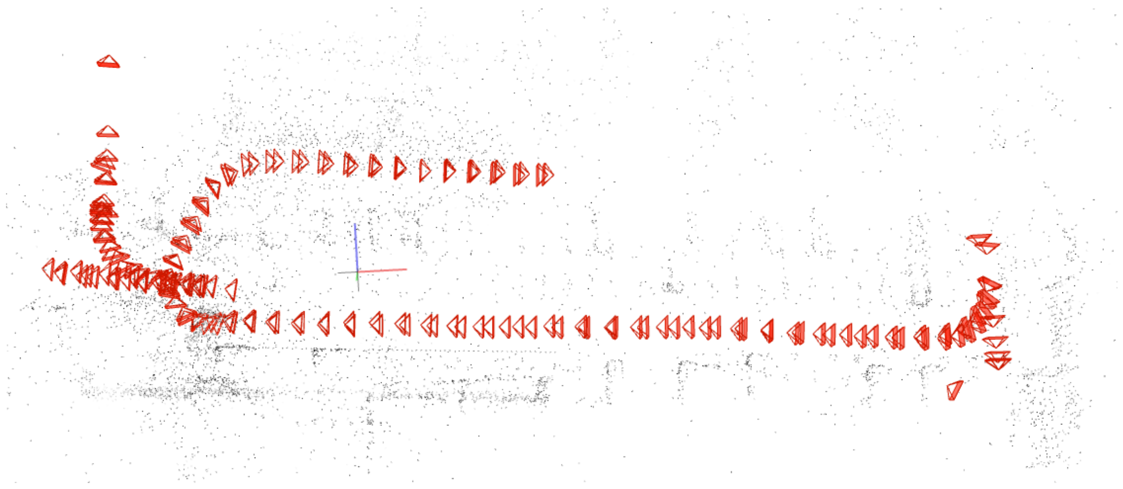


Figure 4.4: `hloc` on two rosbags

A problem arose with trying to create reconstruction from these images (and more broadly in other experiments performed for this thesis): the mapper exits "too early", which results in a reconstruction containing only a handful of the input images and only a couple hundred points in the pointcloud. The COLMAP `mapper` has some parameters that allow for making adjustments in, for example, the amount of iterations in the bundle adjustment process, or the parameter `init_num_trials`, the amount of trials that the `mapper` initializes with. Even with parameters like these set to high values, the mapper would often exit without producing a reconstruction containing more than a handful of the input images.
One of the attempted solutions for this problem was to use a subset of the images for feature matching and reconstruction. In the above reconstruction, five out of every six images were skipped, resulting in the total amount of input images for this reconstruction being 183, out of the 1097 present in the front camera folders of the two rosbags combined. This allowed the `mapper` to create a reconstruction

that uses all input images, but it does mean that fewer points are present in the pointcloud, and the generated path is less precise.

4.1.3 Comparing COLMAP and hloc

SIFT (in COLMAP pipeline), on average, extracted many more features, and matched more features in the front-front and rear-rear pairs than SuperPoint+SuperGlue in the `hloc` pipeline. It is hard to say whether the increased amount of extracted features are of the same quality/usefulness for the reconstruction as SuperPoint's extracted features.
The feature matches shown in the COLMAP examples above proved sufficient for the COLMAP reconstruction functions to create a track of images. The same is not quite true for the hloc pipeline: in figure 4.4, the track looks straight, although one part of it is in the wrong place.
Producing a reconstruction from images from the same camera, the COLMAP pipeline's biggest problem was not being able to get the angles of the turns taken by the vehicle right, which also resulted in an incomplete loop. The hloc pipeline seems not to produce this problem.
With both pipelines the COLMAP (hierarchical) mapper often exited, like mentioned in Section 4.1.2. Sometimes running the (hierarchical) mapper on the same database again helps, and does produce a reconstruction.

4.2 Front-rear image pairs

In order to a model that encompasses different perspectives of the environment (as explained in Section 3.4.1), tests were run that involved both images from the front and the rear camera. The results will be discussed below: first for feature matching, then reconstruction results.

4.2.1 Feature matching

For feature matching, tests were run as described in sections 3.4.2 and 3.4.3, with varying matching windows. In the feature matching tests of this subsection 4.2.1, images are paired only with images from *the other* camera; front-rear image pairs only, no front-front or rear-rear. For COLMAP, its SIFT feature matching tested. For `hloc` , SuperPoint+SuperGlue is tested.

Results for both the COLMAP and hloc pipelines are presented here (their feature extraction and matching components, meaning SIFT for COLMAP and SuperPoint+SuperGlue for `hloc`) . For each pipeline, a table and two graphs are

shown. The table shows the mean amount of matched features in the images for different combinations of the lower_threshold and upper_threshold. If an image *a* has 20 feature matches with image *b* and 30 feature matches with image *c*, image *a* will account for 50 points being added to the total score. This total score will then be divided by the amount of images to acquire the mean amount of matched features in the images (the 20 points for image *b* and 30 points for image *c* are not added to the score, since each match would be doubled then).
The graphs shows two lines: the amount of extracted and the amount of matched features per image. There is a graph for the front camera images and one for the rear camera images. The images are sequentially ordered along the x-axis, with the time since the first image was taken shown as the x-axis ticks. This means each datapoint is the data of one image.
Say, an image *a* has 500 features extracted, its point on the blue line will be at 500. And say, this image *a* has 20 feature matches with an image *b* and 30 with an image *c*, its point in the red line will read a value of 50.
The graphs were generated using the database produced by the widest matching window: 6 to 23 seconds. This means that features in a front camera image that are matched with features from a rear camera images taken 6 to 23 seconds later are counted towards the score.
The feature matching experiments were also performed with mirrored rear camera images (elaborated upon in Section 3.4.3). The tables and graphs for these experiments can be found in Appendix D. After each subsection, some considerations and sub-conclusions discussing the graphs and tables will be presented.

COLMAP

With the COLMAP pipeline, the described experiments were performed for multiple camera models as well as for both settings (true/false) of the guided_matching parameter, as described in Section 3.4.3. However: these all yielded the same results: these settings are only relevant in the reconstruction process. The images used here are from rosbag 2018-10-22-18-55-41, with the weather tag "cloudy".

The mean amount of keypoints extracted from images from this rosbag is 965.42, the highest is 2462, the lowest 534.

time (s)→ time (s)↓	12	13	14	15	16	17	18	19	20	21	22	23
6	1.655	1.779	1.845	1.880	1.894	1.887	1.869	1.844	1.813	1.772	1.735	1.699
7	1.861	1.9373	2.022	2.041	2.038	2.017	1.9386	1.9349	1.9308	1.858	1.813	1.770
8	2.079	2.170	2.196	2.193	2.171	2.134	2.088	2.039	1.9387	1.9327	1.874	1.824
9	2.238	2.314	2.316	2.293	2.255	2.202	2.143	2.084	2.023	1.9355	1.895	1.840
10	2.353	2.417	2.394	2.351	2.297	2.230	2.161	2.093	2.024	1.9350	1.885	1.826
11	2.485	**2.515**	2.452	2.384	2.312	2.232	2.151	2.076	2.001	1.9321	1.853	1.791

The highest mean amount of features matched in an image: 2.515, with lower_threshold = 11s and upper_threshold = 13s.

