RAM ROBOTICS AND MECHATRONICS

CONTROL SOFTWARE ARCHITECTURE FOR POWER WHEELCHAIR NAVIGATION: A STEP TOWARDS AUTONOMY

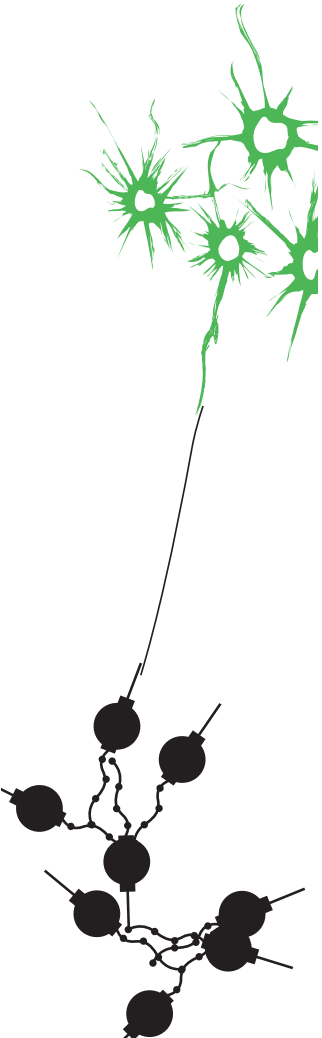
M.B. (Mrinal) Magar

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
dr. ir. E. Dertien
dr. E.H.F. van Asseldonk

July, 2024

046RaM2024
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Acknowledgement

This thesis has been a great learning experience for me, with all its ups and downs. First and foremost, I would like to express my gratitude to my supervisor, Dr. Ir. Edwin Dertien, for his contribution in designing the joystick emulator without which I would not have been able to test my work on the power wheelchair. I am also grateful for his support in guiding me through difficulties and helping me present my work in a better way. Working under Edwin, introduced me to user-centered design. Although I am not very familiar with it yet, I now understand the importance of involving users in design decisions.

I am also deeply grateful to my committee chair, Dr. Ir. Jan Broenink, for his critical and constructive feedback that kept me on track. I learned to be consistent and to pay attention to details. Although I am still chaotic in some ways, I believe I have imbibed a few ways of thinking and working from Jan that will benefit me in the long run.

Being away from home was challenging, and I could never say this in person but I have missed my parents a lot. I cannot thank them enough for their love and constant support.

I also want to say a huge thanks to Mateusz for making me feel at home, even when I was far away from home. And without whose support I can't imagine how I would have managed the past three years.

Lastly, I want to tell my sister that she is a B^3 and I can't wait to trouble her again. Thank you for always brightening my mood.

Summary

Self-driving wheelchairs have been developed but they did not reach the market yet as many of them depend on some propriety hardware of the wheelchair, which changes as manufacturers come up with new designs. Additionally, the power wheelchairs available in the market use a standard joystick module– an input device that is unsuitable for our user due to involuntary muscle spasms. To overcome these challenges, this thesis presents the development of a modular control software architecture for navigating a power wheelchair, with minimal adaptation needed for the underlying power wheelchair hardware. Given the user's requirement to use an eye-controlled tablet, the software prototype was designed to integrate this assistive device for wheelchair control. This project is funded by Ability Tech (Lab) which develops smart assistive devices for people with disabilities.

This work followed a systems engineering approach to find a list of requirements for the system under design, which is the Power-Wheelchair-Add-On Control-System-Unit (PWAO CSU). Based on these requirements, an architecture was designed by choosing from different alternatives. The PWAO CSU software architecture is structured into three layers: the user interface layer, the sequence control layer built on the ROS 2 framework, and the loop control layer. This architecture provides interfaces between the different control layers, establishing communication between the power wheelchair and the user's assistive device. A working prototype was implemented and tested in this work. The prototype also features a PID controller for precise turning motion and a simple obstacle avoidance system for safety.

The PID controller design was tested and it showed that the PW took a long time (30 seconds) to reach a steady state (10% of the reference). The real-time performance of the system was tested and it was found that ROS 2 components can have a high jitter, up to 90% of the period. A performance bottleneck was identified in the communication between the assistive device and the ROS 2 sub-system. The average delay between receiving the user command was found to be 300 ms. A user test was also conducted to provide insights into the user experience. While the user was able to use the eye-controlled tablet for control of the power wheelchair, the video stream was lagging due to which the system did not feel as responsive. The user did not experience eye strain but indicated a preference for autonomous navigation.

Contents

1 Introduction

This report presents the design of a control software architecture, also known as Power-Wheelchair-Add-on Control-System-Unit (PWAO CSU), It serves as a foundation for the creation of a self-driving wheelchair. This work evaluates various middleware and software packages to determine the best option for constructing the datapath from the user interface to the micro-controller unit which connects with the power wheelchair. A prototype of the control software is realized. Finally, a test with the user was conducted to gain insight into the user experience and evaluate the real-time performance.

1.1 Context

Self-driving wheelchairs have been developed but have not reached the market yet as many of them depend on some propriety hardware of the wheelchair. This propriety hardware is changing as PW manufacturers come up with new designs. However, by focusing on the software independently of the underlying wheelchair hardware it should be possible to deploy this software on various wheelchairs from different manufacturers with little modifications. While advancing this project, the aim is to build upon lessons learned from an earlier phase, focusing on a software framework that is expandable to integrate various navigation modes. This ensures continuous improvement in realizing the final goal of a self-driving wheelchair.

This project is necessary for a client who has dyskinesia, which is characterized by involuntary muscle spasms. As a result, the client relies on a wheelchair for mobility and uses an eye-controlled tablet as a communication tool. Most of the PW available in the market use a joystick module– an input device that is unsuitable for our client due to involuntary spasms. As the user is already experienced with using an eye-controlled tablet for his daily tasks, it was a need to able to control the wheelchair from his tablet. This project would be a step towards enabling the user to gain some autonomy and be able to navigate from point A to point B using the wheelchair.

1.2 Design objectives

The objective and goals of this thesis are listed below:

Main Design Objective

Design a modular control software architecture for developing a self-driving power wheelchair that integrates the user's eye-tracking assistive device.

Goals:

- Investigate and select the most suitable middleware and software packages to ensure real-time capabilities to the PW control system.
- Develop a control software prototype
- Test and evaluate the functionality and real-time performance of the developed prototype
- Conduct a user test to assess user satisfaction with the developed prototype.

1.3 Report Organization

This report is organized into five main chapters. The Chapter on background information provides the necessary foundation for understanding the project. It includes a brief overview of the real-time concepts and their application in hierarchical control software. Then ROS is introduced with a focus on quality of service capabilities. This is followed by an explanation of use of micro-ROS on a resource-constrained environment. Then WebSockets and their ap-

plication in the project are discussed. Chapter 3 delves into the system under design, which is the PWAO CSU. In this Chapter a requirement list is derived from various activities like stakeholder analysis, scope of the system, and use case scenarios. The outcome of this Chapter will guide the subsequent design phase. Chapter four offers a detailed view of the expected final prototype. It provides and justifies the design choices made during the development process. It also details the breakdown of the software architecture and how the components interface with each other. The next Chapter, on testing and evaluation, presents the performance of the prototype and provides critical insights into the design choices made earlier. It also includes observations made from user testing, providing an understanding of the user experience. The conclusion and recommendation Chapter addresses the objectives that were achieved and the shortcomings of this project, which can be improved upon further.

2 Background

This Chapter introduces the necessary concepts and background information for understanding this thesis. Real-time concepts used in cyber-physical systems are briefly touched upon, followed by an introduction to the middleware frameworks and software packages used in the design of the software. The final section presents work related to this thesis, which includes an overview of advancement in the field of self-driving wheelchairs and contributions made by a previous researcher on this topic.

2.1 Concepts

Understanding real-time concepts and their classification is important for designing control software for a self-driving wheelchair, where deviation in timing constraints may lead to unacceptable performance or disaster for the user. It also aids in system design to implement methods minimizing latency and jitter thereby improving the system performance.

2.1.1 Real-time concepts

A real-time task means that the time taken for execution of a task is deterministic. One definition found in Bruyninckx (2002) is:

"A real-time operating system is able to execute all of its tasks without violating specified timing constraints."

Real-time tasks are typically categorized into two main types: Hard Real-Time (HRT) and Soft Real-Time (SRT). HRT tasks must meet their deadlines and any failure to do so results in catastrophic consequences. SRT, on the other hand, has more flexibility regarding its deadlines. While missing the deadline does not result in a catastrophe, the usefulness of the task's outcome becomes less relevant over time till it is not useful. Additionally, a third category called Firm Real Time (FRT) is defined for periodic real-time systems in Boode (2018) as: *" infrequent deadline misses, less than k deadline misses in a given time frame of t seconds, will not be catastrophic for the system"*.

Latency and Jitter

The difference between the expected time a task should have been completed or (started) and the actual time it does is defined as latency in Bruyninckx (2002). In a network, latency means the time it takes for data to transfer across the network, and the variation in latency is defined as jitter. For a real-time system, the jitter is bounded.

Software architecture for embedded systems

To develop the control software architecture, a layered approach will be followed. Figure 2.1 shows the different layers in an embedded control software.

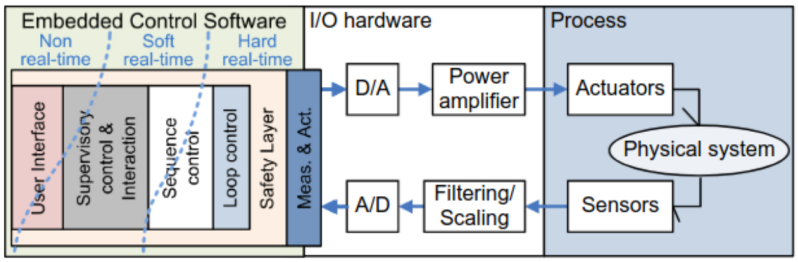


Figure 2.1: Overview of the software architecture of a cyber-physical system by Broenink and Ni (2012)

This allows for incorporating the different control layers with different timing requirements. For this project, the Firm-Real Time tasks will be implemented on a Teensy 4.0 microcontroller. The soft real-time control is implemented on a ROS 2 subsystem on a computer.

2.2 Middleware and Software Packages

This section provides brief background information on the various software packages that are used in this work.

2.2.1 ROS

Robot Operating System (ROS) is a set of open-source software libraries and tools for building robotics applications (Open Robotics, 2024). It is used widely by a large community from hobbyists to researchers for their projects. There are two versions of ROS, namely ROS 1 and ROS 2. Both these versions are similar in how the software in ROS is organized. They also share the concept of the ROS Computation Graph. The ROS Computation Graph is a peer-to-peer network of ROS processes and the communication between them. The main concepts in the ROS graph are nodes, topics, messages, services, and ROS Master (only in ROS1).

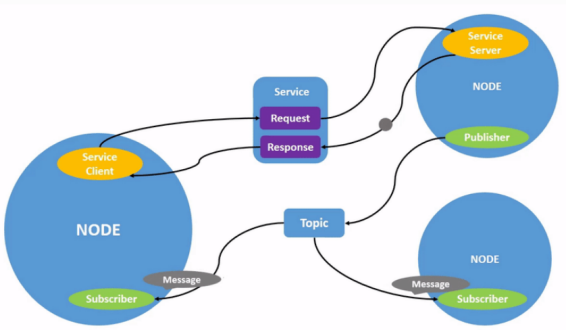


Figure 2.2: ROS 2 nodes communicating via topics and services(Open Robotics, 2024)

- **Nodes:** A node is a process that is ideally responsible for a single task. Nodes communicate with each other over topics or via services.
- **Messages and Topics:** A message is a data structure of typed fields. ROS nodes send or receive data on the network using messages. There are different message types available e.g. LaserScan msg for lidar scan data or a custom message can be used. Messages use a publish/subscribe model, where the message is published on a topic by a node. Another ROS node can subscribe to this topic to listen to the data.
- **Services:** Services are another way the nodes can communicate. These are based on a call-and-response model. Here, services only provide data to clients if it is specifically requested unlike the publish/subscribe model.
- **ROS Master (only ROS 1):** It is used to provide name registration and lookup of nodes, which enables them to locate one another to exchange messages.

ROS 2 does not have a ROS Master but it is built on DDS for its middleware which provides the transport capabilities and discovery. This was done to be able to make ROS2 real-time friendly.

2.2.2 Data Distribution Service (DDS)

Data distribution service (DDS) is an open middleware protocol by the Object Management Group (OMG) (Object Management Group, 2024). It is based on the publisher-subscriber messaging pattern. DDS does not use an intermediate message broker like the ROS Master but instead, it directly shares meta-data via IP multicast, and the routing of the messages depends on the discovery of the publishers and subscribers.

The DDS middleware shown in figure 2.3 is the software layer between the application layer and the platform layer. Basically, DDS separates the application from the operating system and it manages the transfer of messages between applications and the system.

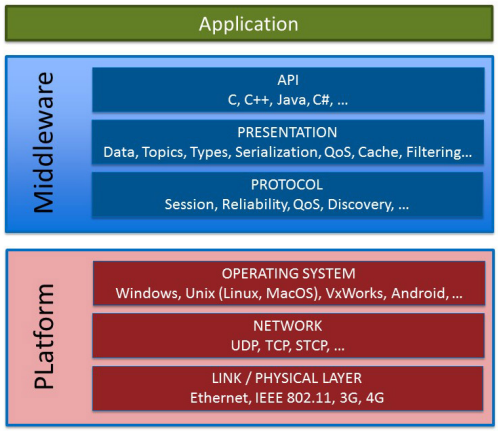


Figure 2.3: DDS middleware (in blue) manages the low-level details like data wire format, discovery, protocols, QoS, etc (Object Management Group, 2024)

Real-time computing is an important feature in applications where safety is critical like in autonomous vehicles and other robotic systems. So, ROS 2 was designed with real-time performance in mind and it supports different DDS implementations. ROS 2 uses eProsima's Fast DDS as the middleware by default.

Quality of Service

Quality of Service (QoS) policies in DDS define the behaviour of the systems which use DDS as middleware. These policies allow developers to control the data flow through the system for example reliability of message delivery, lifespan of messages, etc. ROS 2 makes use of these QoS policies for Topics, Publishers, and Subscribers which control how the data is sent and received by nodes. The QoS policies can be customized as per the application requirements. A QoS profile of a Topic, Publisher or Subscriber has many settings. Some of them are:

- **Reliability:** Can be set to either Reliable, ensuring delivery of all messages, or Best Effort, where delivery is attempted but not guaranteed.
- **Durability:** It specifies how the data should be available for late-joining subscribers.
- **Deadline:** Specifies that a publisher updates data sample within a specified time period.
- **Liveliness:** Determines if the data source is still active based on periodic updates.

Although QoS profiles can be customized for each publisher and subscriber, it is important to ensure compatibility between the profiles because data is only transferred if a publisher-subscriber pair has a compatible profile.

2.2.3 micro-ROS

Many applications require the use of microcontrollers along with a standard computer with a normal operating system (OS). Micro-ROS is a framework that brings concepts in ROS 2 onto a microcontroller. It bridges the communication between a micro-controller device like an Arduino and a ROS2 computer.

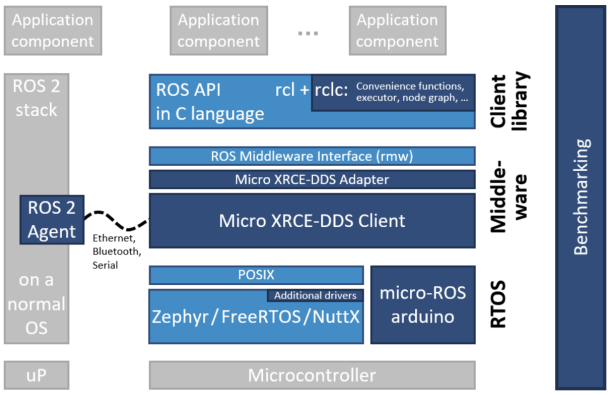


Figure 2.4: Micro-ROS architecture(eProsima, 2024)

micro-ROS uses eProsima Micro XRCE-DDS as its default communication middleware. Micro XRCE-DDS is implemented with the specifications of DDS on eXtremely Resource Constrained Environment (DDS-XRCE) an open-source wire protocol by the OMG group. It uses a client-server architecture, where the client is the resource-constrained device and is connected to the server, called the Agent. For communication between Arduino (a client) and the ROS2 host computer (Agent), a custom Serial transport protocol by Micro-XRCE is used. The micro-ROS Arduino library also provides functions for easy creation and finalization of publishers and subscribers.

2.2.4 WebSockets

WebSocket is a communications protocol that establishes a simultaneous bidirectional communication channel over a single Transmission Control Protocol (TCP) connection with the server. The client connects once with the server. Then the server can transmit data without requiring the client to send a request every time. This ensures periodic updates of sensor data which can be used for for real-time data interaction and visualization.

eProsima Integration service

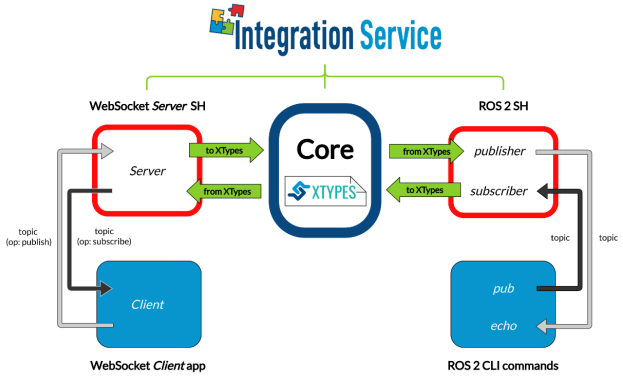


Figure 2.5: Integration Service ROS 2 - WebSocket bridge(eProsima, 2021)

eProsima's Integration Service is a software package that bridges the gap between different communication protocols used in distributed systems, including but not limited to, ROS2 systems. It provides a ROS 2 to WebSocket plugin which converts the protocols to the xtype specification, which is a common language representation within Integration Service. A YAML (YAML Ain't Markup Language) text file is used to configure the settings of topics.

2.3 Related Work

The section on related work first gives an overview of the state of the art in self-driving wheelchairs. Finally, it highlights the contributions of the predecessor to this work.

2.3.1 Literature review on Advancements in Autonomous Power Wheelchairs

A survey conducted by Fehr et al. (2000) showed that it was extremely difficult or impossible to use a power wheelchair for activities of daily living by 9-10 percent of all patients who received a power wheelchair training. Specifically, 40% of patients reported difficulty in maneuvering tasks. Clinicians surveyed indicated that an automated navigation system would be useful for nearly half of the patients unable to use a conventional power wheelchair.

Smart wheelchairs (SWs) can be distinguished based on input methods and operating modes as done in Leaman and La (2017). There has been quite significant work done on exploring a variety of alternatives as inputs (other than the traditional joystick) to PWs. This would allow people use the interface best suited for their particular needs. Many SWs developed have explored various input methods including multimodal interfaces. NavChair used voice recognition along with a standard joystick (Simpson et al., 1998). Previous work carried out at the University of Twente by De Jong (2023), implemented an eye-tracking technology as input to a power wheelchair. In addition to input methods, there have been various prototype SWs that demonstrate different operating modes integrated to allow SW users to select the best behaviour of the wheelchair. A group of SWs offers both semi-autonomous and autonomous navigation like the VAHM (Bourhis and Agostini, 1998) wheelchair. Within the subset of semi-autonomous navigation some, like NavChair (Simpson et al., 1998) and OMNI (Borgolte et al., 1998) offer multiple behaviours designed for a specific task.

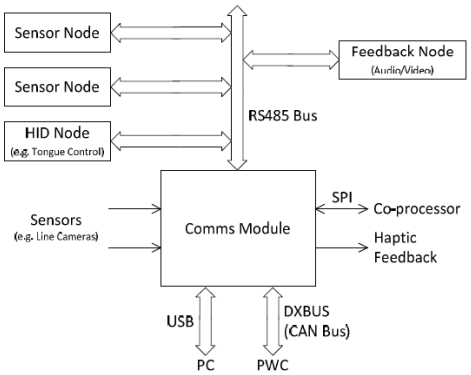


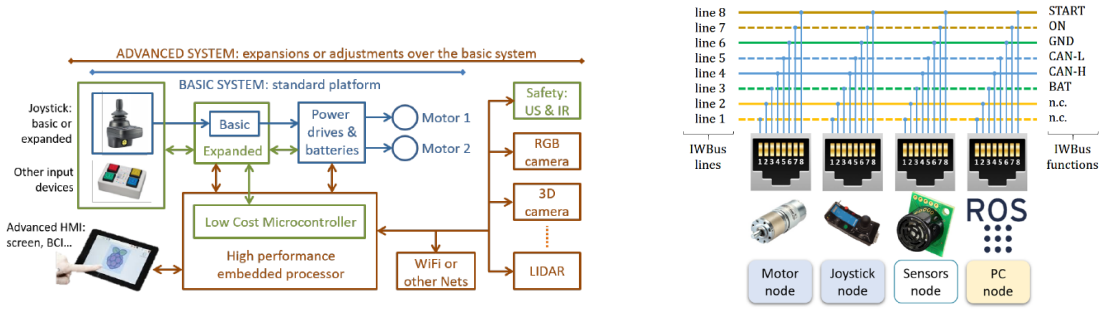
Figure 2.6: Wheelchair platform topology by Henderson et al. (2014). Here HID means Human Input Device.

In order for the system to be modular and upgradable, it needs to be able to connect to the available power wheelchairs in the market. Fewer works have attempted to solve this problem.

