DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Design and implementation of configuration synchronization and WebGUI interface for de-centralized multi LiDAR setups

Uday Navaneeth Menon

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Design and implementation of configuration synchronization and WebGUI interface for de-centralized multi LiDAR setups

Design und Implementierung von Konfigurationssynchronisation und WebGUI-Schnittstelle für dezentrale Multi-LiDAR-Setups

Author: Uday Navaneeth Menon
Supervisor: Prof. Dr.-Ing. Pramod Bhatotia
Advisor: Johannes Meßmer
Submission Date: 15.08.2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.08.2022 Uday Navaneeth Menon

Acknowledgements

First and foremost, I would like to thank my advisor, Johannes Meßmer, for his constant guidance and constructive criticism that has made it possible for me to undertake this endeavor. His proficiency in the field and the ability to explain the subject paved the way for my improved understanding of the field. The guidance he has provided me with regards to framing the research topic, drawing up a plan for the structure of the thesis, and helping in tackling challenges that I faced along my research.

I would also like to thank my supervisor Prof. Dr.-Ing. Pramod Bhatotia, for granting me the opportunity to pursue this research with the chair of Decentralized Systems Engineering in the Department of Informatics at the Technical University of Munich.

My colleagues at Blickfeld GmbH have also supported me a lot during my research, supporting me with some issues I faced and providing feedback during the course of my research. I would like to thank them for the fantastic support that they have provided me in completing my research.

A special thanks to my family, whose moral support has granted me the mental fortitude to successfully complete this thesis.

Abstract

Blickfeld GmbH is a Munich-based company that was founded in 2017. Their innovative LiDAR solution combines LiDAR hardware and software. The compact and versatile sensors offer the best combination of performance, reliability, and price. A key factor that drives this product is the software. A customer looking for a LiDAR would prefer one that can integrate into their workflow easily and does not require extensive training. With regular increments of features and the services required to run them, the interface needed to cope with these changes.

One of the exciting new features proposed to be added is the capability of multiple LiDARs to combine their output to create a consolidated point cloud. One of the key requirements for this functionality is that the cubes be decentralized. They must be able to detect and communicate with other cubes on the network. This brings us to the aim of this thesis, to design and implement a configuration synchronization system for a cluster of LiDAR devices and a modular WebGUI for the new generation of LiDARs to go with it. The goal of the web interface is to be modular and scalable. The modularity will aid in adapting the interface to the discrete needs of a customer.

Kurzfassung

Die Blickfeld GmbH ist ein Münchner Unternehmen, das 2017 gegründet wurde. Ihre innovative LiDAR-Lösung kombiniert LiDAR-Hardware und -Software. Die kompakten und vielseitigen Sensoren bieten die beste Kombination aus Leistung, Zuverlässigkeit und Preis. Ein Schlüsselfaktor, der dieses Produkt vorantreibt, ist die Software. Ein Kunde, der ein LiDAR sucht, möchte eines, das sich leicht in seinen Arbeitsablauf integrieren lässt und keine umfangreiche Schulung erfordert. Da die Funktionen und die dafür erforderlichen Dienste regelmäßig erweitert werden, muss die Schnittstelle mit diesen Veränderungen Schritt halten können.

Eine der interessanten neuen Funktionen, die hinzugefügt werden sollen, ist die Möglichkeit, dass mehrere LiDAR-Geräte ihre Ergebnisse zu einer konsolidierten Punktwolke zusammenführen können. Eine der wichtigsten Voraussetzungen für diese Funktion ist, dass die Geräte als dezentrales Netzwerk arbeiten. Sie müssen in der Lage sein, andere Geräte im Netz zu erkennen und mit ihnen zu kommunizieren. Dies bringt uns zum Ziel dieser Arbeit, ein System zur Konfigurationssynchronisation für einen Cluster von LiDAR-Geräten und ein dazugehöriges modulares WebGUI für die neue Generation von LiDARs zu entwerfen und zu implementieren. Das Ziel des Webinterfaces ist es, modular und skalierbar zu sein. Die Modularität wird dazu beitragen, die Schnittstelle an die individuellen Bedürfnisse eines Kunden anzupassen.

Contents

Acknowledgments **iii**

Abstract **iv**

Kurzfassung **v**

1 Introduction **1**
1.1 Context for project . . . 1
1.1.1 Rotor based mechanical LiDAR . . . 2
1.1.2 Solid-state LiDAR . . . 3
1.2 Motivation . . . 3
1.3 Problem statement . . . 5
1.3.1 Configuration synchronization . . . 5
1.3.2 WebGUI . . . 6
1.4 Thesis structure . . . 6

2 Overview **7**
2.1 Blickfeld Qb2 . . . 7
2.2 Qb2 architecture . . . 8

3 Background **9**
3.1 Network discovery . . . 9
3.1.1 Zero configuration networking . . . 9
3.1.2 Multicast Domain Name System . . . 9
3.2 Distributed key-value storage . . . 10
3.3 grpc Remote Procedure Call . . . 14
3.4 Related works . . . 17

4 Design **19**
4.1 Swarm concept . . . 19
4.1.1 Swarm core features . . . 20
4.1.2 Swarm rules . . . 21
4.1.3 Swarm applications . . . 21

4.2 Swarm architecture 23
4.2.1 Configuration service 25
4.3 Swarm workflows 27
4.3.1 Create Swarm 27
4.3.2 Join Swarm or add device(s) 28
4.3.3 Configuration conflict 30
4.4 Swarm user interface design 30
4.4.1 Swarm dashboard 31
4.4.2 Registration process 32
4.5 Web-GUI 33
4.5.1 Design goals 34
4.5.2 Design process 34
4.5.3 Module structure 36

5 Implementation **39**
5.1 Swarm 39
5.1.1 Discovery service 40
5.1.2 Configuration service 40
5.1.3 Results 41
5.2 Web-GUI 43
5.2.1 Technology stack 43
5.2.2 Websocket gRPC proxy 44
5.2.3 Routing 46
5.2.4 Dynamic components 47
5.2.5 Results 50
5.3 Test setup 51

6 Summary and Conclusion **53**

List of Figures **55**

Listings **57**

Bibliography **58**

1 Introduction

1.1 Context for project

LiDAR stands for Light Detection and Ranging. The core principle is not unlike that of RADAR (Radio Detection and Ranging). In both cases, the distance to an object is calculated based on time of travel, the difference being medium [OA21]. A RADAR is based on radio waves, whereas a LiDAR uses laser pulses. LiDAR operates on a time-of-flight concept. We know the speed at which light can travel in our atmosphere, based on this we can figure out the distance based on the time taken for the emitted laser pulse to reflect from the object and reach the receiver; this is the fundamental principle of how a LiDAR works.

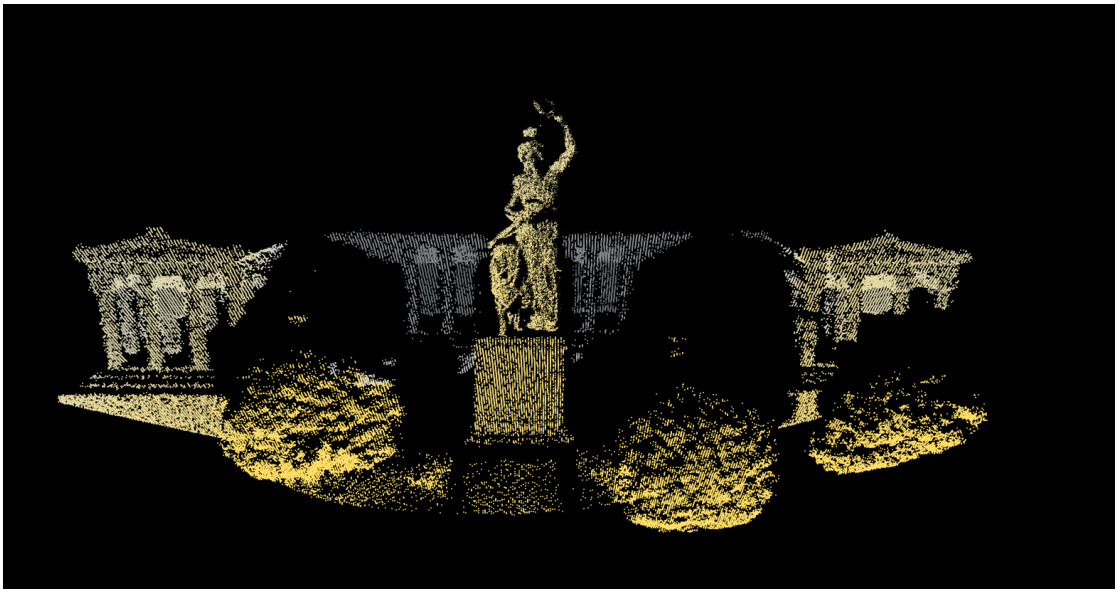

Figure 1.1: An example of 3D LiDAR point cloud.

Every day new and novel applications for LiDARs are discovered. We are witnessing a huge growth of automation in the industry with massive advancements in robotics. A key driving factor for this could be the pandemic and the shortage of labor it induced.

If we take a look at the logistics industry, we see that they are not new to the concept of automation and robotics. The industry where efficiency is the key has relied on scanners and conveyer belts for decades [Sab22]. But these are robots that work in a controlled environment that is under constant monitoring. Controlling the environment in which a robot works is not always a feasible solution. For a robot to function in a dynamic environment in close proximity to humans, they have to be able to perceive their environment accurately. The prime factor that governs the operation of a robot in the real world would be its ability to perceive its surroundings accurately, with little to no delay. LiDARs would be ideal for this situation as it provides the robot with the ability to sense depth. Modern LiDARs are capable of capturing 3D information in great detail from the surroundings [RB19].

The current leading LiDAR systems are beam steering sensors that make use of a rotor to scan the entire environment and solid-state beam steering sensors that do not require a rotor [RCG22].

1.1.1 Rotor based mechanical LiDAR

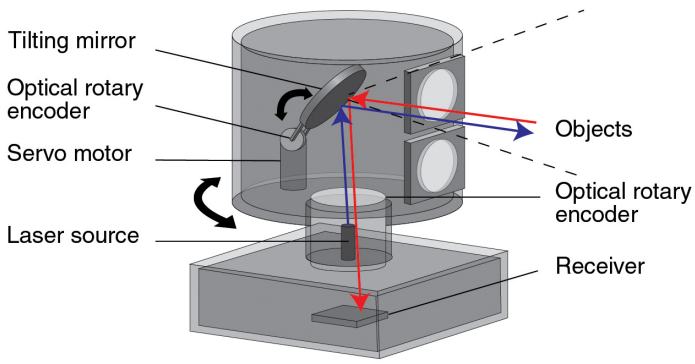

Figure 1.2: Diagram of rotor based mechanical LiDAR [HW92].

Rotor-based LiDARs are currently the preferred type of imaging sensor in the automotive industry [Mot20]. the construction of the rotor-based LiDAR uses high-grade optics that sits inside a rotating assembly. This spinning assembly is key for the 360° horizontal field of view. This amazing benefit, although comes with some caveats. This type of LiDAR is usually very large and unwieldy [Kha+21]. The components needed for this type of LiDAR to perform optimally need to be tested repeatedly and

continuously monitored for maintenance. Due to its size, it is also quite challenging to retrofit existing systems and limits the possible use cases.

1.1.2 Solid-state LiDAR

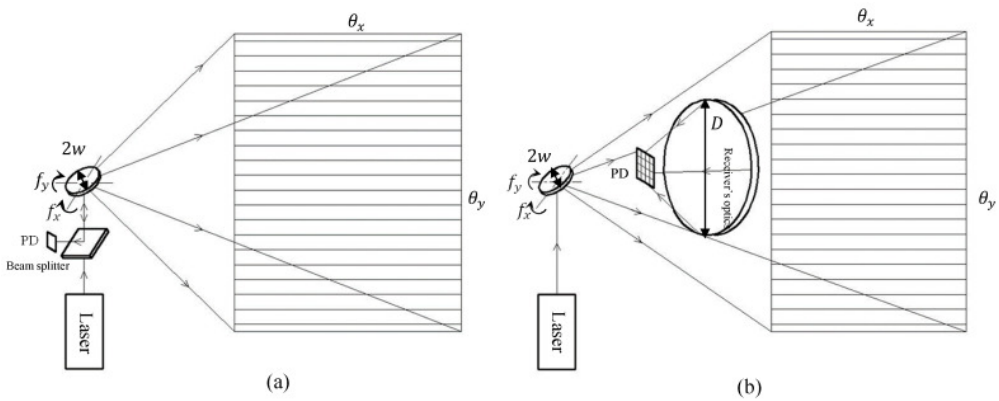

Figure 1.3: Diagram of solid-state LiDAR [WWX20].

Solid state LiDARs, on the other hand, have no need for spinning rotors. They employ a Micro Electro Mechanical system(MEMS). This type of LiDAR makes use of tiny electromechanical mirrors that oscillate at a high frequency. MEMS-based scanning mirrors basically have two main categories, resonant and non-resonant MEMS mirrors. The factor that differentiates them is the frequency they operate at with reference to their mechanical mode [Kha+21] [YC18]. By eliminating the bulky rotors and the housing needed to house that, the solid-state LiDAR occupies a much smaller footprint.

However, compared to the rotating mechanical LiDAR the solid state MEMS bases LiDAR has a limited field of view. To tackle this problem, we can use multiple sensors [YC18]. We can combine the output of multiple sensors and create a consolidated 3D render of the surroundings with a large field of view. Solid state LiDARs are much cheaper compared to their rotating mechanical counterparts when scaled.

1.2 Motivation

The explanations above provide us with a key shortcoming of solid state LiDAR devices over rotor-based mechanical LiDARs. Solid state LiDARs have a limited field of view. However, this issue can be tackled by combining the data streams from multiple distributed LiDARs. The fact that solid-state LiDARs are inexpensive helps in creating

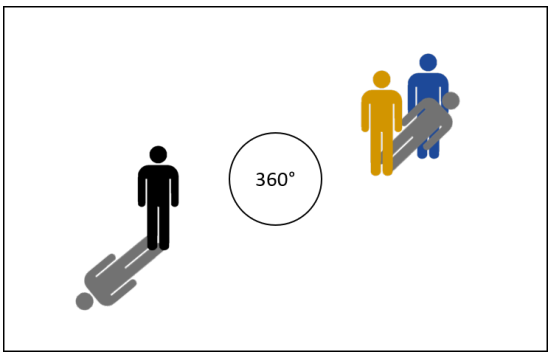

Figure 1.4: Illustration of rotor based lidar deployed in a room.

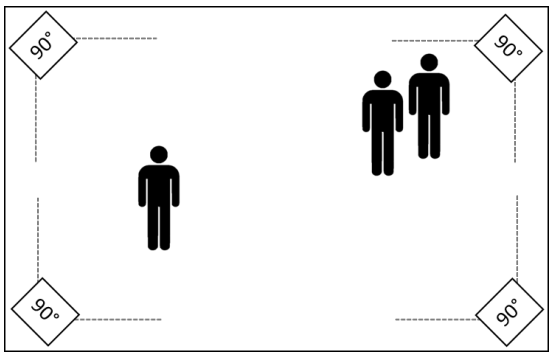

Figure 1.5: Illustration of solid state LiDARs deployed in a room.

a distributed network that is capable of adapting to different types of environments. Let's take a basic example of getting the 3D point cloud of a room.

In figure 1.4, an illustration of a rotor-based mechanical LiDAR, which has a 360-degree field of view, is placed in the middle of the room. This is the optimal positioning for a LiDAR of this type to capture the maximum possible data from its environment. The issue with such a LiDAR placement is that the objects close to the device could cast a shadow over the objects behind it, hiding parts of the object. This complicates the object detection process and could lead to misclassification of the object. For example, if we aim to calculate the number of people in a room like 1.4, we can immediately see an issue in the scenario of the person represented by yellow and the person represented by blue. Yellow will cast a shadow on blue as it is closer to the LiDAR. The amount of shadow cast would depend entirely on their distance from one another. In this scenario, the object detection might have issues classifying blue as a person, thus affecting the count.

In figure 1.5, we have placed solid-state LiDARs in the corners of the room. Even though we are using multiple LiDARs, the cost would still be significantly lesser than a rotor-based mechanical LiDAR. The data from these LiDARs can be taken and combined in a single 3D point cloud to provide us with more detailed data. This can help improve the object detection process significantly because, there would be little to no shadows. This method of gathering the 3D information of an area using multiple solid-state LiDARs and combining is endlessly flexible and can be adapted to a variety of situations.