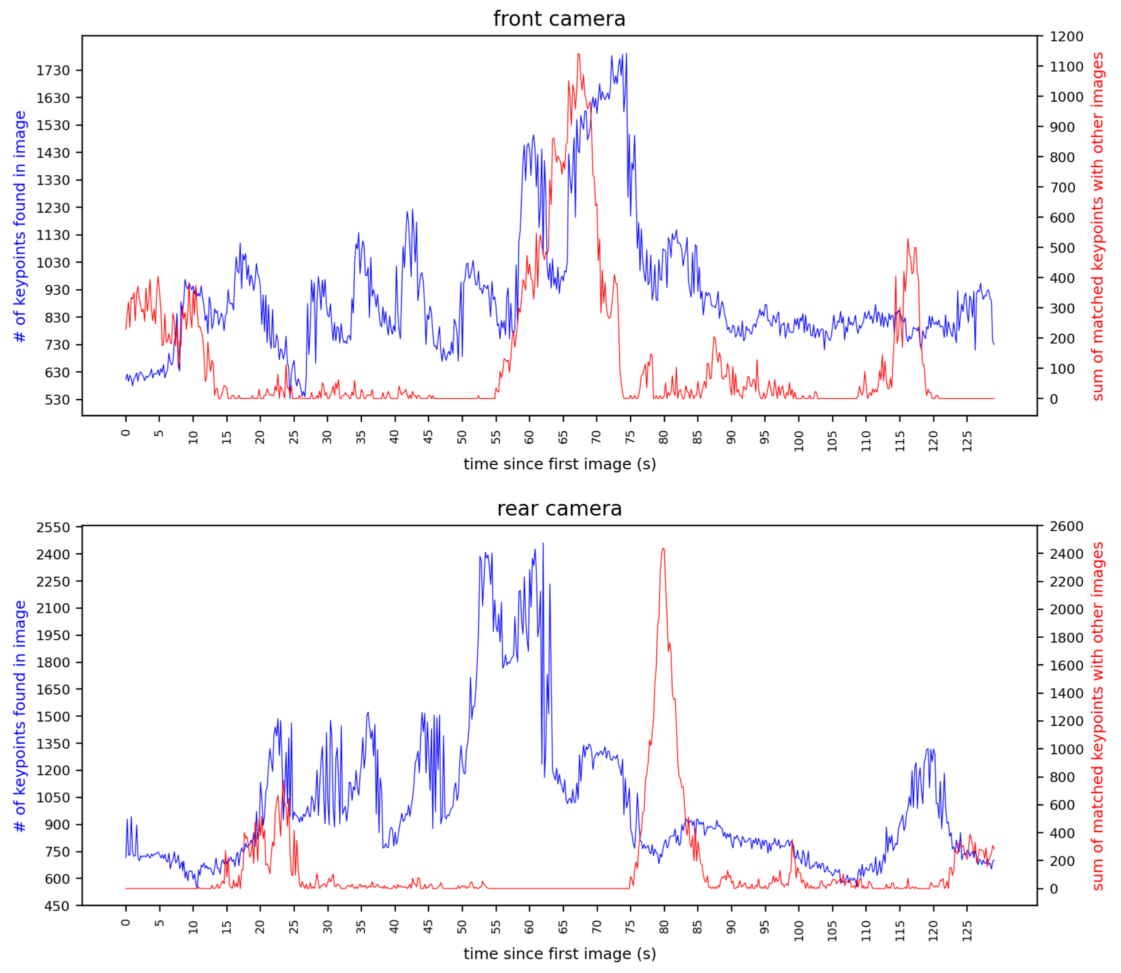
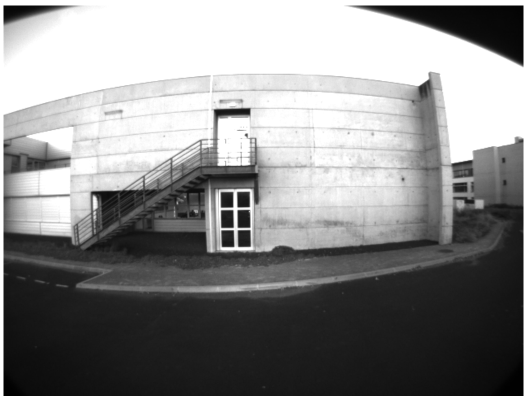


Figure 4.5: COLMAP, no mirrored images

A clear spike can be seen in the red line in the graphs, especially the rear camera one. If t0 is the time at which the first image was taken, the front camera spike peaks at t+68 seconds while the rear camera spike peaks at t+80 seconds. The front image at t+68 s and the rear image at t+80 s are shown in figure 4.6 below. The structure appearing in these images apparently provided the COLMAP pipeline with keypoints that were easy to match. One remarkable aspect about this is that there is no real spike in the blue line coinciding with the spike in the red, especially in the rear camera graph, where the blue line dips a little at t+80 s. The amount of features extracted from an images does not seem to have a large influence on the amount of matched features in an image in this test case (also clearly visible in other parts of the graph), suggesting that COLMAP extracts many features that it is subsequently unable to match (in this test case).


(a) front camera image at t+68s

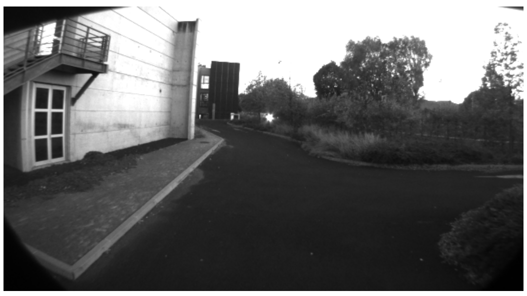

(b) rear camera image at t+80s

Figure 4.6: images from 2018-10-22-18-55-41.bag

COLMAP considerations
The graph clearly shows that some images had many features matched, while other had close to zero features matched (red line). This does not seem to be proportional to the amount of features extracted from an image (blue line): while that number varied much as well, the peaks do not coincide with those of the amount of matched features. It should be noted that the COLMAP feature matching parameter `min_num_inliers` was set to 10 in these tests, down from the COLMAP default of 15. This means that any image match with fewer than 10 features matched does not end up in the database, and is subsequently not shown in the graphs. Setting this variable below a value of 10 yielded unstable results. This threshold of 10 matches is one of the causes of the flat parts in the graphs: some images never get paired with an image they have 10 or more feature matches with, so the sum of their feature matches will then be 0.
Upon inspection of the graphs of the test with mirrored rear images in Appendix D.1, it is apparent that the amount of feature matches in front-back image pairs are considerably larger in number when the rear camera images are mirrored. This

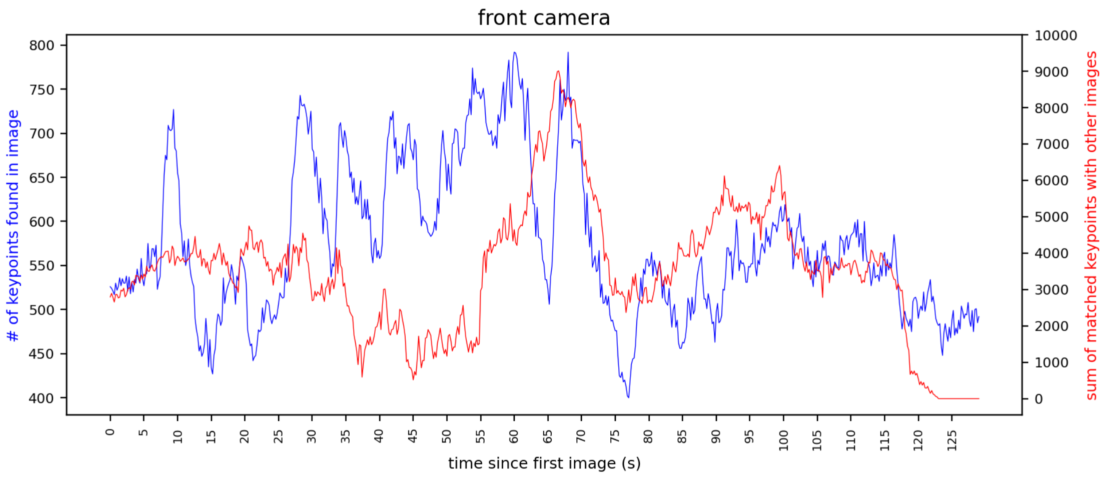
means COLMAPs SIFT does have trouble finding the keypoints in these images, when taken from the opposite perspective.

