Vision Pipeline for Self-Driving Cars

Going Beyond the Image Space

Matej Hladky
1903585

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Bachelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

May 3, 2022

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

To Mia and Gina.

Abstract

Perception and scene understanding is one of the key components of self-driving cars. Over the years, many traditional techniques for reconstructing the environment have been introduced, followed by innovative methods exploiting the advancements in Deep Learning. Moreover, we have seen the transition to more vision-based approaches, as cameras are a crucial exteroceptive sensor due to their range of benefits for the self-driving cars domain - they are cheap and suitable for many Computer Vision applications, such as object detection or visual localisation. In this document, we demonstrate the importance of Computer Vision and 3D reconstruction research not only in the context of autonomous driving. Emphasis is placed on the use of stereo visual input and leveraging Deep Learning techniques to enable perception in 3D space, rather than being limited to the 2D image space. We present a comprehensive survey of various methods used in typical reconstruction systems and show specific instances in both industry and academia. Furthermore, we demonstrate a system for generating dense and accurate 3D reconstruction of large-scale environments. Our system uses a modular pipeline consisting mainly of modules for localisation, depth estimation, and semantic segmentation. The reconstructed environment is integrated into a volumetric data structure with a multi-value hash map backend that uses parallelizable and spatial hashing for implicit surface representation. While our processing pipeline prototype is far from being real-time, we provide an implementation using modern methods and libraries that allow for faster performance via hardware acceleration and parallelization. Our results mainly provide a proof of concept and demonstrate the usefulness and importance of our method. We evaluate our results on the KITTI benchmark suite and present suggestions for future work.

Acknowledgements

I would like to thank my supervisor Dr. Michael Edwards for his continuous support, patience, and invaluable advice during the development of this project.

Contents

1 | Introduction

The promise of self-driving cars of making our transportation safer, more convenient, and efficient is clearly immensely attractive. However, even though the dream of autonomous driving with no human intervention is likely as old as the automobile itself, and despite the incredible progress made since the first demonstrations in the late 1980s [1, 2], complete autonomy remains one of the grand challenges of Computer Science and Engineering. The interest and advancements in its development across research centres, universities, and companies of different industries, clearly demonstrate the importance of this cutting-edge technology. So why hasn't the full autonomy been solved yet, when driving a car seems to be such a simple activity at first glance? Firstly, self-driving cars must operate in a highly complex and dynamic environment under strict latency requirements. The amount of data required for the system to generalize well in these unpredictable situations is inconceivable. This has been addressed with physically accurate and scalable simulation platforms [3–5], where we can easily source large amounts of data, especially rare or dangerous scenarios. Secondly, the computer's abilities to reason and make informed decisions through perception are still inferior to those of a human, despite the progress in the development of models for vision and cognition.

Solutions for both aforementioned challenges share a common prerequisite - a precise understanding of the environment's geometry: Dealing with unpredictable situations requires using the spatial information for decision-making directly in the vector space; and to exploit the advantages of simulated training, we need precise 3D reconstructions of the environment. The 3D geometry can be inferred from sensory input, with cameras being a natural choice as the primary source of input - they are extremely cheap in comparison to e.g. LiDAR, which can cost as much as $\$75,000$ for a single car, and they can cover a wide range of tasks required for autonomous driving, as most of the navigational aids (road signs, traffic lights, ...) are designed for human vision. However, cameras require an additional processing step, which is one of their main disadvantages compared to the direct range measurements of active sensors. Furthermore, making decisions only in the image space is not optimal, as it does not reflect the multi-dimensional nature of the real world.

In principle, vision alone should be sufficient to solve self-driving, as humans are able to navigate through the environment using only visual cues - this is one of the reasons the industry

leader Tesla recently announced their cars now rely only on eight cameras positioned around the car [6]. Apart from the domain of autonomous driving, contributing to computer vision research is clearly desirable due to the rise of technologies such as virtual and augmented reality, applications in healthcare, robotics, and many more. Moreover, other fields also benefit from 3D construction methods, as it allows us to gain an insight into qualitative features of an object nondeductive from a single plane of sight.

This project aims to contribute to the research of vision-based systems for self-driving cars by providing a study of both traditional and innovative approaches for 3D reconstruction and scene understanding. We demonstrate the fundamental concepts through comprehensive background research and provide the groundwork for further development by introducing a prototype of a modular pipeline for perception and 3D reconstruction. Given the benefits of such an overview and pipeline, we believe our work is fundamentally interesting for both industry and academia alike.

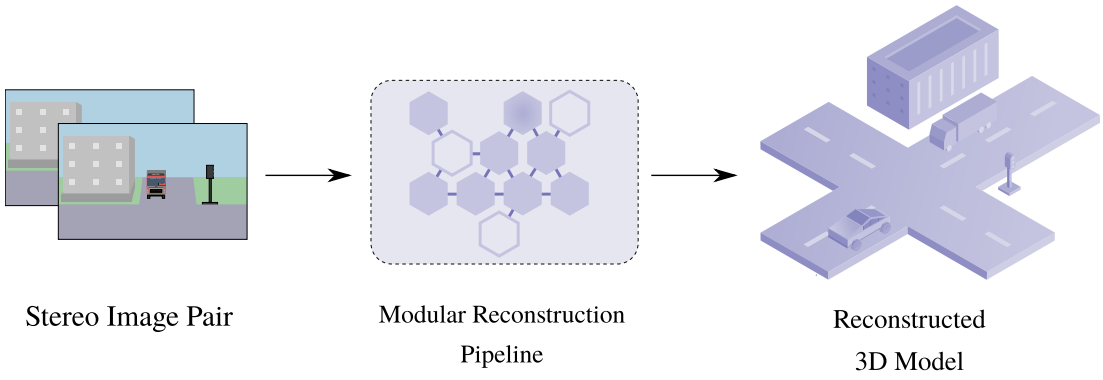


Figure 1.1: **High-level System Overview.** A stereo image input is processed by the modules in our pipeline implementation, resulting in a 3D model of the environment the car has driven through.

1.1 Aims & Contributions

This project aims to develop a prototype of a modular pipeline for stereo 3D reconstruction based on the research of state-of-the-art methods and innovative approaches leveraging Deep Learning. A high-level overview of such pipeline is shown 1.1. The aims and specific contributions of this project are as follows:

1. *Research the wide range of components and methods used in autonomous driving, particularly reconstruction systems.* We consider both traditional and innovative algorithms,

and we select the ones that are robust, manageable in terms of implementation complexity and external requirements, and utilise a reasonable amount of computational resources.

2. *Identify the set of tools available.* For the pipeline implementation, we generally require any tool used to be open-source, actively maintained, and able to perform in ideally real-time settings.

3. *Build a pipeline prototype.* Based on the findings, we aim develop a modular implementation with the following features:

    - Localisation and sparse feature mapping
    - Stereo vision-based geometrical reconstruction of the vector space
    - Reconstruction enhancement with deep learning methods
    - Integration with modern and open-source technologies, as well as update of popular open-source algorithms to work on modern systems
    - Simple extendability and adjustment of the pipeline modules
    - Interface for surface reconstruction in different formats for direct export for e.g. simulation platforms
    - Temporal scenario remodelling
    - Framework for both qualitative and quantitative evaluation of the results

1.2 Document Structure

This document is structured as follows: First, the **Background** section introduces the theory essential for understanding the project, survey of the computer vision and reconstruction methods, and some of the related work in this context. Existing work directly related to 3D reconstruction systems in both industry and academia is discussed in the **Related Work** section. The pipeline prototype implementation, together with a justification for the design decisions, are discussed in-depth in the **Approach & Development Methodology** section. Evaluation of the performance and results of our work are analysed in the **Experimental Evaluation** section. Finally, in the **Conclusion** section, we reflect on the initially proposed project management timelineand aims, together with the challenges and the personal experience gained. We then conclude this document with suggestions for future work and directions that remained unexplored.

2 | Background

In this section, we set the stage for the rest of the document by introducing some of the fundamental concepts, particularly the essentials of Deep Learning and neural networks, Computer Vision and camera-based systems in the context of autonomous driving. Then, we will discuss the components and various methods of 3D reconstruction systems. We assume the reader is familiar with the basics of numerical computation, probability and statistics, and linear algebra. If the reader has no previous exposure to these concepts, we recommend consulting other resources such as [7], [8] and [9].

2.1 Machine Learning and Deep Learning Priors

Machine Learning (ML), often termed "Software 2.0", is a programming paradigm defined as *the ability of computers to learn from experience without being explicitly programmed.*[1] Based on the type of input data, programmers choose an ML algorithm for training an ML model in order for the model to make accurate predictions on data previously unseen by discovering patterns in the input data. Traditionally, ML algorithms are split into three major categories [12]:

- *Supervised learning*: Can be described as a function approximator between the input and output; the algorithm is provided with a set of both input and the target output (i.e., the ground truth) and iteratively updates its parameters to increase the similarity between the predicted output and the ground truth.
- *Unsupervised learning*: Is not provided with the ground truth data and uses only the implicit properties of the model, the parameters and the variables it observes.
- *Reinforcement learning*: The use of *agents* that learn to take *actions* in an *environment* to maximise a cumulative *reward*.

[1] This definition is attributed to Arthur Samuel, a pioneer in the field of artificial intelligence [10], who coined the term "machine learning" in 1959. This definition is likely a paraphrase of [11], as it's not found verbatim in this publication.

Deep Learning (DL) is a subset of Machine Learning that uses algorithms loosely inspired by the functionality of the human brain. Their aim is to extract high-level features from the data progressively, which is achieved with *deep neural networks* (DNNs), that consist of multiple *layers* of *neurons* and *synapses* that have *weights* and *biases* (the parameters of the network) assigned to them, together with functions that control the behaviour of the DNN. Input is passed into the first layer, transformed in the inner layers (called the "hidden layer"), and turned into an output/prediction in the final layer - this process is called the *forward pass* or *forward propagation*. As DNNs are typically an instance of supervised learning, we can formally define a neural network with L layers as an approximation of some function $f^* : \mathbb{R}^n \to \mathbb{R}^p$. Given a set of input-output tuples ($\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^p$), the neural network learns the parameters of a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b})$ that lead to the best approximation. The notion of a *network* then comes from $f(\boldsymbol{x})$ being a composition of many multivariate functions:

$$f(\boldsymbol{x}) : g \circ f^{(L)} \circ f^{(L-1)} \circ ... \circ f^{(2)} \circ f^{(1)}(\boldsymbol{x}) \tag{2.1}$$

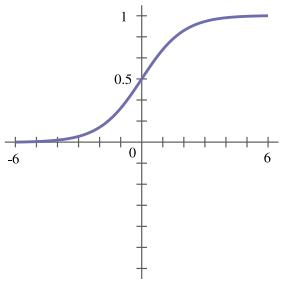
where each of the intermediary functions $f^{(l)} : \mathbb{R}^{n^{(l)}} \to \mathbb{R}^{n^{(l-1)}}$ corresponds to a single hidden layer, and computes a linear combination of the output of the previous layer, the weights matrix $\boldsymbol{W}^{(l)}$ and the bias vector $\boldsymbol{b}^{(l)}$, passed through some activation function ϕ:

$$f^{(l)}(\boldsymbol{x}^{(l-1)}) = \phi(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}) \tag{2.2}$$

The activation function defines the output of a neuron based on the input it receives. Typically, we use nonlinear activation functions that are able to solve nontrivial problems.[2] Often used examples of activation functions include:

- Sigmoid function:

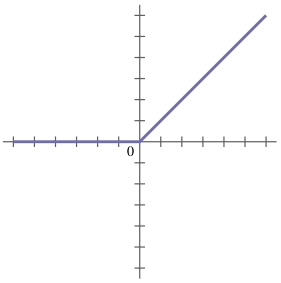
$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$



[2] With linear functions, the network can only model a linear relationship between the inputs and outputs with any number of layers, as summation of linear functions would simply be another linear function.

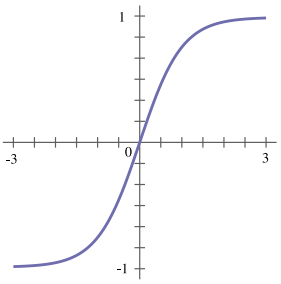
- Rectified linear unit:

$$ReLU(x) = max\{0,\ x\} \qquad (2.4)$$



- Hyperbolic tangent:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (2.5)$$



This model can be conveniently illustrated as an acyclic graph, where each edge represents the synapse between two nodes, i.e. neurons(Fig. 2.1), giving another justification for the choice of the term "network".

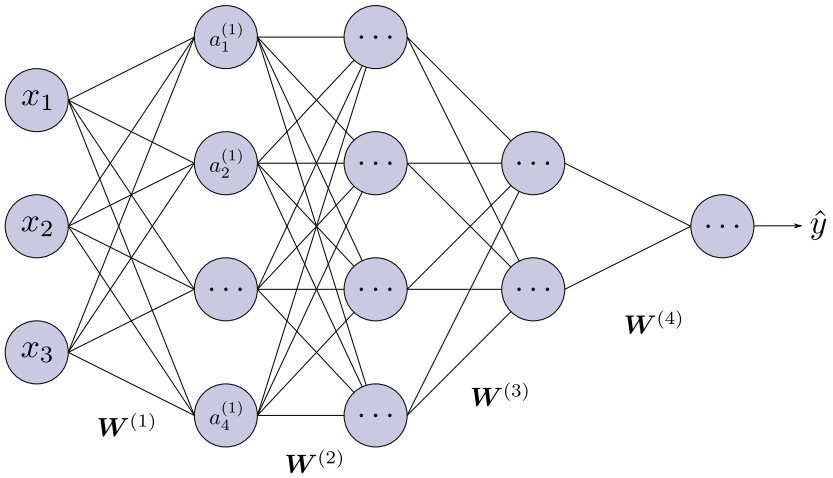


Figure 2.1: **Neural Network.** A simple example of a neural network with 4 input neurons and 2 hidden layers.