Henderson et al. (2014) solved this by using a general communication module by Dynamic Controls as shown in Fig. 2.6. This was used to communicate with DX and DX2 (no official full form available) control systems which were then present in many of the commercially available

PWs. However, the drawback of this solution is that it focused only on the power wheelchairs available at the time. The DX/DX2 drive systems are phased out and not used in newer PWs.

García et al. (2023) solved the problem by introducing a modular 'Intelligent Wheelchair' system where they propose an open bus architecture to allow easier communication with the PW. As the solution of using an open bus architecture as developed by García et al. (2023), shown in Fig. 2.7 or a standard communication protocol has not been adopted by the industry, another solution could be to solely focus on the software aspects of an 'add on' unit. Keeping this in mind, this study proposes a modular detailed robotic software architecture independent of the hardware. It will be possible for power wheelchair manufacturers to integrate this 'add on' module to their wheelchairs with the help of their device drivers, configuration files, plugins, etc.



(a) Block diagram of the Intelligent Wheelchair

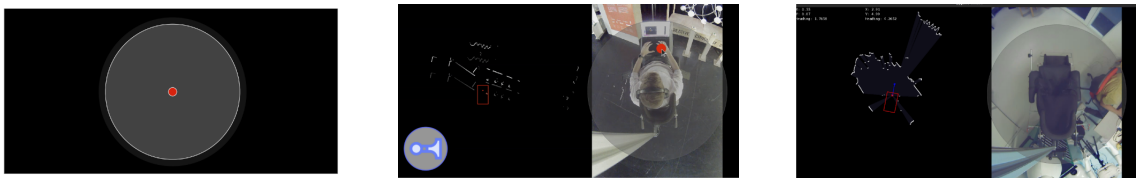
(b) Detail of the CAN bus lines used to interconnect the nodes.

Figure 2.7: Intelligent Wheelchair concept by García et al. (2023)

As the PW manufacturing industry has not yet embraced open bus architecture or standardized communication protocols, there is a need for an alternative approach involving a singular focus on the control software aspects of the 'add-on' unit. Although there have been many works that have implemented different operating modes and input methods, there is still a lack of a software architecture that is future-proof and can effectively incorporate the work done previously. Leaman and La (2017) suggested merging the most promising components from the research community's prototype to create a modular, upgradable system. This approach would yield a flexible system with the potential to cater to a wide academic audience. Furthermore, it opens the door for other researchers and engineers to expand upon the software by adding their modules rather than starting from scratch thereby streamlining the process of improvement and research.

2.3.2 Previous project

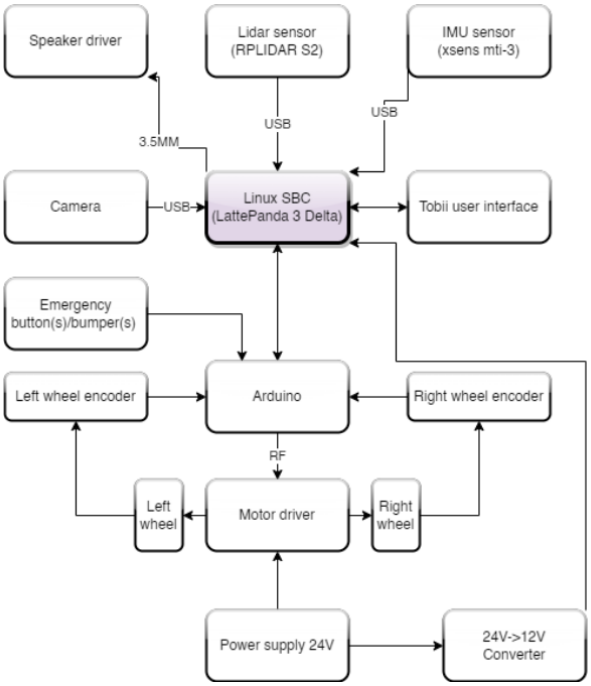
This project's initial development by De Jong (2023) was focused on developing an eye-controlled wheelchair system. It involved selecting appropriate hardware and implementing the software within a ROS1 Noetic environment for wheelchair control. Various eye-tracking-based Graphical User Interfaces (GUIs) were designed (Fig. 2.8) and tested with the user to optimize eye-tracking functionality for controlling the wheelchair.



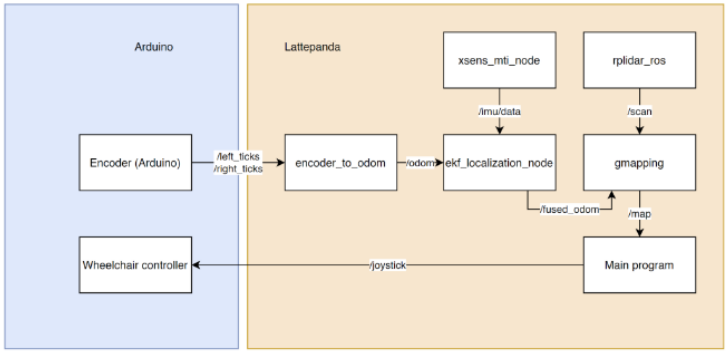
(a) UI joystick **(b)** UI joystick and LiDAR **(c)** UI self-driving

Figure 2.8: User interfaces developed by De Jong (2023)

The previous iteration used ROS 1 due to incompatibility with the Xsens IMU library and challenges integrating Arduino Leonardo with ROS 2 on a LattePanda platform. The study concluded that focusing on a GUI joystick continuously caused considerable eye strain for the user. It proved difficult for the user to concentrate on a point to control the wheelchair.



(a) Block diagram of the hardware setup used



(b) Block diagram showing the ROS nodes for self driving function

Figure 2.9: First iteration of eye-controlled power wheelchair system by De Jong (2023)

3 Analysis

This chapter defines the intended functionality and requirements of the control software architecture for a self-driving power wheelchair. To derive the requirements of this project, the systems engineering approach outlined in Faulconbridge and Ryan (2014) was followed. First, the stakeholder and business needs are defined by identifying the stakeholder and defining the scope of the system. To define functional requirements use-case scenarios are explored. The result of these activities is a list of system requirements, which are then prioritized to focus on the goals of this thesis.

3.1 Stakeholder analysis

3.1.1 Identifying the stakeholders

Stakeholder analysis was done to identify the potential individuals and organizations that influence the development of this system. The list of stakeholders and their relation with the system is listed below.

Stakeholder	Relationship
User	Interacts with or uses the system
Caregiver	Responsible for the user(e.g. parent): may interact with the system
University	Sets time constraints for the project
Ability Tech Lab	Organisation responsible for funding and support
Technicians	Provide support for developing the project
Researchers and Developers	Will work on it in the future iterations
Ethics committee	Responsible for approving user tests
Hardware manufacturers	Provide hardware (for. e.g. sensors) used in the system
Wheelchair supplier	Provide the power wheelchair on which the system is deployed
Government	Sets rules and regulations for self-driving technology
Health care professionals	Give advice or approval on use of self-driving wheelchair

Table 3.1: Relationship of the stakeholders with the system

To aid the prioritization of requirements, the most relevant stakeholders for this project were categorized on the power-interest matrix, which is shown in Fig. 3.1. The breakdown of the stakeholders as shown in the different categories based on power was first given by Mitchell et al. (1997). Hardware manufacturers and wheelchair suppliers fall under the low interest and power as they do not need any attention at this point but should be kept into account for any changes made to their products in the future. The University and Ethics Committee have greater power as they provide permission for testing and set the time frame of the project. The stakeholders in the 'Keep informed' category support the project's development and should be given regular updates. Finally, the user and the caregiver fall under the high interest and power category. They have to be satisfied and managed closely as they are directly affected by the project and are important to the success of the product. The feedback provided by this group drives improvements in the design and if satisfied they carry the capacity to promote the project.

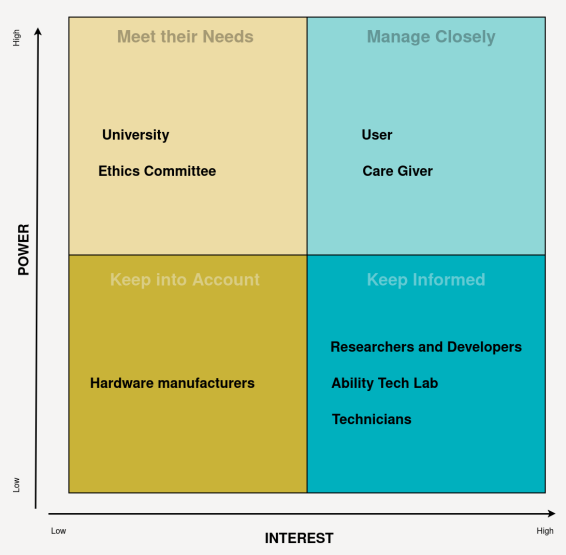


Figure 3.1: Stakeholder power-interest matrix (Interaction Design Foundation - IxDF, 2017)

Thus the needs of the user and caregiver would fall under the 'must' category of the MoSCoW classification, but the constraints set by the University and Ethics committee also influence how far this project can progress. The needs of other stakeholders can be helpful for the progress of the project.

3.1.2 Stakeholder needs

An interview (refer to Appendix A) was carried out with the user's caretaker to gain a better understanding of their specific needs and expectations. This was done to help tailor the project's outcome to meet the stakeholder's requirements effectively. The key needs identified were:

1. Navigate from point A to B
2. Detect obstacles on the path
3. Alert caretaker in case of emergencies

3.2 Scope of the system under design

A context diagram (Fig. 3.2 is used here to define the scope of the system. It provides a high-level view of the system and its interaction with external interfaces. The system under design is the Power-Wheelchair-Add-On Control-System-Unit (PWAO CSU). The sensors provide environment data to the system which processes the information and based on the user commands, sends control signals to the power wheelchair interfacing device to maneuver the wheelchair. This work does not focus on the choice of hardware as it was already done in the last iteration (De Jong, 2023), instead, it focuses on defining the interface between the hardware and the PWAO CSU and providing a software framework to develop further functionalities.

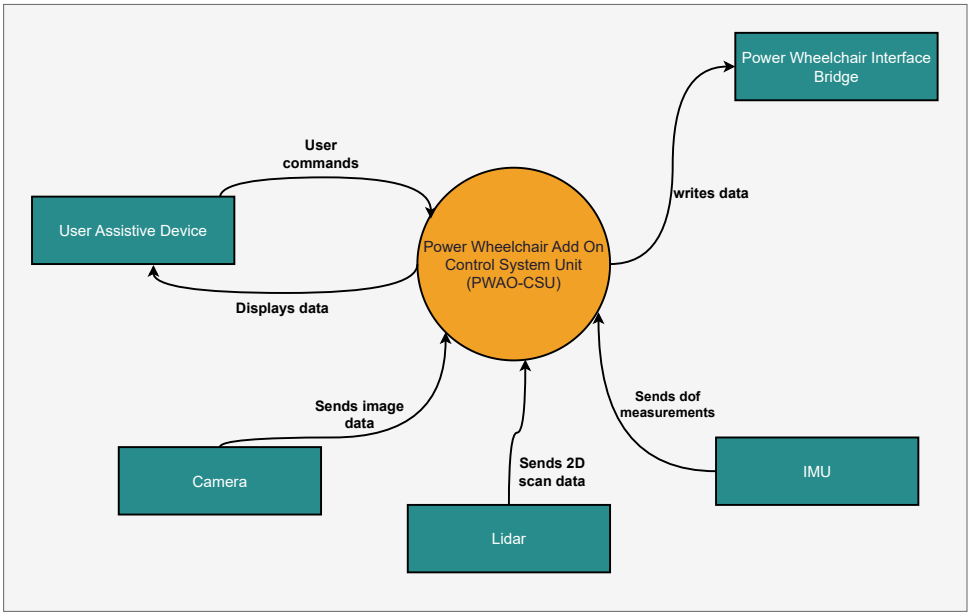


Figure 3.2: System context diagram

3.3 Use case scenarios

Use case scenarios are used to gather the functional requirements of the system under design. The intended end user of this system is our client, who has dyskinesia. Four scenarios were developed that describe much of the functionality of the system under design. They are listed below:

1. **Manual Control:** The wheelchair is powered on and the 'add-on unit'(which is placed on the wheelchair) is also powered on. The assistive communication device interfaces with the ' add-on unit' to control the PW. The user with the assistive communication device decides to control the power wheelchair manually. The assistive device which uses eye-tracking software sends instructions (depending on how the user interface is designed) to the PWAO CSU unit. These instructions are decoded in the PWAO CSU unit to generate control signals for the motor housed in the power wheelchair. If an obstacle comes in the way, the PWAO CSU unit checks if an obstacle is in the way of the user's motion. If yes, then it does not allow the command to be executed.
2. **Autonomous Navigation:** Here, the user decides to use autonomous navigation, which contains different modes, like line following, person following, and map-based navigation. The user selects the map-based navigation. The user selects a place on the map to navigate to. The map is continuously updated in case the environment has changed. As the power wheelchair navigates to the desired location, it simultaneously avoids any obstacles. At any point during the navigation, the user is free to change the destination and the power wheelchair will be redirected to the new goal. Once, the destination is reached the device alerts the user that it has reached.
3. **System Failure:** At any time the PWAO CSU is being used, if there are errors then the system will try to recover and if it is unable to do so, then it will disable the components that are not working properly. If disabling these components does not cause system failure then, the user can still use the PWAO CSU in certain cases which will be highlighted. In case of a system failure, the user will be notified and an emergency alert will be sent to the caretaker.
4. **Expanding the Software:** A user requests for customization of the add-on unit by wanting voice-controlled GPS-based navigation. The developer has to expand on the existing

software to include these particular modules. To do this, the developers refer to the documentation to understand how it works and what they need to do to add the custom module replacing the ones not required.

3.4 Constraints

Project Specific Constraints

- **Time constraints:** The project timeline is constrained to approximately 32 weeks from inception to completion. This time frame influences the scope and scale of the project deliverables.
- **Budget:** Since this project is going to be built upon existing hardware and open-source software. There are not any material costs associated with it.

External Constraints

- **Regulatory compliance:** The operation of the power wheelchair must comply with regulations regarding speed limits. This constraint ensures the safety and legality of the wheelchair in its intended environment.

Design Constraints:

The project is required to utilize only the available hardware. This constraint affects choices in software design and system capabilities.

3.4.1 List of Hardware:

The hardware used in the project is listed below. It consists of the computing units, sensors, and the power wheelchair used. To design the software architecture it is important to also look at what possibilities the hardware brings to the table.

- **Lattepanda 3 Delta:** It is a single board computer with an intel N5105 processor, 8GB Memory, and 64GB Embedded MultiMediaCard (emmc) storage. An Arduino Leonardo is integrated with the computer. This is the main computer that houses the sequence and supervisory controllers and interfaces with the user-assistive device.
- **Teensy 4.0:** The Teensy micro-controller is an alternative to the integrated Arduino leonardo or can be used along with it. This board is supported by micro-ROS library allowing communication between computer hosting ROS2 nodes and micro-controller. In addition to this, it also has more serial hardware ports, so many sensors with serial communication protocols can be connected to Teensy.
- **RPLiDAR S2:** This is a 360 degree 2D laser scanner. It can be connected to the computer by USB or to a microcontroller via serial communication. It outputs distance (mm) which is the current measured distance value between the rotating core of the RPLiDAR and the sampling point, angle (degrees) which is the current heading angle of the measurement and a boolean flag of a new scan.
- **IMU BNO055 - STEMMA QT:** The Inertial Measurement Unit (IMU) BNO055 outputs 9 degrees of freedom (dof) measurements, basically accelerometer, gyroscope, magnetometer readings. It communicates with the microcontroller via I2C protocol.
- **Luxonis OAK-D-Lite DepthAI Stereo Camera:** The Oak-D-Lite will be used for visual odometry of the power wheelchair. There is an official ROS2 driver for this camera provided by the manufacturers. Use of OAK-D-Lite requires USB C 3.0 (also known as SuperSpeed USB). Otherwise, it causes problems like significant frame loss or the ROS 2 driver for the camera suddenly stops running or does not start.
- **Quickie Salsa M2 PW:** This is the PW on which the control software will be tested. It is manufactured by Sunrise Medical and comes with two joysticks, one for the user on the wheelchair and one at the back for the caregiver. The control system used in this PW is

the Rnet Omni Control. The back joystick was disconnected and a joystick emulator was used for tapping into the control system of this PW.

- **AD5282 Digital Potentiometer:** This is a dual channel digitally controlled variable resistor that is used to set the required voltage as required by the PW control system to move the actuators. It mimics the joystick which was provided with the PW.

3.5 Requirement List

Requirements were derived from the activities described in the previous sections. They are categorised in 5 types listed below:

- Interface requirements: This focuses on real-time communication between the control software and the PW control system and the user interface device.
- Safety requirements: This lays focus on the safety needs from the systems.
- Operational requirements: Deals with different ways that the system can be operated.
- Maintenance requirements: This addresses the need for clear documentation and the ability to update it, which is important in a long-term project.
- Testing Requirements: These address the importance of establishing performance metrics for bench-marking and validating the system's performance.

Type		Requirement	Rationale	Traced from	MoSCoW
Interface	1.1	Communicate with the PW in real time	The communication will allow control of the motors of the PW which is needed for user control or self driving	Scenario 1	Must
	1.2	Communicate with user's interfacing device in real time	This allows the user to send commands to control the PW	Scenario 1 and stakeholder needs	Must
Safety	2.1	Implement a safety mechanism to detect errors	If a component disconnects or fails to start then it can cause a disaster	Scenario 3	Could
	2.2	Notify caretaker in case of emergency situation (e.g. system failure or accident)	The caretaker can be informed during emergencies so they can take control	Stakeholder needs	Won't
	2.3	Implement real time obstacle detection	Serves as an additional safety net in case of manual operation but can be integrated with autonomous behaviour	Scenario 3	Should
Operational	3.1	Support 2 operational modes	This allows for customization based on the user's specific requirements	Scenario 1 and 2 and stakeholder needs	Could
	3.2	Support 2 different user inputs	Should allow both the user and the caretaker to control the PW which may be different inputs. Also allows for customization	Scenario 1 and 2 and stakeholder needs	Could
Maintainance	4.1	Provide documentation for developed system	Aids in the development and maintainence of the system	Scenario 4	Should
Testing	5.1	Measure the performance of the developed prototype	This will help in refining the constraints and requirements further	Project Objectives	Must

Figure 3.3: List of requirements derived from previously listed activities. They are prioritized using the MoSCoW method

3.6 Performance Metrics

To measure and evaluate the performance of PWAO CSU it is important to establish specific metrics that will allow for the assessment of the functionality. This project involves data/com-

mands being transferred from the user interface to the power wheelchair and includes various intermediate components. This indicates that the responsiveness of the system to the user input should be measured. In addition, any potential delays have to be identified. The following metrics are commonly used to evaluate such systems:

Latency: Measures the time delay between sending a request/command and receiving a response in a network. It is critical to minimize latency to ensure the system responds promptly to user inputs.

Jitter: This refers to the variation in latency. Consistent performance requires low jitter to avoid unpredictable behavior.

Response Time: This is the total time taken from when a request is sent to the execution of an action. It includes both latency and processing time, giving a comprehensive view of the system's performance.

4 Control Software Architecture and Prototyping

Building upon the functional requirements outlined in the previous Chapter, this Chapter provides a comprehensive view of the control software architecture and its implementation. First, a high-level overview is provided to gain an understanding of how the system under design interacts with external systems. The later section focuses on the interfaces with external systems and within the system. As the PW is being controlled by our client, the user assistive device which is an eye-controlled tablet is considered to develop an interface compatible with the device. The Chapter concludes with specifics on the implementation of the control software.

This thesis emphasizes software framework to speed up prototyping for the future and it also builds upon the findings of the previous study by De Jong (2023). A different control strategy, involving step-by-step control of the wheelchair, was designed and a virtual joystick, tested by De Jong (2023) was implemented again with this framework but with a different UI to reduce fatigue on the user. Both operational methods featured an obstacle detection system to improve safety and addressed the need to reduce actions to be taken by the user, especially for emergencies.

4.1 Overview of the prototype

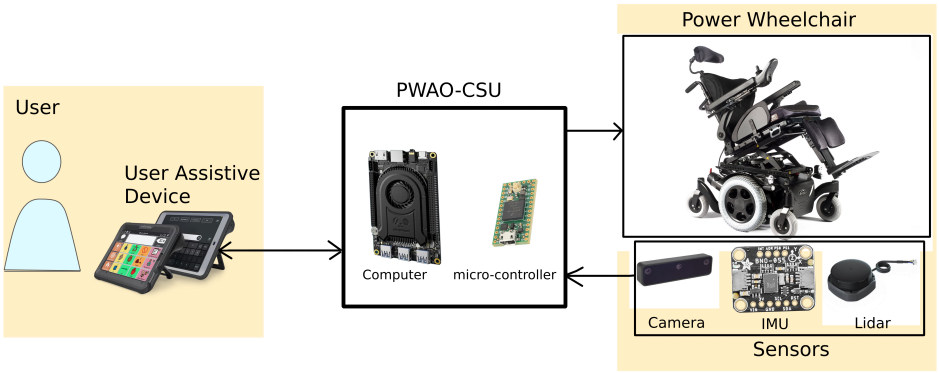


Figure 4.1: Overview of the system under design and interaction with external systems

Fig. 4.1 provides a bird's eye view of the physical interfaces of the PWAO CSU with external systems. The PWAO CSU gets data about the environment from the sensors that are attached to the power wheelchair. The user maneuvers the power wheelchair using his assistive device. The user commands are translated to signals required to control the actuators on the power wheelchair. The prioritized requirements, illustrated through a use case diagram (Fig. 4.2) aid in identifying the key functionalities the actors or external systems will utilize. The end-user will operate the user assistive device (actor) to communicate with the system under design, which in turn will control the power wheelchair (actor).

