Thomas Endrè Ystenes, Alejandro Miguel Talley Grønhaug, Anders Marcelius Hofoss Frostrud, Elias Hanken Fiskerstrand

App/Web Application for Control of Omnidrive Robot

Bachelor's thesis in Computer Science
Supervisor: Ibrahim Hameed
Co-supervisor: Girts Strazdins
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU

ABSTRACT

As the use of automated machines in the logistics industry expands, the need for supervision of these machines increases. Solwr has recognized the growing demand for a mobile/web application to extend the Omnidrive Autonomous Guided Vehicle's (AGV) control capabilities to other platforms.

This bachelor thesis covers the process of the development of an application for manual control of a AGV. The purpose of the application is to control the AGV in the event that something goes wrong, and the automated system is not able solve the issue on its own.

The main product resulting from this project is a functional mobile application developed using React Native and Node.js. The application serves as a robot control system, allowing users to control and interact with the robot through their mobile devices.

While the developed mobile application is fully functional according to Solwr's requirements, further work is needed to make it production ready.

SAMMENDRAG

Etter hvert som bruken av autonome maskiner i logistikkbransjen øker, øker også behovet for oversyn av disse maskinene. Solwr har annerkjent den økende etterspørselen etter en mobil-/webapplikasjon som utvider kontrollfunksjonaliteten til Omnidrive Autonomous Guided Vehicle (AGV) til andre plattformer.

Denne rapporten dekker utviklingsprosessen for en applikasjon for manuell styring av en AGV. Formålet med denne applikasjonen er å styre AGV-en i tilfeller der noe går galt og det automatiserte systemet ikke er i stand til å løse problemet på egen hånd.

Hovedproduktet av dette prosjektet er en funksjonell mobilapplikasjon utviklet ved hjelp av React Native og Node.js. Applikasjonen fungerer som et robotkontrollsystem som lar brukeren styre og samhandle med roboten via sin smarttelefon.

Selv om mobilapplikasjonen er funksjonell i henhold til kravene Solwr stiller, er det behov for videre arbeid for å gjøre den klar for generell bruk.

PREFACE

This paper is our thesis report in the development and planning of a prototype web- and mobile phone application used to remotely control and manage AGV warehouse robots. As well as the methodologies we used. This report has been a collaborative effort between four computer science students at the ICT faculty of NTNU Ålesund.

But before we begin we would like to extend our heartfelt gratitude to the staff at Solwr for their support in the creation of our application solution. We would like to thank Sondre Rodahl, Computer Engineer at Solwr, he has been our point of contact between Solwr and its various business partners, as well as our main guide on this project. Yevhenii Petrichenko, also known as "Sprootkit", whose in-depth knowledge of the robot and resourcefulness has proved invaluable to us. Olivier Roulet-Dubonnet, project lead at Solwr, initial contact with NTNU and supervisor. Elise Hjemly, whose expertise in user friendly design and user testing was of great help to us during the planning stages. A warm thanks to all, we could not have done this without you.

The work in this report was commissioned by Solwr through NTNU's Bachelor program and assigned to bachelor group 2. Our work began in early January 2023 and has concluded as of May 2023. The application detailed herein is a tool for remotely controlling and reporting on the status of special purpose autonomous guided vehicles. We hope you find this paper to be informative and useful to your purposes.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ARP** Address Resolution Protocol
- **AGV** Autonomous Guided Vehicle
- **BLE** Bluetooth Low Energy
- **EDA** Exploratory Data Analysis
- **GNSS** Global Navigation Satellite System
- **Mamsl** Meters above mean sea level
- **NTNU** Norwegian University of Science and Technology
- **PCA** Principal Component Analysis

CHAPTER

ONE

INTRODUCTION

This chapter introduces the project this report is based upon. The project background, requirements, boundaries, and subject areas are all introduced to provide an understanding of the project and its value.

1 Background

The rapid advancements in the robotics and autonomous systems have led to the emergence of sophisticated mobile robots with the ability to perform various tasks in diverse environments. Solwr Robotics is developing a robot with the ability to move and store inventory in a grid-like system. To ensure error handling and the possibility to manually control a robot with the use of a mobile device, they aim to create a mobile phone application that enables the user to easily operate robots as needed.

At the current stage, Solwr has a working prototype mobile application where you can connect to a single robot and control it. While this is functional, being able to control and error handle multiple robots from the same interface would enable Solwr to act on anything happening in a system as it happens and respond in real-time, allowing Solwr to take immidiate actions based on the information the user receives. This real-time response capability can be especially valuable in situations where quick decision-making is necessary to prevent or mitigate negative consequences. By being able to act on events as they occur, Solwr can optimize processes, improve efficiency, and enhance overall performance in various systems and applications.

This project was requested by Solwr on behalf of their Robotics team, who are responsible for the creation of the AGV (Automated guided vehicle) robots. This project is a part of a larger initiative to develop and implement a comprehensive mobile application for the user to have a overview of all robots and its information at any time on a single device.

2 Specified Project Description

The goal of the project is to develop an application for controlling omnidrive (short for "omnidirectional drive") robots, which are robots that can drive in any direction on a horizontal plane. These robots are intended to be used for moving warehouse inventory.
The application should be able to effectively communicate with multiple robots, identify available robots on the network, and allow for seamless control over their movement. The application will be both web and mobile-native based, consistent with Solwr Robotics' user interface (UI) strategy, and will support cross platform availability between IOS and Android with the same code base.
The scope of this project encompasses the detection and driving of the robots, not the handling of any inventory, or the mechanisms by which this is achieved. Nor will it involve any robust use of login or other user authentication, other than connecting to the relevant network, with any security measures that this will require.

3 Problem Statement

Solwr Robotics is developing an autonomous mobile robot that requires manual control in certain situations. The current method of controlling the robot using a joystick connected to a single robot is not always practical. A mobile phone application would provide a more convenient and flexible option. This would include a visible view of all robots. This project aims to develop a web-based mobile phone application for controlling the robots, which must be easy to use and able to discover robots on the local network. The application will communicate with a backend server running on the robot PC, which will call a web API to control the robot. However, there are safety aspects to be considered, as the robot must stop if the connection is broken at any point.

The project will explore different methods for robot identification, such as Bluetooth, QR codes, and sonic signals, to determine the most effective approach to establish a connection between the application and the robot. The developed control system will utilize a third party's API to facilitate communication between the mobile app and the robot's backend server, which runs on Ubuntu Linux.

To ensure safety, the system will be designed to halt the robot's operations in case of any connection disruptions. The application will be developed using Javascript/Typescript for the frontend and Node JS for the backend.

4 Requirements

1. The application should provide a user-friendly interface for controlling the robot, which includes options for manual control and error handling.

2. The application should allow the user to discover multiple robots on the local network and switch between controlling them on a single interface.

3. The application should communicate with a backend server running on the robot PC using a web API to control the robot, with the ability to handle broken connections and stop the robot accordingly.

4. The application should be web-based and easy to use, with a mobile app written in the default system language(s).

5. The application should provide a real-time response capability, allowing Solwr to act on anything happening in the system as it happens and respond in real-time, enabling immediate actions based on the information the user receives.

6. The application should optimize processes, improve efficiency, and enhance overall performance in various systems and applications, contributing to the overall objective of developing and implementing a comprehensive mobile application for the user to have an overview of all robots and its information at any time on a single device.

7. The application should be developed iteratively, with the requirements and design evolving based on feedback from Solwr Robotics and end-users.

These requirements should guide the development and evaluation of the mobile phone application for controlling the autonomous mobile robot. The application should be evaluated based on its safety, effectiveness, and usability, with the goal of producing a solution that meets the needs of Solwr Robotics and its end-users.

5 Report structure

This report is divided into six chapters.

- Chapter 1: Introduction
- Chapter 2: Theory
- Chapter 3: Method
- Chapter 4: Results
- Chapter 5: Discussion
- Chapter 6: Conclusions

CHAPTER

TWO

THEORY

This chapter provides the essential background theory for the project, covering relevant methodologies, technologies, and tools utilized throughout. The aim is to equip the reader with the necessary theoretical knowledge to comprehend the tools and technologies discussed in the subsequent chapters.

1 Agile development methodology

Agile software development methodology is an iterative and flexible approach to creating software. It emphasizes collaboration, adaptability, and continuous improvement throughout the development process. Instead of following a rigid plan, agile teams work in short iterations called sprints, where they focus on delivering small, functional increments of the software. Regular feedback and communication with stakeholders play a crucial role in shaping the product's direction. Agile methods promote self-organizing teams that are empowered to make decisions and adjust their plans as needed. This iterative process allows for quicker responses to change and promotes transparency. By embracing agility, software development teams can foster innovation, increase efficiency, and deliver high-quality products that meet the needs of the users.[**Agile_methodology**]

2 Scrum

Scrum is a management framework and tool for the development of products and is highly popular in software development. It is primarily an agile method and emphasises iterative progress by using sprints. The goal is to solve complex problems or projects by iterating through a backlog that is composed of work, tasks and issues form the project that is to be completed. The framework itself consists of 3 concepts; roles, artifacts and meetings. All these concepts are combined to make Scrum a very effective framework that delivers continuous improvements, consistency and adaptability to new or changing requirements. [1]

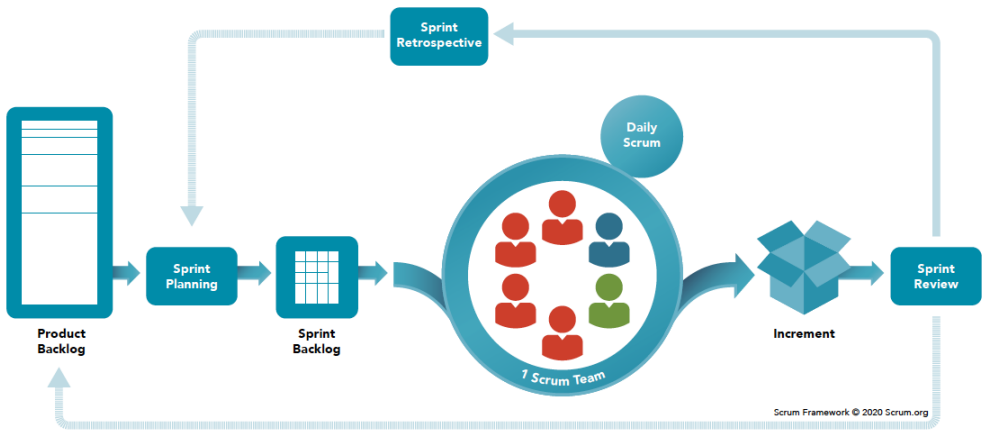


Figure 2.1: Illustration of a standard Scrum workflow

2.1 Scrum Roles

The Scrum framework contains 3 roles that is fundamental to the workflow: Scrum Master, Product owner and the developers.

2.1.1 Scrum master

The scrum master is the facilitator for the Scrum team. The Scrum master is also the one responsible for the implementation of the Scrum framework in the team and that the members are coached in the Scrum framework. Other responsibilities include making sure that Scrum events takes place, that the teams progress is not hindered by inside or outside events and that the overall goals and end-definition of the product is kept updated for the team. A scrum master is also the link between the team and the product owners and is responsible for the communication between the two parts regarding goals, requests, end definitions etc. The backlog is also managed by the master on behalf of the owner and must ensure that said backlog is organized in a way that facilities progress.[1]

2.1.2 Product owner

The role of the Product owner is to represent either the customers, stakeholders or in some cases a committee. This role exists to ensure that the product receives its maximum value from the Scrum framework. It differs from the Scrum master as the product owner is supposed to handle the business end of the framework and act as a liaison between the Scrum team and the invested partners. The ability to communicate and transmit priorities, status messages and changes between the two parts is vital. This role can be combined with Scrum Master, but this is highly advised against, as the product owner is expected to focus solely on the business end of the development.[1]

2.1.3 Developer

The last role is the developers. As the name suggests, they are the ones that carry out the work required for the sprints. More specifically, developers encompasses a wide range of expertise that is used in the project and do not specifically have to be software developers. Their responsibilities can also include work such as design, testing, researching, analysts etc. The role also includes responsibilities such as helping creating Sprint plans, adjusting and adapting said plans, attending Scrum meetings and also make sure that the work principles of the Scrum framework are implemented amongst the team. The team is expected to be self-organized but support of a fellow developer on a professional level is highly encouraged.[1]

2.2 Scrum Artifacts

The artifact concept revolves the documentation-keeping part and management of the Scrum framework. It consists of three parts. Product backlog, Sprint backlog and Increments.

