

# Bachelor-Thesis an der Hochschule Luzern - Technik & Architektur

**Titel** **Roboterdaten auf dem S-bot visualisieren und Befehlsausführung mit WebUI**

**Diplomandin/Diplomand** **Samuel Huser**

**Bachelor-Studiengang** **Bachelor Digital Engineering**

**Semester** **FS23**

**Dozentin/Dozent** **Dipl. El.-Ing. ETH Annina Blaas**

**Expertin/Experte** **Dipl. El.-Ing. ETH Philipp Spaeti**

## Abstract Deutsch

Diese Bachelor-Thesis ist im Rahmen des Forschungsprojekts «Roboterbasierte Vegetationskontrolle» und in Zusammenarbeit mit der SBB als Industriepartner entstanden. Das Projekt entwickelt einen autonomen Industrieroboter namens S-bot zur Bekämpfung von Bewuchs auf Gleisanlagen. Die Thesis beschreibt die Entwicklung eines webbasierten Benutzerinterfaces (WebUI) zur Fernüberwachung und -steuerung des S-bots. Das WebUI ermöglicht es Benutzern, den S-bot aus der Ferne zu überwachen, seine Position zu verfolgen, Roboterdaten abzurufen und Steuerbefehle zu senden. Im Entwicklungsprozess wurden umfangreiche Technologierecherchen durchgeführt, eine bidirektionale Datenverbindung implementiert und eine cloudbasierte Lösung mithilfe des Azure IoT Hub integriert. Die Anforderungen wurden durch iterative Prozesse und die Bewertung verschiedener Interessengruppen priorisiert. Die Ergebnisse zeigen, dass das WebUI die grundlegenden Funktionsanforderungen erfüllt und eine effektive Fernüberwachung und -steuerung des S-bots ermöglicht. Die Architektur und Technologien bieten eine Grundlage für zukünftige Erweiterungen und Forschungsarbeiten, welche Verbesserungen, zusätzliche Funktionalitäten und Analysemöglichkeiten beinhalten, um den S-bot weiter zu optimieren.

## Abstract Englisch

This Bachelor's thesis was written within the research project "Robotbased Vegetation Control" and in cooperation with SBB as the industrial partner. The project aims to develop an autonomous industrial robot called S-bot to control vegetation on railway tracks. The thesis describes the development of a web-based user interface (WebUI) for remote monitoring and control of the S-bot. The WebUI allows users to remotely monitor the S-bot, track its position, retrieve robot data, and send commands. The development process involved extensive research in technology, implementing a bidirectional data connection, and integrating a cloud-based solution using Azure IoT Hub. Requirements were prioritized through iterative processes and the evaluation of various stakeholders. The results show that the WebUI meets the basic requirements and enables effective remote monitoring and control of the S-bot. The architecture and technologies provide a basis for future research, which could include possible improvements, additional functionalities, and further analytics to optimize the S-bot.

Ort, Datum **Luzern, 09.06.2023**  
**© Samuel Huser, Hochschule Luzern – Technik & Architektur**

---

Alle Rechte vorbehalten. Die Arbeit oder Teile davon dürfen ohne schriftliche Genehmigung der Rechteinhaber weder in irgendeiner Form reproduziert noch elektronisch gespeichert, verarbeitet, vervielfältigt oder verbreitet werden.

Sofern die Arbeit auf der Website der Hochschule Luzern online veröffentlicht wird, können abweichende Nutzungsbedingungen unter Creative-Commons-Lizenzen gelten. Massgebend ist in diesem Fall die auf der Website angezeigte Creative-Commons-Lizenz.

## Vorwort

Die vorliegende Bachelor-Thesis (BAT) wurde aus dem Forschungsprojekt Roboterbasierte Vegetationskontrolle (RoVeKo) der Hochschule Luzern initiiert, welches durch die SBB als Industriepartner beauftragt wurde. Die Bachelor-Thesis des Studiengangs Digital Engineer | Robotik & Big Data der Hochschule Luzern (HSLU) Technik & Architektur ist eine individuelle, komplexe Projektarbeit, welche im Kontext der Vertiefungsrichtung "Digital Transformation" steht. Das Ziel der BAT besteht darin, die zentralen Elemente der Bachelor-Ausbildung Digital Engineer in einem praxisbezogenen Kontext zu vereinen. Dabei werden verschiedene Kompetenzbereiche abgedeckt. Die Fachkompetenzen sollen die eigenständige Umsetzung der Projektaufgabe auf dem Niveau "Stand der Technik" mit Fokus auf wirtschaftlich und ökologisch vertretbaren Lösungen ermöglichen. Die Methodenkompetenzen umfassen systemisches Denken, kreative Lösungsansätze, Bewertungsverfahren und den effektiven Einsatz von Recherche- und Analysetools. Im Bereich der Personalkompetenzen wird das kritische Hinterfragen, verantwortungsbewusstes Handeln, Selbstständigkeit, Belastbarkeit, Teamfähigkeit sowie die Fähigkeit zur Konfliktlösung und sicheren Kommunikation mit Auftraggebern und Behörden betont. Das übergeordnete Ziel ist es, die erworbenen Fähigkeiten und Kompetenzen in einem praxisnahen Umfeld anzuwenden und die Ergebnisse überzeugend zu präsentieren.

Besonderer Dank gebührt Lukas Tanner für seine Vertretung als Industriepartner sowie dem Experten Dipl. El.-Ing. ETH Philipp Spaeti, die beide ihre umfangreiche Industrieerfahrung einbrachten. Zusätzlich geht ein Dank an Lukas Müller, dem Studiengangleiter Digital Engineer, für die Flexibilität und Übernahme der administrativen Aufgaben.

Die Unterstützung, fachliche Expertise und wertvollen Beiträge von den Marco Grossmann, Jonas Düggeli und Prof. Dr. Clemente Minonne waren entscheidend für den Erfolg dieser Arbeit. Durch ihre engagierte Zusammenarbeit und wertvollen Einblicke trugen sie massgeblich zur Umsetzung und Weiterentwicklung des Projekts bei.

Ein ganz besonderer Dank gebührt der Betreuerin Dipl. El.-Ing. ETH Annina Blaas, die das Projekt während des gesamten Prozesses begleitete und unterstützte. Ihre fachkundige Betreuung, konstruktives Feedback und wertvolle Anregungen waren von unschätzbarem Wert und trugen massgeblich zum Gelingen dieser Arbeit bei. Die Zusammenarbeit mit dem Forschungsprojekt RoVeKo, der HSLU und der SBB hat nicht nur für diese Arbeit, sondern auch für die persönliche und berufliche Entwicklung eine grosse Bedeutung. Die gewonnenen Erkenntnisse, Erfahrungen und Netzwerke werden einen nachhaltigen Einfluss auf meine weitere Karriere haben.

# Inhalt

<b>Vorwort</b>	<b>2</b>
<b>Abbildungsverzeichnis</b>	<b>3</b>
<b>Abkürzungsverzeichnis</b>	<b>5</b>
<b>1 Einleitung</b>	<b>7</b>
<b>2 Grundlagen</b>	<b>9</b>
2.1 Ausgangslage . . . . .	9
2.1.1 S-bot . . . . .	9
2.1.2 Anforderungen aus dem MVP . . . . .	10
2.1.3 Agiles Projektmanagement . . . . .	10
2.2 Technologien . . . . .	12
2.2.1 ROS . . . . .	12
2.2.2 Standards, Protokolle und Frameworks . . . . .	13
2.2.3 Geolocation . . . . .	15
2.3 WebUIs . . . . .	16
2.3.1 Evaluation Husqvarna-UI . . . . .	16
2.3.2 WebUIs mit ROS . . . . .	18
2.3.3 Architektur . . . . .	21
2.4 Cloud . . . . .	22
2.4.1 Cloud Robotics . . . . .	22
2.4.2 Cloud mit Azure IoT . . . . .	23
<b>3 Methodik</b>	<b>27</b>
3.1 Vorgehen zur Anforderungserhebung . . . . .	27
3.1.1 Definition . . . . .	28
3.1.2 Gewichtung . . . . .	28
3.1.3 Beschreibung . . . . .	28
3.2 Erarbeitung WebUI . . . . .	28
3.3 Konzept der Cloud-Architektur . . . . .	29
<b>4 Anforderungserhebung, Konzeption und Entwicklung WebUI &amp; Azure Cloud</b>	<b>30</b>
4.1 Anforderungserhebung . . . . .	30
4.1.1 Erste Anforderungserhebung . . . . .	30
4.1.2 Gewichtung der Features . . . . .	31
4.1.3 Konsolidierung der Features . . . . .	33
4.2 Konzept und Architektur des WebUI-Grundgerüsts . . . . .	35
4.2.1 Software-Architektur WebUI . . . . .	37
4.2.2 Webpage . . . . .	38
4.3 Aufbau der Cloud mit Azure . . . . .	41
4.3.1 IoT Edge . . . . .	41
4.3.2 Cloud Services . . . . .	42
<b>5 Diskussion und Ausblick</b>	<b>43</b>
5.1 Diskussion . . . . .	43
5.1.1 Anforderungserhebung . . . . .	43
5.1.2 Ausarbeitung WebUI . . . . .	43
5.1.3 Konzipierung Cloud-Architektur . . . . .	44
5.2 Ausblick . . . . .	44
5.2.1 Zweite Iteration der Anforderungserhebung . . . . .	44
5.2.2 Demo-Use-Case . . . . .	45
5.2.3 Weitere Cloudservices . . . . .	46
<b>Literaturverzeichnis</b>	<b>47</b>
<b>Anhang</b>	<b>49</b>

# Abbildungsverzeichnis

1	Prototyp S-bot während eines Feldtests	7
2	Design Konzept S-bot	9
3	Kosten für Änderungen über Projektlaufzeit	10
4	Kanban Board	11
5	ROS Computation Graph Level [8]	12
6	Crossbar.io	15
7	Husqvarna Fleet Services - Bestand	16
8	Husqvarna Automower App	17
9	ROS Explorer	18
10	Webviz	19
11	Foxglove Studio	19
12	ROSboard	20
13	Zenoh für ROS2	20
14	Serviceorientierte Architektur SOA	21
15	RoboWeb System	21
16	Generische Architektur für ein Cloud Robotics System	22
17	Generelle Azure IoT Architektur	23
18	Azure IoT Central-basierte Architektur	24
19	Azure-Dienste in einer PaaS-basierten IoT-Architektur	24
20	Azure IoT Edge-Runtime	25
21	Twin-Peaks-Model	27
22	Anforderungserhebung nach ©iSREM	28
23	Stakeholderanalyse - blau markiert die befragten VertreterInnen	30
24	Roadmap Entwicklung WebUI	34
25	globale Architektur	35
26	WebUI Architektur	37
27	Datenfluss WebUI Architektur	38
28	Ausschnitt aus dem WebUI mit Feature der bidirektionalen Verbindung	38
29	Canvas des WebUI mit Feature von GPS-Koordinaten	39
30	Sequenzdiagramm für das WebUI	40
31	Cloud Architektur	41
32	Telemetriedaten nach Drilldown auf dem Storage Container	42
33	Vorlage einer Paarvergleichs-Matrix für ein Epic aus [48]	45



## Abkürzungsverzeichnis

- ADM** Anforderungsdefinitions-Matrix. 44
- AMQP** Advanced Message Queuing Protocol. 46
- API** Application Programming Interface. 13
- ASM** Anforderungsspezifikations-Matrix. 45
- BAT** Bachelor-Thesis. 2
- BLOB** Binary Large Object. 26
- ConOps** Concept of Operations. 11
- CSS** Cascading Style Sheets. 14
- DOM** Document Object Model. 14
- DPS** Device Provisioning Service. 24
- GIS** Geographisches Informationssystem. 16
- GNU** GNU's Not Unix. 16
- GPS** Global Positioning System. 15
- GUI** Graphical User Interface. 14
- HSLU** Hochschule Luzern. 2
- HTML** Hypertext Markup Language. 14
- HTTP** Hypertext Transfer Protocol. 13
- IaaS** Infrastructure as a Service. 23
- IoRT** Internet of Robotic Things. 23
- IoT** Internet of Things. 23
- IP** Internet Protocol. 14
- iSREM** integrated Structured Requirements Elicitation Method. 27
- JSON** JavaScript Object Notation. 13
- KI** Künstliche Intelligenz. 25
- MVP** Minimum Viable Product. 10
- NIST** National Institute of Standards and Technology. 23
- OpsCon** Operational Concept. 11
- OS** Operating System. 9
- OSM** OpenStreetMap®. 15
- PaaS** Platform as a Service. 23
- Power BI** Power Business Intelligence. 25
- RaaS** Robot as a Service. 22

**REST** Representational State Transfer. 13

**ROS** Robot Operating System. 9

**RoVeKo** Roboterbasierte Vegetationskontrolle. 2, 7

**RPC** Remote Procedure Call. 12

**RTOS** Real-Time Operating System. 24

**S-bot** Prototyp autonomer Roboter des RoVeKo-Projektes. 7

**SaaS** Software as a Service. 23

**SBB** Schweizerische Bundesbahnen. 2

**SDK** Software Development Kit. 23

**SLAM** Simultaneous Localization and Mapping. 9

**SOA** Service Oriented Architecture. 21

**SOAP** Simple Object Access Protocol. 13

**TCP** Transmission Control Protocol. 14

**TF** Transform Library. 22

**UI** User Interface. 7

**URDF** Unified Robot Description Format. 22

**URI** Uniform Resource Identifiers. 13

**URL** Uniform Resource Locator. 13

**UX** User Experience. 23

**WAMP** Web Application Messaging Protocol. 14

**WebUI** Webclientbasiertes User Interface. 8

**WSDL** Web Services Description Language. 13

**XML** eXtensible Markup Language. 13

# 1 Einleitung

Die Bekämpfung von Bewuchs auf Gleisanlagen in der Schweiz erfolgt häufig mithilfe des umstrittenen Herbizids Glyphosat, um die Sicherheit, Verfügbarkeit und Langlebigkeit der Anlagen zu gewährleisten. Derzeit stehen keine kostengünstigen Alternativen zur Verfügung. Die chemiefreie, roboterbasierte Unkrautbekämpfung hat sich in der Landwirtschaft als vielversprechend erwiesen und bietet auch für die Bahninfrastruktur neue Möglichkeiten. Im Rahmen des Forschungs- und Entwicklungsprojekts “Roboterbasierte Vegetationskontrolle”, kurz RoVeKo, an der Hochschule Luzern Technik & Architektur wird in Zusammenarbeit mit der SBB als Industriepartner und Auftraggeber das Ziel verfolgt, einen autonomen, für den Bahnbetrieb geeigneten Roboter zur Vegetationskontrolle zu entwickeln und zu erproben. Das Hauptziel besteht darin, einen kosteneffizienten Roboter inklusive der nötigen Peripheriegeräte zur Bekämpfung von Pflanzenbewuchs zu entwickeln und mindestens einen Technologiereifegrad zu erreichen, der den Übergang zur Vorserienproduktion ermöglicht.

Seit 2018 wurden in verschiedenen Disziplinen umfangreiche Forschungs- und Entwicklungsarbeiten durchgeführt. Der aktuelle Entwicklungsstand umfasst einen funktionsfähigen Roboter mit dem Namen S-bot, der derzeit manuell gesteuert werden kann. Eine Abbildung des Prototyps ist in Abbildung 1 zu sehen.



Figure 1: Prototyp S-bot während eines Feldtests

Trotz dem vorausgesetzten autonomen Arbeiten im Feld, muss eine Fernüberwachung und auch -steuerung möglich sein. Wie eine Bedienung des Benutzers vor Ort, aber auch jene des Fernlenkers, bzw. des Überwachers aussehen könnte, ist noch nicht klar definiert. Im Rahmen der vorliegenden Arbeit soll vor allem die Überwachung und damit auch die Diagnostik und Analyse genauer evaluiert werden.

Um eine geeignete Überwachung und Fernsteuerung des S-bots zu gewährleisten, braucht es ein benutzerfreundliches UI, welches die Möglichkeit bietet, einerseits Informationen des Roboters zu visualisieren und andererseits Befehle und Aktionen auszuführen. Zu Beginn wurden innerhalb der Projektskizze folgende grundlegende Funktionsanforderungen definiert, welche als zentrale Forschungsthemen zu verstehen sind:

1. Bidirektionale Datenverbindung zwischen S-bot (Server) und WebUI (Client).
2. Informationen des S-bot (Metriken, Position, Batterieladestatus,...) auf dem Web-UI darstellen.
3. Roboterdaten lesen und visualisieren, Roboterdaten und Steuerungsbefehle schreiben.
4. Karten in spezifischem Format (Gridmap SBB) im Web-UI laden, darstellen und an den S-bot für dessen Navigation weiterleiten.

Angedacht war also die Konzipierung und Entwicklung eines Webclientbasierten User Interfaces, kurz WebUI, welches vor allem als Diagnosetool dienen soll, um bei Fehlern oder Problemen im Feld schnell reagieren zu können. Ein weiterer, wichtiger Anwendungsfall für das WebUI ist der Einsatz in (Kunden-)Demos. Für die Steuerung vor Ort im Follow-Me-Modus ist gezwungenermassen ein anderes Tool nötig. Auf die Begründung dieser Argumentation wird in Kapitel 4.2 eingegangen.

Auch wurde schnell klar, dass es zahlreiche Features gäbe, welche für ein WebUI wünschenswert wären. Das Bestimmen, welche Informationen relevant sind, wird in einer ersten Iteration der Anforderungserhebung 2.1 evaluiert. Das Vorgehen dazu ist in Kapitel 3.1 genauer beschrieben. Dafür wurden verschiedene Stakeholders befragt, um deren Vorstellungen zu Funktionalitäten innerhalb des WebUIs aufzunehmen. Dadurch konnte eine strukturierte Anforderungsliste abgeleitet werden, welche einen iterativen Entwicklungsprozess in Gang setzen soll 4.1.

Der Schwerpunkt und der Fokus der Arbeit gilt jedoch der Erfüllung der initial bestimmten Funktionsanforderungen durch das WebUI. Dazu ist durch agile Prozesse, nach einer ausführlichen Recherche 2.3 und Vorbereitung 3.2, gleichzeitig und iterativ ein Konzept und Grundgerüst für die bidirektionale Datenverbindung entwickelt worden 4.2.1.

Neben dem WebUI steht die unidirektionale Verbindung zu der Cloud und dessen Architektur ebenfalls im Fokus. Dazu ist ebenfalls, nach gründlicher Recherche 2.4 und Vorbereitung 3.3, ein Konzept ausgearbeitet und dessen Nachweis realisiert worden 4.3.

Die vorliegende wissenschaftliche Dokumentation folgt einem durchgängigen roten Faden, der sich wie folgt strukturiert: Im ersten Teil eines jeden Kapitels wird das Thema Anforderungsmanagement behandelt. Der zweite Teil widmet sich dem WebUI, während der dritte Teil die Thematik der Cloud-integration behandelt.

Zur besseren Lesbarkeit wird in der vorliegenden Arbeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Es wird das generische Maskulinum verwendet, wobei beide Geschlechter gleichermaßen gemeint sind.

## 2 Grundlagen

Dieses Kapitel widmet sich den grundlegenden Prinzipien und der Ausgangslage, aus der die Projektarbeit startet. Zudem soll es den aktuellen Stand des Wissens, auf dem die Arbeit aufbaut, vermitteln. Es erfolgt eine kurze Darstellung des aktuellen Entwicklungsstands des S-bot und seiner Systeme. Zudem werden die Ergebnisse der Recherche zu WebUIs, Technologien und Architekturen präsentiert, die den aktuellen Kenntnisstand bilden und als Grundlage für das Design des WebUI dienen. Des Weiteren werden theoretische Grundlagen, Begriffsdefinitionen, Modelle und Architekturen im Bereich der Cloud vorgestellt.

### 2.1 Ausgangslage

Die Entwicklung eines autonom agierenden Roboters erfordert nicht nur ein effektives Projektmanagement, sondern auch eine gründliche Erfassung der Anforderungen. Basierend auf dem aktuellen Entwicklungsstand existieren bereits verschiedene Soft- und Hardwaresysteme, auf die in dieser Arbeit aufgebaut wird.

#### 2.1.1 S-bot

Der Prototyp, welcher in Abbildung 1 zu sehen war, verfügt über folgende Technologien:

- Aufbau:
  - 4-Rad-Antrieb mit elektrischen Motoren
  - Einzelradaufhängung
  - Schneidwerk
  - Li-Po-Akkus
- Sensoren
  - Ouster Lidar
  - Alphasense Kamera mit Visual Simultaneous Localization and Mapping (SLAM) System
  - ublox GPS Empfänger
  - IMU
- Intelligenz
  - Intel nuc12 mit Linux-basiertem OS Ubuntu 20.04
  - Motorcontroller und Encoder
  - Robot Operating System (ROS) mit der Distribution Noetic Ninjemys

Die Alphasense Kamera ist momentan nicht mehr in Betrieb. Eine Kamera im System ist aber vorgesehen. Im Moment wird eine Überarbeitung des Chassis und Designs vorgenommen. Ein neuartiges Designkonzept ist in Abbildung 2 zu sehen.



Figure 2: Design Konzept S-bot

### 2.1.2 Anforderungen aus dem MVP

Das Minimum Viable Product (MVP) des RoVeKo-Projektes unterscheidet zwischen Muss- und Wunsch-Anforderungen. Die relevanten für die vorliegende Dokumentation sind die Folgenden:

#### Muss-Anforderungen:

1. Keine Steuerung des Roboters durch unbefugte Personen. Zugriff auf die Steuerung nur nach Authorisierung.
2. Der Roboter muss durch einen Beobachter in einen sicheren Zustand und eine sichere Lage wechseln können oder diese autonom einnehmen. Manueller Nothalt durch lokalen Bediener oder über Fernsteuerung.
3. Der Roboter muss sich autonom fortbewegen und die Vegetation autonom entfernen können. Betrieb ohne Begleitpersonen vor Ort möglich für die Use Cases, bei denen dies so vorgesehen ist.

#### Wunsch-Anforderungen:

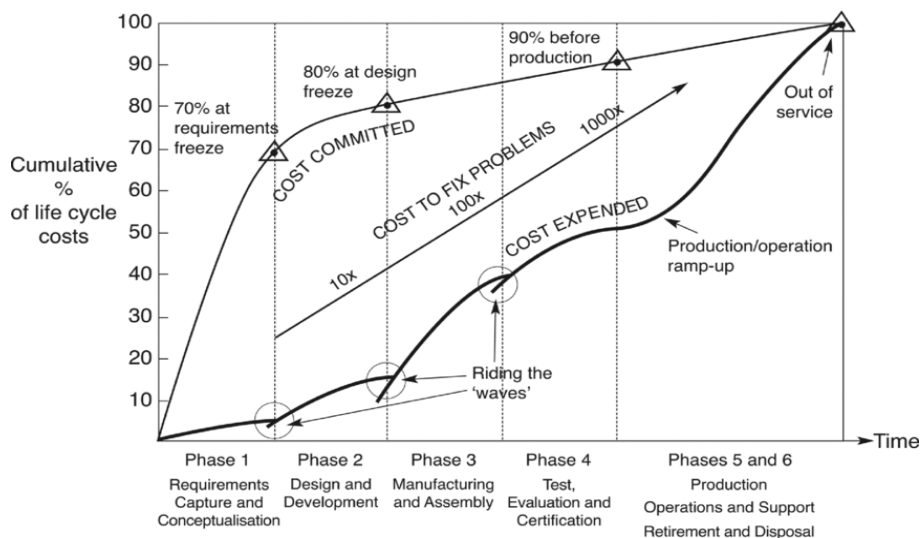
1. Das Arbeiten in einem Roboterverbund (Schwarm) soll geprüft werden (inkl. ein direkter Datenaustausch zwischen den Robotern).
2. Die Robotereinsätze müssen digital protokolliert und ausgewertet werden können (Weg, Zeit, Zustände, Lage, wichtige Entscheide).
3. Die Daten, an welcher Position der Roboter wieviel Vegetation erkannt und entfernt hat, müssen digital protokolliert und ausgewertet werden können.
4. Störungen am Roboter müssen eine Meldung auslösen, welche an zentraler Stelle als Fehlercode ausgelesen werden können.

Basierend auf dem aktuellen Entwicklungsstand und diesen Vorgaben existieren bereits verschiedene Systeme, auf die in dieser Arbeit aufgebaut wird.

### 2.1.3 Agiles Projektmanagement

Das Problem für die beste Wahl für ein Vorgehensmodell (Agil oder Wasserfall-Modell) ist oft nicht mehr in der Auswahl der verfügbaren Methoden, sondern in der adäquaten Nutzung der vorhandenen Methoden. Jedes Vorgehensmodell ist für bestimmte Projekttypen mit definierten Kriterien besonders gut geeignet [1].

Nach dem agilen Manifest der Softwareentwicklung fördern agile Prozesse eine nachhaltige Entwicklung [2]. Agiles Projektmanagement ist ein iterativer Ansatz für das Management von Softwareentwicklungsprojekten. Im Mittelpunkt stehen dabei fortlaufende Releases und die Einbeziehung von Kundenfeedback in jeder Iteration. Das Arbeiten mit agilen Methoden bedeutet, dass der eingeschlagene Weg flexibel bleibt. Dennoch sollte beachtet werden, dass die Kosten für Änderungen stark ansteigen [3]. Gerade deshalb ist eine iterative Anforderungserhebung mit konstanter Überprüfung, Planung und Implementation sehr wichtig.