SuperPoint+SuperGlue

The images used here are from rosbag 2018-10-22-18-55-41, with the weather tag "cloudy", same as with COLMAP.
The mean amount of keypoints found in images from this rosbag is 571.70. The highest amount of keypoints found is 925, the lowest 359.

time (s)→ time (s)↓	12	13	14	15	16	17	18	19	20	21	22	23
6	43.078	44.045	44.878	45.475	45.9321	46.184	46.345	46.462	46.521	46.494	46.468	46.444
7	44.893	45.731	46.453	46.9336	47.277	47.438	47.505	47.540	47.526	47.428	47.342	47.265
8	46.566	47.247	47.845	48.206	48.437	48.491	48.465	48.419	48.334	48.170	48.026	47.9302
9	48.000	48.502	48.9375	49.214	49.338	49.291	49.175	49.057	48.9308	48.684	48.491	48.327
10	48.9374	49.325	49.715	49.858	49.898	49.765	49.578	49.402	49.205	48.9334	48.704	48.511
11	49.674	49.855	50.202	**50.261**	50.230	50.021	49.770	49.548	49.313	49.002	48.744	48.531

The highest mean amount of features matched in an image: 50.261, with lower_threshold = 11s and upper_threshold = 15s.



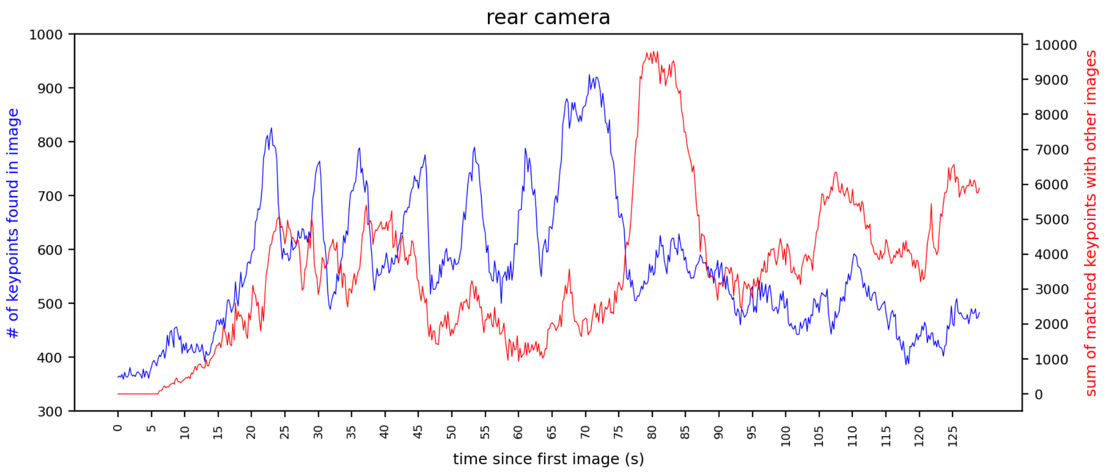


Figure 4.7: SuperPoint+SuperGlue, no mirrored images

SuperPoint+SuperGlue considerations
The SuperPoint+SuperGlue pipeline shows interesting periodic peaks in the amount of extracted features, which is probably either the trees coming into view or the trees leaving the frame.

Keypoint matching considerations

The tables and graphs show SIFT, used in COLMAP, extracting considerably more features from the images than the SuperPoint+SuperGlue pipeline. It does not manage to match them very well though, in these front-back image pairs. SuperGlue performs considerably better on the feature matching. Perhaps SuperGlue's ability to "reason about the 3D scene" [7] enables it to avoid confusion caused by the different viewpoints when matching front-back camera image pairs.

With both pipelines, the highest mean amount of matched features (the "optimal setting" mentioned in 4.2.1) is produced by a short matching window: pairing rear camera images with front camera images taken 11 to 12 or 15 seconds later. This short time window yields the highest mean matched amount of features per image, but is most likely not as suitable for the reconstruction process, since only a small "slice" of matched features gets created at each physical point in the reconstruction.

4.2.2 Reconstruction

COLMAP

One experiment involved images from the front and rear camera folders from a rosbag, and the COLMAP-default

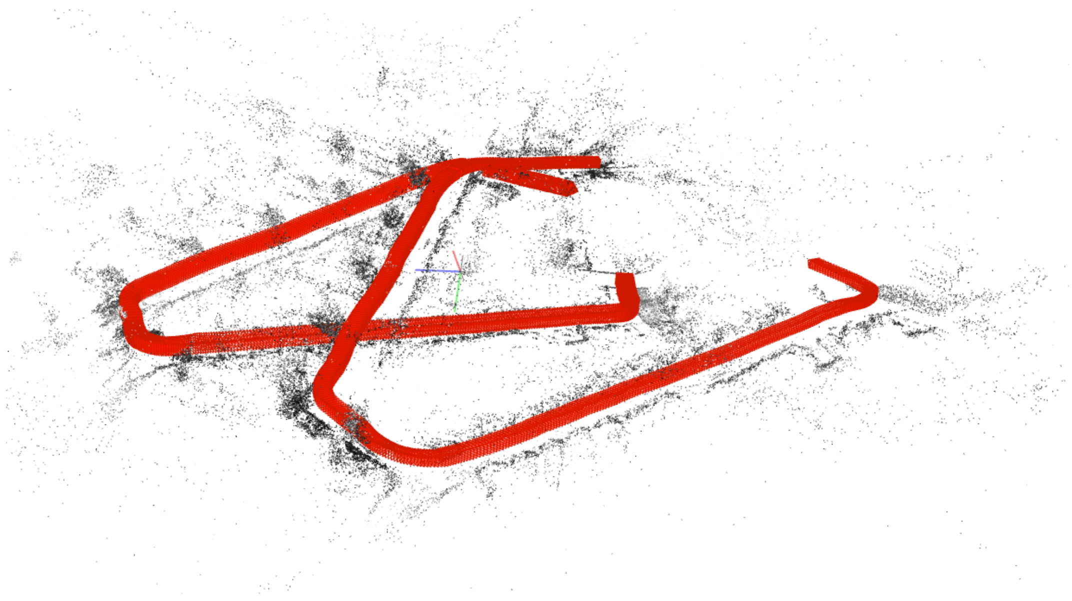


Figure 4.8: COLMAP pipeline

This figure shows the COLMAP pipeline reconstructing two different "tracks": one of front camera images and one of rear camera images. It does not manage to bring them together into one reconstruction. One cause of this could be that COLMAP did not generate enough feature matches in the front-back image pairs (like shown in figure 4.5). This did however, result in the shape of both "tracks" being similar, albeit not very accurate: the corners taken by the vehicle do not look as sharp as the corners seen in figure 2.4, causing both track to not be completed into a closed shape.

hloc

The following results were obtained using the hloc pipeline (SuperPoint feature extraction, SuperGlue feature matching, COLMAP `mapper` (see Section 2.8.1). A custom matching window with a lower threshold of 7 seconds and an upper threshold of 21 seconds was used; the camera model used was OPENCV.