DNNs typically involve an optimisation step, which refers to the reduction of the error margin on the predictions measured by a *cost function* (also called *loss function*), which the DNN, therefore, seeks to minimise. Choosing an appropriate cost function comes down to the nature of the problem at hand. A commonly used cost function is, for example, the *Mean*

Squared Error (MSE), often referred to as the empirical risk in the context of empirical risk minimisation. If we denote the predictions vector of the i^{th} sample outputted by the network as $\hat{\boldsymbol{y}}^{(i)}$, and the ground truth as $\boldsymbol{y}^{(i)}$, then the loss function $C = MSE(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)})$ for the whole dataset of m samples is defined as:

$$C = \frac{1}{m}\sum_{i=1}^{m}(f(\boldsymbol{x}^{(i)}) - f^*(\boldsymbol{x}^{(i)}))^2 = \frac{1}{m}\sum_{i=}(\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)})^2 \tag{2.6}$$

After computing the error, the network then adjusts its parameters through a process called *backpropagation*[3], which uses gradients and optimisation algorithms to compensate for each error in a single iteration by updating the parameters, so that the approximation $f(\boldsymbol{x})$ is gradually driven to match $f^*(\boldsymbol{x})$ - this is referred to as the "training" process of the neural network.

Backpropagation computes the gradient in the parameter space w.r.t to the computed loss - the loss/error is "propagated" from the prediction layer back to the input layer, updating the parameters. Each component of the gradient in terms of the individual weights and biases is, therefore, a partial derivative computed by the chain rule:

$$\frac{\delta C}{\delta w_{jk}^{(l)}} = \frac{\delta z_j^{(l)}}{\delta w_{jk}^{(l)}}\frac{\delta a_j^{(l)}}{\delta z_j^{(l)}}\frac{\delta C}{\delta a_j^{(l)}} \tag{2.7}$$

$$\frac{\delta C}{\delta b_{jk}^{(l)}} = \frac{\delta z_j^{(l)}}{\delta b_{jk}^{(l)}}\frac{\delta a_j^{(l)}}{\delta z_j^{(l)}}\frac{\delta C}{\delta a_j^{(l)}} \tag{2.8}$$

where j, k are used for indexing the current and previous layer respectively, and a is the output of some activation function, i.e. $a^{(l)} = \phi(z^{(l)}) = \phi(\boldsymbol{w}^{(l)}a^{(l-1)} + \boldsymbol{b}^{(l)})$.

To be able to propagate the updates backwards, we also need to take into account the derivative of C with respect to the activation, which is affected by all of the parameters of the previous layer (i.e. we cannot directly change the activation of a neuron):

$$\frac{\delta C}{\delta a_k^{(l-1)}} = \sum_{j}^{n^{(l)}-1} = \frac{\delta z_j^{(l)}}{\delta a_k^{(l-1)}}\frac{\delta a_j^{(l)}}{\delta z_j^{(l)}}\frac{\delta C}{\delta a_j^{(l)}} \tag{2.9}$$

[3]The term backpropagation is often used for describing the whole learning algorithm of any neural network. However, backpropagation is only the method for computing the gradients, which are then used by an optimisation algorithm to perform the actual "learning step". This also means that backpropagation isn't a technique specific only for neural networks; in fact, it can compute derivatives of any funcion [12].

There are many important aspects of backpropagation and neural networks that address the inefficiency of computing the gradients individually, such as vectorised implementation, or recursive computation of the gradients only for a single layer at a time. The details of these techniques are, however, out of the scope of this document, and we refer the reader to the formal introduction of backpropagation in the context of neural networks in [13] and detailed discussion in [12]. As a side note, the reader might have noticed the relatively small size of the instruction set used by neural networks compared to the classical imperative approach, which essentially amounts to applying a nonlinearity to matrix multiplication. This computational homogeneity, together with its implications, such as constant computational resource requirements, is one of the paramount advantages of this paradigm.

2.2 Computer Vision

Computer Vision is an interdisciplinary field that aims to develop techniques for analysing and gaining understanding from digital images and video sequences. Common tasks include object classification, object tracking, semantic segmentation, pose estimation, facial recognition, ..., as well as intersections with robotics, signal processing or neurobiology. In recent years, we have seen the trend of Computer Vision applications being used in conjunction with various ML and DL techniques, as the accuracy of these models surpassed the accuracy of the traditional methods across benchmarks [14]. In this section, we discuss this intersection of Deep Learning and Computer Vision in the context of self-driving cars and 3D reconstruction, as well as the inherent properties of the image input space.

2.2.1 Convolutional Neural Networks

The ability to approximate functions in very high dimensions using a multilayered structure of a DNN became very popular for many applications in the Computer Vision domain. Particularly, the *Convolution Neural Network* (CNN) architecture popularized in [15] became one of the key components for many Computer Vision tasks, as it directly exploits the grid-like topology of 2D images. A typical CNN architecture uses a combination of several specialized layer types, which is one of the reasons we use a different way of illustrating the architecture (Fig. 2.2):

- *Convolutional layer* is the main component of any CNN. It performs a mathematical

operation called convolution,[4] that computes the dot product of a *convolutional kernel*, also called filter, with the layer's input. Typically, a single layer uses several kernels whose values are learnt during the training procedure. The kernel creates *feature maps* of the image, i.e. it extracts only the features it deems important.

- *Pooling layer* is an optional layer that has no learnable parameters. It reduces the spatial dimensions of the convolutional layer output by combining sub-clusters of the output into a single value, which makes the receptive field grow faster. Popular types are the max, min and average pooling. Modern architectures [17] tend to use strided convolutional instead of pooling [18].

- *Fully connected layer* is usually used in the last layer. The feature map produced by the previous layers is flattened into an input vector and used as in a classical DNN. Since this "unrolling" increases the number of parameters, some architectures use average pooling before flattening the feature map.

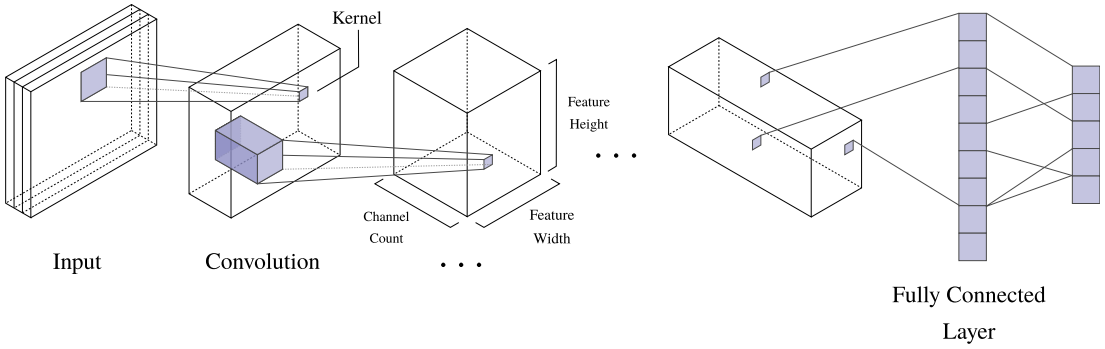


Figure 2.2: **Convolutional Neural Network.** A simple example of a CNN architecture.

Generally, classical convolution is an operation on two functions f and g, denoted as $f * g$. In CNNs, we typically work with a 2D grid of values. The kernel represents a "patch" of values used as a single input connection - this sparse connectivity and parameter sharing allow for equivariant representation and are one of the key ideas behind the invention of CNNs. In the context of Computer Vision, a simple example is a 2D kernel K extracting the features of an image I by "sliding" across the image:

[4] Neural networks, in fact, typically use an operation *similar* to convolution called cross-correlation. For details, see the commutative property of convolution and kernel flipping [16].

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.10}$$

As a side note, the inspiration for using a kernel comes from neuroscience, particularly the visual cortex, in which a group of neurons typically responds only to a certain region of the input. By combining these regions, we can cover the entire visual field. For more information about the neuroscientific basis of CNNs, and perhaps one of the most interesting backgrounds of an ML algorithm invention, we encourage the reader to look into the work of Nobel Prize laureates David Hubel and Torsten Wiesel [19].

2.3 Components of Stereo 3D Reconstruction

While there is a wide variety of techniques and methods for the reconstruction task, most of them share a similar set of core principles. This section analyses these principles in the context of a large-scale reconstruction system based on stereo vision and reviews some of the existing work in terms of the individual components. Existing work related to the overall 3D reconstruction system will be discussed in the following section.

2.3.1 Problem Definition

Without dedicated sensors for extracting the spatial information around the car via direct range measurements, the 3D structure has to be estimated using the information embedded in the image space.[5] The essence of 2D images is projecting the 3D points in the scene onto a 2D plane; thus, inferring the 3D geometry at the most fundamental level boils down to "reversing" this process. Generally, the sequence of steps that constitute such a process involves the following:

- *Camera Calibration* - obtaining the intrinsic and extrinsic parameters of each camera that directly describe the image projection. These are essential for the rest of the core components.

[5] We use the terms global/world *reference frame* and *coordinate system/frame* interchangeably in this work. While there might a distinction between the definitions, we believe it is more important to clarify the usage in the context of a document rather than taking a rigid stance on the meaning of each of the terms.

- *Localisation* - refers to the process of estimating the rigid transformation of the camera with w.r.t. to the global reference frame and is typically given by the extrinsic parameters.
- *Depth Estimation* - computing the per-pixel disparity in the stereo image pair and deriving the depth information of each point.
- *Registration* - projecting the collected depth maps using the estimated pose and camera calibration, and optimizing the alignment of individual depth maps over time.
- *Surface Extraction and Texture Application* - reconstructing the surface from the registered depth maps and optionally applying material color to the model.

2.3.2 Camera Models & Calibration

Projecting a 3D point to a 2D image plane requires knowledge of the calibration parameters of the camera system. The *extrinsic* parameters represent the 6 Degrees of Freedom (6-DoF) camera pose w.r.t the global reference frame. In contrast, *intrinsic* parameters describe the perspective projection of a scene point from the camera reference frame onto the image plane - these are given by the chosen forward imaging/camera model. A commonly used approximation of any conventional camera [20] is the *pinhole camera model* [21], as it is easy to work with and accurate enough for most of the Computer Vision applications, especially for cameras with long focal length of the lenses [22]. This imaging model and the process of of projection a point p in the 3D space onto the image plane is shown in Fig. 2.3

In other cases, the model needs to account for the radial distortions, caused by the bending of the light ray, or the tangential distortions, caused by the lens's vertical axis being non-parallel to the image plane. Popular models that address the distortions are the *Brown Model* [23], the *Kannala-Brandt Model* [24], and the popular *Omnidirectional Model* [25], which extends the Brown Model with a mirror offset.

2.3.2.1 Intrinsic Parameters & Perspective Projection

Intrinsic parameters describe the model's internal geometry, i.e. they describe how a 3D point in the camera coordinate system $\boldsymbol{p}_c = [x_c, y_c, z_c]^T$ is mapped to the image coordinates $\boldsymbol{p}_i = [x_i, y_i]^T$. From Fig. 2.3 and the properties of similar triangles, we know that:

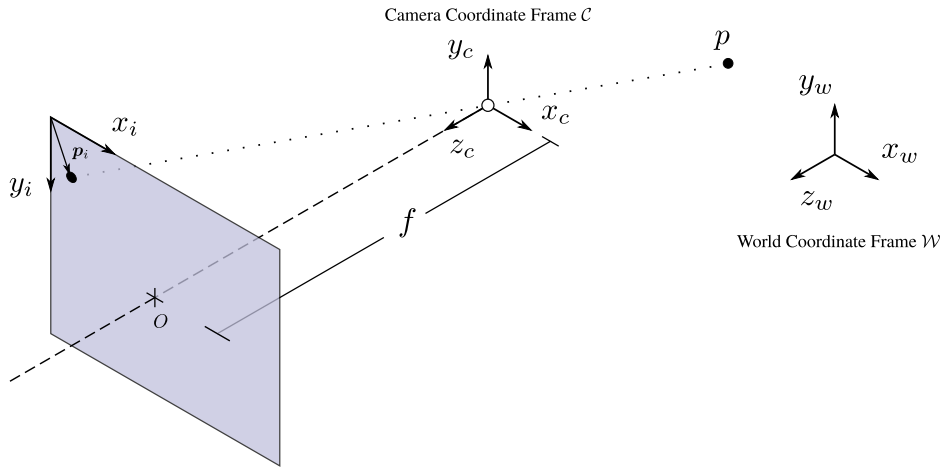
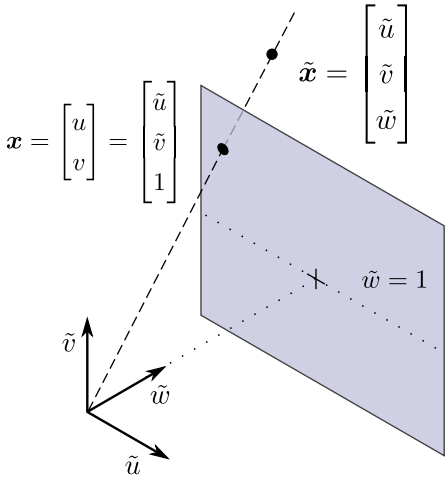
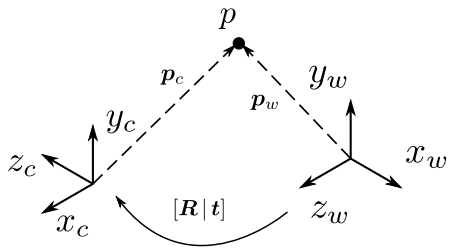


Figure 2.3: **Pinhole Forward Imaging Model.** A point p with the 3D world coordinates (x_w, y_w, z_w) is first transformed to the camera coordinate frame $\mathcal{C}$, and then mapped onto the image plane via perspective projection defined by the focal length f and the optical center O



A. **Homogeneous Representation.** A point on the image plane with coordinates (u, v) described in the homogeneous coordinates $(\tilde{u}, \tilde{v}, \tilde{w})$, assuming the image plane is placed at $\tilde{w}$ = 1



B. **World to Camera Transformation.** The rigid transformation $[\boldsymbol{R} \mid \boldsymbol{t}]$ between the world and camera coordinate systems given by the extrinsic parameters.

