University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

# LaFlector: Passive Tracking based on LiDAR

*Lukas Mueller*
*Zurich, Switzerland*
*Student ID: 15-929-821*

Supervisor: Bruno Rodrigues, Eder Scheid, Prof. Dr. Burkhard Stiller
Date of Submission: March 21, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

# Abstract

Die zuverlässige Messung von Objekten in Zeit und Raum ist die Voraussetzung für viele zukunftsweisende Technologien wie zum Beispiel autonomes Fahren. Es gibt verschiedene Methoden, die auf unterschiedlichen Messinstrumenten basieren. Eine dieser Messmethoden zur Distanzmessung und Objekterfassung ist Light detection and ranging, kurz LiDAR, genannt. So setzt beispielsweise Volvo, bekannt für hohe Fahrzeugsicherheit, bei der nächsten Fahrzeuggeneration auf LiDAR Sensoren [5]. Aber nicht nur Bewegungen von Verkehrsteilnehmern auf der Strasse sind interessant, sondern auch Bewegungen in einem Raum, beispielsweise um die Besucherfrequenz an einem Messestand zu ermitteln.

Diese Bachelorarbeit befasst sich mit dem Entwurf eines Systems zum Tracken von statischen und bewegten Objekten, in diesem Fall Menschen, in einem geschlossenen Raum mit Hilfe eines LiDAR Sensors. Hierzu wird eine Schnittstelle zwischen einem LiDAR Scanner und einer Datenbank entwickelt. In der unabhängigen, nachfolgenden Datenverarbeitung werden die Daten auf Objekte überprüft. Das entwickelte System namens LaFlector kann mehrere Objekte gleichzeitig erkennen, klassifizieren und verfolgen. Die erkannten Objekte werden aufgezeichnet und dem Anwender in einem Kordinatensystem dynamisch angezeigt. Die Auswertung zeigt, dass das System unter Berücksichtigung der Eigenschaften eines LiDAR Scanners Objekte zuverlässig erkennt, klassifiziert und verfolgen kann.

The reliable measurement of objects in time and space is a prerequisite for many future technologies such as autonomous driving. There are various methods based on different measuring instruments. One of these measurement methods for distance measurement and object gathering is called light detection and ranging, or LiDAR for short. For example, Volvo, known for its high vehicle safety, is using LiDAR sensors in its next generation of vehicles [5]. But not only movements of road users are interesting but also movements in a room, for example to determine the visitor frequency at an exhibition.

This bachelor thesis is about the design of a system for tracking static and moving objects, in this case people, in a closed room with the help of a LiDAR sensor. For this purpose an interface between a LiDAR scanner and a database is developed. In the independent, subsequent data processing the data is checked for objects. The developed system called LaFlector can detect, classify and track several objects simultaneously. The detected objects are recorded and dynamically displayed to the user in a coordinate system. The evaluation shows that the system can reliably detect, classify and track objects, taking into account the limitations of a single LiDAR scanner.

# Acknowledgments

I would like to thank the supervisors of this work Bruno Rodrigues, Eder Scheid and Prof. Burkhard Stiller. In particular, I would like to mention Bruno Rodrigues, who supported me with frequent inputs and assistance. I would also like to thank the Communication Systems Group for providing the LiDAR scanner for this thesis.

# Contents

# Chapter 1

# Introduction

It is technically simple to track the way a visitor takes across a website. It is even possible to predict the user experience according to the movement of the mouse cursor [19]. The data gained from this tracking can optimize the sales process and discover new customer needs. These insights further lead to advantages over the company's competitors. Such insights are equally important in physical stores. It is crucial to understand how customers interact with a product or service and which offer attracts the most interest. Understanding the customer builds the foundation of a successful marketing strategy. However, tracking visitor flows is not only relevant for marketing. In public locations, security measures such as emergency routes can be enhanced by analyzing crowd behavior.

There are many approaches and techniques for physically tracking people or devices. Considering the scenario where it is necessary to observe people's flow in a public environment of passers-by. Requiring people to associate their mobile devices with an access point is not a feasible strategy. Thus, a passive approach, which does not require pedestrian interaction is desirable. Within the range of passive tracking approaches, it is possible to use a LiDAR (Light Detection and Ranging) scanner. A LiDAR can be used as a single device or in combination with a second LiDAR or another passive tracking technique. Passive WiFI tracking can be mentioned here as an example. This approach relies on probe request frames sent from devices to detect Access Points.

## 1.1 Motivation

There are many fields of applications for LiDAR systems. A recent example is the built-in LiDAR scanner in the Apple iPad Pro 2020 for augmented-reality purposes. Another example is autonomous driving systems where LiDAR (combined with other sensors) is used to create a map of a vehicle's surroundings. The technical progress in these and many more areas makes LiDAR a highly topical field of research. A reliable tracking system based on LiDAR could potentially be used for on-the-spot tracking at expositions, trade shows, or stores and gain valuable data for exhibitors and companies.

## 1.2 Thesis Goals

This thesis attempts to accomplish the following linked goals:

- **LiDAR Interface**: An interface shall be engineered, which reads the raw data from a LiDAR sensor by using the corresponding software development kit. The data must be recorded in a suitable structure for subsequent data processing.
- **Tracking Technique**: Through research, a technique shall be developed to detect, distinguish and track static and moving objects.
- **Data Output**: The collected data should be in a standardized form consisting of X and Y coordinates, and a timestamp so that the data can be correlated with other data sources. Besides, a dynamic plot shows the running object tracking.

## 1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 presents the background concepts representing the foundation on which the core elements of this thesis are built. Also, Chapter 2 discloses the related work representing specific instances of those concepts embedded in a comparable manner with the thesis's goals. Chapter 3 presents the prototype design based on requirements listed through literature review and aligned with the objectives of the thesis. In the following, Chapter 4 presents implementation details of relevant architectural elements and components, and Chapter 5 presents the prototype's evaluation. Lastly, Chapter 6 concludes this thesis by providing a summary of its achievements, final considerations, and future work.

# Chapter 2

# Background

The Background section gives an overview of the technologies that are needed to build and understand the prototype. As previously described, this work focuses on the LiDAR approach and will not cover passive WLAN tracking. The related work section refers to existing work and reflects the current state of research.

## 2.1 Passive Tracking

Tracking can be divided into passive and active tracking. Active tracking requires participation and configuration of the client for a specific measurement technique. Passive tracking works without adaptations in the environment to be analyzed. This does not mean that signals from devices to be tracked are not necessary. However, these signals occur without pre-configuration. For example, in the case of WiFi tracking, these are outgoing probe requests from the clients [26]. In particular, LiDAR tracking is a passive tracking technique as it works independently of the tracked object's signals.

## 2.2 LiDAR

This section focuses on the layout and resulting technical characteristics of a LiDAR sensor.

### 2.2.1 Background

LiDAR is a technique for distance measurement. It consists of a laser and a sensor to measure the reflections of the environment. The distance can be estimated by measuring the time it takes for the laser light to return and knowing the speed of light.
The basic principle for LiDAR technology came up in the early 1960s right after the foundation was laid with the laser's development in 1960. Like many technical developments,

LiDAR was mainly driven by military institutions for measuring distances and weapon guidance [18].

### 2.2.2 Properties

Each measurement technique has its strengths and weaknesses. This section clarifies these characteristics concerning the LiDAR technique.

#### 2.2.2.1 Strengths

- **Precision:** The precision of LiDAR scanners is relatively high. A Slamtec M1M1 scanner with a range of 20 meters has an accuracy of 2 centimeters [28]. In comparison, the MAC Scavenger as a WiFi-based system achieved a deviation of 1.1 meters at a maximum distance of 10.8 meters under good conditions since RSSI values are not reliable[2].
- **Captures objects, not devices:** Android has enabled MAC address randomization by default since version 10. Apple has extended MAC address randomization with iOS 14. This trend of the recent past is expected to continue, and device fingerprinting will become increasingly difficult. LiDAR is not affected by this development. It moreover allows tracking objects without a smartphone/tablet unlike Bluetooth or WiFi tracking [12] [17].
- **Privacy:** Data protection regulations have become increasingly strict in recent years. Tracking via WiFi or Bluetooth data could be complicated because the MAC address is classified as personal data according to the GDPR [4]. LiDAR tracking is unproblematic as it does not process any information about the tracked object.
- **Range:** LiDAR sensors can provide accurate results over long distances (up to 200m in vehicles) which a radar for example cannot [16]. This is why LiDAR technology is often used in vehicle construction in combination with other detection systems [32].

#### 2.2.2.2 Weaknesses

- **Objects behind objects:** Due to the principle of operation a LiDAR scanner cannot detect objects that are behind other objects. The front object reflects the laser beams completely. This limitation could be compensated by a second scanner which captures the object from a different angle.
- **Large data sets:** Depending on the resolution of the scanner, the aperture angle, and the use case, the number of data points can become very large and many data points derive from static objects that are usually not of interest. The processing of LiDAR data can be more extensive than that of other measurement techniques.
- **Robustness:** The performance of a LiDAR scanner strongly decreases in heavy rain or fog, which can reduce the detection rate of objects by up to 50% [6] [16].

### 2.2.3 Legal situation

Lasers are classified according to the IEC 60825-1 standard. The Slamtec M1M1 scanner used in this thesis contains a Class 1 laser with a power of 28 watts. Even the Slamtec M2M1 model with an extended range of 40 meters is classified as Class 1 [28] [29].
According to the Federal Office of Public Health of Switzerland a Class 1 laser is absolutely safe. Even direct exposure to the laser beam will not cause any damage to the human eye. A warning sign is not necessary [23]. Due to the classification as a Class 1 laser a LiDAR scanner can be used anywhere and at any height (even eye level).