Distributed LiDARs

A distributed LiDAR system is a cluster of LiDAR devices that operate in a network with constant communication with each other to share the processing load and to generate a consolidated point could. There is no central device that dictates what each LiDAR must do or collect data. Each LiDAR that is part of a cluster must be able to detect the other members of the cluster. A distributed approach is favorable for avoiding a single point of failure. In a distributed system of LiDARs, even if one fails or is generating noisy data, the others would continue to operate without any effect.

This system of a cluster of LiDARs is called a 'Swarm' in Blickfeld. In this thesis, the term Swarm will be used for a cluster of LiDARs.

1.3 Problem statement

This thesis will cover the design of the Swarm, the concept, the architecture, the components needed, and the complete user flow. For the implementation, we will be covering the configuration synchronization mechanism and the modular web interface that controls the LiDARs.

1.3.1 Configuration synchronization

The aim of the configuration synchronization component is to maintain the homogeneity of the LiDARs in the cluster by making sure that all the LiDARs have the same configuration and also detect changes in configuration and replicate them to all members of the cluster. To achieve this, first, a device discovery method needs to be created. This service is needed to search the network for all Blickfeld LiDAR devices. This service can then be used to determine if the device is an ordinary device or if it belongs to a Swarm. In the case of a scenario where the network has multiple Swarms, this can help in determining which Swarm a device belongs to.

When the user wants to create or modify a Swarm, the configuration process would first use the discovery service to gather the information of the devices and swarms on the network. If the user wants to create a Swarm by combining two devices, the configuration synchronization service would first verify if the devices are already the part of another Swarm. If both the devices are Swarm capable the configuration synchronization service will then compare the configurations that are currently active on the devices. If it detects any mismatch it raises a configuration conflict, which the user will have to resolve. Once a common configuration is set, the service would then update the properties of both the devices so they show up as Swarm devices on the network.

1.3.2 WebGUI

The interface to control the LiDAR is a web-based interface that is loaded onto the LiDAR devices. It is a part of the device's firmware. The design of the WebGUI focuses on modularity and lightness. In the thesis, the design process, architecture, and implementation of the interface will be discussed. The aim of the interface is to be as lean and flexible as possible. The flexibility is achieved by the modular design. Modules can be activated and deactivated based on the requirements provided by a customer. There is a focus on making components that can adapt dynamically or be retrofitted with ease to fulfill the requirement. The thesis will also contain some steps that were taken to improve the overall speed of prototyping and development.

1.4 Thesis structure

Introduction Covers the context of the project and the problem statement of the research.

Overview It gives an understanding of the Qb2 device made by Blickfeld GmbH; it gives a general idea of how the device functions.

Background An in-depth understanding of the technologies researched and used in this project, along with the related technologies considered for this project.

Design The first part covers the introduction of the concept of Swarm (cluster of LiDAR devices), followed by its features, design, and architecture. The second part covers the design process of the WebGUI.

Implementation This section covers the initial implementation of the configuration synchronization service, which the Swarm module would use, and the implementation of the modular WebGUI.

Summary and Conclusion Provides a summary, a conclusion for the thesis, and some ideas for future implementation.

2 Overview

This chapter provides insight into the Blickfeld Qb2 device, the capabilities of the device, its applications, and a brief description of the major components responsible for the device's function.

2.1 Blickfeld Qb2

Figure 2.1: Blickfeld Qb2 [Gmba].

Qb2 is an integrated intelligent solid-state LiDAR that allows capturing and processing of 3D data on a single device. Blickfeld aims to provide its customers with an intuitive and cost cost-effective system while maintaining the simplicity of installation and operation. The device has been designed with an emphasis on flexibility and scalability, so it can be adapted to various unique use-cases the customer might have. The device has multiple applications, such as smart infrastructure, crowd analytics, and perimeter security, without requiring additional computers or edge computing devices. A very high level of scene information is gathered, information like the position of an object, the velocity it is traveling at, and the number of objects that may be occupying a user-defined zone. All such information is accessible to the user in the form of an object or in the form of event lists. [Gmba]

On-device processing

The Qb2 uses a combination of light detection and ranging technology combined with powerful processing hardware that can be used to run a suite of perception applications.

Easy of use

To interact, control and manage a Qb2 device, an intuitive web interface has been developed; this enables users to modify the device's configuration like scan pattern configuration and actions pertaining to perception. The output data, point cloud, can be easily viewed on any web browser. It does not require the use of special hardware or installation of any custom software.

Software defined LiDAR

Blickfeld LiDARs are flexible and adaptable to a variety of use cases. The sensor features a frame-to-frame configuration of the field-of-view, point density and size, the number of scan lines, and scan rate. All these configuration points can be modified with little to no effort, thanks to the intuitive web GUI.

2.2 Qb2 architecture

The aim of software development at Blickfeld GmbH is the flexibility and modularity of the product. Therefore the goal is to create several modules that will have their own interfaces and behave like a microservice.

The primary components that run the device are:

Server The server provides the endpoints for all internal and external communications. It includes the WebSocket that enables the communication of the web interface to the device. The Inter Process Communication (IPC) that occurs between the core modules.

Core processing The core processing is involved with the pre-processing of the large datasets of Field Programmable Gate Array (FPGA) and Inertial Measurement Unit (IMU). The result of this core processing is passed on to other modules.

Edge processing This module handles the on-device processing of algorithms for tasks like object detection and the ROS2 service node for generating point clouds.

System controller The module is focused on device control and maintenance functionality. It handles processes like firmware updates, network configuration, time synchronization, device diagnostics, and recovery.

3 Background

This chapter of the thesis provides detailed information about the technologies that have been used to design and implement the project. The related works segment also explains the technologies that were taken into consideration.

3.1 Network discovery

Network discovery is a process that enables devices to detect other devices on the same network. This process is the first step in establishing communication between LiDARs on a physical network.

3.1.1 Zero configuration networking

Zero configuration networking or Zeroconf allows the automatic creation of a network of devices without the requirement for the manual configuration of a DHCP server, DNS services, or network settings for each of the devices that you wish to connect to that network [IBM21].

A great example of zeroconf is Apple's Bonjour. It's a zero-configuration networking application developed by Apple, and it allows the automatic creation of a network of devices in which the hosts and services can detect and establish a connection to one another without requiring any manual user configuration. Each device has a service that is broadcasted on the network, the other devices on the network can look for the specific service they wish to establish a connection with [IBM21].

Zeroconf utilizes the multicast networking concept to attain the goal of eliminating configurations in service discovery [Lee+07]. A network can have multiple LiDARs, each with its own IP. For the Swarm to work, the devices must be able to access services provided by another LiDAR without the user needing to configure each device. For the particular use case of the Swarm, python-zeroconf was used.

3.1.2 Multicast Domain Name System

Multicast Domain Name System (mDNS) is a service designed to help with name resolution on smaller networks. mDNS uses a different path than known DNS services.

Instead of sending a request to a defined name server, all participants in the network are addressed directly.[Wal21]

3.2 Distributed key-value storage

A distributed key-value store is a go-to solution for handling data exchange in distributed device networks. It is built to run on multiple systems working together, thus enabling the capability to work with large data sets. This enables the data to be distributed among more servers with more memory. By distributing the data storage across multiple servers, you can increase processing performance and significantly improve fault tolerance [Haz21]. For the scope of this project, etcd was chosen as the most promising distributed key-value storage method.

etcd

etcd is a consistent distributed key-value store. Mainly used as a separate coordination service in distributed systems. It is designed as a common layer for large-scale distributed systems. etcd stores metadata in a consistent and fault-tolerant way. The main advantages of etcd are the quality of reliability, scalability, and performance [etca]. Distributed systems can use etcd as a consistent key-value store for configuration management, service discovery, and workload distribution. Organizations use etcd for creating production systems such as container schedulers, discovery services, and distributed data storage. Some common distributed patterns that use etcd include leader election, distributed locks, and monitoring the health of a system [etca]. etcd uses a grpc Remote Procedure Call (gRPC) based API for its messaging protocol [grpb].

To maintain a steady state of a distributed system, etcd, makes use of the Raft consensus algorithm. Raft is a consensus algorithm, and it was designed with the idea of simplicity at its core. In distributed systems where you want to maintain fault tolerance, the consensus is a big problem. Consensus involves comparison and agreement of values among all the servers involved in the distributed system. For consensus algorithms to work optimally, they need most of their servers to be active. They also need a minimum number of servers to function at all. This minimum number or quorum is denoted by a simple expression.

$$quorum = \frac{n+1}{2}$$

In a cluster that has three servers, a maximum of one failure is acceptable; for a cluster of five servers, a maximum of two failures are tolerable. If the number of failures

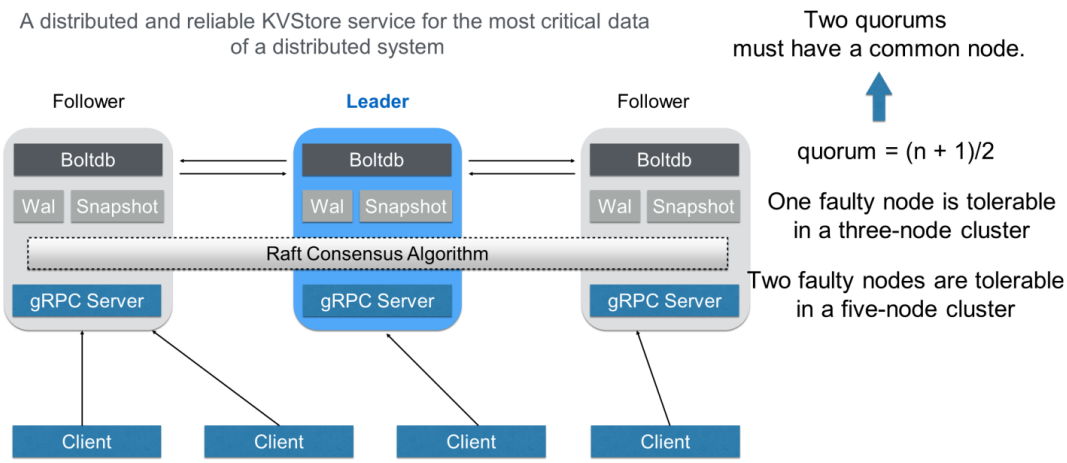

Figure 3.1: etcd architecture. [Nat]

exceeds the required quorum, there won't be any more progress, and no result will be returned. [OOb]

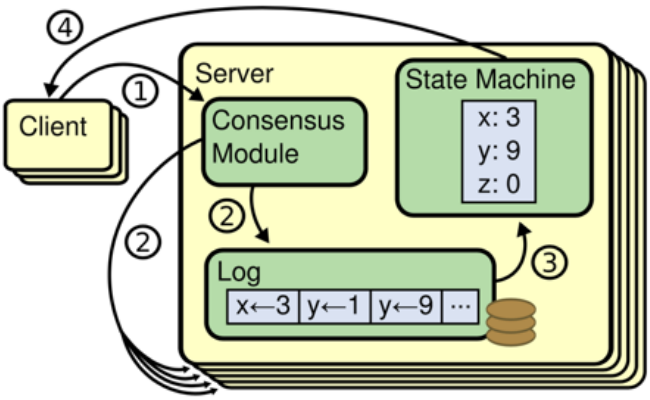

Figure 3.2: Replicated state machine. [Sof]

Building a fault-tolerant system requires the use of replicated state machines. Each server in this scenario has a state machine and a log. The state machine is the one that needs to be made fault-tolerant, in this case, a key-value store. For all intents and purposes, the user will interact with the system as if they were operating a single entity, a single state machine. Every state machine takes instructions from its log. In our key-value store, for example, if we want to set the vertical scan angle of our scan

pattern to ninety-three degrees. The consensus algorithm will be used here to decide and synchronize the command in the logs. The consensus algorithm is then responsible for making sure that all the state machines perform the operations in the order that has been decided by consensus. This makes sure that all the state machines perform all the commands in exactly the same order as the other state machines in the cluster.

The Raft algorithm is equivalent to the Paxos algorithm in the properties of fault tolerance and performance. In the Raft algorithm, the first step is to elect a leader from the devices in the cluster. This leader will then become the primary controller of the replicated log. The leader will accept any changes made by the user and update them onto the other state machines in the cluster. The leader has the authority to modify the replicated log without any prior communication or acknowledgment from the other members of the cluster. The hierarchy is clear, and data always flows from the leader to the other members. But, a server that is a leader could fail. In this scenario, a new election for a leader is conducted among the remaining devices [OOb].

The Raft algorithm can be broken down into three components:

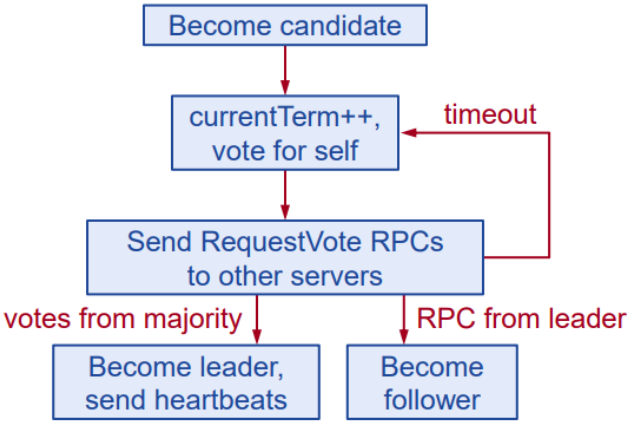

Figure 3.3: Leader election process. [OOa]

Leader election The leader election process flow is shown in figure 3.3 The leader election process should always be run whenever there is no current leader in a cluster or when an elected leader fails.

Log replication The leader must actively maintain the replicated logs and push the changes to the other members of the cluster.

Safety All servers must have the same order of commands in their logs, and there must be no difference in the order.

As mentioned in the Swarm requirements, the critical factor for the proper functioning of a Swarm is consistency.

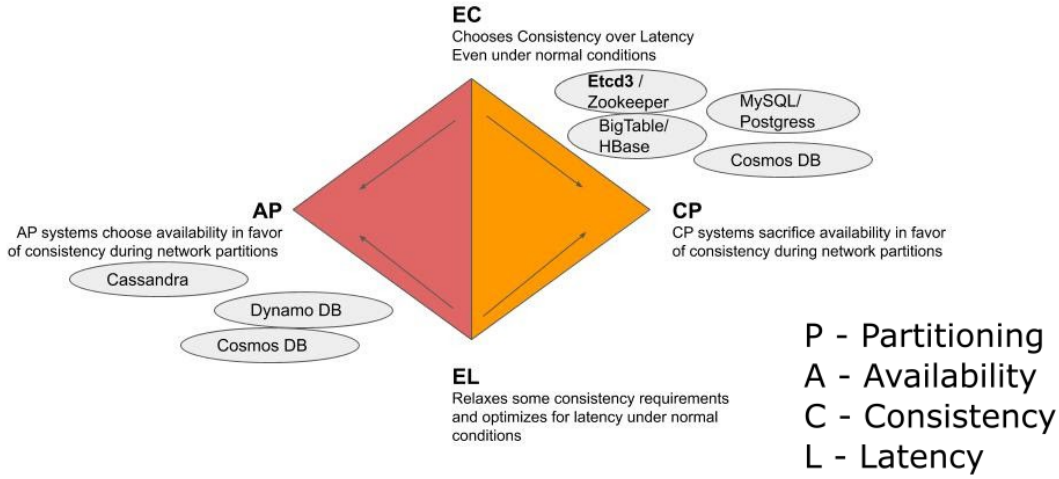

Figure 3.4: PACELC representation of key-value storage technologies. [OOa]

PACELC theorem is an extension of the CAP theorem [Fin21]. It is used to describe the tradeoffs made in a distributed system when thinking about partitioning, availability, consistency, and latency [Aba12]. In figure 3.4 we can see the way different key-value storage systems have prioritized the factors that are important to them. The reason for selecting etcd for this project is its priority for consistency.

The etcd API provides a set of guarantees.

Completeness An etcd operation will only be considered complete if it is committed through consensus. It must be logged in the storage engine as 'executed.' The client will know the operation has competed only when it receives a response from the etcd server.

Atomicity This means that an operation has either completely succeeded or completely failed; there is no middle ground. All events generated by one operation will be in one watch response for watch requests. The watch will never observe partial events for a single operation.

Strict serializability It is the strongest isolation guarantee of distributed transactional database systems. Read operations will never observe any intermediate data.

Linearizability etcd ensures linearizability; the user perceives each operation applied by concurrent processes takes effect instantaneously [HW90]. Linear requests use the raft consensus process.

There are a set of reasons that made etcd a suitable choice for a distributed key-value storage system. The Qb2 uses gRPC as a means for backend communication; it follows the client-server architecture, which maintains the integrity of the messages. etcd also uses gRPC API as a method of communication; this removes the need to bring in another communication stack [etcb].