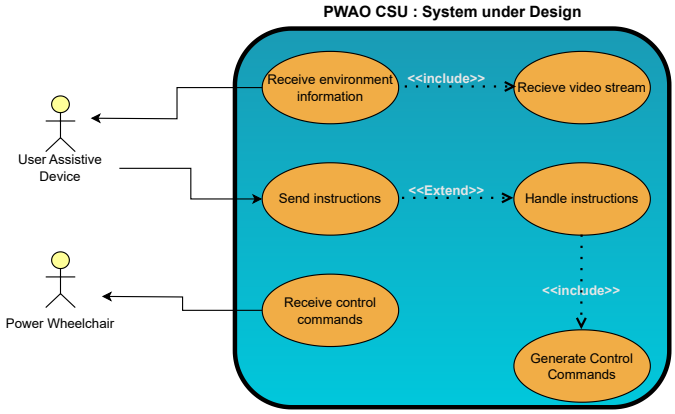


Figure 4.2: Use case diagram which shows the functionality of PWAO-CSU developed based on prioritized requirements in the earlier Chapter.

4.2 Control System Unit: Architectural View

The control software consists of two parts, the soft real-time and the firm real-time part. This distinction allows for understanding and ensuring real-time guarantees which are crucial in the case of a self-driving wheelchair.

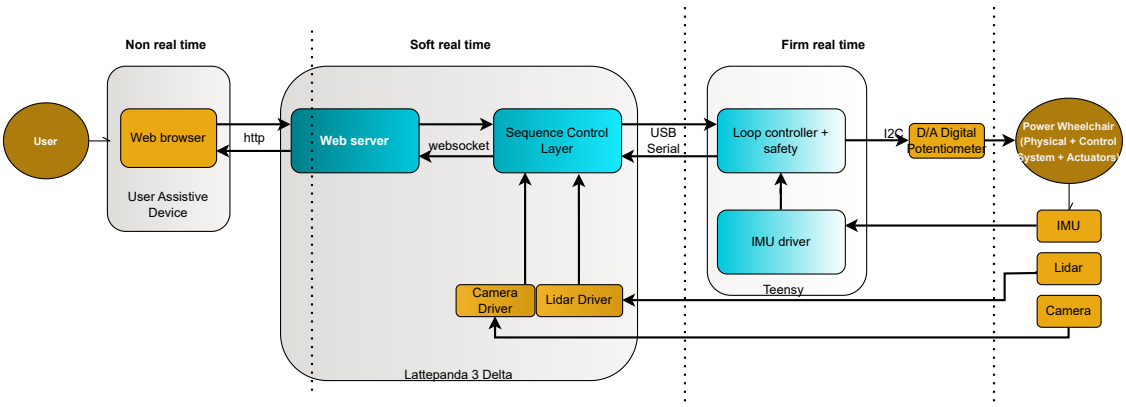


Figure 4.3: Top level architecture of the proposed prototype. The blocks in blue are the part of the system under design and the yellow blocks are external systems.

Firm Real-Time:

- Loop Controller + Safety: This is critical for real-time performance where timely and predictable responses are necessary. It includes an emergency yellow button to switch off the PW and a simple obstacle avoidance that stops the PW.
- IMU Driver is responsible for getting measurements from the IMU which are needed for the wheelchair's navigation and orientation. The IMU uses an I2C interface. Although the LattePanda 3 Delta offers an I2C interface and could potentially be used for this purpose, it has not been utilized in this project because exploring and implementing a new hardware interface would require extensive testing and validation, which was not feasible within the project's timeframe. So, the IMU was interfaced with the Teensy as a driver library on Arduino was already provided by the manufacturer
- The micro-controller acts as the central unit for executing the firm real-time operations which includes processing inputs from the IMU, and running the loop-controller.
- The Digital to Analog (D/A) device, Digital potentiometer AD5282, converts digital signals into analog voltage signals to control the power wheelchair actuators.

Soft Real Time:

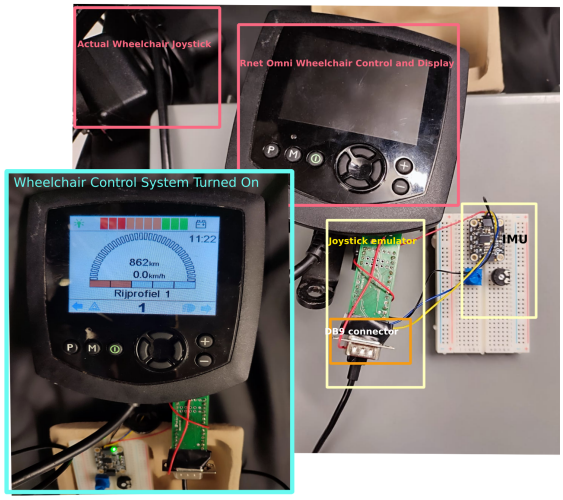
- The web server serves as the interface for remote monitoring and control from the user assistive device and hosts the required web services.
- The websocket protocol facilitates communication between the web server and other sub system components for transmitting commands and sensor feed to the user.
- The sequence control is responsible for managing the desired states based on the user commands and communicating them to the loop-controller.

4.3 Interfaces

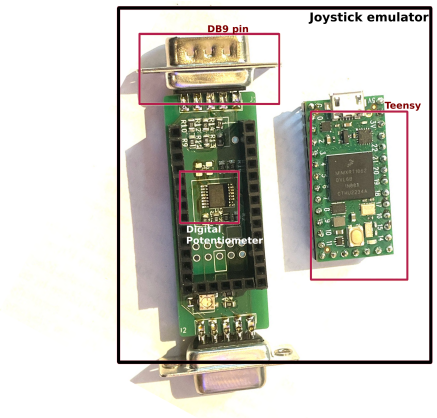
This section delves into the interfaces in the communication pipeline from the end-user to the power wheelchair.

4.3.1 Communicating with the Power Wheelchair

To communicate with the power wheelchair, a joystick emulator was used. The PW Omni Control Unit (Fig. 4.4a) has a DB9 pin connection. The digital potentiometer varies the voltage between 5-7V to turn the motors. Schematic of the joystick is shown in Appendix B.



(a) Interfacing with the power wheelchair using the joystick emulator (on right)



(b) Joystick emulator

Figure 4.4: Interfacing of the micro-controller (here Teensy) with the power wheelchair control system

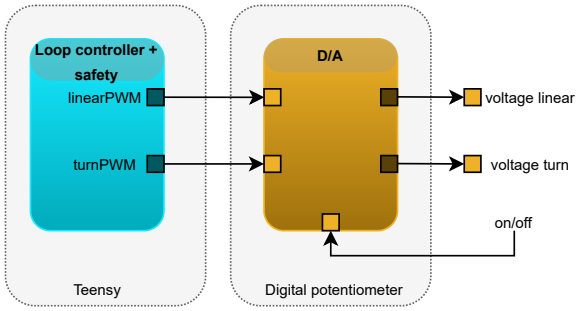


Figure 4.5: Interfaces of the loop-controller. The potentiometer has two voltage outputs which controls the linear motion and the turning motion.

```
\\ Address of the I2C device
Wire.beginTransmission(0x2C);
\\ Writing to a register
Wire.write(0x00);
Wire.write(linearPWM);
Wire.endTransmission();
Wire.beginTransmission(0x2C);
\\ Writing to the other
    register
Wire.write(0x80);
Wire.write(turnPWM);
\\ Ending the transmission
Wire.endTransmission();
```

Listing 4.1: Code snippet of teensy communicating with the digital potentiometer

The I2C transmission was tested at various frequencies to evaluate the control of the wheelchair's motors:

Loop Frequency	Observations
1000 Hz	No issues were observed; the wheelchair moved smoothly
100 Hz	No issues were observed; the motion was smooth to the eye.
10 Hz	The wheelchair motors exhibited switching between starting and stopping. No turning motion observed.

Table 4.1: Observations of different loop frequencies on the motion of the wheelchair

4.3.2 Firm Real-Time and Soft Real-Time

The sequence control algorithms are implemented on a resource-rich platform like the Lattepanda 3 Delta computer, while the loop-controller is implemented on a micro-controller. The sequence controller is responsible for setting the setpoints which the loop contoller has to follow. Fig. 4.6 shows that the setpoints are passed from the sequence controller to the loop-controller. As the IMU data is available on the micro-controller, it is sent to sequence controller for any required computation. The choice of how to establish the communication between the two parts is presented below, along with the choice of operating system and middleware used on the lattepanda computer.

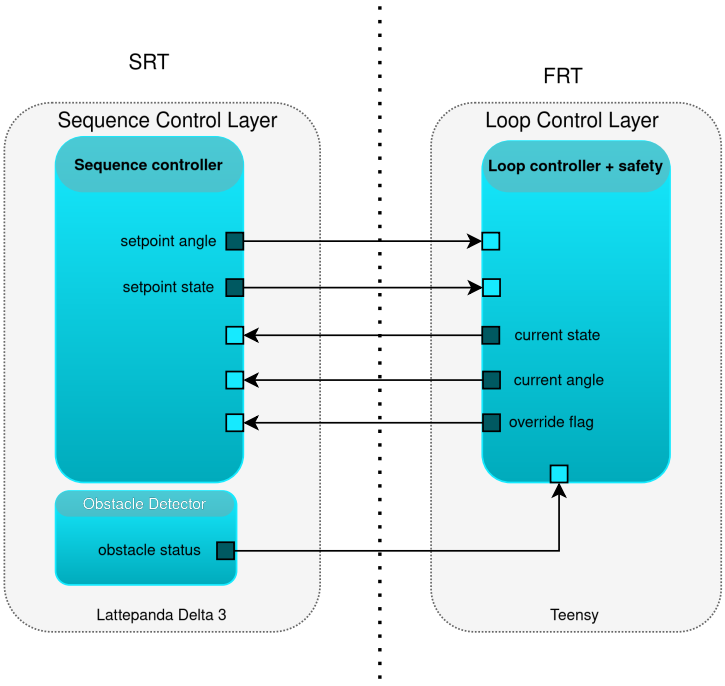


Figure 4.6: Software interfaces between the soft real time part and the firm real time part

Choice of Operating System and ROS version

The Lattepanda single board computer supports both Ubuntu and Windows operating systems. De Jong (2023) used ROS 1 to develop the software. However, considering that the development timeline for self-driving wheelchair could span couple years, it is important to consider the support and updates available for ROS versions. The following ROS versions were explored:

ROS version	Ubuntu Version	EOL
ROS1 Noetic	Ubuntu Linux - Focal Fossa	April 2025
ROS2 Foxy	Ubuntu Linux - Focal Fossa	June 2023
ROS2 Humble	Ubuntu Linux - Jammy Jellyfish	May 2027

Table 4.2: ROS distributions with required OS and the End of Life(EOL) date for the ROS distros

ROS 1 and 2 both use a publisher-subscriber mechanism. But ROS 2 was designed to be real-time friendly. The DDS implementation for ROS 2 will not be changed in this work and the default eProsima Fast DDS will be used. As ROS 2 provides different DDS implementations, it can be possible to explore Connext DDS for future work, which ROS recommends for real-time capabilities. Real-time capabilities are more important for an autonomous wheelchair, hence ROS 2 is preferred.

Given that the system to be developed has to be modular, a decentralized architecture as in ROS 2 is a better option for scalability. ROS 1 has a centralized ROS Master, the failure of which would cause all the nodes to stop working.

Every ROS distribution is dependent on the host Ubuntu version, so the choice of either ROS 1 or ROS 2 would also determine the Ubuntu version. ROS Noetic is the last distribution in the series of ROS 1 version, and it ceases to be supported in April 2025. ROS Foxy was no longer supported when this project started. Another advantage of using ROS 2 is that the software developed can be rolled forward, with minimal changes. ROS 2 emerged as the better choice as the middleware framework.

Consequently, ROS 2 Humble was chosen as the preferred distribution due to the official support extending till May 2027. The choice of this distribution requires the use of Ubuntu 22.04 as the operating system, due to its compatibility with ROS2 Humble.

Micro-controller Selection and Communication

The project hardware included two micro-controller options: an Arduino Leonardo (integrated onto the Lattepanda) and Teensy 4.0. The decision of whether to use the Arduino Leonardo or Teensy depended on the method of communication with the ROS2 system. ROS provides a ROSSERIAL package which is used with ROS 1 system. Since the ROS 2 was the preferred middleware choice, it was not possible to use the ROSSERIAL package.

Micro XRCE-DDS provides the bridge between the ROS 2 system and the micro-controller with the help of micro-ROS API. micro-ROS does not support Arduino Leonardo but Teensy 4.0. Another possibility was that Arduino Leonardo can send and receive messages over the serial port, without using the ROSSERIAL package, as it is integrated with the Lattepanda 3 Delta.

Criterion	Weight	Serial Port		micro-ROS	
		Description	Points	Description	Points
Scalability	1	Requires FSM	-	Uses topics	+
Reliability capabilities	1	No	-	Yes	+
Ease of use	1	Easy to setup	+	Steep learning curve	-
Total			**-1**		**1**

Table 4.3: Weighted decision table for choice of communication with the micro-controller

Table 4.3 compares the two choices by scoring them for different criteria. Scalability measures if the communication method used would allow for adding more interfaces than the ones already present in Fig 4.6 (if required in the future). Use of the serial port without ROSSERIAL or

micro-ROS package means that a Finite State Machine (FSM) or any other mechanism would be required to get the correct data. This is not the case for micro-ROS as it uses topics for communication. As the number of interfaces would increase, the number of topics or the states in a state machine would also increase. In this case, it is easier to manage topics than have large FSMs.

The next criterion of reliability capabilities, measures if the communication supports best effort and reliable communication. The constraint on the maximum transmission unit (MTU) in the case of the best effort profile is 490B for Teensy 4.0.

Finally, ease of use was also considered. The communication over serial port is straightforward to set up, while micro-ROS requires a beginner to follow tutorials to understand how to use it.

To conclude, micro-ROS was chosen as the interface between the ROS 2 system and the microcontroller, leading to the choice of Teensy 4.0 as the microcontroller.

4.3.3 Communication with the User Assistive Device

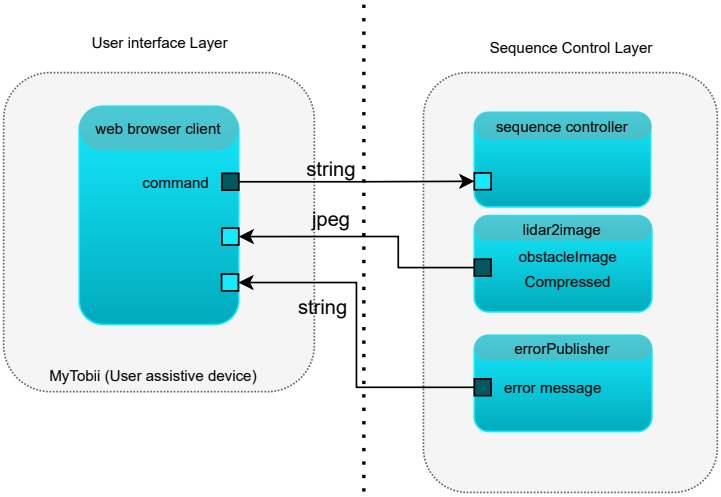


Figure 4.7: Interfaces between the ROS2 subsystem on the Lattepanda and the web client on the user assistive device(in this case a browser). The user command is sent to the ROS 2 subsystem. The user receives image and error messages as feedback.

The user interface for this project is a MyTobii Dynavox tablet, which supports WiFi communication. Similarly, the Lattepanda also has built-in WiFi capabilities, making communication over LAN the preferred method. A drawback to this is that the user interface device is not solely dedicated to communicating with PWAO CSU, and there can be other processes that access the WAN. Depending on the network interface(WiFi chip) capabilities of the user interface device, it can increase latency and jitter and degrade real-time performance.

ROS–Websocket bridge

As seen in Fig. 4.7, the communication between the two devices is bidirectional. To facilitate this, two software packages that provide a bidirectional communication layer using the WebSocket protocol were identified: Integration Service by eProsima and Rosbridge Server. For details about these software packages refer to Chapter 2.

Criterion	Weight	Rosbridge server		eProsima Integration service	
		Description	Points	Description	Points
Real-timeness	1	Higher Latency	-	Lower Latency	+
Compatibility	1	No	-	Yes	+
Support	1	Established	+	Not much	-
Scalability	1	Yes	+	Yes	+
Ease of use	1	Easy to setup	+	Easy to setup	+
Total			**3**		**4**

Table 4.4: Weighted decision table for ROS2 and websocket communication

The first criterion of real-timeness compares the real-time performance of the two packages. The results are presented in Appendix C. The tests showed that Integration Service had better performance.

The second criterion, which is compatibility, actually checks for QOS compatibility. Some messages, e.g. sensor messages use best-effort reliability settings to ensure timely updates. Integration service allows for customization of the QOS settings, however, ROSbrige_server has not yet implemented this possibility. Currently, it allows for subscription to topics with different QOS settings, but it does not guarantee compatibility.

Rosbridge_server has an active support community and more users compared to eProsima Integration Service. Both packages have similar steps for installation and usage.

Server choices

To enable WebSocket connection on the client browser, an HTTP request is sent from the client to the server to upgrade the connection to Websocket protocol. So, after the first handshake in HTTP, the WebSocket connection is open and unlike HTTP there is no need for a handshake for every server response. This requires an HTTP port. During the initial development of this prototype, an HTTP server was implemented using Python SimpleHTTPServer because it can be set up using just one command on the terminal. However, enhanced functionality including better latency measurements required to serve Network Time Protocol (NTP). Flask server was chosen for further development as it provided the necessary features and was already familiar to the Ability Tech Lab development team.

4.4 Implementation

This section outlines each software component developed. First, the user interface layer is explained, and then the SRT part and the FRT part is detailed.

For this prototype, two operating modes were implemented:

1. **Step Motion Mode**: In this mode, each user command translates into a discrete, small action or movement. For instance, if the user presses a button, the wheelchair moves a short, predetermined distance or angle and then stops until the next command is received. This allows for precise, controlled movements.
   De Jong (2023) found that for the user with an eye-tracking tablet, focusing continuously at a point for longer periods was fatiguing and straining to the eye. So, here, a step motion mode was implemented that would allow the user to more control at his pace, reducing eye strain.
2. **Continuous Motion Mode**: As long as the user continues to issue a command, the wheelchair will keep moving in the specified direction. The movement stops only when the command stops, allowing for smooth and continuous motion control.
   This mode was designed to mimic a joystick and provide natural and intuitive control. It might be a better option than step motion to use for large open spaces. The user inter-

face designed for this prototype is different from the one implemented in De Jong (2023). Therefore, it was necessary to test how the user would interact with this mode and help determine if it can be further improved.

4.4.1 User interface layer

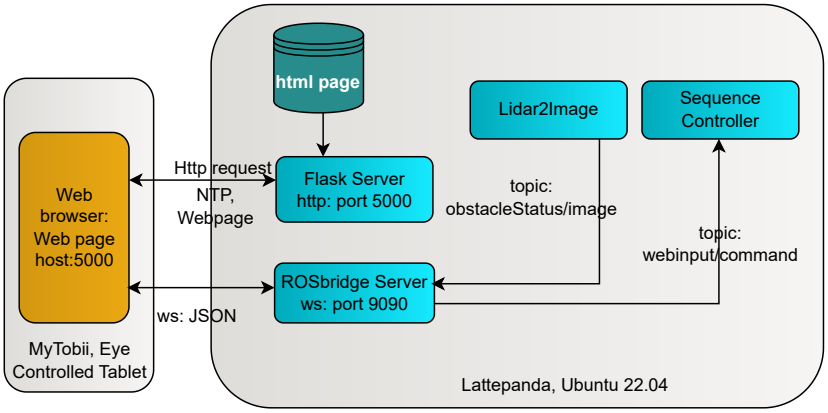


Figure 4.8: Data transfer from ROS 2 nodes to the web-client

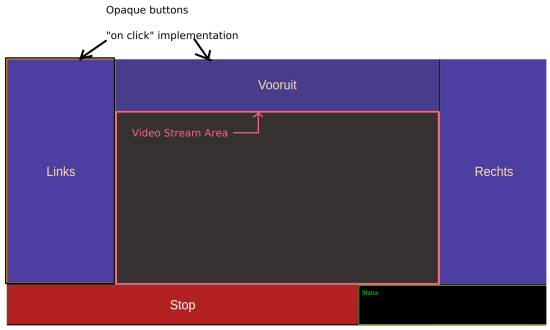
The ROS messages are transferred over the Websocket protocol. The webpage hosted on the server is accessed by entering the host's IP address in a web browser. The HTTP handshake is conducted on port 5000, which establishes the initial connection between the client and the server. After this handshake, the subsequent WebSocket communication occurs on port 9090.

The webpage uses the ROS Javascript library – ROSlibjs, to publish and subscribe to topics. The webpage creates an instance of ROSLIB.Ros object which specifies Websocket server's IP address (including port 9090). This instance is responsible for managing the connection to the ROS– WebSocket bridge.

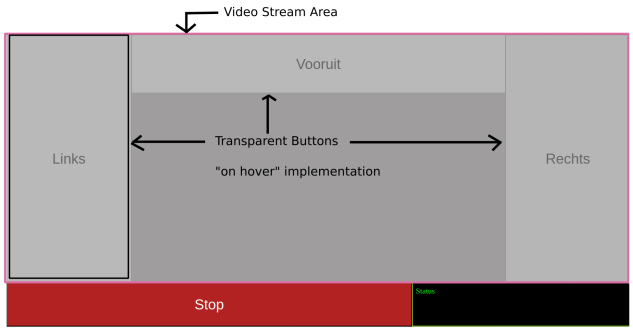
```
    var ros = new ROSLIB.Ros({
    url: 'ws://<host_ip>:9090'
})
```

The HTML layout of the webpage was based on the interface of another application (4.9c) that the user uses to play 'sjoelen', a Dutch shuffleboard game. This interface has already been tested and proven effective for the user, thus making it the logical choice for the design. The buttons are large so that the user does not need to focus on a small area which reduces eye strain.

