GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

ISSN 1612-6793

Master's Thesis

submitted in partial fulfilment of the
requirements for the course "Applied Computer Science"

Fusion of Lidar and Stereo Point Clouds using Bayesian Networks

Felix Schimpf

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

15. August 2018

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
℻ +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 `www.informatik.uni-goettingen.de`

First Supervisor:	Jun.-Prof. Dr.-Ing. Marcus Baum
Second Supervisor:	Dr.-Ing. Andreas Leich

I hereby declare that I have written this thesis independently without any help from others and without the use of documents or aids other than those stated. I have mentioned all used sources and cited them correctly according to established academic citation rules.

Göttingen, 15. August 2018

[illegible]

Abstract

Recent developments in the field of Advanced Driver Assistance Systems (ADAS) have constantly brought technology closer to fully autonomous driving. Nevertheless, several reports of fatal crashes caused by autonomous systems have demonstrated that there are still deficiencies in perception and processing of the environment. A way to enhance these systems is to improve the collaboration of different available sensor systems, an active field of research called sensor or data fusion.

In this work, a novel approach for low level fusion of point clouds is presented. The proposed framework is tailored to two very popular ranging sensors, stereo cameras and Light Detection and Ranging systems (Lidar). A special focus is laid on intelligently handling sensor disagreement by exploting the complementary characteristics of the sensors. A Bayesian network is employed to reason about possible sensor failure based on visual evidence of the scene in question. The framework is then tested and evaluated on real world data from the KITTI dataset.

Contents

List of Figures

List of Abbreviations

ADAS	Advanced Driver Assistance Systems	1
AGIS	Active Gated Imaging Systems	33
API	Application Programming Interface	
DAG	Directed Acyclic Graph	8
ICP	Iterative Closest Point	2
IMU	Inertial Measurement Unit	
Lidar	Light Detection and Ranging	6
MPI	Message Passing Interface	19
OpenCV	Open Source Computer Vision Library	20
PCL	Point Cloud Library	20
ROS	Robot Operating System	19
SGBM	Semi-Global Blockmatching	26

Chapter 1

Introduction

1.1 Motivation

In the minds of most, the dream of autonomous driving is still placed in the world of science fiction. In the recent years, the development of Advanced Driver Assistance Systems (ADAS) has started to move this vision from fiction to science. With increasing availability of cheap and reliable sensor systems as well as efficient processing hardware, more and more tasks can be automated to support the driver. As the main cause of accidents remains human error [2], this isn't merely a convenient feature, but has the potential to greatly improve street safety.

Currently, ADAS can reliably fulfill several tasks, including adaptive cruise control, automatic parking, emergency brakes, lane departure warning systems, and lane keeping systems. While they still require frequent human intervention, existing systems already provide a solid basis to further automatize control of street vehicles.

Several commercial projects have emerged with the goal of developing fully autonomous vehicles for both personal transport and logistics. Some notable companies testing nearly autonomous cars on public streets include Waymo, Tesla and Uber, and the tests already show promising results. The systems are able to perform autonomously most of the time, but all of the vehicles still require operators who can intervene at any time. Curiously, most accidents involving autonomous cars are not caused by system failure, but by mistakes of other drivers [3]. Some accidents that were caused by malfunction of the system however, resulted in fatal crashes and received great media coverage: In 2015, a Tesla Model S crashed into an 18 wheel tractor trailer killing the driver [4]. In 2018, a woman was run over by a self-driving car from Uber, making it the first incident where a pedestrian was killed by an autonomous car [5], leading to the suspension of further research at Uber [6]. In both cases the systems were not able to identify the obstacles as reason to stop.

To ensure a high level of safety of autonomous systems, it is necessary to have reliable information about its surroundings. In the past, many different sensor systems have been developed for the use in the automotive sector. Most approaches like lidar or ultrasonic are purely based on ranging, while radar sensor can provide additional information about an objects relative speed. The concept of vision, either using a single camera or a stereo setup, can be coupled with image based object recognition to supply additional semantic data for better scene understanding.

With the use of multiple sensors generating largely redundant information, the question arises how to process the vast amount of data and make good use of the complementary capabilities of each sensor. The field of sensor fusion (also called information fusion or data fusion), has evolved into a broad discipline with the goal to develop mathematically sound concepts for the integration of heterogeneous data and is applicable in a wide field of applications.

1.2 Fusion Methodology and Related Work

Data fusion can be grouped into three coarse levels: low, intermediate and high level fusion [7]. Intermediate and high level fusion approaches work on various layers of abstraction and generally have no access to single raw measurements. Instead, the data has already been processed and concentrated into a more semantic format. Common examples are the extraction of objects from the raw data of each sensor before fusion, or even tracking said object over several time steps.

The low level operates directly on the sensor data. As the data can come from very different sources, preprocessing may be necessary to achieve a common basis that makes fusion possible. Preprocessing may include synchronization, alignment and correlation of input data [8]. The approach taken in this thesis operates on point cloud data generated by lidar sensors and a stereo camera and falls in the this category.

A very similar sensor setup and is discussed by Moghadam et al. [9]. The introduced framework however was designed for use with an indoor robot and is not suited for use in the automotive sector. The limitation arises mainly from the use of occupancy grids, which are difficult to maintain in scenes with many dynamic (moving) objects. Additionally, the ranges covered by automotive sensors are usually limited to about 100 m, which is often traversed in a short time and further restrain the convergence required for occupancy grids.

Yang et al. [10] proposed a framework for complete three-dimensional reconstruction of streets, largely based on the Iterative Closest Point (ICP) method. The main disadvantage is the negligence of sensor characteristics. In particular, no measures are taken to resolve conflicting detections and filter false positive measurements, which can have grave consequences when used as input for automotive control. Furthermore, no information about computational performance is given and thus no conclusions about real time suitability can be drawn. Similarly, the related technique of point set registration [11] is designed for finding an appropriate transformation between point clouds and is not adequate for the desired fusion.

The method proposed by Baltzakis et al. [12] uses stereo images to refine range measurements generated by lidar. This imposes a clear master-slave hierarchy and conflicts the goal of easy extensibility and hinders the use of more than two sensor systems.

1.3 Thesis Goal

To overcome the mentioned limitations of previous research, a new approach for the fusion of stereo and lidar data needs to be developed.

Many applications in the field of autonomous driving feature a basic pipeline that performs object detection, recognition and tracking. At some point in this pipeline, measurements from different sensor have to be combined to provide a unified representation of the environment. In many approaches, fusion is performed after objects have been extracted from the sensor data, or even later after track-lets or tracks have been determined for the objects. While easy to manage, this process has several drawbacks. Firstly, even after object extraction a lot of information is already lost due to filtering of measurements and abstractions in the object representation. The second problem is that the fusion frameworks used on upper layers are often quite specialized on the type and number of sensors.

The goal of this thesis is to design and implement a software module that provides a low level fusion interface that can be used transparently *before* objects are extracted from the sensor data. It takes point clouds from an arbitrary number of ranging sensors and emits a refined point with increased accuracy. At this point in the pipeline, the algorithm has access to the full bandwidth of information provided by a sensor and can make use of a rather detailed sensor description. Changes of the sensor configuration, like adding or removing sensors, are completely independent of the down-stream processing pipeline and may even be applied during runtime.

As much research regarding measurement fusion has been done with the goal of improved accuracy, this work will mostly focus on the problem of sensor disagreement. To solve this problem arising from contradictory information supplied by the individual sensors, it is only sensible to exploit the different characteristics of the sensor systems. Visual information can be used to make assumptions on the environmental conditions which might lead to sensor failure. A Bayesian network is then employed to decide the trustworthiness of each sensor under these conditions.

1.4 Thesis Structure

In the following chapter 2, a short introduction of the the sensor systems used in this thesis is given and the relevant fusion concepts are explained. Chapter 3 will then state the formal problem definition and develop the concepts used to solve it. In chapter 4, the tools used to implement these concepts are introduced and the basic structure of the software is outlined. The performance of the framework is then evaluated on real-world data in chapter 5, where the algorithm is tested with several difficult scenarios. The last chapter will then draw a conclusion and present an outlook on further improvements.

Chapter 2

Background

2.1 Sensors

2.1.1 Stereo Vision

Concept

Stereoscopic vision is a concept that has been researched for several decades now and remains an active field. It is largely inspired by nature – binocular vision can be found in large parts of the animal kingdom. Binocular vision provides a number of advantages over monocular vision: Among an overall improvement of perception, it is especially useful for enhanced depth perception. The general idea is to combine two two-dimensional images from different viewpoints to reconstruct three-dimensional features and support scene understanding [13]. While in nature several visual cues contribute to perception of depth and distance [14], most computer vision algorithms focus solely on binocular disparity.

Many different algorithms have been published over the years [15], the general idea however remains the same. A certain object recorded by two (horizontally) displaced cameras, will be depicted in two different areas in the respective images. The task of a stereo correspondence algorithm is to identify distinct feature in both images and compute the offset between them, the *disparity*. A low disparity value expresses little offset between the images and therefore indicates that the object is far away. When the parameters of the camera projection are known, disparity maps can be converted directly into a set of three-dimensional points.

Characteristics

Classic computer stereo vision approaches are purely passive, no additional light emitting component is used. Because a setup of two common cameras is sufficient and no specialized hardware is needed, stereo systems are relatively cheap compared to other ranging sensor like lidar or radar. However, stereo algorithms are computationally expensive and differ mostly in the heuristics used to solve the underlying NP-complete minimization problem [16].