## 2.3 Object Tracking

This part is about the steps that are usually performed to identify and track objects from the unprocessed laser data. Since the scanner used in this thesis is a 2D scanner, no 3D-specific process steps are observed, such as surface matching.

### 2.3.1 Data Read

Due to the design of a 360-degree LiDAR scanner, it is convenient to use the Polar Coordinate System to specify data points. A data point is defined as the distance and angle from a starting point. The starting point corresponds to the position of the scanner. In the first step, these data are converted into Cartesian coordinates. It makes sense to filter out non-valid data points to reduce the number of data points and increase the relative data quality.

### 2.3.2 Segmentation

A data set of a LiDAR scanner always contains all reflecting objects in a room. As a rule, static objects such as walls, windows, or fixed objects are no longer of interest after the initialization phase. The goal of segmentation is to filter out the objects that are in motion. By subtracting the set of static objects from a newly acquired data set, the resulting difference is the set of points associated with objects in motion. If the difference is zero there are no moving objects in the room that could be tracked [27].

### 2.3.3 Classification

If an object is detected that consists of valid data points and do not belong to the static objects' data set, it must be classified. With a 2D scanner, the possibilities are limited by the lack of surface matching. This work uses for the classification the relation between the distance and number of data points and distinguishes between human and non-human. If, for example, an object is close to the LiDAR scanner and covers fewer data points than expected, it will not be identified as a human.

### 2.3.4 Tracking

If an object could be identified as a human, the movement is recorded. A LiDAR scanner has no way to distinguish person 1 from person 2 if they leave the room or exceed the maximum measurement distance. A simple motion analysis should also be possible to maintain the tracking even if two people cross each other.

## 2.4 Related Work

This section covers already existing techniques of tracking using LiDAR scanners. There are very different approaches and options for object detection. To properly classify related work, it is important to know the different ways a LiDAR can be used since classification methods depend on the deployment scenario.

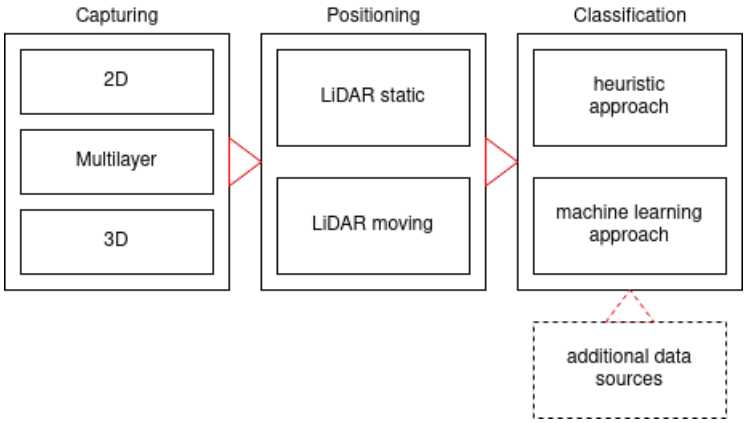



Figure 2.1: LiDAR Categorization

### 2.4.1 LiDAR Categorization

#### 2.4.1.1 Capturing Mode

A 2D LiDAR angled slightly downward can be used on vehicles to distinguish flat and uneven surfaces. From this, in turn, the course of the road can be derived. This is achieved by evaluating the distance between the single measuring points. On roads without any curbs the system achieves at least a score of 92% in detecting the edge of the road. If curbs are present the value drops to 80%. This can be explained by smaller variations in the measured values caused by a sidewalk than in uneven ground [10].

Multilayer capturing requires multiple 2D LiDAR sensors mounted at different heights. The approach thus provides more data than a single 2D scanner but less than 3D LiDAR. This allows an object measured in a particular layer to be verified with the other layers in terms of tracking. In a 50 second test run in which the LiDAR was mounted on a

car driving through a street, the pedestrian detection rate increased from 70.5% with one layer to 91.6% with 4 layers [7].

A 3D LiDAR is characterized by the fact that it can capture multiple angles, thus providing significantly more data points. The points distributed over different heights also allow effective surface matching. This is valuable data for classification. For example if an object is completely flat over its entire height it can be eliminated that the object is human. In an experiment in which two people walk through a room and hide behind obstacles and come out again, the false-negative rate was halved from 30 frames to 15 frames using surface matching for a total of 696 frames [27].

#### 2.4.1.2 Positioning

Basically, the use cases can be divided into two areas. There are applications where the LiDAR is in motion and others where the LiDAR remains statically in place.

While static LiDAR scanners usually focus on detecting objects, for moving LiDAR sensors the own positioning is of additional interest. This process is also known as SLAM (simultaneous localization and mapping). This can for example be used for measuring rooms. Samsung also uses LiDAR sensors for this purpose in its new robot vacuum cleaners [3]. To create the map of an environment, a robot can move to different points. Its movements can be determined by the LiDAR itself and depending on the construction also by wheel positions. To correct errors, positions are also approached multiple times to correct errors accumulated during the movement. This is also called loop detection (approaching a location twice) and loop closure (error correction) [11].

While LiDAR is used mounted directly on vehicles, as shown previously, they can also be used statically at road intersections. Based on the size of the set of points, the distance from the LiDAR, and the direction taken, it is possible to decide whether it is a pedestrian or a vehicle. This information can be used to optimize the flow of traffic at intersections. In a potentially further step in connected driving, approaching vehicles could be warned of pedestrians or even force an emergency brake [34].

#### 2.4.1.3 Object Classification Strategies

Most approaches are based on the ROS (Robot Operating System). The ROS is an open-source framework for robot software development [25].

**Heuristic Approach**

Heuristics in the sense of computing is defined as *"proceeding to a solution by trial and error or by rules that are only loosely defined"* in the Oxford Dictionary of English [22]. In terms of tracking, this means expecting a moving object's appearance at a certain time and place. As a loose rule, the object's current motion can give an expected value for a future point in time.

**Machine Learning Approach**

While in the heuristic approach, rules are defined manually, in machine learning, these rules are automatically constructed based on a training data set. The classification of objects belongs to the supervised learning methods. This means that, for example, to recognize a leg, you need a training set that contains legs and non-legs representations and the information whether it is a leg or not. From this information, a model can be derived using a learning method. This model can then be used to make predictions [15]. Neural networks are suitable as a learning method for the classification of image and sound files [1].

**Additional data sources**

Regardless of whether a LiDAR is used in motion or static, the technology is also supplemented with other tracking systems such as video cameras [21].

### 2.4.2 Existing systems

- LD: Leg Detector is the most popular ROS package for people detecting with a LiDAR sensor. Recognition at LD is based on a machine learning classifier [24].
- PeTra (People Tracking) is a system based on a Convolutional Neural Network (CNN). It is developed by the Robotics Group of the University of León, Spain. CNN is a subset of neural networks [1]. PeTra was developed for use on mobile robots. This means that the scanner's working height was assumed to be 30 to 50 cm above the ground. Accordingly, the system was designed and trained to detect human legs. In comparison with the LD Package PeTra was able to achieve higher accuracy [9].
- Nemati et al. (2016) developed an approach for tracking people using heuristics and a LiDAR sensor. The team was able to identify an object that was obscured by another object through the prediction from the reappearance. But it was also noted that a changing velocity of a hidden object leads to a mismatch between the hypothesis and the reallocation. In any case, this leads to the result that the object can no longer be identified with confidence [20].

### 2.4.3 Discussion

It can be stated that the same goals can be addressed with both moving and static LiDAR sensors. The detection of persons can take place at intersections as well as at the vehicle itself. It depends on the purpose of the data whether it is captured statically or in motion. For the purpose of this thesis, the tracking of people in a room, it is more appropriate to keep the LiDAR static. The LiDAR scanner is not mounted on a robot or a moving object and is not used for localization (SLAM). Due to the static positioning, the height of the LiDAR can be freely chosen. The system is designed to be operating at heights of

1.20 to 1.50 meters. This, in turn, leads to the result that the system should not recognize legs but upper bodies.

This thesis uses a 2D LiDAR, allowing a deployable system to be constructed at lower cost than with 3D sensors.

Since no training data set is available for a neural network supporting the given LiDAR, the heuristic approach is chosen. In the classification, as in related works, the width of the object is used. Techniques such as surface matching cannot be used since the measurement consists only of a single layer.

Theses have been done on LiDAR tracking, but most focus on moving LiDAR sensors. This thesis follows the static approach. The system is designed to be extensible with a second LiDAR. In combination of with a low-cost 2D LiDAR sensor and detection at body height instead of leg, the thesis provides additional value over existing related work. In contrast to the recognition of legs, where it must always be taken into consideration that legs could be behind each other, the recognition at body height offers chances for higher recognition rates. The expandability to add a second LiDAR sensor provides further research opportunities for the future.

# Chapter 3

# LaFlector's Prototype Design and Implementation

The system is based on a distributed architecture consisting of a server and one or more node(s). A node is connected to a LiDAR device either over WiFi or Ethernet. Since this thesis focuses on a single LiDAR setup, only one node illustrates the systems' architecture.

The whole system consisting of node and sink is called LaFlector. A wordplay from *laser beam*, which is emitted by the LiDAR, and the *reflection* caused by an object and thus the basis of each single measuring point.