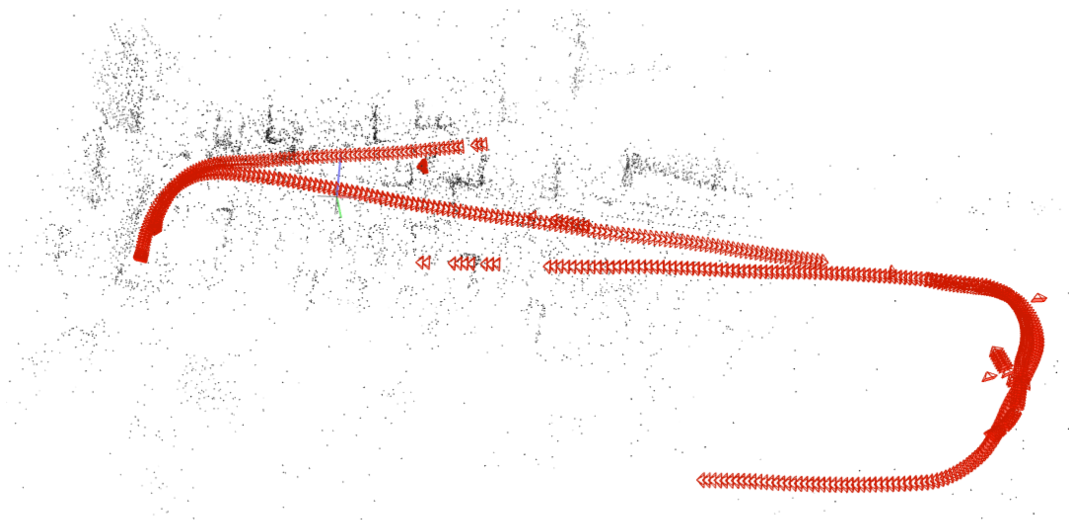


Figure 4.9: hloc pipeline

In this reconstruction, the hloc pipeline managed to bring together some pictures of the font and rear cameras. This reconstruction also shows a more accurate shape: the corners look like they have the correct angle. However this reconstruction still has a significant part missing, seen in the top right of the image.
Where one of the "tracks" seems to go in the right direction, the other seems to have veered off, causing it to seperate and fall apart into two directions. This causes the cameras in the reconstruction to have different viewpoints, with a messy pointcloud.

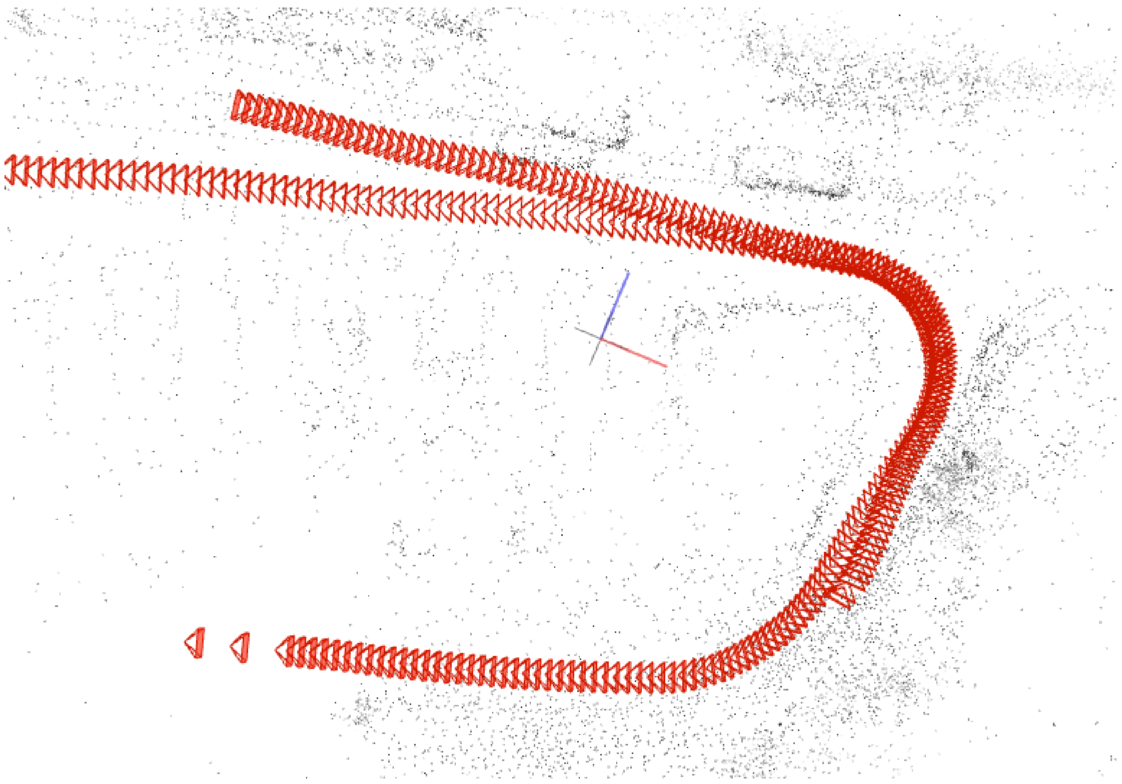


Figure 4.10: hloc pipeline

In this reconstruction, one can also see that the corners are at the right angles (this corresponds to the left side of image 2.4), and there is an overlap between the front and rear camera images. It is, however, incomplete.

Reconstruction considerations

In none of the performed experiments, the COLMAP pipeline was able to integrate the front and rear camera images into one reconstruction. The hloc pipeline had some more success: in figure 4.10, tracks in opposite directions are merged. The feature matches provided by SuperPoint+SuperGlue prove more valuable for the reconstruction than those provided by SIFT.
The hloc pipeline also did not have much trouble with the angle of the corners in the reconstruction.

4.3 Camera models

Different camera models were used in the different reconstruction attempts: the COLMAP *simple_radial* model, for example, the *opencv_fisheye* model, the *thin_prism_fisheye* model and the custom Unified Camera Model, which when used also allowed for insertion of the camera parameters beforehand. The difference in results was not very clear, multiple reconstructions attempts had difficulties in the "sharpness" of the corners.

Chapter 5

Conclusion

The feature extraction and matching experiments in this thesis showed SIFT extracting and matching more features than SuperPoint+SuperGlue in pairs of images taken from a similar perspective. Both the COLMAP and the hloc pipelines had little trouble reconstructing a vehicle track. However, in images from opposite perspectives (front-rear pairs), SIFT often did not find any feature match in a given image pair, whereas SuperPoint+SuperGlue managed to find enough to, sometimes, bring two image tracks (front and rear camera) together.

One of the main difficulties in generating reconstructions from feature matches generated by the COLMAP pipeline was the angles of the turns taken by the vehicle not being sharp enough in the reconstruction. At first glance, the problem seems to lie with the camera model involved: the WorldToImage and ImageToWorld (un)projection functions play a large role in generating the 3D structure from the features matches. However, the hloc pipeline does not seem to experience this problem. Since both pipelines use the same reconstruction `mapper` or `hiermapper` , the difference seems to stem from the feature matching step in the pipelines. SuperPoint+SuperGlue provides the `mapper` with features/feature matches that make for a better reconstruction.
In addition, the use of the customly implemented UCM and the a-priori camera parameters provided in the IPLT dataset had little effect. COLMAPs ability to calculate and refine camera parameters seems to be performing well.
Future research could be isolate the variable of the fish-eye lens, and explore the effect of the different camera models more in-depth.