As stereo correspondence algorithms purely rely on visual features to determine the disparity, the main problem for the matching process arise from badly textured surfaces. While largely homogeneous areas do not present any features at all, repetitive patterns introduce several possible matchings. Another weakness is the dependency on lighting. Insufficient illumination can pose as much of a problem as excessive overexposure, as in both cases the extraction of features is rather difficult. In general any influences compromising image quality can prove difficult. In the context of autonomous driving, the main sources of irritation arise from harsh weather conditions like rain or fog.

Once appropriate features are selected, the distanced calculation is essentially a triangulation problem. Given the fixed angular resolution of the images, the accuracy decreases for long distances. The uncertainty of a distance measurement can be modeled by a Gaussian with a distance dependent standard deviation [17].

2.1.2 Light Detection And Ranging

Concept

In contrast to stereo vision, Light Detection and Ranging (Lidar) is an active technique, requiring a transmitter and a receiver. While several realizations can be used [18], all of them are based on the time-of flight principle: A light pulse is sent out into the scene by a laser and the reflection is recorded by a photo sensor. The distance can then be computed from the time difference between the sending the signal and the receiving the reflection. A wide field of view is achieved by a rapid sequence of measurements while changing the direction of the light ray, either with a swiveling mirror or prism, or by rotating the whole device, making even a 360° view possible. A three dimensional scan of the environment on multiple layers can be achieve by combining several measurement units.

Characteristics

As the setup of lidar is rather complex, the hardware is usually more expensive than the two cameras needed for a stereo setup. In return, no complex post processing of the data is necessary to retrieve the distance measurements.

Most of the problems presented for stereo vision do not apply to the lidar sensor. As it does not rely on the presence of distinct visual features, plain or repetitive textures do not pose a problem. Additionally, the use of laser light makes lidar mostly independent of external lighting conditions and can be operated in absolute darkness.

Since lidar is still an optical system, weather conditions like rain and fog can still compromise the reliability [19]. In addition, the reliance on the reflection of light imposes it's own problems. Lidar may encounter problems when facing dark or glossy surfaces, or when the angle of incidence is too steep. In those cases the signal gets scattered or absorbed and the light reflected back to the photo receptor may be insufficient to be distinguished from noise and fail to be classified as a measurement. Compared to stereo vision, it is generally more reliable. Although it may not recognize some objects, false positives occur considerably less frequent.

The accuracy of the distance measurements largely depends on the time resolution determining the time-of-flight. As this is independent from the absolute time passed, lidar can generally be considered to have a constant accuracy. Measurement noise can thus be modeled by a Gaussian with fixed standard deviation.

2.2 Sensor Fusion

2.2.1 Basics

Like stereo vision, the concept of sensor fusion is inspired by nature. Nearly all animals are equipped with several different senses and are inherently good at assembling the different kinds of information. The use of multiple different sensory systems has several advantages. For one, a certain redundancy is introduced, making perception more reliable and the loss of one sense can be compensated. A greater benefit is offered by the use of heterogeneous sensors: Complementary characteristics allow for compensation of individual weaknesses.

While nature has perfected this procedure over millions of years, its application in robotics and automation is still challenging. Over the years, many different techniques for various scenarios have been developed. Approaches range from low level concepts operating directly on the sensor data, for example linear least squares, to methods working on processed data, like objects and tracks. Especially the latter often employ more sophisticated procedures, for example the famous Kalman filter [20] or neural networks [21].

This thesis focuses on a low level approach, it operates directly on point clouds generated by each sensor. The algorithm works on single snapshots only, no additional tracking over time is performed. The fusion of individual points is performed using a weighted mean, the specific mathematics will be presented in section 3.3. To handle the case of sensor disagreement, the more advanced notation of Bayesian networks is required, which will be introduced in the following.

2.2.2 Bayesian Fusion

Bayesian Inference

Formally, the fusion task usually boils down to the estimation of a state variable or vector X. The true state is never directly observable and must be estimated on the basis of a set of measurements Y. Due to the inherent uncertainty, the state is represented by the distribution $P(X \mid Y)$, called the *posterior*. Since generally the sensor models only provide the likelihood $P(Y \mid X)$, the famous Law of Bayes

$$P(X \mid Y) = \frac{P(Y \mid X) \cdot P(X)}{P(Y)}$$

is used to infer the state. Note that probabilities in the Bayesian sense are interpreted in a wider sense as degree of belief, rather than representing statistical frequencies in the empirical sense. As this involves integration over the marginal likelihood $P(Y)$, many algorithms have been developed that implement the computation as efficiently as possible [22].

Bayesian Networks

Bayesian Networks provide a formal representation that allows the organization of conditional dependencies in a clean and descriptive way. A Bayesian Network is represented as a Directed Acyclic Graph (DAG), where nodes represent either observable quantities, latent variables or hypotheses. Each edge gives the conditional dependency between two variables. Newly attained information can be integrated at any node an will be propagated through the network to infer the state of other nodes. The network as a whole represents the joint distribution of all variables, which can incorporate arbitrarily many and complex conditional relationships [23].

In many cases it is sufficient to store discrete distributions in the nodes. The value of each category the represents the belief that this is the true state. The conditional probabilities are stored as tables, mapping each possible combination of parent states to a distribution in the child node. While the tables can be filled by hand, possibly using expert knowledge, techniques for automatically learning appropriate connection weights exist [24].

In general, Bayesian networks provide a complete model for all variables and their relationships. Observation of further evidence will be propagated to all other nodes. Additionally it is possible to make arbitrary queries given some evidence. The Bayesian network used in this thesis however requires only the determination of the value of a single node, which can be attained using a pure feed forward model. By this simplification, a number of complex update rules can be omitted and thus their explanation will be skipped.

Chapter 3

Design

In this chapter the design principles of the fusion framework are described. At first, the problem and requirements are stated and formalized, afterwards the basic principles of preprocessing are explained. The last part will discuss the fusion process, which will be explained in two parts, the measurement fusion and the confidence evaluation.

3.1 Problem Definition and Requirements

The goal is to develop an algorithm which takes two or more point clouds and a camera image as input. An input point cloud $\mathcal{I} := (P_1^{(\mathcal{I})}, \dots, P_N^{(\mathcal{I})})$ is a list of N points $P^{(\mathcal{I})} := (x, y, z)$, each consisting of three coordinates x, y and z. The coordinate system is setup such that $+x$ points forward, $+y$ points left, and $+z$ points upwards.

The output of the algorithm is a single point cloud $\mathcal{O} := (P_1^{(\mathcal{O})}, \dots, P_M^{(\mathcal{O})})$. In addition to the x, y and z coordinate, each output point is augmented with a confidence value c, indicating the reliability of a measurement concerning false detections.

The fusion process has two objectives: The fusion of related points from the input point clouds into representative points, and the classification into noise and true measurements. The algorithm emulates a virtual sensor, which is placed at the point of origin and casts virtual sensor rays into the scene. The aim of the fusion is to deliver improved accuracy of the measurements, thus the respective measurement models should be incorporated in the calculation.

The purpose of the confidence evaluation is to supply hints whether a measurement originates from a detected object or sensor failure, especially in cases of sensor disagreement (detection vs. no detection). The camera image is used to assess the environment and evaluate the reliability of the input measurements.

The algorithm assumes that the inaccuracy of each measurement is gaussian, where the standard deviation depends on the sensor model and the measured distance. Furthermore, during this chapter it will be presumed that the input data is readily prepared (synchronized in time and aligned in a common coordinate system), and the sensor model $\sigma : \mathbb{R}^3 \rightarrow \mathbb{R}$ is supplied, which maps the coordinates of a point to its standard deviation. A detailed description of this data acquisition and preparation can be found in the following chapter.

3.2 Preprocessing

Before the prepared point clouds can be fused together, several preprocessing steps are necessary to concentrate the input data. These steps are performed on each input point cloud independently and result in an efficient representation of the measurements.

The goal of the preprocessing is to overcome difficulties introduced by different angular resolutions of the sensors: Higher resolution results in denser point clouds. As high angular resolution can not be equated with a higher accuracy of the measured distance, the fusion algorithm would then have to explicitly account for the difference in resolution to avoid over-representation of the denser point cloud, when calculating the weighted mean. This would require very precise and difficult tuning of parameters.

Instead, the problem is overcome with a more robust approach, the binning and segmentation, resulting in a uniform representation of measurements.

3.2.1 Binning

In the binning step, each input point cloud is partitioned into equally sized bins. Each bin covers a small fraction of the sensors field of view, mimicking the measurement beams cast by a ranging sensor from the origin of the coordinate system (Figure 3.1).

The sensors horizontal field of view, denoted by the interval $[\Phi_{min}, \Phi_{max})$, is thus divided into a series of l equally sized, adjacent bins $B_1, \ldots, B_l$. Each bin then represents a small section of $B_i := [\Phi_{min} + i \cdot \delta, \Phi_{min} + (i+1) \cdot \delta)$, where $\delta := \frac{\Phi_{max} - \Phi_{min}}{l}$. A point $P^{(\mathcal{I})}$ is assigned to the i-th bin, iff its angle $\phi(P^{(\mathcal{I})}) := \operatorname{atan2}(y, x)$ falls in to this interval. Any points that lie outside this field of view are not assigned to any bin and are discarded.

While the concept of bins resembles the measurement beams of a sensor, the bins usually do not correspond to the actual measurement beams of the sensor. Especially for the sensors with a higher angular resolution, the bins combine multiple measurements to artificially lower the resolution for further processing. Even on the lower resolution sensors the bins might contain multiple measurements to reduce the total amount of data.