## 3.1 Materials

The laser scanner used is a Slamtec Mapper M1M1 with a distance range of 20 meters and a sample rate of 7000 Hz (laser points sampled per second by radar). The device has built-in functions for simultaneous localization and mapping, which were not used. Instead, the prototype works with the raw, not pre-processed data of the laser. The node can be adjusted to work with cheaper devices like Slamtec RPLIDAR A1/A2/A3 with low effort. The sink and node are driven by an x86-based system running Arch Linux (Kernel 5.9.16). The code of the node can be cross-compiled for the Armv7 platform. The sink controls the behavior of the node through sockets and provides an InfluxDB instance. InfluxDB is an open-source time-series database (TSDB) and is used to store the data points delivered by the node.

## 3.2 Requirements

The following feature requirements are derived from the objectives of this work. They are subdivided according to components. The prototype is tested against the requirements in the evaluation.

### 3.2.1 Node

The data collection section describes the needed steps from capturing the environment to a set of data points. The following requirements:

| | |
|---|---|
| Req. ID | Req 1.1 |
| Title | **Data fetching** |
| Description | The node must fetch the data points from the connected LiDAR device using the Software Development Kit provided by Slamtec. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 1.2 |
| Title | **Discard data points that are not valid** |
| Description | The Slamtec SDK returns a boolean with every data point determining if the data point is valid. The node must discard the data point if it is not valid. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 1.3 |
| Title | **Add data to database instance** |
| Description | The node must write the data points to the database specified by a configuration file. |
| Priority | 1 |

### 3.2.2 Node controlling/Communication

This section focuses on the controlling of the node. It determines how the node is connected to the sink and what commands are accepted.

| | |
|---|---|
| Req. ID | Req 2.1 |
| Title | **Establish socket connection** |
| Description | The node must connect to the sink socket server specified by a configuration file. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 2.2 |
| Title | **Socket commands** |
| Description | The node must accept the following commands: start (starts recording of data points), stop (stops recording of data points), exit (ends execution on both node and sink) |
| Priority | 1 |

### 3.2.3 Sink

The sink is responsible for interpreting the collected data by the node. It performs the tasks of segmentation, classification, and tracking. This leads to the following requirements:

| | |
|---|---|
| Req. ID | Req 3.1 |
| Title | **Initialization mode** |
| Description | The sink must start/stops the node to get initial data points of the surrounding. From this, it creates the data set of static objects, which is used for the segmentation. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 3.2 |
| Title | **Classification** |
| Description | By using predefined thresholds, the sink must classify data points as potentially human or non-human. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 3.3 |
| Title | **Tracking** |
| Description | The sink must be able to follow an object classified as a human. |
| Priority | 1 |

| | |
|---|---|
| Req. ID | Req 3.4 |
| Title | **People crossing** |
| Description | The sink must distinguish crossing people by calculating expectation values for the position. |
| Priority | 2 |

| | |
|---|---|
| Req. ID | Req 3.5 |
| Title | **Visualization** |
| Description | The sink must be able to display the tracking in a coordinate system graphically. |
| Priority | 3 |

## 3.3 Assumptions

The system was designed to work under certain conditions. Therefore, the following assumptions are considered for an ideal operation:

- **Positioning:** The LiDAR scanner is optimally placed and horizontally aligned. The optimal placement depends on the environment. There should be no dead spots in the environment if possible. The floor of the environment is flat and level.
- **Height:** The object classification expects a solid body with a certain width concerning the measured distance. The laser must be operated at upper body height. Using it at foot level would lead to incorrect classifications and results.
- **Disturbances:** It is assumed that no other devices are continuously transmitting signals at the same wavelength to which the LiDAR scanner would respond. This applies, for example, to laser pointers or infrared remote controls.

## 3.4 Architecture

In a follow-up to this thesis, it could be interesting to add a second LiDAR scanner to the system. Therefore, attention was given to the extensibility. A client-server approach, called node and sink, was chosen to ensure this characteristic. This node-sink structure allows multiple nodes (LiDAR, data collection) to be run on one sink (data processing). To enable extensibility and meet the above requirements, the following architecture was designed.

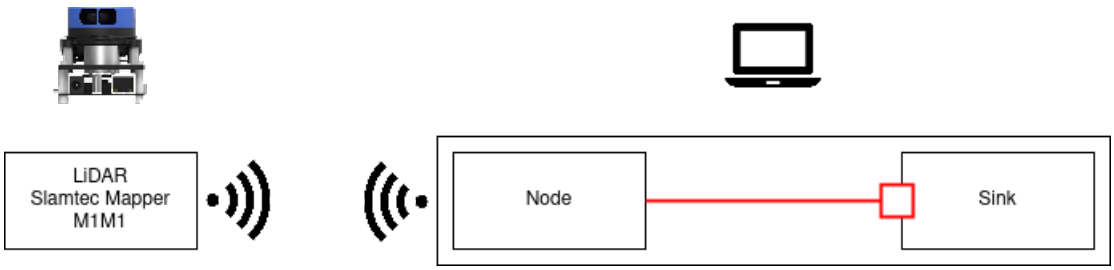



Figure 3.1: Architecture

The LiDAR scanner is connected to the Node via Wireless LAN. Alternatively, a LAN connection can also be created with the device being used. A node communicates with the sink via a socket. In the context of this thesis, the node and the sink are running on the same physical device, but distribution to different devices in the network is possible.

### 3.4.1 Components

A deeper view of the individual components is shown in this diagram. It is divided into LiDAR, Node and Sink and shows the respective sub-components.

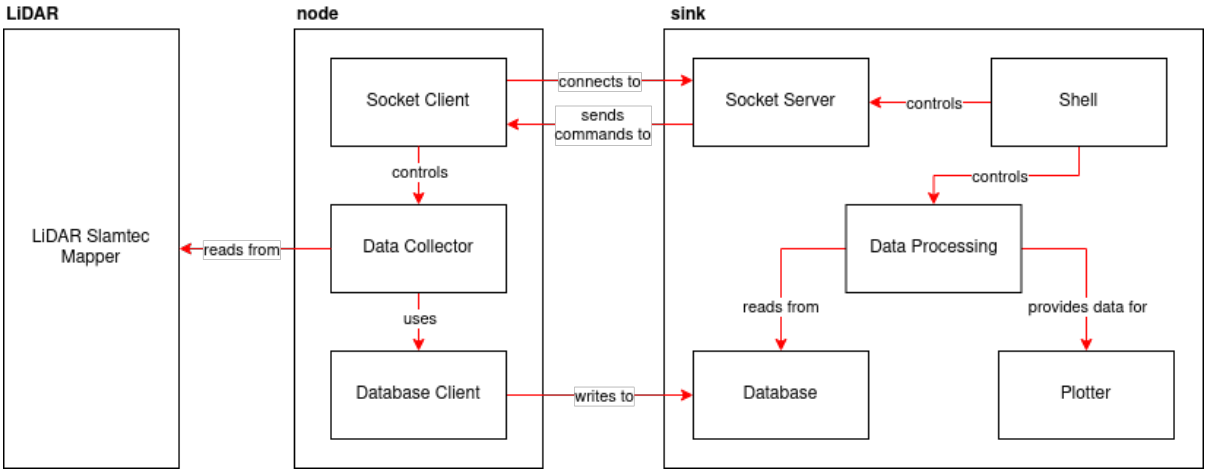



Figure 3.2: Component Diagram

The different components are explained in the following:

**LiDAR:** The LiDAR scanner consists of no other visible sub-components and is basically handled as a black box. The Slamtec SDK provides the needed functions to work with the device. The detailed implementation of the LiDAR scanner itself is not part of this thesis.

**Node:** The node is composed of the modules Socket Client, Data Collector and Database Client. The Socket Client connects to the Socket Server of the sink and receives the commands from it. Depending on the command, the behavior of the data collector is changed. The Data Collector, as the name suggests, collects the data from the LiDAR and passes it to the Database Client. The Database Client writes the data to a database provided by the sink.

**Sink:** The sink is the largest component and consists of five sub-components. The Socket Server ensures the connection to the node. The shell component captures the user inputs, forwards them to the socket server and controls the behavior of the Data Processing. The Data Processing is the most complex and computationally intensive component. It uses the database as data source, processes this data and makes it available to the Plotter. The plotter visualizes the processed data.

### 3.4.2 Flow diagram

The time sequence in which the components interact with each other is shown in the following diagram. The sink must be started first because the node tries to establish a connection to the sink directly after execution. If the connection fails because the sink was not yet ready the node must be restarted.