Bildquelle: Rotorcraft virtual engineering [3]

Figure 3: Kosten für Änderungen über Projektlaufzeit

Bei der Entwicklung von Softwaresystemen sind Agile-Frameworks wie Scrum oder Kanban stark verbreitet. In der Entwicklung des RoVeKo-Projektes ist "Jira" von Atlassian das zugehörige Tool für die Projektierung in Verwendung.

Scrum ist eine Methodik des Projektmanagements, die nicht nur die Entwicklung von Produkten beeinflusst, sondern auch die gesamte Organisation mit ihren Rollenverteilungen betrifft. Scrum fördert das Wachstum der Teams und trägt zur Etablierung eines neuen Managementverständnisses bei, wie von Gloger [4] beschrieben. Aufgrund des hauptsächlichlichen Fokus auf der Entwicklung von Features im WebUI erwies sich das Kanban-Projektmanagement-Framework als passende Wahl, da es eine einfache und effektive Methode bietet.

### 2.1.3.1 Kanban

Kanban ist ein Vorgehensmodell der agilen Softwareentwicklung, das auf visuelle Aufgabenverwaltung setzt [5]. Mithilfe von Karten, Spalten und kontinuierlicher Verbesserung in Kanban Boards können Aufgaben in Backlog, Doing, ToDo und Done eingeteilt werden. Bei Agile Teams steht jede Karte für eine User Story. Auf dem Board helfen diese visuellen Signale, schnell zu verstehen, woran gerade gearbeitet wird.



Bildquelle: [https://commons.wikimedia.org/wiki/File:Abstract\\_Kanban\\_Board.svg](https://commons.wikimedia.org/wiki/File:Abstract_Kanban_Board.svg)

Figure 4: Kanban Board

Jede Karte zeigt den aktuellen Status der Aufgabe an und kann zusätzliche Informationen wie Priorität, Verantwortlichkeiten und geschätzter Zeit enthalten. Verwendete Methoden und einige Begriffe werden im Folgenden erklärt. Da beim Projekt mit dem Kanban Board von Jira / Atlassian gearbeitet wurde, wird für ausführlichere Informationen auf <https://www.atlassian.com/de/agile/kanban> verwiesen.

### 2.1.3.2 Produkt-Roadmap &-Backlog

Eine Produkt-Roadmap dient als Projektplan und skizziert die Entwicklung eines Produkts oder einer Lösung über die Zeit. Product Owner nutzen sie, um zukünftige Produktfunktionen und Veröffentlichungszeitpunkte festzulegen. Das Produkt-Backlog ist eine priorisierte Liste von Aufgaben, abgeleitet von der Roadmap und ihren Anforderungen [6].

### 2.1.3.3 Stories, Epics & Initiativen

Stories sind narrative Beschreibungen von Szenarien, die aus der Perspektive des Benutzers oder Kunden (Concept of Operations (ConOps)) formuliert werden. Sie dienen dazu, kurze Anforderungen oder Anfragen in einer verständlichen Sprache zu erfassen und ermöglichen eine effektive Kommunikation zwischen Kunden und Stakeholdern. Dabei stellen sie eine gemeinsame Basis für das Verständnis und die Zusammenarbeit dar. Epics hingegen sind grössere Aufgabeneinheiten, die in kleinere Stories aufgeteilt werden können, um eine detaillierte Umsetzung zu ermöglichen. Initiativen wiederum sind eine Sammlung von Epics, die ein gemeinsames Ziel verfolgen und strategische Ausrichtung bieten.

### 2.1.3.4 Agile Anforderungen

Agile Anforderungen sind von unschätzbarem Wert für den Product Owner. Im Gegensatz zur detaillierten Spezifizierung (Operational Concept (OpsCon)) im klassischen Anforderungsmanagement basieren agile Anforderungen

auf einem gemeinsamen Verständnis des Kunden, das vom Product Owner, Designer und Entwicklerteam geteilt wird. Durch dieses gemeinsame Verständnis können sich Product Owner auf allgemeinere Anforderungen konzentrieren und die Implementierungsdetails den qualifizierten Entwicklern überlassen. Dabei müssen die Anforderungen die folgende Eigenschaften erfüllen: Vollständigkeit, Zugänglichkeit, Überprüfbarkeit und Priorisierbarkeit.

## 2.2 Technologien

Bei der Entwicklung eines WebUIs für einen Roboter mit ROS kommen verschiedene Technologien zum Einsatz.

### 2.2.1 ROS

ROS Noetic Ninjemys wurde hauptsächlich für die Kompatibilität mit Ubuntu 20.04 (Focal Fossa) entwickelt. "ROS ist ein Open-Source-Meta-Betriebssystem für die Entwicklung von Robotik-Anwendungen" [7]. Es bietet eine Vielzahl von Werkzeugen, Bibliotheken und Funktionen, die es Entwicklern ermöglichen, komplexe Robotik-Systeme zu erstellen, zu testen und zu deployen. ROS ist auf Modularität und Wiederverwendbarkeit ausgelegt, was es zu einer beliebten Wahl in der Robotik-Industrie macht. Es unterstützt zahlreiche Programmiersprachen und ermöglicht die Integration von Hardwarekomponenten von verschiedenen Herstellern.

#### 2.2.1.1 ROS-Computation-Graph

Der ROS-Computation-Graph stellt das Netzwerk von ROS-nodes und deren Verbindungen innerhalb eines ROS-Systems dar. Er visualisiert den Datenfluss zwischen verschiedenen nodes in einem ROS-System und zeigt, wie Informationen über topics, services und actions ausgetauscht werden. Der Computation-Graph bietet einen Überblick über die gesamte Systemarchitektur und die Kommunikationswege zwischen den nodes. ROS ist nicht darauf ausgelegt, deterministische Echtzeitanforderungen mit extrem geringen Latenzzeiten zu erfüllen, wie es beispielsweise in sicherheitskritischen oder hochpräzisen Anwendungen erforderlich ist. Daher wird ROS als Soft-Realtime eingestuft.

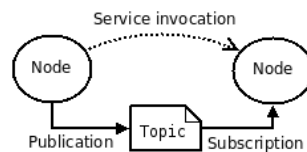


Figure 5: ROS Computation Graph Level [8]

Im Folgenden werden nur die verwendeten Dienste genauer erläutert.

#### 2.2.1.2 nodes

Ein ROS-node ist ein Prozess, der eine bestimmte Rechenaufgabe ausführt oder eine bestimmte Funktionalität innerhalb des Systems bereitstellt. Nodes werden zu einem Graphen zusammengefasst und kommunizieren miteinander über Streaming-topics, RPC-services und den Parameter-Server. Ein Robotersteuersystem besteht normalerweise aus vielen nodes. Nodes, die RPCs durchführen müssen, d.h. eine Antwort auf eine Anfrage erhalten, sollten stattdessen services verwenden. Ausserdem gibt es den Parameter Server, der kleine Mengen von Daten verwaltet. Diese und genauere Informationen zu nodes sind auf der offiziellen Webseite von ROS [9] zu finden.

#### 2.2.1.3 topics

Ein ROS-topic ist ein benannter Bus, der die Kommunikation zwischen verschiedenen nodes in einem verteilten ROS-System ermöglicht. Es dient als Kommunikationskanal, über den nodes messages veröffentlichen und abonnieren können. Ein topic folgt einem Publish-Subscribe-Messaging-Muster, bei dem nodes Nachrichten in einem bestimmten topic veröffentlichen können und andere nodes dieses topic abonnieren können, um die Nachrichten zu empfangen und zu verarbeiten. Dieser Kommunikationsmechanismus ermöglicht eine lose Kopplung zwischen den nodes, was die Flexibilität und Skalierbarkeit der Systemarchitektur ermöglicht. Es kann mehrere Herausgeber und Abonnenten für ein topic geben. Topics sind für eine unidirektionale, strömende Kommunikation gedacht. Diese und genauere Informationen zu topics sind auf der offiziellen Webseite von ROS [10] zu finden.



#### 2.2.1.4 messages

Eine ROS-message ist eine einfache Datenstruktur, die aus typisierten Feldern besteht. Unterstützt werden primitive Standardtypen (Integer, Gleitkomma, Boolean usw.) sowie Arrays primitiver Typen. Nachrichten können beliebig verschachtelte Strukturen und Arrays enthalten. Die verschiedenen Typen verwenden die Standard-ROS-Namenskonventionen: *Name des Pakets/Name der.msg-Datei*. Zum Beispiel hat `std_msgs/msg/String.msg` den Nachrichtentyp `std_msgs/String`. Diese und genauere Informationen zu messages sind auf der offiziellen Webseite von ROS [11] zu finden.

#### 2.2.1.5 rosbrowser\_suite

“Rosbridge bietet eine JSON-API für ROS-Funktionen für Nicht-ROS-Programme. Es gibt eine Vielzahl von Frontends, die mit rosbrowser zusammenarbeiten, einschliesslich eines WebSocket-Servers, mit dem Webbrowser interagieren können. Rosbridge\_suite ist ein Meta-Paket, das rosbrowser, verschiedene Front-End-Pakete, wie ein WebSocket-Paket und Hilfspakete enthält” [12]. Rosbridge ermöglicht die Kommunikation zwischen ROS und anderen Systemen oder Frameworks. Die rosbrowser\_suite besteht aus zwei Hauptkomponenten:

- Rosbridge Server: Dieser Server stellt eine WebSocket-basierte Schnittstelle bereit, über die externe Anwendungen mit ROS kommunizieren können. Er ermöglicht den bidirektionalen Austausch von Nachrichten und Daten zwischen ROS und anderen Systemen.
- Rosbridge Library: Diese Bibliothek bietet eine API und Tools, um die Entwicklung von Anwendungen zu erleichtern, die die Rosbridge-Server-Schnittstelle verwenden möchten. Sie enthält Client-Bibliotheken in verschiedenen Programmiersprachen (z.B. Python, JavaScript), die es externen Systemen ermöglichen, ROS-Nachrichten zu empfangen und zu senden.

### 2.2.2 Standards, Protokolle und Frameworks

Das Internet basiert auf einer Vielzahl von Standards und Protokollen, die die Kommunikation und den Austausch von Informationen zwischen verschiedenen Computern und Geräten ermöglichen. Diese Standards und Protokolle stellen sicher, dass Geräte unabhängig von ihrer Art, ihrem Betriebssystem oder dem Standort einfach miteinander kommunizieren können.

#### 2.2.2.1 REST

Representational State Transfer (REST) ist ein Architekturstil für den Entwurf von verteilten Systemen. Es basiert auf dem Konzept der Ressourcen, die über eindeutige URIs identifiziert und durch standardisierte HTTP-Methoden (GET, POST, PUT, DELETE) manipuliert werden können. REST ermöglicht die lose Kopplung zwischen Client und Server, indem es eine zustandslose Kommunikation fördert und den Fokus auf einheitliche Schnittstellen legt, die skalierbar, erweiterbar und einfach zu verstehen sind [13].

#### 2.2.2.2 SOAP

Simple Object Access Protocol (SOAP) ist ein Kommunikationsprotokoll, das verwendet wird, um strukturierte Informationen zwischen verschiedenen Systemen über das Internet auszutauschen. Es basiert auf XML und definiert eine standardisierte Methode zum Aufrufen von Remote-Prozeduren über das Netzwerk [14]. SOAP ermöglicht die Interaktion zwischen Client und Server durch den Austausch von XML-basierten Nachrichten, die Operationen, Parameter und Rückgabewerte enthalten. Es bietet eine plattformunabhängige und sprachneutrale Möglichkeit der Kommunikation, die in Web Services und anderen verteilten Systemen weit verbreitet ist.

#### 2.2.2.3 WSDL

Web Services Description Language (WSDL) ist eine XML-basierte Sprache, die zur Beschreibung von Webservices verwendet wird. WSDL-Definitionen bieten dem Client Informationen über den Aufbau einer Anfrage an einen Webdienst und beschreiben die Schnittstelle, die vom Anbieter des Webdienstes bereitgestellt wird. Es beschreibt auch die verwendeten Nachrichtenformate, den Zugriffspfad URL des Webdienstes und die unterstützten Transportprotokolle. Durch WSDL kann auf die Funktionalitäten eines Webdienstes zugegriffen werden, ohne detaillierte Kenntnisse über dessen interne Implementierung zu haben. Es ermöglicht die Interoperabilität zwischen verschiedenen Plattformen und Programmiersprachen, da WSDL eine einheitliche Beschreibungssprache für Webdienste bietet [15].

#### 2.2.2.4 CSS

Cascading Style Sheets (CSS) ist eine Stylesheet-Sprache, die verwendet wird, um das Aussehen und das Layout von HTML-Dokumenten zu definieren. Dadurch kann eine logische Struktur, sowie eine einheitliche Formatierung und Erscheinung in der Dokumentpräsentation erreicht werden.

Bootstrap ist ein populäres CSS-Framework, das vorgefertigte CSS-Styles und JavaScript-Komponenten bereitstellt [16]. Bootstrap enthält eine umfangreiche Sammlung von vorgefertigten CSS-Klassen und -Komponenten, die einfach in HTML-Code integriert werden können.

#### 2.2.2.5 Vue.js

“Vue ist ein JavaScript-Framework zur Erstellung von Benutzeroberflächen. Es baut auf Standard-HTML, -CSS und -JavaScript auf und bietet ein deklaratives und komponentenbasiertes Programmiermodell, das hilft, einfache oder komplexe Benutzeroberflächen effizient zu entwickeln” [17]. Das Konzept hinter Vue.js ist eine Verbindung zwischen der Objektinstanz und der Ansicht. Mittels `new Vue(){...}` wird ein Objekt mit dem HTML-GUI verbunden und somit wird jede Änderung an verknüpften Daten sofort im GUI wiedergegeben. Direktive sind deklarative Angaben, wie sich der DOM bei Änderungen verhalten soll.

#### 2.2.2.6 WebSocketProtocol

Für ein WebUI wird eine bidirektionale Kommunikation zwischen Client und Server benötigt. Bei einer Verwendung von HTTP als Protokoll können verschiedene Probleme auftreten:

Für das Senden und Empfangen von Informationen muss der Server jeweils für jeden Client eine individuelle TCP-Verbindung erstellen. Ausserdem muss das clientseitige Skript die Zuordnungen der Verbindungen loggen. Dies führt zu einem grossen Overhead, da jede Nachricht einen eigenen Header generiert.

Die Lösung dieser Umstände ist die Verwendung einer einzigen TCP-Verbindung für eine bidirektionale Kommunikation. Genau das bietet das WebSocket-Protokoll.

“Das WebSocket-Protokoll ermöglicht die Zwei-Wege-Kommunikation zwischen einem Client, auf dem ein nicht vertrauenswürdiger Code in einer kontrollierten Umgebung läuft, und einem entfernten Host, der sich für die Kommunikation mit diesem Code entschieden hat. Das hierfür verwendete Sicherheitsmodell ist das von Webbrowsern üblicherweise verwendete ursprungsbasierte Sicherheitsmodell. Das Protokoll besteht aus einem Eröffnungs-Handshake, gefolgt von einem einfachen Nachrichtenrahmen, der über TCP gelegt wird. Ziel dieser Technologie ist es, einen Mechanismus für browserbasierte Anwendungen bereitzustellen, die eine Zwei-Wege-Kommunikation mit Servern benötigen, ohne dass mehrere HTTP-Verbindungen geöffnet werden müssen (z.B. mit XMLHttpRequest oder <iframe>s und langem Polling)” [18].

Nach erfolgreichen Handshakes von Client und Server kann die Datenübertragung erfolgen [19]. Durch die bidirektionale Kommunikation können beide Seiten unabhängig von der anderen nach Belieben Daten senden. Sendet ein Peer einen Kontrollrahmen (control frame) wird ein abschliessender Handshake initiiert, welcher einfacher ist und auf dem TCP closing handshake basiert. Damit wird die Verbindung beendet.

Konzeptuell ist das WebSocketProtocol eine Schicht über TCP mit folgenden Aufgaben [20]:

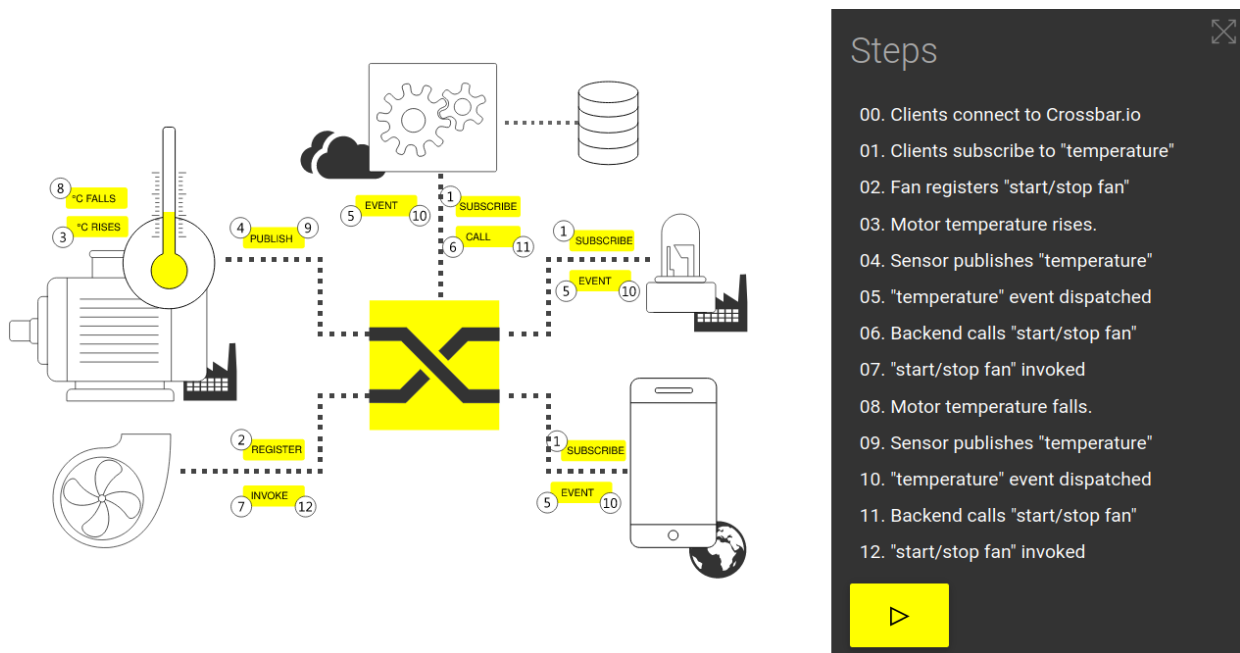
- Hinzufügen eines auf dem Ursprung basierenden Sicherheitsmodells für Browser.
- Hinzufügen eines Adressierungs- und Protokollbenennungsmechanismus zur Unterstützung mehrerer Dienste an einem Port und mehrerer Hostnamen an einer IP-Adresse.
- Einen Framing-Mechanismus auf TCP legen, um zum IP-Paketmechanismus zurückzukehren, auf dem TCP aufbaut, jedoch ohne Längenbegrenzung.
- Einen zusätzlichen abschliessenden Handshake enthalten, der so konzipiert ist, dass er bei Vorhandensein von Proxys und anderen Vermittlern funktioniert.

Das WebSocket Protokoll ist nicht REST-konform. Obwohl WebSockets und REST unterschiedliche Protokolle sind, können sie dennoch in Kombination verwendet werden. Zum Beispiel kann eine REST-API genutzt werden, um Ressourcen zu verwalten, während WebSockets für Echtzeitaktualisierungen oder Benachrichtigungen verwendet werden.

#### 2.2.2.7 WAMP

Web Application Messaging Protocol (WAMP) ist ein offenes Protokoll auf Anwendungsebene, das zwei Nachrichtenmuster bietet: Geroutete Remote Procedure Calls und Publish & Subscribe. Es verwendet verschiedene Serializer für die Nachrichtenkodierung (z.B. JSON) und kann über verschiedene Transporte laufen (z.B. WebSocket-Subprotokoll).

Das WAMP-Protokoll ist ein Gemeinschaftsprojekt und die Spezifikation wird unter einer offenen Lizenz kostenlos zur Verfügung gestellt, damit jeder sie nutzen und implementieren kann. Der ursprüngliche Entwurf und Vorschlag wurde von Crossbar.io-Entwicklern im Jahr 2012 erstellt und die WAMP-Entwicklung wird seitdem von Crossbar.io (dem Unternehmen) gesponsert [21]. Crossbar.io ist eine Open-Source-Netzwerkplattform für verteilte und Microservice-Anwendungen. Es handelt sich um eine funktionsreiche, skalierbare, robuste und sichere Implementierung von WAMP. Es bietet eine stabile und sichere Verbindung zwischen verschiedenen Komponenten einer Anwendung, unterstützt verschiedene Programmiersprachen und enthält eine Reihe von Funktionen wie Lastverteilung, Skalierbarkeit und hohe Verfügbarkeit. Crossbar.io kann verwendet werden, um Anwendungen auf verschiedenen Plattformen wie Web, Mobilgeräte, IoT-Geräte und Cloud-Plattformen zu verbinden.



Bildquelle: <https://crossbar.io/>

Figure 6: Crossbar.io

### 2.2.3 Geolocation

Für den letzten Punkt der Funktionsanforderungen ist eine Recherche bezüglich verschiedenen Kartenformaten und Technologien durchgeführt worden. Für das Auslesen der aktuellen Position des S-Bot kann der ublox-GPS-Sensor verwendet werden. Der ROS-node dafür ist bereits entwickelt. Der node namens /ublox schreibt messages (UbloxInfo.msg) auf das topic /ublox\_info. Die messages enthalten die Koordinaten mit dem Datentyp float64

- Latitude: `message.WGS_84_latitude`
- Longitude: `message.WGS_84_longitude`
- Altitude: `message.WGS_84_altitude`

Detailliertere Informationen zum node ist auf dem zugehörigen gitlab-Projekt von RoVeKo zu finden: [https://gitlab.switch.ch/sbot/ublox\\_driver](https://gitlab.switch.ch/sbot/ublox_driver).

Für die Darstellung der Position auf dem WebUI gibt es verschiedene Möglichkeiten. Eine davon ist die Verwendung von OpenStreetMap® (OSM) [22]. Mittels `leaflet.js` können Positionsdaten auf der Karte angezeigt werden. Leaflet ist eine Open-Source-JavaScript-Bibliothek für mobilfreundliche interaktive Karten [23].

Mit Leaflet ist es sehr einfach, Kernfunktionen zu entwickeln. Weitere Funktionalitäten können durch Plugins von Drittanbietern implementiert werden. Wenn beispielsweise die Karten auch offline zur Verfügung stehen müssen, so bietet Leaflet verschiedene "Tile & image layers"-Plugins an, um die Karten mit sogenannten Vector Tiles darzustellen. Auf <https://openmaptiles.org/> können komplette Datensätze in verschiedenen Formaten von Kartenausschnitten heruntergeladen werden.

Eine andere Möglichkeit besteht darin, mittels QGIS die Tiles eines definierten Bereiches zu extrahieren. “QGIS ist ein benutzerfreundliches Open Source Geographisches Informationssystem (GIS), das unter der GNU General Public License lizenziert ist” [24]. Nachfolgend müssen die Tiles entsprechend dem Zoomlevel in die richtige Ordnerstruktur gebracht werden. Je nach Grösse und Level können so die Datensätze sehr gross werden, Details dazu sind auf [https://wiki.openstreetmap.org/wiki/Zoom\\_levels](https://wiki.openstreetmap.org/wiki/Zoom_levels) zu finden.

## 2.3 WebUIs

Ein angedachtes, benutzerfreundliches WebUI für den S-bot zu gestalten bedarf neben der Anforderungserhebung auch einer ausgedehnten Recherche zu bestehenden Überwachungssystemen. Die dazu verwendeten Technologien und Architekturen bieten die Grundlage für die Ausarbeitung einer massgeschneiderten Benutzeroberfläche.

### 2.3.1 Evaluation Husqvarna-UI

Fernüberwachungssysteme für Roboter gibt es in der Industrie bereits zahlreiche. Ein Beispiel dafür ist Husqvarna Fleet Services [25], welches einerseits ein WebUI zur Überwachung und die Automower App für die Steuerung verwenden.