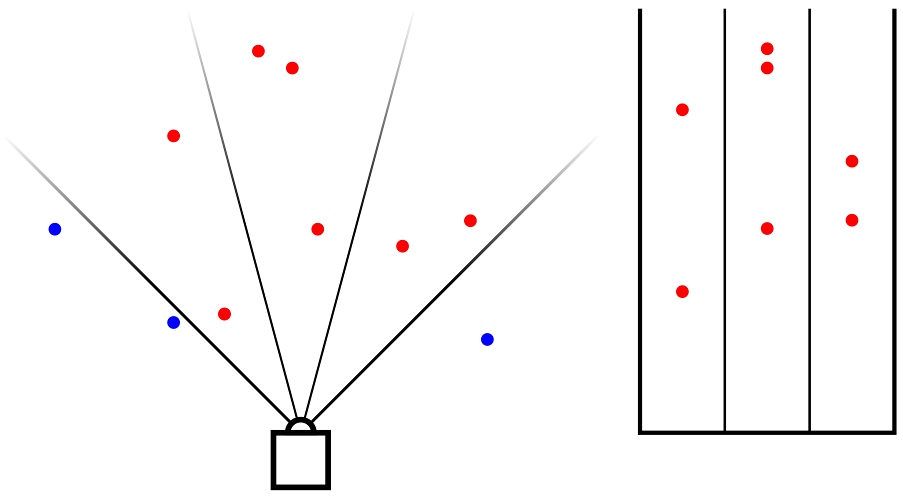
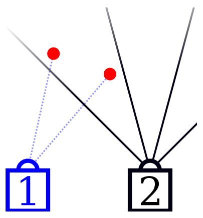


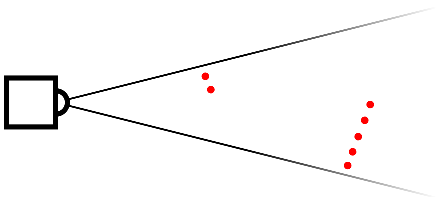
Figure 3.1: Visualization of a point cloud partitioned into 3 bins. The blue points are outside of the configured field of view and thus discarded.

3.2.2 Segmentation

The segmentation step is applied to each bin individually. By grouping several nearby measurements into representatives, the overall amount of data is reduced while preserving most of the information. The notion of a bin representing a measurement beam suggests, that each bin is reduced to a single measurement. However, this would potentially result in loss of important information, as multiple objects may be recorded in a single bin: The first scenario arises from data preparation, when the input point cloud is translated during alignment, causing an offset between the sensors coordinate system and the coordinate system used for processing. This leads to a situation where two objects which are seen side by side one sensor, now appear behind each other and are assigned to the same bin (Figure 3.2a). The second scenario is caused by the binning concept itself: If the range of a bin covers multiple measurements, multiple objects can be recorded in the bin (Figure 3.2b).



(a) Different objects detected by the original sensor (1) might appear behind each other when viewed from the virtual sensor (2).



(b) If the bin is much wider than the angular resolution of the sensor, multiple object may be recorded.

Figure 3.2: Multiple objects might be recorded in a single bin

In both cases using only one representative measurement for a bin would lose one of the objects. Instead, a series of representatives is extracted from each bin to preserve most of the geometric information. The points in one bin are thus grouped together based on the distance to each other. Each group is then merged into a single representative. Instead of employing a complex clustering algorithm, the measurement beam notion is exploited and each bin is interpreted to be one dimensional. Each point can therefore be evaluated based only on its distance and the standard deviation supplied by the measurement model. This way, a much simpler segmentation algorithm can be used, which only requires tuning on one parameter.

The output of this step is a list of segments. A segment is an augmentation of the input point, denoted as four-tuple $S := (x, y, z, r)$. The x, y and z coordinate are inherited from the representative point, while the range value r denotes the expansion of the segment. In the following

$$d(P) := \sqrt{x^2 + y^2 + z^2}$$

denote the (euclidean) distance of a point P. As the segment inherits the coordinates, the notion $d(S)$ is the equivalent for segments.

To simplify the segmentation, the points are sorted in ascending order according to their distance. The segments are then chosen as sequence of consecutive points, such that the condition

$$d(P_i^{(\mathcal{I})}) + a \cdot \sigma(P_i^{(\mathcal{I})}) \leq d(P_j^{(\mathcal{I})}) - a \cdot \sigma(P_j^{(\mathcal{I})})$$

holds true for every pair of adjacent points $P_i^{(\mathcal{I})}$ and $P_j^{(\mathcal{I})}$. That is, two points are merged when their respective standard deviations, stretched by a constant factor a, overlap. The first point of the sequence is chosen as the representative and the range value r is set to the distance difference of the first and the last point. An example segmentation is show in Figure 3.3.

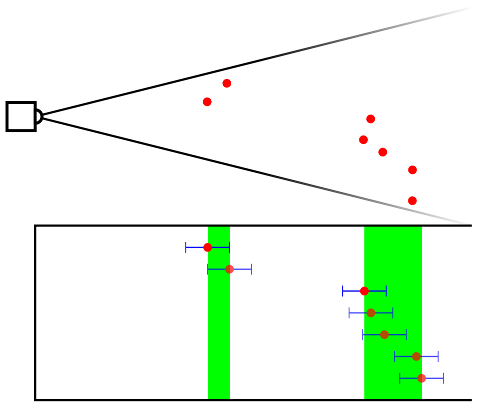


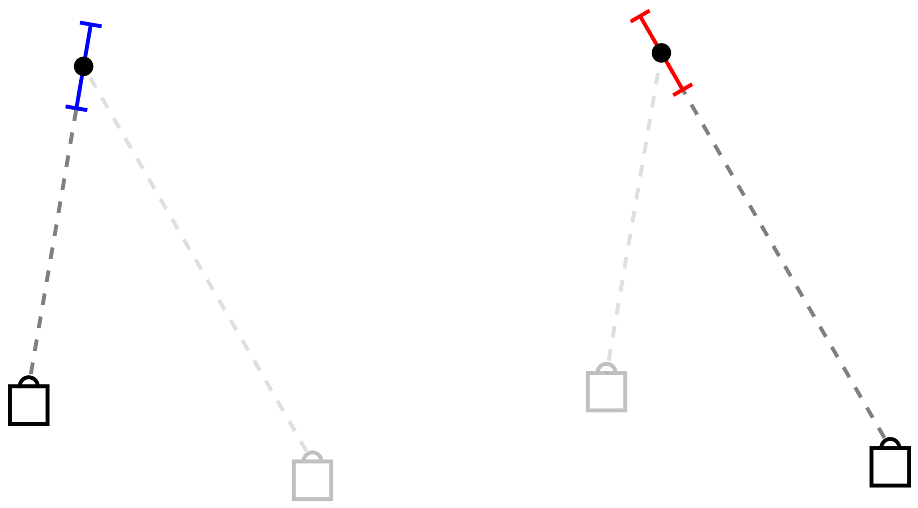
Figure 3.3: Example segmentation of a bin: blue whiskers indicate standard deviation, green areas show extracted segments.

3.3 Measurement Fusion

The fusion process is split into two parts: measurement fusion and confidence evaluation. While in practice these steps are performed simultaneously to avoid redundant calculations, the two concepts are portrayed as separate steps for better understanding. This section will deal exclusively with the fusion of measurements, i.e. generating the output point cloud $\mathcal{O}$ from a series of input clouds $\mathcal{I}_1, \dots, \mathcal{I}_k$, while section 3.4 will focus on the computation of the confidence value c for each output point.

3.3.1 Assumptions

Several assumptions are made about the sensors and the overall setup. As indicated in the previous sections, inaccuracies of the measurements are assumed to be zero-mean gaussian white noise. All participating sensors should be as close together as possible. This way, perspective differences causing a slightly rotated interpretation of the standard deviation can be neglected (Figure 3.4). As the incoming point clouds have automatically been transformed into a common coordinate system during data preparation, a bad setup will likely not affect the overall result but drastically decrease the accuracy. Further it is assumed that all measurements are statistically independent regarding time, proximity and sensor. The algorithm is agnostic of time and only processes single snapshots of the scene, no additional data from previous time steps is processed.



(a) Detection as seen from the original sensor. The blue bar visualizes standard deviation.

(b) Detection as seen from the virtual sensor. The red bar visualizes the rotated interpretation of the standard deviation.

Figure 3.4: Perspective difference changes interpretation of the measurements accuracy

3.3.2 Fusion

The fusion process emulates a virtual sensor, casting rays into the scene from the point of origin. The rays are distributed evenly over the field of view of the virtual sensor and each ray is processed independently.

First, the angle of the ray is determined. For each input point cloud, the bin representing that angle is selected. The segments contained in all the selected bins are then sorted and collected into disjunct groups, similar to segmentation step: Groups are formed by overlapping segments, each segment that does not overlap with any other is put into a separate group containing only itself. Two segments S_i and S_j are considered to overlap when the condition

$$d(S_{i'}) + r_{i'} + b \cdot \sigma(S_{i'}) \leq d(S_{j'}) - b \cdot \sigma(S_{j'})$$

is fulfilled for either $i' = i, j' = j$ or $i' = j, j' = i$. An example is shown in Figure 3.5.