3.3 grpc Remote Procedure Call

gRPC is a modern open source high-performance Remote Procedure Call (RPC) framework that was designed to run in any environment. It was initially developed by Google. gRPC is an efficient connection and load balancing interface for distributed computing systems. It has additional plugins for tracking, health checking, and authentication. It is also capable of connecting distributed systems to their backend services. gRPC uses protocol buffers to process the data and uses HTTP/2 to transport streams between remote computers. [grpb]

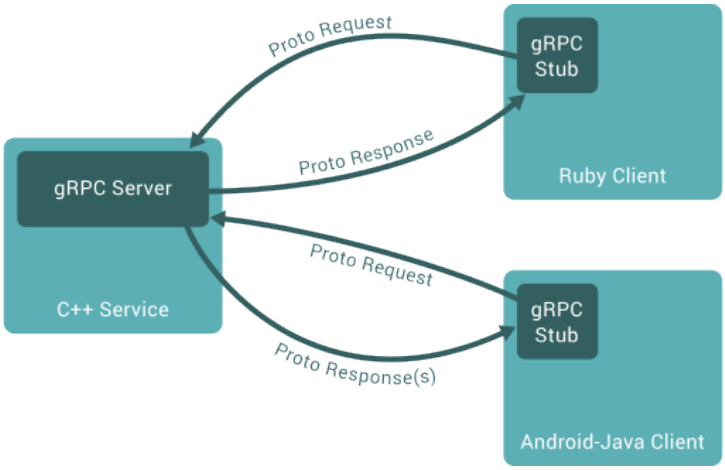

Figure 3.5: GRPC architecture. [grpb]

RPC and gRPC both are based on service definition. Methods can be called using their parameters and return types. gRPC uses protocol buffer to describe both the service interface as well as the structure of the payload.[grpa]

The main API of the Qb2 is based on gRPC and HTTP/2. The device instantiates several micro-services to serve multiple functions. It is used for high throughput

communication such as point cloud streams and for REST-like configuration. A proxy server based on Nginx distributes the service requests according to their request name and namespace, which is transported via the HTTP/2 header. As gRPC is binary encoded, the Protobuf schema files are required to encode and decode the communication. The schema files are published for the individual modules internally or via Blickfeld Github repositories.

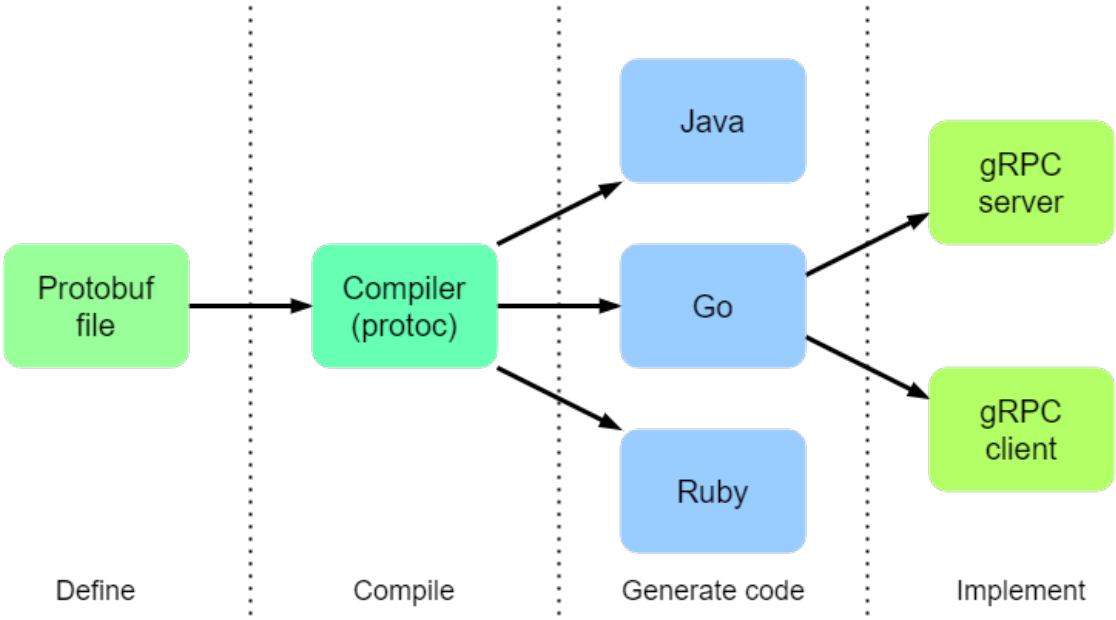

Figure 3.6: gRPC workflow. [Ion]

To understand the gRPC protocol, we need to look at its workflow, and its workflow is majorly divided into four steps [Ion]:

Definition of a service contract The definition of the services, the parameters, and their return types must be defined, which can be called remotely. This is required for IPC.

gRPC code generation Generation of gRPC code using the .proto file. A .proto file has some similarities with a JSON file. However, the difference being you can run a compile on the .proto file that can generate the gRPC code, which then enables the data to be read and written in any programming language [Ama].

Implementing the server A server needs to be implemented in the desired language.

Creating client stub This is responsible for calling the service. The client and server then take care of the request.

HTTP/2 protocol is key for efficient exchange of messages from gRPC and data. It also brings some additional benefits to the table, transfer of compact binary data and header compression that drastically reduces the volume of requests and responses on the network [Ion].

One of the core concepts of Remote Procedure Call (RPC) is the ability to stream data. With streaming, there is a continuous flow of data to the client with a single request needed from the client.

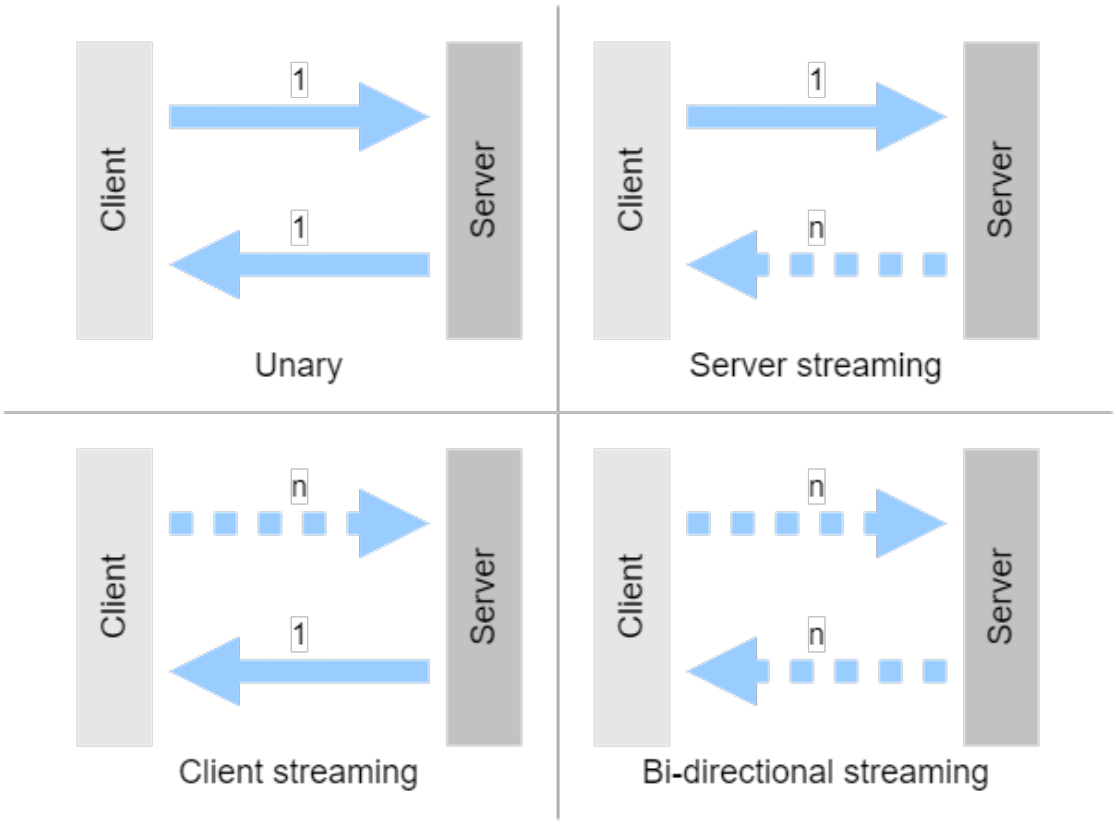

Figure 3.7: gRPC streaming. [Sin]

Server streaming RPC In server streaming, the client can send a single request and will continue to receive multiple responses from the server. For example, if a client access the website of a news provider. The page is bound to have multiple articles. If the client has to wait for all the articles to be loaded at once, it'll take

a very long time. In this case, server streaming can help by sending the data in small bundles after the initial request and keep loading more bundles in the background, so the user experience is not hindered.

Client streaming RPC Client streaming, as the name suggests, is when the client is sending multiple requests to a server, and the server only gives a single response. For example, if a client is uploading a large file, it is broken into chunks and uploaded one at a time. After receiving the final chunk, the server will respond with an acknowledgment.

Bi-directional streaming RPC In bi-directional streaming, both client and server are continuously sending each other a stream of data. This data is not dependent on each other, so they don't need or wait for a response from the target recipient.

There is also the unary RPC that does not involve any streaming; in this method, there is a single request sent by the client to the server, and the client then receives a singular response from the server.

3.4 Related works

There are a lot of technologies available in the distributed key-value storage market, and each has its particular set of advantages and disadvantages. In distributed network communication, there are a set of factors that govern the efficacy of a system. They are partitioning, availability, consistency, and latency. Unfortunately, no system can be good at all the features mentioned above; at most, they can excel in two factors.

The other distributed vital value storage services researched for this project are explained below.

Apache Zookeeper

Apache ZooKeeper is a centralized service used to maintain configuration information, naming, providing distributed synchronization, and providing group services. Implementation of Apache Zookeeper requires much maintenance as race conditions are a common occurrence. Apache ZooKeeper was the base inspiration from which etcd was developed, so naturally, it made improvements over the Apache zookeeper. [Zoo]

Considering the use case of maintaining information of a cluster with strong fault tolerance, ZooKeeper falls short of this goal. It lacks the multi-version concurrency control, making it unsafe for use in a multi LiDAR setup like Swarm, where multiple users could be interacting with the same Swarm.

In terms of reliability in proportion to the database size, Apache ZooKeeper is quite close to the reliability achieved by etcd.

A major shortcoming of Apache ZooKeeper is that it does not support linearizability, i.e., the order of operation is not maintained; this factor is of utmost importance for the Swarm as the efficiency of a Swarm depends on the consistency with which data is received [Ran].

Consul

Consul is a framework that provides end-to-end service discovery. It features built-in health checking, provides DNS services, and also supports failure detection. Consul also exposes a key-value store with RESTful HTTP APIs. However, this could have been used for this project, but the current version of the Consul storage system does not scale properly. A sudden surge in traffic could lead to high latencies and memory pressure. The lack of reliable streaming watches also poses a great disadvantage. [etca]

Consul and etcd are solving different types of problems for use as the end-to-end cluster service discovery Consul would be a better choice; however, for key-value storage with consistency, etcd would be the suitable choice.

4 Design

This section of the thesis covers the design of the configuration synchronization and the WebGUI interface. The design process and the deliberation behind the decisions are explained in detail. The section also introduces the concept of Swarm (cluster of LiDAR devices), the need for Swarm, the features it delivers, and the possible applications for such a feature.

4.1 Swarm concept

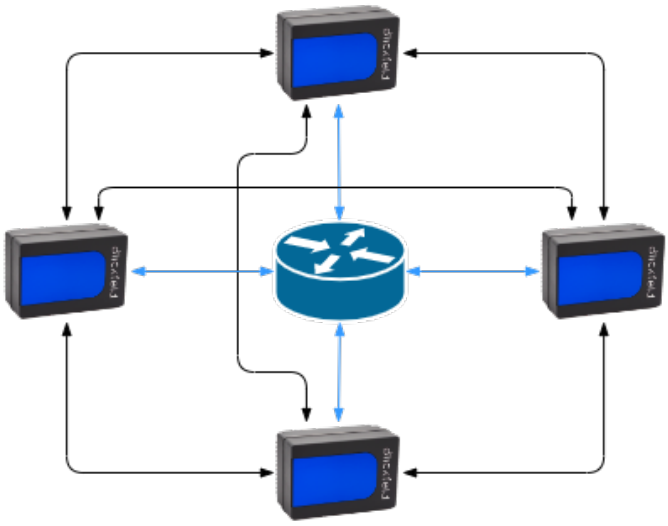

Figure 4.1: Swarm concept.

The concept of a distributed LiDAR system is called a Swarm at Blickfeld. A Swarm is a cluster of Qb2 devices installed on the same network at a physical location. They work as a cluster with constant intercommunication, generating a combined output.

4.1.1 Swarm core features

Registration The registration process is to update the properties of the cube with the Swarm identifier, and this will mark the cube as a Swarm device. This identifier can then be used by other devices in the Swarm to detect the device.

Alignment of point clouds The alignment process is required to identify the position of the device relative to other devices of the Swarm. This information is critical to creating a high-quality 3D point cloud, especially when the field of view of the devices might overlap. A miscalibrated system will lead to distortion at the point of overlap.

Configuration synchronization This process maintains the consistency of the data received from the devices. The configuration synchronization process is responsible for maintaining the homogeneity of the system. It makes sure that all the devices have the same scan pattern configuration as well as detects changes made to a single device and replicate those changes on all other devices in the Swarm. The configuration synchronization will also need to control the time and frame synchronization so that the resulting output will be clean enough for the object detection to work properly.

Distributed configuration storage Swarm provides a distributed configuration storage system. This is especially useful if, due to some reason, one of the devices in the Swarm is inactive; during that time, if a configuration change is performed, the inactive device won't receive the changes. When the device is powered back on, it can then simply replicate the changes found on the distributed configuration storage.

Maintenance Swarm also provides a great advantage in maintaining multiple devices simultaneously. In a scenario where a customer has multiple Qb2 devices, if they want to perform a firmware update, instead of doing it individually for all the devices, they only need to push the firmware update onto one device that is part of a Swarm. The device will then push the firmware update to all the other participants of the Swarm. The Swarm also helps in giving an overview of the current state and health of the Qb2 devices. If a device in a Swarm is having an issue like overheating, it will broadcast this information to the other devices on the Swarm. This will help the customer troubleshoot the issue in case the device fails. The Qb2 device has a system that monitors the health of the device. This system can let the customer know which particular component in the device is the cause of the failure. In case of a catastrophic failure, this diagnostically relevant information can be recovered from the other devices in the Swarm.

4.1.2 Swarm rules

These are a set of rules which need to be followed to achieve optimum output from the Swarm.

Number of devices The Swarm needs to have a minimum of two devices, and the current maximum number of members in a Swarm is six. This limitation of the maximum number of devices is due to the limited compute performance onboard the device. The planned method to overcome this limitation to increase the upper limit is the use of a perception processor. The perception processor will not be a LiDAR but rather a computer that has a much better computation power. The perception processor will mimic a LiDAR on the network, it can then be added to a Swarm. The device is simply meant to help in relieving the processing load on the other devices in the Swarm.

Positioning The physical positioning of the devices in a Swarm relative to each other must be extremely precise. The geometric pose of all the devices in a Swarm must be static. This relative positioning information of the devices is taken into account, as this will help in creating a consolidated point cloud.

Connection The connection between all the devices of a Swarm must be on the same physical network. This is needed to maintain a consistent flow of information. Peer-to-peer communication must take place with the lowest latency possible. A time server must be available on the network against which the devices will perform time synchronization.

Configuration All the devices in a Swarm must have the same configuration profile. Variation in configuration will lead to inconsistent data and a difference in the rate at which data is transmitted. A device, for this reason, can only be part of one Swarm.

4.1.3 Swarm applications

The Swarm functionality enables the consolidation of the 3D input from multiple LiDARs. This gives the Blickfeld perception system more data with which it can improve its object detection capabilities. This enables in improving the applications of Qb2 tenfold.

People counting For accurate people counting, the primary requirement is high-quality, consistent data. The shortcoming of using the data from a single LiDAR is that in scenarios with a higher number of people, there could be people casting a

shadow on other people, thus affecting the accuracy of the data. With Swarm, when multiple LiDARs are placed in an environment with overlapping field of view, we can combine the data from all the devices and then run the object recognition algorithm on the consolidated data. Virtual zones can be set up in the point cloud, and people entering these pre-defined zones can be counted and aggregated over time. The advantage of the Swarm is that these zones can be created on the consolidated point cloud data to cover larger areas. These statistics can then be analyzed by a customer to generate a forecast of the possible foot traffic in an area.