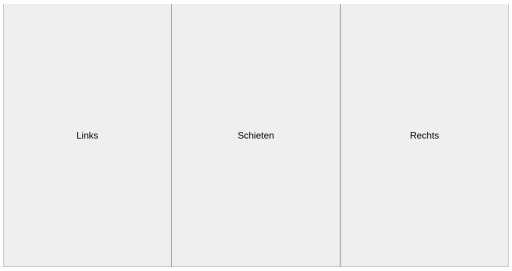
In the first iteration of this project, the interface buttons were implemented using an 'on click' mechanism (4.9a) requiring the user to dwell on the button for some time to register a click. However, based on feedback from the user's father (4.9d), this was changed to a 'hover' mechanism. Now, a click is registered when the mouse hovers over the button, reducing the strain on the user and making the interface more user-friendly but it gives rise to problems of accidental and unwanted clicking. This was partially addressed for 'step motion mode' as it requires the user to hover in and out of the area to register 1 click. The image stream area was also increased to improve visibility and the buttons were made transparent(4.9b). The interface is in Dutch, as that is the language of communication of the user.



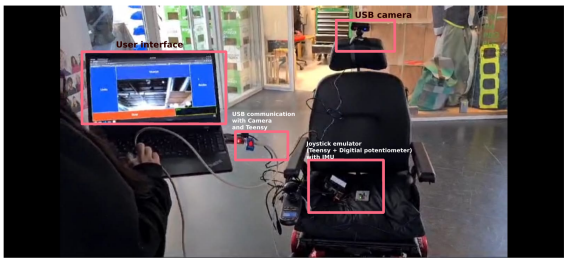
(a) HTML template used for published on the server for the user interaction, first iteration. 'Vooruit' translates to 'Forward', 'Links', and 'Rechts' to 'Left' and 'Right' respectively.



(b) Second iteration of the user interface. Here, the 'Stop' button also uses 'on hover' but it is kept opaque to stand out.



(c) HTML template used for sjoel robot



(d) First iteration of the prototype, which was shown to the user's father to get feedback

Figure 4.9: Webpage layouts

The buttons send a message to 'webinput/command' topic in a 'string' format. These messages are parsed by the 'sequence_controller' ROS 2 node to generate a setpoint for the loop_controller. Fig. 4.7 shows that the web client also receives an Image from the 'lidar2image' ROS 2 node. This message is of the 'sensor_mgs/CompressedImage' type. This image is generated from the front-facing Oak-D-Lite camera and it is layered with a line that indicates obstacles present in front, left, or right of the PW. Additionally, the web client can also receive error messages of 'string' msg type. The same interface layout shown in Fig. 4.9b is used for both the operational modes, with a slight difference in the interpretation of mouse 'hover' as a click.

4.4.2 SRT: Sequence Control Layer

The Sequence Control layer comprises ROS 2 nodes and libraries, detailed below. This layer is resonsible for translating commands from the User Interface layer into setpoints for the power wheelchair to follow. It also includes the sensor ROS 2 drivers, as well as image processing and obstacle detection ROS 2 nodes.

Sequence controller node

The sequence controller node is responsible for generating setpoints for the system. It subscribes to the 'webinput/command' topic, which uses the ROS message type 'std_msgs/String'. Upon receiving a message, the sequence controller node updates the next state and angle. These are then published at on a timer of 1ms to the topics 'setpoint/state' and 'setpoint/angle'. The setpoint angle is in degrees, and the setpoint state is represented by the values 0, 1, and 2, corresponding to the states Stop, Forward, and Turn. The distinction between left and right turns is handled by the loop-controller based on the 'setpoint/angle'. Due to message size con-

straints on the serial transport when using micro-ROS, it was necessary to keep the message size as small as possible, so the 'std_msgs' message type were used.

Additionally, the sequence controller node receives feedback from the Loop Control Layer to adjust its setpoints and states. The current state and angle (degrees) are published by the loop-controller on 'loop/currentState' and 'loop/currentAngle'. 'loop/override_flag' is a boolean value that if True, overrides the user's command and resets the setpoint state to Stop. It also resets the setpoint angle to the current angle.

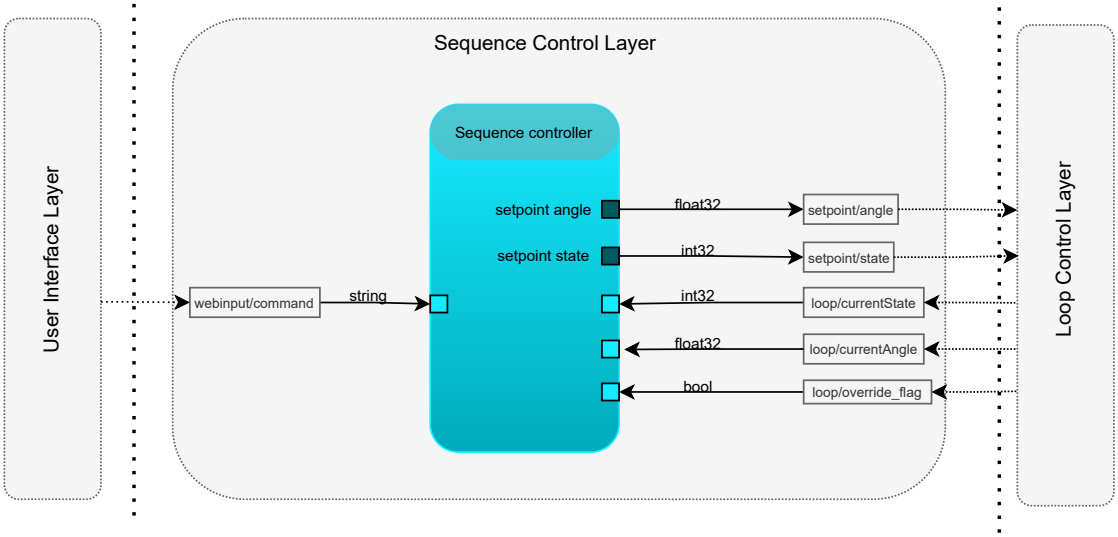


Figure 4.10: Sequence controller ROS 2 node with topics

For both modes described earlier, the implementation of the interfaces for the sequence controller remains the same. However, the generated setpoint angle differs. For the 'step motion' mode, the angle is incremented or decremented by 10 degrees. For the other mode, instead of using degrees, the setpoint angle publishes a value of -1 or 1, depending on the direction of the turn, and 0 for moving forward. Also, for the 'step motion' mode, the forward motion is set on a timer of 2 seconds after which it stops, while there is no timer in the 'continuous motion' mode for the forward state.

Error Handler

To handle system errors, like disconnection of a sensor or failure of a node to run, the ErrorHandler class was realized. The idea was to use this class methods' in different subsystems to detect some standard errors. Currently, this class logs errors based on the severity level, which can be expanded upon further to take necessary actions. To demonstrate the functionality of the class, an error publisher node was created which uses the Errorhandler public methods. The ErrorPublisher node subscribes to sensor topics and if a callback is not initiated within a 2-second window, it utilizes the public methods of the ErrorHandler class to log a "No sensor message" error, indicating that a specific sensor node is not actively publishing messages.

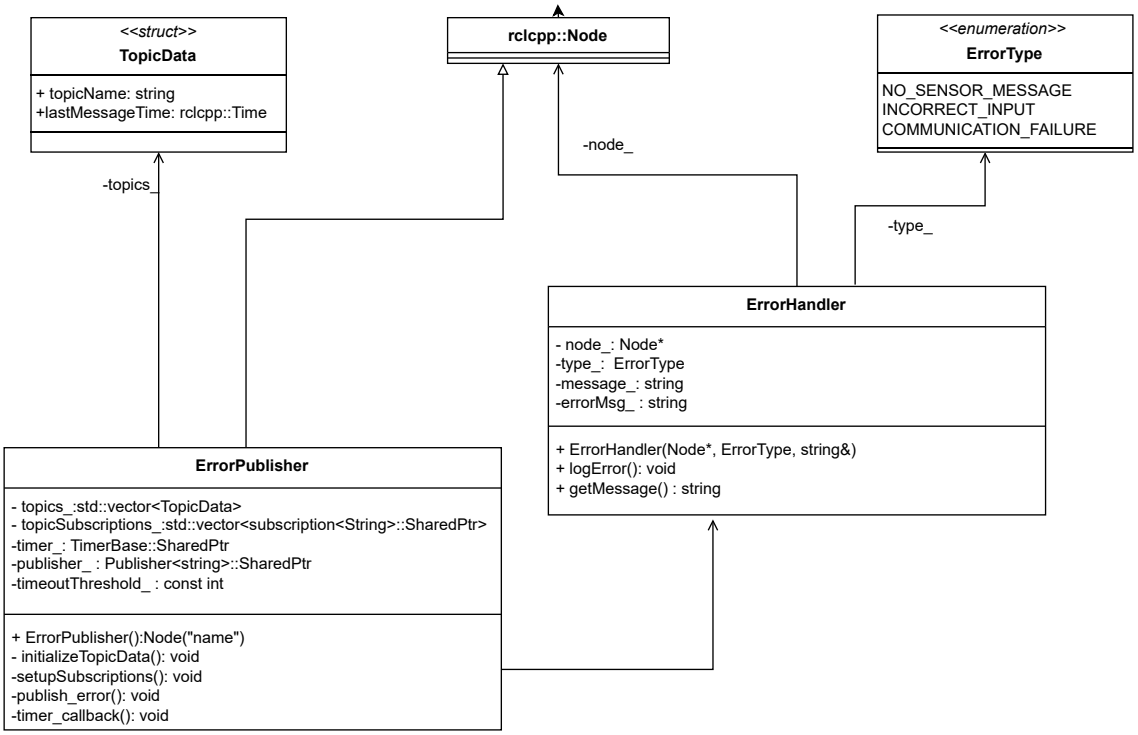


Figure 4.11: Class diagram of ErrorHandler showing the use of it's methods in a ROS 2 node

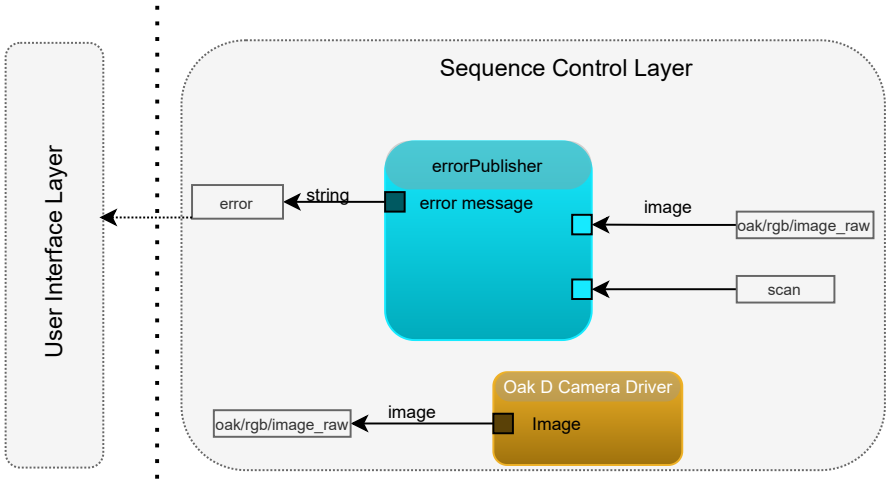


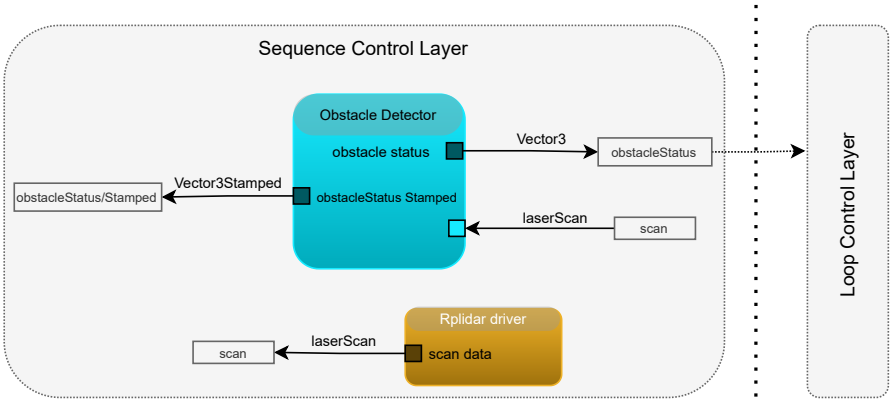
Figure 4.12: errorPublisher node which relays errors to the user interface

The TopicData structure is used to store the name and the time of the last message received. This is used to monitor the status of active nodes. The error publisher node subscribes to a list of topics that are to be monitored. If a node fails to publish data for '2' seconds, then an error message is logged. The errors are associated with nodes and categorized using the 'ErrorType' enumeration. This was done to provide a standard way to categorize and handle common errors.

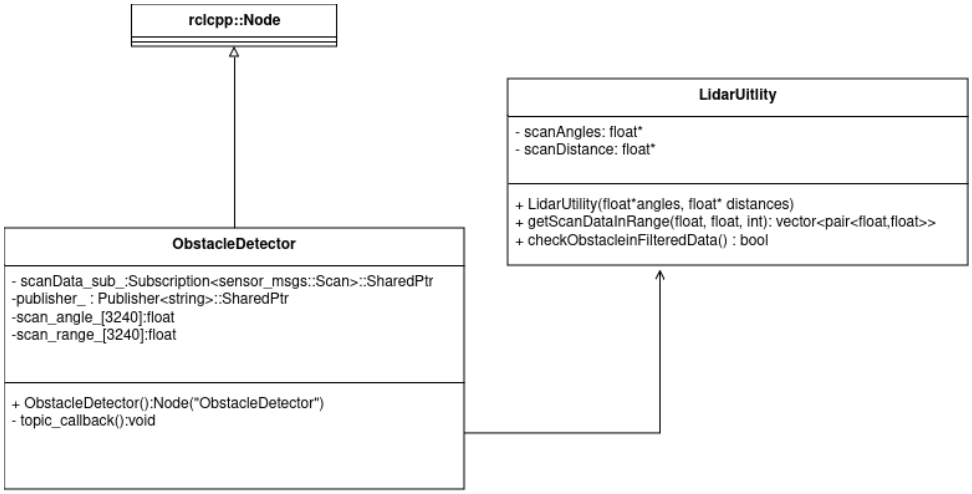
Obstacle Detection

The obstacle detector node is implemented on the SRT because the LiDAR is interfaced with the ROS 2 subsystem. Instead of transmitting the entire scan, which is too large for serial communication, only obstacle detection flags are sent. This reduces the message size significantly.

Additionally, this approach allows for further sophistication within the ROS 2 subsystem. The obstacle detector node publishes two messages to the topics: A 'geometry_msgs/Vector3' type to 'obstacleStatus' and 'geometry_msgs/Vector3Stamped' to 'obstacleStatus/Stamped'. Both the messages are vector of 3 elements to represent if an obstacle is present or not in Front, Left, and Right(see Fig. 4.14. The Vector3Stamped message type is used to send time stamps with the data for time synchronization of obstacle status and image data which is used for the image stream feedback to the user.



(a) Obstacle detector node with topics



(b) Class diagram of Obstacle Detection

The LiDAR utility class was designed with scalability in mind, as a single LiDAR may not be sufficient. It provides methods to select specific ranges of data rather than using the full 360-degree scan since parts of the LiDAR's view are often obstructed by the wheelchair. By focusing on the unobstructed sections, faster scanning is ensured. This separate class can also serve as a reusable library for other ROS 2 nodes, facilitating the development of more complex obstacle avoidance systems and reducing repetitive code. An obstacle is detected if it is present within a specified distance. For example: If the obstacle is present in the angle range 90 deg to 30 deg, then it is detected on the left.

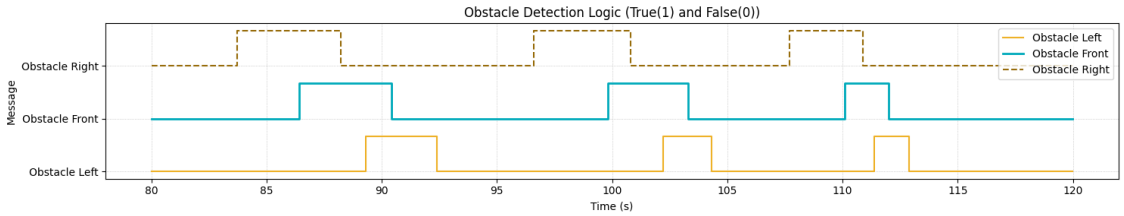


Figure 4.14: The vector3 message type has 3 elements which indicate if an obstacle is present

The pattern shown above was seen when an object was moved from right to left in front of the lidar like in Fig. 4.15.

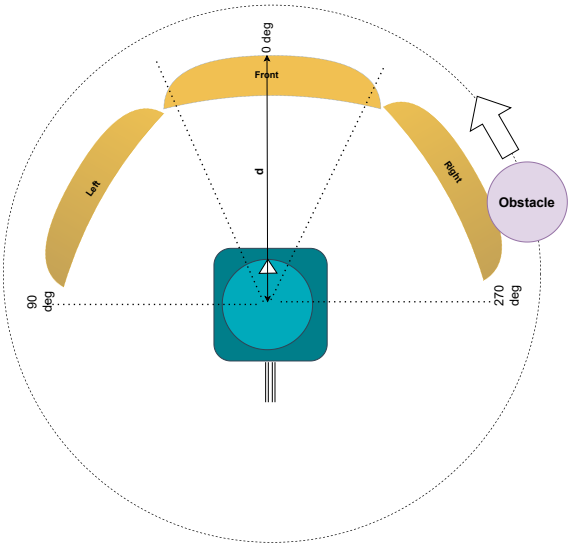


Figure 4.15: This diagram shows how the obstacle detection works. 'd' is the distance from the center of the 2D lidar to the obstacle.

Image Stream with Obstacle Status

To provide user feedback on obstacle locations, 'lidar2image' node subscribes to both 'oak/rgb/image_raw' and 'obstacleStatus/Stamped' topics. It overlays the image (from the Oak D lite camera) with a line divided into three segments, indicating whether an obstacle is on the left, right, or center. The line is green if an obstacle is not present and red if present. The stamped messages are necessary for time synchronization which relies on timestamp comparisons. The node publishes a compressed image because the ROSlib library requires compressed images for efficient transmission over the WebSocket protocol, rather than sending raw images.

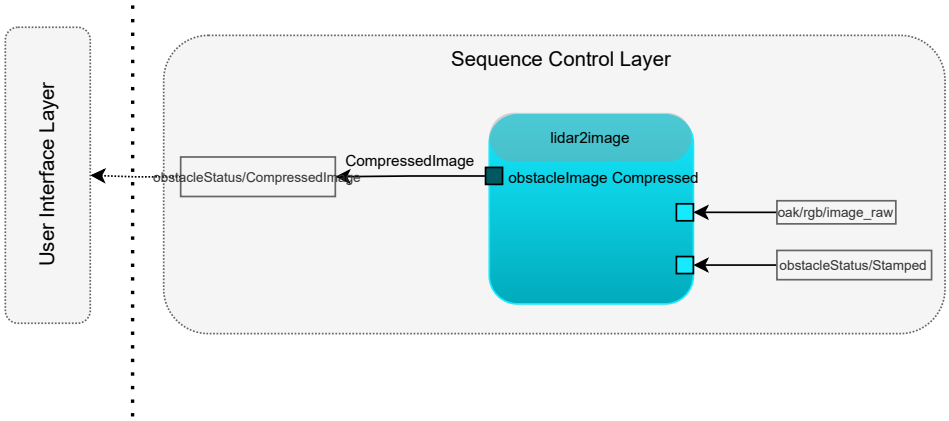


Figure 4.16: The lidar2image node publishes the compressed image which is used for user feedback.

4.4.3 FRT: Loop Control Layer

The loop-controller is responsible for generating commands to control the PW motors. For the step motion mode, PID control was used to turn the PW to the desired angle.

Loop Controller with Obstacle Avoidance

The loop-controller node on Teensy subscribes to 'setpoint/state' and 'setpoint/angle' topics. It also subscribes to the 'obstacleStatus' topic to have the micro-controller perform immediate reaction checks (obstacle detection flags) and take emergency actions (like stopping), to ensure that the vehicle can react quickly to obstacles, independent of the main sequence controller node on the ROS2 system. This adds a layer of redundancy and provides an alternative in the case a more complex obstacle avoidance algorithm on the ROS 2 subsystem fails or is delayed.

Fig 4.17 shows the steps performed in the loop-controller node. For both the operational modes a similar flow diagram is applicable except the block 'Compute PID Controller Output' is not used for 'Continuous Motion' Mode. Instead a fixed output value is sent to the digital potentiometer.