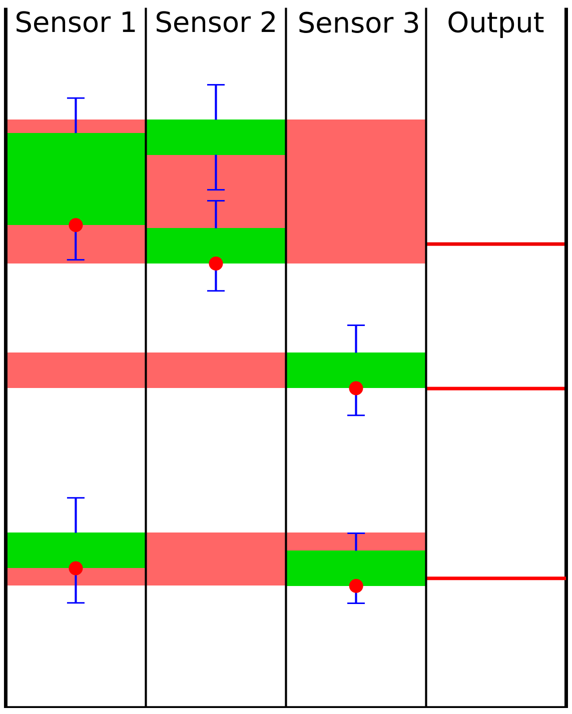


Figure 3.5: Grouping and fusion. Green areas are the extracted segments, blue whiskers indicate standard deviations inherited from the first point, red areas show the grouping. The red dots represent the points included in the fusion.

In the next step, the segments belonging to a group are fused into a single point. While it is possible that multiple segments from the same bin are grouped together, in this case only the segment which is closest is considered for the fusion and all other segments in the same group from that bin are ignored. The representatives are fused using the the weighted arithmetic mean, where the weights are determined by the respective standard deviation:

$$P^{(\mathcal{O})} = \sum_{i=1}^{k} \frac{(\sigma(P^{(\mathcal{I}_i)}))^{-2} \cdot P^{(\mathcal{I}_i)}}{\sigma^*}, \sigma^* = \sum_{i=1}^{k} \frac{1}{(\sigma(P^{(\mathcal{I}_i)}))^2}$$

where σ^* is used for normalization. The resulting point $P^{(\mathcal{O})}$ is then added to the output cloud $\mathcal{O}$. Note that the fusion calculates the mean over all three dimensions, as the exact location is later required by the confidence evaluation. This procedure is repeated until all groups are processed and the algorithm proceeds with the next ray.

3.4 Confidence Evaluation

After computing an output point, it is tagged with a confidence value. This value expresses the belief, whether a point represents an object or is the result of sensor noise (false positive). The confidence value is relevant if a point only receives support from a fraction of the sensors, i.e. the sensors disagree whether there is an obstacle or not. Note that this is quite different from the *accuracy* of a measurement, as now the subject of interest is not *where* the object is, but whether it actually exists.

This can be accomplished by exploiting the complementary features of the deployed sensors: While both stereo camera and lidar are optical sensors, they are based on entirely different concepts and in turn have different strengths and weaknesses. The main difference in performance can be observed in excessive brightness or low illumination. The visual information from the camera image can be used to define the local brightness. Assumptions on the environment can be drawn from the brightness value and possible reasons for sensor failure can be inferred. Low values often indicate surfaces with low reflectivity, potentially posing a problem for the lidar sensor. High values can be caused by overexposure, compromising the reliability of the stereo system.

3.4.1 Concept

Let $O \in \{0, 1\}$ be the event, that an object is in the path of the measurement beam and should in turn generate a detection. Its true value is unknown and must therefore be estimated given the detection $D_i \in \{0, 1\}$ and local lighting conditions $L_i \in \{dark, normal, bright\}$ from sensor i. In favor of a clean notation, an auxiliary variable, the *sensor trust* T_i is introduced.

The sensor trust expresses the trustworthiness of the sensors assessment whether an object is present or not. In mathematical terms

$$T_i = p(O = D_i) = \sum_{l,d} p(O = d \mid L_i = l, D_i = d) \cdot p(L_i = l) \cdot p(D_i = d)$$

where $p(L_i = l)$ and $p(D_i = d)$ only take binary values according to the measurement of the respective sensor, thus the formula decays to

$$T_i = p(O = d \mid L_i = l, D_i = d)$$

The result can then be translated directly into the quantity of interest, namely

$$p_i(O) = \begin{cases} T_i & if D_i = 1 \\ 1 - T_i & if D_i = 0 \end{cases}$$

The resulting set of equations can be represented as the bayesian network depicted in Figure 3.6. The network is purely feed forward, and only requires tuning of the Trust node, where sensor specific weights are required.

The brightness is determined by projecting the point coordinates onto the image plane. For each sensor that provides a detection, the confidence is evaluated for the respective point, while sensors reporting free space use the fused point instead. The total confidence is then retrieved by calculating the product of odds from each sensor

$$C = \prod_i \frac{p_i(O)}{1 - p_i(O)}$$

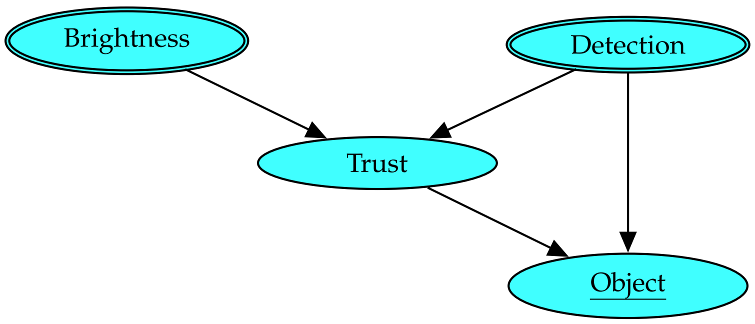


Figure 3.6: Bayesian Network representing computation of the confidence value

Chapter 4

Implementation

This chapter will illustrate how the concepts presented previously can be implemented. At first a short introduction of the ROS framework will be given and a number of relevant tools will be presented. Afterwards, the preparation of input data will be explained, followed by an outline of the configuration which gives an impression of the general program structure.

4.1 ROS Framework

4.1.1 Overview

The Robot Operating System (ROS)[1] is an open source framework designed for application in the wide field of robotics. While the name suggests a complete operating system, it is rather a collection of useful tools with the goal to provide a robust and generic framework that makes the design and operation of robots easier and encourages collaboration of research teams. A variety of modules for data acquisition (sensor drivers), processing, visualization, and actuator control is already included.

The framework is mainly written in C++ and Python but also features language bindings for Lisp, Java and JavaScript. The software modules are organized in packages, which may have further dependencies on other packages, and can be built and managed with the included CMake-based[2] build tool. Each package may contain several programs, called *nodes*, which run independently and are able to communicate with each other via ROS messages, an extension of the Message Passing Interface (MPI). The communication is organized in *topics*, to or from which individual nodes can *publish* or receive messages. A central master node organizes the communication and maintains an overview over active nodes and topics on different machines in the network.

[1]`http://www.ros.org`
[2]`https://cmake.org`

4.1.2 Used modules

Point Cloud Library

The Point Cloud Library (PCL)[3] [25] project is a collection of different tools and algorithms useful for the processing 2D or 3D point clouds. While the original project is stand-alone, an integration into the ROS framework exists, which supplies the necessary encapsulation to send and receive point clouds between nodes.

Point clouds are implemented as lists of individual points and contain a header with a time stamp and a frame ID. Each point consists of the mandatory x, y and z coordinates, but may hold additional information, for example color, brightness, or orientation of a point, depending on the application. The point cloud library implicitly handles conversion between different point types and provides easy means to transform point clouds between different frames (coordinate systems).

tf2

The transform library 2 [26] is part of ROS and provides spatial transformations between local coordinate systems, called *frames of reference*, or frames in short. The transformations are organized in a tree structure representing the setup of different components, for example sensors. Among the rotation and translation, each dataset includes a time stamp which allows rewinding of time dependent transformations, either relative to each other or relative to fixed reference point.

OpenCV

The Open Source Computer Vision Library (OpenCV)[4] is an image processing framework supplying a wide variety of image analysis and manipulation algorithms. While written in C++ to optimize performance, APIs for Python and Java exist as well. Like the point cloud library, it is integrated into ROS providing the standard representation of images. The `cv_bridge` package provides the necessary conversions between OpenCV images and ROS messages.

Visualization

ROS includes several useful tools for visualization of the processed data. For one, `rqt` provides a Qt[5] based user interface that can be used to display images, various graphs, and introspect messages and topics. The second tool, `rviz`, is based on the Ogre3D[6] engine and allows data

[3]http://www.pointclouds.org
[4]https://opencv.org
[5]https://www.qt.io
[6]https://www.ogre3d.org

visualization and introspection in three dimensional space. Both tools include plug-ins that support most of the standard ROS messages, including OpenCV images (`rqt`) and PCL point clouds (`rviz`) out of the box.

Storage

The ROS framework uses a special file format to store recordings of messages, the *rosbag*. A rosbag can either be created by live recording messages between nodes or directly by storing the data via the rosbag API. The bag can then be replayed in real-time, for example to reconstruct an experiment or change parameters without having to repeat the entire test run.