Figure 2.4: Pinhole Camera Model - **Homogeneous Representation of Image Coordinates** (left) and **World to Camera Transformation** (right).

$$\frac{x_i}{f} = \frac{x_c}{z_c}, \;\; \frac{y_i}{f} = \frac{y_c}{z_c}$$
$$x_i = f\frac{x_c}{z_c}, \;\; y_i = f\frac{y_c}{z_c}$$
$$\implies \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} f\frac{x_c}{z_c} \\ f\frac{x_c}{z_c} \end{bmatrix} \tag{2.11}$$

Typically, we also separate focal lengths to allow for different pixel aspect ratios in each direction. We also assume that the optical center $O = [o_x, \, o_y]^T$ coordinates are unknown:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} f_x\frac{x_c}{z_c} + o_x \\ f_y\frac{x_c}{z_c} + o_y \end{bmatrix} \tag{2.12}$$

However, this is a nonlinear relationship, thus we convert the image coordinates to homogeneous representation, as shown in Fig. 2.4A. - this simplifies the estimation process. Furthermore, since any point $[\tilde{u}, \;\; \tilde{v}, \;\; \tilde{w}]^T$ corresponds to $[x_i, \;\; y_i]^T = [u, \;\; v, \;\; 1]^T$, we can multiply the homogeneous coordinates with the constant z_c to obtain:

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_x & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = [\boldsymbol{K} \mid \boldsymbol{0}] \, \tilde{\boldsymbol{x}_c} \tag{2.13}$$

where $\boldsymbol{K}$ is called the *calibration matrix*, with $(f_x, \, f_y, \, o_x, \, o_y)$ being the *intrinsic* parameters.

2.3.2.2 Extrinsic Parameters & World-to-Camera Transformation

As shown in Fig. 2.4B., the extrinsic parameters describe the rigid transformation of the camera pose relative to the world reference frame. Let $\boldsymbol{p}_c$, $\boldsymbol{p}_w$ be the coordinates of a point p in the camera and world reference frame, respectively, $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ (or $\mathbb{R}^{3\times4}$ in homogeneous coordinates) the rotation matrix of the camera relative to the world reference frame, and $\boldsymbol{t} \in \mathbb{R}^3$ the translation. Therefore, $\boldsymbol{p}_c$ is given as:

$$\boldsymbol{p}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \Leftrightarrow \tilde{\boldsymbol{p}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.14}$$

2.3.2.3 Projection Matrix

Now the model is calibrated and both operations required for projecting a point onto the image plane are known to us. Thus, we can chain both steps together and define a single projection matrix $\boldsymbol{P}$, so that for any point in the 3D space, we can find its projection in the image coordinates:

$$\tilde{\boldsymbol{u}} = \boldsymbol{P}\,\tilde{\boldsymbol{p}}_w = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]\,\tilde{\boldsymbol{p}}_w \tag{2.15}$$

Estimating the parameters is often done with the use of a fiducial target (an object of known geometry), usually a board with a checkboard pattern. While this approach is used by the vast majority of existing work, [26–30] on of the trends in recent work is to remove the need for such target by relying on the features occurring in the car's environment [31–33].

2.3.3 Localisation, Odometry and Mapping

Knowledge of the global position within the environment is essential for navigation and mapping of the environment. Typically, the algorithms for such tasks assume the real world, albeit highly dynamic, is strictly rigid, and the rigidity constraint is used to make many of the subsequent problems tractable. General-purpose localization subsystems are based on GPS, but they are not applicable on the scale self-driving cars operate, as the GPS signal is not available in many areas such as tunnels or urban canyons [34]. Systems independent of the modern satellite options are therefore highly relevant. As the focus of this work is on vision, we are mainly interested it exploiting the features available in the image space, which is a well-studied problem [31,35,36] with many methods taking advantage of Deep Learning techniques [37–39]. Many of the terms such as localisation, egomotion estimation, odometry, mapping, ..., are often used in the same context without making a clear distinction between them. Thus, we follow the separation into general localisation, visual odometry techniques for egomotion estimation, and SLAM [20].

2.3.3.1 Localisation

We define the term *localisation* as the problem of determining the global pose, i.e. the rigid transformation $[\boldsymbol{R}|\boldsymbol{t}]$ between the fixed world reference frame $\mathcal{W}$ and the camera reference frame $\mathcal{C}$. In other words, localisation is done w.r.t an existing/known map. Localisation is typically split into two types:

- *Metric Localisation*: Directly follows the aforementioned definition; traditionally addressed using Monte Carlo methods. [40, 41]
- *Topological Localisation*: Typically a graph-based representations of a finite set of possible locations, from which an estimation is made. [42–44]

For a better overview and more details about these methods, as well as alternatives addressing their trade-offs, we refer the reader to [20, 45].

2.3.3.2 Egomotion Estimation via Visual Odometry

Egomotion describes the relative 6DoF transformation between consequent input frames, which is estimated with *visual odometry* techniques. As the estimation is done only relatively between two frames at time steps t and $t+1$, visual odometry is sensitive to error accumulation over time. Hence, other techniques need to be introduced for better global precision. General odometry relies on any type of sensors; in autonomous driving, Intertial Measurement Units (IMUs) or wheel encoders are the most common ones. Visual odometry naturally exploits only the visual cues and image features from the cameras, and it can be either indirect or direct. We will talk about these in more detail in the following section.

2.3.3.3 Simultaneous Localisation and Mapping

Visual odometry and other egomotion estimation methods only optimise two consecutive frames. The error accumulation and local-only precision is addressed in Simultaneous Localisation and Mapping (SLAM), where one of the primary key features is loop-closure detection. SLAM is a well-known problem, especially in the robotics domain, in which the tasks of mapping an unknown environment and simultaneously localising the agent in this environment are tackled jointly. Building an online map while localizing the agent in it is a difficult, com-

putationally expensive task, especially in the complex outdoor and large-scale setting.[6] [49] Note that the focus is on operational compliance, rather than creating a dense, geometrically accurate, and visually flawless map.

SLAM system jointly optimizes both the consecutive poses and the global map of the environment. It is usually described in a probabilistic fashion due to the uncertainty in error accumulation, and differences in state definition depending on the application context. In this work, we will solely focus on vision-based SLAM methods. [7]

Generally, visual SLAM algorithms are either indirect (i.e. feature-based), or direct. *Feature-based* algorithms focus on a sparse set of feature points extracted from the image, such as SIFT, ORB, ... The features (also called keypoints) consist of the feature descriptor and its location - this data is used for the tracking and mapping task. Due to the sparsity of the feature set, indirect methods are suitable for embedded settings but often fail in textureless environments [50, 51]. *Direct* methods rely on the intensity and alignment of the input images while minimizing the photometric error. This approach typically leads to a denser mapping, as it considers the pixel-level information rather than a sparse set of keypoints, but also requires more computational resources.

2.3.4 Stereo Vision and Depth Estimation

While there is enough monocular cues in the real world to estimate the depth without the need for stereo input, such as the perspective, motion parallax, the shape and size of familiar objects, lighting effects, . . . , stereo vision allows us to directly exploit the lateral displacement of the cameras (assuming the optical axes are collinear). More specifically, we can find the difference in horizontal per-pixel correspondence for a stereo image pair, and compute the *disparity map* using the intersection of these projection rays via triangulation. We can then easily obtain the 2.5D depth map from the disparity values, as they are inversely proportional to the per-pixel depth.

[6] In the context of self-driving cars, SLAM is often presented as the *solution* to the difficulties with large-scale localisation and mapping, and an alternative for Structure-from-Motion algorithms, that are often used in 3D reconstruction pipelines [46–48]. We will talk about Structure-from-Motion and this comparison in the 3D Reconstruction section.

[7] The notion of vision-based SLAM algorithm might be slightly ambiguous, as these methods are often further classified based on the type of sensor used. The cateogories are e.g. visual-only methods, visual-inertial methods and RGB-D methods. Given this classification, we focus on the visual-only methods, as they rely directly on the monocular/stereo image input. For an in-depth discussion and analysis of these methods, see [50]

2.3.4.1 Image Rectification via Epipiloar Geometry

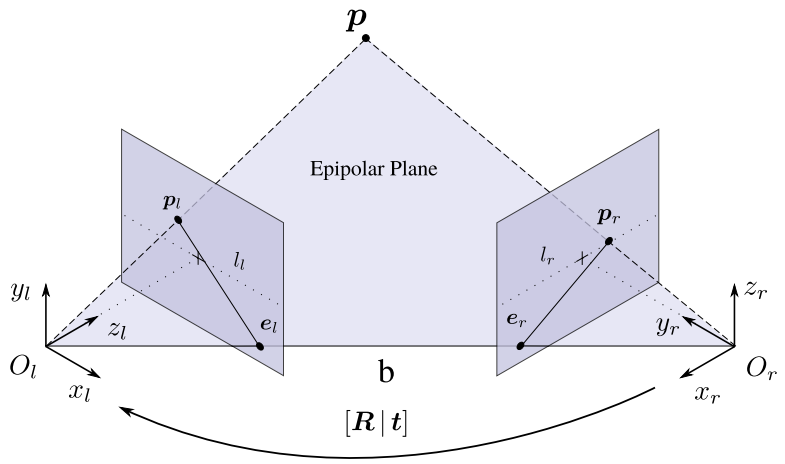
With a stereo camera system, we assume the image planes of each camera are not co-planar.[8] - however, it is desired to transform the image planes onto a common plane parallel to line between the optical centers of each camera, as it significantly reduces the complexity of the three fundamental problems of Computer Vision [52] - the correspondence problem [53].

The calibration process of a stereo camera system is completely described by its epipolar geometry. Consider the left and right camera of a stereo system in Fig. 2.5A., each placed at the optical center/origin O_l, O_r, respectively, of their own coordinate frame. The line between these two origins is called the *baseline* b of the stereo camera pair. Together with a scene point $\boldsymbol{p} \in \mathbb{R}^3$, baseline defines a unique *epipolar plane* w.r.t $\boldsymbol{p}$. The pair of points in the image reference frame of one camera corresponding to the origin of the second camera are called the *epipoles* $\boldsymbol{e}_l$, $\boldsymbol{e}_r$. The intersection of the epipolar plane with the image planes are the *epipolar lines* w.r.t $\boldsymbol{p}$; and we can consequently see the epipoles lie at the intersection of the stereo baseline and the epipolar lines. Furthermore, while any scene point visible by both cameras defines a unique epipolar plane and epipolar lines, the epipoles are independent of this point, as they refer to the projection of the camera origin to the coordinate frame of the other camera. The epipolar plane is used to define an *epipolar constraint*, which includes the relative transformation between the camera pair. In Fig. 2.5A., the system is not rectified, as the epipolar lines are not parallel. For details about the math behind the epipolar geometry, constraint and image rectification, we refer the reader to [54].

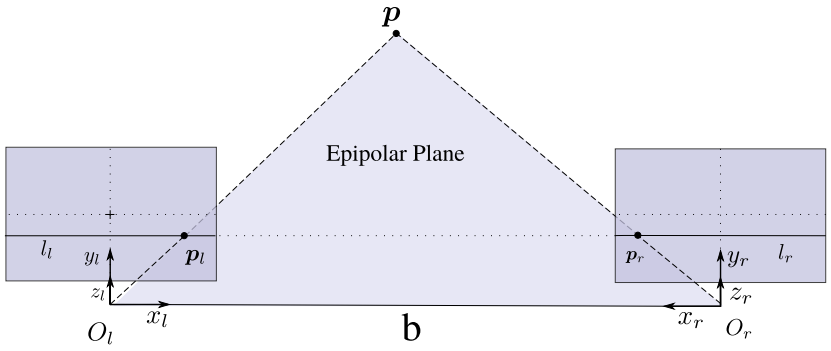
2.3.4.2 Block Matching

After rectification, the correspondence search space is reduced to a single dimension. Next step is to "slide a window" along the line and deploy a cost function to find the lowest dissimilarity score. A commonly used function is the Sum Squared Differences (SSD) or Zero Normalized Cross-Correlation (ZNCC) [55]. This Winner-Takes-All [56] solution by simply accepting the lowest cost, however, leads to noisy estimates due to ambiguities, half occlusions, non-lambertian surfaces,and assumptions made by the algorithm (e.g. all pixel within a single window are displaced by the same amount). The hyper-parameter tuning in the form of

[8] This assumption is posed even for a "perfectly" aligned cameras due to possible precision errors.



A. **Stereo Setup Before Rectification.**



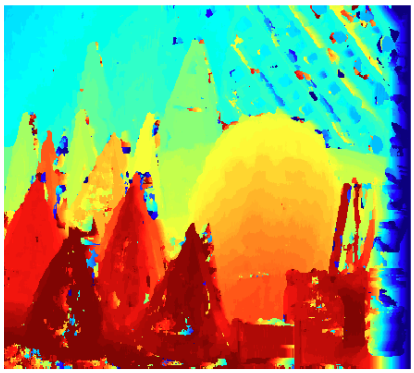
B. **Stereo Setup After Rectification.**

Figure 2.5: **A Stereo Camera System Before (top) and After (bottom) Rectification**. In A., each camera has its own coordinate frame. Baseline b between the camera origins forms an epipolar plane with a 3D point p - each 3D point lies on a unique epipolar line Origins of both cameras as viewed by the other camera define epipoles e_l, e_r. The intersection between the image plane and the epipolar plane, i.e. the line between the point projection and the epipole of one camera, are called the epipolar lines. As the epipolar lines are not parallel, the camera system is not rectified. In B. the system is rectified, as the lines are parallel

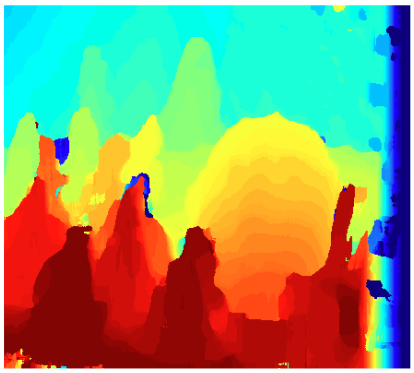
choosing a good window size is a common issue - small windows are prone to ambiguities, large windows are less detailed (Fig. 2.6).

Left Image

Right Image



7x7 Window



19x19 Window

Figure 2.6: **Effect of Different Window Sizes.** An example from the Cones dataset - smaller windows lead to noisy results, while larger windows suffer from precision loss.