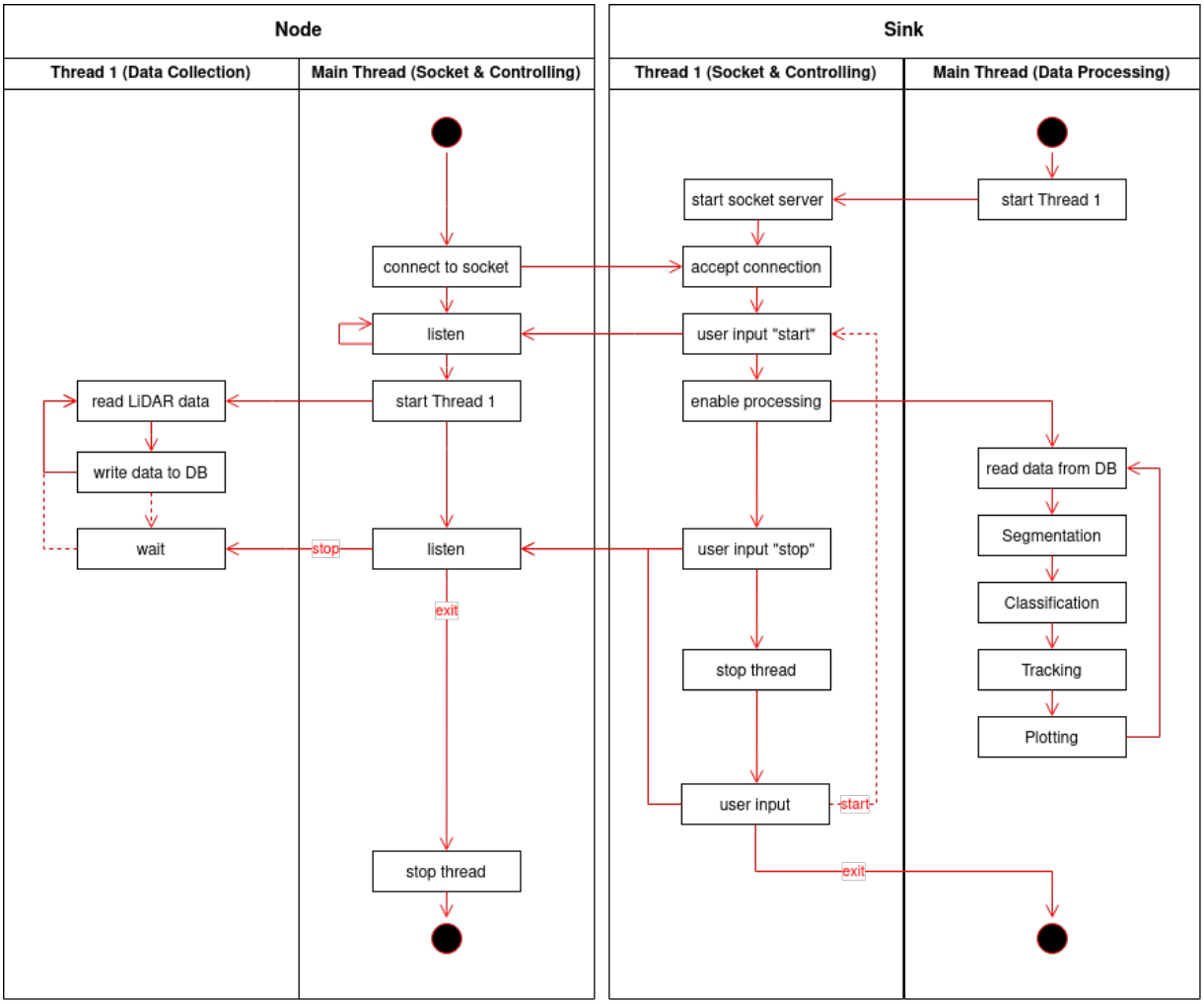



Figure 3.3: Flow Diagram

If the socket connection between node and sink was successfully established, the node could be controlled via the sink and the commands start, stop and exit.

- **Node:** When the node receives the start signal, data collection is started in a thread. In the main thread the socket client is listening for new commands coming from the sink. This data collection in thread 1 runs until the node receives the stop command. When the stop signal is detected, the atomic Boolean variable *run* is set to false and the loop in thread 1 is suspended. By setting the variable back to true via the start command, the data collection is started again.

- **Sink:** First the separate thread 1 is started, in which the socket server is running. The main thread is reserved for data processing, because the matplotlib library needed for plotting only works in the main thread [31].

As soon as the data acquisition is running, the process of data processing is started. Data processing is briefly delayed so that the node could already write data sets to the database. The loop of the data processing runs until the stop signal is given by user input.

## 3.5 Implementation

### 3.5.1 Global Configuration

Node and Sink use parameters for establishing the connection to the LiDAR device, database connector, socket setup and tracking settings. For simplicity, the same config file can be put on Node and Sink. The respective components use only the parameters that apply to them.
The config file is formatted as a YAML file. YAML is a data serialization standard and is often used for config files. Unlike JSON, it supports comments[33], which is helpful for the numerous (especially tracking) parameters. The most important parameters are listed and explained below:

| **Parameter** | **Default value** | **Description** |
|---|---|---|
| initialization-rotations | 1000 | The creating of the static dictionary uses this number of rotations. |
| rotations-per-loop | 3 | A single rotation does not provide enough data points. This parameter determines how many rotations are merged. |
| static-moving-distance | 0.3 | If an object is less than than this value from a static object away it will not be identified |
| human-width | 0.15 | If a object should be identified as human, it must at least have this width. This value prevents small moving objects from being classified as human. |
| split-distance | 0.8 | If two objects are right next to each other, they cannot be distinguished. If the objects move apart, this parameter specifies the threshold at which they are classified as two objects instead of one. |
| existing-new-threshold | 0.8 | When a new object is captured, this value is used to decide whether it belongs to an existing object. The existing object must be less than this value away, otherwise it is considered to be a new object. |
| way-points | 10 | This parameter specifies how many of the last position way-points are used for the direction vector. |
| die | 50 | A loop consists of rotation-per-loop many rotations. The die value indicates in how many loops an object may no longer be detected before it dies respectively is considered inactive. |
| born | 10 | A loop consists of rotation-per-loop many rotations. The born value indicates how many loops an object must be sighted before it is recognized as such. |

Table 3.1: Configuration Parameters

### 3.5.2 Node

#### 3.5.2.1 Language, SDK, and Libraries

The SDK provided by Slamtec for the LiDAR scanner is written in C++. To keep the node consistent and prevent issues using C/C++ wrappers for Python, the node was also developed in C++. All functions except the InfluxDB client were implemented without additional libraries. To establish the connection to InfluxDB the lightweight header-only library *influxdb-cpp* was used.

Multi-threading was enabled as described in 3.4.2 in order to simultaneously acquire data and process commands via the socket. The std::thread class is available as of C++11. Before that, POSIX threads or pthreads had to be used. The LiDAR SDK requires GCC 4.8.x and was also written according to C++0x standard, with higher GCC versions the build process could not be completed successfully. With modifications to the makefile (CXX flags and LD flags) it was nevertheless possible to use the standard C++11 threading class with the SDK and GCC 4.8.

#### 3.5.2.2 Code

```
1     while (run && connected) {
2         rpos::features::system_resource::LaserScan laser_scan = sdp.
              getLaserScan();
3         std::vector<rpos::core::LaserPoint> laser_points = laser_scan.
              getLaserPoints();
4         //do one full rotation
5         for (std::vector<rpos::core::LaserPoint>::iterator it =
              laser_points.begin();
6               it != laser_points.end(); ++it) {
7              // if the data point is not valid, skip
8              if (it->valid() != 0) {
9                  // initialize datapoint vector
10                 std::vector<float> datapoint;
11                 //push angle and distance to datapoint vector
12                 datapoint.push_back(it->angle());
13                 datapoint.push_back(it->distance());
14                 //add the datapoint vector to the datapoints vector
15                 datapoints.push_back(datapoint);
16             }
17        }
18        write_datapoints(iterator, datapoints);
19        iterator = iterator + 1;
20    }
```

Listing 3.1: Data Collection Snippet

Collection of data is controlled by an atomic boolean, that can be written and read across threads. Thus, if the node receives the start command, the variable run is set to True. The data collection runs until the variable is set back to False again.
Since a rotation does not always have the same number of valid points, it makes sense to use the C++ standard vector class instead of an array. The snippet uses two vectors: the datapoints vector and the datapoint vector.
The datapoint vector corresponds to a single measured value. The SDK returns three parameters: angle, distance, and a validity boolean. If a data point is invalid, the SDK sets the distance value is *100000*. This information does not offer additional value, so invalid points are already discarded at this point and not processed further. It follows that the datapoint vector only contains the parameters angle and distance.
The datapoints vector consists of datapoint vectors – exactly as many as valid measured values were acquired in the last full rotation. The datapoints vector is then passed to the influx-db client and the values are written to the database. For performance reasons, it was decided not to write each single datapoint directly into the database, but to collect a full rotation since the client needs to reconnect to the database for every insert.

### 3.5.3 Sink

#### 3.5.3.1 Language and Libraries

The Sink was written in Python since Python has more and easier to use extensions for data collection, manipulation, and visualization. That is why Python is the most used programming language in the field of Data Science [14]. Python interpreter was running in version 3.9. Apart from the Python standard libraries, the following additional packages were used:

- **InfluxDB:** To read the values of the node from the Influx database, this package was used in version 5.3.1. The package is also officially recommended by Influx when using Python [13].
- **PyYAML:** Instead of writing a simple YAML parser like in the node, the PyYAML package in version 5.4.1 was used. This is also because the parameters for the sink are grouped and therefore a bit harder to read.
- **NumPy:** This very comprehensive package provides data structures and functions for mathematical operations. It was used for the conversion of cartesian and polar coordinates and the midpoint and distance determination. It is also a dependency for the matplotlib library. Version 1.20.1 was installed for this thesis.
- **Matplotlib**: To visualize the results the matplotlib library (version 3.3.4) was used. It can represent numpy arrays in coordinate systems.

Independently of Python, Tk must be installed. This toolkit enables the creation of graphical interfaces. If matplotlib runs in interactive mode, the package must be installed [30]. Tk is available for Arch Linux (Extra Repository) and Ubuntu (Universe Repository).