Another problem experienced during this research is the exiting of the `mapper` , with a reconstruction of only a handful of the input images. The workaround, only using a subset of the images, causes the reconstruction to be less detailed and dense. Ideally, more experimentation with the `mapper` yields a configuration in which the `mapper` does not exit prematurely, opening the door to use significantly more images for a single reconstruction.

The approach taken to image pair generation in this thesis was using a temporal threshold (see Section 3.4.2). A downside of this is that the image pairs are not always arranged according to what is in the view of a camera at a given moment, seeing as the vehicle speed is not constant. A better approach would be a spatial threshold. The COLMAP spatial matching mode, used with the IPLT GPS data, is not perfect for front-rear image pairs, since images from the front camera taken behind the location where a rear camera image was taken will not be a good match. A custom spatial threshold for front-front/rear-rear pairs and for front-rear pairs could solve this.

The different weather conditions present in the IPLT dataset and their effects on the tested pipelines have not been thoroughly studied in this thesis. To further discover the advantages of using SuperPoint+SuperGlue or other neural network approaches in a reconstruction pipeline with a challenging dataset, this could be interesting further research.

Appendices

Appendix A

IPLT dataset

A.1 Wheel odometry

The wheel odometry data for some of rosbags with tag "sun" is plotted here.

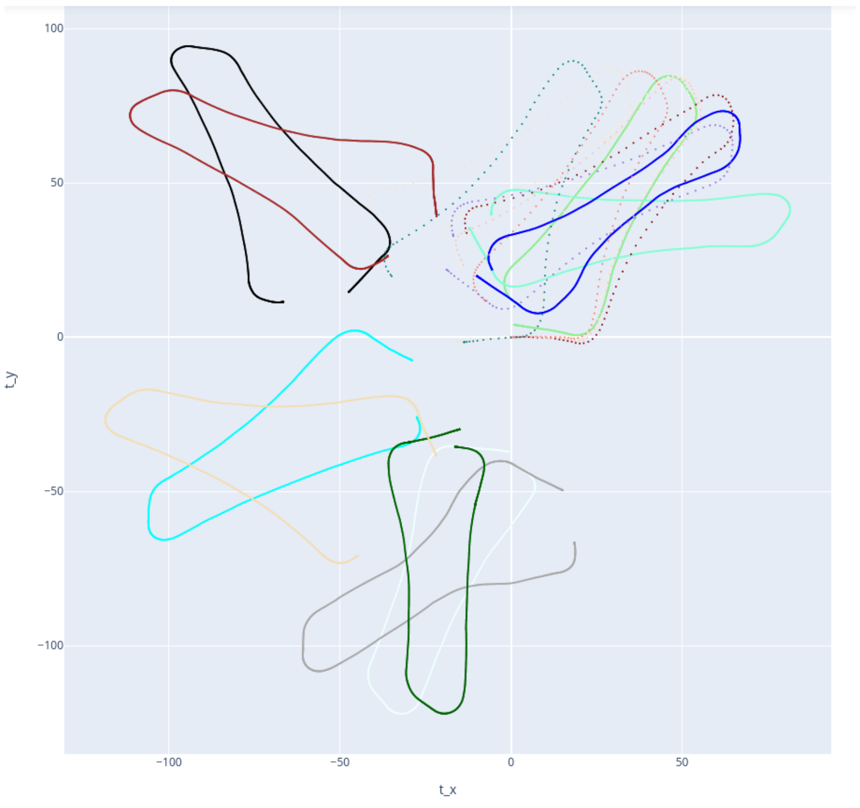


Figure A.1: Wheel odometry data for "sun" rosbags

A.2 Images

(a) sun

(b) cloudy

(c) dusk

(d) night

(e) fog

(f) rain

A.3 Groups of different conditions tags

tag	rosbag names
sun	2018-10-19-10-54-31.bag, 2018-10-19-10-59-14.bag, 2018-10-19-15-06-22.bag, 2018-10-19-15-08-49.bag, 2018-10-22-15-29-04.bag, 2018-10-22-15-31-42.bag, 2019-10-02-15-03-40.bag, 2019-10-02-15-05-44.bag, 2019-10-02-15-08-10.bag, 2020-01-15-11-13-20.bag, 2020-01-15-11-15-33.bag, 2020-01-15-13-20-56.bag, 2020-01-15-13-23-09.bag, 2020-01-15-13-25-22.bag, 2020-01-16-14-03-10.bag, 2020-01-16-14-05-24.bag, 2020-10-05-16-20-11.bag, 2020-10-05-16-22-29.bag, 2020-10-05-16-25-05.bag, 2020-10-05-16-27-52.bag
cloudy	2018-10-22-18-24-58.bag, 2018-10-22-18-29-07.bag, 2018-10-22-18-42-29.bag, 2018-10-22-18-55-41.bag, 2018-10-22-19-00-29.bag, 2018-10-26-08-10-02.bag, 2018-10-26-08-16-54.bag, 2018-10-26-08-23-22.bag, 2018-10-26-08-29-00.bag, 2018-10-26-08-35-24.bag, 2018-10-26-08-42-04.bag, 2018-10-26-08-48-22.bag, 2018-10-26-09-01-00.bag, 2018-10-26-09-05-09.bag, 2018-10-26-09-11-01.bag, 2018-12-11-16-53-49.bag, 2018-12-11-17-04-02.bag, 2018-12-11-17-13-18.bag, 2018-12-12-10-12-15.bag, 2018-12-12-10-20-11.bag, 2018-12-12-10-30-44.bag, 2019-01-09-14-32-39.bag, 2019-01-09-14-35-22.bag, 2019-03-08-14-57-34.bag, 2019-03-08-15-03-07.bag, 2020-01-20-10-15-30.bag, 2020-01-20-10-18-02.bag, 2020-01-22-10-17-33.bag, 2020-01-22-10-19-51.bag, 2020-01-22-10-22-06.bag, 2020-01-31-16-00-24.bag, 2020-01-31-16-02-46.bag, 2020-01-31-16-07-34.bag, 2020-02-05-17-53-21.bag, 2020-02-05-18-01-14.bag, 2020-02-05-18-04-18.bag, 2020-10-07-18-41-17.bag, 2020-10-07-18-45-00.bag, 2020-10-07-18-53-57.bag, 2020-10-07-19-09-15.bag, 2020-10-07-19-13-46.bag, 2020-10-07-19-21-02.bag, 2020-10-07-19-24-29.bag
dusk	2018-10-22-19-06-21.bag, 2018-10-22-19-09-43.bag, 2018-10-22-19-15-15.bag, 2018-10-22-19-20-21.bag, 2018-10-22-19-25-34.bag, 2018-10-26-07-44-01.bag, 2018-10-26-07-50-29.bag, 2018-10-26-07-56-31.bag, 2018-12-11-17-23-53.bag, 2018-12-11-17-33-30.bag, 2020-02-05-18-14-00.bag, 2020-02-05-18-19-19.bag, 2020-02-05-18-25-23.bag, 2020-10-07-19-30-56.bag, 2020-10-07-19-36-16.bag, 2020-10-07-19-40-03.bag
night	2018-10-22-19-30-22.bag, 2018-10-22-19-35-38.bag, 2018-10-22-19-40-27.bag, 2018-10-26-07-27-00.bag, 2018-10-26-07-31-09.bag, 2018-10-26-07-39-10.bag, 2020-02-05-18-29-17.bag, 2020-02-05-18-37-10.bag, 2020-02-05-18-41-39.bag, 2020-02-05-18-46-02.bag, 2020-10-07-19-44-33.bag, 2020-10-07-19-50-14.bag, 2020-10-07-19-54-47.bag
fog	2018-12-13-10-32-31.bag, 2018-12-13-10-36-57.bag, 2018-12-13-16-06-52.bag, 2018-12-13-16-09-32.bag
rain	2019-01-10-17-09-16.bag, 2019-01-10-17-21-20.bag, 2019-01-10-17-32-57.bag, 2019-10-01-16-54-55.bag, 2019-10-01-16-59-02.bag, 2019-10-01-17-01-27.bag, 2019-10-01-17-13-22.bag, 2019-10-01-17-15-45.bag, 2019-10-01-17-17-52.bag, 2019-10-01-17-19-57.bag, 2019-10-22-14-53-20.bag, 2019-10-22-14-57-27.bag, 2019-10-22-14-59-26.bag, 2019-10-22-15-01-25.bag, 2020-10-02-15-17-49.bag, 2020-10-02-15-20-28.bag
dusk rain	2019-01-10-17-41-09.bag, 2019-01-10-17-48-31.bag, 2019-01-10-17-57-03.bag
night rain	2019-01-10-18-05-41.bag
snow	2019-01-23-10-33-15.bag, 2019-01-23-10-36-01.bag, 2019-01-23-10-39-20.bag, 2019-01-23-16-03-09.bag, 2019-01-23-16-05-30.bag
sun rain	2019-02-04-10-58-40.bag, 2019-02-04-11-01-16.bag
rain other-path	2019-10-01-17-03-27.bag, 2019-10-01-17-08-17.bag, 2019-10-01-17-10-34.bag
day/dusk/night	2019-12-05-16-43-56.bag