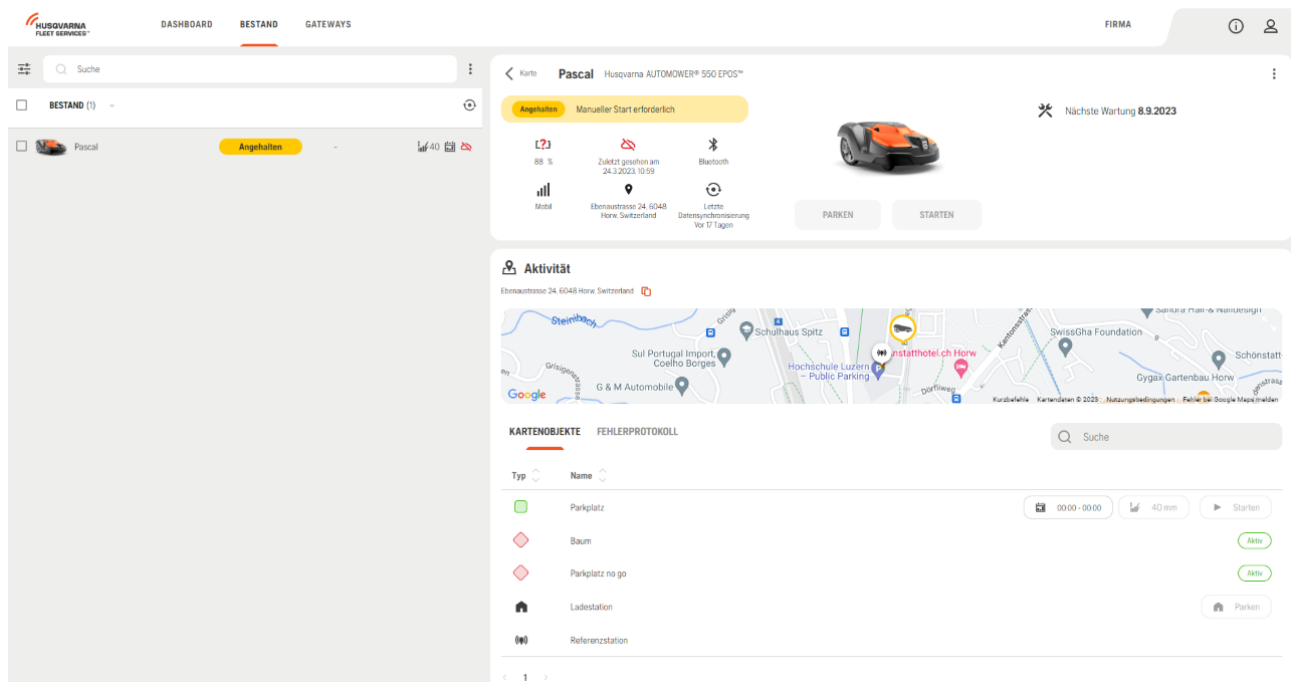
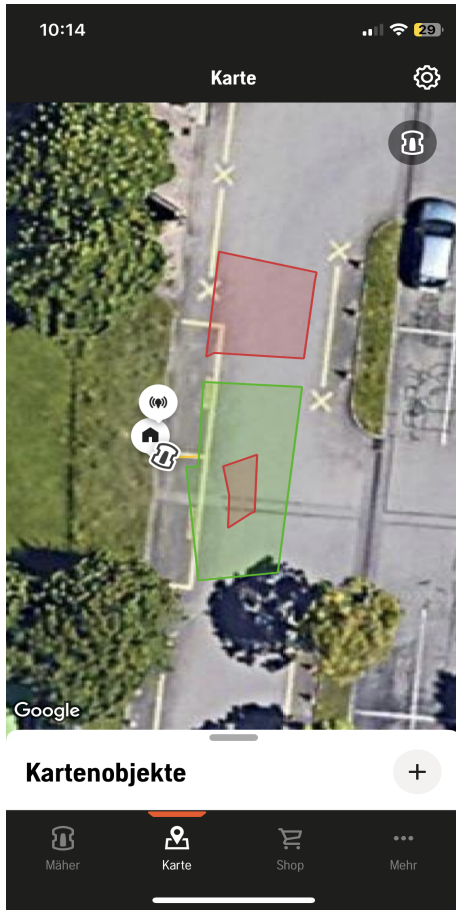


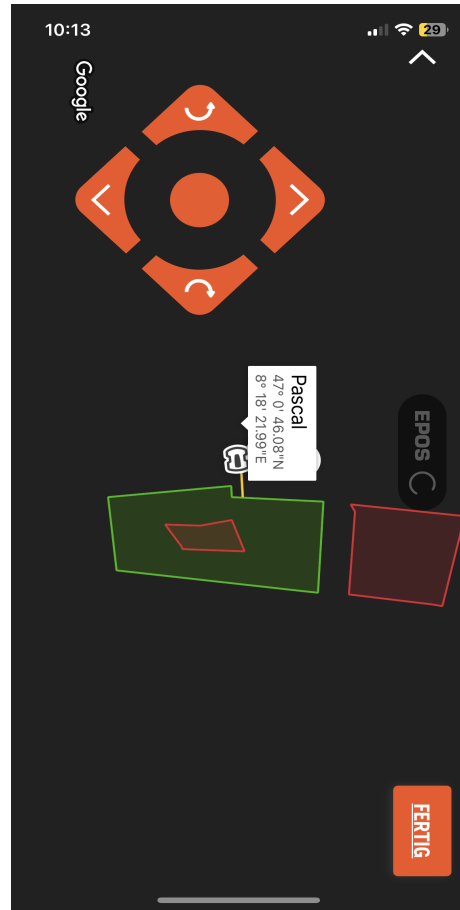
Figure 7: Husqvarna Fleet Services - Bestand

In der Registerkarte “Bestand” (Abbildung 7) sind lediglich die Steuerungsfunktionen “Parken” sowie “Starten” verfügbar. Es sind viele Überwachungsfunktionen möglich, wie beispielsweise der Standort, Batterieladezustand, Fehlerprotokoll usw. Auch das Erfassen eines neuen Gerätes ist hier möglich. Wie und wo die Daten gespeichert werden, ist aus der Beschreibung nicht ersichtlich. Doch im Reiter “Dashboard” kann die Flotte analysiert werden und es gibt Statistiken zur Gerätenutzung.

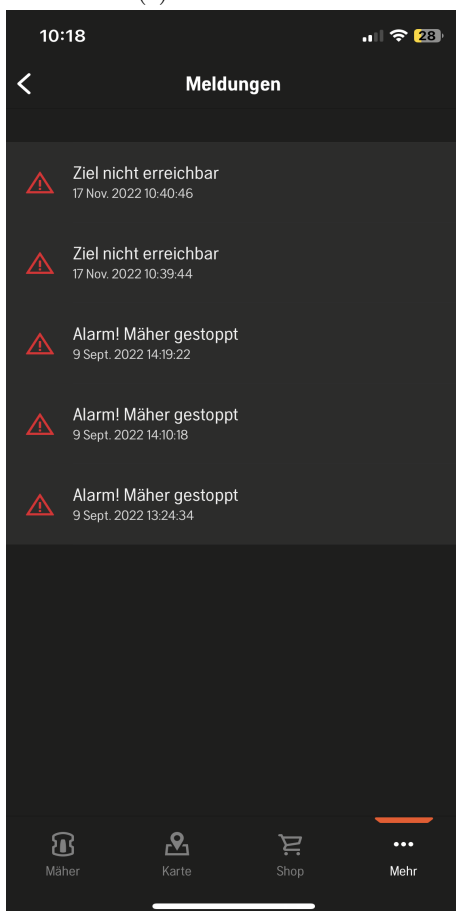
Die Automower App gezeigt in den Abbildungen 8, ist für die Steuerung vor Ort gedacht. Sie bietet verschiedene Funktionalitäten, wie das Festlegen des Mähbereichs, das Auslesen von Statistiken und Fehlern, sowie die manuelle Steuerung mittels einer Art Joystick.



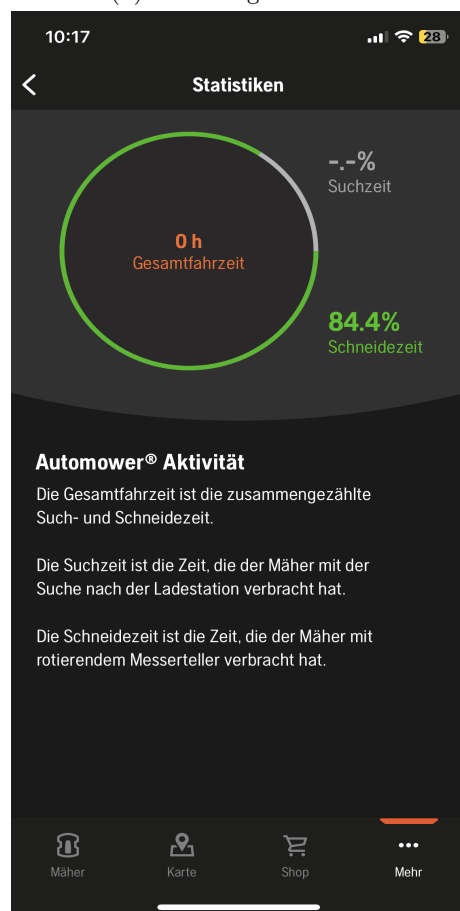
(a) Kartenansicht



(b) Steuerungselemente



(c) Übersicht Fehlermeldungen



(d) Statistiken

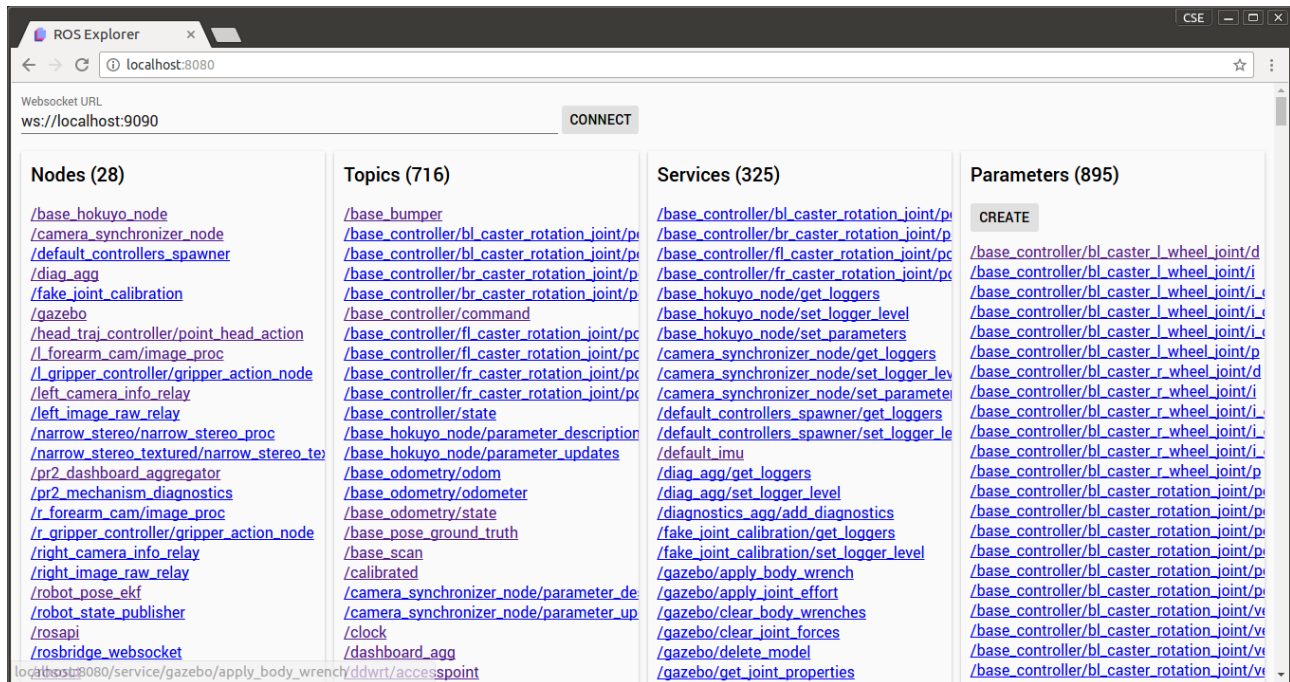
Figure 8: Husqvarna Automower App

### 2.3.2 WebUIs mit ROS

Da der S-bot hauptsächlich Daten über ROS zur Verfügung stellt, wurde zuerst ausgiebig in diesem Zusammenhang recherchiert. Es gibt bereits viele verschiedene Lösungen und Anwendungen für ein webbasiertes UI für Roboter. Was nachfolgend gezeigt wird, sind verschiedene Open-Source-repositories auf github.

#### 2.3.2.1 ROS Explorer

ROS Explorer ist ein webbasiertes Dienstprogramm zum Durchsuchen des ROS-Graphen. Es erlaubt alle nodes, topics, services und parameter anzusehen. Parameter können hier auch kreiert und verwaltet werden. Es basiert auf Node.js und verbindet sich auf den zur Verfügung gestellten Websocket.



Bildquelle: [https://wiki.ros.org/ros\\_explorer](https://wiki.ros.org/ros_explorer)

Figure 9: ROS Explorer

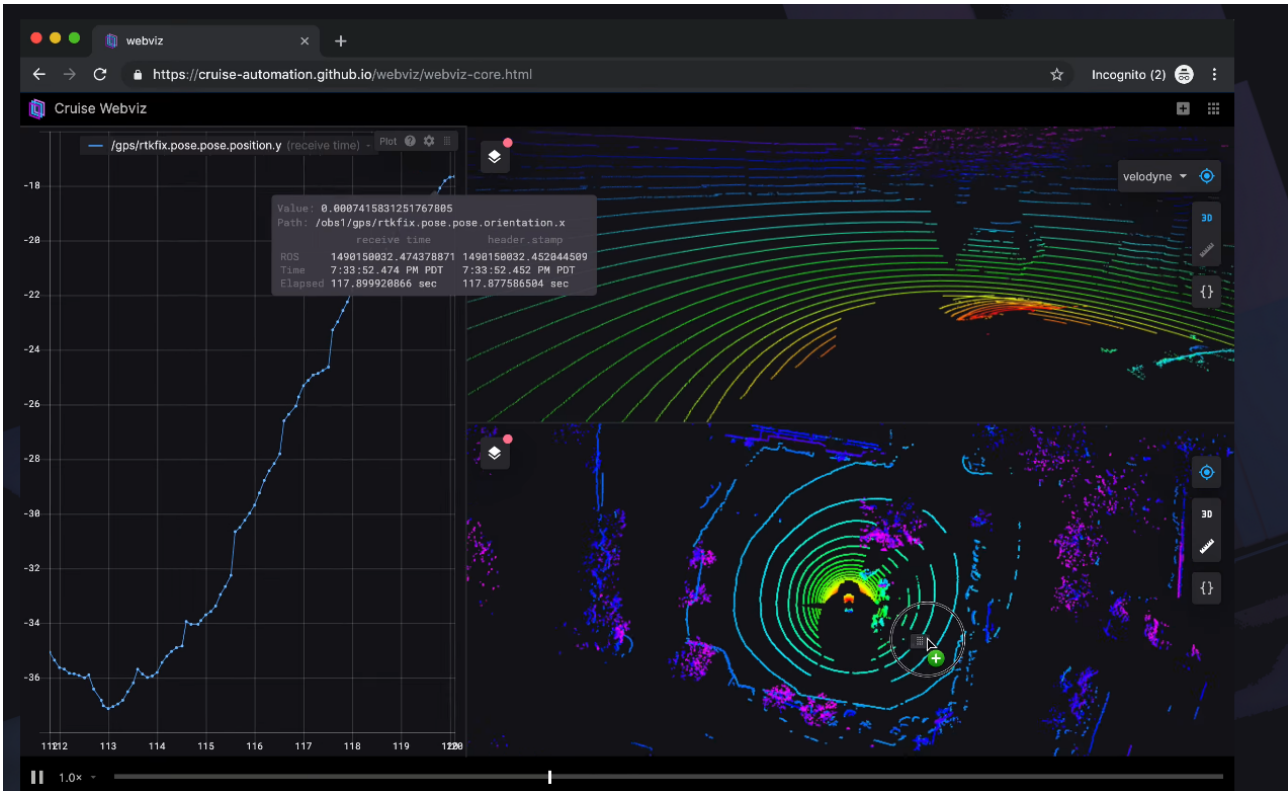
#### 2.3.2.2 WebViz & Foxglove Studio

Webviz ist eine webbasierte Anwendung zur Wiedergabe und Visualisierung von ROS-Bag-Dateien. Es ist auch möglich einen Roboter und die ROS-Daten live darzustellen. Das Problem bei Webviz ist, dass dieses repository nicht mehr stark gewartet wird und sehr viele Abhängigkeiten von Bibliotheken und Versionen bestehen.

Foxglove Studio ist ein integriertes Visualisierungs- und Diagnosewerkzeug für die Robotik, das im Browser oder als Desktop-App unter jedem OS verfügbar ist. Foxglove ist eine Evolution von Webviz. Auch bei Foxglove gibt es sehr viele Adaptionmöglichkeiten.

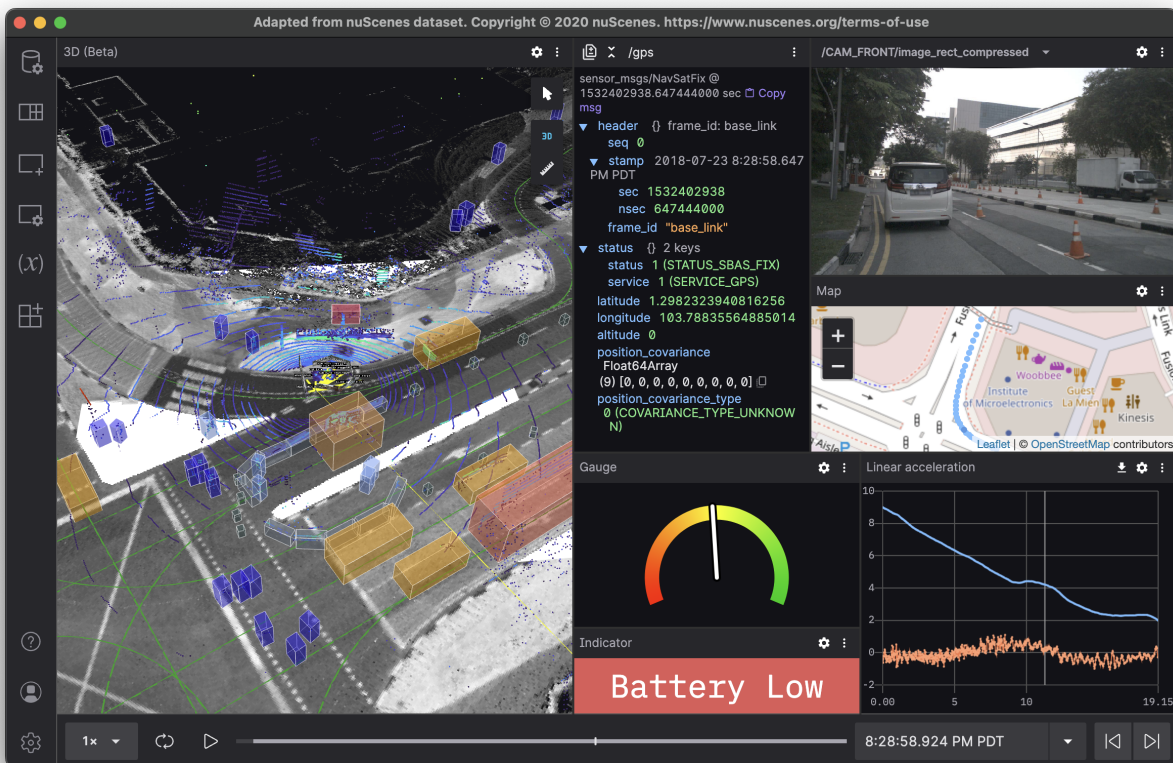
Beide Anwendungen sind auf der folgenden Seite für den Direktvergleich abgebildet.





Bildquelle: <https://webviz.io/>

Figure 10: Webviz

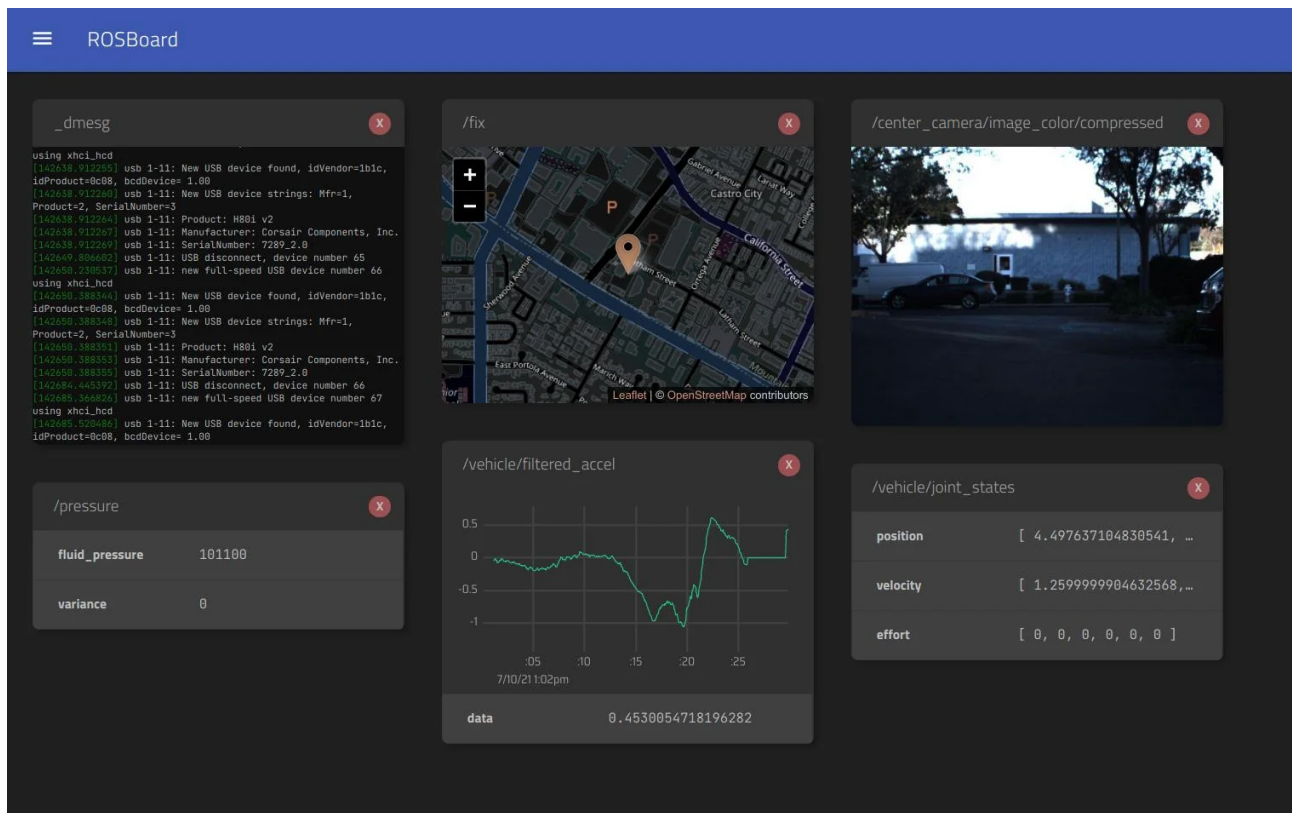


Bildquelle: <https://foxglove.dev/>

Figure 11: Foxglove Studio

### 2.3.2.3 ROSboard

ROSboard ist eine schlankere Adaption von Webviz. Das Dashboard verfügt über vielversprechende Überwachungsfeatures, doch das Problem ist, dass alle Daten über einen einzelnen ROS-node laufen müssen.

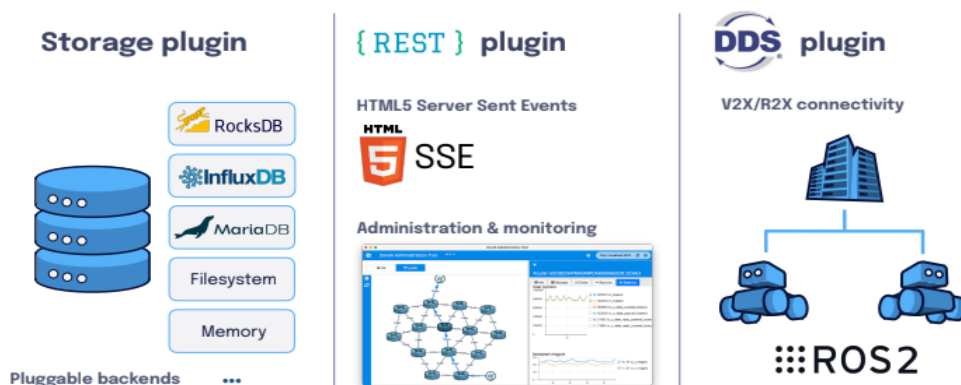


Bildquelle: <https://github.com/dheera/rosboard>

Figure 12: ROSboard

### 2.3.2.4 Zenoh for ROS2

Zenoh for ROS2 ist eine Middleware-Lösung, die speziell für das Roboterbetriebssystem ROS2 entwickelt wurde. Es verbindet traditionelle Pub/Sub-Protokolle mit geverteilter Speicherung, Abfragen und Berechnungen, das somit eine effiziente und zuverlässige Kommunikation zwischen Robotern und ihren Komponenten ermöglicht. Zenoh for ROS2 bietet eine skalierbare und flexible Architektur, die es ermöglicht, Daten in Echtzeit auszutauschen, unabhängig von der Größe des Roboternetzwerks oder der Anzahl der beteiligten Komponenten. Diese Dreiteilung der Architektur in Daten / Administration & Verwaltung / On-Premise-Steuerung ist in Abbildung 13 zu sehen. Zenoh for ROS2 zielt darauf ab, die Leistung, Skalierbarkeit und Robustheit von ROS2-Anwendungen zu verbessern und gleichzeitig die Entwicklungs- und Integrationszeit zu verkürzen [26].



Bildquelle: <https://zenoh.io/media/>

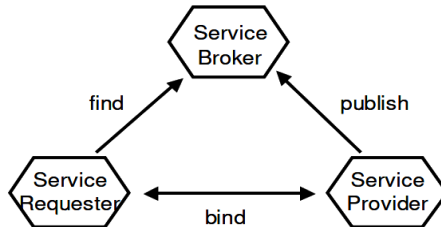
Figure 13: Zenoh für ROS2



### 2.3.3 Architektur

Die bis jetzt gezeigten Tools bieten viele Features zur Überwachung des Roboters. Funktionen zur Fernsteuerung sind jedoch keine vorhanden. Im Zuge der Recherche für die Entwicklung eines WebUIs und deren Architektur sind wichtige Begriffe aufgetaucht, welche nun erklärt werden.