2.3.4.3 Spatial Regularization & Energy Optimization

Energy based models [57, 58] are a popular framework for many Machine Learning and Computer Vision algorithms. The energy is a function of the latent variables configuration, thus, the optimization refers to finding a configuration with the lowest energy. The local ambiguities of block matching and similar methods can be solved with a more global approach, such as spatial regularization [59]. Generally, such global methods incorporate some prior knowledge into the stereo matching process. The idea behind spatial regularization is to pose a constraint on the local disparities; specifically, we can define an energy functional E as a base for the

smoothness constraint of a local patch of pixels. A commonly used form of the functional over the whole disparity map $\boldsymbol{D}$ is as follows [60]:

$$p(\boldsymbol{D}) \propto exp\{-\sum_{i} \psi_{data}(d_i) - \lambda \sum_{i \sim j} \psi_{smooth}(d_i,\, d_j)\} \tag{2.16}$$

i.e. the probability of the disparity map is proportional to the energy term consisting of the matching cost $\psi_{data}(d_i)$ and the smoothness between adjacent pixels on a grid with $i \sim j$ 4 nodes $\psi_{data}(d_i, d_j)$. Thus, this approach maximizes the probability w.r.t to a discrete set of disparities in the grid - this is a NP-hard problem, and we can use methods such as belief propagation or cuts on the graph to obtain good approximations. On the other hand, these are typically computationally expensive (e.g., one Megapixel stereo image pair requires more than 3 GB of RAM [61]). A more prominent solution is for example, Semi-Global Matching proposed in [62].

2.3.4.4 Deep Learning Methods

Naturally, a popular modern approach is to learn the stereo matching problem from data using Deep Learning methods. One of the first end-to-end systems for this task was DispNet [63]. It follows a U-Net-like architecture (one of the most effective networks, that will be discussed in a later section) and encoder-decoder module and has no explicit global optimization. There are other models that learn the prediction directly end-to-end [64, 65], as well as models for that focus on learning richer feature representations [66, 67]. An interesting and popular architecture is the Siamese network [68], that learns the similarity score on small patches via two sub-networks that share weights and their output is combined in the final prediction layer.

2.3.5 3D Representation

Methods for representing 3D scenes, objects and surfaces have been studied and discussed in the literature for a long time. Roughly, we can classify the stereo reconstruction as either a Structure from Motion (SfM) method, or a Multi View Stereo (MVS) method. SfM algorithms estimate the calibration parameters jointly, and they are used rather as a coarse mapping technique, sometimes as a backend for an automatic calibration module. In contrast, MVS assumes the calibration parameters to be known and is typically used for a more dense reconstruction.

With stereo vision and MVS, we can generate multiple 2.5D depth maps from different views and directly fuse them into a larger 3D scene if the global camera poses are known. There

also exist many fusion techniques that do not require the knowledge of global poses beforehand [20, 69], particularly the Plane Sweeping Stereo algorithm [70] popular in large-scale urban scene reconstruction. One of the more recent algorithms is, for example, OctNetFusion [71], which addresses issues with occlusions and noise with deep learning techniques. The process of 3D reconstruction can be split into two subtasks - choosing the method for surface representation and choosing the final data structure for output representation.

2.3.5.1 Surface Representation & Intergation

Generally, we can think of the 3D space and objects within it as as a collection of surface manifolds in $\mathbb{R}^2$, i.e. locally smooth geometric subgifures of a certain dimension. Representation of the surface can be categorized as follows (see Fig. 2.7):

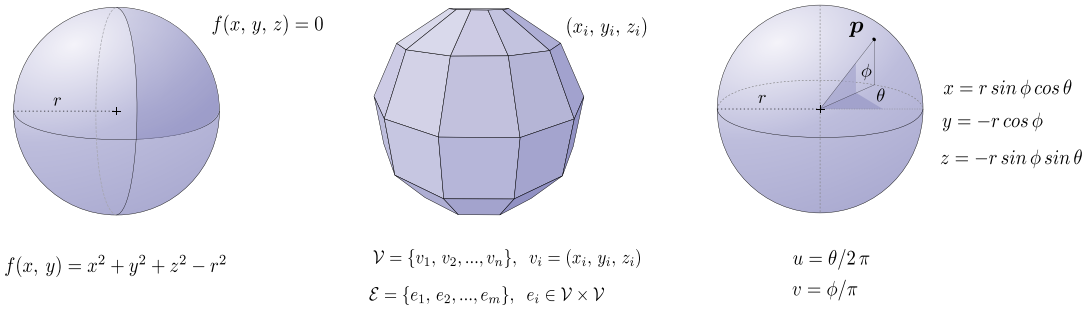
1. *Implicit representation*: The surface is represented as a function of the Euclidian space, more specifically, the surface is defined as the *level set* $\mathcal{L}$ of isocontours of some function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, where the isovalue is 0:

$$\mathcal{L}(f) = \{\boldsymbol{x} \in \mathbb{R}^3 : f(\boldsymbol{x}) = 0\} \tag{2.17}$$

   In this representation, the common convention is that $f(\boldsymbol{x}) > 0$, if $\boldsymbol{x}$ is "outside" of the surface, $f(\boldsymbol{x}) = 0$, if $\boldsymbol{x}$ is a surface point, and $f(\boldsymbol{x}) < 0$, if $\boldsymbol{x}$ is "inside" of the surface.

2. *Explicit representation*: The surface can be approximated with a set of planar polygons. Typically, the explicit representation is in the form of a graph, where each node is a vertex of the surface, and each edge is a polygonal face.

3. *Parametric representation*: The surface is represented as a function of two values, u and v - coordinates of a point $\boldsymbol{p}$ on the surface, i.e. $\boldsymbol{p} = f(u, v)$. Thus, unlike implicit representation, paremetric representation allows us to systematically generate new points on the surface. This attribute is often employed in Computer Graphics applications and APIs, such as OpenGL [72].



(a) Implicit Representation (b) Explicit Representation (c) Parametric Representation

Figure 2.7: Examples of different types of surface representation.

2.3.5.2 Output Representation of the Extracted Surface

While the 2.5 depth maps already summarize the 3D attributes of the scene, they are typically fused into a larger and coherent 3D structure. Commonly used structures include:

1. *Point Clouds*: Simply a collection of the discrete 3D points. While this representation often requires post-processing steps, it is commonly used in Robotics and Computer Graphics. Furthermore, many DL techniques focus on point clouds, for examplem by using them for the prediction representation [73]. One of the most influential approaches used fully connected CNN on each point together with a pooling layer to achieve permutation invariance [74].

2. *Meshes*: Meshes discretize the 3D space into vertices and edges that represent polygonal faces, usually triangles. In DL processing, one of the early works in Geometric Deep Learning used CNNs on the graph spanned by the nodes and face edges [75] for tasks such as classification and segmentation. However, using meshes for the prediction directly does not achieve good results, as the networks tend to suffer from self-intersecting meshes [76, 77] or the inability to guarantee a closed surface [78].

3. *Voxels & Volumetric Representation*: Volumetric representation discretizes the space into 3D cubic subvolumes (*voxels*), each holding a single value of a discrete occupancy function [79], or e.g. a signed distance function [80]. While they are simple to process with neural networks, a naïve implementation of the grid is usually not very efficient, as it requires a large amount of memory ($O(n^3)$) and computation. This has been addressed with a variety of methods, such as data-adaptive discretization of the space with Deluanay

Triangulation [81], or exploiting the sparsity of the 2D surface manifolds by subdividing the 3D space, and proceeding only with the non-empty regions. Among well-established implementations of this approach are for example for example Octrees [71, 82], K-D trees [83], and spatial hashing for hash maps - a method especially popular in the recent years, as it is often scalable for reconstruction tasks [84, 85] and more suitable for parallelization than tree structures, which typically cannot leverage batched operations due to the unbalanced traversal time of spatial queries of individual batches.

One of the most recent and popular methods for scene reconstruction and view synthesis use *Neural Radiance Fields* (NeRF) [86] - a Deep Learning based approach that optimizes a sparse set of input views to generate a full 3D model by optimizing the underlying continuous volumetric scene function. In simple terms, the network is given a set of images together with 5D spatial poses and estimates the volume density and emitted radiance at a given view. By optimizing the differentiable volume rendering, it is able to predict the densities and radiance from unknown poses and synthesize the images into a single model.

3 | Related Work

The previous section highlighted some of the existing solutions for the pipeline components and the theory behind the most prominent ones. In this section, we take a step back and zoom out on the whole reconstruction system. Concretely, we look at some of the design decisions and approaches used by the industry leaders and follow up with a selection of methods shared in academia that directly inspired this work. We consider only systems based on vision.

3.1 Work in Industry

Tesla: Tesla emphasizes the focus on vision, as their cars are currently equipped with only eight cameras, whose input is rectified into a single virtual camera. For the main perception component, Tesla uses a massive neural network architecture called *HydraNet* [87] that consists of a common feature extraction backbone (ResNet-like architecture, namely RegNet [88]) and multiple task-specific heads. As shown during the Tesla AI day [6], Tesla not only implements a reconstruction pipeline, but their whole stack is transitioning to the vector space. For example, their labelling tool allows them to annotate an object only once and then reproject these annotations over time. Furthermore, their reconstruction pipeline can create precise and consistent labels through both space and time automatically, making the labelling process incredibly effective. As Tesla deploys a large number of agents, they are able to maximize the potential of the reconstruction system for e.g. fleet learning and scenario reconstruction, as well as data generation for their advanced simulation platform. An example overview of their pipeline is shown in Fig. 3.1.

NVIDIA: During the GTC keynote 2022 [89], NVIDIA introduced *NVIDIA Drive Map* - an extension to their DRIVE system that serves as a multimodal mapping platform. NVIDIA currently uses three types of sensors for map reconstruction - cameras, LiDAR and radar. Specifically, the camera layer contributes with the visual aspects (lanes, road markings, traffic lights, ...), radar generates the 3D point clouds, which is further enhanced with precise LiDAR measurements. NVIDIA also maintains an Earth's digital twin in their Omniverse engine and uses the generated maps inside the NVIDIA Drive Sim platform.

Waymo: Waymo, a subsidiary of Google, very recently published a large-scale reconstruction of SanFrancisco from 2.8 million photos using a NeRF-based technique [90]. Compared to the previous methods in large-scale settings, Waymo achieved superb results. Furthermore, Waymo introduced appearance modulation based on the fact their cars drove thrugh the same scenes at different times. Thus, we can fuse the information at any time of day without the full information available. Waymo has a particular use case for this functionality explicitly, as they largely rely on virtual training inside of their simulation platform Simulation City [91] and CarCraft [92].

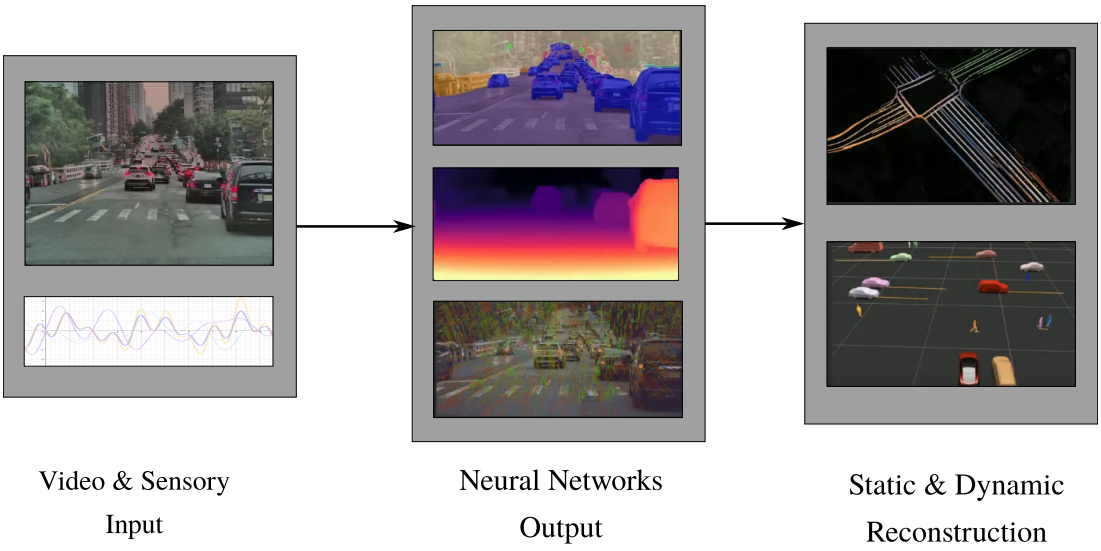


Figure 3.1: **Tesla's Reconstruction Pipeline.** Example components of a reconstruction system shared by Tesla during the Tesla AI Day [6] .

3.2 Work in Academia

We have shown a large body of work from academia focusing on various methods of the individual components. However, these methods are typically presented in the context of a 3D reconstruction system. Furthermore, some of the modules alone, such as SLAM or SfM, are in principle, reconstruction pipelines.. Thus, instead of attempting to draw a line between these two categories, we extend the presented work by mentioning some of the papers that directly influenced the development of this project.

StereoScan [93] presented in 2011 is one of the major works in 3D reconstruction in the autonomous driving domain. This method proposed a feature stereo matching component in conjunction with an efficient and robust VO algorithm and achieved state-of-the-art results on

odometry benchmarks. Furthermore, for the depth measurement, it uses LIBELAS [94] - a library used in this project that will be discussed in the Implementation section. StereoScan uses two worker threads in parallel - one for egomotion estimation and one for disparity map estimation.

One of the first papers we encountered during our preliminary research was work from 2017 published by a group of researchers that introduced the first purely visual perception pipeline [31] focusing on the calibration of multi-camera systems, mapping, localization, and obstacle detection. While the paper's main focus was an automatic calibration of multi-camera fisheye systems via a SLAM-based technique, this paper also introduces a new method for dense height map fusion.

A real-time application for large-scale reconstruction using a direct LSD-SLAM was introduced in [95, 96]. It formulates the 3D reconstruction as an implicit variational energy function, allowing for a fusion with prior depth information and memory-efficient representation. Specifically, they define the model for reconstructing a depth map $\boldsymbol{u}(\boldsymbol{x}) : \Omega \rightarrow \mathbb{R}$ as an energy minimisation [95]

$$E = \int_{\Omega} R(\boldsymbol{u}) + \sum_{i}^{K} C_i^{(d)}(\boldsymbol{u}) + \sum_{i}^{S} D(\boldsymbol{u}, \boldsymbol{i}) \tag{3.1}$$

where $\boldsymbol{x} \in \Omega \subset \mathbb{R}^2$ is the image space, $C_i^{(d)}$ are the data terms for matching the cost. D drives the solution $\boldsymbol{u}$ to be similar to the prior depth information $\boldsymbol{v}_{i..S}$ and $R(\boldsymbol{u})$ enforces the smoothness constraint.

One of the more recent methods that leverage the power of Deep Learning was presented in [97]. Although this work focuses on monocular vision and the 3D reconstruction task servers as a backbone for 3D object detection, it demonstrates the advantages of various Deep Learning based methods and enhancing the reconstruction with information beyond the geometrical structure. The method trains two CNNs for intermediate tasks, such as 2D detection and depth estimation, and deploys a segmentation model in the background. Finally, it uses PointNet [74] for predicting the 3D location, dimension and orientation of each object.