2.2.1 Product Backlog

The Project Backlog is the source of the work that is expected to be done or improved on in the project by the developers in an ordered manner based on when it's needed to be completed. These issues can vary from implementation of features, bug fixes or other requirements depending on what the end-goal of the product is. These issues are also often based on user stories or use cases. When the team initiates a Sprint planning event, the chosen work for the sprint will be selected from the product backlog and is expected to be finished when the sprint is done. The backlog also contains a product goal which serves as the overall long-term objective for the Scrum team and the product.[1]

2.2.2 Sprint Backlog

The Sprint Backlog contains first and foremost the selected Product backlog issues chosen to the sprint and the overall goal of the sprint itself. This goal should be as clear and cohesive as possible on what the aim for the sprint is in order to promote focus for the team. It is also important to create a goal that is attainable and realistic. It is expected to function as a visible plan for the developers for duration the sprint and should ideally be filled with enough work based on earlier team performances from past sprints. The sprint backlog should also include enough detail that the team can easily track their own progress, and preferably be updated along the sprint as the work is being completed and expectations are adjusted.[1]

2.2.3 Increments

Increment is the last artifact of Scrum and is best described as the end-result of the work that is completed for a specific sprint. The sprint may also include several increments depending on the work environment. The increments are decided upon during a Sprint planning meeting and is usually presented at the end of a sprint during the Sprint review. When an increment is considered finished is also something the team must decide with what is called a "definition of done". This functions as a guideline with a set of criteria, such as when it passes all testing, if it has received feedback, zero defects or maybe a completion of a code review. What the definition encompasses is completely dependent on what the team has agreed upon.[1]

2.3 Events

The third part of Scrum is the concept of events. These events make up the planning and organizing of the scrum itself, while also making sure the team is continually making improvements and adaptations. The first of these events is the Sprint Planning. This is the event that initiates the Sprint by deciding on what work needs to be completed during its duration and is chosen from the Product Backlog. The meeting is attended both the Scrum team and the product owner. In this meeting, the team must decide on a Sprint goal and why the Sprint is valuable, and discuss what is an appropriate amount of work to be done during the increment. Planning of how the chosen what work is to be completed is also vital and is usually done by dissecting a backlog item into smaller amounts of work. The final topic of the meeting is the goal of the Sprint and when the work is to be considered done according to their "Definition of Done".[1]

2.3.1 Daily Scrum

After a Sprint has begun there will be a Daily scrum meeting for the developing team. This meeting is, as the name implies, a daily meeting where the main purpose is to do an inspection of the work done by the team, the progress made, and make any necessary adjustments. It's expected to take no more than 15 minutes and the developers are free to choose themselves how the meeting is conducted. If used effectively, a Daily scrum meeting can be vital for the project's development and a strong tool for making sure that the sprint progresses in a positive way and that issues that arises are handled in a good way.[1]

2.3.2 Sprint Review

At the end of a Sprint comes the Sprint Review event. During this event the team and the product owner discuss what work has been completed during the Sprint and what remains in the Product backlog. The developers also sometimes show a quick demonstration the work that has been done and answers any questions the product owner and shareholders might have. Other topics to discuss in the Sprint Review can also be how to proceed forwards regarding the backlog and subsequent Sprint, deadlines and delivery dates and reviews of timelines, budgets and if there are any changes to the potential usage of the product.[1]

2.3.3 Sprint Retrospective

The last event is a Sprint Retrospective which is an event where the team is encouraged to examine the last Sprint regarding the process, work, interactions and tools. In short, a Sprint Retrospective asks the questions; what worked well, what could be improved, what will the team do to improve in the next Sprint. If used correctly, it will help the team identify weaknesses in their workflow and address them while implementing steps and strategies to help improve the effectiveness of the team in the future.[1]

3 Robotics and automation

The robotics and automation industry is rapidly transforming the way warehouses and distribution centers operate. These technologies offer numerous benefits, some of which increases efficiency, reduces costs, and improves safety. Robots used in warehouses are typically designed to work closely with human workers, performing repetitive or physically demanding tasks such as moving heavy objects or transporting inventory across long distances.

By automating tasks like mentioned, robots would allow other workers to focus on higher-level tasks that require creativity and problem-solving skills. Not to mention that robots could work 24/7, increasing the overall efficiency of the warehouse and reducing the need for human workers to work overtime.

The use of robotics in an warehouse environment also presents several challenges, such as the need to ensure the safety of human workers, and the need to integrate robotic systems with existing warehouse infrastructures. As such, it is important to carefully evaluate the benefits and challenges of robotics and automation in warehouse environments to ensure a successful implementation.

4 Version control

Version control refers to the act of progressively tracking and managing the codebase and its changes. It allows developers to track changes in code over time and work on different features without it being irreversible. This also allows developers to simultaneously work on different parts of the code.

4.1 Git flow

Git Flow is a prominent branching mechanism used in software development projects for version management. It is intended to improve cooperation and code quality by grouping code changes into manageable branches. Git Flow has two major branches: "master" and "develop," with "master" containing production-ready code and "develop" containing on-going development work. It also makes use of a number of supporting branches, such as feature branches for new features or updates, release branches for preparing code for deployment, and hotfix branches for dealing with critical issues in production code. Git Flow can help speed the development process and enhance code quality by providing a clear framework for code changes and encouraging collaboration among team members.

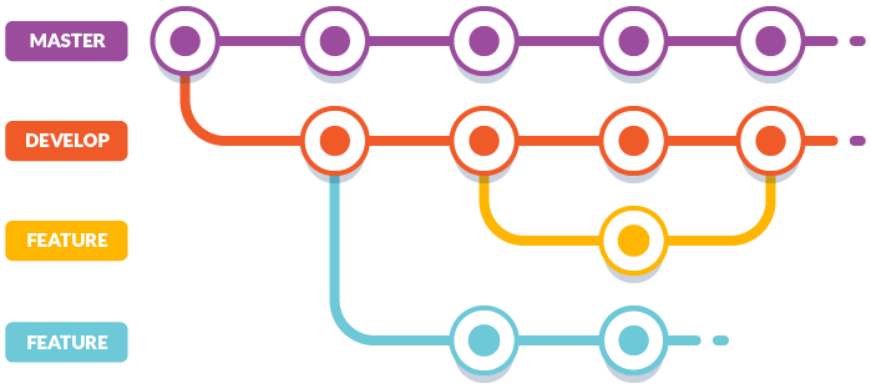


Figure 4.1: Illustration of Git flow

4.2 Pair Programming

Pair programming is a development method where two people work together in developing code and programming. This technique involves two people, where one is responsible for writing the code, while the other is observing the programming work, reviewing and commenting as the work progresses. This method has the benefit of improving code quality as the observer can review the code as it is being typed and give feedback for ideas and improvements and sharing valuable experience. The roles are also expected to switch frequently. While this technique increases man-hour used per line of code, the increased benefit of an observer often outweighs this cost.[2]

5 Design principles

Design principles refer to the fundamental concepts and guidelines that inform the design of a product or system. They are used to ensure that the design is user-centered, efficient, and effective, and to promote consistency and coherence in the final product. Some common design principles include user-centered design, responsive design, modularity, scalability, and security. By incorporating these principles into the design process, designers can create products that are intuitive, adaptable, and secure, and that provide a positive user experience. Design principles are an important consideration for any design project, including those related to software development, robotics, and other technologies.

5.1 User-Centered Design

User-centered design focuses on designing technologies that are intuitive and easy to use for the end user. By considering the user experience and making sure that the controls are user-friendly and easy to understand, designers can create products that are effective and efficient. User-Centered Design can be applied to a wide range of projects and technologies, from software development to product design and beyond. By incorporating this design principle into the design process,

designers can create products that meet the needs of their users and provide a positive user experience.

5.2 Responsive Design

Responsive Design is a design principle that involves creating technology that adapts to different devices and screen sizes. It ensures that the user interface is optimized for the device on which it will be used, providing a consistent user experience across different platforms. This design principle is important for ensuring that products are accessible and usable on a wide range of devices, from desktops to mobile devices.

5.3 Modularity

Modularity involves breaking down complex systems into smaller, more manageable components. Modularity offers several benefits, including:

1. Reusability: By dividing a software system into smaller modules, each module can be reused in different parts of the system or in different software systems altogether.

2. Maintainability: Modules can be developed and maintained independently, making it easier to fix bugs, add new features, or upgrade the system without affecting the entire software system.

3. Scalability: Modularity makes it easier to scale a software system by adding or removing modules as needed.

4. Flexibility: Modularity allows for greater flexibility in designing and building software systems, making it easier to adapt to changing business requirements or technological advancements.

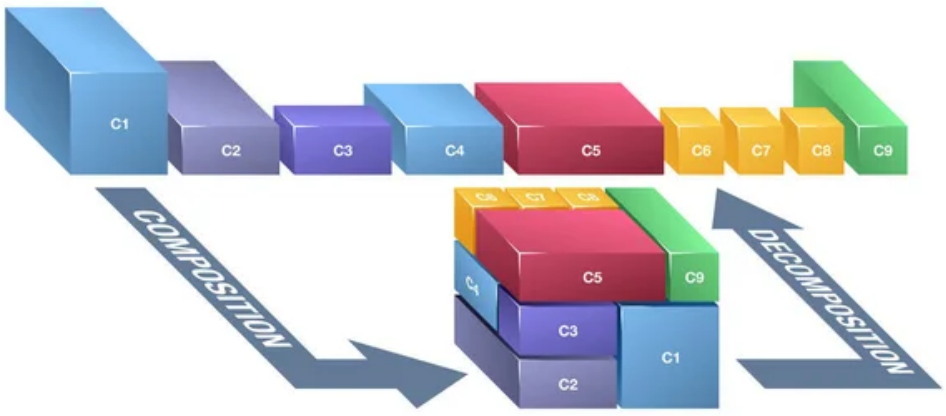


Figure 5.1: Illustration of modularity

6 React Native

React Native is a popular cross-platform framework for mobile application development. Introduced by Facebook in 2015, React Native facilitates the creation of high performance and visually consistent applications for both Android and iOS devices. This is achieved through the use of JavaScript and React, a popular front-end library for building user interfaces. The core advantage of React Native lies in its ability to generate native components for each platform, resulting in an enhanced user experience and improved performance compared to traditional hybrid app development. Furthermore, React Native promotes code reusability and modular architecture, significantly reducing development time and facilitating maintainability. A key concept in React Native is the Virtual DOM, which allows for efficient rendering of UI components by minimizing costly updates to the native view hierarchy. This mechanism, combined with the asynchronous bridge between JavaScript and native code, enables React Native to achieve near-native performance while retaining the flexibility of web-based development.[3]

6.1 React Hooks

React Hooks are a powerful feature introduced in React 16.8 that allow developers to manage state and side effects in functional components without writing class components. The two most commonly used hooks are **useState** and **useEffect**.[3]

The **useState** hook enables functional components to have their own local state, eliminating the need for class components. It takes an initial state value and returns the current state value and a function to update the state.[3]

The **useEffect** hook is used to perform side effects in functional components, such as fetching data from an API or subscribing to events. It runs after the component has rendered and can have a cleanup function for handling any necessary clean-up operations.

React also allows the creation of custom hooks, which encapsulate reusable logic and stateful behavior. Custom hooks provide a modular and reusable way to share logic between components.[3]

7 Node JS

Node.js is an open-source JavaScript runtime environment that allows developers to build server-side applications. It utilizes an event-driven, non-blocking I/O model, providing developers with a scalable and efficient platform for building high-performance applications. Node.js is supported by a vast community of developers who contribute to its extensive library of modules and packages, which can be easily integrated into projects. It also offers a consistent development experience by enabling developers to use JavaScript on both the front-end and back-end of web applications. Node.js is widely used in web development, particularly for creating real-time, data-intensive, and API server applications. Its popularity is due in part to its flexibility, speed, and ability to handle large amounts of data in real-time. As such, Node.js is a significant technology for developers to consider when building server-side applications.[4]

8 Expo

Expo is a comprehensive development platform that greatly simplifies the process of building cross-platform mobile applications using React Native. It provides developers with a range of tools, services, and libraries that streamline various aspects of the development workflow. With Expo, developers can quickly set up and configure their projects using Expo CLI, a command-line interface tool that handles project initialization and management. Expo also offers an extensive library of pre-built components and APIs, enabling developers to easily incorporate features such as navigation, user authentication, and push notifications into their apps. Moreover, Expo provides a seamless development experience by offering a live preview feature that allows developers to instantly see changes made to their code on physical devices or emulators.

9 Expo Go

Expo Go is an integral component of the Expo platform, designed to facilitate testing and previewing of React Native applications on real devices. Available for both Android and iOS platforms, Expo Go eliminates the need for complex setup procedures or emulators by providing a simple QR code-based method for loading and interacting with apps. Developers can scan the QR code generated by Expo CLI, which launches the app on their device through Expo Go. This enables rapid prototyping, debugging, and troubleshooting of React Native projects. Expo Go also supports over-the-air updates, allowing developers to distribute and update their apps without requiring users to download the latest version from app stores. This feature streamlines the deployment process and ensures that end-users have access to the most up-to-date version of the application.[5]

10 EAS Build

EAS Build is a cloud-based build service provided by **8**Expo, designed to simplify the process of building and generating production-ready app binaries for iOS and Android platforms. By leveraging the power of the cloud, EAS Build offers numerous benefits to developers:[6]

10.1 Simplified Build Process

EAS Build eliminates the need for complex local build environments and platform-specific build tool configurations. Developers can focus on writing code while EAS Build handles the compilation and packaging tasks.

10.2 Integration with Expo Framework

EAS Build seamlessly integrates with the Expo framework, a popular platform for developing cross-platform applications. This integration allows developers to take advantage of Expo's simplified development workflow and benefits, such as hot reloading and access to Expo APIs.

10.3 Easy Configuration

Developers can configure the build settings for their application, specifying the target platform (iOS or Android) and any additional build flags or options. This flexibility allows for customization and optimization of the build process.

10.4 Centralized and Scalable Build Infrastructure

EAS Build offers a centralized and scalable build infrastructure, ensuring consistent and reliable builds across different platforms. It eliminates the need for maintaining individual build environments, promoting collaboration within development teams.

10.5 Efficient Collaboration

The cloud-based nature of EAS Build facilitates efficient collaboration within development teams. By providing a centralized build infrastructure, team members can easily access and work on the project, streamlining the development process.

10.6 Improved Developer Productivity

EAS Build saves developers time and effort by simplifying the build process. It reduces the complexities associated with platform-specific build configurations, allowing developers to focus more on writing code and delivering high-quality applications.

10.7 App Deployment

After successfully building the app using EAS Build, developers can proceed with app deployment. For iOS devices, the app can be submitted to the App Store for review and distribution. Android devices can receive the app through various channels such as direct download, app stores, or enterprise deployment platforms.

10.8 Scalability and Resource Allocation

EAS Build's cloud-based infrastructure ensures scalability and optimal resource allocation. Developers can allocate build resources based on their needs, allowing for faster build times and efficient utilization of available resources.

11 Networking

Networking refers to the practice of connecting devices together in order to obtain communication and data transfer. This includes connecting computers, servers and other digital devices over a local area network (LAN) or wide area network (WAN), as well as connecting networks together to form lagers networks.