“Service Oriented Architecture (SOA) ist ein neuer Ansatz, der die Anforderungen an lose gekoppelte, standardbasierte und protokollunabhängige verteilte Datenverarbeitung erfüllt” [27]. “Die SOA bietet einen allgemeinen Rahmen für die Interaktion mit Diensten. Sie beschreibt die drei Rollen Service Provider, Service Requester und Service Broker sowie die drei Operationen Publish, Find und Bind” [28].



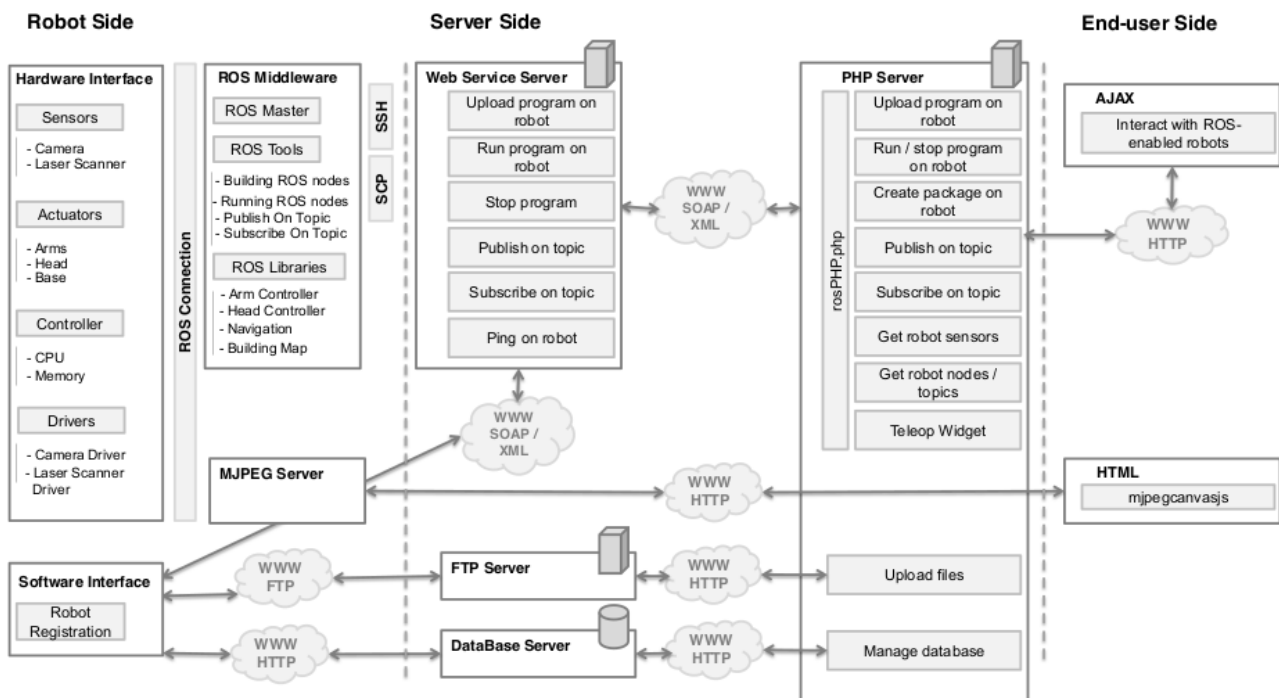
Bildquelle: An Operating Guideline Approach to the SOA [28]

Figure 14: Serviceorientierte Architektur SOA

SOA ist demnach ein Architekturstil, der für die Entwicklung von verteilten Anwendungen und Systemen eingesetzt wird. Dabei werden unabhängige Dienste lose gekoppelt organisiert und über standardisierte Schnittstellen wie Web Services zugänglich gemacht [29]. Web Services sind plattformunabhängig und werden von den meisten gängigen Programmiersprachen und Betriebssystemen unterstützt [30].

Bei der Bereitstellung von Diensten in SOA-basierten Lösungen in der Robotik werden sowohl Web Services verwendet, die mit dem SOAP / WSDL übereinstimmen, als auch solche, die als REST-Dienste beschrieben sind [31]. Dies ermöglicht eine flexible und skalierbare Integration von verschiedenen Diensten, da sie unabhängig voneinander entwickelt, bereitgestellt und verwendet werden können.

Anis Koubaa führte bereits 2014 [32] mit RoboWeb SOAP- und REST- Web Services in ROS ein. RoboWeb ist eine SOAP-basierte serviceorientierte Architektur, die Robotik-Hardware- und Softwareressourcen virtualisiert und sie als Dienste über das Internet zur Verfügung stellt.



Bildquelle: A Service-Oriented Architecture for Virtualizing Robots in Robot-as-a-Service Clouds [32]

Figure 15: RoboWeb System

Mit der ROSJAVA-Bibliothek konnten erstmals ROS-Codes wie klassische Web Services aufgerufen werden. Die heutige, standardmässige, Erweiterung ist roslibjs [33], die zentrale JavaScript-Bibliothek für die Interaktion mit ROS über den Browser. Sie nutzt WebSockets, um sich mit rosbridge zu verbinden und bietet Publishing, Subscribing, Service Calls, actionlib, TF, URDF-Parsing und andere wichtige ROS-Funktionen.

Bei der Verwendung von Webdiensten mit Robotern ist der Begriff Robot as a Service (RaaS) zu nennen. Das Konzept von RaaS beinhaltet den Entwurf und die Implementierung eines Roboters als All-in-One-SOA-Einheit. Diese Einheit umfasst Dienste zur Ausführung von Funktionen, Service-Broker zur Entdeckung und Veröffentlichung sowie Anwendungen für den direkten Zugriff durch Kunden. Dadurch wird der Fernzugriff auf Roboterdienste und deren Anwendungen ermöglicht, einschliesslich der Bearbeitung von Aufgaben. RaaS hat vollständige SOA-Funktionen, d.h. als Service Provider, als Service Broker und als Service Client [34]:

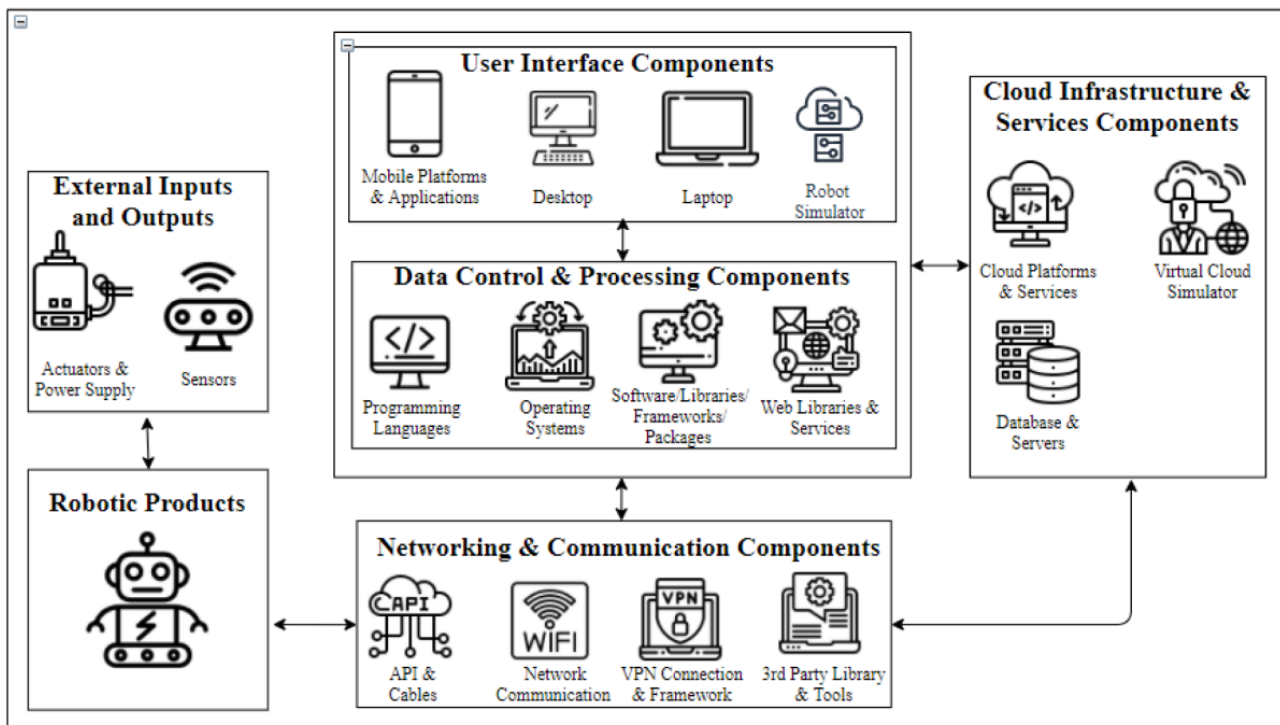
- Ein RaaS-Gerät ist ein Service Provider: Jede Einheit beherbergt ein Repository mit vorinstallierten Diensten.
- Eine RaaS-Cloud enthält eine Reihe von Applikationen, die bereitgestellt werden.
- Ein RaaS-Gerät ist ein Service-Broker: Ein Client kann die im Verzeichnis der Einheit verfügbaren Dienste und Anwendungen nachschlagen. Ein Client kann die auf dem Roboter bereitgestellten Anwendungen und Dienste suchen und entdecken, indem er das Verzeichnis durchsucht.

## 2.4 Cloud

Die Idee von RaaS besteht darin, Roboter mithilfe der SOA-Architektur als Cloud-Einheiten aufzubauen, sodass sie als Geräte registriert werden können und ihre Dienste sowie Anwendungen über die Cloud verfügbar sind.

### 2.4.1 Cloud Robotics

“Es gibt nur wenig veröffentlichte Literatur über eine generische Cloud-Robotik-Architektur, die Informatiker oder Softwareentwickler nutzen und an ein bestimmtes Anwendungsszenario anpassen können, da die meisten bestehenden Cloud-Robotik-Architekturen szenariospezifisch sind” [35, S.1]. Dennoch ist im selben Artikel eine generische Cloud-Robotik-Architektur 16 vorgeschlagen worden.



Bildquelle: Building and evaluating cloud robotic systems: A systematic review [35]

Figure 16: Generische Architektur für ein Cloud Robotics System

Cloud Robotics ist ein Ansatz, der die Vorteile von Cloud-Computing und Robotik kombiniert. Es bezieht sich auf die Integration von Robotersystemen mit Cloud-Infrastrukturen und -Diensten, um erweiterte Funktionen,

Skalierbarkeit und Flexibilität zu bieten.

Bei Cloud Robotics werden Roboter mit dem Internet und der Cloud verbunden, um auf Ressourcen und Daten in der Cloud zuzugreifen, die Rede ist von Internet of Robotic Things (IoRT). Dies ermöglicht es den Robotern, komplexe Aufgaben auszuführen, die eine hohe Rechenleistung, Speicher und externe Daten erfordern. Die Cloud dient als zentraler Hub, der die Verarbeitung, Analyse und Speicherung von Daten unterstützt und Funktionen wie maschinelles Lernen, Datenbanken und IoT-Dienste bereitstellt.

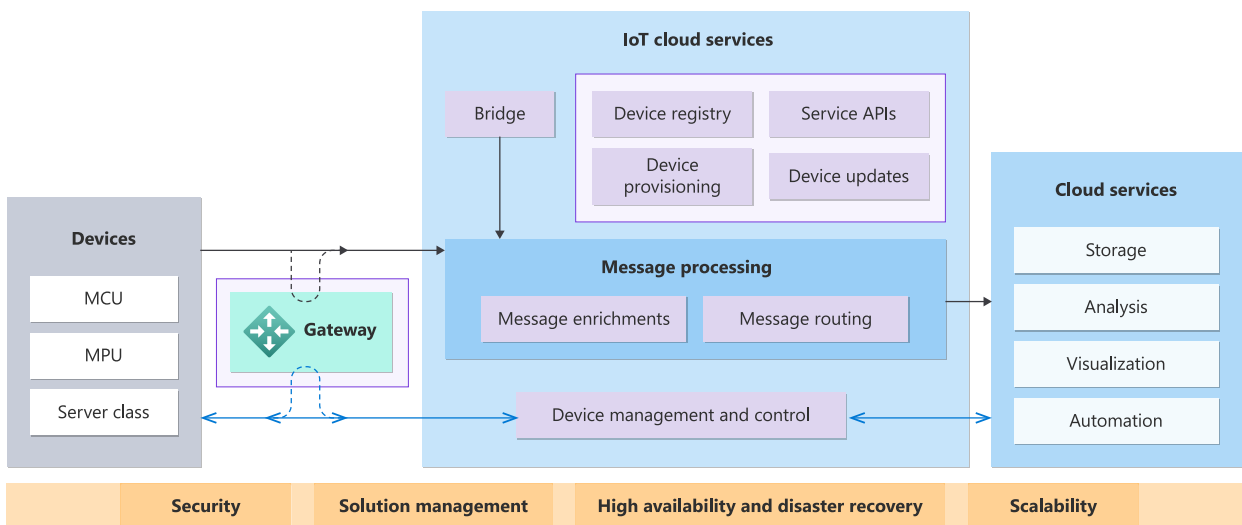
## 2.4.2 Cloud mit Azure IoT

Microsoft Azure Cloud bietet eine Vielzahl von Modellen und Dienstleistungen zur Ermöglichung eines allgemeinen, bequemen und bedarfsgerechten Netzwerkzugriffs auf einen gemeinsamen Pool konfigurierbarer Computerressourcen (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste), die mit geringfügigem Verwaltungsaufwand und Interaktion bereitgestellt und eingesetzt werden können. Im Folgenden werden nur die verwendeten und für die Arbeit relevanten Technologien und Services genauer erläutert.

Innerhalb der NIST-Definition von Cloud Computing gibt es drei Servicemodelle: Software as a Service (SaaS), Platform as a Service (PaaS) und Infrastructure as a Service (IaaS) [36]. Als Teil von Cloud Computing zählt Azure Internet of Things (IoT), welche eine umfangreiche Sammlung von Cloud-Diensten, Edge-Komponenten und SDKs bieten. Diese Tools ermöglichen es, grosse Mengen an IoT-Ressourcen miteinander zu verbinden, zu überwachen und zu steuern. Einfach ausgedrückt besteht eine IoT-Lösung aus intelligenten Geräten, die in der Lage sind, mit den Cloud-Services zu kommunizieren [37, S.29].

Eine Ressourcengruppe ist ein Container, der Cloud Computing Services für eine Azure-Lösung enthält. Die gebrauchten Ressourcen, welche den gleichen Lebenszyklus haben, werden dem verwendeten Abonnement zugewiesen [37, S.41].

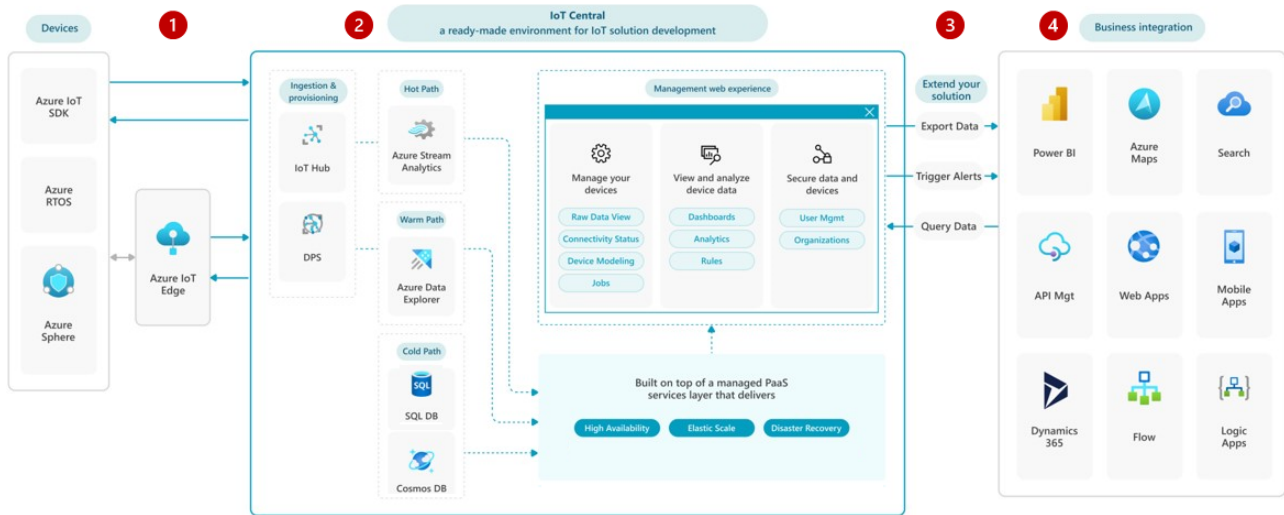
Die folgende Übersicht 17 zeigt die verschiedenen Komponenten, die in einer typischen IoT-Lösung enthalten sind.



Bildquelle: <https://learn.microsoft.com/en-us/azure/iot/iot-introduction>

Figure 17: Generelle Azure IoT Architektur

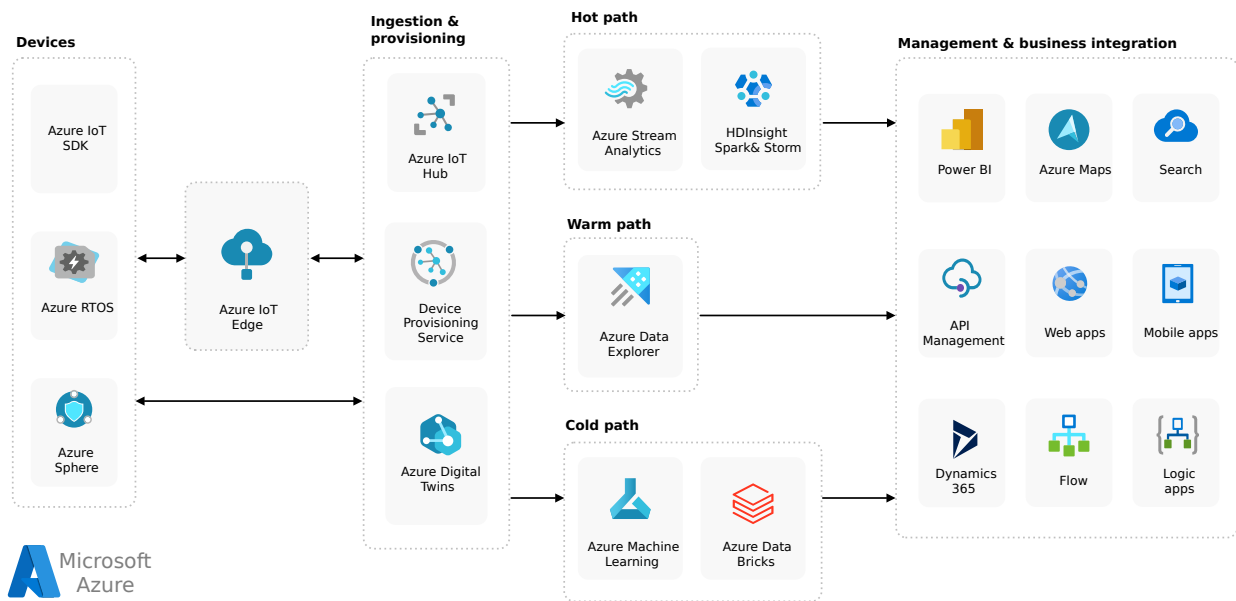
Zu Beginn muss eine Lösung für IoT gewählt werden. Dabei wird zwischen IoT Central und IoT Hub unterschieden. “Die IoT Central-Anwendung Platform-as-a-Service (aPaaS)[sic] bietet bereits die integrierten Azure-Komponenten und -Funktionen, die eine IoT-Lösung benötigt. Eine weitere Option besteht darin, Azure IoT Hub mit anderen Azure PaaS-Komponenten zu kombinieren, um eigene IoT-Lösungen zu entwickeln” [38]. IoT Central bietet ein standardisiertes web-UX und API-Oberfläche, Flottenüberwachung und Serviceverwaltungen einfach gestaltet. IoT-Central-Anwendungen verwenden mehrere IoT Hubs als Teil ihrer skalierbaren und robusten Infrastruktur.



Bildquelle: <https://learn.microsoft.com/de-de/azure/architecture/example-scenario/iot/iot-central-iot-hub-cheat-sheet>

Figure 18: Azure IoT Central-basierte Architektur

Wenn mehr Kontrolle und Anpassungsfähigkeiten gewünscht werden, so können einzelne Azure PaaS-Komponenten verwendet werden, um eine massgeschneiderte IoT-Lösung zu erstellen. Die vier Entitäten in der Abbildung 19 werden nachfolgend kurz erklärt.



Bildquelle: <https://learn.microsoft.com/de-de/azure/architecture/example-scenario/iot/iot-central-iot-hub-cheat-sheet>

Figure 19: Azure-Dienste in einer PaaS-basierten IoT-Architektur

1. Daten können mithilfe verschiedener Azure-Technologien erfasst werden, wie beispielsweise über die Azure IoT-Geräte-SDKs, Azure RTOS, Azure Sphere oder Azure IoT Edge.
2. Für die Bereitstellung, Konnektivität und Verwaltung von Geräten können IoT Hub, Device Provisioning Service (DPS) oder Azure Digital Twins genutzt werden.
3. Bezüglich Datenspeicherung und Analyse gibt es verschiedene Optionen:
  - Der "Hot Path" dient der Echtzeitverarbeitung und Anzeige von Daten. Dieser kann beispielsweise mittels Azure Stream Analytics oder Azure HDInsight realisiert werden.
  - Der "Warm Path" wird genutzt, um eine aktuelle Teilmenge der Daten zu speichern und anzuzeigen. Kleine Analyse- und Stapelverarbeitungsvorgänge werden mit diesen Daten durchgeführt. Dieser kann beispielsweise über den Azure Data Explorer abgewickelt werden.

- Der “Cold Path” dient der Langzeitspeicherung von Daten. Hier werden zeitaufwendige Analysen und Stapelverarbeitung durchgeführt. Dieser kann beispielsweise mit Azure SQL-Datenbank oder Azure Cosmos DB umgesetzt werden.

4. Verwaltungs- und Geschäftsintegrationsdienste umfassen verschiedene Tools und Plattformen wie Power BI, Azure Maps, Search, API Management, Web-Apps, Mobile Apps, Dynamics 365, Flow und Logic Apps. Diese Dienste ermöglichen die Verwaltung und Integration von Geschäftsprozessen und bieten vielfältige Funktionalitäten.

Diese und weitere Informationen, sowie ein tabellarischer Vergleich zwischen IoT Central und der IoT Hub-basierten PaaS-Lösung, sind in den Dokumentationen von Azure zu Anwendungsarchitekturen zu finden [38].

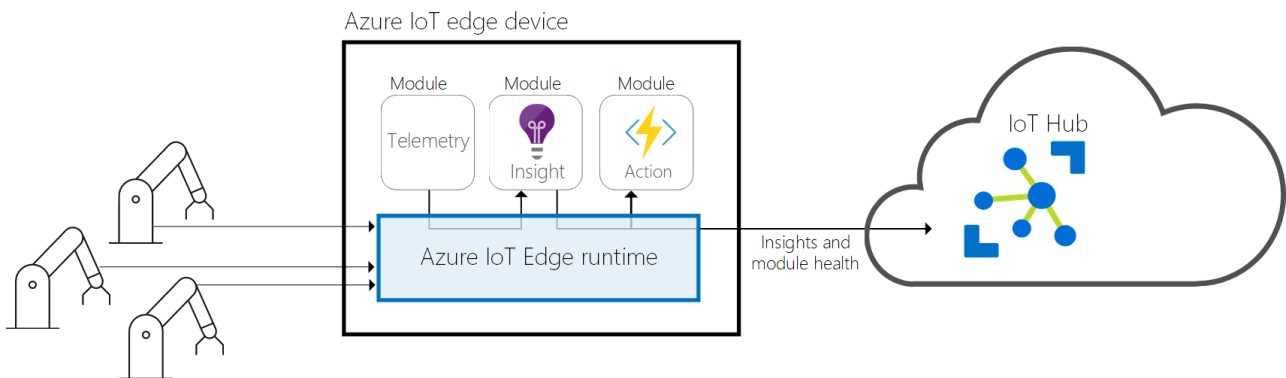
### 2.4.2.1 IoT Devices

Im Falle des RoVeKo-Projektes ist der S-bot das IoT-Gerät und der enthaltene Rechner von Intel nuc12 ist ein “Azure Certified Device” [39]. Die Microsoft Azure IoT-Geräte-SDKs bieten vorgefertigte Codes, die die Entwicklung von Anwendungen auf Geräten erleichtert, welche so eine nahtlose Verbindung zu den Azure IoT Hub-Diensten herstellen können. IoT Hub-Geräte-SDKs ermöglichen auch das Erstellen von Apps, die mithilfe des Geräte- oder Modulclients auf den IoT-Geräten ausgeführt werden. Diese Apps senden Telemetriedaten an den IoT-Hub und empfangen optional von dort Nachrichten, Aufträge, Methoden oder Updates für Gerätezwillinge. Die SDKs ermöglichen die Entwicklung von Geräte-Apps, die die Konventionen und Modelle von Azure IoT Plug & Play nutzen. Mit dem Modulclient können auch Module für die Azure IoT Edge-Runtime erstellt werden.