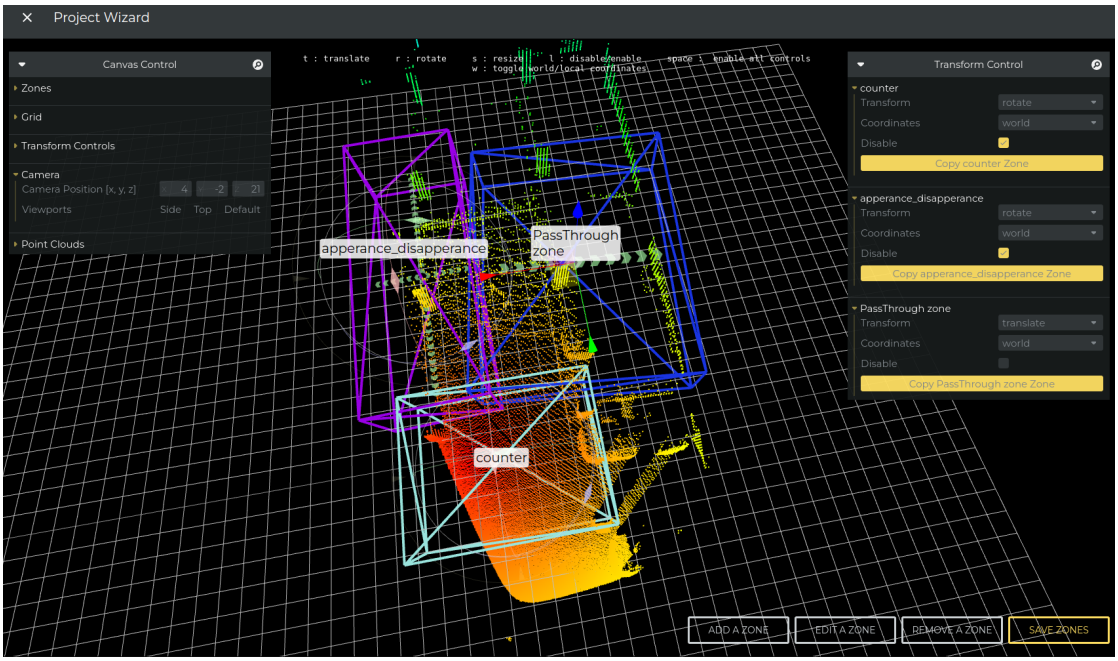

Figure 4.2: Blickfeld zone management system.

Perimeter security The same zone management system mentioned previously can be used to set up security zones. The zones can alert the customer in case a breach is detected. The idea is that object classification can be used to distinguish if the breach is a false positive, such as in the case an animal enters the zone. This could be useful for customers operating in a large secure area.

Volume measurement Volume measurements of containers or vehicles can be done. Real-time volume estimation of material not in a container can also be performed. The use cases include but are not limited to volume estimation of load on passing trucks and estimation of sand and gravel at a sand processing plant.

Traffic Management LiDARs mounted at traffic signals can be used for accurate traffic flow management and vehicle counting. For example, the Swarm setup can be done at a busy intersection to monitor the traffic flow and adjust the signal system accordingly.

Figure 4.2 shows the current zone management system for a singular Qb2 device. In the figure, bounding boxes that provide different functionalities are shown,

Counter In the counter zone, the zone white bounding box will count the number of objects in the zone. For example, this functionality can be used to get the number of people in a room or the number of cars in a parking lot.

Appear/Disappear This zone is depicted with the purple bounding box, showing if some object is present in the zone or if an object is no longer in the zone.

Pass through This zone is depicted using a blue bounding box, which is used for counting the objects that pass through the zone in a particular direction; the direction is shown with green colored arrows. This functionality can be used on a highway to calculate the number of vehicles that travel in a particular direction.

4.2 Swarm architecture

The Swarm architecture is shown in figure 4.3. Swarm is a non-hierarchical multi-master system. Swarm gRPC server delivers three service groups: discovery service, Swarm configuration service, and distributed firmware updates. In this thesis, we will be covering the discovery and configuration service.

Each device connects to the network using the TCP/IP stack. On top of that, there are three other stacks:

gRPC stack gRPC is used as the primary communication means between all the various microservices. All Remote Procedure Call (RPC) and Inter Process Communication (IPC) calls take place using gRPC.

Websocket stack The WebSocket is the bridge between the web-based interface and the individual devices.

zeroconf stack The primary function of the zeroconf stack is the detection of devices with a particular service name. This helps it distinguish Blickfeld LiDARs and Swarm among all the other devices that will be present on the network.

Swarm server

The Swarm server provides three services

Discovery service The discovery service is responsible for detecting all the devices in the network. It gathers information from the devices that can help establish whether they are part of a Swarm. This information will be used to present the user with a list of devices that they can use to create a Swarm and add the device to an existing Swarm. The discovery service can also provide information about any existing swarms on the network. The service makes use of;

- **Python zeroconf** Python zeroconf or pyzeroconf is a mDNS service discovery for python. The service enables discovery of services on the network.
- **Avahi** Avahi provides service discovery capabilities on a local network using the mDNS protocol. This enables the plug-and-play aspect of devices. Connecting to the network allows devices to communicate with other devices on the network without any pre-configuration.

Configuration service Configuration service manages the tasks related to maintaining the Swarm. It takes care of joining or un-joining devices from a Swarm by updating the Swarm identifiers of the devices so any other device on the network can access that information. The configuration service also helps in resolving any configuration conflicts that may occur during the creation of a Swarm or while adding a device to an existing Swarm. Maintaining the information of the device configuration and the configuration changes on the network, the configuration service uses a distributed key-value storage system. A distributed key-value store is critical in maintaining the information in a steady state in a distributed system. For this service, we make use of etcd. etcd is a distributed key-value store that was selected after comparing it with other key-value stores available. The tight integration with gRPC was a factor, but primarily the key requirement that it fulfilled was that it provided strong data consistency; these points are discussed in detail in the background section of this thesis.

Firmware service The firmware service will be responsible for a distributed firmware update. The goal of this service is to provide the user with the capability to update only one device on the Swarm, and then that device pushes the firmware to all the other members of the Swarm. The service will also provide the current status of the firmware update on all the devices and notify the user if all the devices have rebooted successfully after completing the firmware update.

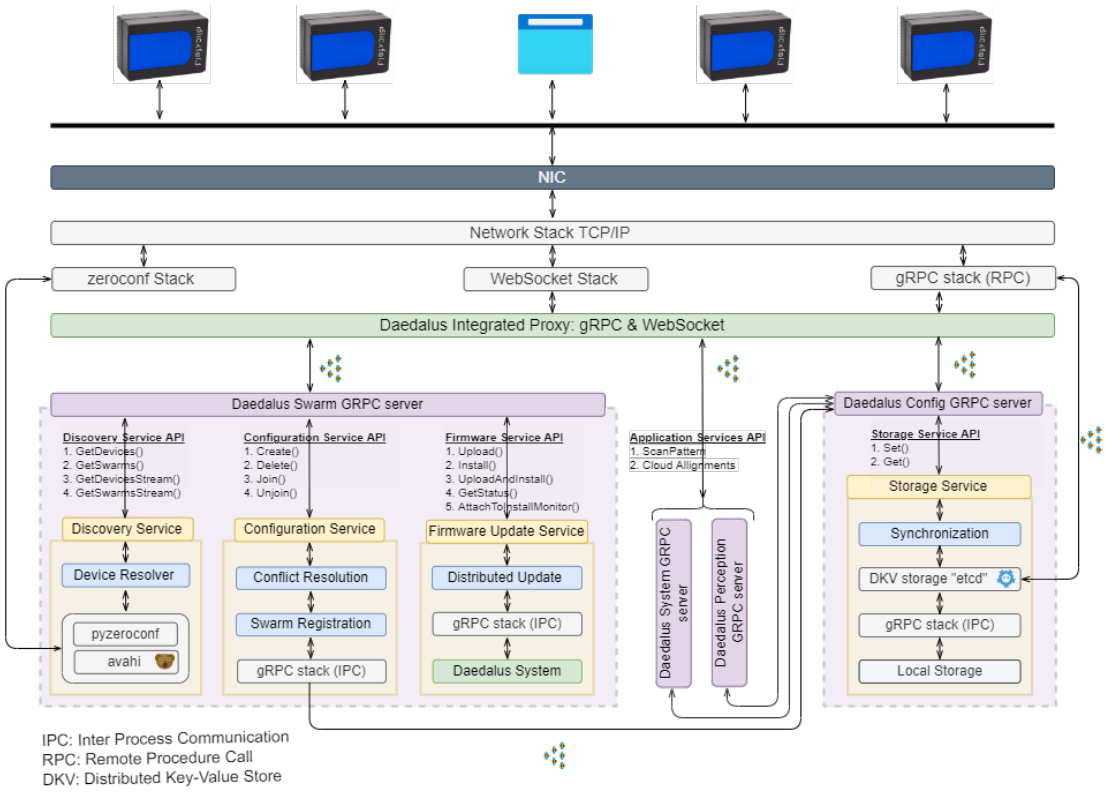

Figure 4.3: Swarm architecture.

4.2.1 Configuration service

Configuration service is a combination of processes that will occur when a user interacts with any functionality from the Swarm management section of the interface, like changing any scan pattern settings.

For the proper functioning of a Swarm, the configuration profiles of all the devices have to be identical. The reasoning behind identical configuration is to prevent inconsistency in data. The Swarm needs a consistent flow of data to function optimally. If one of the cubes is operating with a different scan pattern configuration, the rate at which the data is available would be different from other cubes in the Swarm. This data inconsistency will cause issues when combining the point cloud data of the LiDARs.

The factors affected by the configuration service are:

Swarm identifier A Swarm identifier is used to determine whether the device is part of a Swarm; also, in a network with multiple swarms, it helps to identify if the device is part of a Swarm.

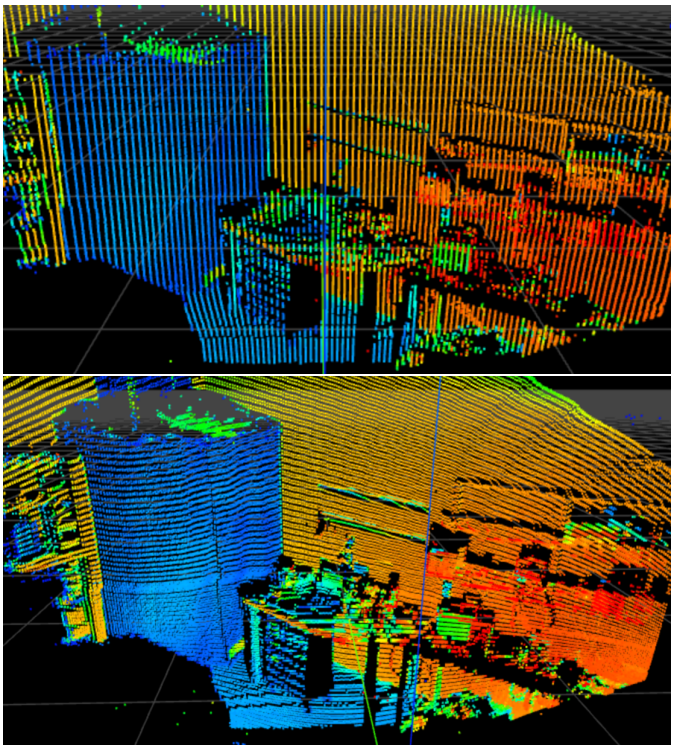

Figure 4.4: Comparison of different scan pattern configurations.

Distributed configuration storage The distributed configuration storage based on etcd will be used to store the active configuration of the Swarm. The members of a Swarm can reference this configuration to update their configuration profile.

Time synchronization Each data packet comes with a time stamp; this time stamp is used to synchronize the output from all the Swarm devices. The time synchronization service ensures that all the devices' clocks are synchronized.

Frame synchronization The frame synchronization process is responsible for making sure the frame rate of the device is constant and that the start of the frame is synchronized. This process inserts small pauses during the device scanning to achieve a constant rate. [Gmbb]

Scan pattern settings The scan pattern settings control the oscillation of the MEMS mirrors. The vertical and horizontal angles determine the field of view. These settings also control the resolution of data. Figure 4.4 shows the comparison between the point cloud data generated by a device with two different scan

pattern configurations. One configuration generates vertical scan lines while the other generates horizontal scan lines; like this in the case of Swarm, if data from two devices with different scan pattern configurations are combined, there will be a significant drop in quality at the points of overlap. The synchronization of these data points among the devices of a Swarm is critical.

Alignment The alignment process is required to establish the relative position of the Swarm devices to each other. This data is then used to place the output from the devices in a 3D point cloud. The alignment process will also allow the user to make fine adjustments in the case of overlapping fields of view to prevent distortion at the point of overlap.

4.3 Swarm workflows

The Swarm workflow has three distinct workflows create a Swarm, join Swarm or add a device and conflict resolution. The availability of these options will be controlled based on the rules of a Swarm. These processes can be accessed by the user in the Swarm section of the device.

4.3.1 Create Swarm

The Swarm creation process will be available for a device that is not part of any other Swarm. This limitation is based on the Swarm rules.

1. After selecting the create Swarm option; the user will be presented with a list of all the Swarm-capable devices on the network.
2. The user then selects the devices that they want to include in the Swarm. A minimum of two devices and a maximum of six devices.
3. The configuration profile and the firmware of the selected devices are then compared to detect differences. If a firmware conflict is detected, the user will be prompted to upload and install the appropriate firmware.
4. Following the firmware update, the device will have to reboot.
5. In case of a configuration conflict, the user is presented with the choice to select the device whose configuration profile they want to apply to the Swarm.
6. The selected configuration is then applied to all the other devices trying to join the Swarm.

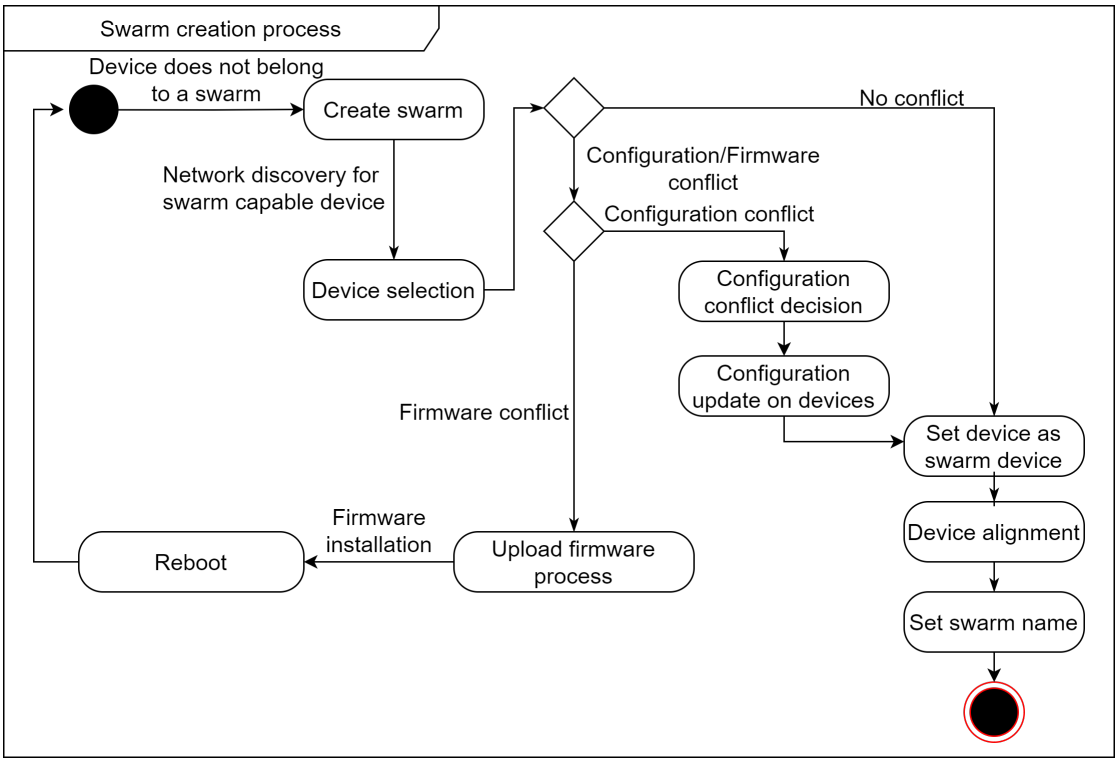

Figure 4.5: Swarm creation state diagram.

7. The Swarm identifier of the devices is then updated, allowing them to recognize the other members of the Swarm.

8. The final step is the alignment process. Here the user will first enter the relative positioning information of the devices. They can also fine-tune the point cloud at points of the overlapping field of view.