4.2 Framework Architecture

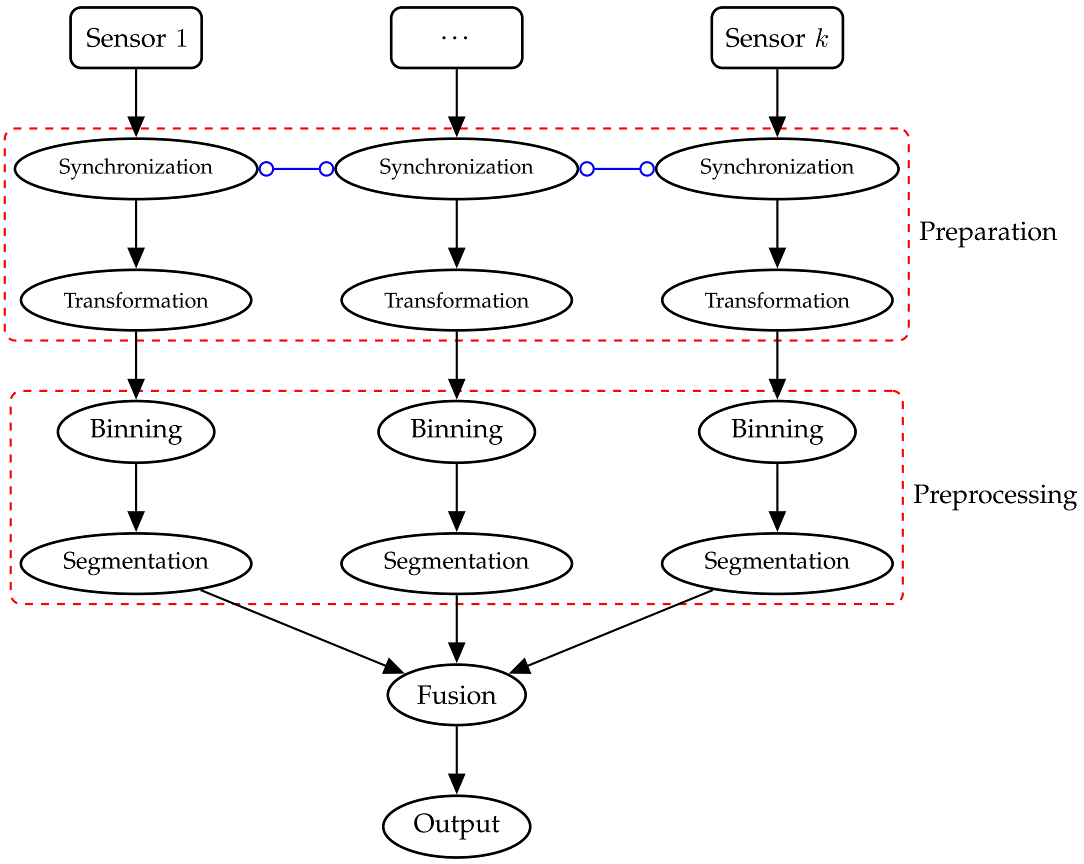


Figure 4.1: Processing pipeline for k different sensors. Synchronization is performed across all data streams.

4.2.1 General Aspects

The entire processing pipeline (Figure 4.1) is implemented in a single ROS node. It is implemented in C++ for maximum performance to allow processing of large point clouds even on limited hardware.

The program is designed to be modular and allows easy reconfiguration without altering the source code. The sensor setup and fusion parameters are read from an XML configuration file and can be changed during runtime. Most core concepts like sensor models and the fusion algorithm are supplied by interfaces. By using factories, the system is easily extensible and hardly any changes are necessary to implement and test different approaches.

As the preprocessing stage is tightly coupled with the fusion, the concepts were already discussed in the previous theoretical chapter and will not be covered further. The focus will now be laid on the data preparation, which is more specific to the implementation.

4.2.2 Synchronization

When using a setup of heterogeneous sensors, the first problem is that data usually arrives not only asynchronously, but often even at different rates and therefore at entirely different points in time. Point clouds from different time points represent different versions of reality, potentially corrupting proper fusion. The fusion process should therefore be triggered only whenever the available data is as homogeneous as possible. Further processing is only done for most recent data received from a sensor, thus potentially omitting data if a sensor produces point clouds at a higher rate than the fusion process. This process of choosing appropriate point clouds is called synchronization.

The implementation of the fusion framework features different possibilities for synchronization: The fusion process can either be triggered every time one of the sensors receives new data, or only when all of the sensors provide updates. For already synchronized data streams, synchronization can instead be based on a sequence number. In this case the fusion is triggered when the counters of all sensors match. Additionally, master/slave configurations (one sensor is responsible for triggering) or periodic updates at a constant rate are possible.

Unless the sensors are synchronized to begin with, small time differences will remain even after synchronization. If the vehicle has means to detect its own movement, for example by odometry, IMU or GPS, the drift resulting from motion can be automatically corrected as part of the transformation step.

4.2.3 Transformation

After synchronization, the data must be transformed into a common coordinate system, called the *target frame*. Usually the point clouds generated from a sensor are in the sensors own local coordinate system, the *source frame*. The need for transformation arises from two factors: Firstly, axes may have a different meaning depending on the sensor, secondly the different sensors are usually placed at different locations.

The most common notation uses x as the forward direction, with y left and z up, which is also used for the target frame. In the field of graphics processing however, the z axis is used as "distance into the image" (thus forward), while the y axis points down and the x axis right.

The appropriate transformations between sensor coordinate systems should be determined by careful calibration of the sensors against each other. The transformations are then announced in proper ROS messages so that they can be automatically managed by the tf library. Since the point clouds include a header containing the frame ID, the correct transformations can be determined automatically and no further manual configuration is necessary. As mentioned above, the tf library can be used to automatically account for movement of the vehicle between different points in time, If a fixed coordinate frame (for example a global frame supplied by GPS) is provided

It is assumed that all sensors use the same units of measurement and no additional scaling is necessary. This reduces the transformation to a rotation and a translation. Since transformation of the input point clouds potentially changes the distance of each point to the origin, the (distance dependent) sensor model is applied to each point *before* the transformation. The resulting standard deviation is stored separately and made available for later processing steps. However, the transformation may introduce a certain inaccuracy in the sensor models assessment, as the interpretation of the standard deviation is changed by the perspective difference, as depicted before in Figure 3.4.

4.2.4 Architecture

The configuration file contains a complete description of the sensors and the setup used and therefore represents the architecture of the software. The file uses XML and is parsed using the TinyXML[7] library, which is bundled with the ROS framework. The file is split into three sections, the sensor, the stream, and the fusion configuration.

The sensor section contains a list of sensors, which may or may not be used in the setup. Each configured sensor must feature a unique name which serves as an identifier, a description of the specific measurement model and a binning configuration. The parameters required for the measurement model depend on the function used. Additionally, two more XML nodes can

[7] http://www.grinninglizard.com/tinyxml/

describe the parameters of the bayesian network evaluating sensor trust. For better readability, this configuration is split into separate nodes specifying values for detection and no detection each.

The stream section configures the individual streams. A stream provides the link between the data received from the sensor and the sensor properties described previously. This way, several sensors of the same type can be used without the need to configure every single one of them. The configuration is rather simple, only the name of the sensor and the topic to which the point clouds are published, are required. While the stream is also responsible for pose transformations, the frame IDs are embedded in the point cloud messages and no manual configuration is needed. Optionally, each stream can be given a name. This is purely cosmetic to produce more readable logging and output.

The last section holds general information necessary for preprocessing and the parameters for the fusion algorithm. This includes the trigger for synchronization, topic names for output and video stream and the frame ID of the virtual sensor. The fusion algorithm is provided as a module and can therefore be exchanged, however, only the bayesian network algorithm described in this thesis is implemented. It requires the camera parameters (focal length and image center), binning specification analogous to the sensor description and brightness thresholds. These thresholds determine which values are considered as dark, bright and normal.

Chapter 5

Experiments

This chapter will evaluate the framework described in the previous chapter using real life data. At first the data set will be introduced along with its preparation for use with the fusion framework. Afterwards, three scenes containing everyday scenarios featuring sensor disagreement are presented, which highlight how this conflict can be resolved by the algorithm. In the end a short analysis of the computational performance is given.

5.1 The KITTI dataset

5.1.1 Setup

The data set used for evaluation of the algorithm is taken from the KITTI project [27]. It was originally intended as a platform to develop and benchmark stereo and tracking algorithms and is available free of charge for non-commercial projects. While several data sets with tagged objects and track-lets are included, only the raw data [28] will be used. The test data consists of a series of short recordings (typically less than a minute) of the test vehicle driving through different scenes in the city of Karlsruhe (Germany) and surroundings.

The test setup includes both a color and gray scale stereo system, a rotating 360° laser scanner, and a GPS module. The sensors are calibrated against each other and the resulting coordinate transformations are supplied in a separate file. The raw data sets used for testing feature synchronized snapshots at a constant rate of 10 Hz. The camera images have been rectified and can be fed directly into stereo algorithms. The lidar sensor provides a 360° view with 64 layers and was originally intended to supply ground truth measurements as a reference for the stereo algorithms. It has a range of up to 120 m with an constant accuracy of 1.5 cm [29].

5.1.2 Data Import

As the KITTI dataset was intended for testing stereo algorithms, it contains no point clouds from the stereo system. Therefore the point clouds required for the fusion framework have to be computed beforehand. To allow an easier replay of the data sets, the data is additionally imported into a rosbag, the native data format of ROS. The import is performed using a modified version of the `kitti2bag` tool [1], a python script that stores the images, point clouds and transformations as time-stamped messages into a single file.

While the original tool maps the data set directly to the rosbag, some additional processing steps are added to reduce file size and machine load on replay. As the fusion framework expects point clouds, the images from the left and right camera are processed into a disparity map using the Semi-Global Blockmatching (SGBM) algorithm [30] supplied by OpenCV. The SGBM parameters used can be found in the Appendix. The disparity map is computed for the left and right image as dominant view and then smoothed using a weighted linear least squares filter [31]. The resulting disparity map is then converted into a point cloud using OpenCVs `reprojectImageTo3D` function.

To reduce file size and required band width, both the point cloud from the lidar and the one generated from the stereo images are filtered. As the fusion algorithm only operates on a plane, a horizontal slice, from 0.9 m to 1.7 m above ground is cut out, discarding all points above or below. Especially the numerous points below 0.9 m would lead to inconclusive fusion results, as many small objects such as curbstones would be included. Additionally, all points behind the vehicle (with a negative x coordinate) are removed from the lidar point cloud.