Appendix B

COLMAP UCM

B.1 Projection functions

The functions WorldToImage, ImageToWorld, Distortion and Undistortion implemented for the Unified Camera Model

```
////////////////////////////////////////////////////////////////////////////
// Unified Camera Model

std::string UnifiedCameraModel::InitializeParamsInfo() {
  return "fx, fy, cx, cy, alpha";
}

std::vector<size_t> UnifiedCameraModel::InitializeFocalLengthIdxs() {
  return {0, 1};
}

std::vector<size_t> UnifiedCameraModel::InitializePrincipalPointIdxs() {
  return {2, 3};
}

std::vector<size_t> UnifiedCameraModel::InitializeExtraParamsIdxs() { return {4};
    }

std::vector<double> UnifiedCameraModel::InitializeParams(const double focal_length
    ,
                                                         const size_t width,
                                                         const size_t height) {
  return {focal_length, focal_length, width / 2.0, height / 2.0, 1e-2};
}

template <typename T>
void UnifiedCameraModel::WorldToImage(const T* params, const T u, const T v, T* x,
                                      T* y) {
  const T f1 = params[0];
  const T f2 = params[1];
  const T c1 = params[2];
  const T c2 = params[3];

```

```cpp
  // Distortion
  Distortion(&params[4], u, v, x, y);

  // Transform to image coordinates
  *x = f1 * *x + c1;
  *y = f2 * *y + c2;
}

template <typename T>


void UnifiedCameraModel::ImageToWorld(const T* params, const T x, const T y, T* u,
                                      T* v) {
  const T f1 = params[0];
  const T f2 = params[1];
  const T c1 = params[2];
  const T c2 = params[3];

  // Lift points to normalized plane
  const T uu = (x - c1) / f1;
  const T vv = (y - c2) / f2;

  // Undistortion
  Undistortion(&params[4], uu, vv, u, v);
}

template <typename T>
void UnifiedCameraModel::Distortion(const T* extra_params, const T u, const T v,
                                    T* du, T* dv) {

  const T alpha = extra_params[0];
  const T d = ceres::sqrt(u * u + v * v);

 T factor;
  factor = 1 / (alpha * d + (1 - alpha));

  *du = u * factor;
  *dv = v * factor;
}

template <typename T>
void UnifiedCameraModel::Undistortion(const T* extra_params, const T u, const T v,
                                      T* du, T* dv) {
 T alpha = extra_params[0];

  const T m1 = u * (1 - alpha);
  const T m2 = v * (1 - alpha);

  const T radius2 = m1 * m1 + m2 * m2;

  const T xi = alpha / (1 - alpha);
  const T xi2 = xi * xi;

 T factor;

  const T numerator = xi + ceres::sqrt(1 + (1 - xi2) * radius2);
  factor = numerator / (1 + radius2);

  *du = u * factor;
  *dv = v * factor;
}
```

B.2 Test case

The specialization of test case for unified camera model to camera_models_test.cc

```
BOOST_AUTO_TEST_CASE(TestUCM) {
  //
  // Using parameters from the Institut Pascal Long-Term Dataset front cam pre 2019-04-01,
  // in rewritten format [fx, fy, cx, cy, alpha]:
  // (see: https://arxiv.org/pdf/1807.08957.pdf)
  //
  // alpha = xi / (xi + 1) = 0.5920532
  // fx = gamma_x * (1 - alpha) = 766.3141 * (1 - 0.5920532) = 312.6154
  // fy = gamma_y * (1 - alpha) = 769.5469 * (1 - 0.5920532) = 313.9342
  std::vector<double> params = {312.6154, 313.9342, 324.2513, 239.7592, 0.5920532};
  TestModel<UnifiedCameraModel>(params);
}
```

Appendix C

UCM camera parameters

The conversion of the UCM camera parameters described in Section 2.5.3 is shown here.

```
from_2018-10-19_to_2019-03-08:
  camera_back
    [gamma_x, gamma_y, cx, cy, xi] = [763.5804, 766.0006, 326.2222, 250.7755, 1.4523]
    // alpha = xi / (xi + 1) = 0.59221954899
    // fx = gamma_x * (1 - alpha) = 763.5804 * (1 - 0.59221954899) = 311.373159894
    // fy = gamma_y * (1 - alpha) = 766.0006 * (1 - 0.59221954899) = 312.360070142
    [fx, fy, cx, cy, alpha] = [311.373159894, 312.360070142, 326.2222, 250.7755, 0.59221954899]

  camera_front
    [gamma_x, gamma_y, cx, cy, xi] = [766.3141, 769.5469, 324.2513, 239.7592, 1.4513]
    // alpha = xi / (xi + 1) = 0.5920532
    // fx = gamma_x * (1 - alpha) = 766.3141 * (1 - 0.5920532) = 312.6154
    // fy = gamma_y * (1 - alpha) = 769.5469 * (1 - 0.5920532) = 313.9342
    [fx, fy, cx, cy, alpha] = [312.6154, 313.9342, 324.2513, 239.7592, 0.5920532]


from_2019-10-01:
  camera_back
    [gamma_x, gamma_y, cx, cy, xi] = [764.4637, 763.1171, 322.6882, 247.8716, 1.4565]
    // alpha = xi / (xi + 1) = 0.59291675147
    // fx = gamma_x * (1 - alpha) = 764.4637 * (1 - 0.59291675147) = 311.200366379
    // fy = gamma_y * (1 - alpha) = 763.1171 * (1 - 0.59291675147) = 310.652188077
    [fx, fy, cx, cy, alpha] = [311.200366379, 310.652188077, 322.6882, 247.8716, 0.59291675147]

  camera_front
    [gamma_x, gamma_y, cx, cy, xi] = [770.0887, 768.9841, 330.3834, 222.0791, 1.4666]
    // alpha = xi / (xi + 1) = 0.59458363739
    // fx = gamma_x * (1 - alpha) = 770.0887 * (1 - 0.59458363739) = 312.206559641
    // fy = gamma_y * (1 - alpha) = 768.9841 * (1 - 0.59458363739) = 311.758736727
    [fx, fy, cx, cy, alpha] = [312.206559641, 311.758736727, 330.3834, 222.0791, 0.59458363739]
```

Appendix D

Feature matching: mirrored rear images

This appendix D contains the results of the feature matching tests where the rear camera images were mirrored, according to the process described in Section 3.4.3. The tables and graphs are structured in the same way as those in the results Section (4.2.1).