Project AutoVision [98] aims to build a perception system for self-driving cars with multi-camera systems. It focuses on the combination of multi-view geometry and Deep Learning methods for perceiving in 3D space. Apart from cameras, AutoVision uses GNSS and IMU sensors for more robust results. The project contributes mainly with a real-time VO implementation and real-time 3D dense mapping with multiple fisheye cameras. The 3D reconstruction component uses Plane-Sweeping Stereo [70] to compute the depth map alignment, TSDF for

surface integration and ray-casting for extraction. Furthermore, it detects dynamic objects with YOLOv3 [99] neural network and masks out the corresponding pixels to avoid trails of artefacts in the 3D map.

A slightly different kind of work was presented in [100], where the authors addressed the issue of sensor simulation and reconstruction scalability. They describe a novel geometry-aware composition process for rerendering objects extracted from video sequences at novel poses and augmenting different sequences by integrating the objects in it. The synthesized result is realistic, geometrically consistent and traffic-aware.

4 | Approach and Development

As we depicted in the background research, a substantial portion of the state-of-the-art implementations are based on very deep and complex neural networks. Since we cannot compete with those methods due to the limitations of the computational resources at our disposal, we focus on providing a simple, modular and easily extendable prototype of a reconstruction pipeline as a groundwork for further research. This section shows a low-level overview of our system and builds on top of the existing work presented in the Background section by providing details of the implemented algorithms theory.

4.1 System Overview

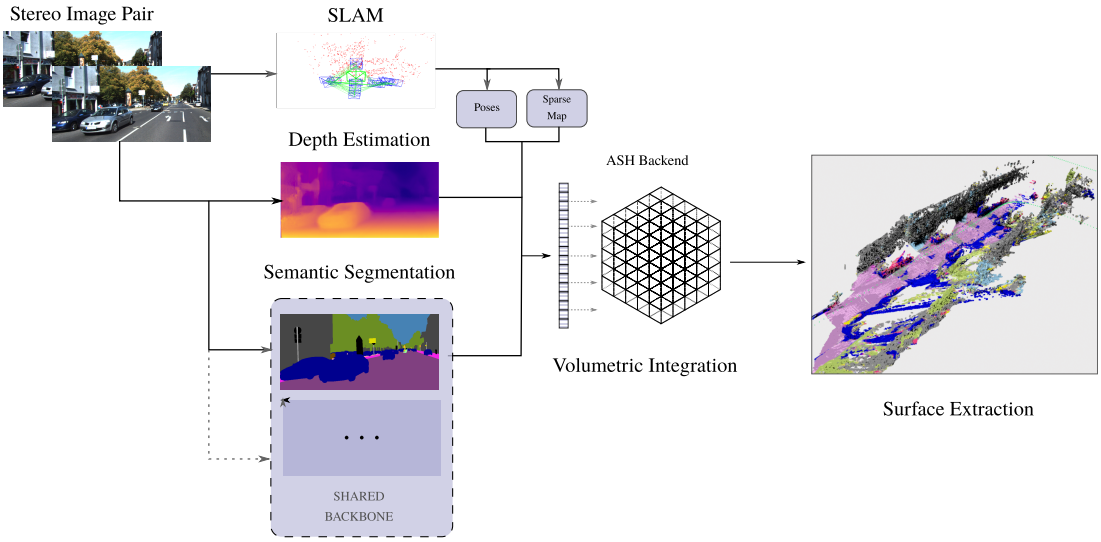


Figure 4.1: Overview of the main components of the pipeline going from a calibrated stereo image input to a 3D mapping.

This project aims to research and provide an overview of various methods for large-scale stereo vision-based 3D reconstruction and implement a pipeline prototype for such a task. Fig. 4.1 provides an overview of our pipeline implementation. Assuming the camera is calibrated, i.e. we know the intrinsic parameters, we first obtain the extrinsic parameters together with a sparse

set of features using a robust SLAM method. The extrinsic parameters are then used by the majority of other modules in the pipeline. Then, the input is fed into a block of parallelizable modules, mainly a dense mapping module, which estimates a dense depth map per camera image, and neural network modules that share a common backbone. Currently, we provide a ResNet [17] backbone implementation; however, only a single neural network head for semantic segmentation is implemented; thus, we separated it from the backbone for the prototype. The semantic segmentation module assigns each pixel in the image to a semantic class. The features extracted from all modules are then fused and integrated into a volumetric representation with a hash map backend described in the Map Representation section. From this map, we can extract the surface via ray casting or in the form of any 3D data structure. Generally, we use point clouds, voxel grids and triangle meshes. Additionally, the depth estimation module can be extended with a completion functionality, as depth maps produced by mainly traditional methods are typically noisy.

Our implementation provides a configuration template containing settings for most of the modules (Listing A.1), including options that are not currently being used (e.g. option to run a calibration module). An example configuration file is provided in Appendix A.

4.2 Tools Used

4.2.1 Programming Languages

Python. Python has been used as the primary programming language, as it is de facto the standard for deep learning and machine learning applications, as well as computer vision. Its simple syntax and high level of flexibility make it a great choice for fast prototyping and iterative development. Many of the libraries used in this project also provide a Python interface.

C++. While C++ hasn't been used extensively in the project directly, many of the tested libraries, frameworks and open-source algorithm implementations use a C++ backend, sometimes providing a Python binding. A substantial amount of time and effort has been spent on either fixing dependency conflicts, testing C++ examples provided by the libraries, or directly implementing new features.

4.2.2 Main Frameworks & Libraries

A large number of existing libraries have been tested during the development of this project. However, many of these libraries are not maintained or cause dependency conflicts with others. For this reason, we only mention the stack used in the final version of the project, despite some of the previously used libraries being a valid alternative.

Open3D. Open3D [101] is "*an open-source library that supports rapid development of software that deals with 3D data*". Open3D provides interfaces for data structures in both C++ and Python, such as point clouds, meshes, and RGBD images, together with various pipelines and processing algorithms. Its backend is highly optimized for parallelization. Furthermore, Open3D provides a set of tools for visualization and debugging, and integration with deep learning libraries via its extension Open3D-ML. Currently, the framework supports two types of implementation interfaces, both natively compatible with NumPy buffers - 1) legacy system 2) new and currently under-development tensor interface. We use the tensor interface whenever possible. Open3D was chosen mainly because of the many core components implemented for 3D manipulation using modern methods, very actively maintained codebase, and the ease of integration with other libraries.

PyTorch. PyTorch [102] was chosen as the main library for deep learning tasks and neural network implementation, despite the original intention to use TensorFlow [103] (as described in the Initial Document of this project), a popular alternative. While an important aspect of this decision was simply a personal preference, PyTorch generally allows for quick prototyping and easier debugging due to its imperative nature (in contrast to TensorFlow's rather declarative style). Furthermore, support for PyTorch in many of the computer vision libraries was more common than for TensorFlow. Lastly, in the self-driving car domain, PyTorch seems to be gaining popularity, as it is being used and recommended by such companies as Tesla, comma.ai, or Nvidia.

Data Science & Other Libraries. Several Python libraries for data science, scientific computing, machine learning and computer vision have been used. These include: *NumPy* [104] - a fundamental package for numerical computation with N-dimensional arrays; *Matplotlib* [105] - a popular library for visualization and plotting graphs; *OpenCV* [106] - a library used mainly for real-time computer vision applications (and the source of the majority of bugs during the

development of this project); *Pillow* [107] - a popular library for image manipulation and processing.

4.2.3 Datasets

Throughout the development of this project, many different datasets and video sequences were used. We list only the most important ones around which the final version of this project is built.

KITTI. KITTI [108–111] is a full benchmark suite for vision tasks for autonomous driving. It contains benchmarks for tasks such as odometry/SLAM, object detection, stereo, optical flow, tracking, ... In 2019, researchers from the University of Bonn shared an extension called "SemanticKITTI" [112, 113], which supports many tasks for understanding LiDAR sequences.

Cityscapes. Cityscapes [114] is a large-scale dataset containing stereo video sequences from 50 different cities. It focuses mainly on semantic segmentation tasks, with 30 available classes, 5,000 fine annotations images, and another 20,000 coarse annotations images.

SceneFlow SceneFlow [115] was used for the major part of training the deep learning-based depth estimation modules. It is a collection of more than 39,000 artificially generated stereo frames for estimating segmentation, depth, optical flow and motion boundaries.

4.3 Pipeline Prototype Implementation

4.3.1 Volumetric Map Representation

We use a volumetric representation of the internal map using a modern, high-performance framework for parallel spatial hashing ASH [116]. This method achieves higher performance on various large-scale 3D perception tasks, and its implementation is open sourced in Open3D. The surface information extracted from the stereo image pairs is embedded in the map implicitly with a signed distance function. We will provide a brief overview of the implementation and map construction, for in-depth discussion and low-level details, we refer the reader to the original paper [116].

4.3.1.1 Initialization: Voxel Block Grid as a Multi-Value Hash Map

Globally, the 2D surface manifolds surfaces occupy only a small part of the 3D space; however they are dense locally (in case of a continuous surface). To represent such structure, we use a voxel block grid with a multi-value hash map backend. The 3D space is first divided into a sparse grid of blocks stored in the hash map by their 3D coordinates. To avoid frequent rehashing that would slow down the performance, a high number of blocks is reserved beforehand. Blocks containing surfaces are further divided into an array of voxels [1]. The hash map supports multiple values for each key, thus we can specify custom properties of each element. By default, Open3D supports truncated signed distance value, element's weight and color. An example overview of the voxel block grid is shown in Fig 4.2

4.3.1.2 Block Activation

To activate blocks containing surfaces, we project the estimated depth map or point cloud in the current viewing frustum. In the ASH implementation, the activation step produces a frustum hash map with the unique block coordinates, and a block hash map used for activating and querying block at these coordinates.

4.3.1.3 Depth Integration with Truncated Signed Distance Function

After obtaining the unique block coordinates, we can accumulate the surface evidence into the corresponding blocks with a truncated signed distance function (TSDF). As this an implicit representation, the surface is defined as the zero-crossing of the function values. Furthermore, as TSDF works like a weighted average filter in the 3D space; integrating more frames leads to smoother surfaces. The integration is a pure geometric process without any hash map operations [101, 116].

4.3.1.4 Surface Extraction

The tensor interface in Open3D currently supports extraction of a triangle mesh and point cloud directly from the voxel block grid. Both use a variation of the marching cubes algorithm [117],

[1] Voxels are stored in a three-dimensional array instead of hash map - this way, we can preserve the data locality within a single block instead of distributing it uniformly into the memory.

which extracts the isosurface from the 3D scalar field. With point cloud, the triangle face generation step is omitted. Both representations can be converted to the legacy system and processes with a wider range of functionalities.

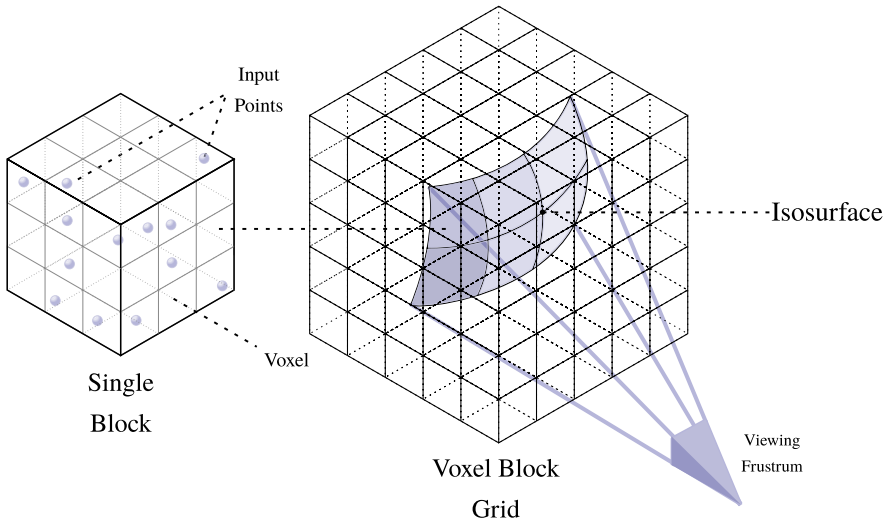


Figure 4.2: **Voxel Block Grid.** Illustration of the volumetric map representation. Blocks in the viewing frustum are activated, and the surface is implicitly represented by the input points integrated with a truncated signed distance function.

4.3.2 Camera Calibration, Image Rectification and Data Preprocessing

All of the datasets used share a development kit containing the calibration files. Thus, we did not implement a custom calibration module, as the intrinsic parameters are known either from the dataset or the camera model, and extrinsics are computed with our SLAM module described in the following section. However, some datasets and/or algorithms use the homogeneous matrix for the extrinsics, while others use quaternion rotation. For this reason, we provide a utility script to convert between the two formats, but the matrix representation is the one used in the pipeline. Similarly, we assume the images are already undistorted and rectified, i.e. the image planes are parallel, as most of the datasets ship in this format. W e do not perform any global preprocessing step before entering the core body of the pipeline, though various adjustments are performed in different modules. These will be discussed in the respective modules. Lastly, we use the pinhole camera model for the point projection.

All modules, especially Deep Learning based components, assume the same dataset structure of *<dataset>/<task>/<split>/<sequence>/<... data>*. Here is an example of this

structure for the KITTI dataset:

```
KITTI
├── disparity
├── raw
└── semantic
    ├── test
    ├── val
    └── train
        ├── img_rgb_left
        ├── semantic_gray_left
        ├── semantic_instance_left
        └── semantic_rgb_left
```

4.3.3 Simultaneous Localization and Mapping

As our prototype implementation is not focused on real-time applications, we decided to rely on robust SLAM solutions and computing the camera poses a priori to the rest of the system, instead of handling the error accumulation from visual odometry (VO). While this setting does not disallow for real-time applications, our implementation described in the following subsections uses a retrospective map and pose optimization at different stages, thus is not suitable for integration with other modules for progressive map building without major adjustments.