The file then contains the two filtered point clouds and the camera image required for the fusion algorithm, along with the respective frame IDs and coordinate transformations. For debugging purposes, the disparity map is exported as well, yielding a total bandwidth of about 25 MiB/s.

5.1.3 Configuration

The fusion framework has been configured for the use with the KITTI dataset. Only the most important parts of the configuration are covered here, the complete XML file can be found in the Appendix.

The field of view for all sensors, including the virtual sensor emulated by the fusion algorithm, ranges from -40° to 40. It is divided into 500 bins, resulting in an angular resolution of 0.16°, which spans two rays of the lidar sensor and about three pixels of the disparity map.

The trust tables parameterizing the Bayesian network performing confidence evaluation are displayed in Figure 5.1. The values are chosen by means of educated guess, based on general

[1]https://github.com/tomas789/kitti2bag

characteristics of the sensors. For any applications in real life situations, the parameters must be revised and determined by appropriate calibration.

T_{Stereo}	Dark	Normal	Bright
Detection	0.85	0.95	0.8
No Detection	0.85	0.9	0.75

(a) Trust table for Stereo

T_{Lidar}	Dark	Normal	Bright
Detection	0.99	0.99	0.95
No Detection	0.7	0.9	0.9

(b) Trust table for Lidar

Figure 5.1: Sensor trust under different conditions

5.2 Results

To present results for the testing, four representative scenes have been selected, three of them depicting common scenarios where sensor disagreement occurs, i.e. one of the sensors yields a detection while the other reports free space.

Figures 5.2 to 5.5 show an example scene without conflict and provide comprehensive descriptions of the different visualization methods. For better understing of the different images, some landmarks will be tagged, like the car ① or the house corner ② in this first example.

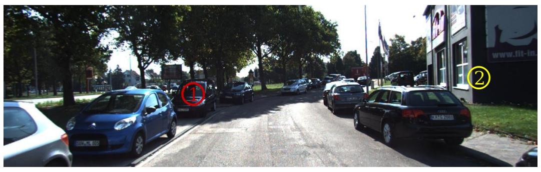


Figure 5.2: Camera image of the scene.

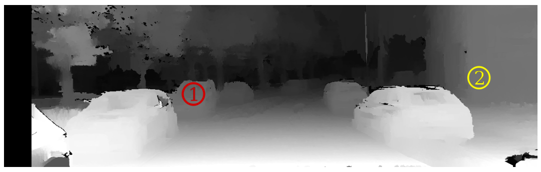


Figure 5.3: Disparity map of the scene computed by the stereo algorithm. Bright parts represent high disparity and thus low depth.

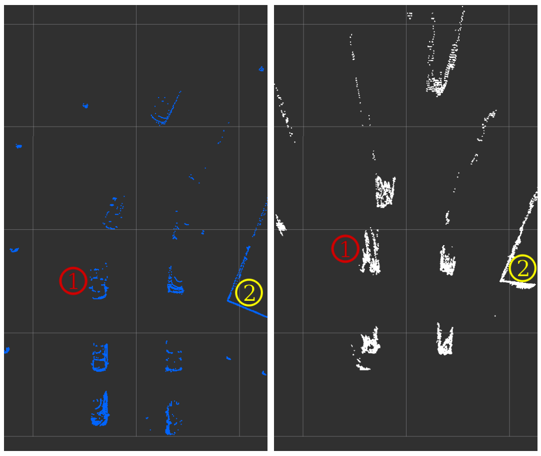


Figure 5.4: Bird's eye view of the scene as perceived by the Lidar (left, blue) and Stereo (right, white) sensors. Each dot represents a single measurement. The parked cars and house corner (②) can be identified easily.

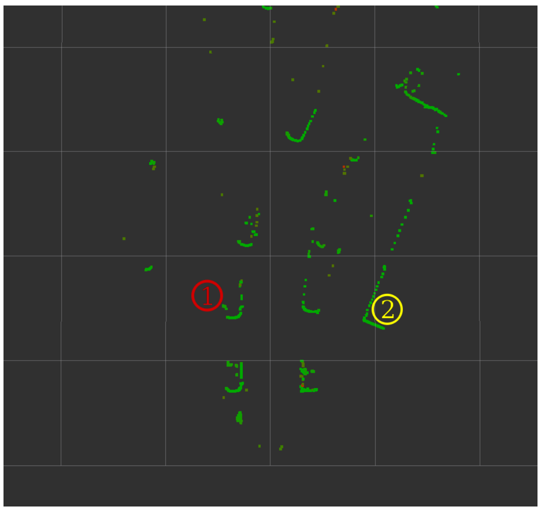
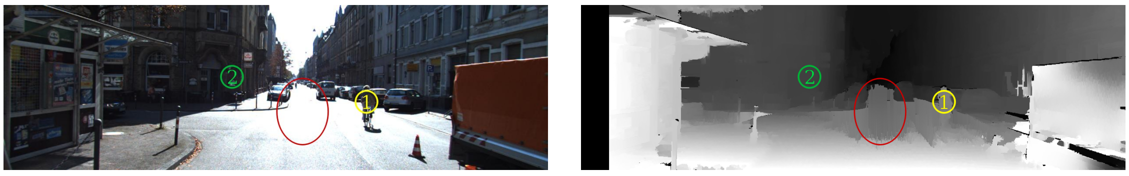


Figure 5.5: Fused point cloud as output by the algorithm. The confidence of each point is color coded. The color ranges from red (low confidence) to green (high confidence).

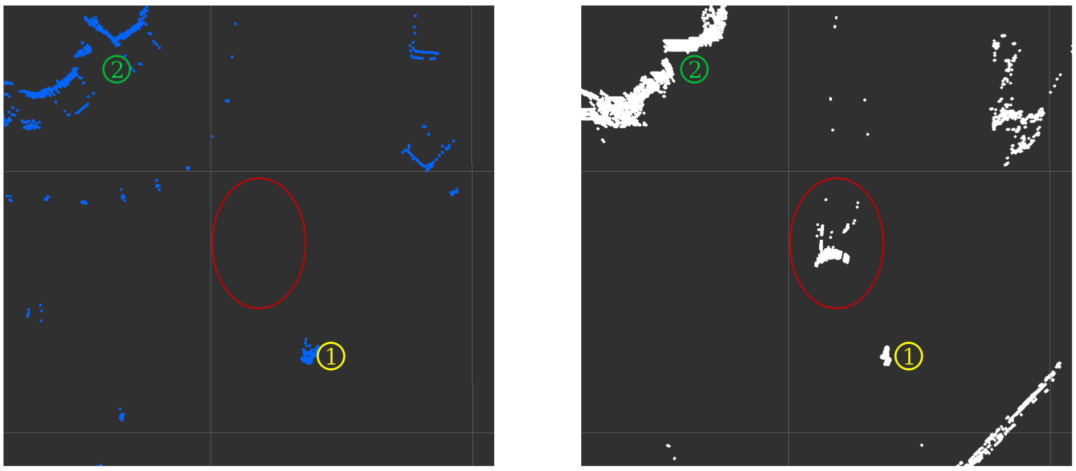
5.2.1 Stereo vision impaired by overexposure



(a) Overexposure of the scene.

(b) The bright spot is recognized as an object.

Figure 5.6: Reflection of the sun blinding the camera



(a) Lidar reports free space in the questionable are.

(b) Stereo detects an object.

Figure 5.7: Point clouds generated by lidar and stereo camera

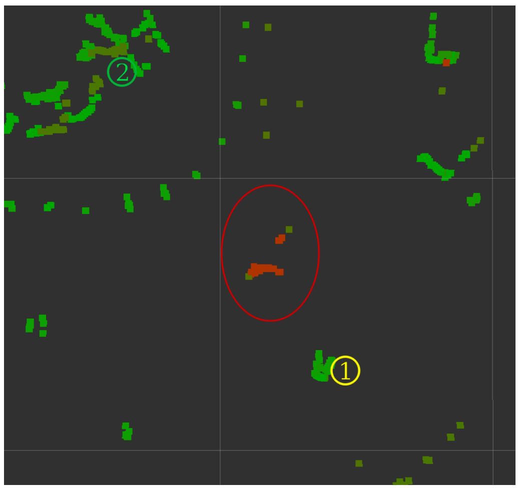


Figure 5.8: The artifact is classified as noise by the algorithm

In this scene[2], the sun creates a bright reflection on the street ahead (Figure 5.6a). The stereo algorithm interprets the bright spot as an object (Figure 5.6b), which appears to be standing on the road next to a cyclist (①), approximately 15 m to 25 m in front of the car. The false positive persists for about 5 seconds, moving slightly back and forward, while the car slowly drives in its direction. As would be expected, the area is correctly classified as free space by the lidar sensor (Figure 5.7a).

The fused point cloud includes the false positive tagged with a confidence value of roughly 31%, therefore recognizing that the detection most probably results from noise (Figure 5.8). This is due to the fact that through the image brightness the area is interpreted as a highly reflective surface, which should thus be detected by the lidar sensor. Additionally, the excessive brightness decreases the trustworthiness of detections from the stereo camera.

Without this classification as noises, an autonomous system would most likely treat the artifact as an actual object and would abruptly apply the brakes or try to avoid the obstacle by steering to the side. This erratic behavior would potentially endanger the passengers or other road users.