#### 3.5.3.2 Modules

**Data Segmentation**

Data segmentation consists of two tasks. The first task is to capture the static objects. In a second step, new measurements must be matched with the static objects and the difference can be built. Since the matching has to be done with every rotation and each rotation contains hundreds of points, this step has to be done with good performance. For this reason, the angle and distance list are written to a dictionary. This has the advantage that each angle occurs only once, because keys are unique in Python dictionaries. This means that for each angle there is only one distance. Another advantage is that the lookup in a Python dictionary runs in O(1) [8].

```
1  for i in range(0, self._initialization_rotations):
2  # get angle and distance list from the last iteration
3  angle_list_latest, distance_list_latest = influx.
       get_ad_from_last_iteration()
4  # merge the lists from the previous rotations
5  angle_list = angle_list + angle_list_latest
6  distance_list = distance_list + distance_list_latest
7  # create empty static dictionary
8  static_dict = {}
9  # merge lists into a dictionary
10 for (angle, distance) in zip(angle_list, distance_list):
11     static_dict[angle] = distance
```

Listing 3.2: Creation of dictionary of static objects

This snippet shows the acquisition and construction of the Static Dictionary. This dictionary remains unchanged for the rest of the current measurement. Once the static dictionary is created, detection of moving objects can start.
Angle and distance are now retrieved again per rotation and written to the moving objects dictionary. The moving dictionary is compared with the static dictionary and as soon as a threshold (parameter: static-moving-distance) is exceeded, the value is stored in a separate angle and distance list. These values are then checked and classified in the next step.

```
1  for angle in scan_dict.keys():
2  temp_angle = angle
3  while temp_angle not in static_dict.keys():
4      temp_angle = temp_angle + 0.001
5      if temp_angle - angle > 0.005:
6          break
7  if temp_angle in static_dict.keys():
8      # do not allow points, that lay behind static objects, if this
           happens, something is wrong in the
9      # room and if the distance difference is bigger than a threshold it
           must be a moving object
10     if static_dict.get(temp_angle) > scan_dict.get(angle) and (
11             static_dict.get(temp_angle) - scan_dict.get(angle)) > self.
                   _static_moving_distance:
12         angle_list_moving.append(angle)
13         distance_list_moving.append(scan_dict.get(angle))
```

Listing 3.3: Comparing points to static objects

### Object Classification

Object classification starts with the two lists *angle_list_moving* and *distance_list_moving*. These lists contain the segmented measured values of the last three rotations. Our tests have shown that single rotations contain too few data points and are therefore not reliable.

```
1  startpoint_index = 0
2  endpoint_index = 1
3
4  # object[i] = object()
5  object_list = []
6
7  # loop until the endpoint (e.g. index 500 is bigger than the length of
       the angle_list_moving list
8  while endpoint_index < len(angle_list_moving) - 1:
9      # save the initial startpoint since startpoint will be shifted. This
           variable must be used to create
10     # the object.
11     initial_startpoint = [angle_list_moving[startpoint_index],
           distance_list_moving[startpoint_index]]
12     while helpers.distance_between_two_p_points(
13         angle_list_moving[startpoint_index], distance_list_moving[
               startpoint_index],
14         angle_list_moving[endpoint_index], distance_list_moving[
               endpoint_index]) < self._split_distance:
15         # if list is finished, break loop
16         if endpoint_index >= len(angle_list_moving) - 1:
17             break
18         else:
19             # shift start and endpoint
20             startpoint_index = startpoint_index + 1
21             endpoint_index = endpoint_index + 1
```

Listing 3.4: Dividing moving point into objects

The first task of classification is to distinguish between multiple objects. To do this, one starts at the first point and compares it with following one. If the distance between the points is smaller than a certain threshold (parameter: split-distance) it must still be the same object. The check points are shifted and it is checked again. This process is repeated until the object is completely captured. After the objects from the lists are separated, they are identified in the next step.

### Object Identification

Object identification has the task to decide whether a set of points, which was classified as a human object, can be assigned to an already existing object or whether the object is new. This task was solved as follows:

```
1  def check_if_existing_object(Object, startpoint, endpoint, iterator):
2      # object_list collecting all possible objects
3      object_list = []
4      # first convert the coordinates into cartesian
5      x1, y1 = convert_point_from_p_to_c(startpoint[0], startpoint[1])
6      x2, y2 = convert_point_from_p_to_c(endpoint[0], endpoint[1])
7      # find midpoint
8      midpoint_x, midpoint_y = c_midpoint_from_c_points(x1, y1, x2, y2)
9
10     for created_objects in Object:
11         if created_objects.isDead is False:
12             midpoint_existing_x, midpoint_existing_y = created_objects.
                   get_midpoint()
13             # if the distance between the midpoints is smaller than the
                   threshold it must be a already existing object
14             if distance_between_two_c_points(midpoint_x, midpoint_y,
                   midpoint_existing_x, midpoint_existing_y) \
15                     < globals.config.get("tracking-parameters").get("
                           existing-new-threshold"):
16                 # add possible object to list
17                 object_list.append(created_objects)
18
19     # if only one object is possible, the identification is unambiguous,
           update the corresponding positioning values
20     if len(object_list) == 1:
21         object_list[0].set_startpoint_from_p(startpoint[0], startpoint
               [1])
22         object_list[0].set_endpoint_from_p(endpoint[0], endpoint[1])
23         object_list[0].update_last_seen(iterator)
24         globals.Logger.log(object_list[0].get_identifier() + "'s
               position updated to [" +
25                            str(object_list[0].get_midpoint()[0]) + "," +
                                   str(object_list[0].get_midpoint()[1]) + "
                                  ]",  2, iterator)
26         return True
27
28     # if more than 1, assume that the objects keep moving in the same
           direction and update start/endpoint accordingly
29     # not update last_seen, if object keeps hidden, better dies because
           the heuristic approach becomes more and more
30     # inaccurate
31     if len(object_list) > 1:
32         for objects in object_list:
33             objects.heuristic_update(iterator)
34         return True
35     # if no object found, then it must be new
36     else:
37         return False
```

Listing 3.5: Object Identification

The set of points is matched with all objects that are still active. Basically, there are three cases that need to be distinguished:

- **No object match**: The points do not correspond to any previously known object. The object must be new accordingly.
- **One object match**: The points can be clearly assigned to an object. In this case, the new position of the object is set.
- **Two or more matches**: In this case, it is not possible to decide which points belong to which object. Assuming that the objects continue to move in the same direction, the expected positioning is determined by means of a direction vector derived from the last way-points.

#### 3.5.3.3 Output

The output is available in two forms.

**Plot**

The position of the detected objects and their direction vectors are displayed in a dynamic X/Y coordinate system. This output form serves for visualization.

**Logger**

The system has a logger which creates a file with the start time at startup. The verbose level determines which data is written to the log file and which is not. This output is used for debugging. Furthermore, the logger function can be quickly exchanged with a database client, so that the corresponding data ends up in a database instead of the log file. This is in case the data should be further processed or combined with another data source.

# Chapter 4

# LaFlector's Evaluation

This chapter describes under which conditions and for which properties the prototype was tested. Furthermore, the results are shown and discussed.

## 4.1 Evaluation Setup

The evaluation was conducted in a 18 square meter room. Larger areas would be easier to track as it becomes difficult as soon as objects are grouped closely. The windows were covered, and there were several static objects in the room. The runs were performed with the parameter's default value. To simulate objects' appearance, both entering the measurement room and spontaneously joining the measurement height were attempted. To test an object's disappearance, the tracked person left the room or suddenly went below the measurement height. The measurement height was 144 centimeters above the ground for all test runs. Four scenarios were run five times each:

- **Single Person Tracking**: One person moves through the room at a walking pace. There are no other moving objects. This scenario is the basis for further evaluation runs and should work flawlessly.
- **Two Persons Tracking**: Two people move simultaneously at a walking pace in the room. They never stand behind each other and are always at least one meter apart. However, the paths can intersect at different times.
- **Two Persons Crossing**: Two people cross paths. For a short time, one person stands behind the other. The persons do not make any hard changes of direction during the crossing.
- **Single Person behind Static Object**: A static object stands in the room, which was captured at the beginning. A person moves behind the object for a short amount of time but keeps the walking direction.

These runs were measured using the following criteria. A criterion can be considered as passed or failed.

- **Positioning**: The tracking is accurate and continuous. The criterion is considered as passed if the object is tracked accordingly to the real position and there are no unexpected jumps in the way-points.
- **Classification**: The person is recognized and correctly classified. No static objects or objects are classified as human objects. The criterion is considered as passed when the number of human objects is correctly detected.
- **Identification**: The same object is always identified as the same. If a person reappearing from behind a static object is classified as a new object, the criterion is considered failed. If two people cross each other and the system can no longer assign the objects because the object identification was lost, the criterion is also considered failed.

## 4.2 Results

The table below shows how often a criterion was not met in 5 runs. In general, it can be said that the positioning worked in every case. There were a few failures in the identification and classification in scenarios 3 and 4.