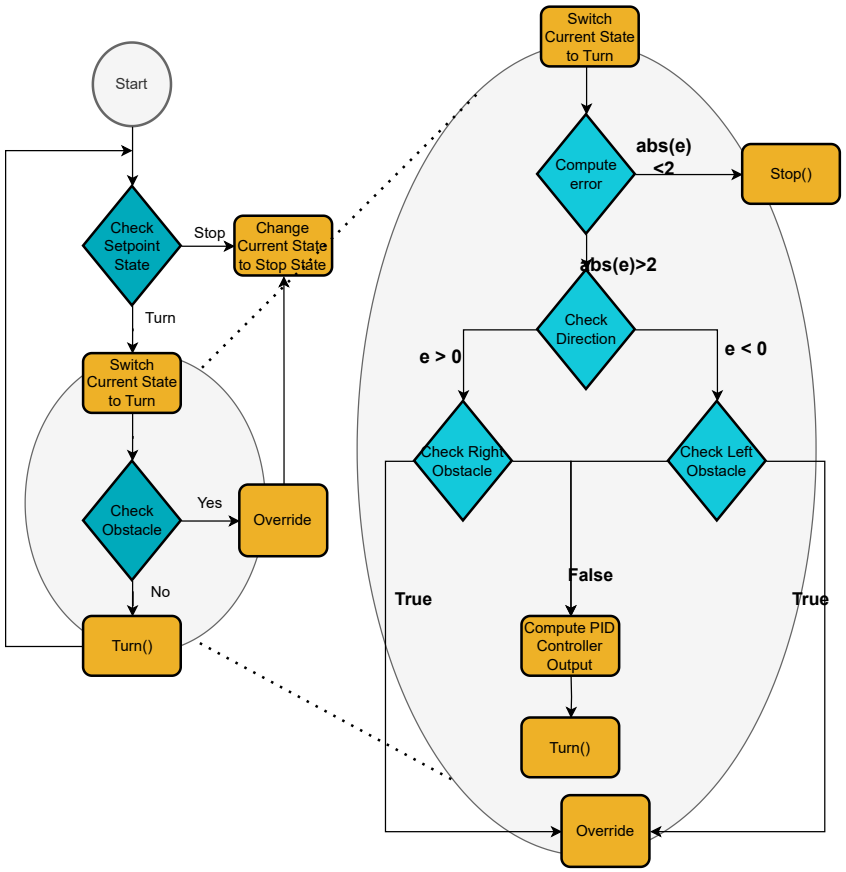


Figure 4.17: High-level flow diagram showing how the loop-controller algorithm works.

The plot below shows 4 variables: Override Flag, Obstacle in Front, Current State MoveForward, Setpoint State MoveForward.

- **Setpoint State MoveForward:** Is '1' when the user clicks 'Vooruit' button, else '0'. It is a variable in sequence controller node.
- **Current State MoveForward:** Follows Setpoint State. Is '1' when the loop-controller state is MoveForward, else '0'.
- **Obstacle in Front:** Is '1' if Obstacle is present in range 30 to -30 deg in front of the lidar, else '0'
- **Override Flag:** If the Setpoint State MoveForward is '1' and the Obstacle in Front is '1', the variable Override Flag is '1', else '0'. It is published by the loop-controller. When this variable is high, the PW breaks and the Setpoint State is overriden.

The loop-controller publishes 'override_flag' message which tells the ROS 2 sequence controller node that the user's command was overridden and the PW is stopped. Fig 4.18 shows how the 'override flag' is used to override the user commands and reset the setpoint State. This reset was needed so that the system would not falsely think that the desired action had been carried out.

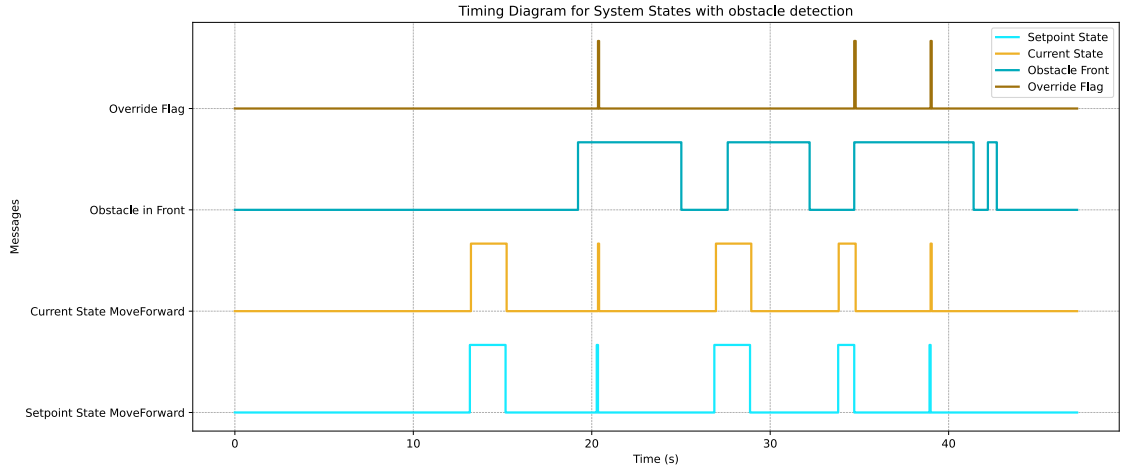


Figure 4.18: Timing diagram of relevant variables showing the working of obstacle avoidance

PID controller

A PID controller based on the Arduino's PID library by Beauregard, B (2017) was implemented in the loop-controller node for the step motion mode. Use of typedef enum pid_controller_t allows for different controllers such as PI, PD, and PID. The PID controller used in the loop-controller has a 'float' data type as input and 'int' data type as output. So, the class uses two template data types: T and U. This design allows for flexibility in defining the types for input and output of the controller and makes the PID controller adaptable to various data types.

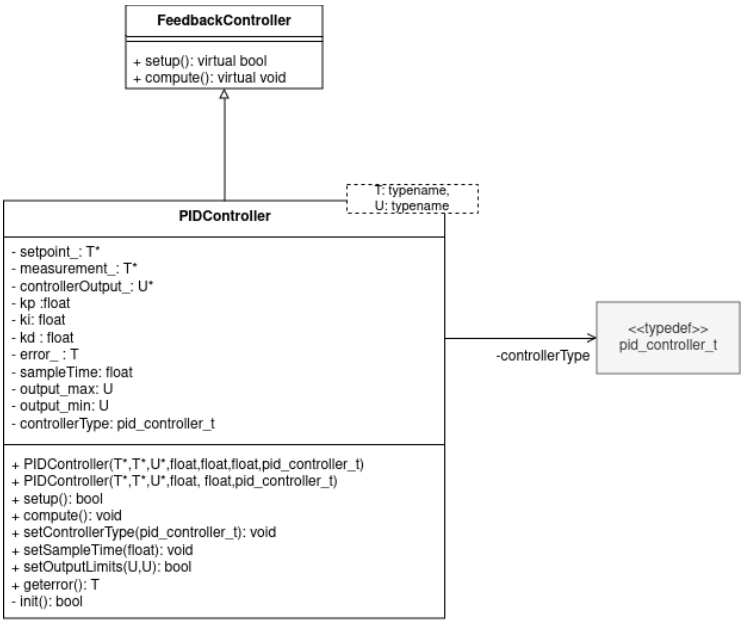


Figure 4.19: Class diagram of PID controller used in the loop-controller

The PIDController class was designed so that it can be utilized by both the sequence controller and loop-controller. It is derived from the virtual Feedback Controller class. The Feedback Controller class is virtual to allow the methods to be overridden by the implementation of different controllers.

5 Testing and Evaluation

This Chapter presents the results of three main tests conducted. First, it presents the PID controller functional test and examines whether PID is a suitable control method for this PW. The second test presents the results of performance tests which evaluate the real-time response of the system to the user input. The final section on user-test provides insights into the user experience of the prototype developed using the proposed control software architecture.

5.1 Functional test: PID Controller

The PID controller developed in the previous chapter is implemented in the 'Step Motion' mode. Currently, only an estimate for the degree of rotation of the PW was available from the IMU, so a PID controller for turning motion was implemented. The PID response of the PW is evaluated here.

5.1.1 Test Setup

IMU BNO055 is used to measure the orientation of the PW wheelchair. A setpoint angle of 10° is generated by the sequence controller node on the ROS2 subsystem, which the loop controller receives. The PID controller takes the setpoint angle as input and outputs an integer value (from 85 to 254) for the digital potentiometer to vary the voltage from 5-7V. The setup used for testing and verifying individual components like obstacle detection and PID controller is shown in Fig. **??**. The block diagram in Fig. 5.1 shows the architectural view of the setup.

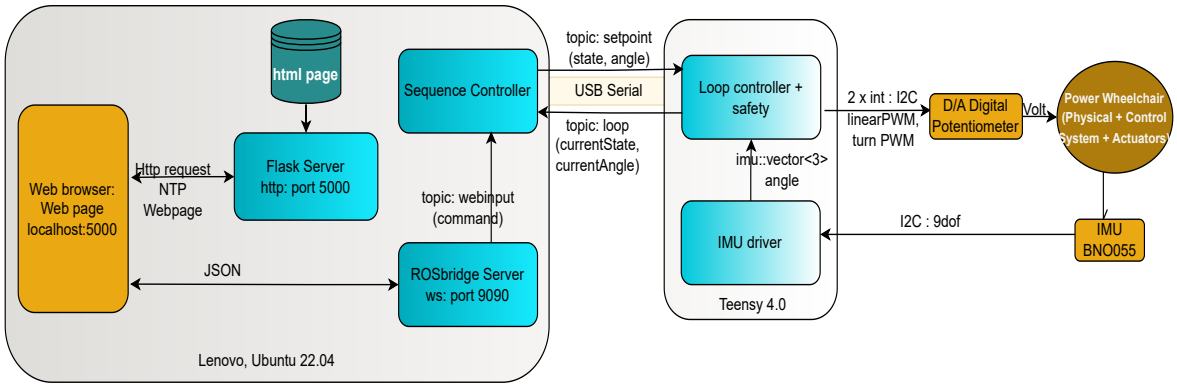


Figure 5.1: Block diagram of the setup used for PID Controller test

The PW has different speed profiles ranging from 1.x to 5.x (where x is the speed setting in a profile). 1.1 is the slowest and 5.4 is the fastest. Each speed profile has 4 levels of speed, so in total there are 20 variations in the speed settings to choose from. The speed of some of these profiles is shown in table 5.1. The PID controller was tested on 3 different profiles: 1.2, 4.1 and 5.1

Profile	Speed (km/h)			
	Forward	Backward	Left	Right
1.2	0.7	0.9	0.4	0.3
2.1	1	1	0.7	0.7
3.1	1.3	1.4	0.8	0.8
4.1	1.6	1.7	1	1
5.1	2	1.4	1	1

Table 5.1: Observed speed at different speed profiles indicated on the PW control system display.

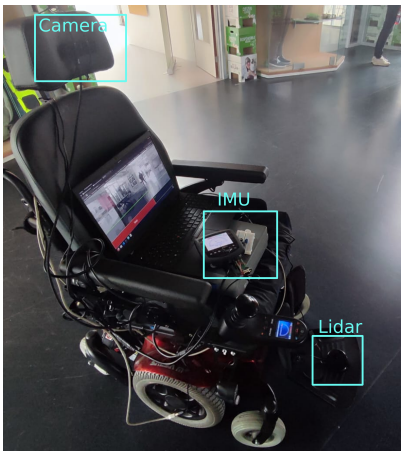


Figure 5.2: Test setup showing the placement of the sensors.

5.1.2 Results

Fig. 5.3 shows the tracking of the setpoint angle by the PW. The first data point shows the rise time of 1.2 seconds from the step change in setpoint angle. The maximum overshoot is indicated in the second data point. The steady-state error is the last data point at 33 seconds.

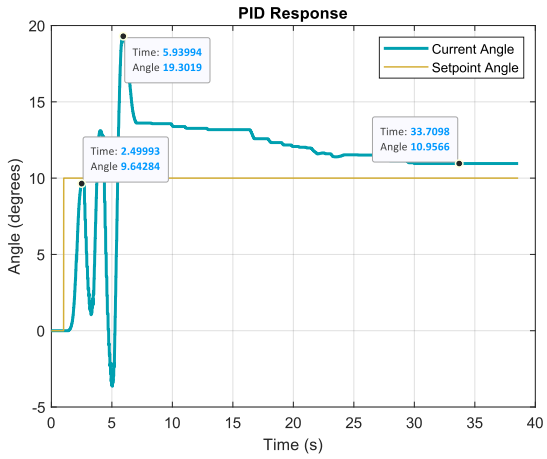


Figure 5.3: PID response with Kp=6, Ki = 0.0005, Kd = 1

The table 5.2 shows the range of PID values for which the PW manages to track the desired angle with a steady state error of ≈ 2°. The K_p, K_i, and K_d values in the table below are valid for speed profiles 4.1 and 5.1.

	min	max
K_p	4	6
K_i	>0	0.0005
K_d	1	2

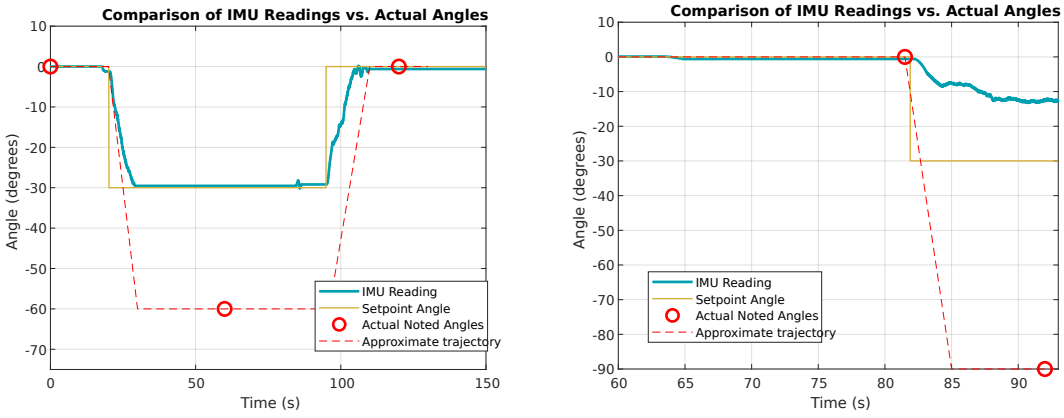
Table 5.2: Range of suitable K_p, K_i, and K_d values.

5.1.3 Evaluation

The PID values were determined through a trial and error process. A PD controller was also tested but the PW could not converge (steady-state error) to the desired angle because of the lack of an integral term. If the integral term was greater than 0.0005, the PW did not converge to the setpoint and became unstable. The rise time of 1.2 seconds indicates that the PW is slow to react due to its large inertia and the delay in response due to the inbuilt control system of the PW. Hence, it overshoots significantly. The maximum overshoot is different for different wheelchair profiles.

Two main challenges of tuning the PID values are listed below:

- **Variety of speed profiles:** Different speed profiles meant that the response time of the wheelchair for the same PID values would be different as the speed of the PW is scaled differently for each speed profile. Finding suitable PID values for slower speed profiles was more challenging due to the wheelchair's large inertia, higher friction, and slower reaction time of the wheelchair. So, wheelchair profiles 4.1 and 5.1 were chosen as they had similar responses for the same PID values.
- **Calibration of the IMU**: The IMU used here requires to be calibrated every time it is powered up. This is quite unsuitable for use outside of the prototype setting, as it has to be physically oriented at different angles to be calibrated before it can be used. If the IMU is not properly calibrated, the output of the IMU is not reliable. This is shown in Fig. 5.4. The solution to this problem was to calibrate the IMU and store the values in EEPROM of the Teensy which can be loaded into the IMU on startup. This still does not guarantee a fully calibrated IMU. The main issue was caused by magnetometers which, when not calibrated resulted in the angle reading to drop while moving. E.g. If turning from 0 to 90 deg, the output would show 0 to 30 deg.



(a) Controller falsely believes that it has reached the setpoint.

(b) The IMU reading indicated that the wheelchair did not turn sufficiently causing it to keep turning.

Figure 5.4: Discrepancy in the IMU readings and actual angle of turning by the PW due to poor IMU calibration.

5.2 Performance test

The performance in terms of latency and jitter was evaluated to indicate how the prototype performs in real-time. These tests are important to characterize the control software's response time to the user input. They help to identify potential bottlenecks and areas for improvement in the communication path. The results obtained in this section might slightly differ when tested on Lattepanda 3 Delta. The sequence control layer was implemented on Lenovo ThinkPad P15v, which houses an Intel i7 processor with 6 cores and 16GB RAM.

5.2.1 Test Setup

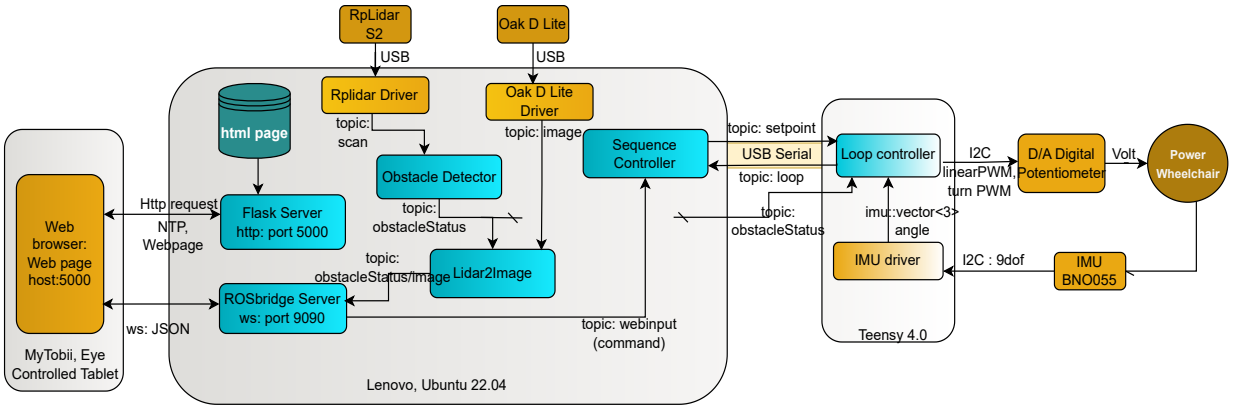
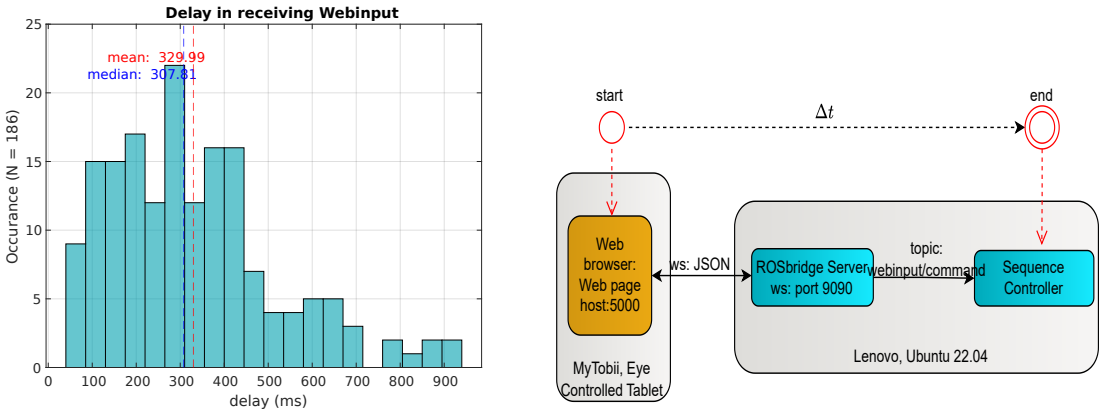


Figure 5.5: Block diagram of the setup

5.2.2 Results

User interface - sequence controller node

Fig. 5.6a shows that 50% of the commands from the user interface device took $\approx 300ms$ to be received by the ROS 2 sequence controller node. Some data points show that the delay can also be greater than $500ms$.



(a) Delay in receiving the command from the user's web client.

(b) Block diagram showing the start and end point for measurement

Figure 5.6: The delay is calculated between the user interface device and the command received by the sequence controller node.

Sequence controller node

The setpoint is published every 1 ms. It is seen in Fig. 5.8a that the maximum delay between receiving a web input command and publishing the setpoint does not exceed 1 ms. This indicates that the node can process and publish within the expected timeframe. The Fig. 5.8b shows that jitter can be 90% of the delay. This is due to the constraint of publishing the setpoint every 1 ms. A better indication of the processing time would be to measure the time between receiving a web input command and the generation of the setpoint.

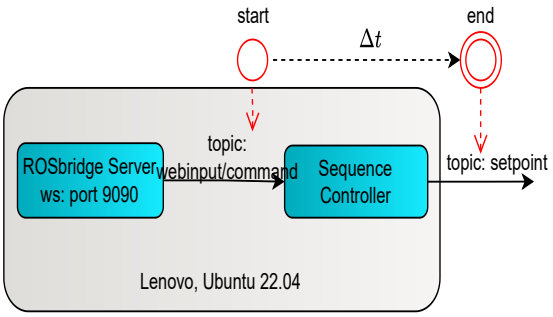
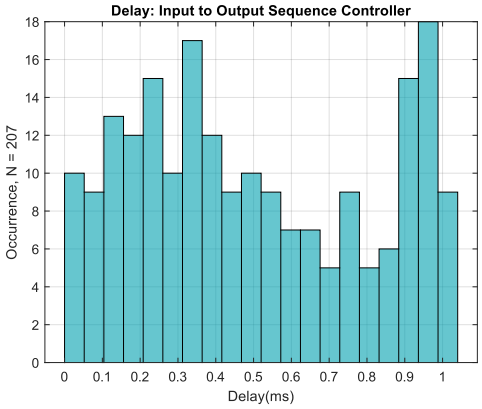
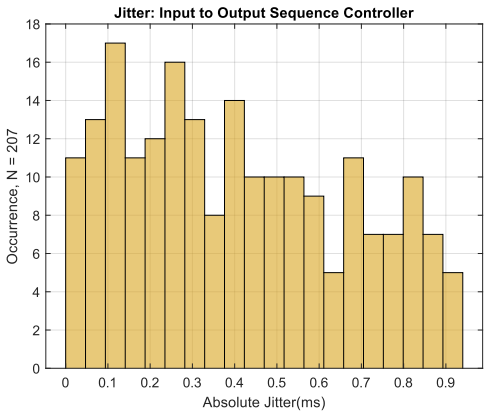


Figure 5.7: The measurement shows how long after receiving the user command it takes for the sequence controller to publish a setpoint.