5.2.2 Black car missed by Lidar

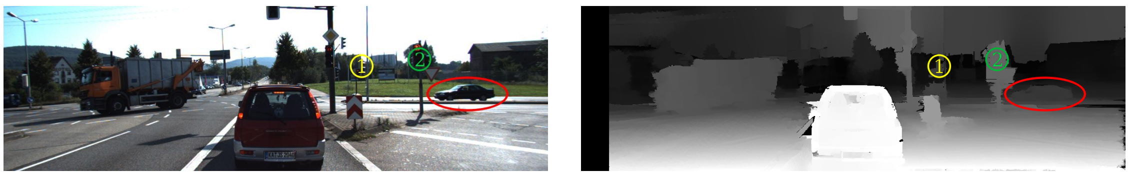
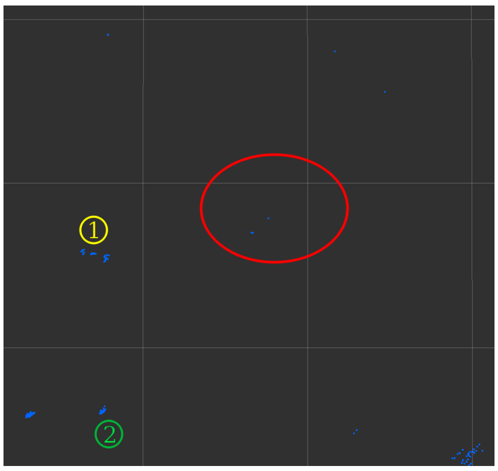
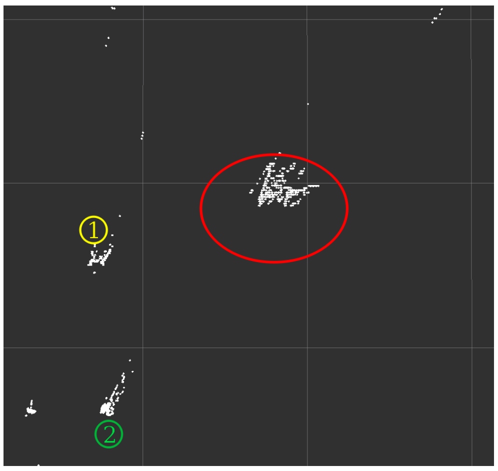


Figure 5.9: Crossroads featuring a black car

[2]Dataset: 2011_09_26, Drive 0005, Snapshot time 11.373 s



(a) Lidar hardly produces any measurements



(b) Stereo detects the car without problems

Figure 5.10: Point clouds generated by lidar and stereo camera.

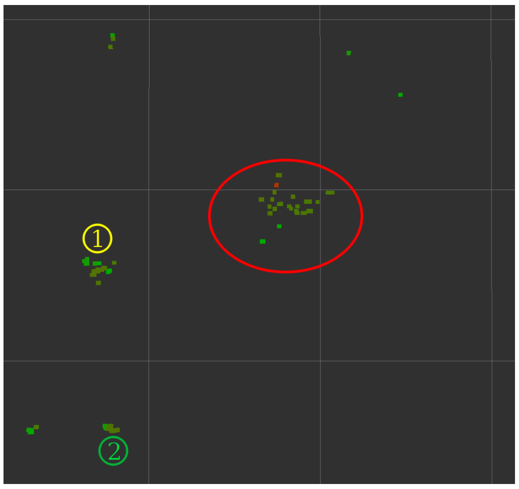


Figure 5.11: The car is identified with high confidence.

The second example [3] features a black car waiting at a traffic light (①), which is almost entirely missed by the Lidar system (Figures 5.9 and 5.10). The stereo camera however has no trouble detecting the car.

Despite the generally high reliability of the lidar sensor, fused points representing the car receive a relatively high confidence of almost 71% (Figure 5.11). This indicates that the algorithm correctly recognized the failure of the lidar sensor caused by the dark surface.

[3]Dataset: 2011_09_26, Drive 0057, Snapshot time 7.88 s

5.2.3 Pedestrian missed by stereo camera

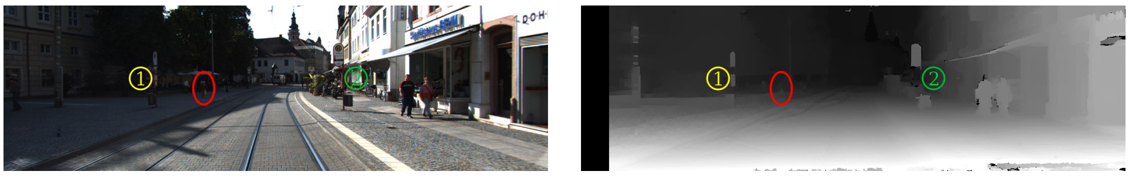
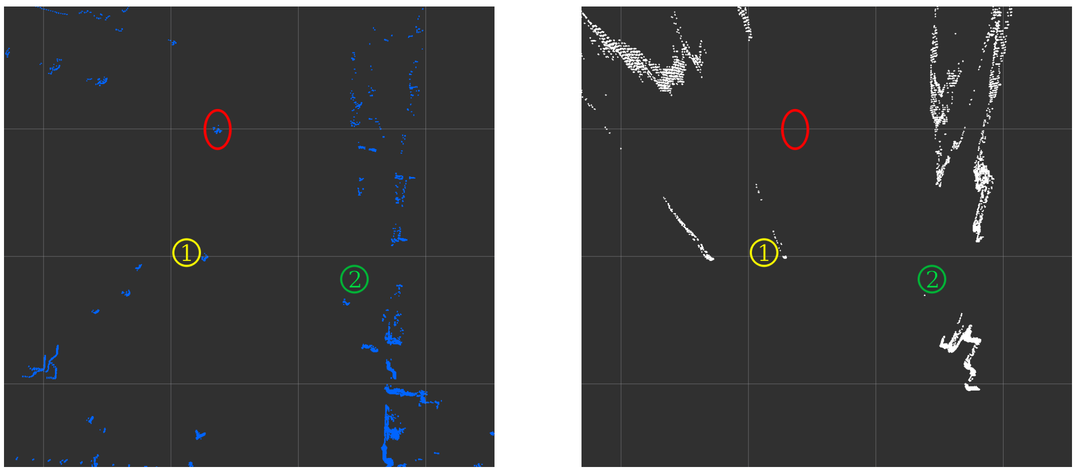


Figure 5.12: A street with trolley line tracks and several pedestrians on both sides.



(a) Lidar reliably detects the pedestrian. (b) Stereo produces no measurements.

Figure 5.13: Point clouds generated by lidar and stereo camera.

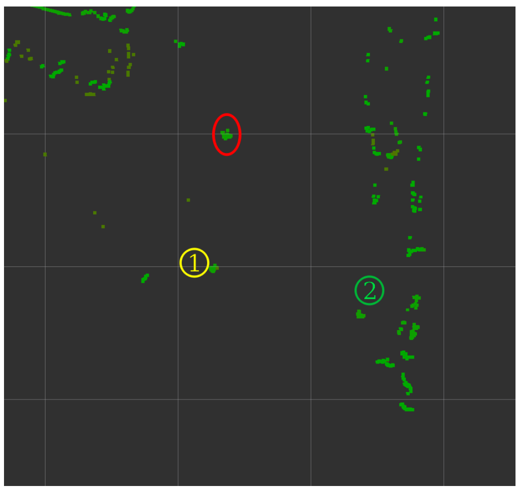


Figure 5.14: The car is identified with high confidence.

In the last example [4], a pedestrian can be seen possibly waiting to cross the road (Figure 5.12). The person is detected reliably by the lidar from the beginning of the recording, when still about 40 m away. The stereo system on the contrary does not recognize the pedestrian at this distance (Figure 5.13).

The area of the image where the person is standing is perceived as rather dark due to the shadow. Even though these conditions impact reliability of the lidar sensor, they only cause false negatives. As the sensor provides detections despite darkness, these are marked as trustworthy. Additionally, the low illumination reduces the trustworthiness of the stereo system, resulting in an overall confidence value of 95% (Figure 5.14).

5.2.4 Computational performance

The main source of the computational complexity arises from point sorting in the preprocessing phase, with an average runtime of $\mathcal{O}(n \log n)$, all other steps are linear in both the number of sensors and points.

A practical test conducted on a system with a 2.7 GHz dual core processor, resulted in an average processor load of about 43%, consuming 35.8 s CPU time during the 82.8 s test run[5]. On average, the algorithm processed over 400,000 points per second (about 100,000 lidar and 300,000 stereo points per second), while generating an output of around 12,000 points per second. This denotes a significant reduction by more than 97%, without sacrificing important information. The average time required for preprocessing and fusion amounts to 35 ms, which is mostly caused by the transformation of incoming point clouds.

Even on weaker systems, the framework can already provide real time performance. The tested implementation only runs in a single thread, optimization for multiple processors could improve the performance further, especially the costly transformation step can be easily parallelized.

[4]Dataset: 2011_09_26, Drive 0091, Snapshot time 2.29 s
[5]Dataset: 2011_09_26, Drive 0022

Chapter 6

Conclusion

The goal of this thesis was to design and implement a framework that can be used to fuse multiple point clouds from different sensors into a single point cloud. A large focus was put on developing a strategy that can handle sensor disagreement intelligently. A software module was developed, which is based on the popular ROS ecosystem and uses standardized interfaces to allow easy integration with existing software.

The software was tested with real life data from the KITTI dataset. Several situations were found, where one of the sensors failed, either by false positives or false negatives. In all of these situations the conflict could be resolved with the use of confidence values computed by a Bayesian network. The algorithm provides high throughput and significant data reduction, making it viable for real time applications.