There is a plethora of various localisation algorithms both SLAM-based and VO-based system. While our initial idea was to exploit a dense mapping from a dense SLAM method for the geometric reconstruction, we soon realized a robust localisation is more important for our needs, as we can easily add better mapping modules further down the pipeline. From a of various SLAM algorithms shown in the KITTI odometry benchmark [108], we identified four algorithms that are often put into comparison and widely used in research: 1) Direct Sparse Odometry (DSO) [118], 2) Semi-direct Visual Odometry (SVO) [119], 3) ORB-SLAM2 [120] and 4) LSD-SLAM [121]. Apart from the generally faster computation of feature-based methods, and higher accuracy in comparison to LSD-SLAM, we chose ORB-SLAM2 for several reasons: 1) it's fully implemented SLAM framework with several features such as relocalization, 2) it's open-source implementation has been the only one maintained enough for us to be able to run it with relatively minor modifications on modern platforms, 3) it directly supports stereo input and provides a configuration for the KITTI benchmark.

ORB-SLAM2 [120] builds on top of monocular-only ORB-SLAM [122] by introducing features such as RGB-D and stereo extension, using Bundle Adjusment (BA) for depth

error minimization instead of the traditionally used Iterative Close Point (ICP) registration, and lightweight localization mode for map reuse with the mapping thread disabled. At the time of publishing, ORB-SLAM2 achieved state-of-the-art accuracy on the KITTI odometry benchmark. We will give a brief overview of the system and refer the read to the original papers [120, 122] for more details. All following definitions and equations are taken directly from the paper.

4.3.3.1 ORB-SLAM2

ORB-SLAM2 is a feature-based SLAM system that uses the Oriented FAST and Rotated BRIEF (ORB) features developed at OpenCV [123] as an alternative to SIFT and SURF, which are patented feature detectors. The system runs on three main parallel threads for 1) tracking, which localizes the camera in every frame by matching the features in the local map and minimizing the reprojection error with motion-only BA, 2) managing and optimizing the local map with local BA, 3) loop-closure detection that corrects the error accumulation by pose-graph optimization. Finallly, the system launches a fourth thread to compute the optimal structure and motion using via full BA - a method for refining the camera pose while providing a sparse geometrical reconstruction [124, 125]. In motion only BA, the poses are optimized via the mapped features in two adjacent frames, i.e. it minimizes the reprojection error between the mapped points $\boldsymbol{X}^i \in \mathbb{R}^2$ and keypoints $\boldsymbol{x}^i_{(.)}$ as follows:

$$\{\boldsymbol{R}, \boldsymbol{t}\} = \underset{\boldsymbol{R},\boldsymbol{t}}{\arg\min} \sum_{i \in X} \rho(\|\boldsymbol{x}^i_{(.)} - \pi_{(i)}(\boldsymbol{R}\boldsymbol{X}^i + \boldsymbol{t})\|^2_{\Sigma}) \tag{4.1}$$

where ϕ is the Huber cost function used, Σ the covariance matrix that gives the scale of the Keypoint, and $(.)$ refers to either stereo or monocular input.

Local BA optimizes the position of the keyframes instead. More specifically, all keyframes K_f that are not in the set of covisible keyframes K_l, but also observe points P_l, contribute to the cost function, but are not optimized. The optimization is defined as follows:

$$\{\boldsymbol{X}^i, \boldsymbol{R}_l, \boldsymbol{t}_i \,|\, i \in P_l, l \in K_l\} = \underset{\boldsymbol{X}^i, \boldsymbol{R}_l, \boldsymbol{t}_l}{\arg\min} \sum_{k \in K_l \cup K_f} \sum_{j \in X_k} \rho(E_{kj}) \tag{4.2}$$

where X_k is the set of matches between points in P_l and keypoints in keyframe k, and

$$E_{kj} = \|\boldsymbol{x}^j_{(.)} - \pi_{(i)}(\boldsymbol{R}_k \boldsymbol{X}^j + \boldsymbol{t}_k)\|^2_{\Sigma} \tag{4.3}$$

Lastly, in full BA, all keypoins as well as the mapped points are optimized.

ORB-SLAM2 uses a policy for deciding when to insert a keyframe, which was described in first version of the algorithm. In principle, the idea is based on inserting the keyframes often and culling the redundant afterwards, rather than relying on a more complex per-frame insertion strategy. The most important feature of any robust SLAM system is the loop detection ability. ORB-SLAM2 loop-closure module consists of two steps - the first one checks if a loop has been detected, followed by a pose-graph optimization if so. With stereo input, ORB-SLAM2 does not suffer from a large scale drift, as the optimization is based on the rigid body transformation instead of similarities, which were used in ORB-SLAM. A full BA is performed after a loop was detected.

The code for ORB-SLAM2 is open-source, however, we faced many issues during building it on our platform, as it is not actively maintained. For that reason, we provide an updated version[2] as a additional contribution of this project. Mainly, it addresses changes introduced in newer versions of OpenCV 4.X and Pangolin 0.6. Furthermore, it's built for the C++14, unlike the original version, which used C++11. Additionally, we provide an extension to the functionality of the algorithm itself - by default, ORB-SLAM2 saves only the camera positions in either matrix or quaternion format, however the mapped keypoints are showed only during runtime in the GUI. Thus, we extended ORB-SLAM2 so that the map is also saved, although we did not use these points in our pipeline prototype.

4.3.4 Depth Estimation

We experimented with several depth estimation methods, as they are the main building block directly affecting the quality of the reconstruction. In this section, we describe only the main ones that were subject to our comparison. Individual results and comparison are shown in the next section. Furthermore, we experimented with several techniques for *depth completion* - another commonly used benchmark for Computer Vision algorithms, which focuses on recovering dense depth maps from sparse measurements. These measurements can be, for example, the sparse keypoints from LiDAR, or the noisy output of stereo matching. However, we did not fully analyse the effects and benefits of this extension, thus we do not discuss it in this document.

[2] https://github.com/Mateo-tech/orb-slam2

4.3.4.1 ELAS

For the first method, we used a Python binding for the Efficient Large-Scale Stereo (ELAS) [94] C++ library. We provide a brief overview of the algorithm as described in the paper, and the integration in the context of our pipeline. For more details about ELAS, we refer the reader to the original paper [94].

ELAS is a generative probabilistic model for stereo matching. As depicted in the background section, algorithms based on local correspondence typically require a subtle hyper-parameter tuning, particularly the window size. ELAS addresses the common issue with high ambiguity in the stereo correspondences by building a prior disparity space estimate by triangulating a set of robust correspondences, named 'support points'. These are computed using the full disparity range, and via Delaunay triangulation [81]. The prior greatly reduces the search space by focusing only a subset of plausible regions, and it does not suffer from poorly-textured and slanted surface, as the disparity prior is piecewise linear. As this process is parallelizable, the authors achieved a performance of 300 MDE/s (million disparity evaluations per second) on a single CPU core - a speed up of up to three orders of magnitude. The algorithm assumes the images are rectified and in grayscale format. An example result of the depth estimation is shown in Fig. 4.3.

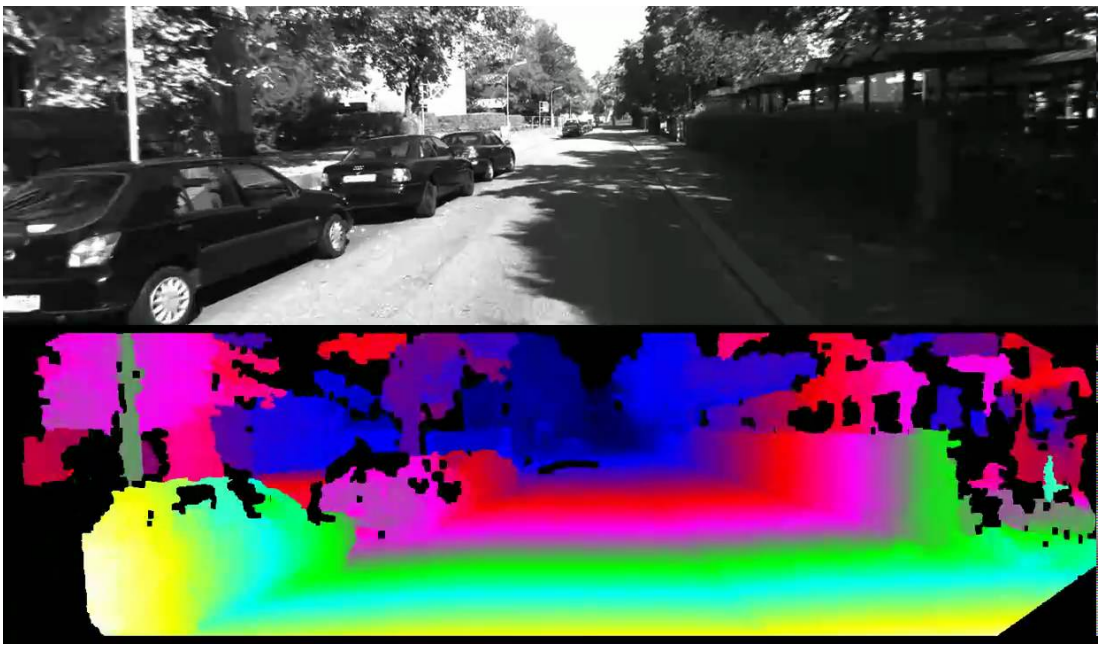


Figure 4.3: Example of a depth map generated by ELAS on the KITTI sequence. Image credits to A. Geiger, author of the ELAS library [94].

4.3.4.2 MobileStereoNet

As we described in the Background section, an alternative approach to stereo matching or spatial regularization is to directly learn the depth estimation problem from data. We use a customized version of the recently published MobileStereoNet (MSNet) architecture [126], trained on the SceneFlow dataset and fine-tuned on the KITTI sequences. MSNet focuses on lightweight architectures and deployment on resource-limited devices without sacrificing accuracy. It provides a "2D" and "3D" model, referring to the type of convolution used in the regularization part - in our pipeline, we use the 2D architecture. It uses a ResNet [17] backbone for feature extraction, and after reducing the channel count, it deploys a newly introduced interlacing cost volume method. While one of the main benefits of this model is its compactness with only 9MB of parameters in our case, the main difficulty we faced was during the training process, which requires several high-end GPUs for larger batch sizes. Due to the memory constraints of our setup, we were only limited to the 2D model trained on a CPU with a batch size of 1. Thus, while MSNet might have been a better option in terms of the reconstruction quality, we have not been able to train a model accurate enough to perform a relevant analysis. Example of a generated depth map is shown in Fig. 4.4.

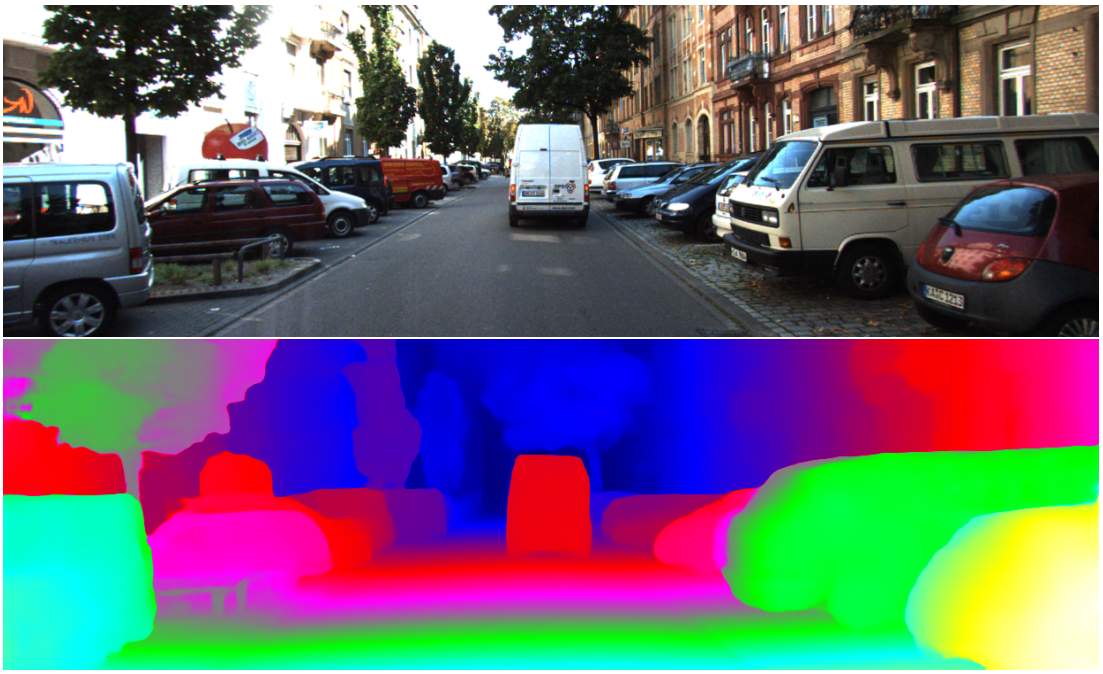


Figure 4.4: Example of a depth map generated by MobileStereoNet on the KITTI sequence. Image credits to the authors of the original paper [126].

4.3.5 Semantic Segmentation

While depth estimation is sufficient for the core geometrical reconstruction, it is common to introduce algorithms for other tasks as to improve the robustness of both the reconstruction and self-driving in general. There are plentiful excellent works introducing various innovative methods, and thanks to the modularity of our pipeline implementation, any of these algorithms can be easily integrated into the system. Semantic information in particular is useful for many of the subtasks, such as localization, obstacle detection or dynamic objects tracking. In Computer Vision and autonomous driving, the goal of semantic segmentation algorithms is to label each pixel with a single semantic class. Furthermore, it can significantly improve the depth estimation process by exploiting prior knowledge about a semantic class, especially in poorly textured areas, where stereo matching failes.

We implemented a neural network based on the U-Net [127] architecture trained on the Cityscapes dataset and finetuned on the KITTI dataset, since KITTI provides only around 200 labelled images. Cityscapes annotations contain 30 classes, however, we use only a subset of 19, as those correspond to the actual objects appearing in the stereo input. The model is easily extendable to a different number of classes by changing the channel output of the prediction layer and adjusting the relabelling function when creating the dataset. Typically, U-Net architectures consist of a contracting ("down") path and an expansive ("up") path. The down part is a classical CNN with convolution layers followed by a ReLU activation and max pooling. The expansive path upsamples the feature maps back to the original size and the final segmentation map is produced. Intuitively, we can think of this architecture as reducing the image size while increasing the channel count to make a class sub-prediction on a spatially small region. These sub-predictions are then upsampled and together with information from the contracting path provided by skip-connections and interpolation and the condensed feature map is expanded to its original size. Our intentions for the semantic segmentation module include:

1. *Redundant Data Removel:* The estimated depth maps usually include a lot of noise, particularly in ambiguous areas or in bad lighting conditions. Semantic segmentation can often recognize these areas a provide a priori information for either the depth estimation module, or before the final fusion. For example, we used the neural network for removing the pixels corresponding to sky, as these were often present in the estimated depth map causing undesired results.