Address Resolution Protocol (ARP) and Multicast Domain System (mDNS) are two networking technologies whom are commonly used in local area networks (LANs) [7].

11.1 ARP

ARP (Advanced Resolution Protocol) is a networking protocol where its functionality is used to map a network address, such as an IP address, to a physical address, like a MAC address, on a local network. When a device on a local network wants to communicate with other devices, it needs some kind of physical address of the target device (MAC address) corresponding to the targets IP address.

ARP's main advantage over other protocols is its simple structure and efficiency resolving IP addresses to MAC addresses. One downside, however, is its lacking ability to access subnets. Devices connected to a different network or subnet cannot be discovered using ARP.

Another limitation of the ARP protocol is that it only works on IPv4 IP-addresses. In cases where IPv6 IP-addresses are being used, the Neighbour Discovery Protocol may be used instead [8].

11.2 mDNS

mDNS is a networking protocol for resolving domain names connected to IP addresses on a local network. It uses multicast DNS queries to discover devices on the network and obtain their IP address that way. mDNS allows devices to be discovered on a local network without the need for any kind of central DNS server. mDNS can work across different subnets and networks, allowing devices to be discovered and accessed from different parts of the network [9].

12 Socket communication

Socket communication is a networking technology that enables two separate processes running on different devices to establish a connection with the intent of communication with each other over a network. A socket is a software endpoint that represents a connection between two processes. The general functionality of this technology is based upon one process that creates a socket and binds it to a network address, while the other process connects to the socket to establish a communication channel.

Sockets can be used to implement various communication protocols, suchs as TCP (Transmission Control Protocol) or UDP (User Datagram Protocol). TCP provides reliable, connection-oriented communication between two processes, while UDP provides connectionless, unreliable communication.[10]

12.1 Bluetooth and RSSI

RSSI stands for Received Signal Strength Indicator. It is a measurement of the strength of a Bluetooth signal, measured in decibels (dBm) [11]. By measuring the RSSI, the group can calculate an estimate of the distance between then smartphone and the robot, using the following formula:

$$Distance = 10^{\frac{p-RSSI}{10*N}}$$

Where

- p is the factory measured RSSI at a range of one meter.
- RSSI is the currently measured RSSI.
- N is a constant depending on environmental factors that could cause interferance with the signal, ranging from 2-4.

Since the goal of this procedure would to simply determine if the user is relatively close to the robot, rather than to get an exact distance, the potential inaccuracy of this estimate is considered a to be acceptable. In this scenario, a Blueio USB dongle would be attached to the robot, to send the signal that was to be used to estimate the distance.[11]

13 Testing and Validation

Testing and validation are critical processes in software development that ensure the quality, reliability, and correctness of a software application. Testing involves systematically executing and evaluating the software components to identify defects, errors, or unexpected behaviors. It aims to verify that the application functions as intended and meets the specified requirements. Validation focuses on assessing whether the software meets the user's needs and expectations. It involves evaluating the application's performance, usability, and user satisfaction. By conducting thorough testing and validation, developers can identify and address issues early on, resulting in a more robust and user-friendly software product.[12]

13.1 Testing Methodology

Testing methodology refers to the systematic approach and techniques used to conduct software testing. It encompasses various testing techniques and practices employed to ensure the quality and reliability of a software application. In the context of this project, the testing methodology adopted involves a combination of unit testing, integration testing, and system testing.

13.1.1 Unit Testing

Unit testing is conducted to verify the functionality of individual components and modules in isolation. The React Native application is divided into various components and modules, each responsible for specific tasks. Unit tests are written using a testing framework such as Jest, which allows the execution of test cases and assertion of expected behavior.

The unit tests cover essential functionalities of components, ensuring they work as intended. They also help in identifying and fixing bugs at an early stage. Mocking and stubbing techniques are employed to isolate the units under test from their dependencies.

13.1.2 Integration Testing

Integration testing focuses on verifying the interactions and compatibility between different components and modules of the React Native application. It ensures that these components work together as expected and do not introduce any conflicts or issues when integrated.

Integration tests are written to simulate real-world scenarios and test the communication and coordination between the application's frontend (React Native) and backend (Node.js). These tests are usually automated and cover critical use cases and workflows.

13.1.3 System Testing

System testing is conducted to evaluate the complete application in its entirety. It verifies that all the individual components and modules work together harmoniously and satisfy the functional and non-functional requirements specified in the project.

System tests cover end-to-end scenarios, emulating real-world usage patterns of the application. This includes testing the user interface, interactions with the backend, and the overall performance and responsiveness of the system.

13.2 Validation

The validation process ensures that the developed application meets the intended objectives and requirements. It involves evaluating the application's functionality, performance, usability, and user satisfaction.

13.2.1 Functional Validation

Functional validation focuses on verifying that the application performs its intended functions correctly and accurately. It involves conducting various tests to ensure that the core features and functionalities, such as robot control, task management, and overview display, work as expected.

13.2.2 Performance Validation

Performance validation assesses the application's efficiency and responsiveness. It includes evaluating the application's speed, resource utilization, and scalability. Performance tests are conducted to measure response times, handling of concurrent users and tasks, and the application's ability to handle a large volume of data.

13.2.3 Usability Validation

Usability validation assesses how easily users can learn and use the application. It involves conducting usability tests with representative users to identify any usability issues, navigation challenges, or areas for improvement. User feedback and observations are collected to refine the user interface and overall user experience.

13.2.4 User Satisfaction Validation

User satisfaction validation aims to measure users' overall satisfaction with the application. This can be achieved through surveys, interviews, or feedback forms, where users provide their opinions and ratings regarding the application's usefulness, ease of use, and satisfaction with its features and functionalities.

CHAPTER

THREE

METHODS

The aim in this chapter is to provide a comprehensive overview of the approaches taken to tackle tasks at hand and to establish the credibility of the methodology employed. This chapter is divided into two parts, each addressing a specific aspect of the project.

The first part discusses the organization and planning of the project, research methodology and intermediate deliveries.

In the second part, the chapther will delve into the technical aspects of the project, encompassing the chosen development methodology, programming languages, tools, and external libraries used, as well as the configuration of the development environment. It will also detail any test setups established for evaluating the system in a manner consistent with its final implementation for the client.

1 Research method

In this project, a research-driven methodology was followed. This allowed the group to explore and learn about the new technologies that were unfamiliar to both Solwr and the bachelor team. Given the unfamiliarities, it was essential to add dedicated time for research, testing, and discovery, ensuring that the group could identify and implement the most effective solutions to solve the problem at hand. This approach not only expanded the groups knowledge and expertise in these unfamiliar areas, but also fostered innovation and creativity withing the group. By embracing a research-oriented mindset, the group were able to navigate the learning curve associated with new technologies while simultaneously optimizing problem-solving strategies, ultimately leading to a more robust and innovative final product.

2 Project Management

2.1 Scrum usage

The organizing and working approach the group took for the project was the Scrum method framework. As described in detail in chapter 2, Scrum allowed the group to make sure that progress project was made steadily and in a controllable manner. The regular duration of the group's sprints were 2-3 weeks. At the start of each sprint, the group convened in a sprint planning event where the issues were chosen based on what was relevant for the time schedule of the project. The backlog of issues was kept in a separate document with the groups estimated deadlines based on the project requirements. As said requirements were changed somewhat during the project and other ideas were tested out, the group updated the backlog document roughly every 2-3 weeks.

Among the other tools of the Scrum framework used were also the daily scrum. While the group did not necessarily meet every day, there were frequent meetings over Discord voice channels. The group also had daily communication between the group members via text channels in the Discord server. At the end of a sprint, there was a scheduled combined sprint review and retrospect which was used to gain overview and perspective of the current project situation, problems and outstanding issues. This helped greatly in planning the development of the project.

2.2 Scrum Roles

The roles used by the group for the project were Scrum master, Product owner and core developer. The scrum master was responsible for making sure that the Scrum framework was implemented and followed by the rest of the group, as well as making sure that the sprints were completed as smoothly and with as few problems as possible. They also had the responsibility of organizing the Sprint events, notifying the group of when said events takes place and making sure that the group had a clear "definition of Done".

Product owner was also used in the Scrum organizing of the group. This role has been accountable for the product backlog and making sure that it was updated

and structured properly. Product owner also had the task to make sure that the end-goal and vision of the project was followed and communicated to the rest of the group. Traditionally this role also has the responsibility of communicating with the stakeholders (which in this case was Solwr) and the rest of the group, but to make communication easy it was decided that group leader would act as the primary communicator to avoid complications or difficulties.

To make sure that each member got experience in both roles, the group had a rotation system were each role had an occupation duration of 1 month before switching with someone who had not had that role earlier. The rest of the time a member was defaulted to the Developer role. This role did not have any specific responsibilities or accountabilities but was still expected to commit to making sure each increment and sprint were planned, executed and completed as well as making sure the group had a positive cooperation and mood.

2.3 Extended Pair Programming

A common technique the group used in the project was the practice of pair programming. With the use of Discord, this was easily achievable as Discord allows the sharing of one's computer screen to the rest of the participants in a voice-chatroom. Thanks to this it was relatively easy to program together and give assistance where it was needed. Although pair programming is usually done in pairs, it was usually extended to be done with 3-4 members at a time if the workload allowed for it. It also allowed for the group to be actively involved in every part of the programming aspect of the project and increasing problem-solving.

2.4 Frameworks and Framework Research

The selection of appropriate frameworks is crucial to the success of any software development project. In the group case, thorough research was conducted to identify the most suitable frameworks for the React Native application and Node.js backend. This section will discuss the frameworks that was considered, the rationale behind the choices made, and the benefits they offered to the project.

2.5 Considered Frameworks

During the initial research phase, several frameworks were considered to build the application, including:

- **Flutter:** A popular open-source UI toolkit developed by Google, Flutter enables the creation of natively compiled applications for mobile, web, and desktop platforms from a single codebase. the group had experience with Flutter from previous projects, which made it a strong contender.[13]

- **React Native:** Developed by Facebook, React Native is an open-source framework that allows developers to build native mobile applications using JavaScript and React. It facilitates the development of cross-platform applications for both Android and iOS with a shared codebase.

- **Xamarin:** A cross-platform app development framework created by Microsoft, Xamarin enables developers to build native mobile applications using C# and the .NET framework.[14]

2.6 Selected Frameworks and Rationale

After careful consideration, it was decided to proceed with React Native for the application and a Node.js backend. The following factors influenced the decision:

- **Future Development:** Solwr, the company the app is developed for, expressed their interest in further expanding the app as a web application using React JS. Given the compatibility between React Native and React JS, this made React Native a more suitable choice.
- **Code Reusability:** React Native allows for a high degree of code reusability between Android and iOS platforms, which would save development time and resources. Additionally, some of the components could be shared with the future React JS web application, further streamlining development.
- **Performance:** React Native offers near-native performance by leveraging native components for rendering. This results in a smoother user experience compared to hybrid frameworks that rely on WebView for rendering.
- **Community Support and Ecosystem:** React Native boasts a large and active community, which ensures a wealth of resources, third-party libraries, and ongoing support. This can be crucial for resolving issues and implementing cutting-edge features.

2.7 Chosen Backend Framework

For the backend, Node.js was chosen, an open-source, cross-platform runtime environment that allows for the execution of JavaScript code on the server-side. This decision was influenced by:

- **Language Consistency:** Using Node.js for the backend enabled the group to maintain consistency in programming languages, as both the frontend and backend were developed using JavaScript. This simplified the development process and facilitated better collaboration among team members.
- **Scalability:** Node.js is known for its ability to support high levels of concurrency, making it suitable for scalable applications with high performance requirements.
- **Extensive Package Ecosystem:** The Node.js ecosystem includes a vast number of packages and libraries available through the Node Package Manager (NPM), simplifying the integration of various features and functionality.

By conducting thorough research on various frameworks and selecting React Native and Node.js for the project, ensuring a seamless development process while also accommodating Solwr's future development plans for a web application using React JS.

3 Technologies and Developmental Methodologies

3.1 JavaScript

The chosen frameworks of **6**React Native and **7**Node.js allows the group to focus more on development than learning a new language. Since both frameworks use the programming language JavaScript, this was the main reason for it being chosen.

In the context of developing a cross-platform mobile application for operating robots, JavaScript stands out as a technically suitable choice due to its versatility and widespread adoption. The use of React Native gives the team a big reduction in development time and resource overhead by allowing code reuse across multiple platforms, such as iOS, Android and Web. Javascript's asynchronous and event-driven nature made it well-suited to handle the concurrent operations and real-time communication required in the system.

Node.js as a server-sided technology leverages the advantages of JavaScript's single-threaded, non-blocking architecture, allowing one to establish multiple client connections and rapid communications with the robots.

Javascript in the project, both on the client and server side, allowed the team to maintain a consistent and unified codebase, streamlining the development process and simplifying further development and maintenance.

3.2 Code Style

As a team the group mostly followed Google's style guide [15] as a formal code style guide. The style guide details how the members of the development team are expected to formulate their code. While good and maintainable code is achievable without a general code style guide, uniformity is achieved when there is a common guide for the developers to follow.

To help enforce the style guide, the group used the Prettier extension for VS Code. Prettier is a code formatting tool that automatically formats code according to a set of rules. These rules were based on the style guide, which was designed to reflect best practices in code organization, naming conventions, and formatting.

3.3 Collaboration

As a team of four, the members locations were spread across Aalesund, meaning there was a need to figure out ways of collaboration and effective communication. After some discussion both internally in the bachelor group and the Solwr team, a decision was made to use a combination of online tools, such as Git and Github for version control, Solwr's internal video meeting rooms in the cloud for video conferencing and meetings with stakeholders and advisors, and Discord for day-to-day communication between team members and the developers on the Solwr team.