### 2.4.2.2 IoT Edge

“IoT-Edge-Module sind Docker-Container mit benutzerdefiniertem Code, die auf Laptops und Servern sowie auf eingeschränkten Windows- und Linux-Geräten laufen. Der intelligente Edge ermöglicht die Erstellung und Bereitstellung von Modulen für KI, die Azure IoT Edge-Runtime und eine enge Integration mit der Azure-Cloud über IoT Hub” [40, S.360].

“Die IoT Edge-Runtime wird auf jedem IoT Edge-Gerät ausgeführt und dient zum Verwalten der Module, die auf einem Gerät jeweils bereitgestellt wurden” [41]. Dadurch wird die Ausführung von Cloud-Workloads auf IoT-Geräten ermöglicht. Diese Auslagerung bietet weitere Vorteile wie Echtzeit-Entscheidungen, Datenfiltrierung und geringere Latenzzeiten. Die Edge-Laufzeitumgebung bietet zudem Funktionen wie Offline-Betrieb, Sicherheit, Skalierbarkeit und Management von Edge-Geräten in einem verteilten Umfeld.



Bildquelle: <https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge?view=iotedge-1.4>

Figure 20: Azure IoT Edge-Runtime

### 2.4.2.3 IoT Hub

Azure IoT Hub ist ein verwalteter Dienst, der in der Cloud gehostet wird und als zentraler Nachrichten-Hub für die Kommunikation zwischen einer IoT-Anwendung und deren angeschlossenen Geräten fungiert. IoT Hub bietet einen Cloud-Speicherplatz für Telemetriedaten, die von IoT-Geräten erfasst werden. Über den IoT-Hub ist es möglich, die Erstellung von Geräten, Geräteverbindungen und Geräteausfälle zu verfolgen [42, S.32].

#### **2.4.2.4 Device Provisioning Service**

“Der IoT Hub Device Provisioning Service wird für die Bereitstellung von IoT-Geräten verwendet, ohne dass hardcodierte IoT-Verbindungsinformationen im Gerät vorhanden sind. Während der Herstellung des Geräts wird das Gerät so programmiert, dass es den Provisioning Service aufruft, wenn es eingeschaltet wird, damit es Verbindungsinformationen und seine IoT-Lösungszuweisung erhalten kann” [42, S.83].

#### **2.4.2.5 Blob Storage**

“Azure Blob Storage ist die Objektspeicherlösung von Microsoft für die Cloud. Sie ist für die Speicherung grosser Mengen unstrukturierter Daten wie Bilder, Videos, Protokolldateien, Backups und Streaming-Inhalte optimiert. Azure Blob Storage unterstützt drei Ebenen: Hot, Cold und Archive” [43, S.30]. Bei einem Binary Large Object (BLOB) handelt es sich um eine grosse Datei, typischerweise ein Bild oder eine Form von unstrukturierten Daten. “Azure Blob-Speicher ist unendlich skalierbar und praktisch unbegrenzt. Azure Blob-Speicher ist die kostengünstigste Art von Speicher in Azure” [44, S.273].

#### **2.4.2.6 Azure Data Explorer**

“Azure Data Explorer ist ein schneller, vollständig verwalteter und hoch skalierbarer Big-Data-Analysedienst. Azure Data Explorer kann grosse Mengen von Streamingdaten aus Anwendungen, Websites und IoT-Geräten in Echtzeit analysieren, um Analyseanwendungen und Dashboards zu bedienen” [45]. Mittels dem Open-Source Projekt auf <https://github.com/Azure/azure-iot-explorer> ist eine Applikation zur Verfügung gestellt, welche die direkte Verbindung zwischen dem IoT-Gerät und dem IoT Hub überwachen lässt.

#### **2.4.2.7 Azure Message Routing**

Über das Portal können benutzerdefinierte Endpunkte genutzt werden, um Nachrichten von Geräten an die Azure-Cloud über Message Routing zu einem Blob Storage weiterzuleiten. Mit einer maximal unterstützten Nachrichtengrösse von 256 KB werden die Nachrichten nahezu in Echtzeit in der Reihenfolge empfangen, in der sie gesendet wurden. Es besteht die Möglichkeit, bis zu 10 benutzerdefinierte Endpunkte und 100 Routen pro IoT Hub zu erstellen [42, S.79-80]. Durch die Konfiguration im “Query” können spezifische Anwendungs- und Systemeigenschaften oder Nachrichtentexte abgefragt werden.

### 3 Methodik

In diesem Kapitel geht es generell um das Vorgehen und die Methodik für die Konzipierung von Software- und Cloud-Architekturen, sowie für die Erarbeitung von Grundgerüsten und Entwicklung von Features.

Die gesamte Arbeit wurde auf der agilen Projektmanagementmethode aufgebaut, jedoch ohne explizite Rollenverteilung. Für die Anforderungserhebung des WebUI wurde daher ein eigenständiges Projekt in Jira erstellt, das auf dem Kanban-Prinzip basiert.

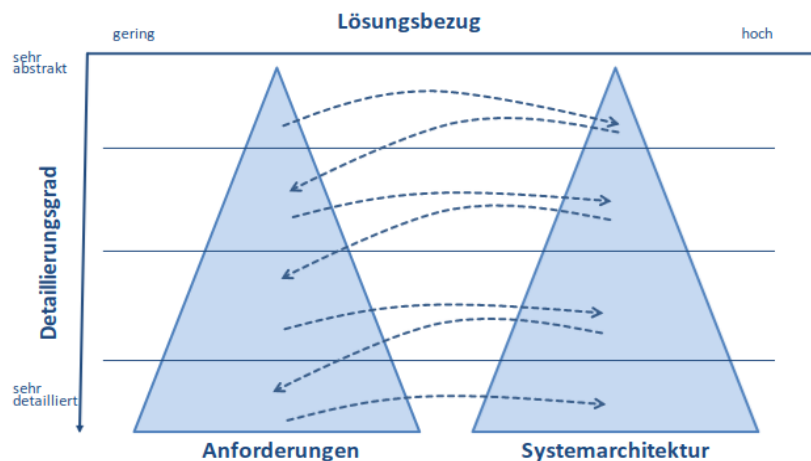
Ein Gantt-Diagramm diente zur Planung und Überwachung des Projektstatus, dieser ist im Anhang 5 zu finden.

#### 3.1 Vorgehen zur Anforderungserhebung

Die verschiedenen Anspruchsgruppen haben unterschiedliche Vorstellungen, wie ein WebUI aussehen könnte und vor allem, welche Funktionen es bietet.

Für die Systementwicklung ist ein gutes Anforderungsmanagement nötig. Auch gilt zu sagen, dass dies in vielen Iterationsschritten geschehen muss. Die erste erfolgte Erhebung hatte das Ziel, die grundlegenden Funktionen zu ermitteln und so einen iterativen Entwicklungsprozess in Gang zu setzen.

Das Twin-Peaks-Model [46], wie in Abbildung 21 dargestellt, veranschaulicht den Ansatz, bei dem eine Anforderungsbeschreibung und eine Systemarchitektur iterativ und parallel entwickelt werden, wobei beide Aspekte sich gegenseitig ergänzen und immer detaillierter werden.



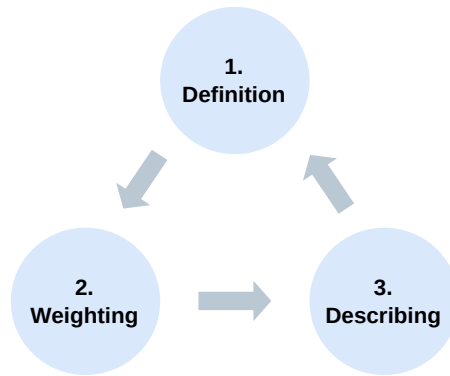
Bildquelle: Handbuch Requirements Management [47]

Figure 21: Twin-Peaks-Model

In der ersten Phase "Anforderungserfassung und Konzeptualisierung" aus Abbildung 3, in der sich das Projekt befindet, wurde bewusst darauf verzichtet, sich sofort in eine detaillierte Beschreibung von Lösungen und Implementierungen zu stürzen. Stattdessen konzentrierte man sich zunächst darauf, die funktionalen Anforderungen zu erfassen.

Ein bewährtes Vorgehen zur Erhebung von Anforderungen ist die dreiphasige Methode nach ©iSREM [48, S.161]. Bei der Erhebungsmethode geht es um Anforderungen an Geschäftsprozesse, doch das Vorgehen für Anforderungen an ein WebUI ist analog.





Bildquelle: Structured Requirements Elicitation [48, S.161]

Figure 22: Anforderungserhebung nach ©iSREM

Eine erste Iteration dieses Vorgehens ist im Rahmen dieser Arbeit durchgeführt worden. Ein Vorschlag zum weiteren Vorgehen und zur nächsten Iteration wird in Kapitel 5.2 gemacht.

Bevor mit der Definition begonnen wurde, wurden die Stakeholder definiert, welche verschiedene Ansprüche und Vorstellungen an ein WebUI haben. Durch die Stakeholderanalyse sind die Anspruchsgruppen nach Interesse und Einfluss eingeordnet worden.

### 3.1.1 Definition

Aus den grundlegenden Funktionsanforderungen, zusätzlichen eigenen Überlegungen und einigen kurzen Besprechungen mit Vertretern der Anspruchsgruppe der Entwickler wurden zahlreiche gewünschte Features, einschliesslich Visualisierungen und Befehlsfunktionen für das WebUI, identifiziert. In diesem Status kann noch keine klar priorisierte Anforderungsliste erstellt werden, mehrfache Iterationen sind dazu nötig.

### 3.1.2 Gewichtung

Die Gewichtung ist wichtig, um die begrenzten Projektressourcen von Anfang an den projektkritischen Anforderungen zuzuordnen. Um eine Bewertung der Funktionen durch die Stakeholder zu ermöglichen, wurde ein Fragebogen erstellt. Darin gab es die Möglichkeit, weitere Funktionen und Feature-Vorschläge anzugeben. Zusätzlich wurde eine kurze Übersicht zum geplanten System präsentiert. Der Fragebogen wurde von verschiedenen Vertretern der Anspruchsgruppen ausgefüllt.

Die Fragebögen wurden anhand der Durchschnittsbewertung und der Standardabweichung ausgewertet. Features mit einer höheren Durchschnittsbewertung werden priorisiert und sollen zuerst entwickelt werden. Eine hohe Standardabweichung deutet auf Uneinigkeit bezüglich des Nutzens oder der Priorität eines Features hin.

### 3.1.3 Beschreibung

Die weiteren Vorschläge für Features wurden gesammelt und konsolidiert. Eine Liste mit allen möglichen Features ist erstellt worden. Die Übertragung auf Jira ist der letzte getätigte Schritt. Dabei galt es die Features zu gruppieren, also in verschiedene Epics zu ordnen und die jeweiligen Stories zu beschreiben.

## 3.2 Erarbeitung WebUI

Um die grundlegenden Funktionsanforderungen erfüllen zu können, war es nötig, gleichzeitig mit der gründlichen Literaturrecherche bezüglich ROS und WebUIs, sich verschiedene Skills und Know-How anzueignen. Dies wurde mit diversen Tutorials und Kursen erreicht:

1. Core ROS Tutorials: Beginner & Intermediate Level [49]
2. Linux for Robotics [50]
3. Developing Web Interfaces for ROS [51]

Durch die Übungen in den Kursen konnte mit der Entwicklung eines Grundgerüsts für das WebUI begonnen werden. Da ROS praktisch auf jeder Linux-Distribution ausführbar ist und mittels roslibs von einem Browser eine Verbindung dazu hergestellt werden kann, war eine zügige Entwicklung möglich.



### 3.3 Konzept der Cloud-Architektur

Für die Erarbeitung der Architektur-Konzepte musste einiges an Recherche betrieben werden, siehe 2.4. Auch hier sind einige Tutorials durchgeführt worden:

1. Erstellen eines IoT Hubs mit der Azure-Befehlszeilenschnittstelle
2. Verbinden eines Gerätes mit IoT Hub
3. Installieren und Verwenden des Azure-IoT-Explorers
4. Schnellstart: Senden von Telemetriedaten von einem IoT Plug & Play-Gerät an Azure IoT Hub
5. Senden von Gerätedaten an Azure Storage über IoT Hub-Nachrichtenrouting

Die Beschreibung der verwendeten Technologien und daraus abgeleiteten Entwürfe sind in den Kapiteln 2.2, respektive 4.3 zu finden.

## 4 Anforderungserhebung, Konzeption und Entwicklung WebUI & Azure Cloud

Das vorliegende Kapitel präsentiert die empirischen Ergebnisse der Arbeit, wobei der Fokus auf praxisorientierten Erkenntnissen liegt. Im Zentrum stehen die erste Iteration der Anforderungserhebung, das Grundgerüst und die Architektur des WebUI sowie das Konzept und der Nachweis der unidirektionalen Verbindung zur Cloud mit Azure.

### 4.1 Anforderungserhebung

Verschiedene Stakeholder wurden zu Beginn definiert und nach deren Einfluss und Interesse eingeteilt. Vertreter der Key Player, sowie einige EntwicklungsingenieurInnen sind bezüglich der Anforderungen befragt worden.

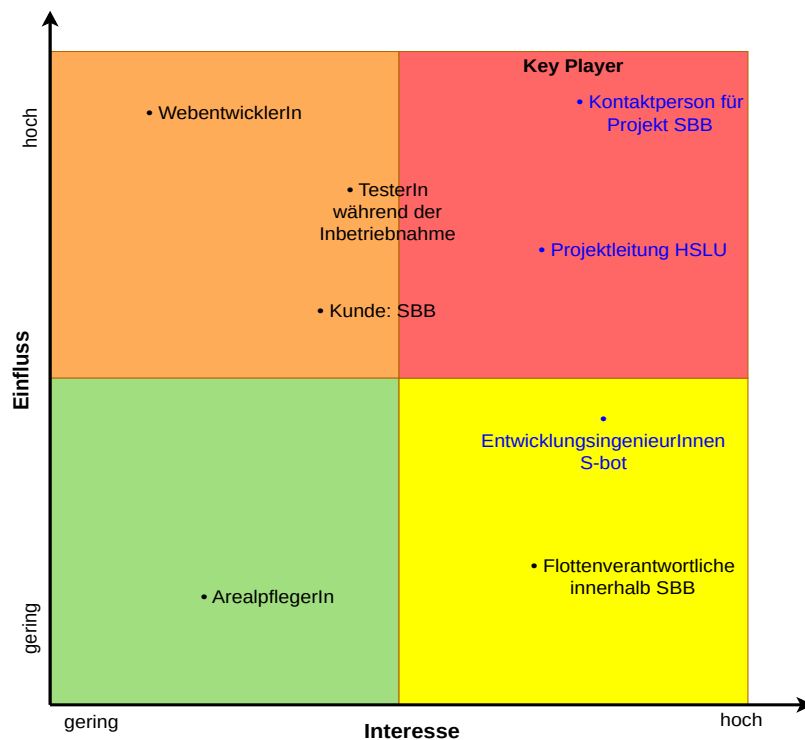


Figure 23: Stakeholderanalyse - blau markiert die befragten VertreterInnen

#### 4.1.1 Erste Anforderungserhebung

Die ersten definierten Anforderungen wurden in Visualisierungen, Befehlsausführung und weitere Features unterteilt. Der dazu erstellte Fragebogen ist im Anhang 5 zu finden.

##### Visualisierungen

1. Anzeige des Batterieladezustand
2. Aktueller Standort mit Anzeige auf der aktuellen Karte
3. Zuletzt abgefahrne Punkte auf der Karte
4. Asynchrone Bilder der integrierten Kamera
5. Visualisierung von Hindernissen und Informationen dazu (durch Sensordaten von Lidar und Kamera) Künstlicher Horizont, Distanzinformationen zu Objekten / zu HOME Wegpunkt
6. Aktueller Zustand: Mähen, Epilieren, Idle, Fahren, ...

## Befehlsausführungen

1. Senden von sofortigem Notstop (hier wäre eine genauere Analyse nachfolgend nötig) / Reset / Reboot / Pause
2. Senden von Karten der SBB / Aktualisieren der Kartendaten
3. Senden von einem Ziel (Punkt auf der Karte) / Abfahren eines Fahrplans
4. Senden von Kommandos wie Parken / Laden / Mähen / ...

## Weitere Features

1. Anbindung des S-bots an die Cloud zur Analyse von Logdaten auf dem Cloud-Anbieter
2. Vollständiges Hosting des WebUIs in der Cloud

### **4.1.2 Gewichtung der Features**

Im Fragebogen bewerteten Vertreter der Stakeholder die Features auf einer Skala von 1 bis 10. Da zeitgleich bereits an der Entwicklung einer globalen System-Architektur gearbeitet wurde, war es hilfreich, eine Übersicht zu erstellen, welche laufend angepasst wurde. In einer finalisierten Version ist diese in 25 abgebildet.

Durch den iterativen Prozess wurden durch die Stakeholder zusätzliche relevante Features identifiziert. Einige dieser Features wurden sogleich erfasst und dann von anderen Stakeholder auch gleich bewertet. Dadurch sind einige Punkte nicht von allen Stakeholder bewertet worden:

- Anzeige einer Statusbar: aktuell ...% gemäht, nächster Vorgang: Laden
- Anzeige des aktuellen Energieverbrauch: Modus des Roboters / Schneider von 0 bis 100%
- Warnlicht / Blitzlicht oder Warnton anschalten
- Anzeige von Temperaturen: Aussen-, Motor-, Motherboardtemperatur
- Möglichkeit zur Wiedergabe von Ton durch den S-bot gesteuert über das WebUI
- Kommunikation über Headset mit Sprachausgabe beim Roboter, um Anweisungen an Personal vor Ort zu geben

Auf der folgenden Seite sind die Anforderungen und deren Bewertungen gelistet.

ID	Feature Beschreibung	Kunde / Industriepartner	CTO Projekt RoVeKo	Projektleitung RoVeKo	Entwickler 1	Benutzer	Entwickler 2	Durchschnitt	Standard-abweichung
1	Aktueller Standort mit Anzeige auf der aktuellen Karte	10	9	10	8	9	10	9,333333333	0,74535592
2	Kommunikation über Headset mit Sprachausgabe beim Roboter, um Anweisungen an Personal vor Ort zu geben	2	2	4		2		2,5	0,866025404
3	Senden von Kommandos wie Parken / Laden	10	8	10	8	10	8	9	1
4	Senden von sofortigem Notstop (genauere Analyse nötig) / Reset / Reboot / Pause	10	8	10	7	10	10	9,166666667	1,213351648
5	Anzeige des Batterieladestand	10	7	10	8	8	10	8,833333333	1,213351648
6	Anzeige von Temperaturen: Aussen-, Motor-, Motherboard-Temperatur	3	4	6		6		4,75	1,299038106
7	Visualisierung von Hindernissen und Informationen dazu (durch Sensordaten von Lidar und Kamera) Künstlicher Horizont, Distanzinformationen zu Objekten / zu HOME Wegpunkt	8	6	10	5	10	7	7,666666667	1,885618083
8	Anzeige des aktuellen Energieverbrauch: Modus des Roboters / Schneider von 0 bis 100%	6	5	10		6		6,75	1,920286437
9	Anzeige einer Statusbar: aktuell ...% gemäht, nächster Vorgang: Laden	9	7	4		10	7	7,4	2,059126028
10	Zuletzt abgefahrene Punkte auf der Karte	9	4	7	7	10	5	7	2,081665999
11	Möglichkeit zur Wiedergabe von Ton durch den S-bot gesteuert über das WebUI	2	3			7		4	2,160246899
12	Asynchrone Bilder der integrierten Kamera	8	6	9	5	9	3	6,666666667	2,211083194
13	Warnlicht /Blitzlicht oder Warnton anschalten	2	6	4		8		5	2,236067977
14	Aktueller Zustand: Mähen, Epilieren, Idle, Fahren, ...	10	5	5	6	10	3	6,5	2,62995564
15	Anbindung des S-bots an die Cloud zur Analyse von Logdaten auf dem Cloud-Anbieter	8	8	3	3	10	7	6,5	2,62995564
16	Senden von Karten der SBB / Aktualisieren der Kartendaten	10	2	5	8	3	5	5,5	2,753785274
17	Senden von einem Ziel (Punkt auf der Karte) / Abfahren eines Fahrplans	10	9	10	8	9	2	8	2,768874621
18	Vollständiges Hosting des WebUIs in der Cloud	5	1	3	3	10	4	4,333333333	2,808716591

### 4.1.3 Konsolidierung der Features

Weitere Vorschläge für Funktionen sind erfasst und nach dieser ersten Iteration konsolidiert worden. Um eine übersichtliche, digitale Version zu erstellen, wurden sie in das neue Jira-Projekt “WebUI” übertragen. Dafür wurden Stories erstellt und den übergeordneten Epics zugeordnet. Die Aufgaben in der Roadmap, die bereit für die Entwicklung sind, können nun von dort auf das Kanban Board übertragen werden, um sie zu planen. Die Abbildung 24 zeigt die Aufgaben in der Roadmap und erleichtert so den Überblick und die Weiterverarbeitung. Um eine verbesserte Übersicht zu erreichen und die Anforderungen in kleinere Aufgaben zu gliedern, wurden einige der Stories in sogenannte “Child Issues” unterteilt. Dadurch wird sowohl die Übersichtlichkeit erhöht als auch die Möglichkeit geschaffen, die Anforderungen in kleinere Aufgaben zu zerlegen.

Sprints		
▼	<b>WEBUI-29 Navigations- und Karteninformationen</b>	
	WEBUI-22 Aktueller Standort mit Anzeige auf der aktuellen Karte	IN PROGR...
	WEBUI-25 Zuletzt abgefahrte Punkte auf der Karte	TO DO
	WEBUI-66 Status der Navigation	TO DO
	WEBUI-23 Anzeige der Bearbeitungsflächen	TO DO
	WEBUI-26 Ort der Ladestationen inkl. Auswahlmöglichkeit der Ladestation.	TO DO
▼	<b>WEBUI-31 Befehle und Steuerungen aus dem WebUI</b>	
	WEBUI-33 Analyse inwiefern ein sofortiger Notstopp möglich ist	TO DO
	WEBUI-32 Senden von Reset / Reboot / Pause	TO DO
	WEBUI-45 Zeitfenster festlegen	TO DO
	WEBUI-54 Kamera-Snapshot auslösen	TO DO
	WEBUI-55 Pull Logs	TO DO
	WEBUI-14 Warnlicht/Blitzlicht oder Warnton anschalten	TO DO
	WEBUI-17 Kommunikation über Headset mit Sprachausgabe beim Roboter	TO DO
▼	<b>WEBUI-46 Anzeige von Sensor-Informationen</b>	
	WEBUI-5 Anzeige des Batterieladestatus	TO DO
	WEBUI-59 Künstlicher Horizont	TO DO
	WEBUI-7 Anzeige einer Zustands-/Statusbar	TO DO
	WEBUI-9 Anzeige des aktuellen Energieverbrauchs: Modus des Roboters / Schneider von 0 bis 100%	TO DO
	WEBUI-10 Asynchrone Bilder der integrierten Kamera	TO DO
	WEBUI-51 IMU-Daten	TO DO
	WEBUI-48 Temperaturen	TO DO
	WEBUI-71 Informationen zu Vegetation	TO DO
▼	<b>WEBUI-6 Befehle auf der aktuellen Karte</b>	
	WEBUI-41 Senden von (Einzel-)Kommandos	TO DO
	WEBUI-36 Fahrplan festlegen	TO DO
	WEBUI-13 Aktualisieren der Kartendaten	TO DO
	WEBUI-34 Zonen festlegen	TO DO
▼	<b>WEBUI-16 Cloud Analytics</b>	
	WEBUI-12 Anbindung an Cloud	TO DO
	WEBUI-30 Hosting von Frontend und Backend in der Cloud - Link zum WebUI	TO DO
	WEBUI-19 Anzeige von Identifikationsnummer / Bezeichnung des Gerätes	TO DO
	WEBUI-79 Flottenmanagement - Registrierung Neugerät	TO DO
▼	<b>WEBUI-67 Fehleranzeige</b>	
	WEBUI-78 Navigations - Fehler	TO DO
	WEBUI-68 Sensor - Daten Fehler	TO DO
	WEBUI-70 Aktoren - Fehler	TO DO
	WEBUI-69 Cloud - Logging Fehler	TO DO

Figure 24: Roadmap Entwicklung WebUI

## 4.2 Konzept und Architektur des WebUI-Grundgerüsts

Im Laufe der Entwicklung eines WebUIs für den S-bot ist ausführlich an einer globalen Architektur gearbeitet worden. Durch die Recherche und das Testen der verschiedenen Interfaces aus Kapitel 2.3 ist ein Konzept für die Fernüberwachung des S-bots erstellt worden.