6.1 Limitations

The proposed framework relies largely on visual hints from a camera. In particular, it is assumed that the image brightness provides a good representation of a materials (diffusive) reflectivity. While in many cases this assumption is sufficiently accurate, it may fail for various reasons: Glossy surfaces like chrome are usually perceived as bright, but the lack of diffusive reflectance may still lead to failure of lidar sensors. Especially on sunny days the presence of dark shadows along with well lit areas can cause wrong estimates, as objects in the shadow will always be perceived as dark and thus unreflective.

A related drawback is that the concept does not work well at night, because all areas are dark and no conclusions about the surface properties can be drawn. Additionally, the stereo system is rendered useless in this case, and there would be no benefit of fusion at all. This problem can be avoided by the use of Active Gated Imaging Systems (AGIS) [32], where the surroundings are illuminated by short infrared light pulses and allow the use of vision based methods even at night.

6.2 Outlook

In the experiments conducted in the previous chapter, the parametrization used for confidence evaluation was chosen "at will". While the possibility to incorporate expert knowledge is certainly a strength of Bayesian networks, the parameters for real world applications should be determined either by appropriate experiments or by directly training the network.

The trustworthiness of a sensor is solely based on brightness of the image area. While this already provides sound evidence for estimating the sensors reliability, other factors could be incorporated. Most notably the distance, as it greatly influences the reliability of all sensors. Not only does the probability of detection deteriorate with increasing distance, but a measurement from one sensor could be entirely out of range for the other. Especially in the latter case, a missing detection should have no, or only very little, negative influence on the total confidence.

Further improvements could be achieved by replacing simple image brightness with more sophisticated image analysis. This could supply a far better estimation of the difficulties a stereo algorithm faces in certain parts of the scene.

Another option would be the incorporation of entirely different ranging sensors. In theory, any ranging sensor can be integrated into the framework. To achieve sensible confidence ratings for sensors not based on optics, the relevant factors influencing these sensors would have to be included. Due to the use of the Bayesian network, these changes and additions can easily be incorporated into the existing algorithm.

Lastly, the framework performs fusion on a horizontal plane, which is sufficient for many applications using two dimensional lidar sensors. Since most of the concepts described in the Design chapter are independent of the dimensionality, the extension to a third dimension can be achieved easily.

All of these suggested improvements do not only serve as slight enhancements, though. When applied to autonomous driving vehicles on public streets, further development of the algorithm will prove necessary to achieve justifiable street safety. Nevertheless this thesis sets solid groundwork for further research and on point cloud fusion using Bayesian networks, bringing the dream of self driving cars one step further on it's journey from fiction to science.

Bibliography

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, ser. Intelligent robotics and autonomous agents. Cambridge, Mass: MIT Press, 2005.

[2] S. Singh, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," Publication DOT HS 812 115, 2015.

[3] F. M. Favarò, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, "Examining accident reports involving autonomous vehicles in california," *PLoS ONE*, vol. 12, no. 9, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5607180/

[4] G. Corfield, "Tesla death smash probe: Neither driver nor autopilot saw the truck." [Online]. Available: https://www.theregister.co.uk/2017/06/20/tesla_death_crash_accident_report_ntsb/

[5] S. Maki and A. Sage, "Self-driving Uber car kills Arizona woman crossing street," *Reuters*, Mar. 2018. [Online]. Available: https://www.reuters.com/article/us-autos-selfdriving-uber/woman-dies-in-arizona-after-being-hit-by-uber-self-driving-car-idUSKBN1GV296

[6] M. Bergen and E. Newcomer, "Uber halts autonomous car tests after fatal crash in arizona," *Bloomberg.com*, 2018. [Online]. Available: https://www.bloomberg.com/news/articles/2018-03-19/uber-autonomous-car-involved-in-fatal-crash-in-arizona

[7] W. Elmenreich, "An introduction to sensor fusion," *Vienna University of Technology, Austria*, vol. 502, p. 29, 2002.

[8] D. L. Hall, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[9] P. Moghadam, W. S. Wijesoma, and D. J. Feng, "Improving path planning and mapping based on stereo vision and lidar." IEEE, 2008, pp. 384–389. [Online]. Available: http://ieeexplore.ieee.org/document/4795550/

[10] Y. Yang, Z. Koppanyi, and C. K. Toth, "Stereo image point cloud and lidar point cloud fusion for the 3d street mapping," 2017.

[11] A. Myronenko and X. Song, “Point-set registration: Coherent point drift,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2262–2275, 2010. [Online]. Available: http://arxiv.org/abs/0905.2635

[12] H. Baltzakis, A. Argyros, and P. Trahanias, “Fusion of laser and visual data for robot motion planning and collision avoidance,” *Machine Vision and Applications*, vol. 15, no. 2, pp. 92–100, 2003.

[13] S. T. Barnard and M. A. Fischler, “Computational stereo,” *ACM Computing Surveys*, vol. 14, no. 4, pp. 553–572, 1982. [Online]. Available: http://portal.acm.org/citation.cfm?doid=356893.356896

[14] H. E. Burton, “The optics of euclid,” *Journal of the Optical Society of America*, vol. 35, no. 5, p. 357, 1945. [Online]. Available: https://www.osapublishing.org/abstract.cfm?URI=josa-35-5-357

[15] D. Scharstein, R. Szeliski, and R. Zabih, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.” IEEE Comput. Soc, 2001, pp. 131–140. [Online]. Available: http://ieeexplore.ieee.org/document/988771/

[16] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[17] L. Matthies, “Toward stochastic modeling of obstacle detectability in passive stereo range imagery.” IEEE Comput. Soc. Press, 1992, pp. 765–768. [Online]. Available: http://ieeexplore.ieee.org/document/223178/

[18] M. Hebert and E. Krotkov, “3d measurements from imaging laser radars: how good are they?” *Image and vision computing*, vol. 10, no. 3, p. 9, 1992.

[19] R. H. Rasshofer and K. Gresser, “Automotive radar and lidar systems for next generation driver assistance functions,” *Advances in Radio Science*, vol. 3, pp. 205–209, 2005. [Online]. Available: http://www.adv-radio-sci.net/3/205/2005/

[20] R. G. Brown, P. Y. Hwang *et al.*, *Introduction to random signals and applied Kalman filtering*. Wiley New York, 1992, vol. 3.

[21] H. Nam and B. Han, “Learning multi-domain convolutional neural networks for visual tracking,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016, pp. 4293–4302. [Online]. Available: http://ieeexplore.ieee.org/document/7780834/

[22] J. Sander and J. Beyerer, “Bayesian fusion: Modeling and application.” IEEE, 2013, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6698254/

[23] R. E. Neapolitan, *Learning Bayesian Networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004, vol. 38.

[24] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[25] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)." IEEE, 2011, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/5980567/

[26] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, 2013, pp. 1–6.

[27] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite." IEEE, 2012, pp. 3354–3361. [Online]. Available: http://ieeexplore.ieee.org/document/6248074/

[28] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[29] C. Glennie and D. D. Lichti, "Static calibration and analysis of the velodyne hdl-64e s2 for high accuracy mobile scanning," *Remote Sensing*, vol. 2, no. 12, pp. 1610–1624, 2010. [Online]. Available: http://www.mdpi.com/2072-4292/2/6/1610

[30] 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. [Online]. Available: http://ieeexplore.ieee.org/document/4359315/

[31] D. Min, S. Choi, J. Lu, B. Ham, K. Sohn, and M. N. Do, "Fast global image smoothing based on weighted least squares," *IEEE Transactions on Image Processing*, vol. 23, no. 12, pp. 5638–5653, 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6942220/

[32] Y. Grauer and E. Sonn, "Active gated imaging for automotive safety applications," 2015, p. 94070F. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2078169

Appendix A

Appendix

A.1 Test Configuration

Below, the configuration file used for experiments in chapter 5 is shown.

```
<?xml version="1.0" ?>
<config>
  <sensors>
    <sensor name="Velodyne HDL-64E">
      <model function="gaussian">
        <param std=0.02/>
      </model>
      <binning min=-40 max=40 count=500/>
      <trust-detect dark=0.99 normal=0.99 bright=0.95/>
      <trust-clear  dark=0.7  normal=0.9  bright=0.9 />
    </sensor>

    <sensor name="KITTI Stereo">
      <model function="sqd-gauss">
        <param coeff=0.002/>
      </model>
      <binning min=-40 max=40 count=500/>
      <trust-detect dark=0.85 normal=0.95 bright=0.8/>
      <trust-clear  dark=0.85 normal=0.9  bright=0.75/>
    </sensor>
  </sensors>
  <streams>
    <stream name="Stereo" sensor="KITTI Stereo" topic="/kitti/stereo/pointcloud">
      <selector type="closest"/>
    </stream>

    <stream name="Lidar" sensor="Velodyne HDL-64E" topic="/kitti/velo/pointcloud">
      <selector type="closest"/>
    </stream>
  </streams>

  <fusion>
    <trigger type="seq"/>
    <output topic="fusion_test"/>
    <frame id="velo_link"/>
    <camera topic="/kitti/stereo/image_rect"/>
    <combiner type="bnet">
      <camera f=721.5377 cx=609.5593 cy=172.854/>
      <binning min=-40 max=40 count=500/>
      <brightness low=0.2 high=0.95/>
    </combiner>
  </fusion>
</config>
```

A.2 SGBM Parameters

The preprocessing of the KITTI dataset includes stereo reconstruction using the SGBM algorithm. The parameters chosen are listed below, using the parameter names from the OpenCV Python API:

minDisparity	0
numDisparities	64
preFilterCap	10
P1	972
P2	7776
blockSize	4
mode	StereoSGBM_MODE_SGBM_3WAY