3.4 Version Control and Git

3.4.1 Git

In the project, Git was utilized as the primary version control system. Git, being a distributed version control tool, played a crucial role in enabling seamless collaboration within the team. It allowed each team member to work on the project and make changes without affecting the main branch. By ensuring that each Git directory on every client acted as a standalone repository, the group was able to independently manage work work. Additionally, Git provides a code review mechanism, enabling team members to review each other's changes and propose improvements.[16]

3.4.2 GitHub

Recognizing the significance of collaboration and version control in the robot control application, an early decision was made to leverage Git and GitHub in the work. By using these widely recognized industry-standard tools, members were able to effectively track and manage changes in the codebase. GitHub, in particular, offered a convenient features for project management, allowing the group to maintain a centralized repository and easily collaborate as a team.[17]

3.5 Discord

Discord was the chosen communication platform for the team and was exclusively used throughout the project. It's a social platform used for voice communication and instant messaging, where users can create their own persistent channels/chat rooms with restricted access. It also allows limited file-sharing which the team used in some instances to share resources. Direct sharing of a screen in the voice channels was also used to give assistance if there was any problems.[18]

3.6 Visual Studio Code

The main code editor for the project was Visual Studio Code. This is a source-code editor made by Microsoft with built-in support for debugging, syntax highlighting, code completion and version control. It can be used with a wide variety of development languages and offers further support by extensions accessed on a central repository.[19]

3.7 Postman

Postman is a program for building and using APIs. It provides an interface that allows the user to write requests, and send them to the desired endpoint, and view any resulting answers from the target API. Postman also provides organizational tools, such as collections to store requests for easy access in the future, and a history of previously sent requests.[20]

4 Libraries used

4.1 JEST

JEST was the chosen testing framework used in the project. It is based on JavaScript and built upon Jasmine and maintained by Meta. It's lightweight and easy to use with functionality covering matchers for testing expected values and thrown errors, easy methods for setups and tear-downs, snapshot testing and support or testing asynchronous code. It also has mock functionality which can be utilised to test only functions from a component or a hook and built in support to collect code coverage information.[21]

4.2 Socket.IO

Socket.IO is a library that is used to enable low-latency, bidirectional and event-based communication between a client and a server. It is built as an extra abstract layer over WebSocket and adds more functions that guarantee features like fallback to HTTP long-polling or automatic connection. In the project, it was used in the mDNS scan to connect to the robots through sockets. Allowing the group to create custom hooks in the React Native frontend to seamlessly communicate with the server-side processes, and ultimately the robots.[22]

4.2.1 Server-side Implementation

To implement Socket.IO in the project, the group first installed the socket.io-client package, which allowed for easy creation and management of socket connections. The package was integrated into the React Native application, allowing initiation of a connection to the server by creating a new socket instance:

```
import { io } from "socket.io-client";
const socket = io("http://server-address:port");
```

The group then set up listeners for various events using the 'on' method. For example, a listener was added for connection, disconnection, and robot discovery:

```
socket.on("connect", () => {
  console.log("Connected to server");
});

socket.on("disconnect", () => {
  console.log("Disconnected from server");
});

socket.on("robot-discovered", (data) => {
  console.log("Robot data received:", data);
});
```

On the server-side, the socket.io package was used to create a server that listens for incoming connections from the client. Once a connection is established, the server can handle incoming events and emit events to the client:

```
const io = require("socket.io")(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
    allowedHeaders: ["my-custom-header"],
    credentials: true,
  },
});


io.on("connection", (socket) => {
  console.log("Client connected:", socket.id);

  socket.on("disconnect", () => {
    console.log("Client disconnected:", socket.id);
  });

  socket.on("start-scan", async () => {
    console.log("Received start-lan-scan event");
    console.log("Starting LAN scan");
    // Handle LAN scanning and send updates to client
  });

  socket.on("stop-scan", () => {
    console.log("Stopping scan...");
    // Handle stopping LAN scan
  });
});
```

4.2.2 Client-side Implementation

On the client-side, the group implemented a custom React hook called ‘useRobots‘ to manage the socket connection, robot discovery, and state updates. This hook is responsible for creating the Socket.IO instance, handling the socket events, and updating the application state accordingly.

To create the socket instance, the group first imported the ‘socket.io-client‘ package and used the ‘io‘ function to establish a connection to the server:

```
import io from "socket.io-client";
const socket = io("http://server-address:port");
```

The group then used the ‘useEffect‘ hook to set up event listeners for the socket connection, which included listeners for connection, disconnection, robot discovery, robot updates, locations discovery, go-to-location status, and scan completion:

```
useEffect(() => {
  const newSocket = io("http://localhost:3000");
  setSocket(newSocket);
```

```
      newSocket.on("robot-discovered", (robotServices) => {
        //setScanStatus("discovering");
        setRobotDiscoveryFinished(false);
        robotServices.forEach((robot, index) => {
          setRobots((prevRobots) => [...prevRobots, robot]);
          if (index === robotServices.length - 1) {
            // If this is the last robot in the array, set the flag to indicate
            setRobotDiscoveryFinished(true);
          }
        });
      });

      newSocket.on("robot-updated", (updatedRobot) => {
        setRobots((prevRobots) => {
          const updatedRobots = prevRobots.map((robot) => {
            if (robot.agv_id === updatedRobot.agv_id) {
              return { ...robot, ...updatedRobot };
            }
            return robot;
          });
          dispatch(updateRobots(updatedRobots));
          return updatedRobots;
        });
      });

      newSocket.on("locations-discovered", (locations) => {
        setLocations(locations);
        dispatch(updateLocations(locations));
      });

      newSocket.on("scan-complete", () => {
        setScanStatus("idle");
      });

      newSocket.on("go-to-location-complete", (data) => {
        console.log("go-to-location-complete", data);
        setGoToLocationStatus({
          type: data.type ? data.type : "",
          text: data.text ? data.text : "",
        });
      });
}, []);
```

The ‘useRobots‘ hook also contains functions for starting and stopping the mDNS scan, which emit events to the server using the ‘socket.emit‘ method:

```
 const startMdnsScan = () => {
    if (socket) {
      setRobots([]);
```

```
      dispatch(updateRobots([]));

      setLocations([]);
      dispatch(updateLocations([]));

      setScanStatus("scanning");
      setSearching(true);
      socket.emit("start-scan");
      socket.emit("get-locations");
    }
  };

const stopMdnsScan = () => {
  if (socket) {
    socket.emit("stop-scan");
    setSearching(false);
    setScanStatus("idle");
  }
};
```

The hook returns an object containing the discovered robots, scanning status, and related functions, which can be used by other components in the application.

```
return {
    robots,
    locations,
    goToLocation,
    goToLocationStatus,
    startMdnsScan,
    stopMdnsScan,
    searching,
    scanStatus,
    socketConnected,
    error,
  };
```

The combination of client- and server-side implementation allows for seamless communication between the Node JS server and React Native frontend. This enables real-time updates during robot discovery and data exchange. By using a custom hook in the React frontend, a clean and modular code structure is maintained, making it easy to manage the socket connection and state updates. The server efficiently handles LAN scanning, robot data retrieval, and client requests, while the client can send commands and receive updates through the established socket connection.

4.2.3 Advantages

The use of Socket.IO in the project provided several advantages, such as:

- **Real-time communication**: Socket.IO enabled real-time, bidirectional communication between the client and the server, allowing the application to quickly react to changes in robot status or send commands instantly.
- **Event-based architecture**: The event-driven nature of Socket.IO allowed for easy management of complex interactions between the client and server, making the code more modular and maintainable.
- **Fallback mechanisms**: Socket.IO provided built-in fallback mechanisms for when WebSockets were not supported or available, ensuring that the application remained functional even in less-than-ideal network conditions.
- **Wide compatibility**: Socket.IO has support for various platforms, including React Native, which was crucial for the project's mobile application development.

The integration of Socket.IO into project improved efficiency and responsiveness of the robot control application, enabling seamless communication between the client and server, and ultimately the robots.

4.3 State Management with Redux and Redux Toolkit

In the React Native application, the group utilized Redux and Redux Toolkit for efficient state management. Redux is a predictable state container for JavaScript applications, providing a centralized store to manage the application's state. Redux Toolkit is a package that simplifies the process of working with Redux, reducing boilerplate code and enhancing developer productivity [23].

4.3.1 The Need for State Management

The application involved complex interactions between multiple components and data sources. A reliable and scalable way to handle the application's state to ensure data consistency and facilitate seamless communication between different parts of the application was needed.

By adopting Redux and Redux Toolkit, The group was able to maintain a single source of truth for the application's state, making it easier to manage and synchronize data across various components. Additionally, Redux's predictable state management tracks state changes and implements efficient UI updates, enhancing the overall user experience.

4.3.2 Redux Concepts

Redux operates on three fundamental concepts: actions, reducers, and the store.

Actions represent events or user interactions that trigger changes in the application's state. In the application, actions were dispatched from different components to initiate updates to the robot-related data, such as updating the robot IP address or retrieving the list of available robots.

Reducers specify how the state should change in response to dispatched actions. They are pure functions that take the current state and an action as input and return a new state. Redux Toolkit's `createSlice` function defines reducers in a concise and intuitive manner, reducing the amount of boilerplate code traditionally associated with Redux.

The store is the central hub that holds the application's state. It manages the state and provides methods for accessing and updating it. With Redux Toolkit's `configureStore` function, the group was able to create the store instance, configure middleware, and combine reducers into a root reducer effortlessly.

4.3.3 Store Configuration

To set up the Redux store, the `configureStore` funciton from Redux Toolkit was used. In the `store.js` file, this function was imported along with the `getDefaultMiddleware` function. The `persistStore` function from the `redux-persist` library was imported. The `persistConfig.js` file contained the configuration for persisting the Redux store.

The group combined the reducers using the `combineReducers` function and created a `persistedReducer` using the `persistReducer` function, which allows the state to be persisted across app restarts. The final store configuration looked as follows:

```
import { configureStore, getDefaultMiddleware } from "@reduxjs/toolkit";
```

```
import { persistStore } from "redux-persist";
import persistedReducer from "./persistConfig";

const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: false,
    }),
});

const persistor = persistStore(store);

export { store, persistor };
```

The `serializableCheck` option in the `getDefaultMiddleware` allowed the group to include non-serializable values in the state, such as functions or promises, without encountering errors during serialization.

4.3.4 Async Storage

To handle storage on different platforms, the the `AsyncStorage` library from `@react-native-async-storage/async-storage` for mobile platforms and `localStorage` for web platforms were utilized. A `storage` module that provided consistent `getItem`, `setItem`, and `removeItem` methods to interact with the appropriate storage based on the platform, where also implemented. The `isWeb` flag determined the current platform, and the corresponding storage method was used accordingly.

The `storage.js` file contained the following implementation:

```
import { Platform } from "react-native";
import AsyncStorage from "@react-native-async-storage/async-storage";

const isWeb = Platform.OS === "web";

const storage = {
  getItem: async (key) => {
    return isWeb ? localStorage.getItem(key) : AsyncStorage.getItem(key);
  },
  setItem: async (key, value) => {
    return isWeb
      ? localStorage.setItem(key, value)
      : AsyncStorage.setItem(key, value);
  },
  removeItem: async (key) => {
    return isWeb ? localStorage.removeItem(key) : AsyncStorage.removeItem(key);
  },
};

export default storage;
```

4.3.5 Slice Implementation

The `createSlice` function from Redux Toolkit was used to define the robot slice, which represented a specific portion of the applicaiotn state. The `robotSlice.js` file contained the definition of the `robotSlice` and the corresponding reducer functions.

The slice was initialized with the initial state, including properties such as `robotIP`, `robots`, and `locations`. Each reducer function defined in the slice modified the state in a predictable manner. For example, the `updateRobotIP` reducer updated the `robotIP` property, the `disconnectRobot` reducer cleared the `robotIP`, and so on.

The complete implementation of the `robotSlice.js` file was as follows:

```
import { createSlice } from "@reduxjs/toolkit";

export const robotSlice = createSlice({
  name: "robot",
  initialState: {
    robotIP: "",
    robots: [],
    locations: [],
  },
  reducers: {
    updateRobotIP: (state, action) => {
      state.robotIP = action.payload;
    },
    disconnectRobot: (state) => {
      state.robotIP = "";
    },
    updateRobots: (state, action) => {
      state.robots = action.payload;
    },
    clearRobots: (state) => {
      state.robots = [];
    },
    updateLocations: (state, action) => {
      state.locations = action.payload;
    },
  },
});

export const {
  updateRobotIP,
  disconnectRobot,
  updateRobots,
  clearRobots,
  updateLocations,
} = robotSlice.actions;
export default robotSlice.reducer;
```

These reducers were exported as named actions, dispatching these actions to update the state from different components.

4.3.6 Root Reducer and Persistence

The `combineReducers` function provided by Redux Toolkit was used to combine all the reducers into a root reducer. The root reducer, named `rootReducer`, combined the `robotSlice.reducer` from the robot slice with other slices if present.

The `persistReducer` function from `redux-persist` was then used to create a persisted reducer. The `persistConfig` object specified the key for persisting the root state and the storage module to use.

The final implementation of the `persistConfig.js` file was as follows:

```
import { persistReducer } from "redux-persist";
import { combineReducers } from "@reduxjs/toolkit";
import { robotSlice } from "./robotSlice";
import storage from "./storage";

const persistConfig = {
  key: "root",
  storage,
};

const rootReducer = combineReducers({
  robot: robotSlice.reducer,
});

const persistedReducer = persistReducer(persistConfig, rootReducer);

export default persistedReducer;
```

The `persistedReducer` was the result of applying the `persistConfig` to the `rootReducer`. This persisted reducer served as the main reducer for the Redux store, ensuring that the state persisted across app restarts.

5 Testing

Testing played a crucial role in ensuring the quality and reliability of the client-side React Native and server-side Node.js application. The group adopted a systematic approach to testing, including unit testing and integration testing, to verify the correctness and functionality of the software.13

5.0.1 Unit Testing

For unit testing, the focus was on testing individual components or functions in isolation to validate their behavior. The group utilized the Jest testing framework, a popular choice for React Native applications, to write and execute unit tests.