D.1 COLMAP

In this test, the rear camera images were mirrored (see Section 3.4.3).
The mean amount of keypoints extracted from images from this rosbag is 965.37, the highest 2462, the lowest 532.

time (s)→ time (s)↓	12	13	14	15	16	17	18	19	20	21	22	23
6	9.167	9.327	9.409	9.448	9.429	9.376	9.271	9.165	9.041	8.9308	8.755	8.590
7	9.529	9.658	9.706	9.714	9.664	9.583	9.450	9.321	9.175	9.022	8.852	8.670
8	9.836	9.9332	9.9343	9.9320	9.838	9.729	9.567	9.416	9.250	9.079	8.892	8.695
9	10.038	10.108	10.087	10.036	9.9326	9.792	9.605	9.434	9.251	9.065	8.864	8.653
10	10.278	10.293	10.221	10.133	9.9389	9.826	9.611	9.420	9.218	9.018	8.802	8.579
11	**10.436**	10.380	10.254	10.136	9.9361	9.775	9.535	9.328	9.113	8.9301	8.675	8.442

The highest mean amount of features matched in an image: 10.436, with lower_threshold = 11s and upper_threshold = 12s.

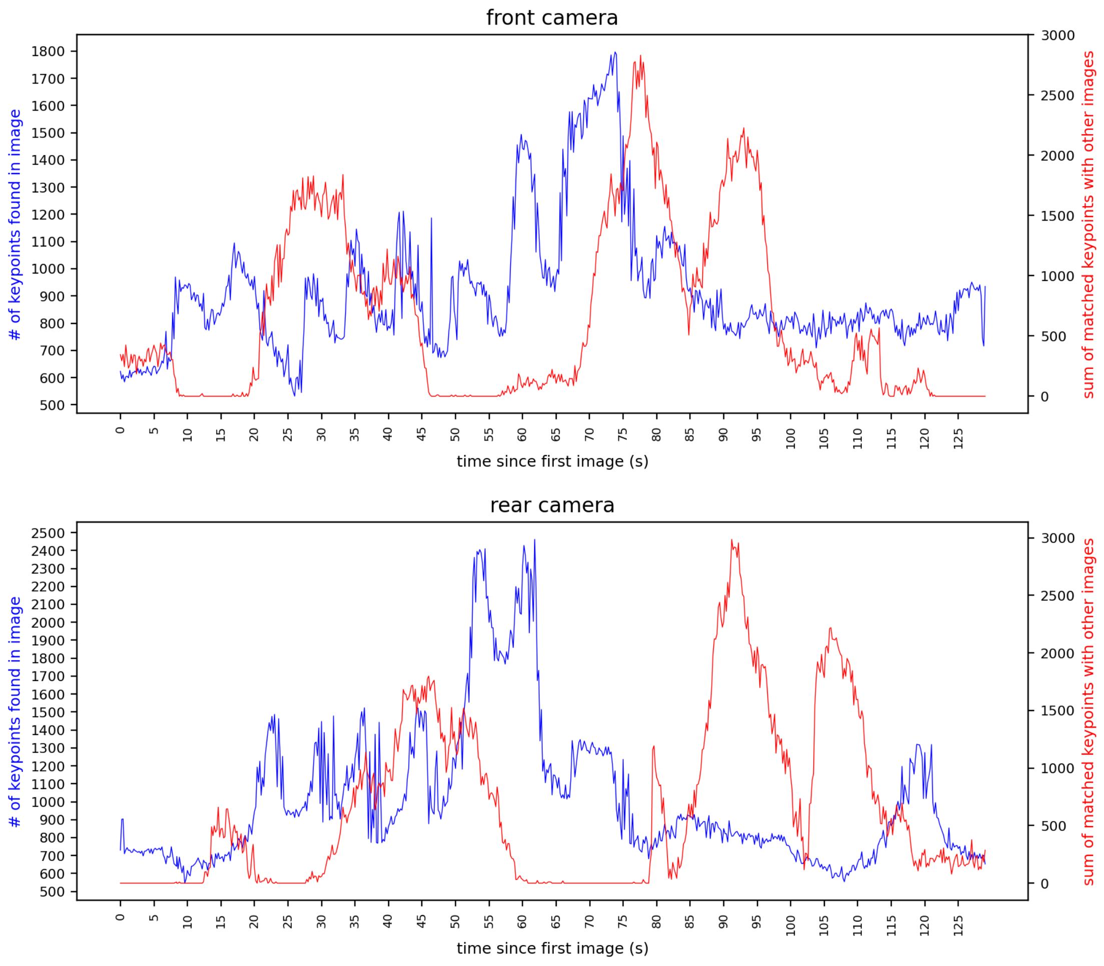


Figure D.1: COLMAP, mirrored rear cam images

D.2 hloc

The images used here are from rosbag 2018-10-22-18-55-41, tag "cloudy". The rear camera images were mirrored (see 3.4.3).
The mean amount of keypoints extracted from images from this rosbag is 572.20. The highest amount of keypoints found is 925, the lowest 359.

time (s)→ time (s)↓	12	13	14	15	16	17	18	19	20	21	22	23
6	74.789	75.333	75.738	76.032	76.119	76.199	76.150	76.040	75.837	75.585	75.249	74.891
7	76.250	76.650	76.9331	77.119	77.099	77.093	76.9362	76.778	76.504	76.188	75.790	75.377
8	77.363	77.626	77.796	77.891	77.774	77.696	77.492	77.245	76.9310	76.539	76.087	75.626
9	78.095	78.245	78.328	78.353	78.155	78.019	77.758	77.459	77.074	76.656	76.159	75.658
10	78.660	78.674	78.672	78.635	78.357	78.173	77.859	77.515	77.084	76.626	76.087	75.551
11	**78.745**	78.724	78.705	78.650	78.311	78.104	77.753	77.376	76.9311	76.421	75.848	75.283

The highest mean amount of features matched in an image: 78.745, with lower_threshold = 11s and upper_threshold = 12s.