(a) Processing time for publishing setpoint after the user command is received



(b) Distribution of the jitter

Figure 5.8: The time between receiving a command and publishing the corresponding setpoint was calculated.

Sequence controller and Loop controller node

The delay and jitter in the Round Trip Time (RTT) between publishing the 'setpoint/state' msg and receiving the corresponding 'currentState' msg were measured. The distribution of the RTT in Fig. 5.10a shows that 50% of the response time is within 1.05 ms. The trailing value of Jitter in Fig. 5.10b indicates that it is not bounded. Jitter in approximately 80% of the data points is less than 2 ms.

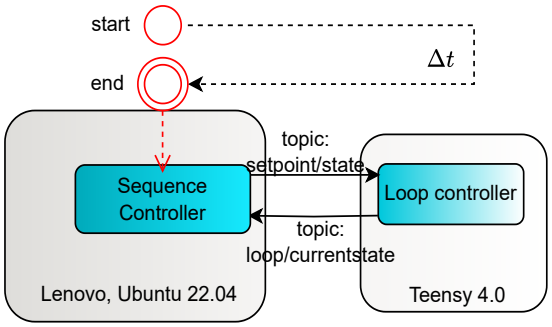
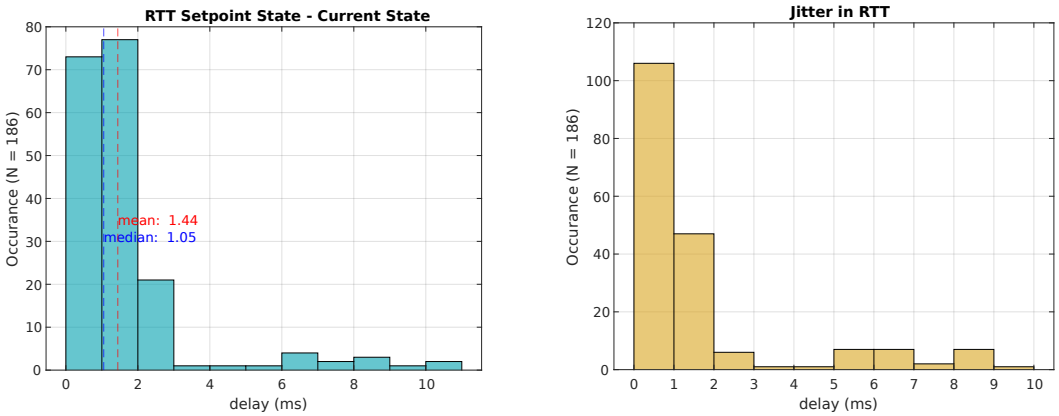


Figure 5.9: Round Trip Time from publishing the setpoint state to receiving the corresponding current state.



(a) RTT between setpoint state and current state

(b) Jitter distribution

Figure 5.10: Distribution of the delay and jitter in RTT between the sequence controller and loop controller node.

5.2.3 Evaluation

The distribution in Fig. 5.6a hides that some delayed packets are sent simultaneously. This can give a wrong indication about the timing. So, here real-time behaviour was not observed. This was expected as it was considered in the non-real-time part while designing the control software architecture. For a vehicle, the response time would vary depending on its speed. In the case of manual operational mode like the one introduced in this work, a 300 ms delay in receiving the 'stop' command if the PW is moving at 1.5km/h could result in a position error of 20 cm. This could lead to unacceptable consequences. This delay is caused due to the time taken by the browser on the user-assistive device to parse JSON messages using the 'roslibjs' library. So, it is device-dependent. To mitigate it a possible solution would be to parse the messages on the server-side(i.e. the computer running the sequence control layer and the web server).

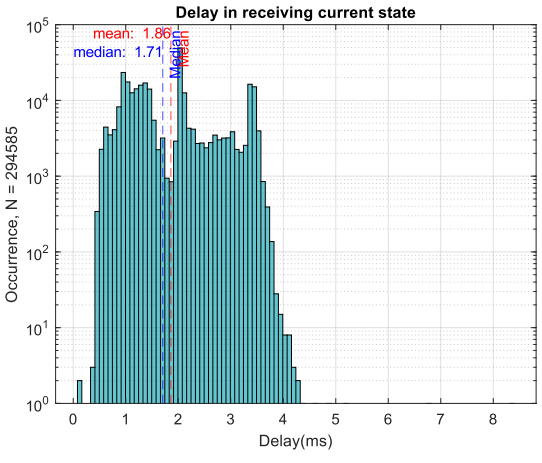


Figure 5.11: The loop controller publishes current_state at 850 hz.

The jitter in Fig. 5.10b is likely caused due to the variability in receiving the messages published by the loop-controller node shown in Fig. 5.11. The maximum delay is approximately 4 times the expected publishing period (1.176ms), except for a couple of outliers which have a delay of 8 ms. The An interesting thing to note was the update frequency of the messages published by the loop controller. The actual observed frequency is different from the ideal.

Message	QOS policy	Ideal Publishing Frequency	Observed Frequency
angleError	Best effort	1000Hz	450Hz
currentAngle	Best effort	1000Hz	450Hz
currentState	Reliable	1000Hz	850Hz
override_flag	Reliable	1000Hz	850Hz

Table 5.3: Frequency of messages published by the loop controller

The reason for this behavior was the constraint limit of 490B MTU. Although the message size was ensured to be small (less than 32 bits except for the use of geometry_msgs/Vector3 msg: 192 bits), the queue size was kept as default at 5. This caused the reliable QOS messages to be prioritized and therefore have a higher frequency than the messages published using the best-effort policy. The solution to this would be to reduce the queue size to 1.

To conclude, the major bottleneck in the response time is due to the delay in getting messages from the user interface device to the sequence control layer.

5.3 User test

Two operational methods were developed for the user to test the prototype. One is a 'continuous motion' control and the other 'step motion' control. For the continuous motion control operation, the PID controller was not used, instead, the commands from the user's web client were translated to control signals for the wheelchair. As long as the user was 'hovering' over a button on the web page, the wheelchair moved in that direction. The second method involved a step-wise control, where if the user 'hovered' over a button then the motion was carried out in steps e.g. if the mouse was on the area of 'Left' or 'Right', then the expected motion should be 10 degrees to the 'left' or 'right'. If 'forward' then the wheelchair would move forward for 2 seconds. The user has to leave the button area and re-enter to do the motion again.

5.3.1 Test Setup

The test setup is shown in Fig. 5.12. The final prototype was not tested on the Lattepanda 3 Delta computer, as it was quite difficult to set it up on the wheelchair with limited space available and the problem of having a portable suitable power supply for the Lattepanda computer. For this test, the user was remotely controlling the PW from a distance and was not seated on the PW itself. The block diagram of the setup is the same as shown in Fig. 5.5.

Figure 5.12: Physical setup for testing with the user

The images from the Oak D Lite had to be severely compressed to reduce latency, hence they appear discolored and have low resolution as shown in the figures below. There was a problem with using Integration Service for the image stream, as it outputs the JPEG compression code instead of the actual image. So, this test was carried out using the ros_bridge package. The lidar feedback (refer Fig. 5.13 is a straight line split into three parts to indicate where the obstacle is. If no obstacle is present within a 0.5m radius then the line is green and if an obstacle is detected it turns red.

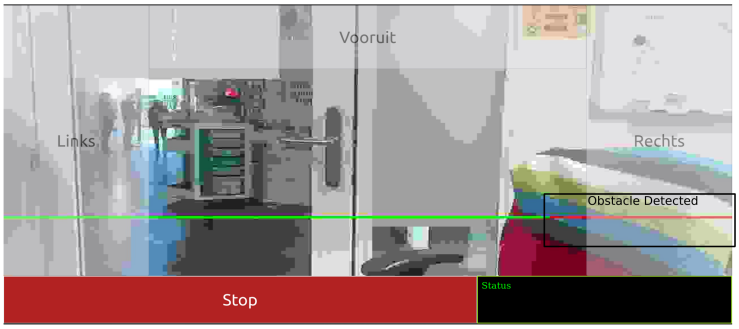


Figure 5.13: User interface as displayed on the user's web browser.

5.3.2 Results

Details of the user test are presented below:

Test Date:	03/05/2023
Time:	16:20
Location:	Design Lab, UT
Who:	Researcher(me), end-user, parent and Ability Tech Lab engineer
Ethical Request number:	240313 refer to Appendix: E

Table 5.4: Testing was done at the UT campus at the Design Lab

The observations are as follows:

1. The joystick emulator fails to connect with the wheelchair if a command from the web browser on the user's tablet is already being sent before the micro-ROS agent is connected to the client(Teensy).
2. The web client did not connect to the UT wifi, so a mobile hotspot network was used.
3. There was considerable delay in the video feedback on the user's tablet, so the resolution of the camera was decreased to the minimum value. This was done using OpenCV's imwrite() function with the JPEG format and JPEG format with a quality setting of 1 (least), the highest value is 100. Even then there was a delay of about 10-15 seconds. This delay was different on different clients, for.e.g. on a Redmi phone it was 2 seconds, and on a Lenovo Laptop (of Ability Tech Lab engineer) it was negligible.
4. There was also an observable delay in the commands from the user to the motion of the wheelchair. Around 1-2 seconds.
5. The PW bumped sideways into a cupboard.
6. If the PW has a relatively higher speed, even though the obstacle is detected in time, the wheelchair takes a fraction of a second to stop.

User experience:

- The user was able to send control commands to the system using his eye-controlled tablet, MyTobii. However, it did not feel responsive to the user because of the huge lag of 15 seconds in the video stream.
- The user did not feel strained while using the UI, this was because it was based on an already-tested UI for the Sjoelen robot. However, he would prefer less manual control, so more preference for autonomous navigation. The user did not have a preference at the time for which operational mode was used, as he was still exploring. It might be difficult to assess the difference in the functionality of the two methods because of the video lag and similar UIs.
- The user and the parent thought the feedback from the lidar was intuitive enough. They would like to see more possibilities.

Because of the delay between the MyTobii tablet and the PWAO CSU, the status of the system was not displayed to the user. This can done with the help of the ErrorPublisher node. If there are no errors then an 'OK' message is sent to the UI on the bottom right corner.

The PW takes approximately 25-30 seconds to settle within 10% of the desired setpoint angle. To improve the response of the PID controller, a cutoff threshold of 2 degrees was applied to the controller, which meant that if the current angle reading was within this threshold the wheelchair would stop turning. This depends on when the most recent IMU reading was available, as the IMU updates at 100 Hz.

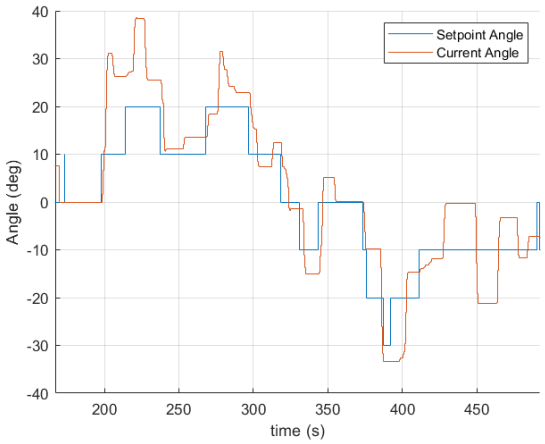
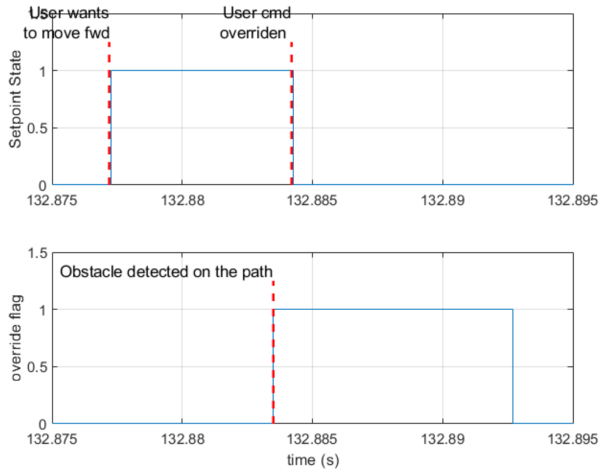


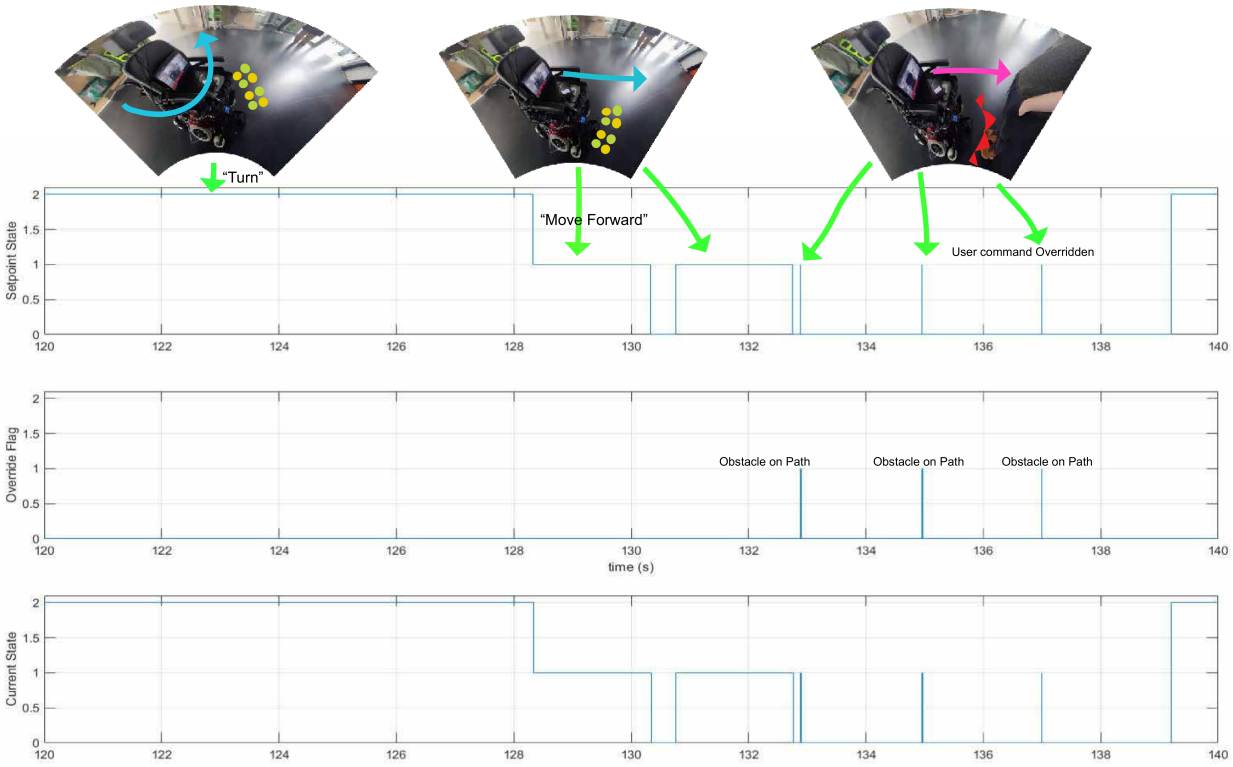
Figure 5.14: PID controller response during user test

The figure below shows that emergency obstacle avoidance, which is applying a break (or stopping), can override the user's command if there is an obstacle in the direction the user directs the PW. Fig. 5.15a shows the zoomed-in portion of an instance when the system decides to stop. The bottom figure shows the dataset with multiple such instances. The threshold of obstacle detection was chosen to be 0.5 m from the center of the lidar based on the width of the wheelchair which was approximately 0.75 m side to side. The setpoint state is the user's desired action, whereas the current state shows the wheelchair state at that time.

The wheelchair images at the top of Fig. 5.15b show the motion of the wheelchair based on the setpoint. The blue arrow indicates that the motion is successfully carried out. The red arrow indicates that the motion is not carried out. Note: Images with the wheelchair were constructed later to illustrate the diagram clearly.



(a) User command being overridden by the system(zoomed)



(b) Part of the dataset generated during the user test is plotted here

Figure 5.15: User interaction with the system

Although the data presented above does verify that obstacle detection and avoidance works, it has some drawbacks which were mentioned in the list of 'observations'. Additionally, it is also important to mention that one 2D lidar is not sufficient to develop a safe and reliable obstacle detection mechanism.

5.3.3 Evaluation

The inferences corresponding to the observations listed in the previously are presented below:

1. If the user already sends a command before the micro-ROS agent is connected, it changes the potentiometer voltage (due to a change in setpoint) which the wheelchair control system recognizes as 'joystick not connected'.
2. The cause of this is not known yet, as in previous tests the web client connected with the server over UT's 'eduroam' network.
3. Although this could be exacerbated due to hotspot, the major reason most likely is the user interface device's hardware and the web browser capabilities. This can be mitigated by removing JavaScript execution from the client's side. Also, a different simple camera, which can be directly connected to the WebSocket server without having to go through the ROS network should be used. Another point to note is that here, Integration Service was not used.
4. The network traffic caused by delayed images along with the the javascript execution used for sending ROS2 commands over WebSocket protocol are the most probable causes.
5. The cupboard was to the left side of the wheelchair and the user was clicking forward, which was detected as safe to go as there was no obstacle in front but the left side handles bumped into plates protruding out of the cupboard, which the lidar did not detect as it was at a lower height. Another cause of this behaviour is how the lidar scan area is sectioned to detect obstacles.
6. The response time of the PW to changing control signals is quite important to keep in mind. The threshold of obstacle detection has to be increased to avoid stopping too close to an obstacle.

It is important to keep in mind the effect of response time on the user experience, especially on a web-based application. Nielsen (1993) states that the user perceives that the system responds instantaneously if the response time is under 0.1ms and can already feel the delay if it is close to 1.0s. This can make the user feel like they are not operating directly on the data (Nielsen, 1993). This can be true for a manual operation of the PW.

6 Conclusions and Recommendations

6.1 Conclusions

A control software architecture that interfaces with the eye-controlled assistive device was proposed in this work. The requirements of the PWAO CSU listed in Chapter 3 laid the foundation for implementing the control software prototype. Design alternatives were explored in Chapter 4 and the architecture was developed on real-time-friendly frameworks and software packages. Chapter 4 also provides the interfaces between different layers of the control software architecture, including the interface with the user-assistive device. The software was developed keeping re-usability in mind which was done with the use of classes and nodes.

A prototype was developed using these interfaces and tested. The results of PID controller for turning motion are presented in Chapter 5, along with real-time performance and user-test. Although the project did not accomplish full autonomous navigation, it established a framework that can serve as a foundation for future prototypes. The PID response showed that it takes a long time for the PW to reach a steady state due to its large inertia. The limitations of using only the IMU for estimating rotation were highlighted.

Performance testing highlighted that it was not desirable to process JSON messages on the MyTobii eye-controlled tablet. Unfortunately, Integration Service by eProsima could not be implemented for transferring images. Special attention has to be paid to adhering to the message constraint set by micro-ROS serial transport. The loop-controller publishing frequency did not match the ideal due to larger than recommended total message sizes. These results provide a direction for improving the system's real-time performance. They will be important when deploying more complex navigation algorithms. Additionally, insights gathered from user feedback indicated a preference for increased autonomous capabilities. The user test revealed that obstacle avoidance has flaws and more tests are needed to find an optimal distance threshold for detecting obstacles. This feedback will help in guiding the next stages of development, ensuring that subsequent iterations of the prototypes are more responsive to user needs.

An important safety issue that was not addressed here was that unplugging Teensy without turning off the power wheelchair causes the PW to become unstable and start moving which can be hazardous.

6.2 Recommendations

The next step would be to implement a real-time visual-inertial odometry algorithm for pose estimation. This will expand the capability of PWAO-CSU to add more operational modes. It would also aid in designing a better-suited controller.

In addition to this, it is quite important to eliminate the processing of data on the client web browser. This can likely be solved by using Integration Service but would require debugging the current issue. An alternative could be to use SocketIO.

De Jong (2023) designed a physical emergency break which can be installed on the PW used in this work. This will improve safety and ensure that the PW does not crash with an object in the surroundings. For improved obstacle detection multiple 2D lidars would be needed. The placement of lidar has to be carefully decided so that it might also be possible to detect drop-offs.

Currently, the micro-ROS client-agent connection is established when the Teensy is plugged in and the Agent is active within 5 seconds. If the connection is not established the Teensy has to be plugged in again. A finite state machine can be programmed on the loop controller to check for the connection and try to reestablish it so that a physical disconnect is not needed.

The user and the father mentioned that the lidar feedback was intuitive enough, but there was no feedback when the user command was overruled. It would be good to point it out to the user when it happens.

A Interview with the stakeholder

Mobility needs and challenges

Q) Can you describe user's mobility needs and challenges in your daily life?

No controlled movement of limbs, involuntary movements. Dyskinesia. Relative good control of head, he can do 'no' movement. But has to be supported to avoid upward movements. Whole body may move to keep his eyes focused.

Goals

Q) What goals might you have in mind that this product could help the user accomplish?

Now its still abstract for the user, he sees possibilities. It comes on the way. Its an interaction thing. Have to push him when it city center, so to do this himself. Go to the park nearby home by himself. Sense of independence. Riding along a line in the house. He wants to move the seats to be more comfortable. Seat tailor made. It will be unnatural to sit in the same way whole day.

Q) What problems do you expect it can solve for you?