4.3.2 Join Swarm or add device(s)

The option to join a Swarm is available when the user is accessing a singular device. The add device option is available when the user's point of entry is the Swarm view. Both these workflows borrow a key process from the Swarm creation process, the conflict detection, and the resolution process.

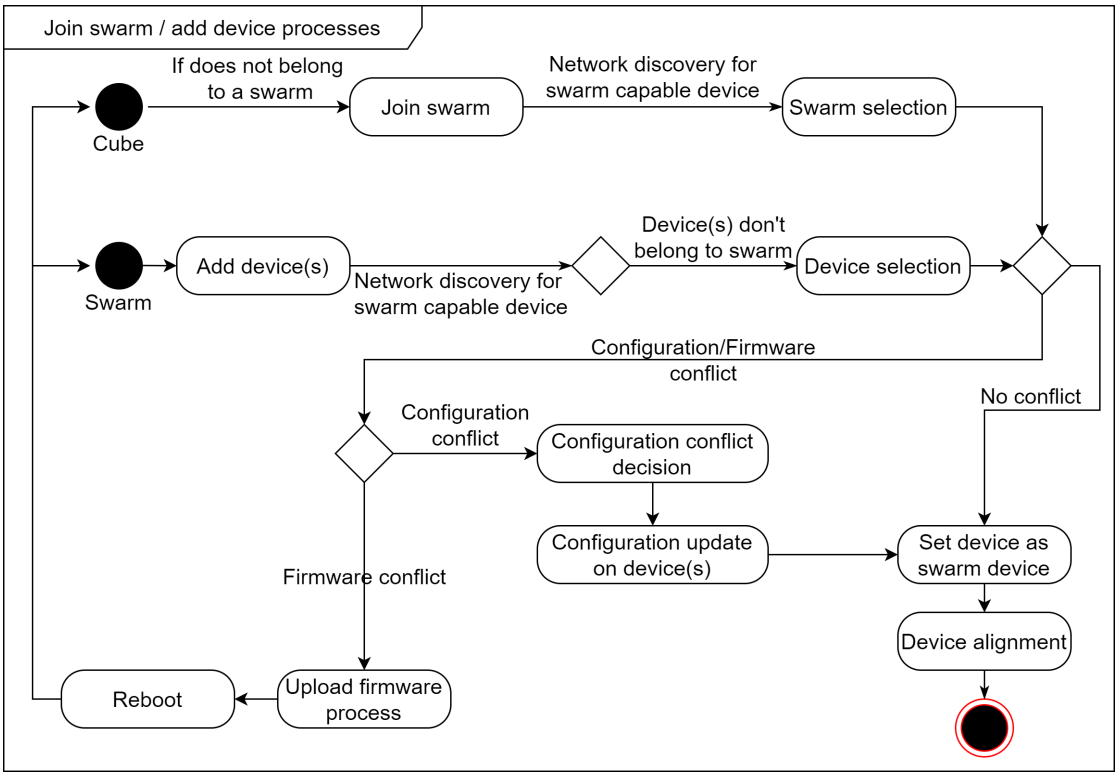

Figure 4.6: State diagram for joining Swarm or adding the device to Swarm.

Cube

1. The 'Join Swarm' option is only available if the device is not part of any other Swarm.
2. The user will be presented with a list of all the Swarms available on the network.
3. After the Swarm is selected, the conflict detection and resolution process will be carried out, asking the user about which configuration they want to apply to the Swarm.
4. After this step, the alignment process will take place.

Swarm

1. If the entry point of the user is from the Swarm view, they will have an option 'Add device(s)' to the Swarm.

2. The user will then be presented with a list of devices that are Swarm capable and do not belong to another Swarm.
3. The selected devices will then go through the conflict detection and alignment processes.

4.3.3 Configuration conflict

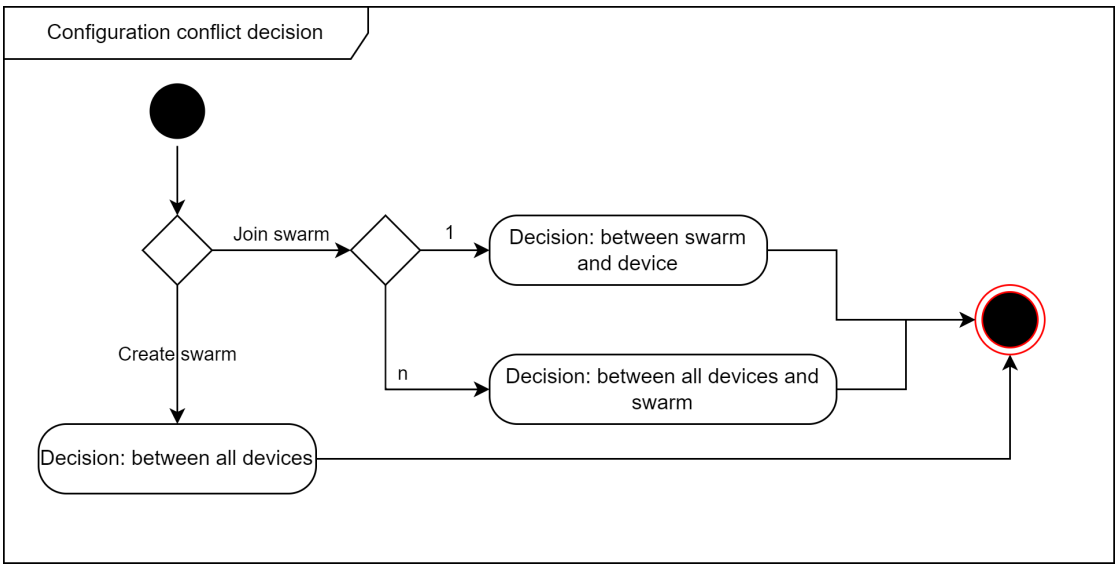

Figure 4.7: Configuration conflict state diagram.

During the above-mentioned workflows of creating a Swarm, joining a Swarm, and adding a device to an existing Swarm, there is a possibility that the configuration profile of the devices is not identical. Since homogeneity of the configurations is a requirement for a properly functioning Swarm, the difference is called a configuration conflict. When such a situation arises, the user will be presented with options to make a decision and select the configuration profile they want to proceed with.

4.4 Swarm user interface design

The following are the design renders for the user interface related to Swarm. The design is based on the components that have already been created and will be used to implement this module. The design of these components is explained in detail in

the Web-GUI section. These designs do not represent the final state; they are still being worked on and improved continuously.

4.4.1 Swarm dashboard

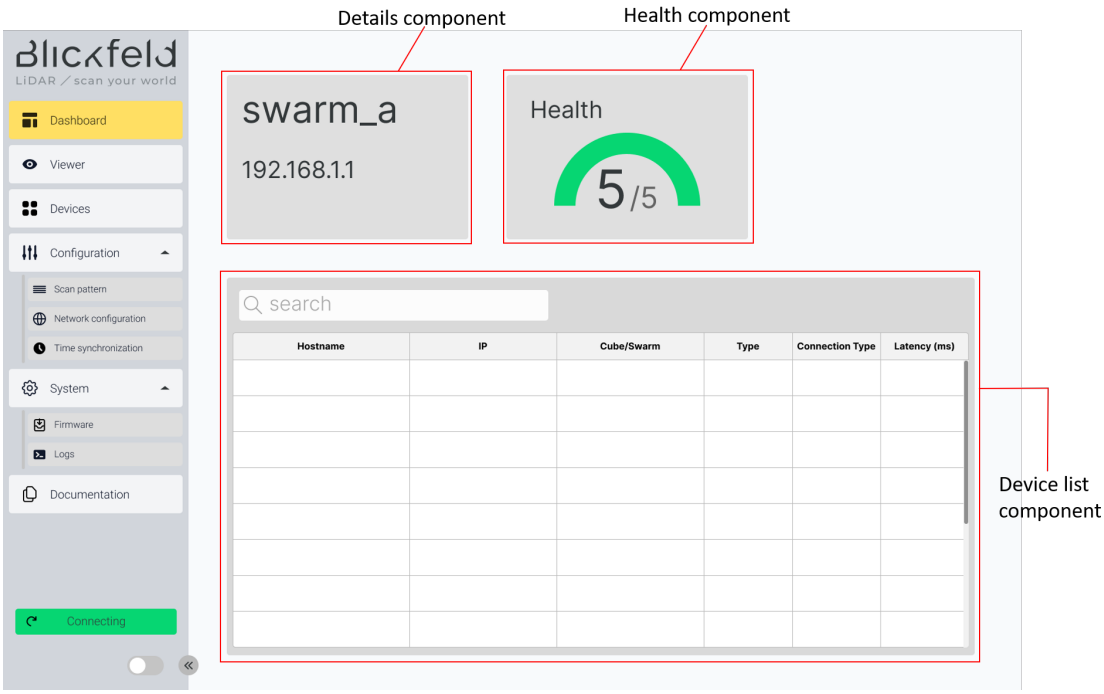

Figure 4.8: Swarm dashboard concept design.

The Swarm dashboard is shown in figure 4.8. This will be the landing page when the user enters the Swarm view. The dashboard is where the user can view the status of the Swarm at a glance. Here there will be three main components Swarm details, Swarm health, and device list component. These components will be ported from the user interface used for a single device. The information regarding the Swarm identifier, the name of the Swarm, and its IP address on the local network will be shown in the Swarm details component. The Swarm health component will be a more advanced version of the health component used for a single device, as this will have to show additional information such as latency issues or time synchronization errors. The number of active devices in the Swarm will be shown. In case of a failure of a device on the Swarm, the error will be shown here, and the user will be redirected to the diagnostic report.

4.4.2 Registration process

The registration process component is shown in figure 4.9. This component will be used when the user wants to create a Swarm or add or remove devices from it. The process has three main steps:

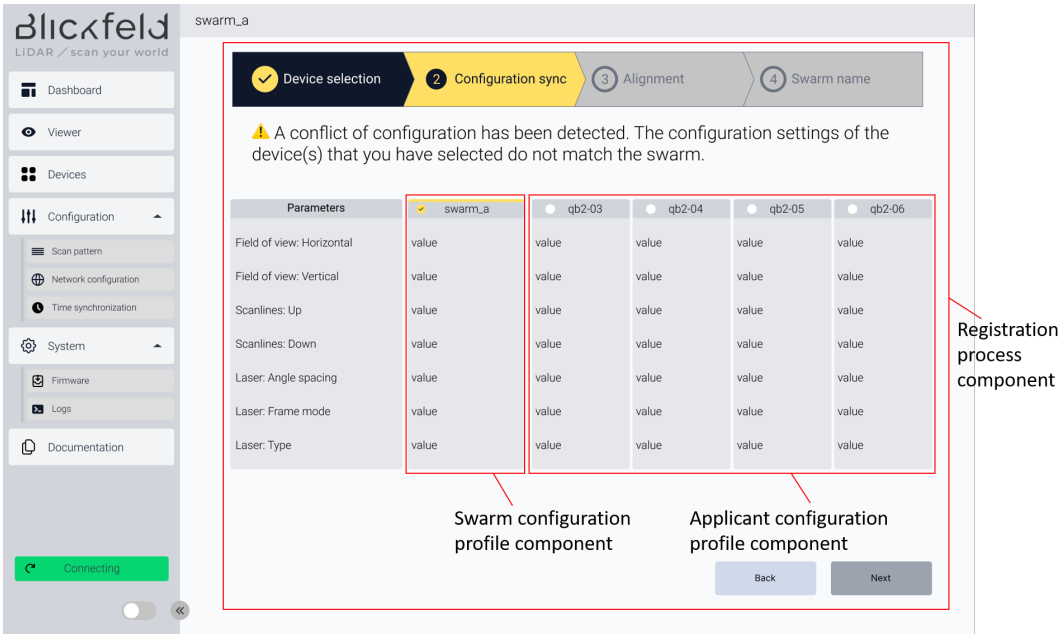

Figure 4.9: Configuration conflict resolution concept design.

Device or Swarm selection The user will be presented with the device list component shown in figure 4.8. The user has to then select the device(s) they would like to add to the Swarm, or they can select a Swarm that they want to add to the current device.

Configuration synchronization This page is dedicated to the resolution of configuration conflicts. It will only be shown if a difference in the configuration profiles has been found; else, it skips directly to the alignment process. Conflicts that arise due to different configuration profiles on devices will be presented to the user in this view. In figure 4.9 the user is trying to add four new devices, each with a different configuration profile to the Swarm. The user has to then choose to either continue with the configuration profile of the Swarm or select from one of the profiles from the devices attempting to join the Swarm. The selected profile will then be updated on all the devices.

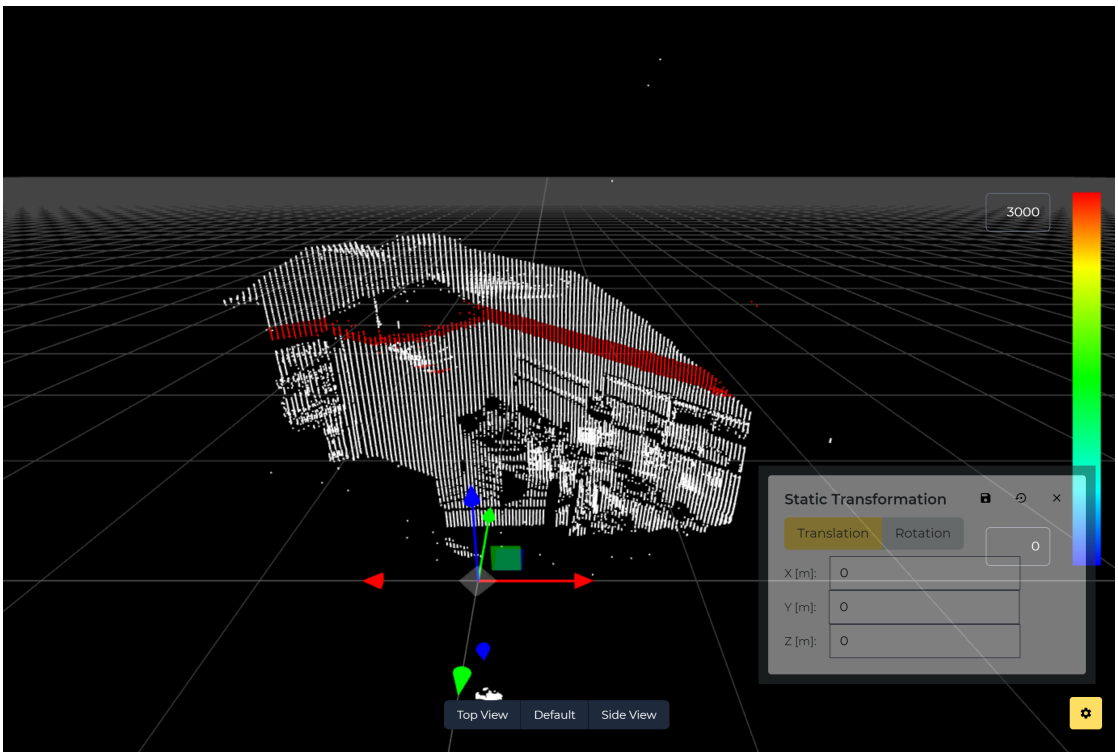

Figure 4.10: Alignment process for a singular cube.

Alignment The user will be presented with a live point cloud render. They can fine-tune the positioning of the devices here so that the point clouds of the different LiDAR overlap accurately. In figure 4.10 the current alignment process for an individual cube is shown. This component will be retrofitted to work with point cloud data from multiple sources so that they can be then aligned with reference to a common geometric plane.

4.5 Web-GUI

This section covers the steps and thought processes involved in designing the architecture, user experience flow, user interface, and the connecting bridge between the web interface and the Blickfeld Qb2 device. The web interface will be referred to as web-GUI in this thesis.

4.5.1 Design goals

The design goal was to construct a system architecture that is agile, something that can be easily scaled up or toned down based on the requirements presented by the clients. To achieve this, some critical steps need to be followed.

Modular structure A modular structure denotes the various large services that run on the device. The modules can be included and excluded from the final bundle by simply changing some parameters.

Development rules A rule set needs to be defined for the developers to follow. Such as the project structure, component structure, and good documentation practices. The rules are essential in a system that would be worked on by multiple developers. Not having these rules is what leads to poorly maintained code bases.

Plugin-based architecture This architecture is a design pattern used for flexibility, extensibility, and low-coupling of application features. Plugins can be added or removed from the core system anytime without affecting other plugins.