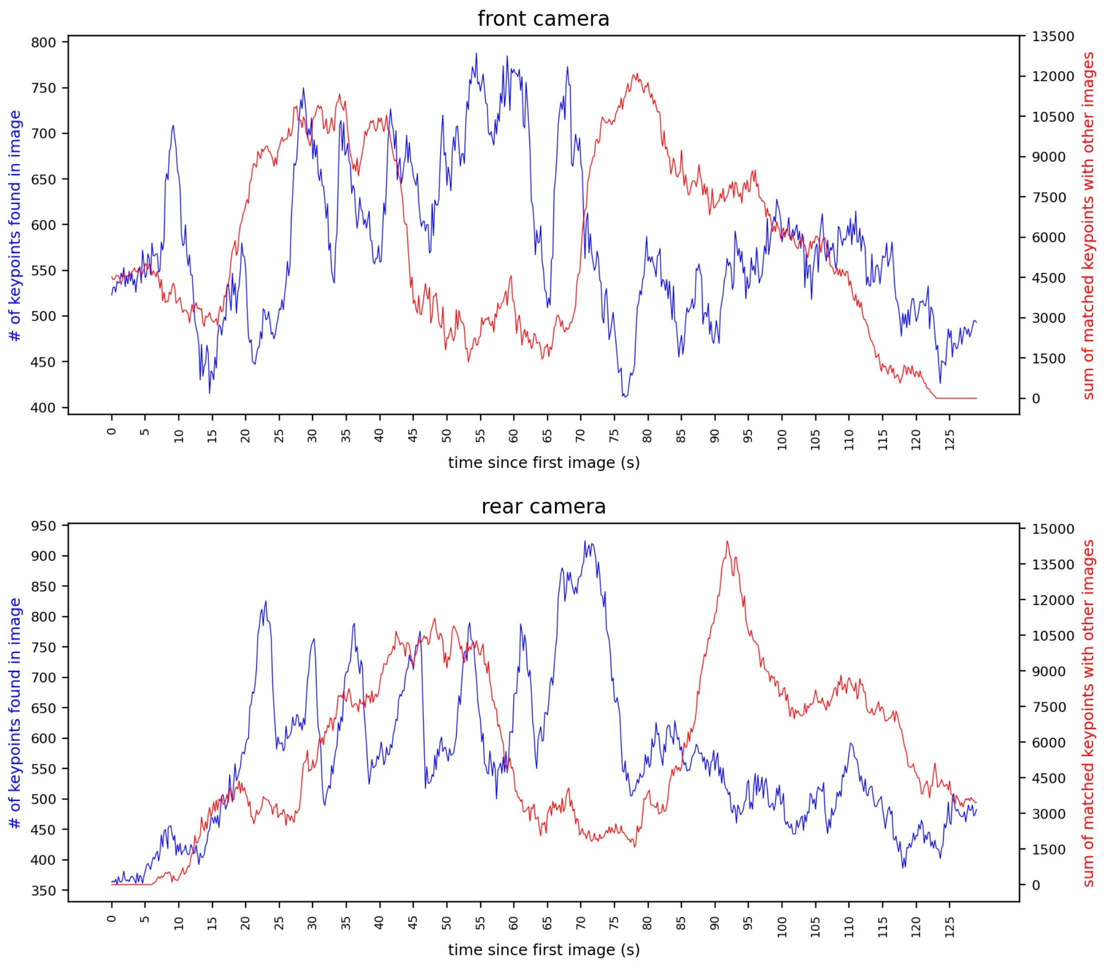


Figure D.2: SuperPoint+SuperGlue, mirrored rear cam images

Bibliography

[1] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,” Tech. Rep., 2015.

[2] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, “A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 829–846, 2018.

[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.

[4] A. Singandhupe and H. M. La, “A review of slam techniques and security in autonomous driving,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 602–607.

[5] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[6] S. Vijayanarasimhan, S. Ricco, C. Schmid, R. Sukthankar, and K. Fragkiadaki, “Sfm-net: Learning of structure and motion from video,” *CoRR*, vol. abs/1704.07804, 2017. [Online]. Available: http://arxiv.org/abs/1704.07804

[7] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superglue: Learning feature matching with graph neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4938–4947.

[8] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk, “From coarse to fine: Robust hierarchical localization at large scale,” *CoRR*, vol. abs/1812.03506, 2018.

[9] D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superpoint: Self-supervised interest point detection and description,” 2018.

[10] P.-E. Sarlin, “Hierarchical-localization,” https://github.com/cvg/Hierarchical-Localization, 2020.

[11] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[12] Y. Bouaziz, E. Royer, G. Bresson, and M. Dhome, “Institut pascal long-term dataset,” in *RFIA*, 2020.

[13] J. P. Barreto, “Unifying image plane liftings for central catadioptric and dioptric cameras,” in *Imaging Beyond the Pinhole Camera.* Springer, 2006, pp. 21–38.

[14] A. Witkin, “Scale-space filtering: A new approach to multi-scale description,” in *ICASSP’84. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 9. IEEE, 1984, pp. 150–153.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017.

[16] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM,” *CoRR*, vol. abs/2007.11898, 2020. [Online]. Available: https://arxiv.org/abs/2007.11898

[17] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 834–849.

[19] O. Özyesil, V. Voroninski, R. Basri, and A. Singer, “A survey on structure from motion,” *CoRR*, vol. abs/1701.08493, 2017. [Online]. Available: http://arxiv.org/abs/1701.08493

[20] B. Streckel and R. Koch, “Lens model selection for visual tracking,” in *Pattern Recognition*, W. G. Kropatsch, R. Sablatnig, and A. Hanbury, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 41–48.

[21] J. L. Schönberger, "Camera models," https://colmap.github.io/cameras.html, 2020, [Online; accessed 10-December-2021].

[22] V. Usenko, N. Demmel, and D. Cremers, "The double sphere camera model," *2018 International Conference on 3D Vision (3DV)*, Sep 2018. [Online]. Available: http://dx.doi.org/10.1109/3DV.2018.00069

[23] F. Devernay and O. Faugeras, "Straight lines have to be straight," *Machine vision and applications*, vol. 13, no. 1, pp. 14–24, 2001.

[24] A. Basu and S. Licardie, "Modeling fish-eye lenses," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, vol. 3. IEEE, 1993, pp. 1822–1828.

[25] C. Geyer and K. Daniilidis, "Catadioptric projective geometry," *International journal of computer vision*, vol. 45, no. 3, pp. 223–243, 2001.

[26] X. Ying and Z. Hu, "Can we consider central catadioptric cameras and fisheye cameras within a unified imaging model," in *Computer Vision - ECCV 2004*, T. Pajdla and J. Matas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 442–455.

[27] B. Khomutenko, G. Garcia, and P. Martinet, "An enhanced unified camera model," *IEEE Robotics and Automation Letters*, vol. 1, pp. 1–1, 12 2015.

[28] "Bags - ros wiki," http://wiki.ros.org/Bags, [Online; accessed 4-January-2022].

[29] J. L. Schönberger, "Colmap tutorial: feature matching," https://colmap.github.io/tutorial.html#feature-matching-and-geometric-verification, 2020, [Online; accessed 10-December-2021].

[30] J. L. Schönberger, T. Price, T. Sattler, J.-M. Frahm, and M. Pollefeys, "A vote-and-verify strategy for fast spatial verification in image retrieval," in *Asian Conference on Computer Vision.* Springer, 2016, pp. 321–337.

[31] "Workshop on long-term visual localization under changing conditions," https://www.visuallocalization.net/workshop/cvpr/2020/, 2020, [Online; accessed 15-December-2021].

[32] T. Sattler, W. Maddern, C. Toft, A. Torii, L. Hammarstrand, E. Stenborg, D. Safari, M. Okutomi, M. Pollefeys, J. Sivic *et al.*, "Benchmarking 6dof outdoor visual localization in changing conditions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8601–8610.

[33] H. Taira, M. Okutomi, T. Sattler, M. Cimpoi, M. Pollefeys, J. Sivic, T. Pajdla, and A. Torii, “Inloc: Indoor visual localization with dense matching and view synthesis,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7199–7209.

[34] C. Hane, C. Zach, A. Cohen, R. Angst, and M. Pollefeys, “Joint 3d scene reconstruction and class segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[35] J. L. Schönberger, “Colmap github,” https://github.com/colmap/colmap, 2020, [Online; accessed 16-December-2021].

[36] ——, “Colmap source code, camera_models.h,” https://github.com/colmap/colmap/blob/dev/src/base/camera_models.h, 2020, [Online; accessed 10-December-2021].

[37] ——, “Command line interface,” https://colmap.github.io/cli.html, 2020, [Online; accessed 24-December-2021].

[38] ——, “Tutorial,” https://colmap.github.io/tutorial.html, 2020, [Online; accessed 23-December-2021].