It would be important to have seat tumbled , caretakers have to be called but he should be able to do it himself. The basic support system if can be lifted by himself. If we could do more things on the wheelchair, changing diapers is intense. Change diapers on wheelchair instead of lifting him up and putting him on the stretcher

Environment

Q) Can you describe the physical environment in which the product will be used? What types of environments does the user most frequently navigate (e.g., indoors, outdoors, uneven terrain)?

Relatively safe environments. What situations have value, still controllable like park (no cars, no traffic, not many people)

Q) Are there specific challenges related to the terrain in your daily life that the wheelchair should address?

Quite good wheelchair adapted space at home. But we have doors and furniture's. It should know what a chair is and what a door is. Stairs detection.

Usage

Q) How long does the user use the power wheelchair? How long is the expectation to use the new product?

Not that long, he had a manual wheelchair. 5 years he has a power wheelchair. He sits it in the whole day, depends on people assisting him. He like to walk around. Wheelchair bus, or on bike. He walks atleast 20 mins or more. His wish would be the same as us. Charge it at night.

Q) How many power wheelchairs has been used so far?

from since he was a baby. Need a bigger one when he grows.

Q) What are the challenges for you and the user when transitioning from one wheelchair to other?

Don't do that a lot. The old one is for spare. In the beginning he has to get used to. Always have to adjust, he is used to old one.

Q) What does the user like about the current wheelchair? What do you want to retain? And replace?

Sitting the whole day is convenient. The current one does not have joystick but have to push it. A joystick would be more convenient. Now there is electrical support, but just a switch to drive (like an accelerator) driving a car without cruise control

About the autonomous wheelchair

Q) What words would you use to describe the product?

the user can go where he wants, safety, autonomy.

Q) Are specific portions of the product more critical than others for performance, reliability, security, safety, availability, or other characteristics?

Whole package, everything should be according to standards , lets say if he is in a park and stuck then the caregiver should get a notification.

Q) What type of control interface would you(the user and the user's father) prefer (e.g., joystick, voice commands, touch screen)?

not really thought about it, but joystick sounds good. Also, control from a distance.

Q) What autonomous features do you believe would be most beneficial? (e.g., obstacle avoidance, navigation, autonomous docking)? Are there some features that would benefit you as a caretaker/parent?

you need all, emergency stopper and obstacle avoidance. The most challenging is going from A to B. arrive fresh where you want to be

Q) How much control and autonomy would you and the user prefer the wheelchair to have in various situations?

can you imagine what it is? He has to find out on the way. He wants to succeed in the eye controlling . When the user was 1.5 year old, he tried power wheelchair and he was quite good at it. It was brain controlled, it was intensive.

Q) How would you like the wheelchair to communicate with you (the user's father and the user) (e.g., voice commands, visual displays, audible alerts)?

useful by phone. Really some urgency then text or spoken words.

Q) Are there specific situations or conditions where you'd like the wheelchair to provide alerts or notifications?

when the user feels there is a need. Or if there is a system failure then it should notify the user's father

Q) How would you judge whether this product is a success?

anything which would help him do on this own. Is not the end result but it helpful and tells you about what comes later

Other questions

Q) What other assistive devices have you and the user tried and how where they helpful? What did you not like about them? What did you like about them?

Tobii dynavox, i12 switching to 13, computer. Communication: writings most of the time, facebook, emails. Accessibility of whatsapp will be there. Text to speech. He understand spoken text better.

Q) Are there any constraints or rules to which the product must conform?

Depends on how accessible the platform is. It's a medical device. Last study year they decided to go for add on.

B Schematic of the joystick emulator

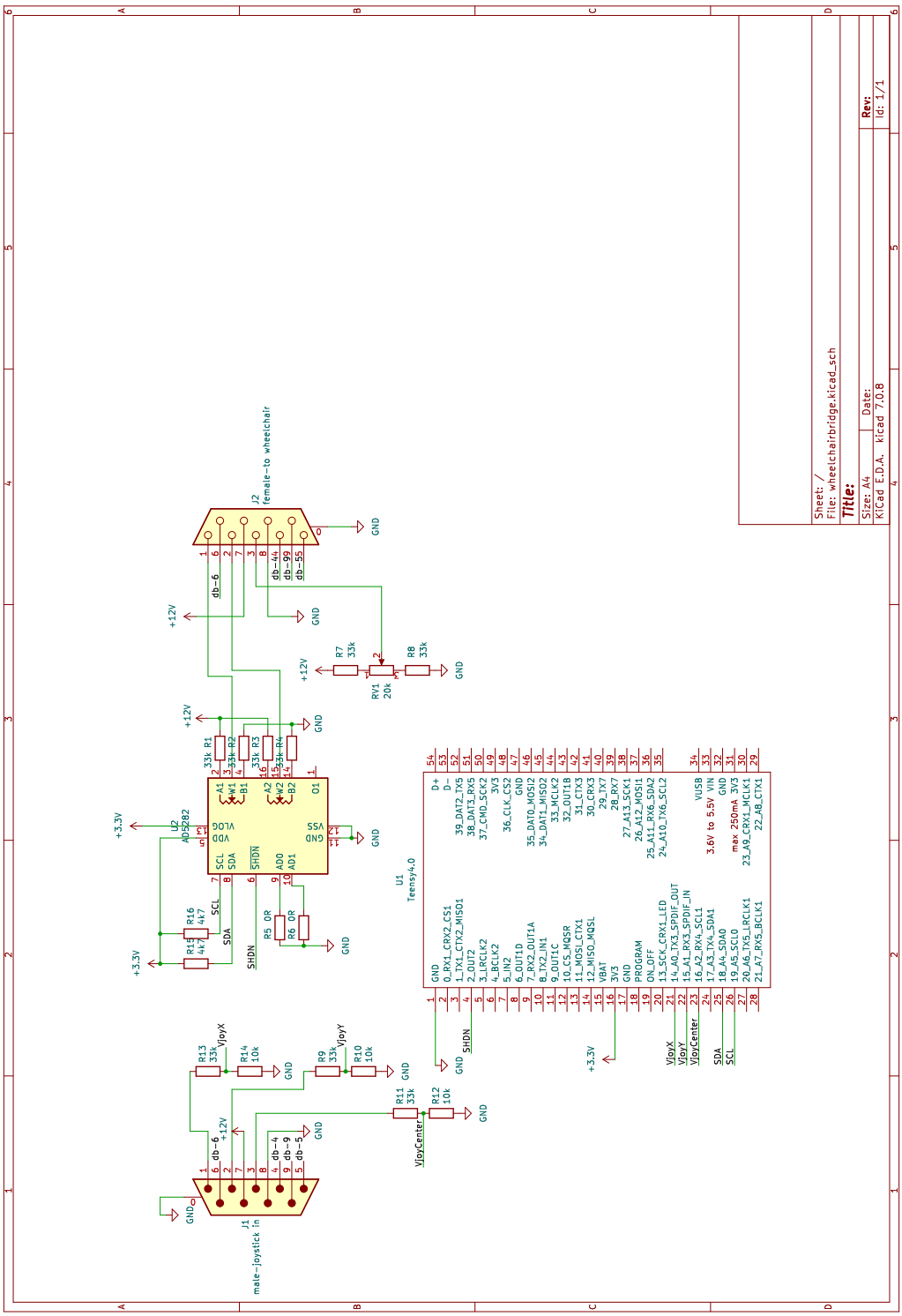


Figure B.1: Wheelchair bridge schematic

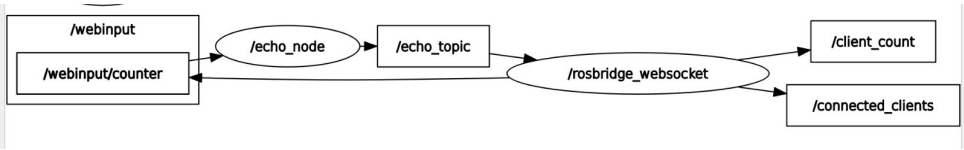
C Additional test results

C.1 Rosbridge vs eProsima Integration service

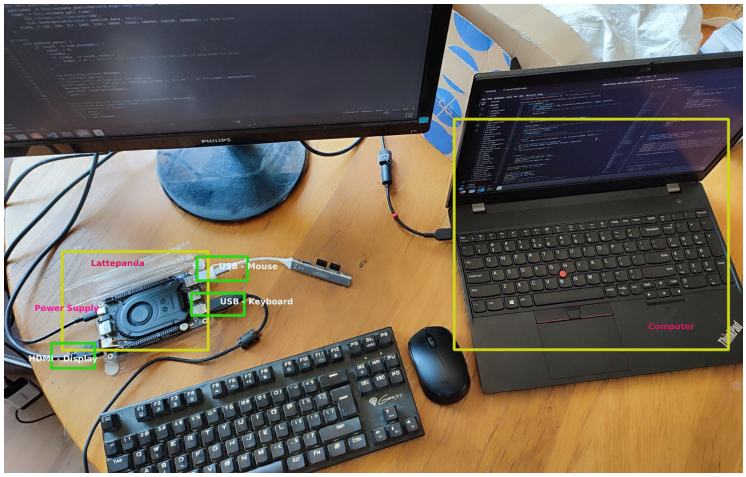
C.1.1 Test Setup

To determine whether Rosbridge or eProsima Integration service should be used for communication between the user's assistive device and the PWAO CSU it was important to test the performance of the two alternatives. Two tests were conducted. The first test compared the performance metrics: Round trip time and Jitter. The second test was done to test the limitation on scalability of the message size, and here the performance metrics were: One way latency and jitter.

Test1: For the first test, a standard laptop running google chrome was used for the web client. A 1000 messages of 28B size on a timer of 300ms were sent from the web client to lattepanda 3 delta which hosted the server and the ros2 node. The computers communicated via LAN on port 5000, which is typically used by 'flask server'. This size of message was chosen as it typically reflects the command message size, which the user sends by clicking a button. The Fig. below shows the ROS graph and the physical test setup used.



(a) ROS node and topic graph showing the test setup for rosbridge setup. Similar setup was used for Integration service



(b) Physical Test Setup

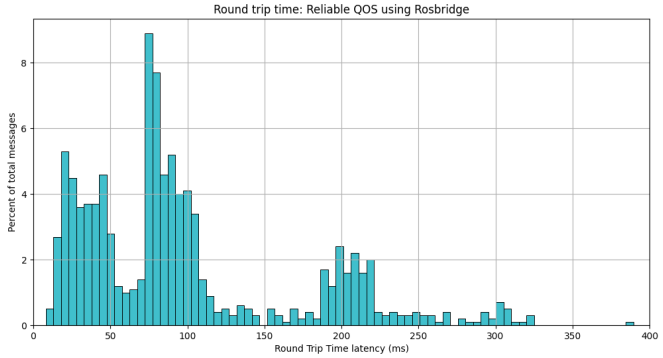
Figure C.1: ROS graph and test setup

Test2: The scalability in the size of messages, under Rosbridge and Integration service was investigated in this test. Here, the same physical setup as shown in Fig. C.1b was used. The ROS graph is different as only one-way communication was tested. This was done to understand the behaviour of the two software (rosbridge and integration service) when data of varying sizes has to be made available on the user interface. This data can vary from a small string (like an error message) to a sensor message of large size.e.g. image data. This test also provides a constraint on the message size to be used. This is quite important as some sensor messages like images can be quite big. In this setup, one way latency was measured from ROS2 node(on lattepanda) to web client on the laptop. This is chosen as typically sensor messages are only sent one way

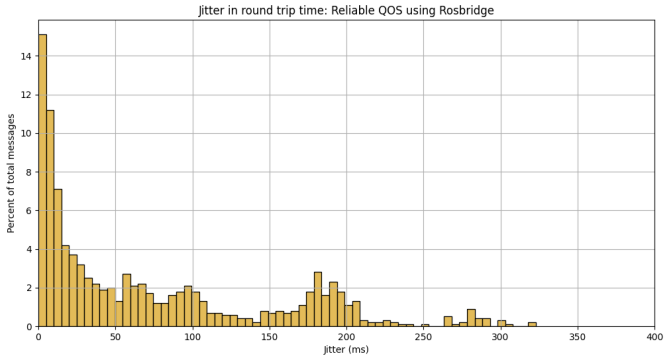
and don't require any acknowledgement to be received. The sample size of each message size is 1000 and the publishing frequency was 100ms.

C.1.2 Results

Test 1



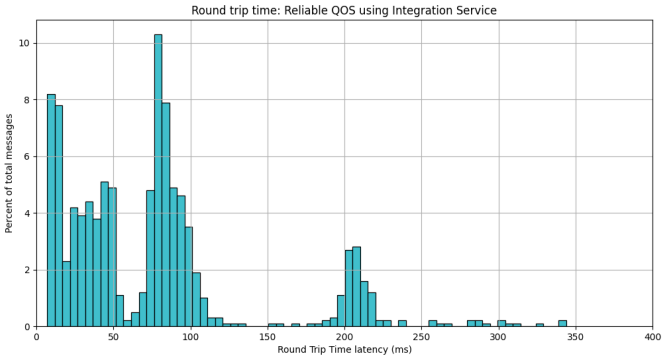
(a) Round Trip time for a 28B message size



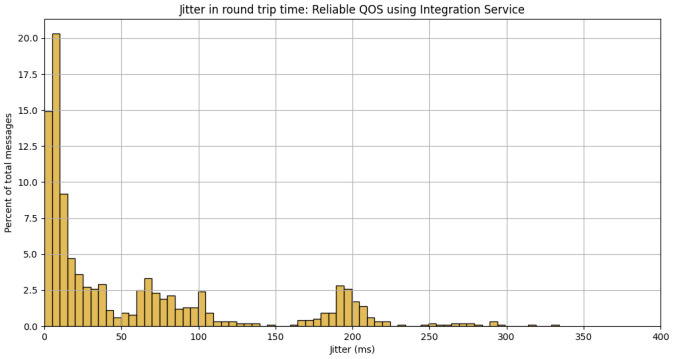
(b) Jitter for round trip time for a 28byte

Figure C.2: Latency and Jitter using Rosbridge

Fig. C.2, shows the latency for a 28B size message for a round trip communication. This test gives the estimate of how much time it would take if a user were to send a command to ros2 node and receive an acknowledgement message. The interval for the bins in the histogram are 5ms. The Y axis represents the percent of messages and, the X axis, time in ms. As can be seen from the graph C.2a majority of the messages take under 100ms to be sent and received by the web client. To find the one way latency, ie. time taken from user interface device to ros2 node, the round trip time (RTT) can be divided by half. So, most of the messages would be received under 50ms by the ROS2 node. The jitter tells how much variance can be expected. Approximately, 50% of the messages have a variance less than 50ms in round trip time, where 14% of the messages are in the 0-5ms bin.



(a) Round Trip time for a 28B message size

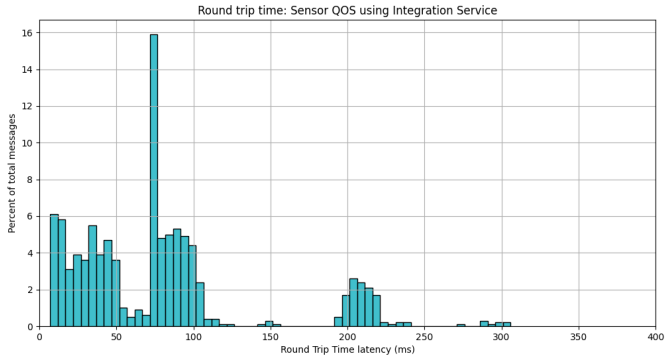


(b) Jitter for round trip time for a 28byte

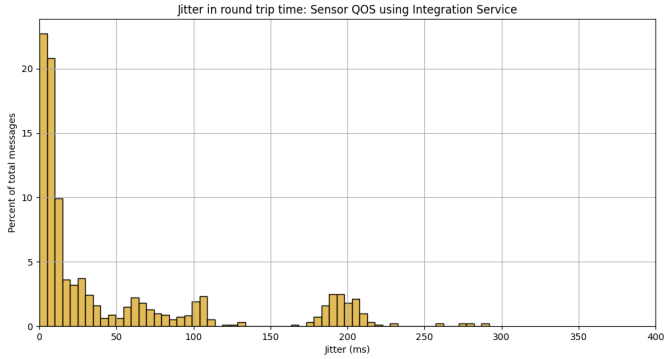
Figure C.3: Performance of Integration service with reliable QOS

As the rosbridge package does not support different Quality of Service settings(QOS) yet, it was compared against Integration service using the same Reliability policy and queue size, which are 'reliable' and a queue size of 10. From the graph C.3, it can be seen that there is considerable improvement in the round trip latency. It is evident from the graph C.3b where 35% of the messages are under 0-10ms as compared to 25% in rosbridge. Another notable observation is that the maximum time taken is lower for Integration service by slightly more than 50ms.

Clearly, Integration service performs better. As the QOS of integration service can be changed, the RELIABLE QOS results were compared with SENSOR QOS, which is a QOS configuration provided by ROS2. It had BEST EFFORT for reliablity and the queue size is 5. From the Fig. C.4, the improvement is seen in the maximum value of latency. So, all of the messages using SENSOR QOS are received back in approximately 300ms compared to 350ms under RELIABLE QOS. Jitter is less if SENSOR QOS is used as compared to RELIABLE QOS setting.



(a) RTT using Sensor QOS

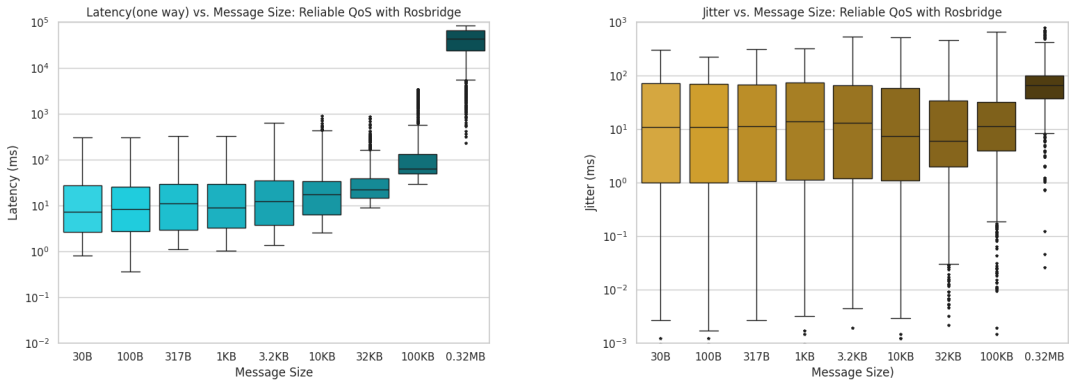


(b) Jitter

Figure C.4: Performance of Integration service using Sensor QOS

Test 2: Scalability

Fig. C.5 shows the one way latency and jitter in publishing messages of size varying from 30B to 0.32 MB when using rosbridge for websocket transport. This is compared to the results obtained when using Integration service plugin, with a RELIABLE QOS and SENSOR QOS setting (Fig. C.6 and C.7 respectively). These are plotted using boxplots.



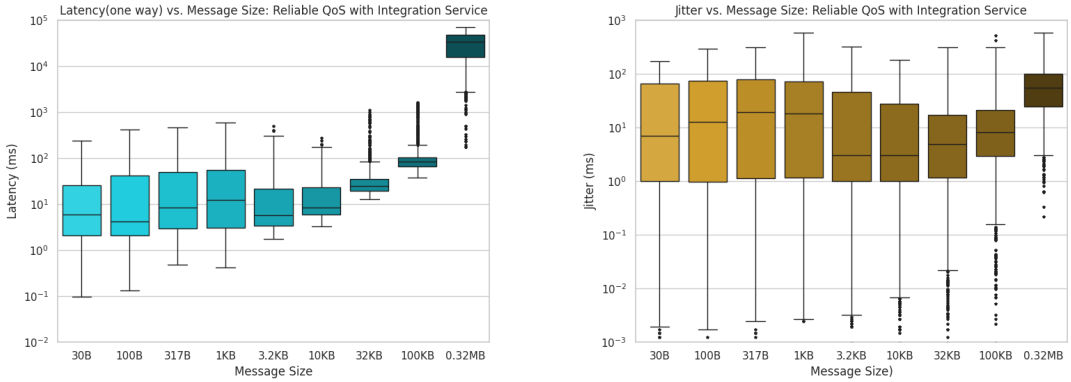
(a) Latency (one way)

(b) Jitter

Figure C.5: Latency and Jitter in one way communication vs message size for rosbridge

The performance of both Integration service and Rosbridge is similar for larger size of messages. Also, another point to note is that as the latency increases the range of jitter decreases.

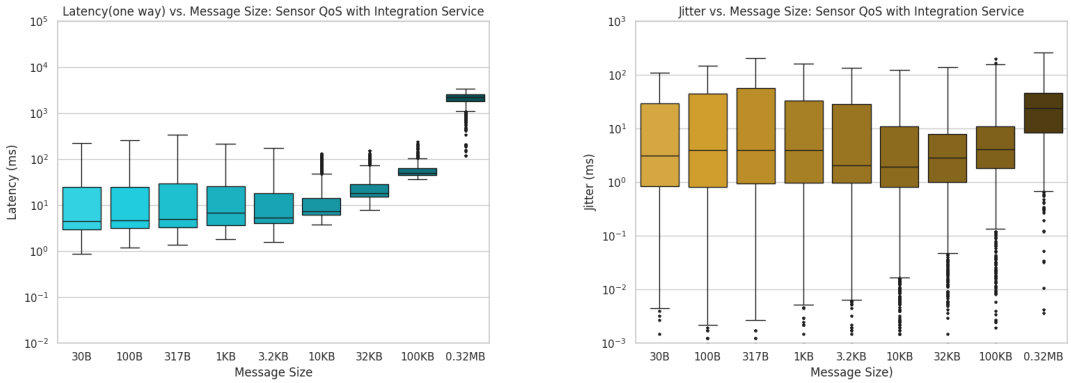
Here, a low jitter and high latency means that the latency would not vary much and most of the messages will be received very late.