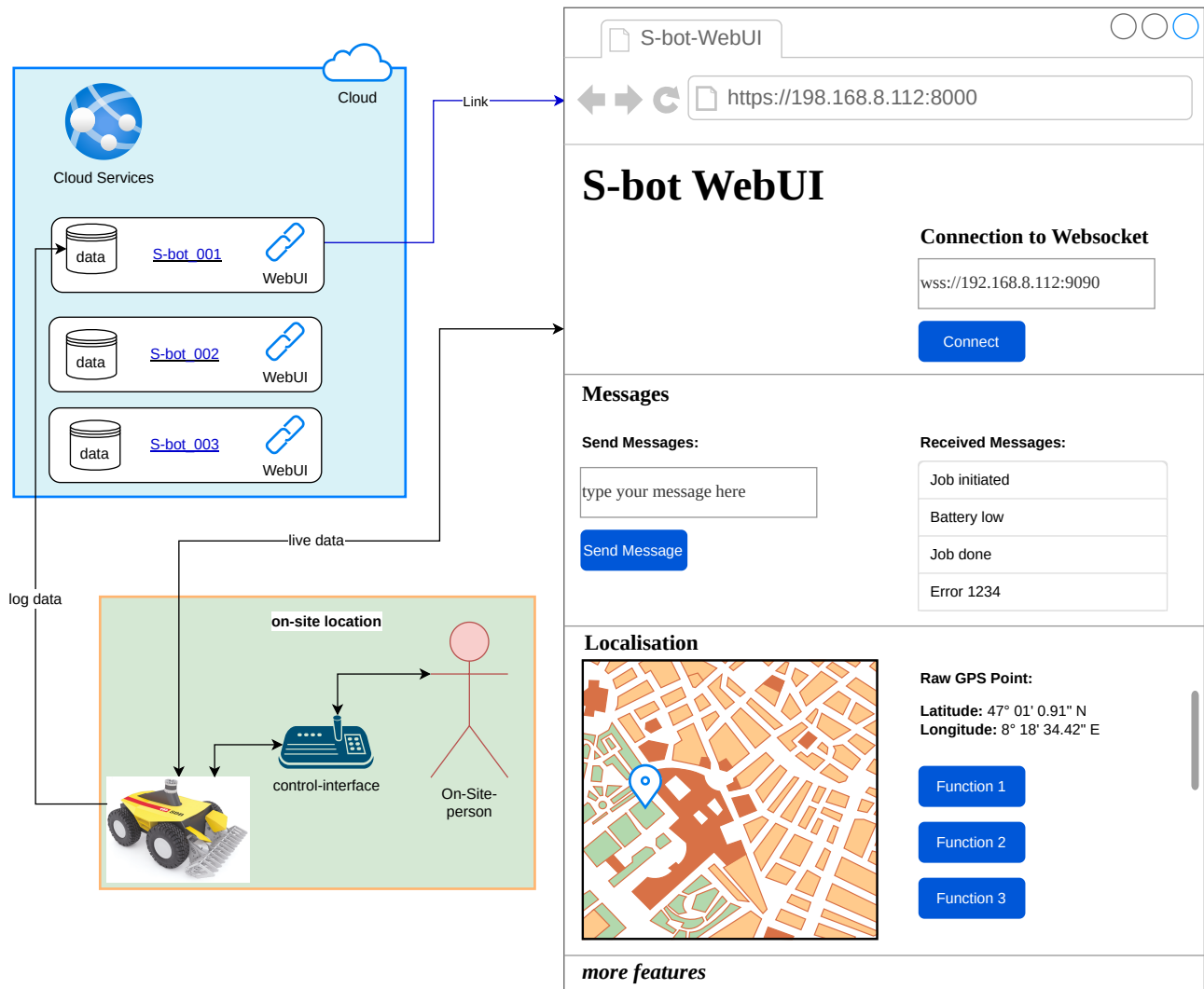


Figure 25: globale Architektur

Dabei ist eine Dreiteilung folgendermassen sinnvoll:

### 1. WebUI:

- Echtzeitüberwachung: Benutzer können den aktuellen Zustand des Roboters, wie Position, Geschwindigkeit oder Batteriestand, überwachen.
- Fehlerdiagnose: Das WebUI bietet die Möglichkeit, Fehler und Probleme am Roboter zu identifizieren und zu diagnostizieren.
- Datenvisualisierung und Analyse: Das WebUI ermöglicht die Visualisierung von Sensordaten und anderen relevanten Informationen des Roboters.
- Fernsteuerung: Benutzer können Befehle senden, um den Roboter zu bewegen, Aktionen auszuführen oder verschiedene Funktionen zu aktivieren.

## 2. Cloud

- **Datenvisualisierung:** Benutzer können Fehlerprotokolle einsehen, Sensordaten analysieren, die Flotte überwachen, usw.
- **Datenanalyse:** Benutzer können Daten analysieren, Trends erkennen und Leistungsstatistiken anzeigen.
- **Konfiguration:** Registrierung und Konfiguration eines neuen Gerätes.

## 3. On-Site Steuerung: Steuerung vor Ort durch den/die ArealpflegerIn. Die Evaluation dieses UIs ist nicht Teil dieser Arbeit.

Die genannten Features sind in vielen Punkten mit der Aufstufung aus der Roadmap 24 kongruent.

Aus der Perspektive eines Bedieners wäre es natürlich wünschenswert, eine einzige eigenständige Lösung zu haben, anstatt mehrere Benutzeroberflächen verwenden zu müssen. Aber eine Aufteilung der UIs in Überwachung (WebUI), Analyse (Cloud) und Steuerung vor Ort ist aus mehreren Gründen sinnvoll:

1. **Trennung von Aufgaben:** Jedes UI hat eine spezifische Aufgabe und Verantwortung. Das WebUI ermöglicht die Echtzeitüberwachung des Roboters und bietet Benutzern einen umfassenden Überblick über den aktuellen Roboterstatus. Die Cloud als Analyse-Schnittstelle dient der umfangreichen Datenanalyse, -verarbeitung und -visualisierung für langfristige Trends und Leistungsanalysen. Die Steuerungsschnittstelle vor Ort ermöglicht die direkte Interaktion und Steuerung des Roboters vor Ort.
2. **Skalierbarkeit und Flexibilität:** Durch die Aufteilung der UIs kann jeder Teil unabhängig von den anderen skaliert und weiterentwickelt werden. Zum Beispiel können die Überwachungsfunktionen im WebUI erweitert werden, um zusätzliche Roboterparameter oder Sensordaten anzuzeigen, während die Analyse-Schnittstelle in der Cloud neue Algorithmen zur Datenverarbeitung implementieren kann. Dies ermöglicht eine bessere Anpassungsfähigkeit an spezifische Anforderungen und erleichtert die Integration neuer Funktionen.
3. **Sicherheitsaspekte:** Die Trennung der UIs bietet zusätzliche Sicherheitsebenen. Die Überwachungsschnittstelle im WebUI kann mit sicherheitsrelevanten Zugriffssteuerungen und Authentifizierungsoptionen versehen werden, um unbefugten Zugriff zu verhindern. Die Steuerungsschnittstelle vor Ort kann lokal abgeschirmt oder durch geeignete Sicherheitsmassnahmen geschützt werden, um Manipulationen oder Angriffe zu verhindern.
4. **Netzwerk- und Latenzaspekte:** Durch die Aufteilung der UIs kann die Kommunikation und Datenübertragung zwischen den verschiedenen Teilen optimiert werden. Die Überwachungsschnittstelle im WebUI kann über das Internet oder ein vom S-bot zur Verfügung gestelltes Netzwerk erreicht werden, während die Analyse-Schnittstelle in der Cloud von mehreren Standorten aus zugänglich ist. Die Steuerungsschnittstelle vor Ort ermöglicht eine direkte Kommunikation mit geringer Latenzzeit für Echtzeitsteuerungsaufgaben. Diese Verbindung könnte beispielsweise über Bluetooth realisiert werden.

Somit erfüllt jedes UI eine spezifische Aufgabe und ermöglicht es Benutzern, die benötigten Funktionen entsprechend ihren Anforderungen und Verantwortlichkeiten zu nutzen.



#### 4.2.1 Software-Architektur WebUI

Für das WebUI wurde folgendes, in Abbildung 26 dargestellte, System entwickelt. Der zentrale Rechner, der Intel nuc12, ist mit dem Betriebssystem Ubuntu 20.04 und ROS1 Noetic ausgestattet. Auf dem Ubuntu wird mittels Python ein lokaler HTTP-Server gestartet, welcher Zugriff auf die Dateien hat [52]. Der Benutzer verbindet sich nun ins gleiche Netz wie der S-bot und kann sich durch einen Browser mittels der IP-Adresse des S-bots auf den Server verbinden.

Der Rosbridge-Server ist ein Teil des `rosbridge_suite`-Pakets und ermöglicht die Kommunikation zwischen Webseiten und ROS über eine WebSocket-Transportschicht. Durch den Einsatz des Rosbridge-Servers können Webseiten direkt mit ROS interagieren und Informationen austauschen.

Ein Webbrowser kann eine Verbindung zu einem WebSocket herstellen, indem er die spezifische WebSocket-API verwendet. Dadurch können bidirektionale Kommunikation und Echtzeitaktualisierungen zwischen dem Webbrowser und dem Server ermöglicht werden, unabhängig davon, ob das WebSocket-Protokoll REST-konform ist oder nicht.

Der konkrete Fall dieser Anwendung am S-bot wird in Abbildung 26 gezeigt. Der Rosbridge-Server wird über das `rosbridge_suite`-Paket gestartet und dieser stellt den `rosbridge_websocket` zur Verfügung. Der WebSocket hat eine spezifische Adresse und einen Port, standardmässig ist die Adresse `ws://0.0.0.0:9090`. Das git-repository `a_bridge` enthält die erforderlichen Skripte, wie `index.html` für das Frontend des WebUI, `main.js` für das Backend der bidirektionalen Verbindung und die Verbindung zum WebSocket, sowie `location.js` für das Anzeigen der aktuellen GPS-Koordinaten. Durch diese Skripte entsteht eine bidirektionale Verbindung zwischen dem WebSocket und dem HTTP-Server. Das repository und die dazu erforderliche Dokumentation befindet sich auf [https://gitlab.switch.ch/sbot/a\\_bridge\(CommitID:15593423\)](https://gitlab.switch.ch/sbot/a_bridge(CommitID:15593423)). Des Weiteren wurde in Abbildung 30 ein Sequenzdiagramm erstellt, das die Interaktion zwischen den Entitäten sowie den zeitlichen Ablauf übersichtlich darstellt.

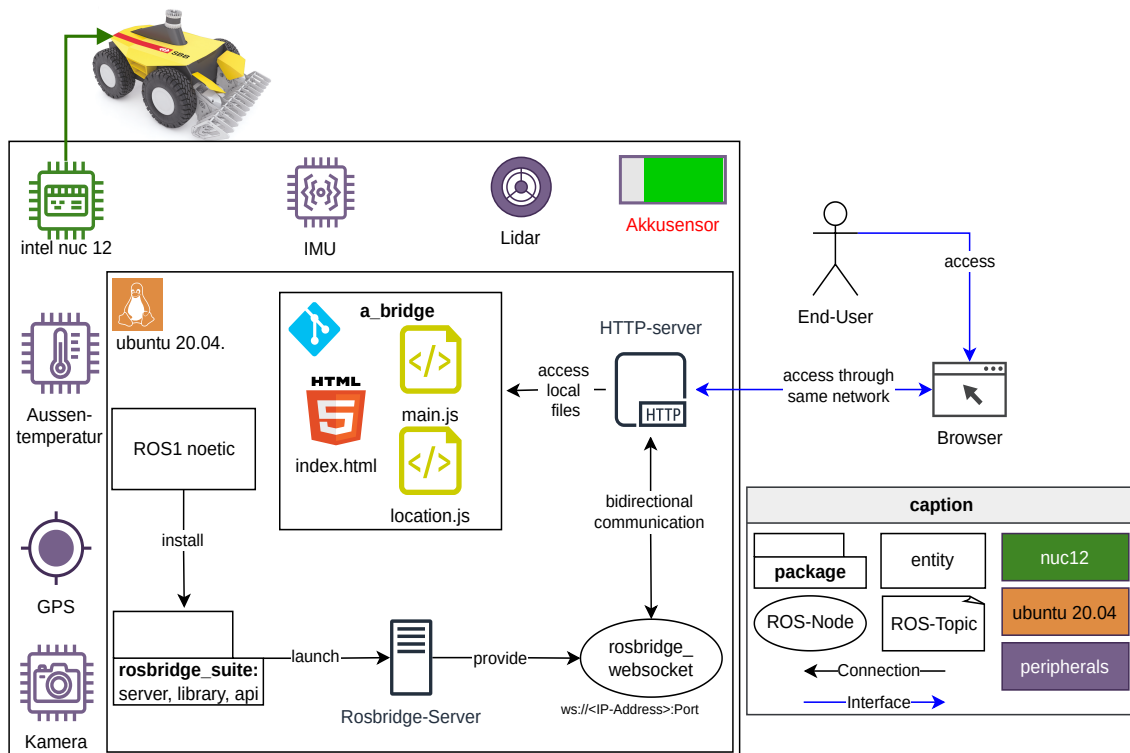


Figure 26: WebUI Architektur

Der Benutzer kann das WebUI verwenden, um Nachrichten an das topic `/chatter` zu senden. Der Datenfluss innerhalb der Architektur ist in Abbildung 27 dargestellt, wobei der Rosbridge-Server die WebSocket-Verbindung herstellt und JSON-Nachrichten an die `rosbridge_library` übergibt, um sie in ROS-Aufrufe umzuwandeln. Die `rosbridge_library` konvertiert wiederum ROS-Antworten in JSON und sendet sie über die WebSocket-Verbindung zurück zum Webbrowser.

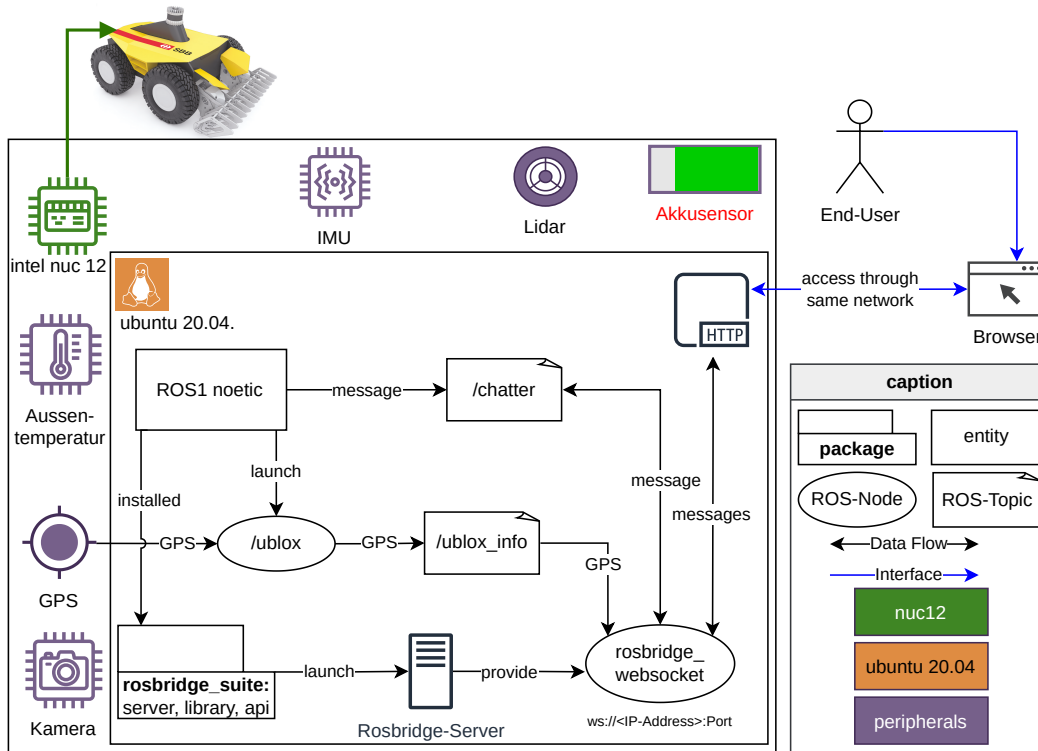


Figure 27: Datenfluss WebUI Architektur

#### 4.2.2 Webpage

Die Webpage wurde bewusst einfach gestaltet. Mithilfe der Skripte von `bootstrap.css` und `vue.js` konnte das HTML-Dokument `index.html` deklarativ programmiert werden. Durch die Verwendung von Vue im Feld "Connection to WebSocket" kann die Adresse des Websocket eingetragen und anschliessend damit verbunden werden. Im Bereich "Send Messages" kann auf das topic `/chatter` eine message gesendet werden. Die Liste im Bereich "Received Messages" zeigt alle bidirektional übertragenen messages des topics an. Dieses klassische Publish/Subscribe-Prinzip wurde mithilfe von `roslib.js` entwickelt, weitere Einzelheiten dazu sind im Sequenzdiagramm 30 dargestellt.

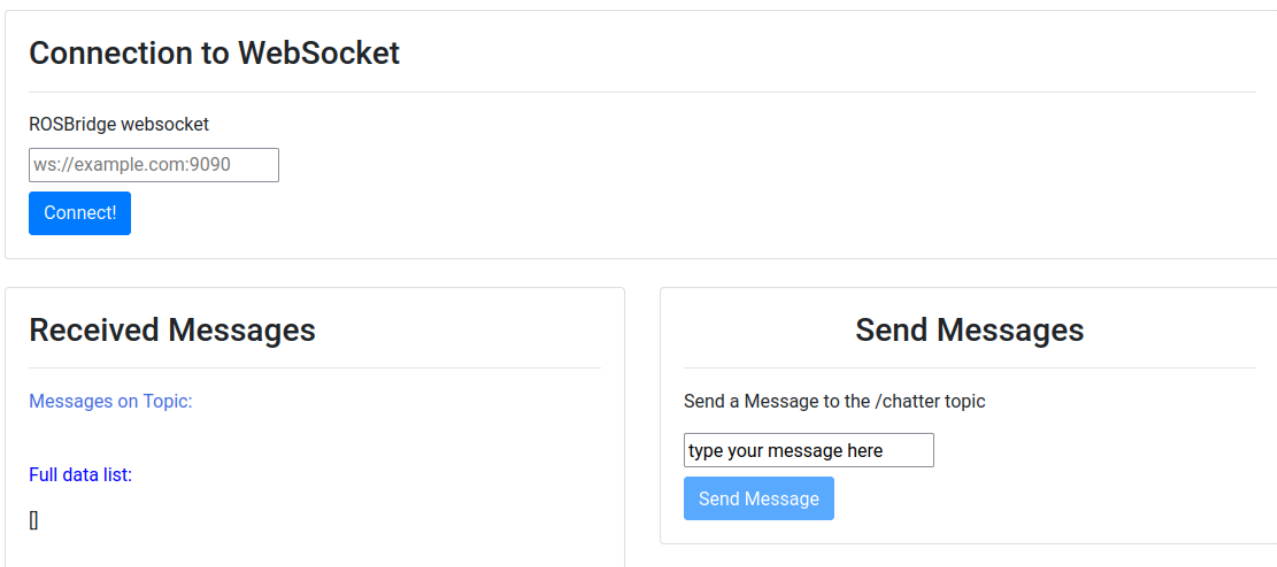


Figure 28: Ausschnitt aus dem WebUI mit Feature der bidirektionalen Verbindung

Ein weiteres Feature ist die Darstellung der GPS-Koordinaten des S-bot auf einer interaktiven Karte. Hierfür wurde das Skript `location.js` entwickelt, das auf umfangreiche Skripte wie `leaflet.js` und das Plugin `Leaflet.TileLayer.MBTiles.js` zurückgreift. Durch diese Integration ist es möglich, das `topic/ublox/ublox_info` während einer aktiven Verbindung zum Websocket zu abonnieren (siehe Abbildung 27). Sobald der GPS-Sensor des S-bots messages veröffentlicht, werden die Positionsdaten in Soft-Realtime auf der Karte angezeigt und gleichzeitig als Gleitkommazahlen unterhalb der Karte präsentiert. Diese visuelle Darstellung ermöglicht es dem Benutzer, die genaue Position und deren Veränderung zu verfolgen.

## Location

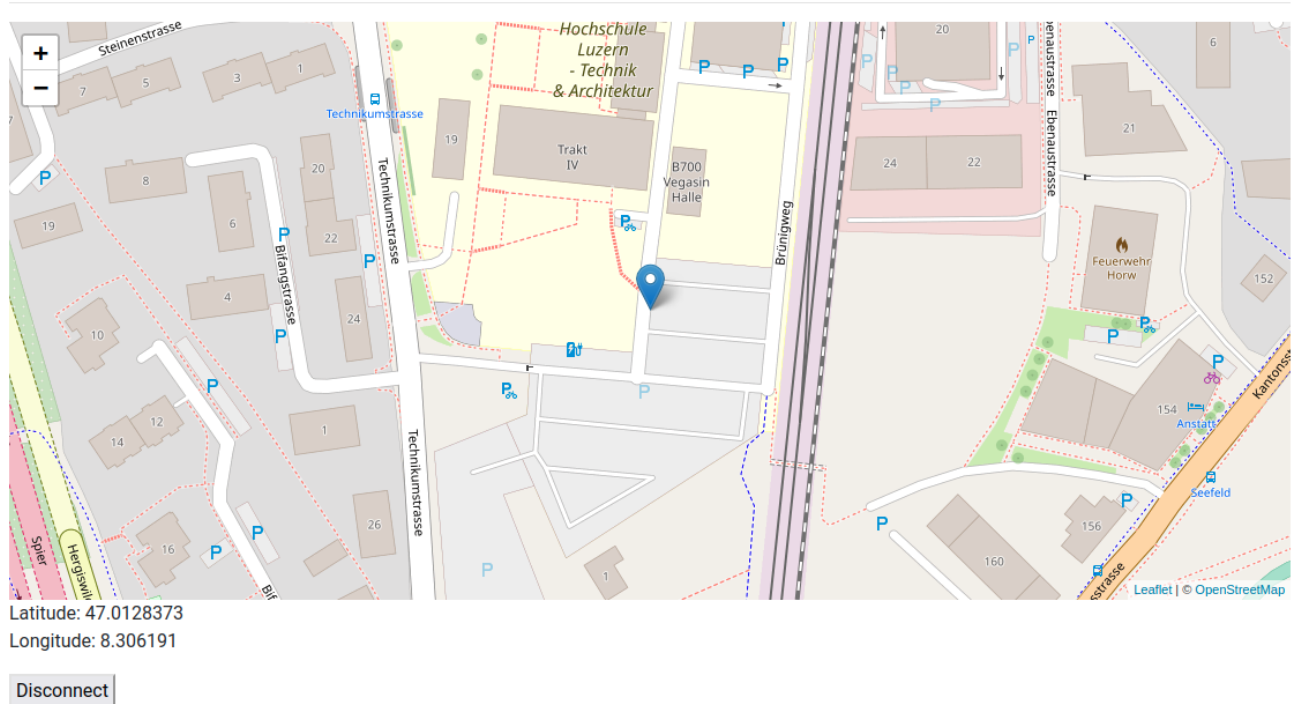


Figure 29: Canvas des WebUI mit Feature von GPS-Koordinaten

Im repository `a_bridge` sind zwei Ordner erstellt worden: `webpage_online` und `webpage_offline`. Wenn der S-bot eine aktive Internetverbindung hat, können alle erforderlichen Skripte zur Darstellung online geladen werden. Falls der S-bot keine Verbindung zum Internet hat, aber der Benutzer sich im selben Netzwerk befindet, können die Karten als Tiles und Layers lokal auf dem S-bot gespeichert werden. In diesem Fall werden auch alle Skripte lokal gespeichert. Die Tiles wurden mithilfe von QGIS extrahiert und nach dem entsprechenden Zoomlevel in die Ordner strukturiert. Der genaue Ablauf ist im Sequenzdiagramm 30 ersichtlich.