2. *Shape Prior Information:* Similarly, in areas where the depth estimation failes due to poorly-textured areas or flat areas, the segmentation can provide information about these aspects and thus complete the depth estimate.

3. *Detection:* Clearly, semantic segmentation can be used for object detection and recognition, as it gives a per-pixel information about every object in the scene. Typically used examples of this is e.g. road segmentation or car detection.

5 | Experimental Evaluation

5.1 Experimental Setup

Due to the limiations of the computational resources at our disposal, we were not able to produce the best results for a consistent comparison and evaluation. Specifically, we run the majority of computation on an Intel i5 3.6 GHz CPU, 16G RAM, and additionally, an NVIDIA GPU GTX 960. However, the GPU provides only 2G of VRAM, and is not therefore suitable for the majority of processing tasks. This limitation was especially noticeable during any experiments that required a tuning of several hyper-parameters.

5.2 Qualitative Reconstruction Results

We reserve 150,000 blocks in the voxel block grid to avoid frequent rehashing for large scenes, and use various voxel sizes and block resolution, depending on the current scene size. We show an example of a extracted 3D point cloud and voxel grid with semantic labels in Fig 5.1. Detail of a single frame is shown in Fig 5.2.

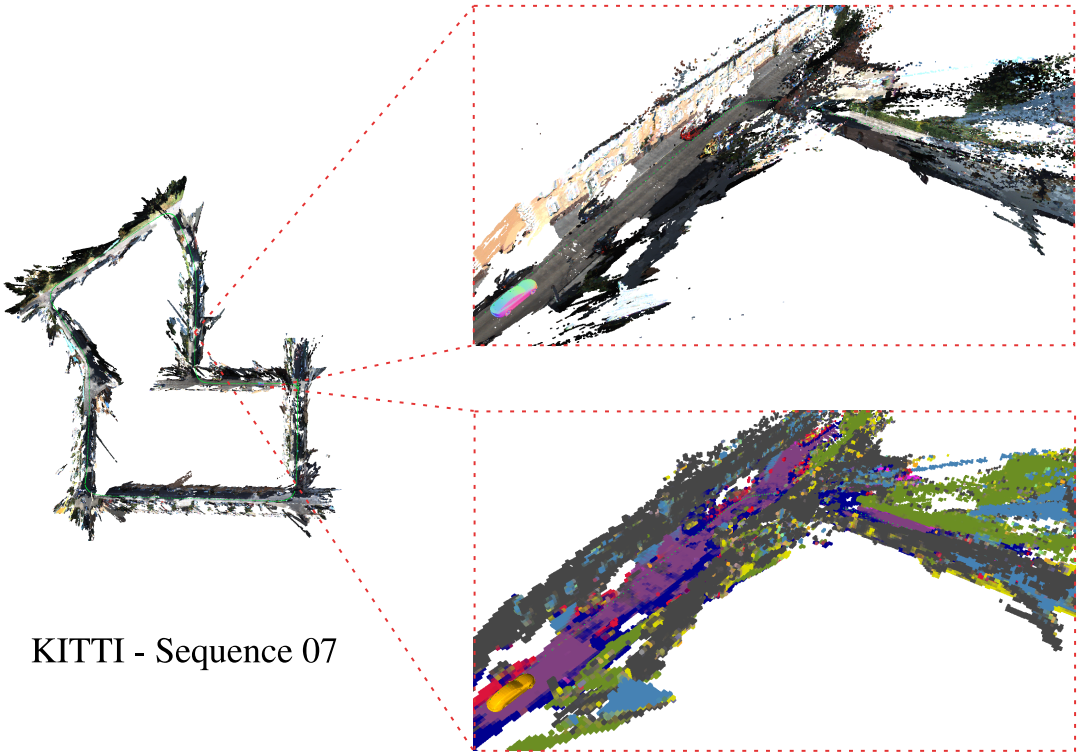


Figure 5.1: Top down view on the reconstrucion result for KITTI sequence 07 (left) with a close view of the point cloud (right top) and a semantic voxel grid (right bottom). The camera pose graph is shown in green.

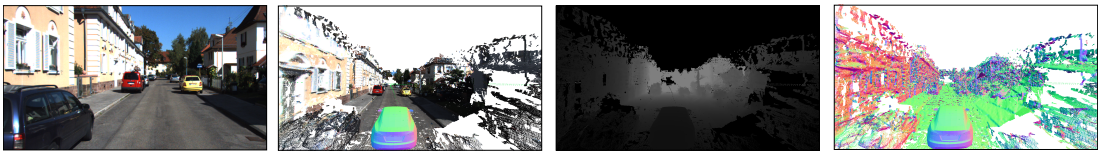
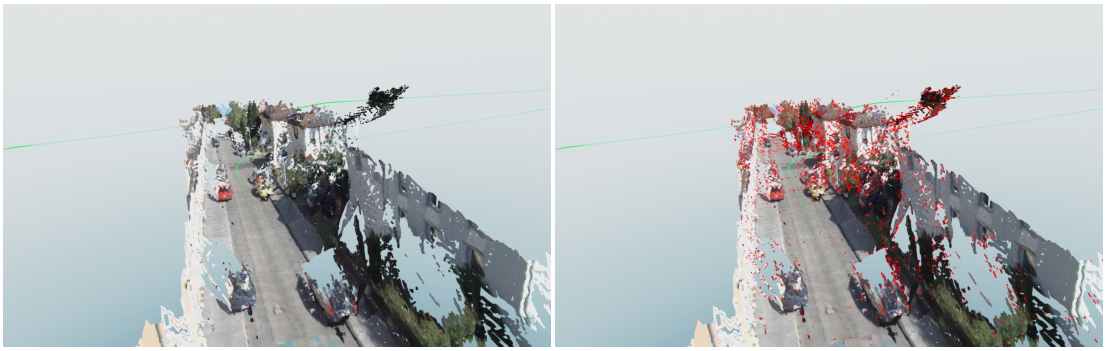


Figure 5.2: Example of a single frame reconstruction (left to right): raw input, reconstructed point cloud, scene depth, normal map.

We scale the image sequence, as lower frequencies usually contain enough information for a dense reconstruction. With more frames per second, we noticed the depth estimation module often leaves artefacts in the form of trails of pixels with infinite depth. We address this with statistical outlier removal:



$input./results/quantitative_analysis.tex$ $input./results/odometry.tex$
$input./results/semantics.tex$

6 | Conclusion

6.1 Reflection on Project Management and Project Outcome

In the initial document of this project, we suggested an iterative development lifecycle built around several deliverables. At the end of each iteration, including the first one, we would have a version of the reconstructed model based on 1) the findings during the research and 2) the rough outline for the pipeline implementation. While the iterative nature of the development process stayed the same, we failed to follow the planned deliverables, as they were either no longer relevant or poorly planned. From this experience, it might seem sensible to focus more on the initial planning to make sure the deliverables are meaningful and well defined. However, we believe that given the broad scope and time constraints of this project, more rigorous planning would only slow the process down due to other difficulties encountered. This risk was mitigated in the initial document and handled accordingly.

One of the risks described in the initial document that became a major issue, was compatibility of various libraries together with the computational requirements of some of the algorithms, particularly Deep Learning models. While we provided a mitigation for these risks, this impacted the development substantially more than expected, as both our initial knowledge and planning were not sufficient. Additionally, at the end of the project development, we faced a critical challenge when due to the memory requirements of one of the models, we lost a substantial part of the codebase. While we followed the mitigating action, we failed to do so to such degree as to avoid the major impact of this issue.

Lastly, the priorization of different aspects of the development process would have admittedly given grounds for more structured work and further research, especially in terms of the result analysis, as the opposite creates a sense of false productivity and fails to produce a more valuable contribution. These issues are not only to be expected, but also a major component of such project. Thus, regardless of the challenges, difficulties, dissapointments and failures encountered, the project outcome is considered a success as it not only met the core requirements, but went into unexpected depths in terms of the theory and range of concepts.

6.2 Future Work

Due to the time constraints and changes in requirements, many of the initially sugested ideas, as well as new ideas disovered during the development, remained unexplored. We give a list of both specific instances and general suggestions for the future.

Depth Completion. As mentioned in the Depth Estimation section, we briefely experimented with depth completion techniques for reducing the sparsity of some of the estimations, or removing the need for a dense mapping solution all together. However, we were not able to produce consistent results that would allow for further analysis and evaluation. As the quality of the depth maps is one of the core aspect of the reconstruction result, depth completion is definitely a direction worth exploring.

Point Cloud Optimization. While we performed several optimization steps during the point cloud generation, such as outliar removal or clipping of the depth map, more advanced optimization is definitely something the project would benefit from, as it directly impacts the quality of the reconstruction.

Parrallelization of the SLAM module. Currently, the SLAM module is the only component of the pipeline that can't be run in parallel, due to the progressive map building and various types of bundle adjustments. As real-time execution of the pipeline might be desirable for specific applications, integrating the SLAM module in parallel with other modules is an interesting feature. However, this would likely drastically affect the design of the overall pipeline.

Evaluation Framework. One of the original aims was a framework for both qualitative and quantitative evaluation. However, due to the scope of the project, we failed to meet this goal, as the number of experiments was too large and we were limited by both time and hardware constraints. Providing a common framework for

Dynamic Object Detection. As the depth maps are generated with each frame, dynamic objects become leave trail artefacts in the final map. While we attempted to address this with an average TSDF filtering and detection using the semantic labels, we did not manage to fully implement this feature.

Deep Learning Methods. We experimented with several Deep Learning models for different components, especially for depth estimation and semantic segmentation. However, due to our limited computation resources, we weren't able to experiment with more advanced architectures or innovative methods, such as the BiFPN's and HydraNet discussed in the initial document, or reach good performance with the models in use. Thus, in the future we would like to transition to a more Deep Learning focused methods.

More Advanced Temporal Remodelling. We believe a reconstruction of a scenario in both space a time is feature of great importance for a pipeline such as ours, as we can directly remodel a situation in which the autonomous driving system failed and gain understanding from it. We provided a very simple utility for showing the car's egomotion based on the pose estimation, however, more advanced implementation that would incorporate e.g. both static and dynamic objects and scene flow is clearly a direction worth exploring.

Simulation. One of the stretch goals discussed in the initial document, as well as an overall focus, was the integration of the reconstruction into a simulation platform and performing various experiments. However, it soon became apparent that focusing on the pipeline itself is signifcantly more valuable, than trying to provide some elementary results in a simulation. Nevertheless, this integration and experiments are still a one of the topics that might be explored, following the development of this pipeline.

Perception Framework. As we gained a deep appreciation for Computer Vision, Deep Learning and self-driving cars through this project, one of the main goal we've set out to achieve in the future, is to extend and generalize this simple prototype to a usable framework that would truly impact the development of autonomous technology, as well as vision applications in general. This aim has not been yet specified in any shape or form, but a sensible direction might be contribution and/or integration with Open3D and its ML extension.

6.3 Summary

This work presented a comprehensive overview of methods for vision-based perception of self-driving cars and 3D reconstruction using stereo input. A comparison of traditional methods and modern Deep Learning techniques was provided in the Background section, followed by a list of specific examples in both industry and academia. From this overview, we discussed

preliminary ideas for a reconstruction pipeline implementation. We then designed and developed a modular, easily configurable and extendable pipeline using mainly Open3D [101] and PyTorch [102], that consists of modules for e.g. SLAM-based localisation and sparse mapping, depth estimation and semantic segmentation. Output of these modules is then fused implicitly into a volumetric representation using an efficient implementation of a parallelizable spatial hashing framework [116]. Our system can generate dense, accurate and efficient representation of a 3D reconstruction of large-scale urban scenes from stereo images. Despite the memory inefficiencies of the volumetric representation, our system can be run on a CPU. We showed a qualitative analysis of our results on the KITTI benchmark suite. We believe our survey of the existing methods, as well as the pipeline prototype, will be valuable for other researchers and companies working towards a full autonomy of self-driving cars. In the future, apart from improving the current pipeline prototype and further analysis, we hope to extend this work to a general perception framework for 3D scene understanding and virtual training.

Bibliography

[1] E. D. Dickmanns and V. Graefe, "Dynamic monocular machine vision," *Machine vision and applications*, vol. 1, no. 4, pp. 223–240, 1988.

[2] C. E. Thorpe, *Vision and Navigation: The Carnegie Mellon Navlab*. Springer Science & Business Media, 2012, vol. 93.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[4] NVIDIA, "Accelerate autonomous vehicle development in the virtual world." accessed: 2022/05. [Online]. Available: https://www.nvidia.com/en-gb/self-driving-cars/simulation/

[5] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *arXiv preprint arXiv:1710.05465*, vol. 10, 2017.

[6] Tesla, "Tesla AI Day," Aug. 2021. [Online]. Available: https://www.youtube.com/watch?v=j0z4FweCy4M

[7] R. Courant and F. John, *Introduction to calculus and analysis I*. Springer Science & Business Media, 2012.

[8] E. T. Jaynes, *Probability theory: The logic of science*. Cambridge university press, 2003.

[9] G. Strang, *Linear algebra and its applications*. Belmont, CA: Thomson, Brooks/Cole, 2006.

[10] J. McCarthy and E. A. Feigenbaum, "In memoriam: Arthur samuel: Pioneer in machine learning," *AI Magazine*, vol. 11, no. 3, pp. 10–10, 1990.

[11] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Automated design of both the topology and sizing of analog electrical circuits using genetic programming," in *Artificial Intelligence in Design'96*. Springer, 1996, pp. 151–170.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[14] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Science and information conference*. Springer, 2019, pp. 128–144.

[15] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[16] Wikipedia, "Kernel (image processing) — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Kernel%20(image%20processing)&oldid=1080112771, 2022, [Online; accessed 25-April-2022].

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[18] R. Ayachi, M. Afif, Y. Said, and M. Atri, "Strided convolution instead of max pooling for memory efficiency of convolutional neural networks," in *International conference on the Sciences of Electronics, Technologies of Information and Telecommunications*. Springer, 2018, pp. 234–243.