(a) Latency (one way) **(b)** Jitter

Figure C.6: Latency and Jitter in one way communication vs message size for Integration service using Reliable QOS

There is considerable improvement in the latency using SENSOR QOS, which places the emphasis on using Integration service for the flexibility in QOS implementation. Also, these results indicate that there is a need to compress image sizes and constrain them to be under 100KB and there is a need for optimization of transport of large image sizes.



(a) Latency (one way) **(b)** Jitter

Figure C.7: Latency and Jitter in one way communication vs message size for Integration service using Sensor QOS

C.2 Results of Accerion Sensor Test

Triton is a sensor that provides SLAM (simultaneous localization and mapping) algorithm. The sensor was tested at the user's house to determine if the floor was suitable for Triton to be used on. The sensor was placed 4 cm above the ground and moved in a zig-zag manner across an imaginary straight line. The sensor measures how many times it was swiped across the line and how many times it detected this. This floor test was recommended to be performed by the company. The results of the test are shown below.

Test ID	Test description	Measured value	Criteria	Result (pass/fail)
i	**Floor qualification (North)**			
i.1	Score for 0.2 - 0.6 m/s:	13/13	>90%	pass
i.2	Score for 0.6 - 1.0 m/s:	35/35	>90%	pass
i.3	Score for 1.0 - 1.4 m/s:	22/22	>90%	pass
i.4	Score for 1.4+ m/s:	10/10	>90%	pass
ii	**Floor qualification (South)**			
ii.1	Score for 0.2 - 0.6 m/s:	9/10	>90%	pass
ii.2	Score for 0.6 - 1.0 m/s:	20/20	>90%	pass
ii.3	Score for 1.0 - 1.4 m/s:	10/10	>90%	pass
ii.4	Score for 1.4+ m/s:	9/11	>90%	fail
iii	**Floor qualification (West)**			
iii.1	Score for 0.2 - 0.6 m/s:	10/11	>90%	pass
iii.2	Score for 0.6 - 1.0 m/s:	19/19	>90%	pass
iii.3	Score for 1.0 - 1.4 m/s:	20/20	>90%	pass
iii.4	Score for 1.4+ m/s:	9/10	>90%	pass
iiii	**Floor qualification (East)**			
iiii.1	Score for 0.2 - 0.6 m/s:	10/10	>90%	pass
iiii.2	Score for 0.6 - 1.0 m/s:	16/17	>90%	pass
iiii.3	Score for 1.0 - 1.4 m/s:	11/11	>90%	pass
iiii.4	Score for 1.4+ m/s:	9/10	>90%	pass

Table C.1: Floor Test for Triton. The measured value is no. of passes detected.

The SLAM algorithm could not be tested on the ROS 1 distribution as the device was not detected on Ubuntu 18.04. The floor test was carried out on Windows platform.

D Running the Demo

D.1 List of Items

1. Computer with Ubuntu 22.04
2. Joystick emulator
3. Teensy + IMU
4. RPLidar S2
5. Oak D Lite camera

D.2 Prerequisites

Make sure you have ROS 2 Desktop version installed on your computer. This thesis used ROS 2 Humble distribution on Ubuntu 22.04. It should be possible to use the software on future ROS 2 releases.

Install the micro-ROS setup. The instructions for the same are available on the official micro-ROS website: Teensy-microROS

Download the official ROS 2 driver packages for Oak-D-Lite camera and RPLidar S2.

The source code and detailed instructions are available on the Ability Lab Github page. Download the code and extract it. Move the src folder to a ROS 2 Workspace source folder. Keep the run_demo.sh script outside the ROS 2 Workspace

First, connect all the peripherals to your Computer

- If using Lattepanda, a USB splitter is required to connect the keyboard and mouse.
- On the Lattepanda if you do not see the Ubuntu login and instead you see a boot screen(see image below). Follow the instructions here.

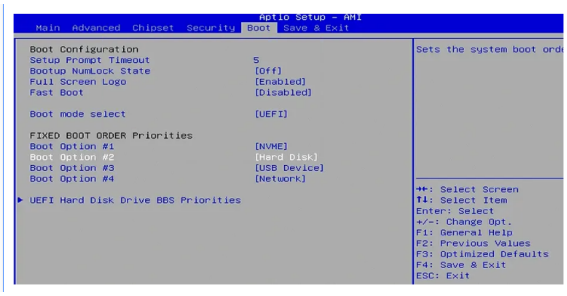


Figure D.1: Boot Manager Screen

Make sure the Teensy (with the imu) is horizontal(flat on the surface) and fixed firmly so it doesn't move.

D.3 Instructions to Use

1. Open Terminal in the folder where you installed the run_demo script.
2. Run **./run_demo.sh**
3. Make sure the teensy is connected to the wheelchair when you run this script.
4. Open a browser on the same computer and navigate to localhost:5000 to check if a connection has been established.
5. Check if the teensy has a yellow light continuously or blinking. If blinking then disconnect Teensy and connect again. If steady yellow light then it should be connected.
6. Turn on the wheelchair and test. Make sure the wheelchair profile is on profile:'5.1' or '4.1'

7. To restart, close all terminal windows. Turn of wheelchair, plug out the teensy usb(white on). Then run the script again(from instruction 1). Connect the white usb back to the teensy and wait for it to output a steady yellow light. Make sure to refresh the browser page and it should work.

Warning: Turn Off The Wheelchair Before Plugging Off The Teensy USB Cable.

E Ethical Review Request

UNIVERSITY OF TWENTE.

Computer & Information Sciences (CIS)

240313 REQUEST FOR ETHICAL REVIEW

Request nr:	240313	Intro form:	7 - Introduction
Researcher:	Magar, M.B. (M-SC)	Middle form:	10 - Computer & Information Science (CIS)
Supervisor:	Dertien, E.C. (EEMCS-RAM)	Outro form:	5 - Submission
Reviewer:	de Willigen, P. (EEMCS-BFD)		
Status:	Positive advice by Reviewer		
Date of application:	15-04-2024		
Request version:	1		

0. GENERAL

0.1. Personal details

Student/employee number: s2689529
Initials: M.B.
First name: Mrinal
Last name: Magar
Email : m.b.magar@student.utwente.nl
Education/department:
Faculty:

0.2. Project title

Control Software Architecture for Autonomous Power Wheelchair

0.3. Summary

Autonomous wheelchair is an essential assistive technology that can accommodate a segment of disabled individuals who cannot operate manual or electric-powered wheelchairs. This technology is necessary for our client with severe spasms, currently controlling his computer (and world) through an eye-tracker. A previous iteration of this project involved using eye-controlled input to steer the wheelchair. This setup was limited to just using an eye-tracker for control.

Currently, there are self driving wheelchairs but the problem with most of them being they were dependent (developed) on some propriety hardware which also keep changing as the power wheelchair manufacturers come up with newer designs. So, the idea for this particular thesis is to separate the hardware of the wheelchair from the software. This way it is possible to use this software on different wheelchairs from different manufacturers.

The main design objective of my thesis is:
To design a control software architecture, with real-time constraints, for developing a self driving power wheelchair.

Most of my work was exploratory work on what middleware and software packages to use and testing to see which works better. And also designing a PID controller, a state controller and a UI for user to control the wheelchair. So, currently the wheelchair can rotate and move forward and has an obstacle detection system using

a 2D Lidar. The last part of the the thesis is to test it with the user. This will fulfill two goals:
1. To validate that the software developed can interface with the user's assistive device
2. To validate the PID controller and get feedback from the user about the timing requirements and other improvements

0.4. Start date (estimated) and end date (estimated) for your research project

Start date: 22-04-2024

End date: 30-05-2024

0.5. If additional researchers (students and/or staff) will be involved in carrying out this research, please name them: [Please include full name and email]

Full name	Email

0.6. In which context will you conduct this research?

Master's thesis

0.7. Please select your supervisor

Dertien, E.C. (EEMCS-RAM)

0.8. Please select an ethical committee

Computer & Information Sciences (CIS)

1. GENERAL

1.1. PRE-ADVICE: Did you already consult an ethics adviser about this request?

No, and I am *NOT* a student in CreaTe or I-Tech

1.2. PRIVACY, GDPR, AND POSSIBLE NEED FOR DPIA: Does the research include any possible access to, gathering, or use, or publication of data that can be traced back to specific individuals, directly or, for instance, by combining data from multiple sources? Or is it possible that you will accidentally access or publish Personal Identifiable Information (PII)?

Yes, and we follow the rules on processing of personal data, including acquiring explicit consent for processing PII (besides possibly the consent for participating in research) and including a possibly necessary GDPR registration.

1.3. RESEARCH DOMAIN: Regarding the nature of your research, does one or more of the following statements apply to your research?
the research is in a potentially medical domain such as illness, assessment and diagnosis, prevention, cure, or carethe research addresses a health outcomethe research gathers health datathe research involves a hospital or other medical settingthe research may be potentially medical for some (other) reason

No, the research is not medical, health related, or close-to-medical in any way whatsoever

2. HUMAN RESEARCH PARTICIPANTS

2.1. HUMAN PARTICIPANT RESEARCH: Does the research include
a) active involvement of human research participants during the research, and/or
b) gathering new data from individuals
such as measurements or responses from interviewees, survey respondents, participants, informants, or simply people whose data is measured because they are present in a certain place at a certain time?

Yes, my research falls under "research with human participants'

3. RESEARCH POPULATION

3.1. RESEARCH POPULATION: Please provide a brief description of the intended research population, including inclusion and exclusion criteria, number of participants, and recruitment strategies.

```
Main research participants will be the client for which the wheelchair software
prototype will be developed, along with his father.
```

3.2. LACK OF CAPACITY TO CONSENT: Do you have participants who are formally NOT able to give informed consent?

```
No, all participants have the capacity to consent
```

3.3. VULNERABLE PARTICIPANTS: Does your research target vulnerable participants such as focusing on specific ethnic groups, people in another country, minor (<16 years), people with physical or cognitive impairments (regardless of their capacity to consent), people under institutional care (e.g., nursing homes, hospitals, prisons), or any other particular group that may be more vulnerable than other people in the general population?

```
Yes
```

3.4. POWER RELATIONS: Does your research target participants somehow dependent on, or in a subordinate position to the researcher (e.g., students or relatives)?

```
No
```

3.5. APPROACH TO VULNERABLE PARTICIPANTS: Please elaborate in what way you include participants that are vulnerable in any way, and how you will take this into account in your plans. Take into account your answers in the previous questions in this section.

```
Since the patient suffers from locked-in syndrome. He is unable to speak or move.
He can however answer questions using his eyes.
I will communicate with his father, who will be the intermediatary person. I will
make sure to include the participant only when his father is present, this way
it will be safer for the patient. Questions will be asked as yes/no or multiple
choice questions as much as possible, which the patient can answer by looking or
looking away from my hand. For open questions which can't be asked in a yes/no or
multiple choice way, either the question will be asked to the father.
```

4. RESEARCH PROCEDURE AND RISKS

4.1. RESEARCH TYPES: Which of the following research types do you employ in your research?

```
Interviewing and surveys: paper/online questionnaires, survey, face to face or
online interview, focus group

Participating in non-experiment activity: can be formative evaluation of
prototypes, but more in general providing artificial tasks, including triggering
stimuli and tasks to elicit observable behavior and responses; measured with e.g.
observations, interviews, and manual or automated data collection
```

4.2. CONTEXT OF REAL LIFE ACTIVITIES: Do the activities of participants that people do, included in your research, include activities in a real life setting?

```
No
```

4.3. MATERIALS, PROTOTYPES AND DESIGNS: Do the activities include interaction with a prototype, design, mockup, product, interaction technology, etc?

```
Yes
```

4.4. ASSIGNING TASKS TO PARTICIPANTS: Do the research procedures include activities performed specifically for the sake of the research?

```
Yes
```

4.5. LOCATION: Where will the research activity take place?

At the university, in design lab.

4.6. TIME INVESTMENT: How much time will each participant spend?

1 session, 1 hour max

4.7. DESCRIPTION OF RESEARCH PROCEDURE: What is the research procedure, in terms of setting, tasks, activities, content, and stimuli?

The research proceduce will most likey be checking if the user can use his assistive device to control the wheelchair.
Next, the test will be to validate the software prototype. To check if it behaves the way it is supposed (does it rotate and move forward. does it detect obstacle) to and get feedback on improvements to be made from the user. The feedback will be along the lines of, if the control is fast enough, or too slow, or too fast. The user will control the wheelchair either remotely or sitting on the wheelchair (if feasible, but this will be discussed with the father beforehand and only with his approval). It will take place in open space in design lab. The wheelchair speed will be kept minimum, which is actually slower than walking pace. There is also an emergeny button which will turn off the wheelchair if it is needed.

4.8. MEASURES: What measurements, recording tools, discussion topics will you employ?

The feedback will be in written form, in a text document.
Other measurement include data from lidar and imu measurement to plot the behaviour of the wheelchair. This is not related to the user but just for measuring the performance of the software.

4.9. RISK OF ADVERSE EFFECTS: Is there a risk for adverse (or: negative) effects of the research for certain participants, and how do you deal with these risks?

No

4.10. BURDEN TO THE PARTICIPANT: Are there other short-term or long-term burdens and/or risks to the participants?

No

4.11. ACCIDENTAL FINDINGS: Does the method used allow for making an accidental, diagnostic finding that the experimental participant might have to be informed about?

No, the method does not allow for this possibility

4.12. COVID19: Are you aware of departmental/UT rules regarding experimentation under COVID19 and will you follow them?

Yes I know the rules and will follow them
Yes I know the rules and will follow them
Keep distance as much as possible, no touching

5. (DE-)BRIEFING, DECEPTION & CONSENT PROCEDURE

5.1. BRIEFING. Will you inform potential research participants (and/or their legal representatives in case of legally non-competent participants) completely about the aims, activities, burdens and risks (such as to their health and well-being) of the research and about other relevant information before the decide to take part in the research? How will you do this?

Yes, participants are fully briefed beforehand

5.2. Please explain

No information is withheld.
Before the interview, the partipants will be informed about the estimated duration, location, goal and procedure of the experiment.
This will be done in an email before an appointment is made.

5.3. If applicable, upload your information letter as a PDF

```
Information Letter.pdf
```

5.4. INFORMATION ON WITHDRAWAL OF CONSENT. Will you inform potential research participants (and/or their legal representatives in case of legally non-competent participants) clearly that they can withdraw from the research at any time without explanation/justification?

```
Yes
```

5.5. DECEPTION. Will you use any Deception in the research procedure? How, and why?

```
No, we will not use any deception
```

5.6. DEBRIEFING: Will the research procedure involve a debriefing after participation, and how will you do this?

```
No
```

5.7. FREEDOM TO PARTICIPATE: Are the participants completely free to participate in the research and to withdraw from participation whenever they wish and for whatever reason?

```
Yes, and we clearly communicate this to them
```

5.8. DIRECT CONSENT OR PROXY CONSENT: Who will provide the consent?

```
Legal representative
```

5.9. TYPE OF CONSENT: Which type of consent will you use?

```
Oral, recorded consent prior to participation
```

5.11. CONSENT FOR FUTURE USE: Will you keep and reuse the newly collected data for future research, and do you obtain adequate consent for this?

```
No, I will only use the data for this research
```

5.12. PERSONAL DATA: Will you gather new personally identifiable data, about the research participants, and does the consent information also address consent for Personally Indentifiable Information (PII), separate from and in addition to consent for research participation and research data collection and use?

```
No, I do not gather personally identifiable data aside from (possibly) the consent
form itself
```

5.13. PUBLICATION OF THE DATA: Will you publish (some of) the newly collected data, and do you obtain adequate consent for this?

```
Yes, I will make (some of) the data publicly available, and I do obtain explicit
consent
```

5.14. REWARDS: Will participants receive any rewards, incentives or payments for participating in the research?

```
No, the participants will not receive any reward
```

6. PRE-EXISTING DATA

6.1. PRE-EXISTING DATA: Will the research involve the inclusion, combination, use, and/or analysis of already existing data sets about people?

```
No
```

7. AI TECHNOLOGY

7.1. AI TECHNOLOGY: Will the project develop AI technology, or will the project involve the deployment and/or use of AI technology for practical applications?

```
No
```

8. CYBERSECURITY

8.1. CYBERSECURITY: Will the research involve any cybersecurity or online privacy issues, such as the possible discovery of security vulnerabilities, experiments with malicious software (e.g., computer viruses), or the discovery and investigation of illegal activities on the Internet?

No

9. UNINTENDED CONSEQUENCES, MISUSE, AND APPLICATION RISKS

9.1. MISUSE: Is it reasonable to anticipate that the research will provide knowledge, products, or technologies that could be intentionally used to threaten, or non-intentionally result in threats, to public health and safety, crops and other plants, animals, the environment, or material infrastructure?

No

9.2. INCLUSIVITY AND SOCIAL INJUSTICE: Is a disproportionally negative impact foreseeable on certain groups of users or non-users, for example, people of a certain age, gender, sexual orientation, social class, race, ethnicity, religion, political orientation, culture, or disability, creating or reinforcing social injustices?

No

9.3. MILITARY APPLICATION: Does your research or prototype have military/police/defense applications?

No

10. OTHER ETHICAL ISSUES

10.1. CONFLICTS OF INTEREST: Do any of the parties involved in overseeing or carrying out the research have a potential conflict of interest?

No

10.2. RISKS TO THE RESEARCHER: Will the study expose the researcher to any risks (e.g. when collecting data in potentially dangerous environments or through dangerous activities, when dealing with sensitive or distressing topics, or when working in a setting that may pose ‘lone worker’ risks)?

No

10.3. OTHER POTENTIAL ETHICAL ISSUES: Do you anticipate any other ethical issues in your research project that have not been previously noted in this application?

No

11. CLOSURE

11.1. I have answered all questions truthful and complete

Yes

12. COMMENTS

No comments have been added to this request.

13. CONCLUSION

Status: Positive advice by Reviewer

Bibliography

Beauregard, B (2017), Arduino-PID-Library, Accessed 22-04-2024.
https://github.com/br3ttb/Arduino-PID-Library?tab=readme-ov-file

Boode, A. (2018), *On the automation of periodic hard real-time processes: a graph-theoretical approach*, Phd thesis - research ut, graduation ut, University of Twente, Netherlands, doi:10.3990/1.9789036545518.

Borgolte, U., H. Hoyer, C. Bühler, H. Heck and R. Hoelper (1998), Architectural Concepts of a Semi-autonomous Wheelchair, Technical report.

Bourhis, G. and Y. Agostini (1998), The Vahm Robotized Wheelchair: System Architecture and Human-Machine Interaction, Technical report.

Broenink, J. F. and Y. Ni (2012), Model-driven robot-software design using integrated models and co-simulation, in *Proceedings - 2012 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2012*, doi:10.1109/SAMOS.2012.6404197.

Bruyninckx, H. (2002), Real-time and embedded guide, *KU Leuven, Mechanical Engineering*.

De Jong, S. (2023), *Development of an Eye-controllable Self-Driving Wheelchair*, Master's thesis, University of Twente.

eProsima (2021), eProsima Integration Service, [Accessed 22-04-2024].
https://integration-service.docs.eprosima.com/en/latest/introduction.html

eProsima (2024), Micro XRCE-DDS compared to rosserial — micro.ros.org, [Accessed 20-06-2024].
https://micro.ros.org/docs/concepts/middleware/rosserial/

Faulconbridge, R. and M. Ryan (2014), *Systems Engineering Practice*, Argos Press, ISBN 9781921138072.
https://books.google.nl/books?id=zDh9ngEACAAJ

Fehr, L., W. E. Langbein and S. B. Skaar (2000), Adequacy of power wheelchair control interfaces for persons with severe disabilities: A clinical survey, *Journal of Rehabilitation Research and Development*.

García, J. C., M. Marrón-Romera, A. Melino, C. Losada-Gutiérrez, J. M. Rodríguez and A. Fazakas (2023), Filling the Gap between Research and Market: Portable Architecture for an Intelligent Autonomous Wheelchair, *International Journal of Environmental Research and Public Health*.

Henderson, M., S. Kelly, R. Horne, M. Gillham, M. Pepper and J. M. Capron (2014), Powered wheelchair platform for assistive technology development, in *Proceedings - 2014 International Conference on Emerging Security Technologies, EST 2014*, Institute of Electrical and Electronics Engineers Inc., pp. 52–56, ISBN 9781479970070, doi:10.1109/EST.2014.20.

Interaction Design Foundation - IxDF (2017), Stakeholder Maps - Keep the Important People Happy, [Accessed 26-06-2024].
https://www.interaction-design.org/literature/article/stakeholder-maps-keep-the-important-people-happy

Leaman, J. and H. M. La (2017), A Comprehensive Review of Smart Wheelchairs: Past, Present, and Future, doi:10.1109/THMS.2017.2706727.

Mitchell, R. K., B. R. Agle and D. J. Wood (1997), Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts, *Academy of*

management review.

Nielsen, J. (1993), *Usability Engineering*, Elsevier Science.
https://books.google.nl/books?id=DBOowF7LqIQC

Object Management Group (2024), What is DDS?,
https://www.dds-foundation.org/what-is-dds-3/, [Accessed 20-05-2024].

Open Robotics (2024), Tutorials, https://docs.ros.org/en/humble/index.html, [Accessed 25-04-2024].

Simpson, R. C., S. P. Levine, D. A. Bell, L. A. Jaros, Y. Koren and J. Borenstein (1998), NavChair: An assistive wheelchair navigation system with automatic adaptation, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1458, ISSN 16113349, doi:10.1007/bfb0055982.