4.5.2 Design process

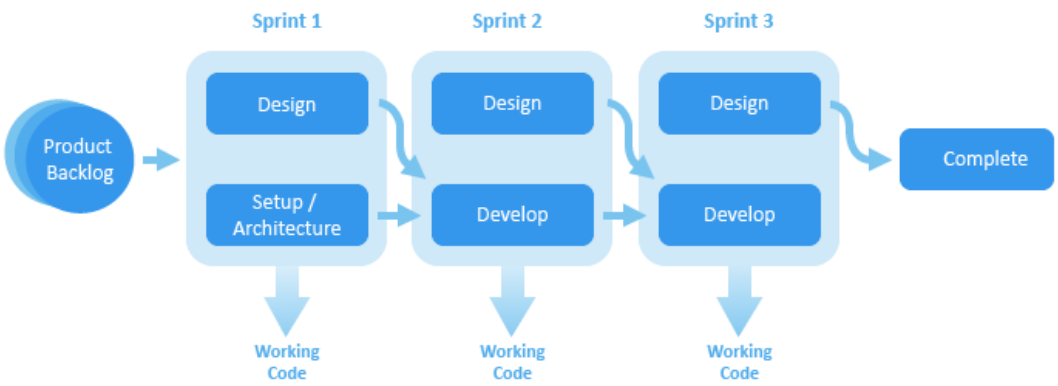

Figure 4.11: Agile design process. [Bus19]

Design is a core element in the successful construction of a product. The design process is not simply linear, and it requires multiple plans to be drawn up, gathering as much of the initial customer requirements. The process is iterative, with continuous revisions needed before the final product starts to take shape.

The design process of the Web-GUI covered the user experience flow and user interface design, all the while maintaining granular control of the Qb2 device. The

iterative design was the only method that could produce the results expected by the company as well as the customers. Agile provides us with a rapid response methodology [Bus19]. The design process takes place in sprints. A sprint is a predefined time period within which a planned set of tasks are to be completed. Each sprint starts with a planning phase where the expected results for that sprint and the overall direction of the product are also discussed. Having shorter sprint durations is extremely beneficial to the developer as its a shorter turnaround time between implementation and feedback from the stakeholder.

User Experience (UX)

Gathering information about the target customer is the foundational step in building the best possible UX for a product. These require extensive discussions with customer-facing teams that have insight into both the product as well as the needs of a customer. After the discussions about customer expectations, brainstorming sessions need to be conducted to gather insights from various teams and team members. Catering to the target audience is crucial in developing a good user experience.

User Interface (UI)

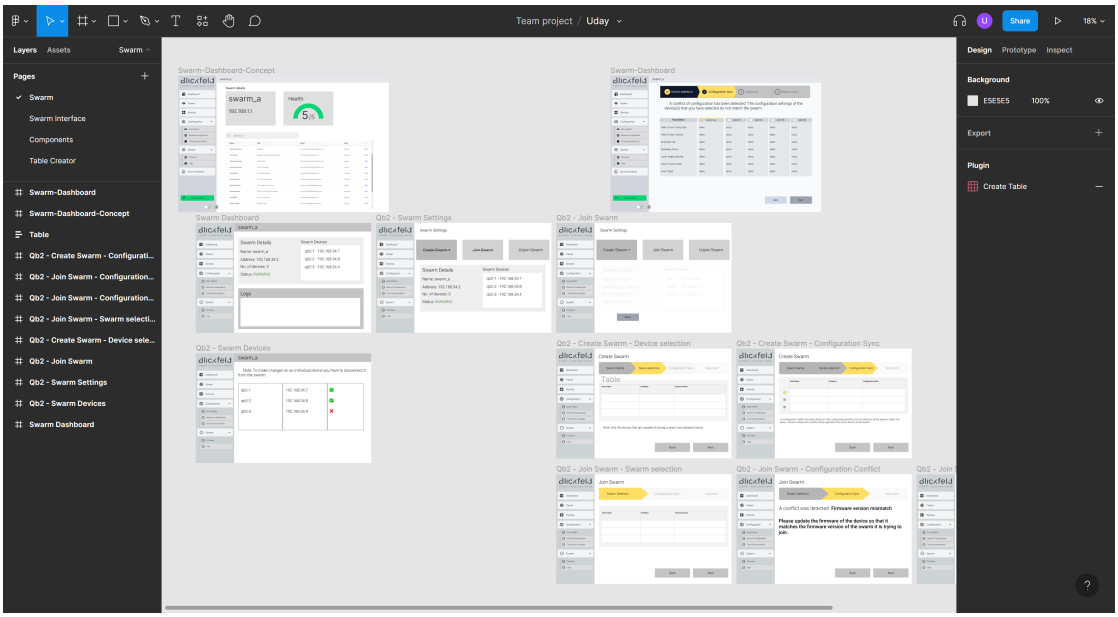

Figure 4.12: Figma design board.

User interface design depends heavily on the required user experience. The design is created based on the style guides set by the company. For rapid and collaborative development of a user interface, there exist multiple web-based platforms. To create the user interface for this project, the tool 'Figma' was used. Figma is an online platform that aids in all facets of the design process, from wireframing to prototyping [Fig]. The collaborative element is key for developing in an agile manner, as it makes getting feedback from the stakeholders a faster, streamlined process.

4.5.3 Module structure

This project was developed using the Angular framework. Angular system in itself supports modularity. The code can be arranged into different modules based on functionality. Therefore, it results in a clear structure and reusable pieces of code. This enables developers to easily arrange the code into modules that can be included or excluded from the final code bundle.

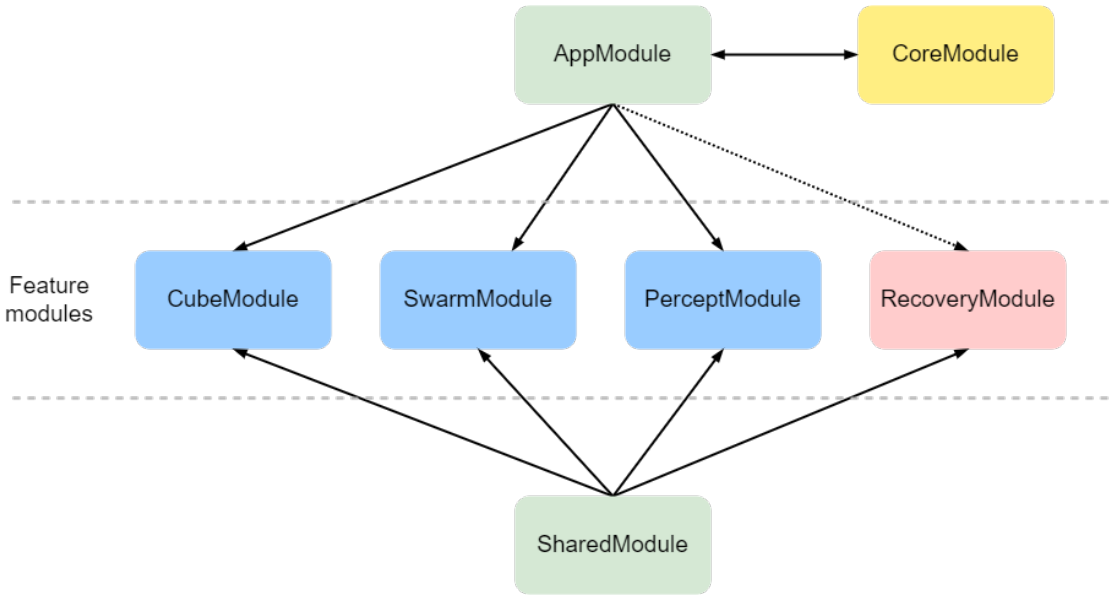

Figure 4.13: Qb2 module structure.

The core of Angular runs on the app module, the root of the application. The app module is extended with other custom modules to improve flexibility. The shared module usually contains the shared components, pipes, and services. The custom modules can make use of these shared resources. This project has a variety of features that would become difficult to maintain and develop if not made using a modular

design. This project will have four primary modules apart from the app module and shared module;

Cube module The cube module is the base module for a Qb2 device. It contains all the configuration and system settings like scan pattern configuration, time synchronization, network management, and firmware update. The cube module also has access to the viewer, which is the canvas using Three.js to display the point cloud data of a singular device.

Swarm module The Swarm module contains all the functionality needed to create and maintain a cluster of Qb2 devices. The initial registration process, configuration management, and point cloud alignment process. The Swarm module will use the base components from the Cube module, which is for a singular device, and add a layer on top of that to handle a cluster of devices.

Percept module The percept module is a standalone system that provides data processing using algorithms for object detection and volume measurement.

Recovery module The recovery module is a minified version of the interface. The recovery module is activated in case of a major failure on the device if one of the critical processes fails. For example, if there is a failure during the firmware installation process. In this scenario, the device will go into recovery mode, and the user will be presented only with the components of the recovery module, which will give the ability to load fresh firmware and gather diagnostically relevant information.

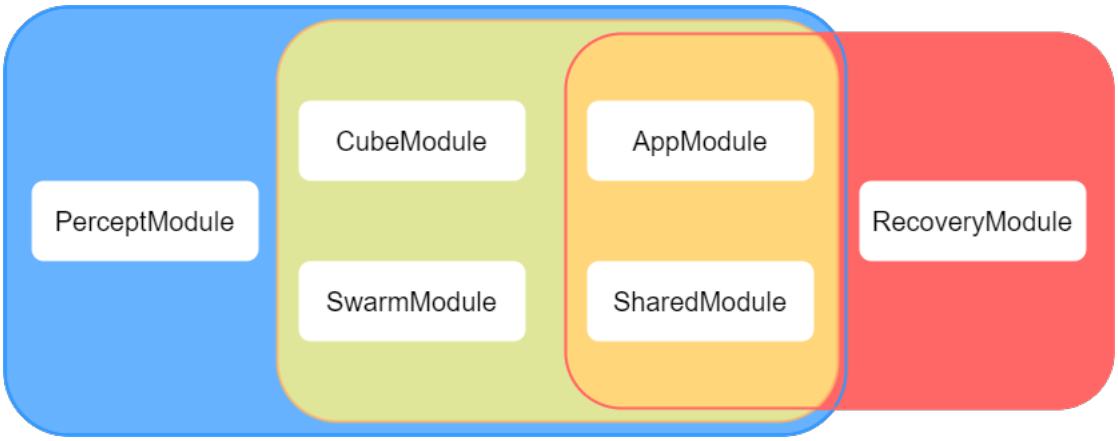

Figure 4.14: Module grouping for different modes the application can run in.

The figure 4.14 is a representation of the different modes the application is capable of running. The base modules are the app module and the shared module; these modules are required for all the modes. They contain all the dependencies, the base routing, and the environment variables that decide the mode in which the application is supposed to run. The shared modules contain components like the sidebar component that is needed by all the modules.

Yellow The cube module and Swarm module contain all the custom components that sit on top of the components from the shared module.

Blue The percept module can be activated and deactivated based on the license opted by the user.

Red This represents the recovery module that only contains the base modules and some extra modules that can help the user with diagnostics and recovery.

The modules are built only if they are needed, keeping the application lean.

5 Implementation

This chapter covers the implementation done for this thesis. There are two primary sections: the Swarm section covers the steps involved in the implementation of configuration synchronization, and the second covers the WebGUI implementation for the Qb2 LiDAR.

5.1 Swarm

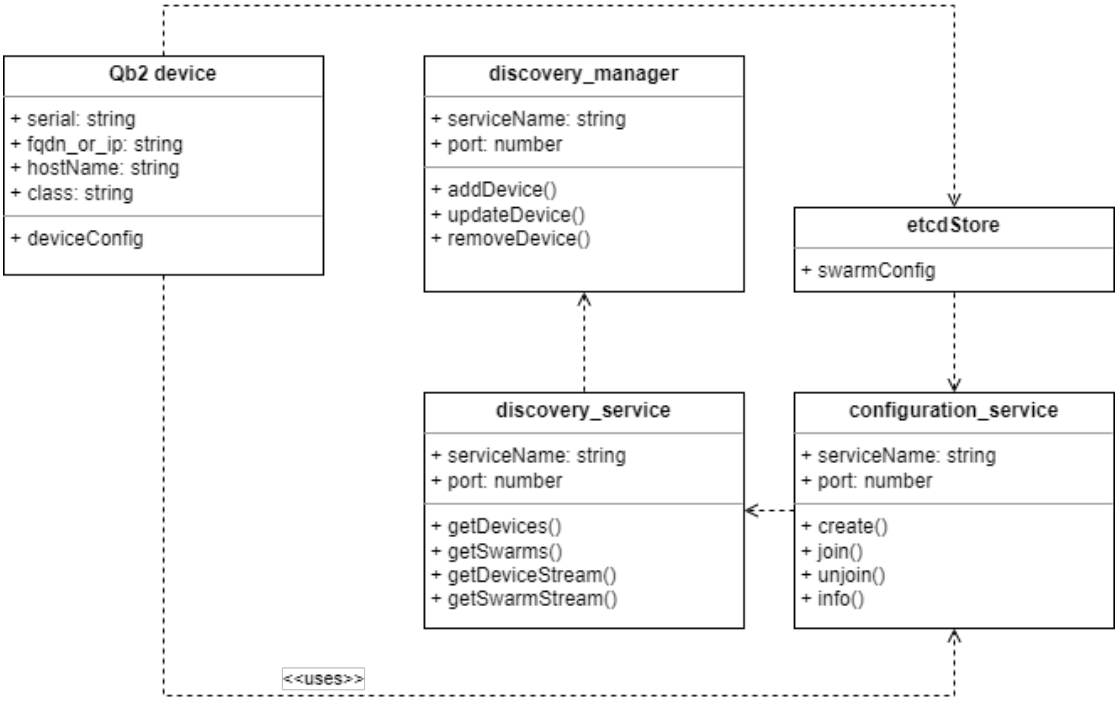

Figure 5.1: Class diagram.

The class diagram shown in figure 5.1 depicts the connection between the services that are needed for a Swarm capable device. The device makes use of the configuration

service to create or join a Swarm. The configuration service has an initial dependency on the discovery service to look for devices on the network. The discovery service uses the discovery manager, which is implemented using pyzeroconf to look for the devices emitting the particular service on the network. If a connection to a Swarm is established, the configuration service sets up an etcd store for the Swarm. This etcd store contains the configuration that the members of the Swarm must have for proper performance. The device can take the configuration information from this shared etcd store.

5.1.1 Discovery service

The discovery service searches the local network for Swarm-capable Qb2 devices and any Swarms that may be active on the network. The Python zeroconf (pyzeroconf) implementation is used to create the discovery service as it provides a suite of functionalities to detect changes on the network.

A discovery manager is used to listen for changes on the network. The discovery manager uses the pyzeroconf library to import the service browser, which is used to target a specific service on the network.

Zeroconf uses a set of services that maintain the current state of the network. The 'add service' is called when the service browser finds a fresh device with the service name it is looking for. The remove service is triggered when that device is no longer detectable on the network. The update service is triggered when a previously detected device, which was inactive, is not active again.

After a device is discovered, it is assigned a unique identifier, a class, which puts the device into a category according to the type of device and a name.

5.1.2 Configuration service

The configuration service gets the information of the device in the network from the discovery service mentioned above. The configuration service has a set of APIs that enables Swarm-based actions, create, join, info, and un-join.

Create

The create API to combine a set of devices into a cluster. After accessing a cube, when another target device is selected for creating a cluster, the create function is called, which passes the cluster name, an optional parameter, and the unique ids of the devices that would become part of the Swarm. The message structure for a create request is shown in listing 5.1.

```
message ClusterCreateRequest {
    optional string cluster_name = 1;
    repeated string member_ids = 2;
}
```

Listing 5.1: Request structure for creating Swarm

Join and Unjoin

The join and un-join services are very similar. They have the same set of rules for errors and conflicts, a device that is not part of a Swarm cannot perform any of these services. The service makes sure that the devices are in the correct state, i.e., if the device corresponding to the target identifier is not in a Swarm, then the un-join service cannot be performed, and if the device is already a part of another Swarm, it cannot be added to the Swarm. This information is gathered using the *GetSwarms* service.

Configuration synchronization

The configuration synchronization process makes use of the etcd distributed key-value storage system. A key-value pair is stored on the cluster using the *EtcdStorageSetRequest* functionality. This method updates the storage service that runs on the member devices. If there is a new member that is added to an existing Swarm, the storage service will automatically pull the latest data from the cluster.

5.1.3 Results

The results will go through the process of discovering Swarm capable devices on the network, creating a Swarm using two emulators of the Qb2 device called *qemu1* and *qemu2*. After the creation of the Swarm, the next steps would be to apply a key-value pair to the Swarm using the etcd distributed key-value storage system.