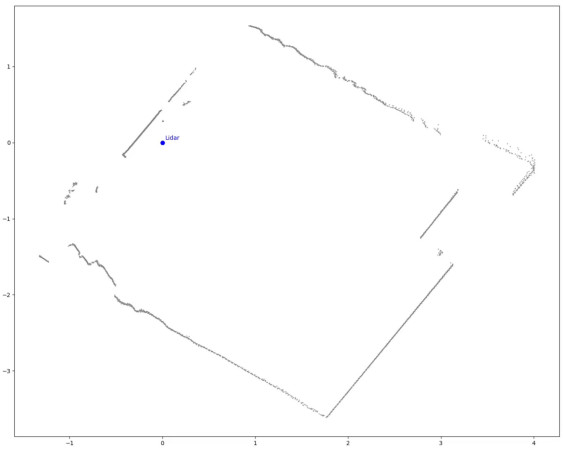
| Scenario | Positioning | Classification | Identification |
|---|---|---|---|
| 1: Single Person Tracking | 0 | 0 | 0 |
| 2: Two Persons Tracking | 0 | 0 | 0 |
| 3: Two Persons Crossing | 0 | 0 | 2 |
| 4: Person behind Object | 0 | 0 | 1 |

Table 4.1: Results Overview

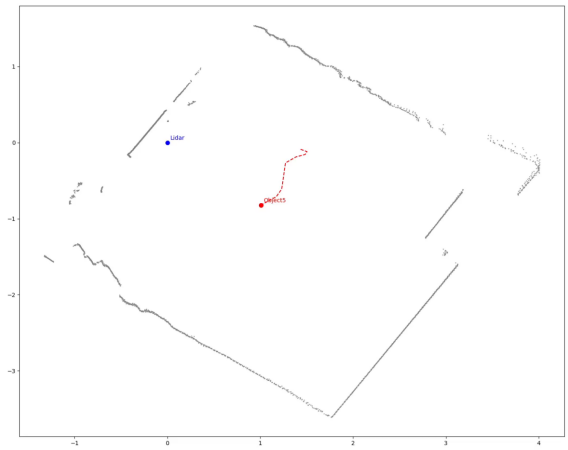
The results are represented with the created plots of the LaFlector. A series of snapshots visualize the first scenario. These snapshots were taken at regular interval (every 5 seconds) during the run. The remaining three scenarios are represented by the last snapshot before the object disappears/dies. Way-points are recorded for the evaluation. In the usual tracking mode, the way-points are not displayed by default. If desired, the plotting of way-points can be enabled in the plotter class.

### 4.2.1 Scenario 1: Single person

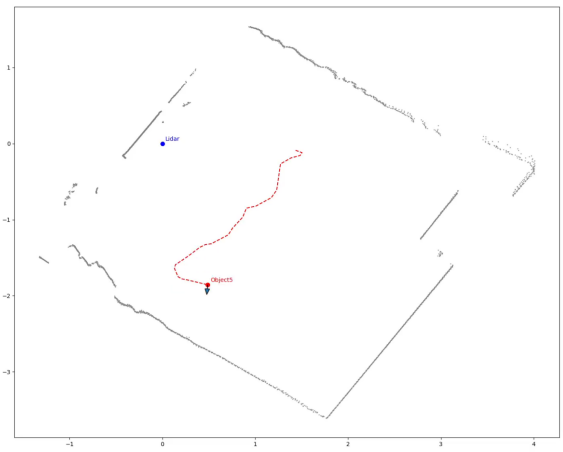
The first scenario was completed successfully. In all five runs, the person was correctly detected, tracked without interruption, and removed again after disappearing (marked as dead). No static objects were classified as people. The following graphics show the progress of the run with a running time of 40 seconds, in which the object alternately moved and remained stationary.



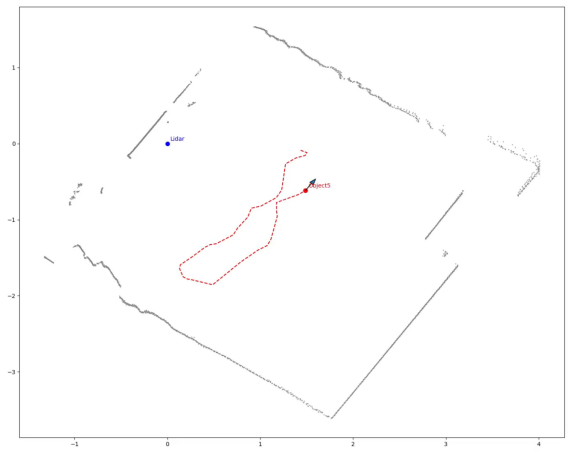

Frame 1 of 8: Object not yet detected



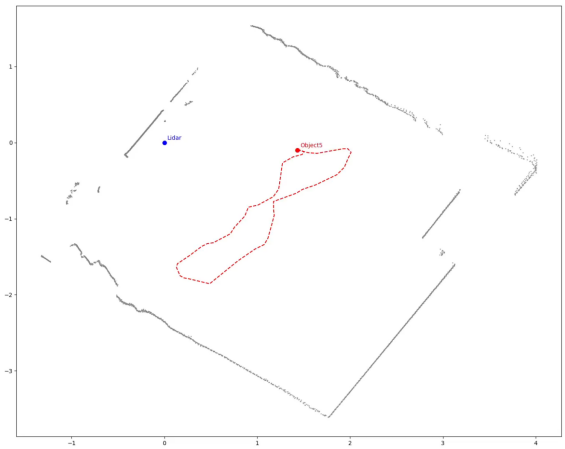

Frame 2 of 8: Object detected, Tracking started



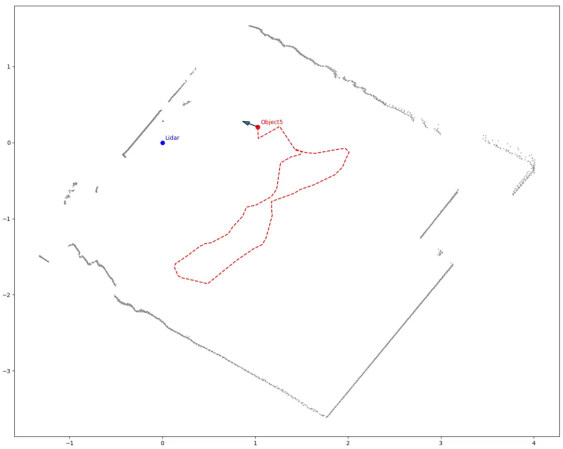

Frame 3 of 8: Active tracking, direction vector visible



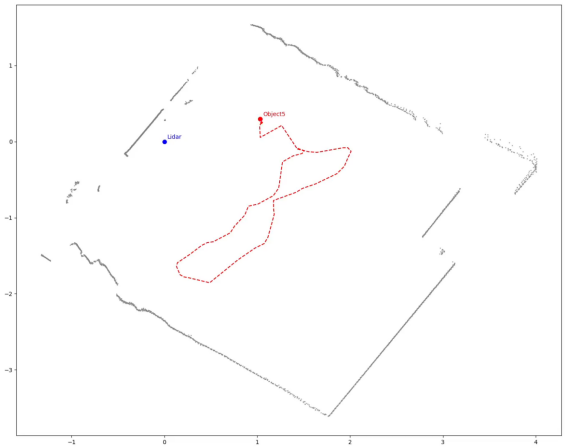

Frame 4 of 8: Active tracking, direction vector visible



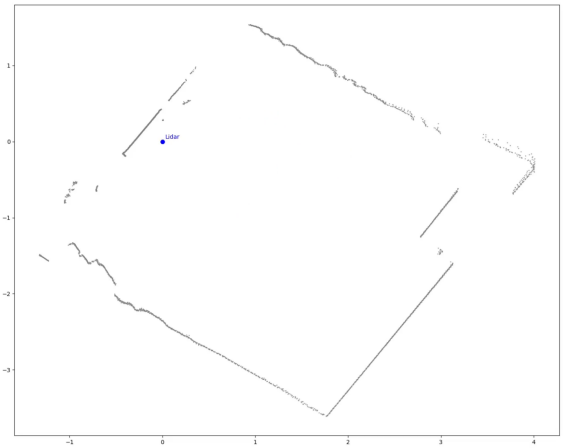

Frame 5 of 8: Object stopped, no direction vector




Frame 6 of 8: Active tracking, direction vector visible




Frame 7 of 8: Object stopped, no direction vector




Frame 8 of 8: Object died, way-points wiped

Figure 4.1: Snapshot series of a tracking run

### 4.2.2 Scenario 2: Two or more persons not crossing

The second scenario in which two people were tracked was also successfully handled. Both persons were recognized at the same time, and their path was tracked correctly. Even when the two people were close to each other (but not behind each other), the identification worked without any issues in all five runs. The shown run took 35 seconds.