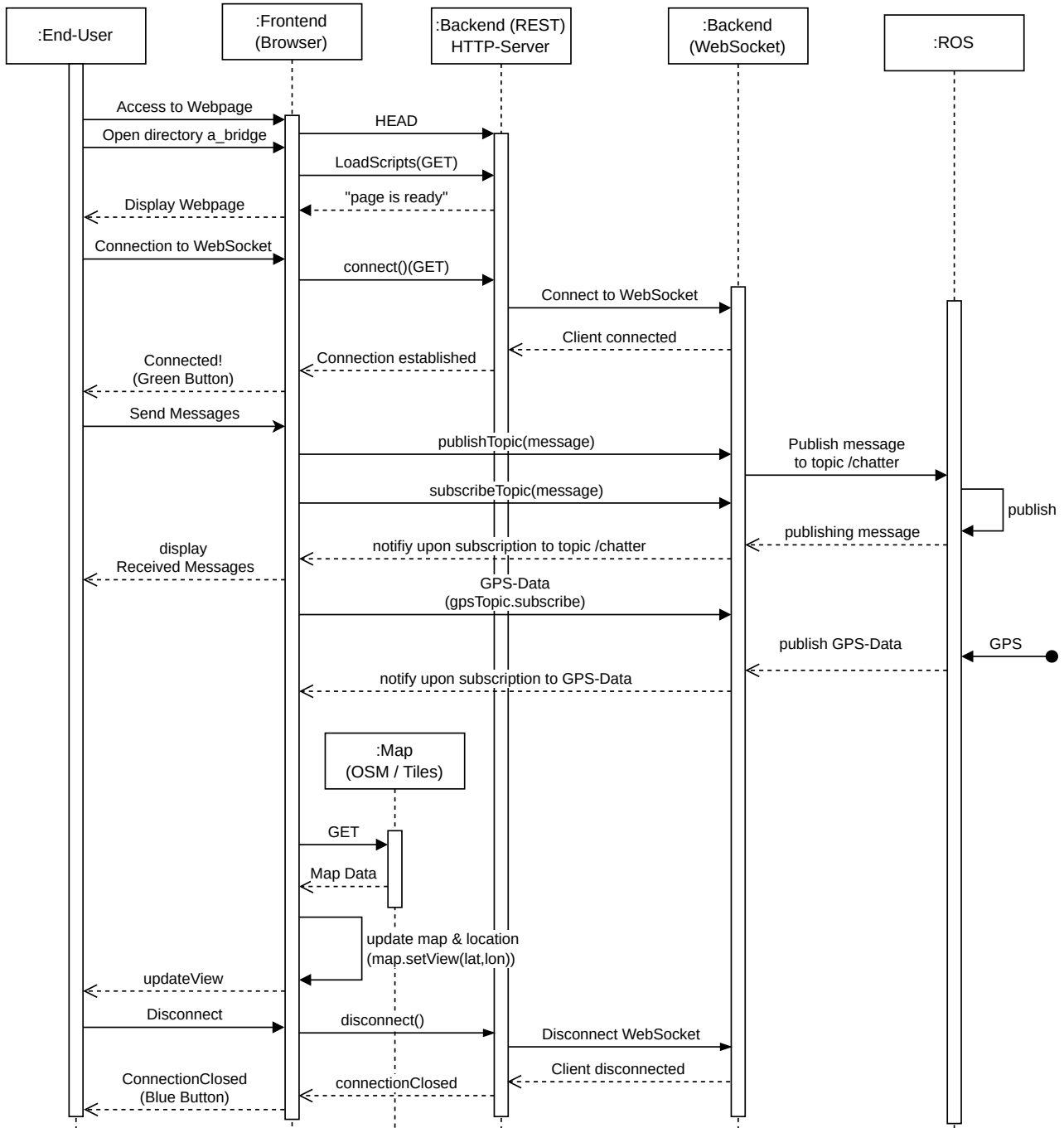


Figure 30: Sequenzdiagramm für das WebUI

### 4.3 Aufbau der Cloud mit Azure

Aufgrund der bereits bestehenden Zusammenarbeit des Industriepartners mit Microsoft wurde Azure als Cloudanbieter für das Projekt ausgewählt. Um den Anforderungen gerecht zu werden, war die Implementierung einer massgeschneiderten Lösung mit umfangreichen Funktionen und Diensten erforderlich. Da IoT Central bestimmte benötigte Features nicht unterstützt, wurde bewusst eine PaaS-Lösung mit dem Azure IoT Hub als zentralem Element gewählt. Hierfür wurde eine dedizierte Ressourcengruppe erstellt, um die erforderlichen Cloud Services zu verwalten. Durch die Integration des IoT Hubs wurde die bestehende bidirektionale Verbindung zwischen dem S-bot und dem Endnutzer um eine unidirektionale Verbindung zur Cloud erweitert. Eine detaillierte Darstellung der entwickelten IoT- und Cloud-Architektur ist in Abbildung 31 zu sehen, die die Komponenten und ihre Beziehungen veranschaulicht.

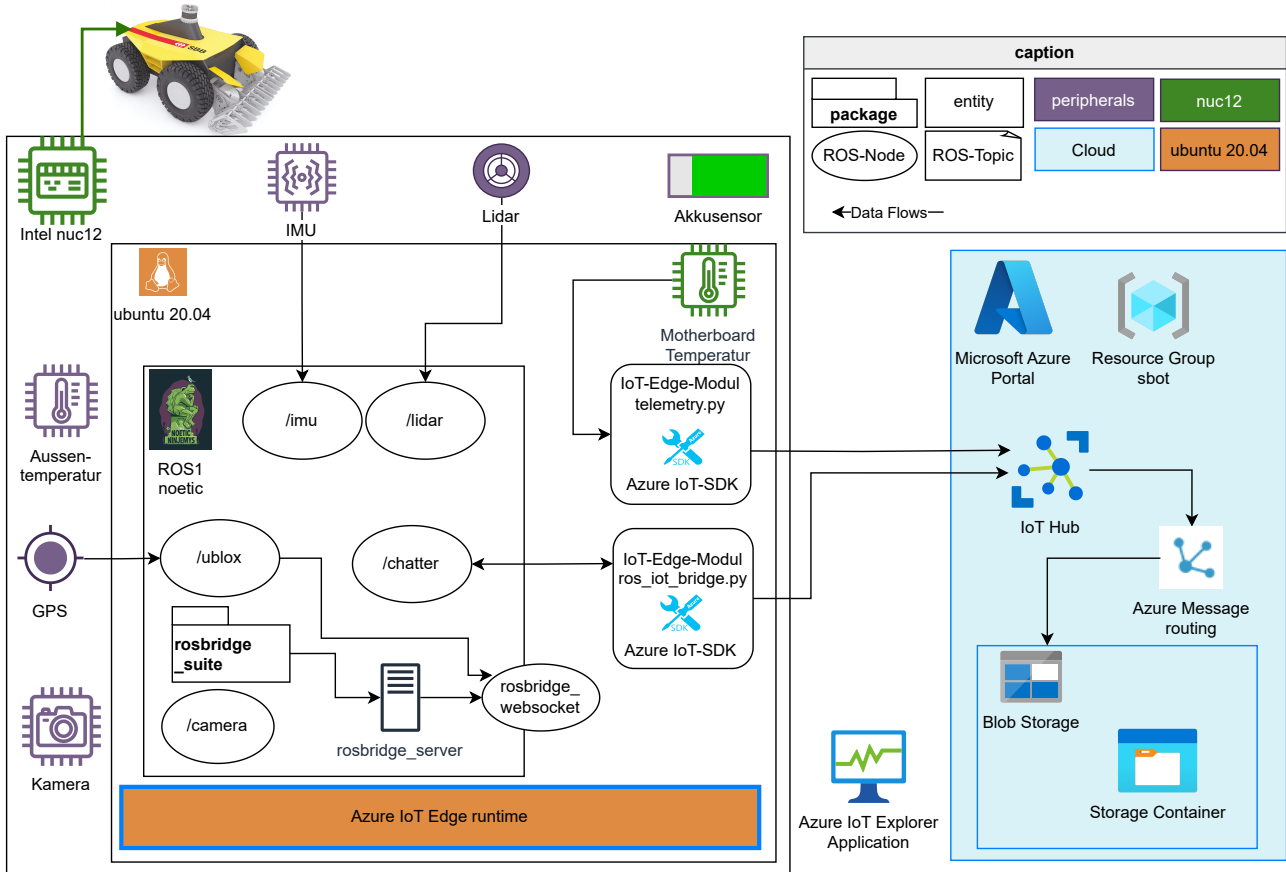


Figure 31: Cloud Architektur

#### 4.3.1 IoT Edge

Mit Hilfe der Azure-IoT-SDKs und Python wurden zwei Skripte namens `telemetry.py` und `ros_iot_bridge.py` entwickelt und auf ein neues gitlab-repository namens `Azure_IoTHub_wss` gepusht. Das repository und die dazu erforderliche Dokumentation befindet sich auf [https://gitlab.switch.ch/sbot/azure\\_iothub\\_wss\(CommitID:e16923ec\)](https://gitlab.switch.ch/sbot/azure_iothub_wss(CommitID:e16923ec)).

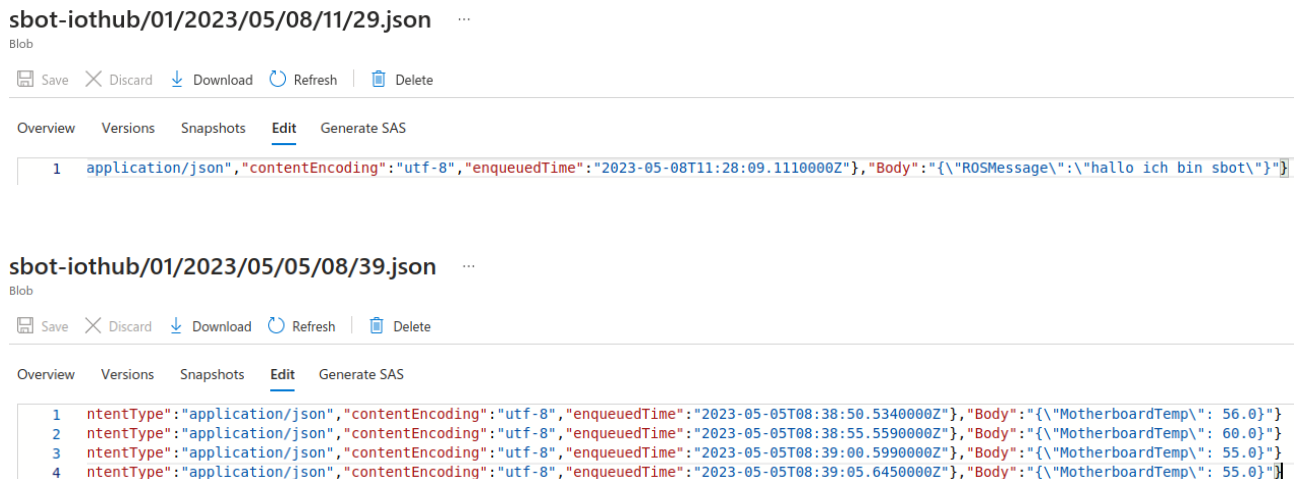
Der erste Code implementiert ein Telemetrie-Beispiel, das Daten vom Motherboard des nuc12 ausliest und diese an den IoT Hub von Azure sendet. Es wird eine Verbindung zum IoT Hub hergestellt und in einer Endlosschleife wird die Temperatur des Motherboards ausgelesen, in ein JSON-Payload verpackt und als Telemetrienachricht an den IoT Hub gesendet. Die Anzahl der gesendeten Nachrichten wird gezählt und am Ende ausgegeben. Bei einer Unterbrechung der Ausführung durch den Benutzer wird das Programm beendet.

Der zweite Code implementiert ein ROS-basiertes Telemetrie-Beispiel. Es verbindet sich mit dem IoT Hub von Azure und wartet auf eine Nachricht, die über das ROS-topic `/chatter` empfangen wird. Sobald eine Nachricht empfangen wird, wird sie ebenfalls in ein JSON-Payload verpackt und als Telemetrienachricht an den IoT Hub gesendet. Die Anzahl der gesendeten Nachrichten wird gezählt und am Ende ausgegeben. Wenn keine Nachricht innerhalb einer bestimmten Zeit empfangen wird, wird das Programm beendet. Bei einer Unterbrechung der Ausführung durch den Benutzer wird das Programm ebenfalls beendet.

Durch die Azure IoT Explorer Anwendung und die Edge runtime konnten die jeweilig gesendeten Nachrichten des S-bots überwacht werden.

### 4.3.2 Cloud Services

Die Telemetriedaten werden über den Azure IoT Hub empfangen und anschliessend mittels Message Routing an den Blob Storage weitergeleitet. Dabei wurde das Routing Query entsprechend den spezifischen Anforderungen der Nachrichten und des gewünschten Endpoints in der Ressourcengruppe konfiguriert. Im Blob Storage wird ein Data Lake erstellt, in dem die Daten sicher gespeichert werden. Die Daten können durch einen Drilldown- oder Exploration-Prozess gezielt abgerufen und analysiert werden. Als Standardformat für die Darstellung und den Austausch der Daten wird das JSON-Format verwendet, das eine strukturierte und maschinenlesbare Repräsentation ermöglicht. Dies eröffnet vielfältige Möglichkeiten zur weiteren Verarbeitung und Analyse der Daten im Azure IoT-Ökosystem.



**sbot-iothub/01/2023/05/08/11/29.json** ...

Blob

Save Discard Download Refresh Delete

Overview Versions Snapshots Edit Generate SAS

```
1 application/json, "contentEncoding": "utf-8", "enqueuedTime": "2023-05-08T11:28:09.1110000Z", "Body": "{\r\n  \"R05Message\": \"hallo ich bin sbot\" }"}]
```

**sbot-iothub/01/2023/05/05/08/39.json** ...

Blob

Save Discard Download Refresh Delete

Overview Versions Snapshots Edit Generate SAS

```
1 ntentType": "application/json", "contentEncoding": "utf-8", "enqueuedTime": "2023-05-05T08:38:50.5340000Z", "Body": "{\r\n  \"MotherboardTemp\": 56.0 }"}
2 ntentType": "application/json", "contentEncoding": "utf-8", "enqueuedTime": "2023-05-05T08:38:55.5590000Z", "Body": "{\r\n  \"MotherboardTemp\": 60.0 }"}
3 ntentType": "application/json", "contentEncoding": "utf-8", "enqueuedTime": "2023-05-05T08:39:00.5990000Z", "Body": "{\r\n  \"MotherboardTemp\": 55.0 }"}
4 ntentType": "application/json", "contentEncoding": "utf-8", "enqueuedTime": "2023-05-05T08:39:05.6450000Z", "Body": "{\r\n  \"MotherboardTemp\": 55.0 }"}]
```

Figure 32: Telemetriedaten nach Drilldown auf dem Storage Container

## 5 Diskussion und Ausblick

In diesem Kapitel wird untersucht, inwieweit die Funktionsanforderungen, die in der Einleitung 1 formuliert wurden, erfüllt und die dazugehörigen Fragen beantwortet wurden. Darüber hinaus werden am Ende dieses Kapitels Ausblicke gegeben und über das weitere Vorgehen informiert.

### 5.1 Diskussion

Zu jedem Thema wird eine kurze Zusammenfassung gemacht. Um die Ergebnisse in den Kontext des Vorgehens und der angewandten Methoden zu rücken, ist eine kritische Reflexion nötig und hier ebenfalls enthalten.

#### 5.1.1 Anforderungserhebung

Die Notwendigkeit einer priorisierten Anforderungsliste ergab sich erst im Verlauf der Konzeption und Entwicklung des WebUIs. Bereits mit den grundlegenden Funktionsanforderungen wurde ein Teil festgelegt. Durch die erste Anforderungserhebung wurde der Anfang eines iterativen Prozesses angestoßen. Um festzustellen, welche Features erforderlich sind und in welcher Reihenfolge sie entwickelt werden sollten, wurden sie von den verschiedenen Stakeholdern bewertet. Die priorisierte Anforderungsliste wurde anhand des Durchschnitts der Bewertungen erstellt. Zusätzliche Vorbedingungen, die teilweise aus individuellen Perspektiven der Stakeholder stammen, wurden in konsolidierter Form in Stories zusammengefasst und anschliessend in Epics gruppiert. Da es sich bei den User Stories um ConOps handelt, sind die Anforderungen nicht in klarer Prosa dokumentiert, wie es in einem herkömmlichen Anforderungskatalog üblich wäre. Dies ermöglicht jedoch eine gemeinsame Sprache zwischen Kunden/Benutzern und Entwicklern.

Retrospektiv wäre es sinnvoll gewesen, von Anfang an eine digitale Version des Fragebogens zu erstellen, da dies die Auswertung insgesamt erleichtert hätte. Des Weiteren wäre es von Vorteil gewesen, die Funktionen im WebUI von den Stakeholder zusammenzutragen, anstatt sie nur aus der Perspektive des Entwicklers zu erheben. Allerdings wäre dies aus zeitlichen Gründen schwierig gewesen. Auf der Grundlage des erstellten Product Backlogs kann nun aufgebaut werden, und dazu wird in Kapitel 5.2.1 ein Vorschlag gemacht. Zudem ist anzumerken, dass eine grössere Aussagekraft bezüglich des Product Backlogs erreicht werden kann, wenn bei zukünftigen Iterationen der Anforderungserhebung mehr Stakeholder einbezogen werden.

#### 5.1.2 Ausarbeitung WebUI

Die Entwicklung des WebUIs für den S-bot wurde durch eine gründliche Recherche, das Studium von Tutorials und umfangreiche Tests webbasierter Interfaces ermöglicht. Diese Schritte bildeten die Grundlage für ein fundiertes Konzept und eine erste Umsetzung des WebUIs, das die Fernüberwachung als auch die Diagnostik des S-bots ermöglicht. Zu Beginn wurde viel mit WAMP ausprobiert, doch es wurden keine nennenswerten Implementationen im Zusammenhang mit Robotik-Anwendungen gefunden und deshalb wurde WAMP nicht weiter berücksichtigt.

Ein zentraler Entwicklungsschritt bestand in der Implementierung einer bidirektionalen Verbindung zwischen ROS, bzw. dem Websocket und dem HTTP-Server. Diese Verbindung ermöglicht den Austausch von Informationen in Soft-Realtime und erlaubt es, Daten in Textform auf ein spezifisches topic zu schreiben, um den S-bot zu steuern. Dadurch konnte die Darstellung von Informationen des S-bots auf dem WebUI realisiert werden. Inwiefern diese Funktion für die Fehlerdiagnose des S-bots zu gebrauchen ist, gilt es zuerst innerhalb der Anforderungen zu evaluieren.

Die Integration eines spezifischen Kartenformats in Form einer Gridmap wurde aufgrund anderer höher priorisierter Entwicklungsschritte im RoVeKo-Projekt nicht umgesetzt. Dennoch wurde die Einbindung von Open-Source-Karten von OSM sowie das Laden lokaler Map-Tiles im Offline-Modus ermöglicht. Durch den GPS-Sensor des S-bots kann dessen Position auf einer interaktiven Karte verfolgt werden. Eine direkte Einbindung in den Navigationsalgorithmus wurde jedoch aus zeitlichen Gründen nicht weiterverfolgt. Trotzdem konnten die grundlegenden Anforderungen hinsichtlich der Kartenfunktionalität weitgehend erfüllt werden.

Die Aufteilung der Benutzeroberfläche in separate Bereiche für Überwachung, Analyse und lokale Steuerung trug zur Schaffung einer skalierbaren und flexiblen Lösung bei, die an verschiedene Anwendungsfälle angepasst werden kann. Diese modulare Struktur ermöglicht die Unterstützung zukünftiger Anforderungen und Erweiterungen. Die entwickelte Architektur und die eingesetzten Technologien bilden eine solide Grundlage für die Weiterentwicklung des Systems.

### 5.1.3 Konzipierung Cloud-Architektur

Ursprünglich war eine Anbindung des S-bots an die Cloud nicht geplant, jedoch wurde im Laufe des Projekts erkannt, dass das Thema IoT eine relevante Rolle spielt. Die Entwicklung einer cloudbasierten Lösung erforderte umfangreiche Recherche sowie die praktische Umsetzung der in den Tutorials genannten Schritte. Dabei wurde zunächst nach bestehenden Systemen gesucht, und es wurde eine Anwendung zur Verbindung von ROS und der Azure Cloud namens “ros.azure.iothub” ([https://github.com/microsoft/ros\\_azure\\_iothub](https://github.com/microsoft/ros_azure_iothub)) gefunden. Allerdings führten Herausforderungen wie fehlende Lizenzen und die Schwierigkeit, ROS mit Microsoft-kompatiblen Systemen zu verbinden, zur Entscheidung eine eigene Lösung zu entwickeln.

Der Azure IoT Hub wurde als zentrales Element verwendet, um eine sichere und skalierbare Kommunikation zwischen dem S-bot und der Cloud zu gewährleisten. Durch die Entwicklung spezifischer Skripte wie `telemetry.py` und `ros_iiot_bridge.py` konnten Telemetriedaten an den IoT Hub übertragen werden. Anschliessend wurden die Daten über das Message Routing an den Blob Storage weitergeleitet und dort sicher gespeichert. Dies ermöglichte eine erste Implementierung des “Cold Path” in Bezug auf die Datenspeicherung.

Verschiedene andere Systeme und darunterliegende Technologien wären für den S-bot auch interessant gewesen, doch die Entwicklung des WebUIs hatte eine höhere Priorität. Dennoch wurden mit der Integration des Azure IoT Hubs und des Blob Storage wichtige Grundlagen für eine Cloud-Integration gelegt. Dieses Grundkonzept der Cloud-Komponente ist vorläufig, bietet jedoch Raum für zukünftige Erweiterungen.

## 5.2 Ausblick

Abschliessend wird nun ein Ausblick und Vorschläge zum weiteren Vorgehen beschrieben.

### 5.2.1 Zweite Iteration der Anforderungserhebung

Der Product Backlog auf Jira ist das Ergebnis der ersten Iteration. In einer zweiten Iteration der Anforderungserhebung nach ©iSREM sollen erneut die drei Phasen durchlaufen werden.

#### 5.2.1.1 Definition

Die Stories beschreiben die einzelnen Funktionen, während die Epics dazu dienen, sie zu kategorisieren. Durch die automatische Nummerierung in Jira sind die Definitionen bereits vorhanden. Dennoch kann das Erstellen einer Anforderungsdefinitions-Matrix (ADM) gemäss [48, S.162-165] dabei helfen, die Anforderungen zu klassifizieren und um den Überblick zu behalten.

#### 5.2.1.2 Gewichtung

Einige Features, die in der ersten Gewichtung berücksichtigt wurden, zeigen eine hohe Standardabweichung. Um die Uneinigkeit hinsichtlich der Bedeutung der zu entwickelnden Features auszugleichen, kann die Methode des paarweisen Vergleichs angewendet werden. Es ist daher wichtig sicherzustellen, dass subjektive Meinungen keinen unangemessenen Einfluss auf die Gewichtung bestimmter Anforderungen haben. Daher sollte der Priorisierungsprozess so objektiv wie möglich gestaltet werden. Die Gewichtung erfolgt in Zusammenarbeit mit Vertretern der verschiedenen Stakeholder und wird für jedes Epic einzeln durchgeführt, um sowohl eine relative als auch eine absolute Bewertung zu erzielen. Eine entsprechende Vorlage, adaptiert von [48, S.166], ist in Abbildung 33 dargestellt. Daraus kann ein priorisierter Product Backlog abgeleitet werden.



1. Paarvergleich der Anforderungen														
Epic WEBUI-31: Befehle und Steuerung aus dem WebUI										Stakeholder A	Gesamt	Gesamt		
										Total Punkte	% - Prozent	Mittelwert Punkte	Mittelwert Prozent	
Legende: wichtiger = 3 gleich wichtig = 2 weniger wichtig = 1														
										nach				
										von				
WEBUI-33 Analyse inwiefern ein sofortiger Notstopp möglich ist											0	#DIV/0!	#REF!	#REF!
WEBUI-32 Senden von Reset / Reboot / Pause											0	#DIV/0!	#REF!	#REF!
WEBUI-45 Zeitfenster festlegen											0	#DIV/0!	#REF!	#REF!
WEBUI-54 Kamera-Snapshot auslösen											0	#DIV/0!	#REF!	#REF!
WEBUI-55 Pull Logs											0	#DIV/0!	#REF!	#REF!
WEBUI-14 Warnlicht/Blitzlicht oder Warnton anschalten											0	#DIV/0!	#REF!	#REF!
WEBUI-17 Kommunikation über Headset mit Sprachausgabe beim Roboter											0	#DIV/0!	#REF!	#REF!
<b>Total Punkte =</b>										0	#DIV/0!	#REF!	#REF!	

©iProcess AG. Alle Rechte vorbehalten: Die in diesem Werk beschriebene Methodik, einschliesslich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung ausserhalb der Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung der iProcess AG gesetzlich unzulässig.

Figure 33: Vorlage einer Paarvergleichs-Matrix für ein Epic aus [48]

### 5.2.1.3 Beschreibung

“In der letzten methodischen Phase werden die in der ersten methodischen Phase definierten und in der zweiten Phase gewichteten und priorisierten Anforderungen konkretisiert. Der resultierende Liefergegenstand ist eine Anforderungsspezifikations-Matrix (ASM), die die Anforderungen technisch bis ins letzte Detail beschreibt und auf der ursprünglichen ADM aufbaut. So sollte ein Entwicklungsingenieur in der Lage sein, mit Hilfe der ASM eine detaillierte Beschreibung der systemtechnischen Implementierungslösungen zu erstellen.” [48, S.168] Dadurch kann ein Refinement Backlog abgeleitet werden, der spezifisch für die Entwickler gedacht ist. Eine Übertragung dessen in das Kanban Board ist der letzte Schritt, bevor neue Features implementiert werden können.

### 5.2.2 Demo-Use-Case

Der weitere Fortschritt der WebUI-Entwicklung hängt massgeblich von der zweiten Iteration der Anforderungserhebung ab. Dabei werden die Features mit höchster Priorität aus dem Product Backlog priorisiert und zuerst entwickelt.

Eine erste Anwendung des WebUIs wird voraussichtlich Ende August 2023 während einer Zwischenpräsentation für den Industriepartner stattfinden. Dabei wird insbesondere die Positionsüberwachungsfunktion präsentiert, um zu demonstrieren, wie eine Fernüberwachung des S-bots realisiert werden kann. Diese Demonstration wird wichtige Einblicke und Rückmeldungen liefern und als Grundlage für weitere Entwicklungen dienen.

### 5.2.3 Weitere Cloudservices

Es gibt zahlreiche Cloudservices, die in Zukunft für den S-bot Anwendungen finden könnten, wie das Registrieren neuer Geräte durch das DPS, die Implementierung fortschrittlicherer Algorithmen zur Datenanalyse in der Cloud oder die Untersuchung weiterer Sicherheitsaspekte, um einige zu nennen. Die Integration von KI-Algorithmen zur automatischen Fehlererkennung und -diagnose oder zur Optimierung von Routenplanung und Navigation des S-bots könnte ein interessantes Forschungsthema sein.

Ein erster Schritt könnte die Verbindung des Websocket zur Cloud mittels Advanced Message Queuing Protocol (AMQP) und dem Tutorial “Auswählen eines Protokolls für die Gerätekommunikation” sein. Damit könnten alle Roboterdaten, die über den WebSocket laufen, direkt in die Cloud eingespielen werden.