Discovering devices on the network

The discovery service is run to get the list of the devices found on the network that have the specific service name unique to Blickfeld devices. In this phase of implementation, a device emulator called QEMU was used to mimic the behavior of Qb2. Listing 5.2 shows a part of the result when the service *blickfeld.swarm.services.Discovery GetDevices* is called. The list contains all the QEMU devices on the network; in this implementation, there are six instances of QEMU running.

```
| devices[4/6] = {DevicesEntry}
| | key.. = "256014ac131b4238b0ce754dd213a4be"
| | value = {Device}
| | | fqdn_or_ip... = "172.19.0.5"
| | | name......... = "qemux86-76.local."
| | | serial_number = "256014ac131b4238b0ce754dd213a4be"
| | | class........ = QB2
| devices[6/6] = {DevicesEntry}
| | key.. = "5cdd5c9bbe2a49a88fee7c95ba5d476a"
| | value = {Device}
| | | fqdn_or_ip... = "172.19.0.2"
| | | name......... = "qemux86-77.local."
| | | serial_number = "5cdd5c9bbe2a49a88fee7c95ba5d476a"
| | | class........ = QB2
```

Listing 5.2: Discovery service GetDevices

In listing 5.3 the output for *blickfeld.swarm.services.Discovery GetSwarms* service shows that there are currently no Swarms on the network.

```
Received message:
| clusters[0/0] = {}
```

Listing 5.3: Discovery service GetSwarms

To create a Swarm, the *blickfeld.swarm.services.Cluster Create* service is used; this service takes the member identifier of the device that is currently being used to create the Swarm and the identifier of the devices that need to be added to the Swarm. For this example the member identifiers for *qemu1* and *qemu2* were passed. If the *GetSwarms* service is run again, the output would have information regarding the Swarm and its members as shown in listing 5.4.

```
| clusters[1/1] = {ClusterInfo}
| | members[1/2] = {ClusterMember}
| | | id = "5cdd5c9bbe2a49a88fee7c95ba5d476a"
| | | ip = "172.19.0.2"
| | members[2/2] = {ClusterMember}
| | | id = "256014ac131b4238b0ce754dd213a4be"
| | | ip = "172.19.0.5"
```

Listing 5.4: Discovery service GetSwarms (active Swarm)

Similar functions have been implemented for joining an existing Swarm and unjoining a Swarm. Now that a Swarm exists, a key value will be stored on the Swarm that will be accessible by all the members of the Swarm. The key-value pair would be applied to *qemu1*, which is a part of the Swarm. The *EtcdStorageSetRequest* function is used to update the key-value pair on the Swarm.

```
EtcdStorageSetRequest(key="blickfeld.keyone/test", value=value_payload)
```

Listing 5.5: Request to set value in etcd store

To verify if the key-value pair has been set to the members of the Swarm, the values storage service of qemu2 has to be retrieved. It can be done as shown in listing 5.6

```
storage_service = EtcdStorageStub(channel)
response = await storage_service.List(Empty())
```

Listing 5.6: Checking etcd key-value store

5.2 Web-GUI

5.2.1 Technology stack

The implementation of the web interface (Web-GUI) based on the design goals required the use of a particular technology stack.

Angular

Angular is a development platform built on TypeScript [Ang]. It is a component-based framework, and this design style helps simplify the development of scalable applications. Angular helps in improving the speed of development as the developer can make use of pre-built libraries that are tightly integrated into angular. It also natively supports routing, form management, client-server communication, and tools that aid development [Ang]. Angular can also be updated easily without breaking the application. The use of the programming language TypeScript was also based on the requirements for the project. TypeScript, which is a superscript of JavaScript, is a strongly typed programming language. Being strongly typed is a key requirement for embedded software that needs to run on the Qb2 device, to prevent any issues during run time. TypeScript can catch mistakes in the editor itself during development.

Tailwind CSS

Tailwind CSS is a utility-based CSS framework that is used for the development of the WebGUI and its components. It is a low-level CSS framework that helps in building entirely custom interfaces with much less coding effort than traditional CSS. Tailwind makes use of utility classes that drastically improve the speed of development. For example, to change the background color of a button, the developer need not create a

class in the style sheet of the project, all they need to do is to include 'bg-green-500' in the class name of the button. This method of development significantly helps in rapid prototyping. This feature helps in maintaining consistency in the development process where multiple developers would be working on different components of the application. Tailwind also facilitates the creation of custom classes and editing of pre-built classes to completely customize the interface of an application. Another important feature of tailwind that makes it suitable for this project is that during building for production, the tailwind does a pruning process that deletes all the unused classes from the final build.

Three.js

It is a library built using JavaScript that allows users to create and present 3D graphics and animations in a web browser. Three.js relies on WebGL, which works on any browser that supports WebGL. There is no dependency on the GPU of the device. Three.js makes it simpler for a developer to use WebGL with the help of various objects, methods, and features.

5.2.2 Websocket gRPC proxy

gRPC streams data over HTTP/2. However, HTTP/2 is not currently supported by browsers, and it is a functionality that is currently under development. Establishing a connection requires a custom proxy to be introduced. A WebSocket proxy is created between the Websocket server and gRPC. This proxy establishes a connection between the Web-GUI and the services exposed by the device. The proxy is built using the socket.io package that handles the WebSocket connection, and to handle the gRPC stack @grpc/grpc-js is used.

The Web-GUI is the client, and it emits the event; the event contains the type of request, the method, the requested data, and the callback function. Inside the WebSocket proxy, the Websocket server receives this event, and a gRPC client is created, which creates a request according to the event. If the request expects a data stream, the callback will contain a Universally Unique Identifier (UUID) that'll be used as an identifier for the request. This UUID will be used by the server to stream data to the client. On receiving the response, the client will deserialize the information so it can be shown on the WebGUI.

Making a server stream request or bidirectional stream request will return a value that has the type clientReadableStream; it is a call that is used for streams from the server side. This value will be saved with the same UUID sent in the callback to the client. The WebGUI can then use the UUID to emit a 'closeStream' event when it wants

to stop the stream. The server will match the stream to the UUID and close the stream of data to the client.

WebSocket service

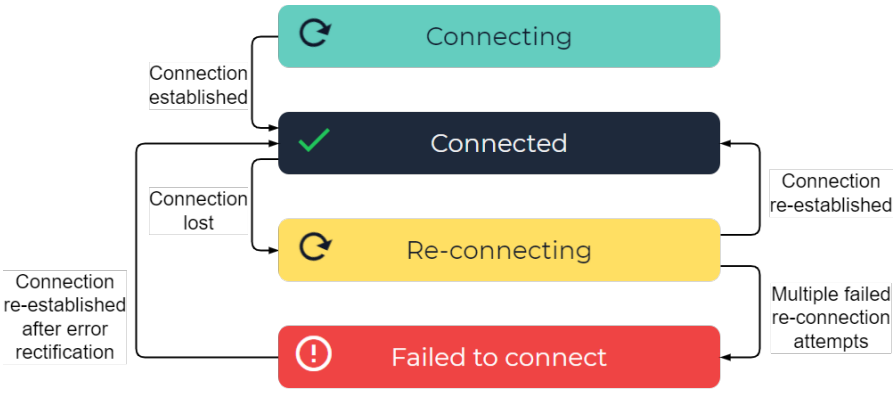

Figure 5.2: WebSocket connection indicator.

The WebSocket service will establish the connection to the WebSocket server in the WebSocket proxy using the 'socket.io-client'. The WebSocket tunnel is opened by specifying the URL, port, and transport type as shown in listing 5.7. The values for establishing the connection, hostname, and port number, are taken from the environment file.

```
this.socket = ws.io(this.document.location.protocol + '//' + (environment.
    hostname || document.location.hostname) + ':' + environment.
    websocket_port, { transports: ['websocket'] });
```

Listing 5.7: Establishing a WebSocket connection

The WebSocket service uses the socket.io client to detect the state of the connection. This information can be used to show the user the current state of the connection, as shown in figure 5.2. The WebSocket service will regularly check for the connection state. If it loses the connection, it will re-attempt multiple times. If the attempts keep failing, the user will be prompted about the issue and the corresponding error code as shown in figure 5.3.

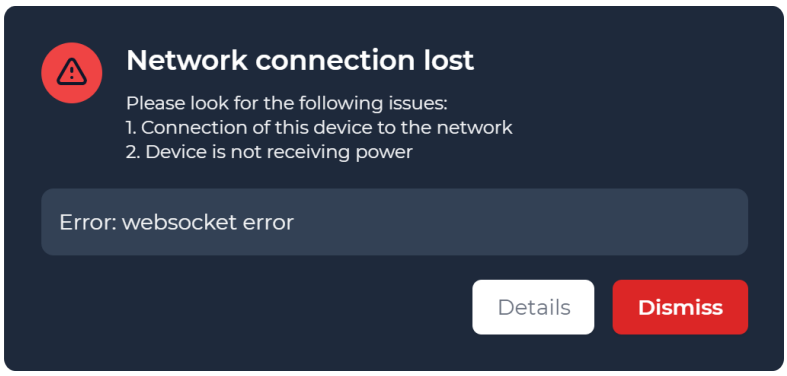

Figure 5.3: WebSocket connection lost.

5.2.3 Routing

Routing in angular is responsible for navigation. The app router that sits at the root of the project contains all the routes needed for navigating to the different sections of the web application. The main app router exists as part of the core app module. The router contains the definitions that register a particular component to a particular path. This information is used by the router module.

In this project, however, since there are multiple separate modules that are active in different conditions, for example, the recovery mode, each module has its own routing module. Since the application has different modes, for each mode, only a specific set of modules will be activated, and the routes specific to that mode needs to be set. In angular custom, routes are set by setting them in the environment.ts file. Every mode will need its own environment file, like environment.recovery.ts. For activating this, a new configuration needs to be added to the angular.json file. Based on the build condition, the specific environment file will be used, and the custom routes mentioned in that environment will be loaded to the app routing module.

```
const routes: Routes = [
    { path: '', redirectTo: 'dashboard', pathMatch: 'full' },
    { path: 'dashboard', component: DashboardComponent },
    { path: 'system/firmware', component: FirmwareComponent },
    { path: 'system/logs', component: LogsComponent },
    { path: 'viewer', component: ViewerComponent },
    { path: 'configuration/alignment', component: AlignmentComponent },
    { path: 'configuration/scan-pattern', component: ScanPatternComponent
     },
    { path: 'configuration/network-setup', component:
    NetworkSetupComponent },
    {
      path: 'configuration/scan-pattern',
```

```
      children: [
        {
          path: '',
          children: [
            { path: '', component: ScanPatternComponent },
            { path: 'viewer', component: ViewerComponent, outlet: 'dual-
    view' },
          ],
        },
      ],
    },
    { path: '**', redirectTo: '/dashboard', pathMatch: 'full', }
  ];
```

Listing 5.8: Cube module routes

In listing 5.8 are some of the routes included in the cube module. All routes for future features will be added here. The cube module will have a lot of different routes for all the different pages for configuring the cube, pages for the viewer, and settings pertaining to the viewer. In the code shown in listing 5.9 that shows the routes for the recovery module, there are significantly fewer routes as none of the other components not included in this module will be built.

For the navigation, there is a sidebar component that is part of the shared module. This sidebar component will adapt to the different active modules. On referring to figure 5.4, it can be seen that the sidebar that the user will see when they connect to a regular functioning Qb2 device. It contains all the routes mentioned in the cube menu list file, which is derived from the routes mentioned in listing 5.8. In figure 5.5, it can be seen that only a limited number of options are available.

```
const routes: Routes = [
    { path: '', redirectTo: 'recovery', pathMatch: 'full' },
    { path: 'recovery', component: RecoveryComponent },
    { path: 'system/firmware', component: FirmwareComponent },
    { path: 'system/diagnostics', component: DiagnosticComponent },
    { path: '**', redirectTo: '/recovery', pathMatch: 'full', }
];
```

Listing 5.9: Recovery module routes

5.2.4 Dynamic components

Angular is a component-driven framework. To achieve modularity and to keep the lean application components that are dynamic, that adapt easily, and reusable were created.

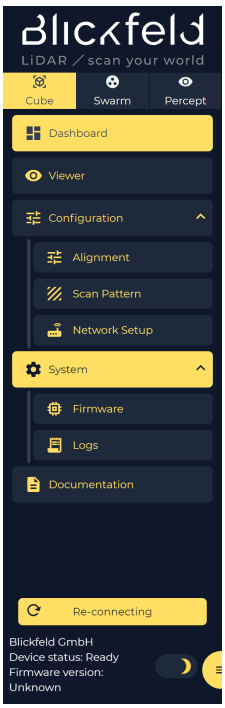

Figure 5.4: Sidebar of Qb2 functioning normally.

Figure 5.5: Sidebar of Qb2 in recovery mode.

Button states

The button component was designed to be completely flexible and capable of showing the current state of the action that was triggered using the button. This is to improve the user experience with interacting with the device; if an action was triggered, the button will track the state of the function called and analyze the response, whether it was successfully completed or whether it failed. When an error occurs, the user will be prompted with the error code and message.

```
<app-button [buttonIcon]="'check.svg'" [buttonValue]="selected ==
    undefined || selected == -1 || !form.dirty ? 'Apply' : 'Save & apply'"
     (buttonClick)="$event.handle(onSaveAndApply())" [buttonDisable]="form.
    invalid ||  (selected == undefined && !form.dirty)"> </app-button>
```

Listing 5.10: Example of using button component

In listing 5.10, an example of the implementation of the button component is shown. The state of the button is detected using promises. A promise is a method to execute asynchronous functions that make the use of callbacks, and the promise waits for the response of the function call triggered by the button click.

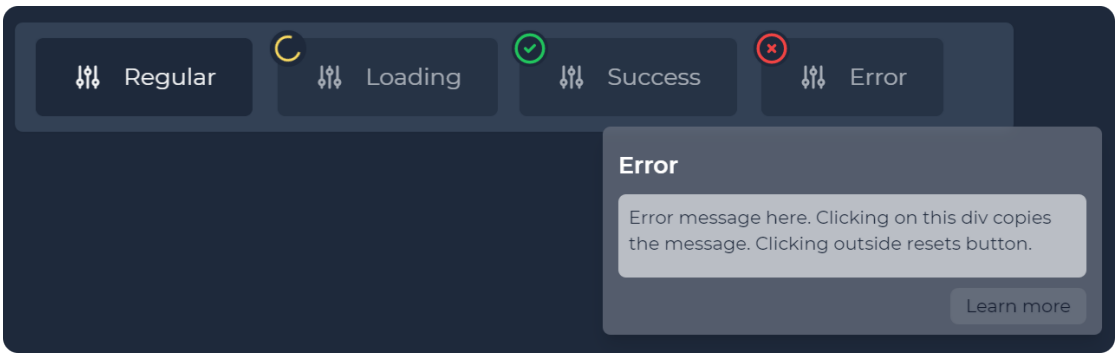

Figure 5.6: Button states.

Range slider

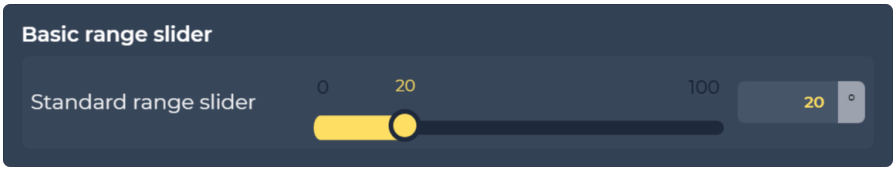

Figure 5.7: Standard slider.

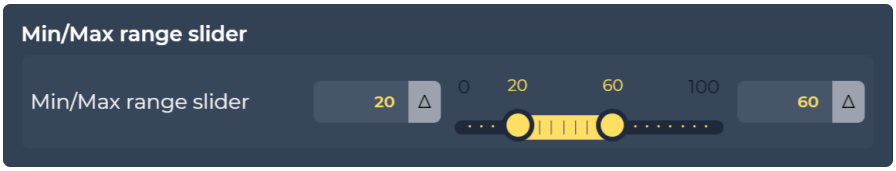

Figure 5.8: MIN and MAX slider.

The range slider component is a frequently used component in this project. It is used in all types of configuration interfaces. Due to this reason, the component needs to be designed in a way that there is no unnecessary code duplication.

```
value_range1: number = 20;
highValue_range1!: 30; // Only needed for MIN/MAX slider
range1: Options = {
    floor: 0,
    ceil: 100,
    step: 1,
    showTicks: false
};
```

Listing 5.11: Slider options

The options needed (as shown in listing 5.11) for both types of sliders are exactly the same apart from the 'highValue' parameter. The slider will turn into a MIN/MAX slider if a value is set for this parameter.