Jest provided a simple and intuitive API for writing test cases, facilitating testing of React components, functions, and utility modules.

Example:

```
import { sum } from '../src/utils';

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

In this example, a simple unit test is demonstrated using Jest. The group tested the `sum` function from the `utils` module, asserting that the sum of 1 and 2 should be equal to 3.

5.0.2 Integration Testing

For integration testing, the aim is to verify the interaction and integration of various components and modules within the application. The group created integration test suites using Jest and leveraged testing utilities specific to React Native, such as `@testing-library/react-native`, to simulate user interactions and test the application's behavior. **Example:**

```
import { render, fireEvent } from '@testing-library/react-native';
import LoginForm from '../src/components/LoginForm';

test('validates user login', () => {
  const { getByLabelText, getByText } = render(<LoginForm />);

  const emailInput = getByLabelText('Email');
  const passwordInput = getByLabelText('Password');
  const submitButton = getByText('Submit');

  fireEvent.changeText(emailInput, 'test@example.com');
  fireEvent.changeText(passwordInput, 'password');
  fireEvent.press(submitButton);

  expect(emailInput).toHaveProp('value', 'test@example.com');
  expect(passwordInput).toHaveProp('value', 'password');
  expect(submitButton).toBeDisabled();
});
```

In this example, the group demonstrate an integration test for the login form component. The group rendered the `LoginForm` component, simulated user input and button clicks, and then asserted that the input values were correctly set and the submit button was disabled.

5.0.3 Test Setup

To ensure consistent and reliable test execution, the group established appropriate test setups. This included configuring test environments, mocking external dependencies, and utilizing testing utilities tailored to React Native.

5.0.4 Continuous Integration

To automate the execution of tests, the group integrated the testing process into the continuous integration (CI) pipeline. This allowed the group to automatically run tests on each code change or pull request, ensuring the stability and correctness of the application.

6 Build and deployment

In the application development process, the group utilized Expo, EAS Build, and app store deployment techniques to ensure smooth distribution across platforms. This section describes how the group leveraged these methodologies for building and deploying the application on the App Store and Android devices.

6.1 Expo

The group took advantage of **8**Expo, a powerful toolchain and platform for developing, building, and deploying cross-platform applications. Expo simplified the workflow by allowing the group to use JavaScript and React Native, abstracting away the complexities associated with native code compilation.

6.2 EAS Build

To simplify the building process, the group made use of EAS Build, a cloud-based build service provided by Expo. With EAS Build, it was possible to generate production-ready app builds effortlessly and efficiently 10.

To initiate the EAS Build process, these steps were followed:

1. Ensured that the application code met all production requirements and was ready for deployment.

2. Configured the project's EAS Build settings, including specifying the target platform (iOS or Android), setting release channels, and defining any necessary build flags.

3. Triggered the build process using the Expo CLI command, providing the required credentials and configurations.

4. Monitored the build progress as a team, addressing any potential issues that arose during the process.

5. Received the built binary files promptly, which were ready for distribution and deployment.

By utilizing EAS Build, the group successfully optimized the build pipeline, ensuring consistent and reliable builds across different platforms.

6.3 App Store Deployment

For iOS devices, the group prepared the application for distribution on the App Store. While the application has not been published on the App Store yet, the group ensured it was ready for submission pending Apple's validation process [24].

Once the application build successfully passes Apple's validation, the distribution process on the App Store involves the following steps:

1. Generating an iOS distribution certificate and provisioning profile from the Apple Developer Portal.
2. Configuring the application's metadata, such as app name, description, screenshots, and keywords.
3. Creating an App Store Connect record for the application and linking it with the corresponding build generated through EAS Build.
4. Submitting the application for review and approval by Apple's App Store review team.

Upon approval, the group will be able to publish the application, allowing users to download and install it from the App Store.

6.4 Android Deployment

To deploy the application on Android devices, the group followed the standard Android deployment process[25]. This involved the following steps:

1. Generating an Android signing key, which included a private key and a corresponding certificate used to sign the application.
2. Configuring the application's build settings, such as target SDK version, minimum SDK version, and release signing configurations.
3. Building a release variant of the application using the APK (Android Package) format.
4. Distributing the generated APK file to users via various channels such as direct download, app stores, or enterprise deployment platforms.

By following these steps, the group made the application available for installation on Android devices, expanding the reach of the solution to a wider range of users.

While the group successfully generated an app store build using EAS Build, the actual publication of the application on the App Store and Android deployment may require additional steps and permissions, subject to the specific requirements and policies of the respective platforms.

CHAPTER
FOUR

RESULTS

This chapter contains the results of the project.

Scientific results covers findings made during the research for the project described in section 3.1. These have no implementation in the project, but rather represent knowledge gained and discoveries made.

Engineering results covers the tangible result of the project; the code written, and the product created. This also includes documentation of the code.

Administrative results covers the groups organization, and its use of developmental methodologies.

1 Administrative results

1.1 Results of Scrum use

The implementation of the Scrum methodology using the agile-based approach was successfully used by the group and greatly helped the development of the product. By focusing on the increments of the sprints, it was a lot easier to create a step-by-step product while simultaneously keeping a strict supervision of the development with the "definition of done" goal in mind. The values and work practice of the Scrum also helped greatly with problem-solving for the project and allowed great flexibility in how the development proceeded.

The roles assigned by the Scrum framework was also useful by making sure the team adhered to the work ethics and standards of the framework. With a dedicated(although monthly) Scrum master there was always someone who took the initiative and made sure the progress of the project and sprints were followed closely by the group. This fostered a more efficient collaboration and resulted in a much more comprehensive and cohesive development. The Product owner also contributed to this, as having an organized product backlog made choosing issues for future sprints much more convenient.

1.2 First iteration of the Back-end solution

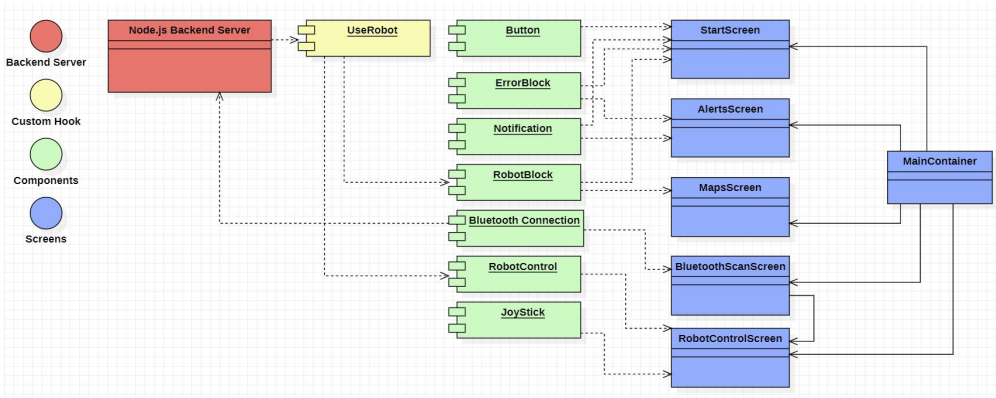


Figure 1.1: Illustration of the first iteration using the backend solution

1.3 Planning phase of the application

As the group had to adjust to the changes of not using Solwr's API's the planning for an application with an integrated backend to handle the communication was commenced. The design was similar to earlier planning, but instead of relying on API's, the communication with the robot was sent through the Node.js-based server. The code was split into several base screens which all were originating from the MainContainer. Each screen would handle their own part of the application with corresponding components. For example, the robot control screen would display the controls of the robot and use the Joystick and RobotControl components

to communicate and transmit the necessary information.

The useRobot custom **6.1**hook would handle the socket connection and work as a link to the backend server. The server was designed to take care of all the REST endpoint communication to the robots for the application. With this design, the goal was to make the code as modular as possible and allowing for flexibility and expansion in the future. The diagram is quite close to the end-result of the project, except for the Bluetooth connection, which was scrapped in favour of Geo-location.

2 Scientific results

Though many of the topics researched for this project did not end up being used in the final product, some potentially useful findings have still been made.

2.1 React Native as a development framework

Although Flutter was the framework initially considered for mobile development of the app, after some consideration it was decided that React Native should be used instead. As React Native has broader support and a higher degree of adoption.

2.2 Measuring Distance using Bluetooth

While the use of the Expo framework made implementation of Bluetooth proximity detection difficult, it could still be a viable solution for applications developed using other frameworks. When deciding to implement this solution, one should keep in mind that the formula used to estimate the distance the signal travels from sender to receiver requires the constant N to be calibrated to the specific environment that it is used in. This means that on-site testing will likely still be necessary for a good result.

2.3 Measuring distance using Sound

At the initial request of the project leader the viability of low frequency sound-based proximity detection was considered. It had several benefits, such as the physical range of sound limiting the availability of the AGVs from outside the warehouse and preventing outsiders from gaining access to the systems, as well as low frequency signals having longer range which can cover the whole warehouse.

3 Engineering results

3.1 AGV Detection

One of the main features of the created application is to find AGVs on the network the host device is running on. This is done by running an ARP-scan (a network scan using ARP). The MAC-addresses are then checked against a list of predefined MAC-addresses to find any IP-addresses belonging to an AGV. A get request is

then sent to the AGV, requesting information such as the name of the device, and to confirm that it is in fact an AGV. These devices are then listed in the applications "Robot list" view, with device name, battery status, current location, current destination (if any) and weather its sensors are clear or not. Each AGV listed also has a corresponding "Connect" button, that when pressed, sends the user to the "Robot Control" view. The following is an example of a get request to retrieve said data:

```
http://<IP-address>:7012/rpc/get_agv_data
```

An example of the response from such a request:


```
    {
"Result":{
"agv_id":"Navitec basement dweller",
"navitrol_state":9,
"state":"UnAvailable",
"loaded":false,
"battery_capacity":0,
"need_charging":false,
"errors":[
],
"warnings":[
{
"error_code":314,
"error_description":"WARNING_HIGH_LOGGING_RATE, Text log write rate is high and may a
},
{
"error_code":324,
"error_description":"WARNING_SAFETY_SCANNER_WARNING_ZONE2, Safety laser scanner warni
},
{
"error_code":601,
"error_description":"WARNING_AGV_LOW_BATTERY, AGV battery level low"
},
{
"error_code":603,
"error_description":"WARNING_ESTOP_ACTIVE, Emergency stop is active"
},
{
"error_code":607,
"error_description":"WARNING_SAFETY_SCANNER_OBSTACLE, Obstacle detected by one of saf
},
{
"error_code":611,
"error_description":"WARNING_AGV_NO_BATTERY_LEVEL, AGV battery level unknown"
}
],
"last_location":"C 1811",
```

```
"is_blocked":false,
"x_coordinate":-61.60258102416992,
"y_coordinate":27.502233505249023
}
}
```


3.2 Moving the AGV

The application is capable of sending requests to the AGV to have it move to a predefined destination, such as the charging station or to a specific shelf in the warehouse. The post request;

`http://<IP-address>:7012/rpc/go_to_location`

is sent, with a body containing the name of the desired destination, such as:

`[["Charging"], {}]`

This will make the AGV move to the charging station. The user of the app may select which destination they want the AGV to move to from a list of points generated from the data returned by the get request:

`http://<IP-address>:7012/rpc/list_location_names`

which returns a JSON object containing all the available destinations.

3.3 Direct Control using Joystick

The group was not able to implement functionality to directly move the robot using an in-app joystick. Doing this required the implementation of a new API by the AVG manufacturer, which they at the time of writing have no plan of doing. After some discussion, it was decided that the joystick user interface should remain in the application for potential future development by Solwr. This interface also contains buttons to switch between the AGV's different drive modes; front, mid, rear, pivoting and parallel, as well as a "Slow mode" switch, which divides the movement speed parameter sent to the AGV by ten for greater manual movement precision. This GUI can be viewed in the application's "Robot Control" view, which also shows the IP-address of the robot currently selected for manual control.

3.4 Local Map

The AGV has a built-in coordinate system. This system has origo set at a static point. A map of the warehouse was created and added to the app. This map was created on a scale of 1:500. Leveraging the preexisting coordinate system, a marker is added to the map, making it easy to locate the selected AGV.

The original image the map was based on contained a large amount of marked locations, making it difficult to read, such as a large amount of predefined AGV destinations. Because of this, and the fact that a more suitable image was not available, a new map was drawn. Since the original image did not declare a scale, a scale needed to be estimated in order to make the new map as accurate as possible.

Since origo was already marked on the original image, and the coordinates for the AGV charging station where known, the group was able to estimate the original image to be on a scale of 1:370. Knowing this, it was relatively simple to draw up a new map of the relevant area based on the original image. The map can be viewed in the "Map" section of the application.

3.5 Geolocation

The proximity detection solution used in the final product was geolocation. In this solution, a point was set near the anticipated operational area of the AGV. The application finds the geolocation of the smartphone hosting it, and calculates the distance to the preset point.

The host device's estimated distance to the preset point is shown in the "Map" section of the app. If the estimated distance is greater than 200 meters, the app is not able to control the AGV. The application gets the location of the host device using Expo's geolocation package expo-location. The distance between the device and the preset location is then calculated using the trigonometric harvesine function, as shown bellow.