[19] E. R. Kandel, "An introduction to the work of david hubel and torsten wiesel," *The Journal of physiology*, vol. 587, no. Pt 12, p. 2733, 2009.

[20] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, pp. 1–308, 2020.

[21] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[22] M. Kutila, J. Korpinen, and J. Viitanen, "Camera calibration in machine automation," in *Human Friendly Mechatronics*. Elsevier Science, 2001, pp. 211–216.

[23] D. Brown, "Decentering distortion of lenses," in *Book title*, 1966.

[24] J. Kannala and S. Brandt, "A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1335–1340, 2006.

[25] C. Mei and P. Rives, "Single View Point Omnidirectional Camera Calibration from Planar Grids," in *Proceedings IEEE International Conference on Robotics and Automation*, 2007, pp. 3945–3950.

[26] B. Li, L. Heng, K. Koser, and M. Pollefeys, "A multiple-camera system calibration toolbox using a feature descriptor-based calibration pattern," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1301–1307.

[27] P. Lébraly, E. Royer, O. Ait-Aider, C. Deymier, and M. Dhome, "Fast calibration of embedded non-overlapping cameras," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 221–227.

[28] A. Geiger, F. Moosmann, O. Car, and B. Schuster, "Automatic camera and range sensor calibration using a single shot," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 3936–3943.

[29] A. Kassir and T. Peynot, "Reliable automatic camera-laser calibration," in *Proceedings of the Australasian Conference on Robotics Automation*, G. Wyeth and B. Upcroft, Eds. Australian Robotics and Automation Association (ARAA), 2010, pp. 1–10.

[30] Q. Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2301–2306 vol.3.

[31] C. Häne, L. Heng, G. H. Lee, F. Fraundorfer, P. Furgale, T. Sattler, and M. Pollefeys, "3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection," *Image and Vision Computing*, vol. 68, pp. 14–27, 2017.

[32] L. Heng, P. Furgale, and M. Pollefeys, "Leveraging image-based localization for infrastructure-based calibration of a multi-camera rig," *Journal of Field Robotics*, vol. 32, no. 5, pp. 775–802, 2015.

[33] L. Heng, G. H. Lee, and M. Pollefeys, "Self-calibration and visual slam with a multi-camera system on a micro aerial vehicle," *Autonomous robots*, vol. 39, no. 3, pp. 259–277, 2015.

[34] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixão, F. Mutz, L. de Paula Veronese, T. Oliveira-Santos, and A. F. De Souza, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742030628X

[35] M. A. Brubaker, A. Geiger, and R. Urtasun, "Map-based probabilistic visual self-localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 4, pp. 652–665, 2016.

[36] J. Ziegler, H. Lategahn, M. Schreiber, C. G. Keller, C. Knöppel, J. Hipp, M. Haueis, and C. Stiller, "Video based localization for bertha," in *IEEE Intelligent Vehicles Symposium Proceedings*, 2014, pp. 1231–1238.

[37] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1983–1992.

[38] G. L. Oliveira, N. Radwan, W. Burgard, and T. Brox, "Topometric localization with deep learning," in *Robotics Research*. Springer, 2020, pp. 505–520.

[39] L. J. Lyrio, T. Oliveira-Santos, C. Badue, and A. F. De Souza, "Image-based mapping, global localization and position tracking using vg-ram weightless neural networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3603–3610.

[40] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, 1999, pp. 1322–1328 vol.2.

[41] S. M. Oh, S. Tariq, B. Walker, and F. Dellaert, "Map-based priors for localization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2179–2184 vol.3.

[42] Y. Li, D. J. Crandall, and D. P. Huttenlocher, "Landmark classification in large-scale image collections," in *IEEE 12th International Conference on Computer Vision*, 2009, pp. 1957–1964.

[43] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven, "Tour the world: Building a web-scale landmark recognition engine," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1085–1092.

[44] J. Hays and A. A. Efros, "Im2gps: estimating geographic information from a single image," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[45] H. Badino, D. Huber, and T. Kanade, "Real-time topometric localization," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 1635–1642.

[46] N. Snavely, S. M. Seitz, and R. Szeliski, "Modeling the world from internet photo collections," *International journal of computer vision*, vol. 80, no. 2, pp. 189–210, 2008.

[47] P. Moulon, P. Monasse, R. Perrot, and R. Marlet, "Openmvg: Open multiple view geometry," in *Reproducible Research in Pattern Recognition*, B. Kerautret, M. Colom, and P. Monasse, Eds. Cham: Springer International Publishing, 2017, pp. 60–74.

[48] S. Fuhrmann, F. Langguth, and M. Goesele, "Mve-a multi-view reconstruction environment." in *GCH*. Citeseer, 2014, pp. 11–18.

[49] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: <italic>common practices and emerging technologies</italic>," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[50] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A comprehensive survey of visual slam algorithms," *Robotics*, vol. 11, no. 1, p. 24, 2022.

[51] S. Y. Loo, A. J. Amiri, S. Mashohor, S. H. Tang, and H. Zhang, "Cnn-svo: Improving the mapping in semi-direct visual odometry using single-image depth prediction," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5218–5223.

[52] X. Wang, "Learning and reasoning with visual correspondence in time," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, September 2019.

[53] W. N. Martin and J. K. Aggarwal, *Motion Understanding*. Springer, 1988.

[54] P. W. Hawkes, *Advances in imaging and electron physics*. Elsevier, 2004.

[55] M. Hisham, S. N. Yaakob, R. Raof, A. A. Nazren, and N. Wafi, "Template matching using sum of squared difference and normalized cross correlation," in *2015 IEEE student conference on research and development (SCOReD)*. IEEE, 2015, pp. 100–104.

[56] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[57] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[58] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[59] J. Huang, A. Lee, and D. Mumford, "Statistics of range images," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, vol. 1, 2000, pp. 324–331 vol.1.

[60] A. Geiger, "Lecture Notes: Computer Vision," Available at https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/autonomous-vision/lectures/computer-vision/, WS 2021/2022.

[61] L. Wang, H. Jin, and R. Yang, "Search space reduction for mrf stereo," in *European Conference on Computer Vision*. Springer, 2008, pp. 576–588.

[62] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[63] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," *CoRR*, vol. abs/1512.02134, 2015. [Online]. Available: http://arxiv.org/abs/1512.02134

[64] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," *CoRR*, vol. abs/1703.04309, 2017. [Online]. Available: http://arxiv.org/abs/1703.04309

[65] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5410–5418.

[66] J. Zbontar, Y. LeCun *et al.*, "Stereo matching by training a convolutional neural network to compare image patches." *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2287–2318, 2016.

[67] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5695–5703.

[68] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015, p. 0.

[69] C. Zach, T. Pock, and H. Bischof, "A globally optimal algorithm for robust tv-l¹ range image integration," in *IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[70] R. Collins, "A space-sweep approach to true multi-image matching," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1996, pp. 358–363.

[71] G. Riegler, A. O. Ulusoy, H. Bischof, and A. Geiger, "OctNetFusion: Learning depth fusion from data," *CoRR*, vol. abs/1704.01047, 2017. [Online]. Available: http://arxiv.org/abs/1704.01047

[72] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[73] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," *CoRR*, vol. abs/1612.00603, 2016. [Online]. Available: http://arxiv.org/abs/1612.00603

[74] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *CoRR*, vol. abs/1706.02413, 2017. [Online]. Available: http://arxiv.org/abs/1706.02413

[75] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *CoRR*, vol. abs/1611.08097, 2016. [Online]. Available: http://arxiv.org/abs/1611.08097

[76] A. Kanazawa, M. J. Black, D. W. Jacobs, and J. Malik, "End-to-end recovery of human shape and pose," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7122–7131.

[77] C. Kong, C.-H. Lin, and S. Lucey, "Using locally corresponding cad models for dense 3d reconstructions from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4857–4865.

[78] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "Atlasnet: A papier-mâché approach to learning 3d surface generation," *CoRR*, vol. abs/1802.05384, 2018. [Online]. Available: http://arxiv.org/abs/1802.05384

[79] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International journal of computer vision*, vol. 38, no. 3, pp. 199–218, 2000.

[80] O. Faugeras and R. Keriven, "Variational principles, surface evolution, pdes, level set methods, and the stereo problem," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 336–344, 1998.

[81] P. Labatut, J.-P. Pons, and R. Keriven, "Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts," in *IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[82] D. Aiger, N. J. Mitra, and D. Cohen-Or, "4-points congruent sets for robust pairwise surface registration," *ACM Trans. Graph.*, vol. 27, no. 3, p. 1–10, aug 2008. [Online]. Available: https://doi.org/10.1145/1360612.1360684

[83] A. Aldoma, Z.-C. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. B. Rusu, S. Gedikli, and M. Vincze, "Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 80–91, 2012.

[84] M. J. Black and A. Rangarajan, "On the unification of line processes, outlier rejection, and robust statistics with applications in early vision," *International journal of computer vision*, vol. 19, no. 1, pp. 57–91, 1996.

[85] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[86] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[87] N. Shazeer, K. Fatahalian, W. R. Mark, and R. T. Mullapudi, "Hydranets: Specialized dynamic architectures for efficient inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8080–8089.

[88] I. Radosavovic, R. P. Kosaraju, R. B. Girshick, K. He, and P. Dollár, "Designing network design spaces," *CoRR*, vol. abs/2003.13678, 2020. [Online]. Available: https://arxiv.org/abs/2003.13678

[89] Tesla, "Gtc 2022 Keynote with nvidia ceo jensen huang," Mar. 2022. [Online]. Available: https://youtu.be/39ubNuxnrK8

[90] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-NeRF: Scalable large scene neural view synthesis," *arXiv*, 2022.

[91] Waymo, "Simulation city: Introducing waymo's most advanced simulation system yet for autonomous driving," accessed: 2021/10. [Online]. Available: https://blog.waymo.com/2021/06/SimulationCity.html

[92] M. DeBord, "A waymo engineer told us why a virtual-world simulation is crucial to the future of self-driving cars," Aug 2018. [Online]. Available: https://www.businessinsider.com/waymo-engineer-explains-why-testing-self-driving-cars-virtually-is-critical-2018-8?r=US&IR=T

[93] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 963–968.

[94] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Asian Conference on Computer Vision (ACCV)*, 2010.

[95] G. Kuschk, A. Božič, and D. Cremers, "Real-time variational stereo reconstruction with applications to large-scale dense slam," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1348–1355.

[96] G. Kuschk, "Efficient large-scale stereo reconstruction using variational methods," Ph.D. dissertation, Technische Universität München, 2019.

[97] X. Ma, Z. Wang, H. Li, P. Zhang, W. Ouyang, and X. Fan, "Accurate monocular 3d object detection via color-embedded 3d reconstruction for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[98] L. Heng, B. Choi, Z. Cui, M. Geppert, S. Hu, B. Kuan, P. Liu, R. Nguyen, Y. C. Yeo, A. Geiger, G. H. Lee, M. Pollefeys, and T. Sattler, "Project autovision: Localization and 3d scene perception for an autonomous vehicle with a multi-camera system," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 4695–4702.

[99] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[100] Y. Chen, F. Rong, S. Duggal, S. Wang, X. Yan, S. Manivasagam, S. Xue, E. Yumer, and R. Urtasun, "Geosim: Realistic video simulation via geometry-aware composition for self-driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7230–7240.

[101] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[102] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[103] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[104] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[105] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[106] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[107] A. Clark, "Pillow (pil fork) documentation," 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[108] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[109] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[110] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[111] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[112] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[113] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss, "Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI Dataset," *The International Journal on Robotics Research*, vol. 40, no. 8-9, pp. 959–967, 2021.

[114] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016. [Online]. Available: http://arxiv.org/abs/1604.01685

[115] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, arXiv:1512.02134. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16

[116] W. Dong, Y. Lao, M. Kaess, and V. Koltun, "ASH: A modern framework for parallel spatial hashing in 3d perception," *CoRR*, vol. abs/2110.00511, 2021. [Online]. Available: https://arxiv.org/abs/2110.00511

[117] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[118] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," in *arXiv:1607.02565*, July 2016.

[119] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "Svo: Semidirect visual odometry for monocular and multicamera systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2017.

[120] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[121] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, September 2014.

[122] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[123] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[124] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[125] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[126] F. Shamsafar, S. Woerz, R. Rahim, and A. Zell, "Mobilestereonet: Towards lightweight deep networks for stereo matching," *Computer Vision and Pattern Recognition*, vol. abs/2108.09770, 2021. [Online]. Available: https://arxiv.org/abs/2108.09770

[127] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

A | Pipeline Configuration Files

The pipeline prototype can be customized for each video sequence via a configuration file, either by adjusting the existing settings, or introducting new. Currently, we support settings for the main modules - SLAM, Depth Estimation, and Map Reconstruction. Here is an example of a configuration file for the KITTI Odometry Sequence 07 [108]:

```
name: kitti_07_config
fragment_size: 100
device: CPU:0
engine: tensor
multiprocessing: false

# ============================
# DATASET SETTINGS
# ============================
path_dataset: '~/Dev/Datasets/KITTI/raw/train/07'
dataset_grayscale: False

# ============================
# CAMERA CALIBRATION SETTINGS
# ============================
run_calibration: False
path_intrinsic: '../configs/calibration/intrinsic_kitti_07.json'
path_color_intrinsic: ''
camera_baseline: 0.54

# ============================
# SLAM SETTINGS
# ============================
run_slam: False
path_extrinsic: '../data/camera_poses.txt'
extrinsic_format: 'matrix'

# ============================
```

```
29 # DEPTH ESTIMATION SETTINGS
30 #==============================
31 depth_estimation_method: 'elas'
32 depth_min: 0.0
33 depth_max: 50.0
34 depth_scale: 1.0
35
36 #==============================
37 # MAP SETTINGS
38 #==============================
39 integration_mode: 'color'
40 voxel_size: 0.02
41 trunc_voxel_multiplier: 8.0
42 block_resolution: 8
43 block_count: 150000
44 surface_weight_thr: 0.5
45
46 #==============================
47 # SURFACE EXTRACTION
48 #==============================
49 surface_extraction_method: 'pcd'
50 optimize_pcd: True
51 pcd_optimization_method: 'statistical'
52 pcd_optimization_radius: 20
53 pcd_optimization_std: 2.0
```

Listing A.1: Example of a partial configuration file for KITTI sequence 07.