5.2.5 Results

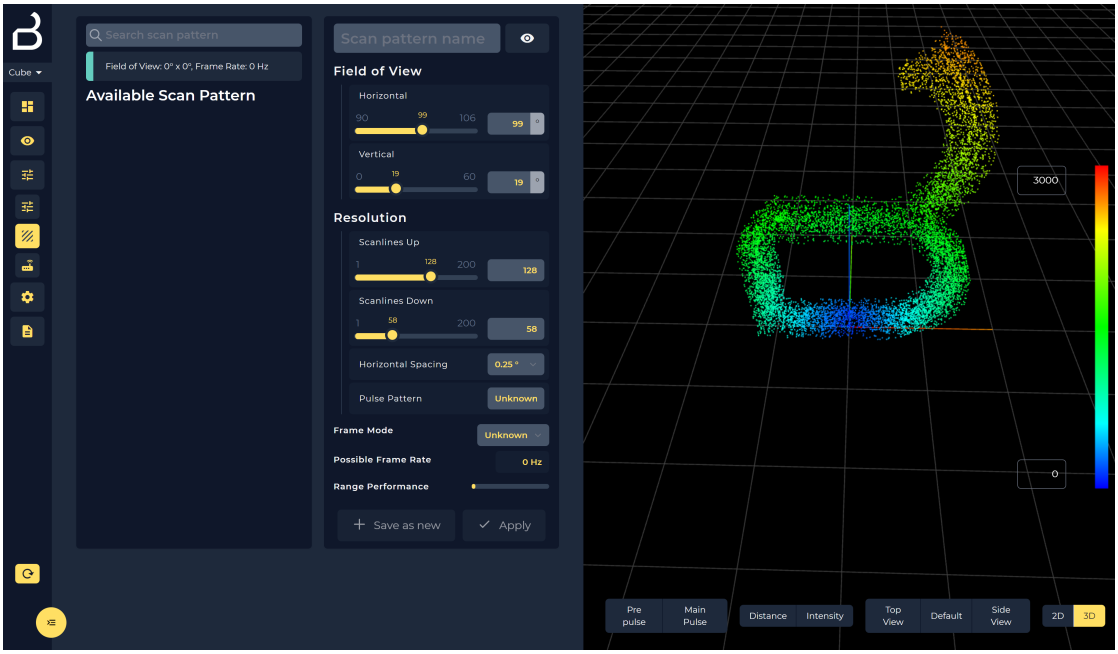

Figure 5.9: WebGUI for scan pattern configuration on Qb2.

In the current state of the WebGUI, all the aforementioned components have been implemented. The current feature list includes scan pattern configuration, network configuration, firmware update, and point cloud viewer. Figure 5.9 shows the scan pattern configuration page, the viewer on the right side of the screen to see the changes being reflected.

In figure 5.10 the implementation of the network configuration page is shown. The entire WebGUI is also built with responsiveness in mind, and special attention has been given to making the interface touch surface friendly for devices like tablets that could be used in remote environments.

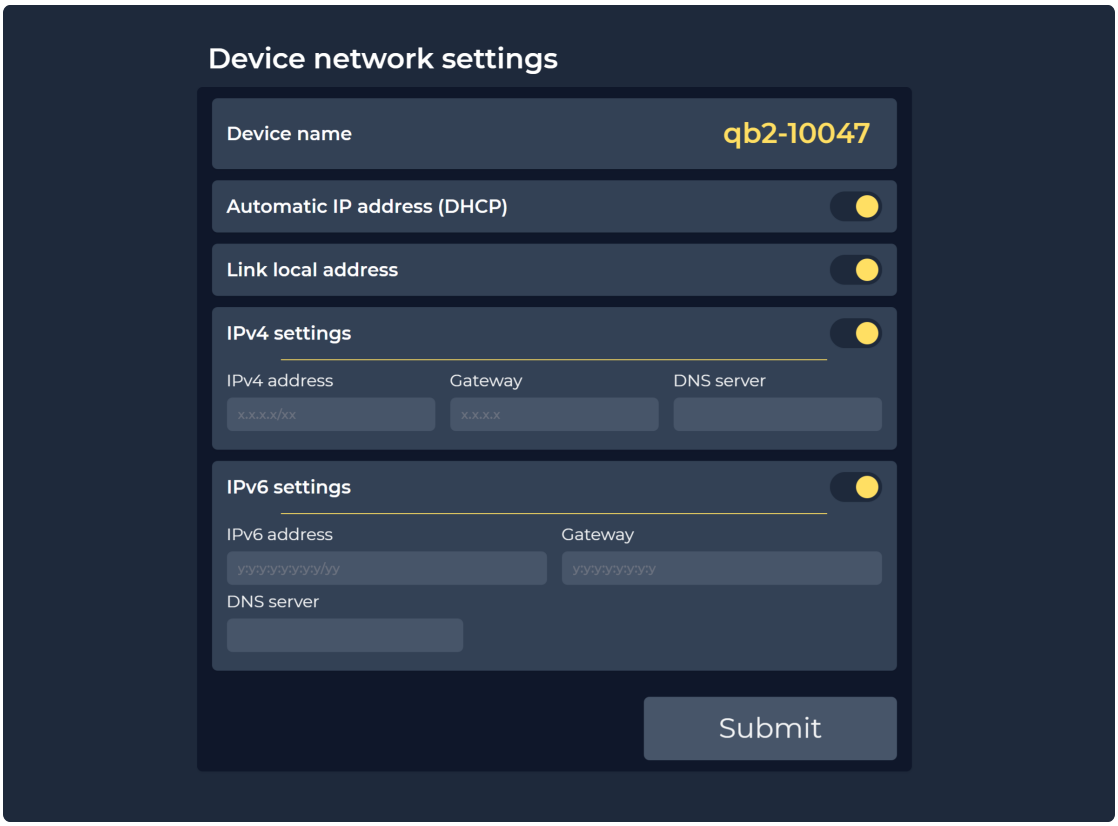

Figure 5.10: WebGUI for network configuration on Qb2.

5.3 Test setup

Testing the discovery service, configuration service, and firmware update service initially could not be done on actual Qb2 devices. For facilitating testing of features and services, a virtual environment is needed.

QEMU is an open-source machine emulator and virtualizer [QEM]. QEMU can run any OS or program made for a specific machine, in this case, the Qb2. The QEMU, in our case, was set up in a docker container, facilitating the creation of multiple instances. Using dynamic translation, QEMU achieves excellent performance. QEMU can achieve near-native performance when used as a virtualizer by executing the guest code directly on the host CPU [QEM].

The QEMU could mimic the limited functionalities of the Qb2. Using QEMU in containers is advantageous as multiple instances of it can be run; this was required to perform tests while developing the discovery service. The QEMUs were set up to

expose a specific service name, which could then be used by the discovery service to detect the QEMUs on the network.

To test out the functionality of the interface of the firmware update QEMU was crucial. Each firmware update process took a significant amount of time, resulting in poor turnaround times between pushing changes to the code and seeing results. The QEMU was set up to receive a custom firmware file that was created, and the target on the Web-GUI was changed from an actual device to the IP address of the QEMU. The QEMU was set up to respond with a data stream that would be precisely like the data stream from an actual device. Having a virtual emulator like QEMU was beneficial to test the system's limits and prepare for edge cases.

6 Summary and Conclusion

Summary

This thesis covers the design and implementation of two major topics, a configuration synchronization mechanism for the Swarm module, which will be used in a LiDAR by Blickfeld GmbH, and a WebGUI interface for the LiDAR devices.

In the first phase of the thesis, we gather the information that provides us with a context for the research topic. We look at the technologies that are currently available in the market. We look at the benefits they provide and their shortcomings. We look at the progression of lidar technologies that have occurred over time. After understanding the current state of the technology in the field of LiDARs, we go through the differences between the two LiDAR technologies that make up a majority of the current market, rotor-based mechanical LiDAR and a solid-state LiDAR. We compare them based on the scenarios they can be used in, the scenarios where one type of LiDAR may have the upper hand, the cost-to-benefit ratio, and the scalability of these systems in different environments. With this data, we establish the motivation for the research topic we are undertaking. The motivation is to design a decentralized system of LiDARs, whose data can be combined to provide a more accurate 3D representation of an environment.

We then look at the tasks planned to be achieved in this thesis. Configuration synchronization and the design of a WebGUI interface for LiDAR devices. To understand the need for these, a background into the device, Qb2, has been provided along with some information on how the device software is structured and the communication methods between the hardware and software of the device. The Swarm concept is then introduced, and the end goal of the module is stated. We go through the process that was taken in designing this concept, understanding its limitations, generating flowcharts to understand the flow, and understanding all the elements that would be required in developing such a module for the Qb2 product. The architecture of the Swarm module is then explained, along with the explanation of the services we would need to implement this module for the Qb2. The design for the WebGUI gives an understanding of the requirements set by Blickfeld and the process that was set in place to achieve this goal. The design of the WebGUI included the design methodology that was used, the planning involved in designing the user experience and the user interface, as well as an understanding of the architecture of the modular structure

that was followed to maintain a high level of modularity as that is one of the key requirements for this product.

To understand the choice of technologies that were used in the implementation of this product, the technologies are explained in detail, along with the reason for the choice. Reasons like compatibility and desired performance or reliability.

The development phase covers the initial implementation steps taken for the configuration synchronization service and the services it depends on, services like the discovery service. The implementation phase covers how the technologies that were chosen for this project were used and the functions that were made to perform actions pertaining to configuration synchronization. The development phase also covers the complete implementation of the WebGUI, the proxy connection that enables the web interface to send and receive data from the device that uses HTTP/2, the custom routing that was made to handle the modular structure of the project, and the dynamic components that can adapt to the different environments and provide a streamlined experience for the user as well as for the developer while developing using those components.

Future scope

The future scope covers the planned features and improvements for the Swarm module of the Qb2 device.

Combining point cloud data After the configuration synchronization process is completely functional, the next phase of design and development will be for the process to combine the point cloud data generated by the LiDAR devices. This process will need to be extremely streamlined and fault-tolerant, as the quality of data depends on the latency of communication between the devices, so in a network that might have a fluctuating traffic flow, methods will have to be developed that can handle these issues and maintain the quality of the data.

Perception processor One of the key reasons for having a distributed LiDAR setup is to avoid single points of failure and to take advantage of the capability to perform load distribution among the devices to perform perception tasks like object detection. On a smaller scale, this is easily achievable without the devices having to use a lot of their computing potential. However, in large-scale scenarios, the devices may struggle to handle large point cloud datasets; to facilitate this, there are plans to develop a percept processor, which in essence is a regular computer that has much larger processing power; this machine will be configured to mimic a LiDAR device and be added to the Swarm. Doing so should reduce the load on the actual LiDAR devices when operating in larger numbers.

List of Figures

1.1 Example of LiDAR 3D point cloud 1
1.2 Diagram of rotor based mechanical LiDAR 2
1.3 Diagram of solid-state LiDAR 3
1.4 Illustration of rotor based lidar deployed in a room 4
1.5 Illustration of solid state LiDARs deployed in a room 4

2.1 Blickfeld Qb2 7

3.1 etcd architecture 11
3.2 Replicated state machine 11
3.3 Leader election process 12
3.4 PACELC representation of key-value storage technologies 13
3.5 GRPC architecture 14
3.6 gRPC workflow 15
3.7 gRPC streaming 16

4.1 Swarm concept 19
4.2 Swarm interpretation 22
4.3 Swarm architecture 25
4.4 Comparison of different scan pattern configurations 26
4.5 State diagram for creating Swarm 28
4.6 State diagram for joining Swarm or adding the device to Swarm 29
4.7 Configuration conflict state diagram 30
4.8 Swarm dashboard concept design 31
4.9 Configuration conflict resolution concept design 32
4.10 Alignment process for a singular cube 33
4.11 Agile design process 34
4.12 Figma design board 35
4.13 Qb2 module structure 36
4.14 Module grouping for different modes the application can run in 37

5.1 Class diagram 39
5.2 WebSocket connection indicator 45

5.3 WebSocket connection lost 46
5.4 Sidebar of Qb2 functioning normally 48
5.5 Sidebar of Qb2 in recovery mode 48
5.6 Button states 49
5.7 Standard slider 49
5.8 MIN and MAX slider 49
5.9 WebGUI for scan pattern configuration on Qb2 50
5.10 WebGUI for network configuration on Qb2 51

Listings

5.1 Request structure for creating Swarm 41
5.2 Discovery service GetDevices 42
5.3 Discovery service GetSwarms 42
5.4 Discovery service GetSwarms (active Swarm) 42
5.5 Request to set value in etcd store 43
5.6 Checking etcd key-value store 43
5.7 Establishing a WebSocket connection 45
5.8 Cube module routes 46
5.9 Recovery module routes 47
5.10 Example of using button component 48
5.11 Slider options 49

Bibliography

[Aba12] D. J. Abadi. "Consistency Tradeoffs in Modern Distributed Database System Design." In: *Yale University* (2012).

[Ama] Amazon. *Protocol Buffer Format for Alexa Gadgets*.

[Ang] Angular.io. *What is Angular?*

[Bus19] L. Bustamante. *Design process into Agile framework*. 2019.

[etca] etcd.io. *etcd versus other key-value stores*.

[etcb] etcd.io. *etcd3 API*.

[Fig] Figma. *About Figma*.

[Fin21] P. Finkelshteyn. *Big Data World, Part 6: PACELC*. 2021.

[Gmba] B. GmbH. *Qb2*.

[Gmbb] B. GmbH. *Start of Frame (SOF) synchronization*.

[grpa] grpc.io. *Core concepts, architecture and lifecycle*.

[grpb] grpc.io. *Why gRPC?*

[Haz21] Hazelcast. *What is a key-value store?* 2021.

[HW90] M. P. Herlihy and J. M. Wing. "Linearizability: A Correctness Condition for Concurrent Objects." In: vol. 12. 3. Carnegie Mellon University. ACM Transactions on Programming Languages and Systems, 1990, pp. 463–492.

[HW92] M. J. Halmos and J. H. S. Wang. "Laser radar systems and applications." In: *Optical Technologies for Aerospace Sensing: A Critical Review*. Ed. by J. E. Pearson. Vol. 10269. International Society for Optics and Photonics. SPIE, 1992, pp. 303–333. DOI: 10.1117/12.161579.

[IBM21] IBM. *What is zero configuration networking?* 2021.

[Ion] Ionos. *gRPC – Paving the way to the client-server communication of the future*.

[Kha+21] M. U. Khan, S. A. A. Zaidi, A. Ishtiaq, S. U. R. Bukhari, S. Samer, and A. Farman. "A Comparative Survey of LiDAR-SLAM and LiDAR based Sensor Technologies." In: *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*. 2021.

[Lee+07] J. W. Lee, H. Schulzrinne, W. Kellerer, and Z. Despotovic. "z2z: Discovering Zeroconf Services Beyond Local Link." In: *2007 IEEE Globecom Workshops*. 2007, pp. 1–7. DOI: 10.1109/GLOCOMW.2007.4437805.

[Mot20] S. C. Motaz Khader. *An Introduction to Automotive LiDAR*. Tech. rep. Texas Instruments, 2020.

[Nat] A. C. Native. *Getting Started with Kubernetes | etcd*.

[OA21] N. Oceanic and A. Administration. *What is Lidar?* Tech. rep. U.S. Department of Commerce, 2021.

[OOa] D. Ongaro and J. Ousterhout. *Designing for Understandability: the Raft Consensus Algorithm*.

[OOb] D. Ongaro and J. Ousterhout. "In Search of an Understandable Consensus Algorithm (Extended Version)." In: *Stanford University* ().

[QEM] QEMU.org. *About QEMU*.

[Ran] A. Rana. *Understanding Etcd3*.

[RB19] S. Royo and M. Ballesta-Garcia. "An Overview of Lidar Imaging Systems for Autonomous Vehicles." In: *Applied Sciences* 9 (Sept. 2019), p. 4093. DOI: 10.3390/app9194093.

[RCG22] R. Roriz, J. Cabral, and T. Gomes. "Automotive LiDAR Technology: A Survey." In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 6282–6297. DOI: 10.1109/TITS.2021.3086804.

[Sab22] W. Sabbatini. *Robots Aren't Done Reshaping Warehouses*. Tech. rep. The New York Times, 2022.

[Sin] M. Singh. *Unary Streaming via gRPC*.

[Sof] Sofastack. *Introduction to the Raft algorithm*.

[Wal21] J. Walter. *mDNS – Multicast DNS*. 2021.

[WWX20] D. Wang, C. Watkins, and H. Xie. "MEMS Mirrors for LiDAR: A review." In: *Micromachines* 11.5 (Apr. 2020), E456. ISSN: 2072-666X. DOI: 10.3390/mi11050456.

[YC18] B. D. Yoo Han Woong Druml Norbert and S. Christian. "MEMS-based lidar for autonomous driving." In: *e and i Elektrotechnik und Informationstechnik* 135.1613-7620 (2018).

[Zoo] A. ZooKeeper. *What is ZooKeeper?*