```
    const distanceHaversine = (
  currentLat,
  currentLong,
  targetLat,
  targetLong
) => {
  const earthRadius = 6371; //Radius of the Earth, in KM.
  const distanceLat = degToRad(targetLat - currentLat); //Diff in latitude
  const distanceLong = degToRad(targetLong - currentLong); //Diff in longitude

  const a =
    Math.sin(distanceLat / 2) * Math.sin(distanceLat / 2) +
    Math.cos(degToRad(currentLat)) *
      Math.cos(degToRad(targetLat)) *
      Math.sin(distanceLong / 2) *
      Math.sin(distanceLong / 2);

  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  const d = earthRadius * c; //Distance in kilometers
  const dm = d * 1000; //Distance in meters

  return dm;
};

function degToRad(deg) {
  return deg * (Math.PI / 180);
}
```

3.6 Real-Time Alerts

Real-time alerts play a crucial role in the application, promptly notifying users whenever errors occur on an Automated Guided Vehicle (AGV). These alerts serve as a vital means of communication, ensuring that users stay informed even when they are not actively using the app.

Recognizing the importance of providing immediate updates, the group sought a reliable solution to notify users about AGV errors in real-time, even when they are not actively using the application. After careful consideration, It was decided that the project would use expo-notification library provided by **8**Expo, which offers a robust and user-friendly approach for implementing push notifications in the React Native application.

By leveraging the expo-notification library, the group successfully integrated push notification capabilities into the application. Whenever an AGV encounters an error, the app generates a push notification that includes relevant details such as the AGV's identification, the type of error, and any additional contextual information to aid users in understanding the issue.

Upon launching the application, users are immediately presented with a main screen that showcases AGVs currently experiencing errors. This intuitive interface enables users to quickly identify affected AGVs and take appropriate actions in response. Users have the ability to acknowledge the error directly from the app and can access real-time details about the error, allowing them to gain a deeper understanding of the issue at hand.

The introduction of real-time alerts significantly enhances the user experience by providing timely and relevant information about AGV errors. By implementing push notifications through the expo-notification library, the group successfully addressed the need to notify users even when they are not actively using the application. This proactive approach empowers users to take immediate action, ensuring the smooth operation of the AGV system and optimizing overall efficiency.

3.7 Final Robot Control solution

In the final solution, the group successfully developed a robust and user-friendly React Native application using **8**Expo Go and React Native for the frontend, complemented by **7**Node.js for the backend. Through the diligent implementation of custom React **6.1**hooks and socket connections, it was accomplished seamless communication between the frontend and backend components, enabling efficient control of omni drive robots.

One of the major achievements was the thoughtful design of the user interface (UI), which provided an intuitive and modern experience for users. By incorporating real-time feedback mechanisms, the group ensured that users received timely updates on robot status, sensor data, and connection status, thereby enhancing

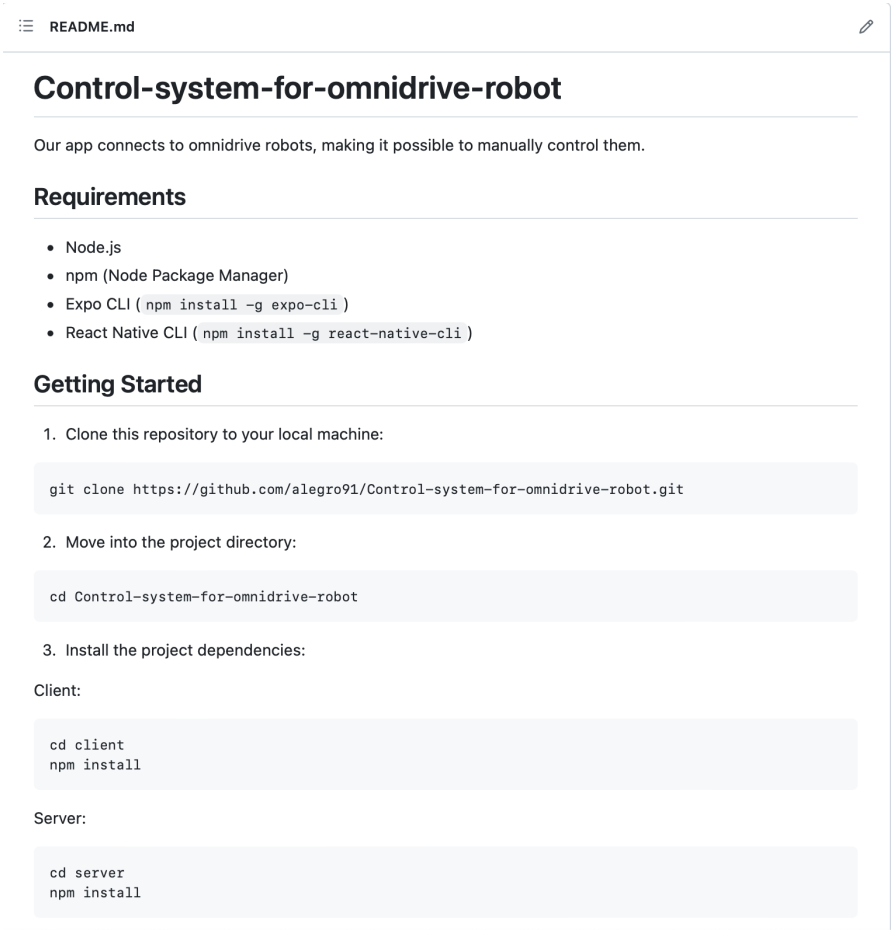
their overall control and monitoring capabilities.

In the frontend development phase, the group dedicated itself to crafting custom React **6.1**hooks that facilitated state management and streamlined user interactions. These hooks served as powerful abstractions, simplifying the complexities of socket connections and enabling seamless command transmission and real-time updates between the frontend and backend.

On the backend side, the group leveraged the capabilities of Node.js and employed Socket.io to establish reliable and bidirectional communication channels. The backend server acted as a bridge, facilitating the smooth transmission of commands from the frontend to the respective omni drive robots. Moreover, it efficiently relayed real-time sensor data and status updates back to the frontend.

3.8 Documentation

The project follows a well-organized component structure, where each component focuses on a specific task. This design approach allows for easy division of work and simplifies maintenance and further development. Additionally, the group have provided detailed comments throughout the codebase to ensure clarity and understanding. These comments serve as a helpful resource for other developers who may wish to contribute to the project, enabling them to navigate the codebase smoothly and make valuable additions.



README.md

Control-system-for-omnidrive-robot

Our app connects to omnidrive robots, making it possible to manually control them.

Requirements

- Node.js
- npm (Node Package Manager)
- Expo CLI (`npm install -g expo-cli`)
- React Native CLI (`npm install -g react-native-cli`)

Getting Started

1. Clone this repository to your local machine:

```
git clone https://github.com/alegro91/Control-system-for-omnidrive-robot.git
```

2. Move into the project directory:

```
cd Control-system-for-omnidrive-robot
```

3. Install the project dependencies:

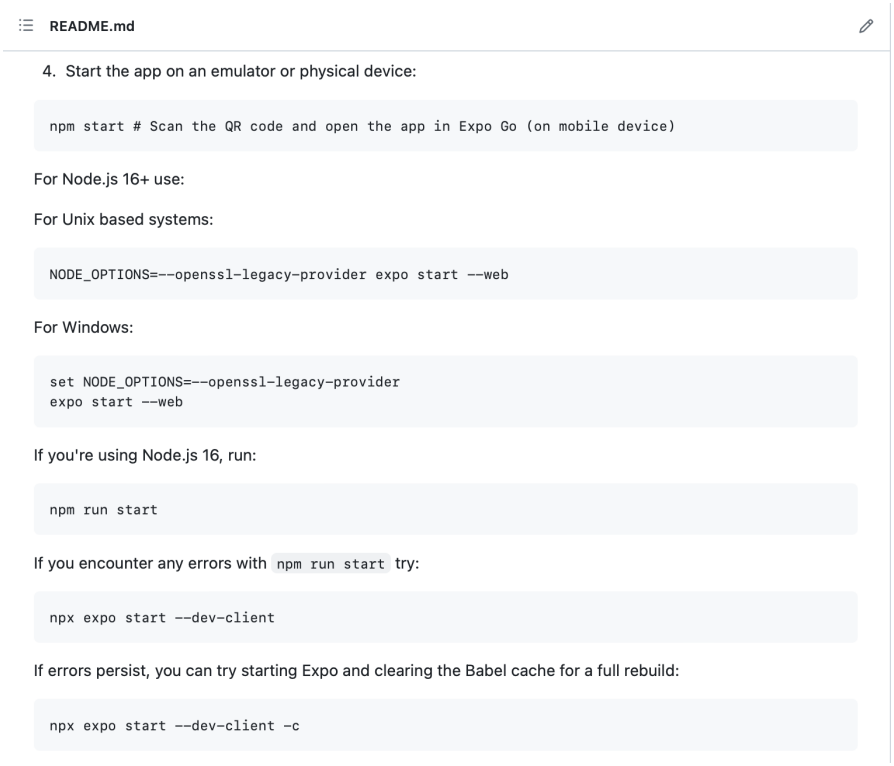
Client:

```
cd client
npm install
```

Server:

```
cd server
npm install
```

Figure 3.1: Screenshot of readme (pt.1) belonging to the application



README.md

4. Start the app on an emulator or physical device:

```
npm start # Scan the QR code and open the app in Expo Go (on mobile device)
```

For Node.js 16+ use:

For Unix based systems:

```
NODE_OPTIONS=--openssl-legacy-provider expo start --web
```

For Windows:

```
set NODE_OPTIONS=--openssl-legacy-provider
expo start --web
```

If you're using Node.js 16, run:

```
npm run start
```

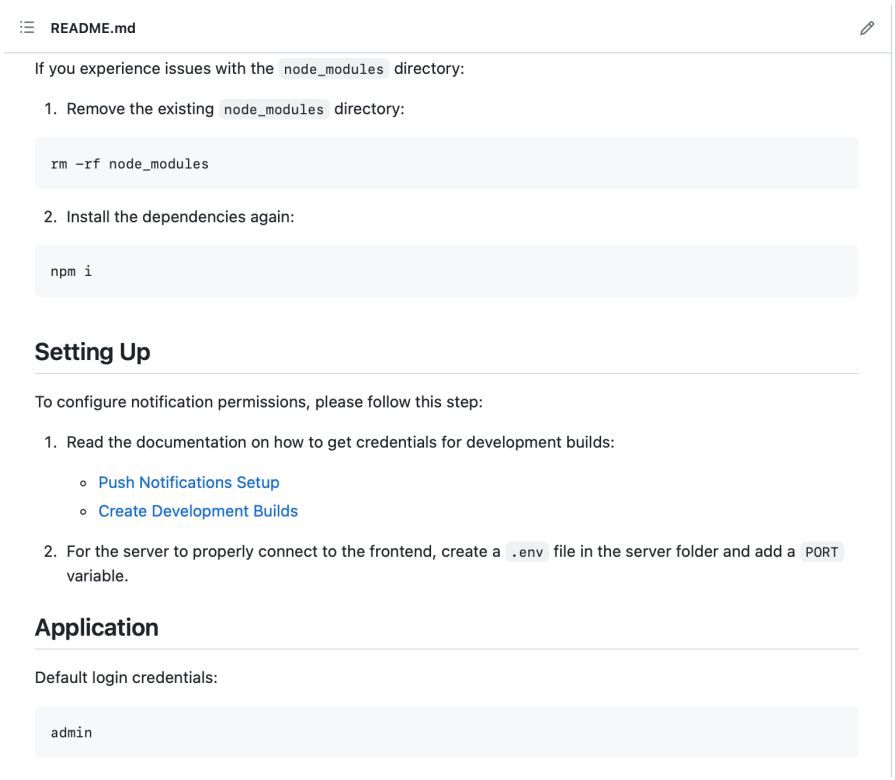
If you encounter any errors with `npm run start` try:

```
npx expo start --dev-client
```

If errors persist, you can try starting Expo and clearing the Babel cache for a full rebuild:

```
npx expo start --dev-client -c
```

Figure 3.2: Screenshot of readme (pt.2) belonging to the application


README.md

If you experience issues with the `node_modules` directory:

1. Remove the existing `node_modules` directory:

```
rm -rf node_modules
```

2. Install the dependencies again:

```
npm i
```

Setting Up

To configure notification permissions, please follow this step:

1. Read the documentation on how to get credentials for development builds:
   - Push Notifications Setup
   - Create Development Builds
2. For the server to properly connect to the frontend, create a `.env` file in the server folder and add a `PORT` variable.

Application

Default login credentials:

```
admin
```

Figure 3.3: Screenshot of readme (pt.3) belonging to the application

Additionally, the project has a comprehensive readme file available in the public GitHub repository. The readme file provides instructions in detail on how a developer can run the project in the correct environment.

CHAPTER

FIVE

DISCUSSION

1 Discussion

This chapter discusses the results covered in the Results chapter, as well as some potential solutions that where considered but not actually implemented in the project. The results of the project in relation to its requirements are explored to evaluate if they meet the expectations and requirements set forth by the client. The methodologies and process used during the project are also discussed here, in an attempt to uncover their strengths and weaknesses.

2 Administrative Discussion

2.1 First iteration and code logic changes

The first iteration of the project and the internal code structure was planned according to the internal APIs used and provided by Solwr which included the machine vendors API and the endpoints on the Grab computer on the AGV itself. As shown in the diagram the code structure was planned to access the endpoints of the GRAB IPC which then would give access to the machine vendors API where the application would retrieve the AGV interface through Port 8080 and give access to the UI control endpoint and interface for the robot. This would then be integrated into the application which would make it possible to manually control the robot. The first project diagrams show how the code structure was originally planned.

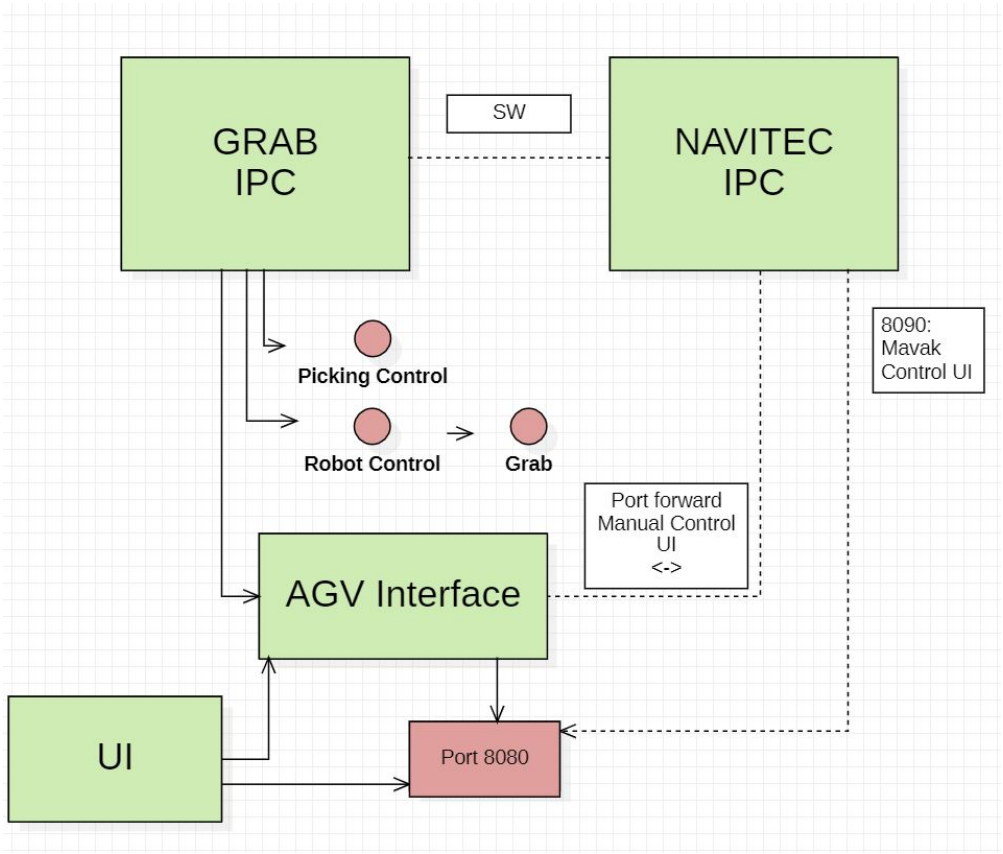


Figure 2.1: Diagram of the flow for the Solwr API's

Before the project was moved to a Node.js-based backend server to handle the POST requests, the code was originally planned to be split into several screens where each would make use of components that would take care of the requests sent to the API endpoints. The network scanning component took care of the POST requests for connecting to and updating the status of the robots themselves which would be displayed in the start screen. Manual mode would send direct requests to the AGV interface to make use of the UI that was planned for use in the application. As such, the original planned project would function quite differently from the integrated backend solution the group opted for instead. While the appearance and functionality of the application would have remained the same, the code structure, setup and logic would be very different.

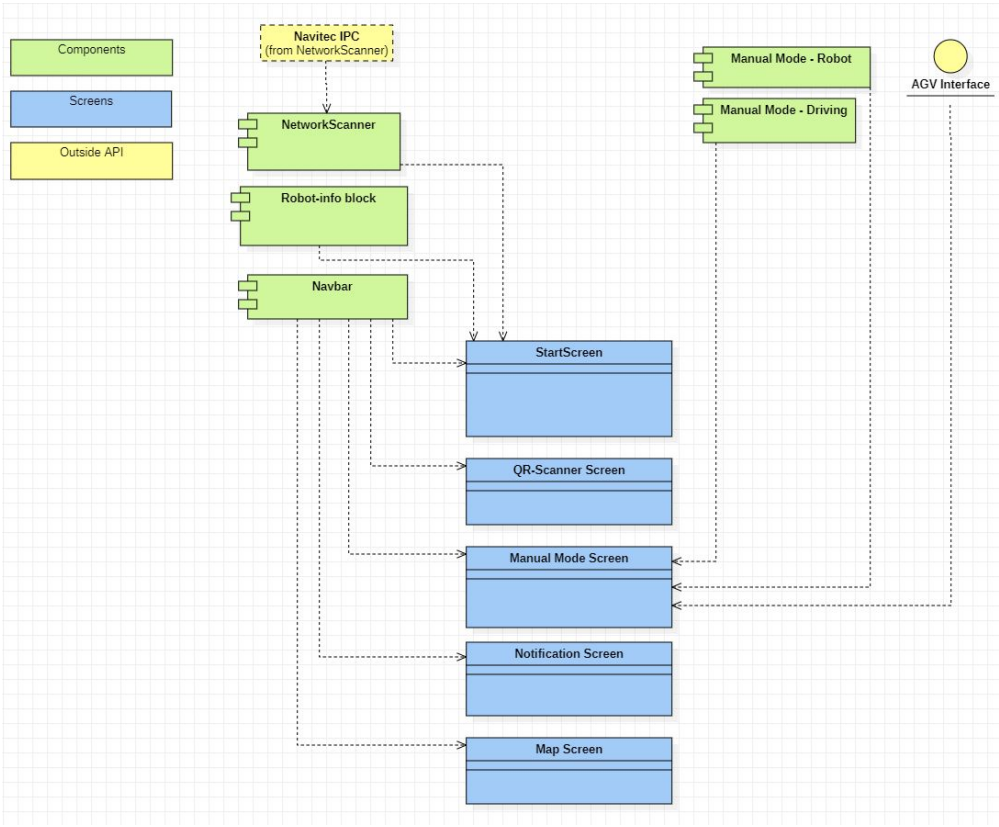


Figure 2.2: Diagram of the original planned code logic using API's

2.2 Scientific discussion

The scientific research conducted throughout this project has yielded valuable insights, despite not all findings being directly applied in the final product. The following discussions highlight the key findings related to the adoption of React Native as a development framework and the potential use of Bluetooth and sound-based proximity detection for distance measurement.

2.2.1 React Native as a Development Framework

The decision to utilize React Native as the development framework for the mobile application proved to be a strategic choice. While initially considering Flutter, the group ultimately opted for React Native due to its wider support and higher degree of adoption within the development community. This decision ensured that the application would benefit from a mature ecosystem, extensive resources, and a large community of developers who could contribute to its ongoing improvement and maintenance.

Moreover, one of the significant advantages of choosing React Native was its close relation to React, the popular JavaScript library for building user interfaces on the web. This relationship allowed the group to bridge the React Native application to be displayed in the same format on both web and mobile platforms. This not only provided a consistent user experience across different devices, but also opened up opportunities for further development and codebase reuse of React

Native components in React web applications. By leveraging the synergies between React and React Native, the group were able to maximize code reusability, and save time and effort in maintaining a consistent application interface across platforms.

2.3 Measuring Distance using Bluetooth

Exploring Bluetooth proximity detection within the Expo framework presented significant challenges, and it was discovered that it was difficult to implement and did not work as intended in the application. However, the group acknowledge that Bluetooth proximity detection can still be a viable solution for applications developed using other frameworks.

The group also identified a potential path for further development of Bluetooth functionality. To achieve better control and native compatibility, it would be necessary to develop custom native components in Expo using Swift for iOS and Java/Kotlin for Android. This approach would provide more flexibility and allow for a more fine-tuned implementation of Bluetooth proximity detection, enhancing the overall performance and accuracy of distance measurement.

2.3.1 Measuring Distance using Sound

In response to the project leader's request, the group investigated the viability of low-frequency sound-based proximity detection. This approach showcased several potential benefits, including limited access to Automated Guided Vehicles (AGVs) from outside the warehouse, enhancing security measures, and the longer range of low-frequency signals which can cover larger warehouse areas. However, practical implementation of this solution poses challenges that warrant further exploration. Factors such as ambient noise interference, signal attenuation, and reliable signal processing algorithms must be carefully addressed to overcome these challenges. The group recommends additional research and experimentation to assess the feasibility and effectiveness of sound-based proximity detection, particularly in industrial settings.

2.4 Engineering Results Discussion

2.4.1 Bluetooth - What Went Wrong

The primary challenge with this approach was that Expo Go was not compatible with JavaScript Bluetooth code libraries out of the box, and some extra configuration needs to be done in order to use the functionality needed by them. These configurations involved creating native builds for both the Android and iOS versions of the application. Despite numerous efforts, the group was not able to make the project run with these extra configurations, and as such, this solution was not used in the final product.

2.4.2 Triangulation using WLAN-routers as references

Another solution that was considered was using the signal from multiple WLAN-routers to triangulate the position of any robots on the network. This method

requires at least three WLAN-routers to be present in the network. Since there is no guarantee that this condition will be met wherever the robots might be deployed in the future, this method was not used in the end.

2.4.3 Sound-based Proximity Detection

A solution involving the use of sound signals to determine the proximity of an AGV was briefly considered. With this solution, the group would have the AGV generate a specific sound pattern, and have the application listen for that signal to determine if the AGV and the smartphone where close to each other. A similar solution was developed by Thiel, Klock and Lukowizc [26], using iPhone 4 smartphones in 2012.

This suggestion was ultimately discarded, as the AGV lacked the capability to produce the low-frequency sound signals used in sound-based proximity detection, and none of the technical specifications. As the project goal was to create an application for controlling the AGVs, the group decided to go for a solution that required a minimal amount of modification of the AGV.

CHAPTER

SIX

CONCLUSIONS

This chapter briefly discusses research and findings. Things that went well and things that didn't go as well. What could've been done differently if time had permitted, or if more research had been done

1 Planning

1.1 Organization of work

Planning and teamwork left some things to be desired. On more than one occasion two or more team members were working on the same task, an inefficient use of time.

1.2 Impractical solutions

Insufficient care was taken to the users' needs and what practical solutions could be made to satisfy these. An example is seen in the early implementation of QR-code scanning as a means of connecting to the robot. The intention was to manually connect to an AGV and assume control using information provided by a QR-code taped to it, the idea was good, but had flaws. Needing to be in close proximity of the Omnidrive robot to scan the code had its difficulties if the robot was in motion. In addition the solution was seen as tedious for the operators in charge of maintaining and overseeing the AGVs.

2 During development

2.1 Frameworks

React Native was chosen because it is more widely supported across platforms, as it renders components natively, as well as its ease of portability from mobile to web. In hindsight this choice proved to be a limitation during development, since the Expo framework used for developing apps in React Native did not support the planned Bluetooth functionality out of the box. For the Bluetooth functionality to work the app would have to be built natively for Android and iOS [27]. As this issue was discovered fairly late into the development, time did not permit for extensive troubleshooting and additional research.
The app could've been developed in Flutter, which does support Bluetooth out-of-the-box and could've cut development time significantly. Last minute modifications to implement geo-location as an alternative to Bluetooth signal strength estimation had to be made and as a result the final solution deviated somewhat from the initial idea.

2.2 Third party APIs

A third party vendor providing the control systems of the robot posed some challenges underway. In order to gain full manual control of the robot, to move it using the joystick on the app access to their API was needed. Upon request from Solwr the vendor replied saying that it would not disclose the API for its proprietary software systems. As such the application is not able to control the robot using the joystick, although the functionality has been implemented and the code produces the desired vectorized movement data, the application has no way of communicating with the robot to perform the movements. At present the AGV robot can only go to a pre-defined location.

3 Future work

3.1 Vendor issues

Addressing the issues with the robot control system third party vendor is crucial for implementing a custom tunnel to the robot. This would involve resolving any communication or compatibility issues to ensure smooth joystick movement and efficient data exchange with the robot. By successfully establishing this connection, the application's control and interaction capabilities can be further enhanced, opening up possibilities for advanced functionalities such as real-time control and navigation.

3.2 Standalone Web/Desktop Application

In terms of further development, one promising direction is to consider porting the existing React Native application to a standalone web application using React or a desktop application using Electron. Since the codebase is already written in JavaScript, this provides the opportunity to easily transfer components to other frameworks like React for web development and Electron for desktop applications. By pursuing this approach, the accessibility and usability of the application across different platforms can be extended, allowing users to access and utilize it on web browsers or as standalone desktop applications. This portability offers the advantage of codebase reuse, enabling efficient development by leveraging existing components and logic from the React Native application.

3.3 Authentication

Authentication plays a crucial role in ensuring user access and the security of the application. Currently, a static login mechanism that provides a basic level of user authentication is implemented. However, it is important to note that the current implementation does not include advanced security measures, such as biometric authentication or multi-factor authentication.

Moving forward, it is recommended to further enhance the authentication process to incorporate more secure login methods. This could include implementing industry-standard security protocols and best practices, such as hashing and salting user passwords, utilizing secure token-based authentication mechanisms, or integrating third-party authentication providers. By implementing these advanced authentication measures, the security and protection of user accounts and data within the application can be significantly improved.

3.4 Encryption

At this point secure tunnels for encrypting data transfer to the robot, or between the front-end and backend services have not implemented.

To enhance the security of the application and protect sensitive data, further work to implement encryption measures for data transfer is vital. This can be achieved by employing secure communication protocols, such as HTTPS or

Transport Layer Security (TLS), when transmitting data between the frontend and backend components. Additionally, for secure communication with the robot, implementing encryption protocols like Secure Socket Layer (SSL) or establishing a secure tunnel can be considered.

REFERENCES

[1] *Scrum*. Visited on April 24, 2023. URL: https://scrumguides.org/scrum-guide.html.

[2] *Pair Programming*. Visited on March 27, 2023. URL: https://www.agilealliance.org/glossary/pairing/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'pair*20programming))~searchTerm~'~sort~false~sortDirection~'asc~page~1).

[3] *React Native*. Visited on May 5, 2023. URL: https://reactnative.dev/docs.

[4] *Node.js*. Visited on March 27, 2023. URL: https://nodejs.org/.

[5] *Expo Go*. Visited on April 12, 2023. URL: https://docs.expo.dev/get-started/expo-go/.

[6] *Expo Go EAS*. Visited on May 10, 2023. URL: https://expo.dev/eas.

[7] *Network Programming*. Visited on March 25, 2023. URL: https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-programming.html.