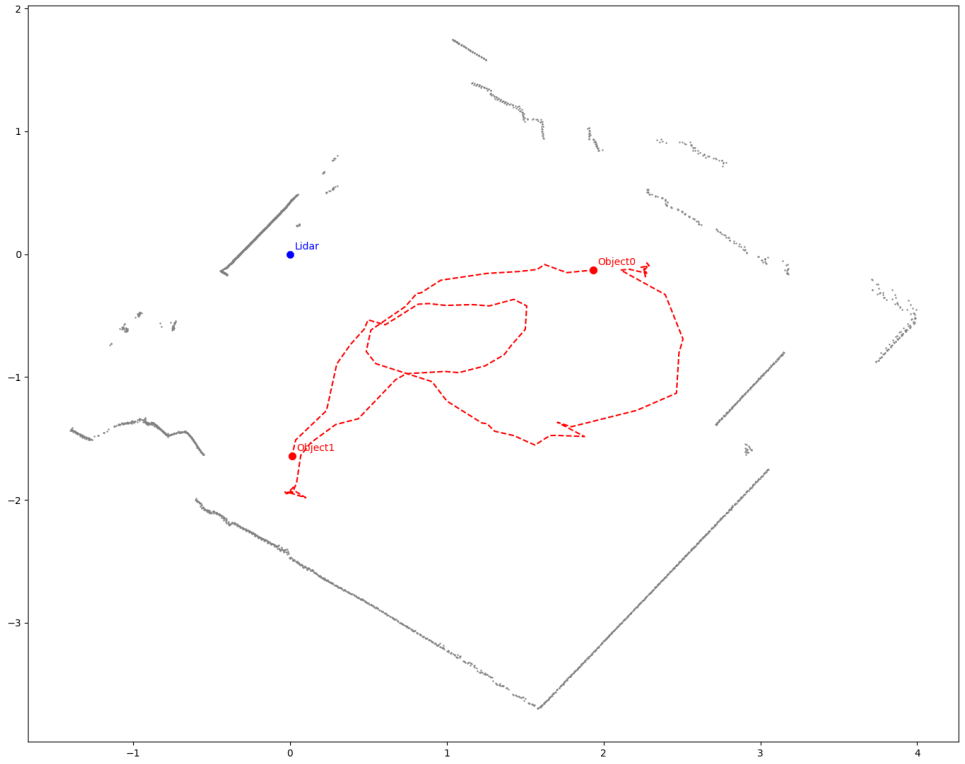



Figure 4.2: Plot: Two persons tracking

### 4.2.3 Scenario 3: Two persons crossing

In the third scenario, two people cross paths, assuming that they do not change their speed significantly. If a person started walking while being covered by the other person, there would be no heuristic possibility to detect this. Since according to the last confirmed information, the person was at rest.
The five crossings were performed with the following estimated crossing angles: twice 180 degrees, once 120 degrees, once 90 degrees, once 60 degrees. In two out of five cases, the individuals could not be positively identified after crossing. The smaller the crossing angle, the higher the probability that the recognition fails because the persons are not distinguishable for the LiDAR for a longer time. Accordingly, the identification of the objects failed at 90 and 60-degree crossing angles. The best results can be achieved when the persons cross at a straight angle (180 degrees). An example of a run with a duration of 8 seconds and a straight angle is shown below.

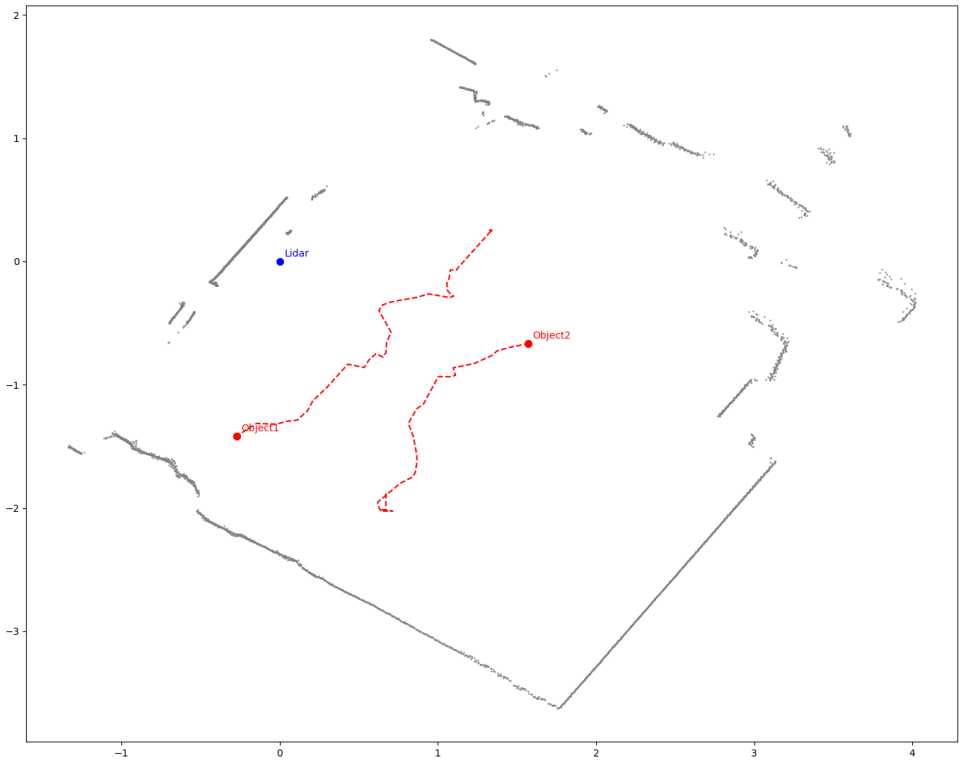



Figure 4.3: Plot: Two persons crossing

### 4.2.4 Scenario 4: Single person behind static object

In the fourth scenario occurred one error: In the second run, after reappearing, the object was identified as a new object, not the original one. This may be due to too great a change in speed or change of walking directly behind the static object. In this case, the heuristic prediction no longer applies.

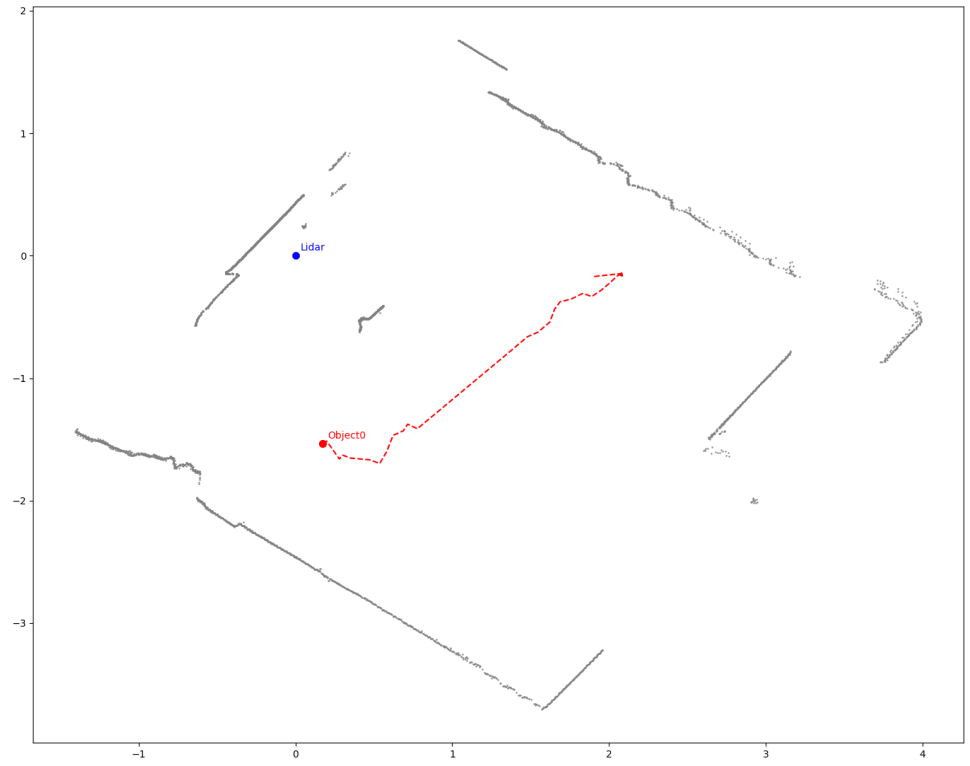



Figure 4.4: Plot: Single Person behind Object

## 4.3 Limitations

The results clearly reflect the strengths and weaknesses of a LiDAR sensor from section 2.2.3. As expected, the positioning was accurate. Things get tricky when objects are obscured by other objects, regardless of whether they are static or in motion. Herein, it can only be evaluated with a certain probability whether the object still exists and where it is located when reappearing. How well this case works also depends significantly on the parameterization. For example, if people are expected to move close to each other, the split-distance and existing-new-threshold parameters should be decreased. This increases

the probability that the persons will be detected. At the same time, it increases the risk that single persons will be detected as several.

## 4.4 Discussion

Comparing the goals set with the results, the following conclusions can be made.

**LiDAR Interface**

This interface had the task of making the data from the LiDAR scanner available for data processing. This goal was achieved. The data processing always received the latest laser data via the Influx database. No errors occurred in the interface during any of the runs. The data acquisition worked without any interruptions. The requirement of controlling the node via a socket was met.

**Tracking Technique**

The most extensive goal can be considered as largely fulfilled. From the first scenario, it can be deduced that the segmentation of static and moving objects works reliably. The moving object was recognized as such and tracked. Further, scenario 2 shows that the system can also detect and track two or more people simultaneously. Errors occurred in scenario 3. The system could no longer determine which of the intersecting persons was in two out of five cases. Scenario 4 further demonstrates that heuristic prediction works. Objects were recognized after disappearance except for one case. It follows that when two crossing objects are in motion, the system appears to be less accurate than when one of the objects is static. It is possible that even better results could be obtained with more complex, finer-grained heuristic prediction. However, this would also require a higher resolution LiDAR scanner. In particular, far away objects from the scanner would otherwise take too long to be reliably detected since the density of measurement points decreases with distance.

**Data Output**

The graphical illustrations from the results are from the plotter of the system. Therefore, the requirement of visualization can be considered satisfied. The log file created with each run contains the desired information according to the selected log level. It is, therefore possible to read all points X/Y that an object had during a run. With a simple code extension, the logger's data could be made available for further processing or combination with another data source. Therefore, the requirement of expandability can also be confirmed.