## Literaturverzeichnis

- [1] T. Thesing, C. Feldmann, and M. Burchardt, “Agile versus waterfall project management: Decision model for selecting the appropriate approach to a project,” *Procedia Computer Science*, vol. 181, pp. 746–756, 2021, CENTERIS 2020 - International Conference on ENTERprise Information Systems / ProjMAN 2020 - International Conference on Project MANagement / HCist 2020 - International Conference on Health and Social Care Information Systems and Technologies 2020, CENTERIS/ProjMAN/HCist 2020, ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.01.227>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050921002702>.
- [2] M. Fowler, J. Highsmith, *et al.*, “The agile manifesto,” *Software development*, vol. 9, no. 8, pp. 28–35, 2001.
- [3] G. D. Padfield, “Rotorcraft virtual engineering; supporting life-cycle engineering through design and development, test and certification and operations,” *The Aeronautical Journal*, vol. 122, no. 1255, pp. 1475–1495, 2018. DOI: 10.1017/aer.2018.47.
- [4] B. Gloger, “Scrum,” *Informatik-Spektrum*, vol. 33, no. 2, pp. 195–200, 2010.
- [5] T. Epping, *Kanban für die Softwareentwicklung*. Springer-Verlag, 2011.
- [6] T. Sedano, P. Ralph, and C. Péraire, “The product backlog,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 200–211. DOI: 10.1109/ICSE.2019.00036.
- [7] Official ROS Wiki, *ROS Introduction*, <https://wiki.ros.org/ROS/Introduction>. (visited on 05/15/2023).
- [8] —, *ROS Concepts*, <https://wiki.ros.org/ROS/Concepts>, 2. ROS Computation Graph Level. (visited on 03/07/2023).
- [9] —, *ROS Nodes*, <https://wiki.ros.org/Nodes>. (visited on 05/15/2023).
- [10] —, *ROS Topics*, <https://wiki.ros.org/Topics>. (visited on 05/15/2023).
- [11] —, *ROS Messages*, <https://wiki.ros.org/Messages>. (visited on 05/15/2023).
- [12] —, *rosbridge\_suite*, [https://wiki.ros.org/rosbridge\\_suite](https://wiki.ros.org/rosbridge_suite). (visited on 02/21/2023).
- [13] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.
- [14] Don Box, DevelopMentor, David Ehnebuske, IBM, Gopal Kakivaya, Microsoft, Andrew Layman, Microsoft, Noah Mendelsohn, Lotus Development Corp., Henrik Frystyk Nielsen, Microsoft, *et. al.*, *Simple Object Access Protocol (SOAP)*, <http://www.w3.org/TR/SOAP>. (visited on 05/18/2023).
- [15] E. Christensen, F. Curbera, G. Meredith, and S. e. a. Weerawarana, *Web Services Description Language (WSDL) 1.1*, 2001.
- [16] The Bootstrap Authors, *Bootstrap project*, <https://getbootstrap.com/docs/5.3/getting-started/introduction/>, 2011–2023. (visited on 05/03/2023).
- [17] Evan You and Vue.js developers, *Vue.js – The Progressive JavaScript Framework v3.0*, <https://vuejs.org/guide/introduction.html>, 2014–2023. (visited on 03/23/2023).
- [18] I. Fette and A. Melnikov, *The WebSocketProtocol*, <https://www.rfc-editor.org/rfc/rfc6455>, Abstract. (visited on 02/20/2023).
- [19] —, *The WebSocketProtocol*, <https://www.rfc-editor.org/rfc/rfc6455>, Chapter 1.2 Protocol Overview. (visited on 02/20/2023).
- [20] —, *The WebSocketProtocol*, <https://www.rfc-editor.org/rfc/rfc6455>, Chapter 1.5 Design Philosophy. (visited on 02/20/2023).
- [21] Crossbar.io Technologies GmbH, *The Web Application Messaging Protocol*, <https://wamp-proto.org/>. (visited on 03/24/2023).
- [22] OpenStreetMap Foundation, *Main Page — OpenStreetMap Foundation*, <https://wiki.osmfoundation.org/w/index.php?title=Licence&oldid=8605>, 2021. (visited on 05/02/2023).
- [23] V. Agafonkin, *Leaflet.js – open-source JavaScript library for interactive maps v1.9.4*, <https://leafletjs.com/>, 2010–2023. (visited on 05/02/2023).
- [24] QGIS is a volunteer driven project., *QGIS Open Source Geographic Information System*, <https://qgis.org/en/site/>, 2002–2023. (visited on 05/03/2023).
- [25] Husqvarna Website, *Husqvarna Fleet Services*, <https://www.husqvarna.com/ch-de/services/fleet-services/>. (visited on 03/21/2023).

- [26] Eclipse Foundation, *Eclipse zenoh*, <https://zenoh.io/docs/overview/what-is-zenoh/>. (visited on 03/25/2023).
- [27] M. P. Papazoglou and W.-J. van den Heuvel, “Service oriented architectures: Approaches, technologies and research issues,” *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, Jul. 2007, ISSN: 0949-877X. [Online]. Available: <https://doi.org/10.1007/s00778-007-0044-3>.
- [28] P. Massuthe, W. Reisig, and K. Schmidt, *An Operating Guideline Approach to the SOA*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Informatik, 2005. DOI: <http://dx.doi.org/10.18452/2443>.
- [29] R. Bouziane, L. S. Terrissa, and S. Ayad, “Semantic web services for ros: A robot as a service approach,” *Automated Software Engineering*, vol. 29, no. 2, p. 49, Jul. 2022, ISSN: 1573-7535. [Online]. Available: <https://doi.org/10.1007/s10515-022-00346-w>.
- [30] A. Koubaa, “Ros as a service: Web services for robot operating system,” *Journal of Software Engineering for Robotics*, vol. 1, p. 1, Dec. 2015.
- [31] R. Bouziane, L. S. Terrissa, S. Ayad, and J.-F. Brethé, “Towards an architecture for cloud robotic services,” *International Journal of Computers and Applications*, vol. 44, no. 9, pp. 863–874, 2022. DOI: 10.1080/1206212X.2021.1964790. [Online]. Available: <https://doi.org/10.1080/1206212X.2021.1964790>.
- [32] A. Koubaa, “A service-oriented architecture for virtualizing robots in robot-as-a-service clouds,” in *Architecture of Computing Systems – ARCS 2014*, E. Maehle, K. Römer, W. Karl, and E. Tovar, Eds., Cham: Springer International Publishing, 2014, pp. 196–208, ISBN: 978-3-319-04891-8.
- [33] Official ROS Wiki, *The Standard ROS JavaScript Library*, <https://wiki.ros.org/roslibjs>. (visited on 03/08/2023).
- [34] Y. Chen, Z. Du, and M. García-Acosta, “Robot as a service in cloud computing,” in *2010 Fifth IEEE International Symposium on Service Oriented System Engineering*, 2010, pp. 151–158. DOI: 10.1109/SOSE.2010.44.
- [35] V. Dawarka and G. Bekaroo, “Building and evaluating cloud robotic systems: A systematic review,” *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102 240, 2022, ISSN: 0736-5845. DOI: <https://doi.org/10.1016/j.rcim.2021.102240>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0736584521001216>.
- [36] P. Mell and T. Grance, *The nist definition of cloud computing*, en, 2011. DOI: <https://doi.org/10.6028/NIST.SP.800-145>.
- [37] R. Stackowiak, “Azure IoT Solutions Overview,” in *Azure Internet of Things Revealed: Architecture and Fundamentals*. Berkeley, CA: Apress, 2019, pp. 29–54, ISBN: 978-1-4842-5470-7. DOI: 10.1007/978-1-4842-5470-7\_2. [Online]. Available: [https://doi.org/10.1007/978-1-4842-5470-7\\_2](https://doi.org/10.1007/978-1-4842-5470-7_2).
- [38] Microsoft Corporation, *Choose an IoT solution in Azure*, <https://learn.microsoft.com/en-us/azure/architecture/example-scenario/iot/iot-central-iot-hub-cheat-sheet>. (visited on 04/11/2023).
- [39] —, *Azure Certified Device Catalog*, <https://devicecatalog.azure.com/featured>. (visited on 05/21/2023).
- [40] J. Soh, M. Copeland, A. Puca, and M. Harris, “Machine Learning and Deep Learning,” in *Microsoft Azure: Planning, Deploying, and Managing the Cloud*. Berkeley, CA: Apress, 2020, pp. 325–368, ISBN: 978-1-4842-5958-0. DOI: 10.1007/978-1-4842-5958-0\_16. [Online]. Available: [https://doi.org/10.1007/978-1-4842-5958-0\\_16](https://doi.org/10.1007/978-1-4842-5958-0_16).
- [41] Microsoft Corporation, *What is Azure IoT Edge*, <https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge?view=iotedge-1.4>. (visited on 05/22/2023).
- [42] R. Stackowiak, “Azure IoT Hub,” in *Azure Internet of Things Revealed: Architecture and Fundamentals*. Berkeley, CA: Apress, 2019, pp. 73–85, ISBN: 978-1-4842-5470-7. DOI: 10.1007/978-1-4842-5470-7\_4. [Online]. Available: [https://doi.org/10.1007/978-1-4842-5470-7\\_4](https://doi.org/10.1007/978-1-4842-5470-7_4).
- [43] J. Soh, M. Copeland, A. Puca, and M. Harris, “Overview of Azure Infrastructure as a Service (IaaS) Services,” in *Microsoft Azure: Planning, Deploying, and Managing the Cloud*. Berkeley, CA: Apress, 2020, pp. 21–41, ISBN: 978-1-4842-5958-0. DOI: 10.1007/978-1-4842-5958-0\_2. [Online]. Available: [https://doi.org/10.1007/978-1-4842-5958-0\\_2](https://doi.org/10.1007/978-1-4842-5958-0_2).
- [44] —, “Azure Storage,” in *Microsoft Azure: Planning, Deploying, and Managing the Cloud*. Berkeley, CA: Apress, 2020, pp. 271–291, ISBN: 978-1-4842-5958-0. DOI: 10.1007/978-1-4842-5958-0\_14. [Online]. Available: [https://doi.org/10.1007/978-1-4842-5958-0\\_14](https://doi.org/10.1007/978-1-4842-5958-0_14).
- [45] Microsoft Corporation, *IoT-Analysen mit Azure Data Explorer*, <https://learn.microsoft.com/en-us/azure/architecture/solution-ideas/articles/iot-azure-data-explorer>. (visited on 05/02/2023).

- [46] B. Nuseibeh, “Weaving the software development process between requirements and architecture,” *From Software Requirements to Architectures (STRAW’01)*, 2001.
- [47] S. Bühne and A. Herrmann, “Handbuch Requirements Management nach IREB Standard,” *Aus-und Weiterbildung zum IREB Certified Professional for Requirements Engineering Advanced Level “Requirements Management”*. IREB eV, 2015.
- [48] C. Minonne, “Structured Requirements Elicitation According to ©iSREM,” eng, in *Digital Business Engineering : Going Beyond Business Models and Getting Down to Digital Business Processes*, 1st ed. Zuerich: vdf Hochschulverlag, 2022, pp. 161–173, ISBN: 9783728140777.
- [49] Official ROS Wiki, *ROS Tutorials*, <https://wiki.ros.org/ROS/Tutorials>. (visited on 02/06/2023).
- [50] The Construct, *Linux for Robotics*, <https://app.theconstructsim.com/Course/40/>. (visited on 02/22/2023).
- [51] —, *Developing Web Interfaces for ROS*, [https://app.theconstructsim.com/Course/42](https://app.theconstructsim.com/Course/42/). (visited on 03/10/2023).
- [52] Python Software Foundation, *HTTP servers*, <https://docs.python.org/3/library/http.server.html>, 2001–2023. (visited on 02/18/2023).

## Anhang

### Fragebogen

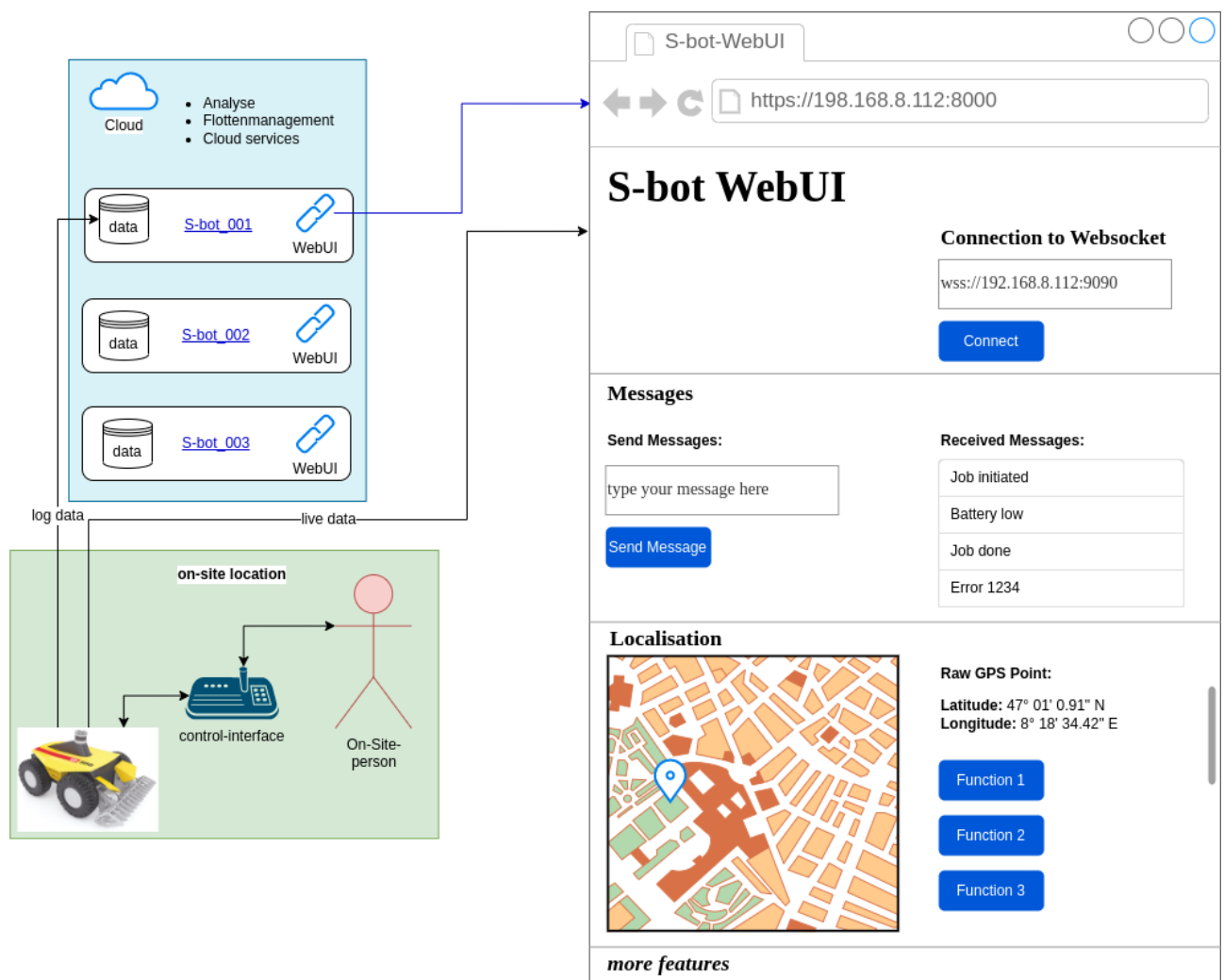
## Evaluation webbasiertes User Interface Vegetationskontrollroboter

Das vorliegende Dokument soll die Funktionen eines benutzerfreundlichem, webbasierten User Interface evaluieren.

Dabei gilt zu sagen, dass diese Evaluation Teil einer Bachelorarbeit ist. Diese leistet einen unterstützenden Beitrag zum Projekt "Roboterbasierte Vegetationskontrolle RoVeKo", aber sie ist nicht ein Teil davon.

Weiter gilt die Evaluation lediglich für den Teil des WebUI. Angedacht ist die Dreiteilung wie in folgender Abbildung dargestellt.

- **Cloud:** Analyse der Daten, Flottenmanagement und weitere Services. Welcher Cloudanbieter und die weiteren Services sind noch undefiniert.
- **On-site location:** Steuerung und Bedienung des S-bots vor Ort. Ein eigenes Interface zur Steuerung und Kontrolle des S-bots.
- **S-bot-WebUI:** Die Daten stammen im Moment live vom Roboter. Überwachung des S-bots. Es geht in dieser Evaluation nur um die Funktionen und Features innerhalb dieser Entität.



Die Wichtigkeit der gewünschten Anforderungen ist auch nach deren Priorität zu Ordnen. Gedacht ist, dass wichtigere Features zuerst entwickelt werden. Durch das Ausfüllen des Fragebogens soll im Anschluss eine priorisierte Anforderungsliste zur iterativen Weiterentwicklung des WebUIs entstehen.

Die Fragen sind unterteilt in Visualisierungen, Befehlsfunktionen und weitere Features.

**Fragen zu den Anforderungen an ein WebUI des S-bots**

1. Welche der folgenden Visualisierungen / Anzeigen sind im WebUI des S-bot Ihrer Meinung nach wichtig?

*Bewertung von 1 – 10 (1 = gar nicht relevant, 10 = absolutes Muss)*

Anzeige des Batterieladezustand

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Anzeige des aktuellen Energieverbrauch: Modus des Roboters / Schneider von 0 bis 100%

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Anzeige von Temperaturen: Aussen-, Motor-, Motherboardtemperatur)

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Aktueller Standort mit Anzeige auf der aktuellen Karte

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Zuletzt abgefahrene Punkte auf der Karte

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Asynchrone Bilder der integrierten Kamera

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Visualisierung von Hindernissen und Informationen dazu (durch Sensordaten von Lidar und Kamera)  
Künstlicher Horizont, Distanzinformationen zu Objekten / zu HOME Wegpunkt

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Aktueller Zustand: Mähen, Epilieren, Idle, Fahren, ...

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Möglichkeit zur Wiedergabe von Ton durch den S-bot gesteuert über das WebUI

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Anzeige einer Statusbar: aktuell ...% gemäht, nächster Vorgang: Laden

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Weitere Vorschläge zu Visualisierungen / Anzeigen :

---

2. Welche der folgenden Befehlsfunktionen sind im WebUI des S-Bots Ihrer Meinung nach wichtig?

*Bewertung von 1 – 10 (1 = gar nicht relevant, 10 = absolutes Muss)*

Senden von sofortigem Notstop (genauere Analyse nötig) / Reset / Reboot / Pause

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Senden von Karten der SBB / Aktualisieren der Kartendaten

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Senden von einem Ziel (Punkt auf der Karte) / Abfahren eines Fahrplans

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Senden von Kommandos wie Parken / Laden / Mähen / ...

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10

Kommunikation über Headset mit Sprachausgabe beim Roboter, um Anweisungen an Personal vor Ort zu geben

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	2	3	4	5	6	7	8	9	10



Warnlicht /Blitzlicht) oder Warnton anschalten

<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7	<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10
-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	--------------------------------

Weitere Vorschläge für Befehlsfunktionen:

---

### 3. weitere Features

*Bewertung von 1 – 10 (1 = gar nicht relevant, 10 = absolutes Muss)*

Anbindung des S-bots an die Cloud zur Analyse von Logdaten auf dem Cloud-Anbieter

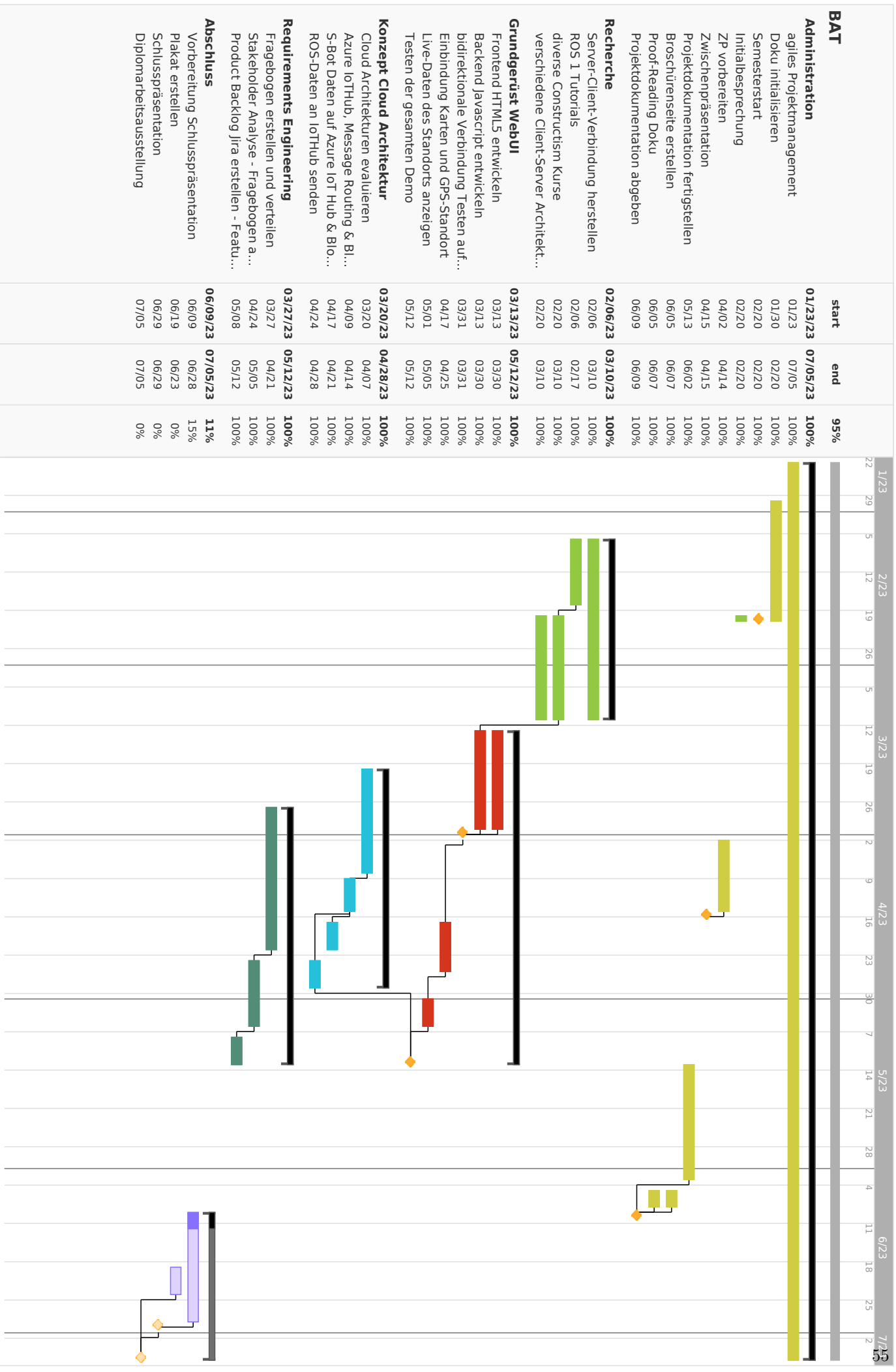
<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7	<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10
-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	--------------------------------

Vollständiges Hosting des WebUIs in der Cloud

<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4	<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7	<input type="checkbox"/> 8	<input type="checkbox"/> 9	<input type="checkbox"/> 10
-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	-------------------------------	--------------------------------

## **Gantt-Diagramm**

Auf der folgenden Seite ist der Gantt-Chart zu sehen, welcher laufend angepasst wurde.



**Bachelor-Thesis an der Hochschule Luzern - Technik & Architektur****Selbstständigkeits- und Redlichkeitserklärung**

<b>Titel</b>	<b>Roboterdaten auf dem S-bot visualisieren und Befehlsausführung mit WebUI</b>
<b>Vorname</b>	<b>Samuel</b>
<b>Nachname</b>	<b>Huser</b>
<b>Geburtsdatum</b>	<b>01.05.1991</b>
<b>Bachelor-Studiengang</b>	<b>Bachelor Digital Engineering</b>

**Selbstständigkeit**

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Hilfsmittel verwendet habe<sup>1</sup>.

**Redlichkeit**

- Sämtliche verwendeten Textausschnitte, Zitate oder Inhalte anderer Verfasser werden ausdrücklich als solche gekennzeichnet<sup>2</sup>.
- Diese Ausgabe der Bachelor-Thesis wurde von keinem Dozenten/keiner Dozentin nachbearbeitet.
- Die für die Portfolio-Datenbank abgegebene digitale Version ist textlich und im Layout mit der gedruckten Version (sofern eine solche abgegeben werden musste) identisch.
- Ich nehme zur Kenntnis, dass meine verfasste Arbeit mit elektronischen Hilfsmitteln auf Plagiate überprüft werden kann<sup>3</sup>.

Ort, Datum **Luzern, 09.06.2023**

Unterschrift Diplomandin/Diplomand \_\_\_\_\_



<sup>1</sup> Diesbezügliche Unredlichkeiten haben gem. Art. 39 und Art. 41 der Studienordnung der Hochschule Luzern Disziplinarmaßnahmen zur Folge.

<sup>2</sup> Zu den fremden zu deklarierenden und zu verifizierenden Quellen gehören auch mittels KI-Software wie ChatGPT generierte Texte bzw. Textteile (KI = «Künstliche Intelligenz»).

<sup>3</sup> inkl. Nutzung von Plagiatserkennungssoftware.