[8] Mariusz Zydyk. "*Mariusz Zydyk*". In: (Sept. 2021). Visited on April 7, 2023. URL: https://www.techtarget.com/searchnetworking/definition/Address-Resolution-Protocol-ARP#:~:text=ARP%20broadcasts%20a%20request%20packet,and%20proceed%20with%20the%20communication..

[9] *Cisco MDNS*. Visited on April 10, 2023. URL: https://community.cisco.com/t5/wireless-mobility-knowledge-base/basic-theory-behind-mdns/ta-p/3148577.

[10] *socket$_d$oc*. Visited on April 09, 2023. URL: https://www.ibm.com/docs/en/i/7.1?topic=communications-socket-programming.

[11] Matthew Li. "*Understanding the Measures of Bluetooth RSSI*". In: (Jan. 2022). Visited on April 10, 2023. URL: https://www.mokoblue.com/measures-of-bluetooth-rssi/.

[12] *React Native Testing*. Visited on May 17, 2023. URL: https://reactnative.dev/docs/testing-overview.

[13] *Flutter*. Visited on May 14, 2023. URL: https://flutter.dev/.

[14] *Xamarin*. Visited on May 14, 2023. URL: https://dotnet.microsoft.com/en-us/apps/xamarin.

[15] *Google's Style Guide*. Visited on April 9, 2023. URL: https://google.github.io/styleguide/.

[16] *Git Docs*. Visited on May 15, 2023. URL: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

[17] *Github*. Visited on May 2, 2023. URL: https://en.wikipedia.org/wiki/GitHub.

[18] *Discord*. Visited on March 24, 2023. URL: https://en.wikipedia.org/wiki/Discord.

[19] *VSCODE*. Visited on April 23, 2023. URL: https://code.visualstudio.com/docs/supporting/faq.

[20] *POSTMAN*. Visited on April 29, 2023. URL: https://www.postman.com/product/what-is-postman/.

[21] *JEST*. Visited on April 2, 2023. URL: https://jestjs.io/.

[22] *Socket.IO*. URL: https://socket.io/docs/v4/.

[23] *Redux Toolkit Docs*. Visited on May 1, 2023. URL: https://redux-toolkit.js.org/usage/usage-guide.

[24] *deploy-ios*. Visited on May 16, 2023. URL: https://docs.expo.dev/submit/ios/.

[25] *deploy-android*. Visited on April 15, 2023. URL: https://docs.expo.dev/submit/android/.

[26] Benjamin Thiel, Kamil Kloch, and Paul Lukowicz. "*Sound-based proximity detection with mobile phones*". In: Visited on May 4, 2023. Sept. 2012.

[27] *Expo Go Customization*. Visited on May 10, 2023. URL: https://docs.expo.dev/workflow/customizing/.

APPENDICES

A - GITHUB REPOSITORY

All code is included in the Github repository linked below. Further explanations are given in the readme-file.

Github repository link

- `https://github.com/alegro91/Control-system-for-omnidrive-robot`

B1 - User test prototype

The print-outs that were used for the user testing of the application.

Figure B.1: UI prototype mockups used during user testing

B2 - Results of User Testing Performed on March 16th

User tests revised based on feedback from Elise were conducted on-site at the Gjørtz warehouse, on March 16th. Before the actual tests, we had a test run with Sondre, where we went through the plan and discussed how we would proceed.

For the main testing, we interviewed two employees at the warehouse, as potential users of the application we are developing, one operations technician and one acting team leader, one at a time.

Before the testing began, each participant was asked to sign a consent form, and informed about the intentions of the test.
For the tests, each participant was asks to perform the same three tasks:

- You need to move the robot out of the way. Find a way to connect to and control it.
- You need to connect to a robot that does not appear on the list. How would you proceed?
- A robot has stopped completely. Find a way to see if anything is wrong.

This was done using the "Wizard of Oz" method, in which the participants were presented with paper print outs of the currently intended GUI design, and pressed the objects on these as if it was an actual mobile application. After the three tasks had been completed, we had a talk with each of them, asking if they had any more comments they would like to add, or any questions/requests about the app.
The following is a rundown of the results from these tests:

- Both users were generally happy with the manual control layout, once they had selected a robot. Users seemed to prefer pressing as few buttons as possible, wanting to forgo incremental steps for a more direct approach.
- Both users seemed somewhat confused about what to do once a robot had been selected from the list of connected robots. The view for the selected robot expanded, but they hesitated to press it again to connect.
- The QR scanner icon does not seem very intuitive (is there a way we can make this clearer? Perhaps simply replace the icon with "QR"?)
- Any error messages should be available with a single press on the affected robot from the main menu (push notification for errors?).
- The "more info" button on robots with errors did not stand out enough as being a button. (Add rounded borders, pulse animation to it?)
- Both testers liked the color scheme of the application.
- There was a request for locking movement to an axis, to avoid the robot not moving in straight lines (8-direction or 4-direction joystick setup?)
- There was also a request for a virtual map in the application, to show robot location.

ADDENDUMS

BLUETOOTH

1: Bluetooth

To find out which robot is closest using Bluetooth, we can equip each robot with a Bluetooth dongle, which sends out a signal containing the ID of the robot, or an IP address and any other information that is needed for establishing a connection. A smartphone will then scan for these signals. The strongest signal (assuming the signals are being sent from equivalent hardware and signal strength) should in theory come from the nearest robot. This method could be somewhat inaccurate, but will be good enough for this purpose.
Another, potentially more reliable way to use Bluetooth to determine the distance to each robot, would be if the signal sent out from the sender on the robots contained their current coordinates. The smartphone could then check that against its own coordinates, assuming it had access to the same map that the robot used to determine its coordinates.
For either of these approaches to work, we will need to equip each robot with a Bluetooth sender, and program it to send the required information through it.

2: Triangulation using WLAN-routers as references

Another potential solution could be to use multiple WLAN-routers (at least three) to triangulate the position of each robot, as well as the smartphone, and have that information sent to the app in some way. As the goal is to check location without using a centralised server, the most sensible thing to do might be to store the locations gathered through this on each robot. Doing this would acquire additional hardware. We would also need to acquire more knowledge on the topic, as we are currently not very familiar with how it is actually done. This process could be very time consuming. Testing/implementation is also expected to be difficult.

3: Using preexisting map

If there is already a map where the robots position is plotted, detecting proximity would simply be an issue of connecting the smartphone to the same map system, plot its coordinates, then compare those to the coordinates of each robot. If such a map system exists already, and we could get access to it, this would be the simplest solution by far.

Prosjektnr 20

Solwr-Robot Control
Forprosjektplan

Versjon <1.1>

Prosjektnr 20

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
27/Jan/23	1.0	Forprosjektsplan Gruppe 2	Thomas Ystenes
20/Maii/23	1.1	Fjernet felt under punkt 6.	Thomas Ystenes

Prosjektnr 20

Innholdsfortegnelse

Prosjektnr 20

1. Mål og rammer

1.1 Orientering

Hvorfor denne oppgaven. Hvordan fikk dere tak i den.

Etter diskusjon og gjennomgang i gruppe, hvor vi gikk over alle tilgjengelige oppgaver, hadde vi en avstemming av hvilke oppgaver folk var interessert i, og dette ble førstevalget. Folk hadde interesse for oppgaven, og syntes roboten virket spennende.

1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Oppgavebeskrivelse, førsteutkast til problemstilling og resultatmål

Solwr robotics is developing a mobile robot. The robot is autonomous but in some situations must be controlled manually. This will done first using a joystick connected to the robot, but using a mobile phone can bepractical in many situations. This project i about developing the mobile phone control application.All our UI are web based so we would like that one to be web based. However it must be easy to use so an app should be written to at least start a browserdirected to the correct address. Android first, maybe IOS if time. The app must also discover the robot on the network. The students can propose other technologies but mDNS DNS-SD could be used for auto-discovery on local network. Otherwise maybe NFC or Bluetooth.Then the backend/web server will be installed on the robot PC which is running Ubuntu Linux, and will call a simple web API available to control the robot. There are a few safety aspects to be taken take care of, if the connection is broken at any point in the communication chain , the robot must stop.Choice of languages can be discussed, but typescript for the front end and Python for back end are default choices since that are use many other places in our system. Apps should be written in default system language(S).If time permit, we propose to develop a simple 2D robot simulator, showing the robot move on command. There are many robot simulators on marked but we need something simple for another project so that could be useful too.

Problemstilling dere skal utforske i prosjektet. Dette er et første utkast som kan endres dersom premisser endres underveis.

I dette prosjektet skal vi altså lage et styringssystem for en omnidrive-robot. Første utfordring her blir å finne en god måte å identifisere hvilke roboter som er tilgjengelige, og opprette kontakt mellom appen og disse. Til dette vil vi trenge å gjøre en del research på de forskjellige alternativene for identifikasjon av roboter, som Bluetooth, QR-koder, sonisk osv.

Neste steg blir å programmere et styringssystem for dem. Dette vil etter planen bli gjort ved hjelp av Navitec sin egen API.

Resultatmålene forteller hva som skal være oppnådd når prosjektet er ferdig.

Målet er å ha en app som finner relevante roboter på nettverket, og kan styre disse manuelt.

1.3 Effektmål

Hva er målet for deg / gruppa og hvilke langsiktige effekter eller gevinster virksomheten søker å oppnå ved å nyttiggjøre seg av resultatet fra prosjektet.

Vi ønsker å få erfaring med å jobbe med kunder som ønsker produkter, hvor vi selv må komme opp med teknologiske løsninger, og utvikle disse produktene. Vi har også et ønske om å knytte kontakter i industrien.

Hva virksomheten angår, så ønsker de å få en prototyp av en løsning som de planlegger å selge til kunder som driver med drift/forvaltning av varehus, med mulighet for videreutvikling.

1.4 Rammer

Behov for penger, utstyr og tid. Spesialbehov materialer og rom.

Det er mulig vi vil trenge å kjøpe inn noe hardware, for eksempel Bluetooth mottaker/sender, lyd-adapter osv., men dette er ikke enda avgjort.

2. Organisering

Hvilke aktører er involvert i prosjektet.

Aktørene i denne oppgaven er Solwr (oppdragsgiver), Ibrahim Hameed (veileder) og bachelor-gruppa, bestpende av Elias Fiskerstrand, Alejandro Grønhaug, Anders Frostrud og Thomas Ystenes.

3. Gjennomføring

3.1. Hovedaktiviteter

Opplisting av hovedaktiviteter.

Hva gjøres, hvem gjør det, hvorfor gjøres det, hvordan gjøres det. Når gjøres det, nødvendige forutsetninger før det kan gjøres, dokumentasjon / resultat av det som ble gjort.

I startfasen består det meste av arbeidet av planlegging/forarbeid og møter, samt en del research. Dette gjøres av gruppa (pluss Solwr og veileder i møter). Det skal også bli tatt referater/notater fra møtene. Dette for å ha klart for oss hva som skal gjøres framover, og for å få minst mulig endringer i senere faser.

I litt senere stadier (planlagt fra uke 5), begynner arbeid med programmering/testing av systemer. Det er også i uke 5 at vi vil begynne med rapportskrivingen.

Så lenge INGA-faget pågår, vil hoveddelen av dette arbeidet bli gjort på torsdager og fredager. Etter det vil det jobbes fulltid med bachelor-oppgaven.

For at vi kan utføre dette, er vi avhengig av kontinuerlig kontakt med Solwr, og vil også muligens trenge å kjøpe inn noe hardware. Vi vil også trenge API fra Navitec, som har utviklet programvaren som den nåværende roboten kjører på. Arbeidet vil utføres av hele gruppa.

3.2. Milepæler

Opplisting av kritiske datoer.

27. januar: Innlevering av førprosjektsplan.
30. januar: Start av prosjektskriving.

3.februar: tilbakemelding på førprosjektsplan
18-20. mars: Eksamen i 194_INGA2300_1_2023_V_1. Etter dette kan hele arbeidsuken dedikeres til bacheloroppgave.
24. april: Muntlig presentasjon på engelsk.
5. mai: Levere rapport til veileder
18. mai: Utarbeidelse av poster
19. mai: Presentasjon av oppgave
22. mai: Innlevering av rapport, vedlegg i Inspera.

Disse datoene er basert på plan for 2022, og kan bli endret for gjennomførelse i 2023.

4. Oppfølging og kvalitetssikring

4.1 Kvalitetssikring

Hvordan sikre kvaliteten på alle arbeidene

For programmering/kode skal alt testes grundig, og alle skal ha en viss grad av innsikt i hvordan koden virker, samt kodegjennomgang i planum. Testing skal gjøres både manuelt, og via unit-tester.

Hva rapporten angår, har alle et felles ansvar for at denne blir skrevet skikkelig, og at eventuelle feil og mangler blir fikset etter hvert som de blir oppdaget.

4.2 Rapportering

Til hvem og hvor ofte

Det er planlagt ukentlige møter med Solwr, hvor blant annet gruppa vil rapportere til Solwr om framgang og utfordringer de står ovenfor. Det er mulig disse møtene vil bli mindre hyppige senere i prosjektet. Veileder vil også holdes oppdatert på framgang og utfordringer.

5. Risikovurdering

Risikoanalyse som vurderer sårbarheter i prosjektet (hendelse, sannsynlighet, konsekvens og tiltak).

Hendelse	Sannsynlighet	Konsekvens	Tiltak
Manglende evne til å produsere resultater i tide	2/5	Kan variere stort basert på hva som mangler. 3/5	Sørge for at alle har oversikt over deadliner, og vet hva som må gjøres. Ha god oversikt.
Endringer i planer, som påfører ekstra arbeid/krever mer tid	4/5	Dersom det skjer relativt tidlig, kan dette håndteres relativt bra. 2/5	Grundig planlegging i startfasen. La endringer skje så tidlig som praktisk mulig.

6. Vedlegg

Følgende dokumenter leveres som separate filer ved innlevering i Blackboard i januar (obligatorisk arbeidskrav), men ikke i endelige leveransen av hovedrapporten den 20. mai!