# Chapter 5

# Summary and Conclusions

The main objectives were achieved. The LiDAR interface works as desired. The data is read from the LiDAR and written to a database to retrieve for further processing. The heuristic tracking approach also works, as the results show. In more complex situations, however, errors cannot be avoided due to the single LiDAR properties. The output of the data is as prescribed in the goals. The data can be extended for combination with another data source, and a plot is created simultaneously.

During the thesis, especially during tracking development, constant test runs were made to check how objects are detected. It was important to learn how sensitive the LiDAR scanner is to certain influences and how many data points it takes to get a reliable result. In the initial phase, this also led to many trial and error attempts. Many parameters depend on the LiDAR device and its specifications.

The biggest stumbling blocks were the usage of the SDK and identification in tracking. The SDK is only moderately documented. Many comments are not written in general, but Microsoft Visual Studio and some documented steps did not work as described. Several requests to Slamtec were needed and answered at the end, but each message took time, which could have been used further. In the work process, various parameters were implemented because it turned out that it depends significantly on the environment how one should set the tracking.

When it comes to identification in tracking, one comes up against the limits of a single LiDAR scanner. When the object is not visible, one tries to assume information through movement patterns that cannot be detected or confirmed by the LiDAR. This process can be very time-consuming and frustrating, as flawless position prediction is not possible.

This thesis offers several options for future works.

- **Second Data Source**: For this thesis, originally, a WiFi tracking system was intended to capture the WiFi signal and combine LiDAR and WiFi tracking. Unfortunately, it turned out that the deployment of the system was more complex than expected during the work process. One option for future work would therefore be to combine the two systems that now exist.

- **Second LiDAR sensor**: Instead of trying to increase the probability of recovering an obscured object, a second LiDAR sensor can be used to validate the measured or heuristically predicted position values. Provided that people's density is not too high, reliable tracking could already be ensured with a second LiDAR.

# Bibliography


[1] S. Albawi and T. Mohammed. "Understanding of a Convolutional Neural Network ". In: *The International Conference on Engineering and Technology* (2017).

[2] Lenz Baumann. "MAC Scavenger - A Passive Method Handling MAC Randomization on Mobile Devices". In: (2020).

[3] CNET. *Samsung's new robot vacuum uses lidar and empties its own bin like a fancy Roomba*. 2021. URL: https://www.cnet.com/news/samsung-jetbot-90-ai-new-robot-vacuum-uses-lidar-and-empties-its-own-bin-like-a-fancy-roomba/.

[4] Brands Consulting. *MAC-Adresse und der Datenschutz â Fingerabdruck des Computers?* 2018. URL: https://brands-consulting.eu/mac-adresse-und-der-datenschutz-fingerabdruck-des-computers (visited on Mar. 18, 2021).

[5] Volvo Car Corporation. *Next generation Volvo cars to be powered by Luminar LiDAR technology for safe self-driving*. 2020. URL: https://www.media.volvocars.com/global/en-gb/media/pressreleases/268323/next-generation-volvo-cars-to-be-powered-by-luminar-lidar-technology-for-safe-self-driving (visited on Mar. 18, 2021).

[6] A. Filgueira et al. "Quantifying the influence of rain in LiDAR performance". In: *Measurement* 95 (2017), pp. 143–148.

[7] S. Gidel et al. "Pedestrian Detection and Tracking in Urban Environment using a Multilayer Laserscanner". In: *IEEE Transactions on Intelligent Transportation Systems* 11.3 (2010), pp. 579–588.

[8] M. Gorelick and I. Ozsvald. "Dictionaries and Sets". In: *High Performance Python*. 2014, pp. 73–75.

[9] Á. Guerrero-Higueras et al. "Tracking People in a Mobile Robot From 2D LIDAR Scans Using Full Convolutional Neural Networks for Security in Cluttered Environments". In: *frontiers in Neurorobotics* 12.85 (2019).

[10] J. Han et al. "Road Boundary Detection and Tracking for Structured and Unstructured Roads using a 2D LiDAR sensor". In: *nternational Journal of Automotive Technology* 15.4 (2014), pp. 611–623.

[11] W. Hess et al. "Real-Time Loop Closure in 2D LIDAR SLAM". In: *IEEE International Conference on Robotics and Automation* (2016).

[12] Apple Inc. *Wi-Fi privacy*. 2021. URL: https://support.apple.com/guide/security/wi-fi-privacy-secb9cb3140c/web (visited on Mar. 18, 2021).

[13] InfluxData. *Getting Started with Python and InfluxDB*. 2018. URL: https://www.influxdata.com/blog/getting-started-python-influxdb/ (visited on Mar. 18, 2021).

[14] Kaggle. *Data Science Survey - 2018*. 2018. URL: https://www.kaggle.com/sudhirnl7/data-science-survey-2018 (visited on Mar. 18, 2021).

[15] A. Krause. *Introduction to Machine Learning*. 2020. URL: https://las.inf.ethz.ch/courses/introml-s20/slides/introml-01-introduction-annotated.pdf (visited on Mar. 18, 2021).

[16] M. Kutila et al. "Automotive LiDAR performance verification in fog and rain". In: *IEEE Intelligent Transportation Systems Conferenc* (2018).

[17] Google LLC. *Privacy: MAC Randomization*. 2020. URL: https://source.android.com/devices/tech/connect/wifi-mac-randomization (visited on Mar. 18, 2021).

[18] P. McManamon. "History of LiDAR". In: *LiDAR Technologies and Systems*. 2019, pp. 29–34.

[19] V. Navalpakkam and E. Churchill. "Mouse Tracking: Measuring and Predicting UsersâExperience of Web-based Content". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2012), pp. 2963 –2972.

[20] H. Nemati, S. Shahbandi, and B. Åstrand. "Human Tracking in Occlusion based on Reappearance Event Estimation". In: *Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotic* 2 (2016), pp. 505–512.

[21] C. Premebida et al. "A Lidar and Vision-based Approach for Pedestrian and Vehicle Detection and Tracking". In: *Intelligent Transportation Systems Conference* (2007).

[22] Oxford University Press. *heuristic*. 2015. URL: https://www.oxfordreference.com/view/10.1093/acref/9780199571123.001.0001/m_en_gb0375970 (visited on Mar. 18, 2021).

[23] Federal Office of Public Health FOPH. *Faktenblatt Laserpointer*. 2018. URL: https://www.bag.admin.ch/dam/bag/de/dokumente/str/nis/laser/faktenblatt_laserpointer.pdf.download.pdf/2018-02-13_Faktenblatt-Laserpointer_D.pdf (visited on Mar. 18, 2021).

[24] ROS.org. *leg_detector*. 2021. URL: https://wiki.ros.org/leg_detector (visited on Mar. 18, 2021).

[25] ROS.org. *ROS Introduction*. 2018. URL: http://wiki.ros.org/ROS/Introduction (visited on Mar. 18, 2021).

[26] J. Scheuner et al. "Probr - A Generic and Passive WiFi Tracking System". In: *IEEE 41st Conference on Local Computer Networks* 88 (2016), pp. 71–78.

[27] J. Shackleton, B. VanVoorst, and J. Hesch. "Tracking People with a 360-Degree Lidar". In: *IEEE International Conference on Advanced Video and Signal Based Surveillance* (2010).

[28] Slamtec. *Introduction and Datasheet M1M1*. 2019. URL: http://bucket.download.slamtec.com/975678f44875b6cb4db5ecf30202163c8910519b/SLAMTEC_mapper_datasheet_M1M1_v1.2_en.pdf (visited on Mar. 18, 2021).

[29] Slamtec. *Introduction and Datasheet M2M1*. 2019. URL: http://bucket.download.slamtec.com/c61c74740d9260bf55b1d849e1dcf46bbcad8477/SLAMTEC_mapper_datasheet_M2M1_v1.1_en.pdf (visited on Mar. 18, 2021).

[30] Matplotlib development team. *Interactive Figures*. 2021. URL: https://matplotlib.org/3.3.4/users/interactive.html (visited on Mar. 18, 2021).

[31] Matplotlib development team. *Working with threads*. 2019. URL: https://matplotlib.org/3.1.0/faq/howto_faq.html#working-with-threads (visited on Mar. 18, 2021).

[32] H. Wang et al. "Pedestrian recognition and tracking using 3D LiDAR for autonomous vehicle". In: *Robotics and Autonomous Systems* 88 (2017), pp. 71–78.

[33] *YAML Ain't Markup Language*. URL: `https://yaml.org/` (visited on Mar. 17, 2021).

[34] J. Zhao et al. "Detection and tracking o fpedestrians and vehicles using roadside LiDAR sensors". In: *Transportation Research: Part C* 100 (2019), pp. 68–87.

# Abbreviations

| | |
|---|---|
| DB | Database |
| GCC | GNU Compiler Collection |
| GDPR | General Data Protection Regulation |
| IEC | International Electrotechnical Commission |
| LiDAR | Light detection and ranging |
| RSSI | Received Signal Strength Indication |
| SLAM | Simultaneous Localization and Mapping |
| TSDB | Time series database |

# List of Figures

# List of Tables

# Listings

# Appendix A

# Contents of Submission Zip File

**Code.zip**

The ZIP archive contains the code for the node and sink. The SDK is already included.

**Documentation**

This folder contains the LaTeX source code of this thesis and therefore also the used figures.

**Literature**

This folder contains the referenced papers.

**Midterm Presentation**

Midterm presentation is included, final presentation is after submission.

**Evaluation Protocol**

The protocol made during the evaluation is included.