ETH zürich

Networking for Educational Robotics: Design and Development of the Smart System Platform

Master's Thesis

Author: Nicolas Y.C. Triebold
Supervisors: Prof. Elizabeth Tilley
Prof. Chris Rogers

16. March 2025

GLOBAL HEALTH ENGINEERING

Tufts UNIVERSITY School of Engineering
Center for Engineering Education and Outreach

Networking for Educational Robotics: Design and Development of the Smart System Platform

Master's Thesis

Author : Nicolas Y.C. Triebold
Supervisors : Prof. Elizabeth Tilley
Prof. Chris Rogers

March 16, 2025

ACKNOWLEDGEMENTS

The completion of this thesis would not have been possible without the support, guidance and encouragement of many people to whom I am deeply grateful.

First and foremost, I would like to thank my parents for their unwavering support throughout this endeavour. Despite the distance from home, their encouragement and belief in me provided the strength and motivation needed to persevere.

I am profoundly grateful to my friends, both old and new, near and far, for their encouragement and companionship that kept me grounded throughout this journey.

I would like to extend my sincere gratitude to Liz for her invaluable guidance throughout my Master's degree and support in shaping this thesis. Her mentorship has been instrumental in making all of this possible.

In addition, I would like to especially thank Chris for his hospitality in hosting me at the Tufts Center for Engineering Education and Outreach (CEEO) and for his continued support, guidance and insights which have been instrumental in fostering my research and personal growth throughout this process.

I would also like to thank Milan and Jenn, who, together with Chris, constituted defence committee and provided me with valuable insights and feedback.

Further, I would like to express my gratitude to all the members of the CEEO, many of whom I consider dear friends. Collaborating with them has been a privilege, and their work, feedback and support have contributed to this project and greatly enriched my visit. To everyone who has been a part of this journey - thank you!

ABSTRACT

In this thesis the design and development of a peer-to-peer networking approach for Smart Motors, an educational robotics platform, is explored. Furthermore, in order to structure and facilitate the approach, a framework called the Smart System Platform is developed. The focus of the research lies on the design and development of the ESP-NOW based networking protocol, how to make this networking capability accessible through the supporting Smart System Platform framework, with an emphasis on simplicity, accessibility and modularity, and how students and educational researchers can use this platform and technology.

The networking approach developed consists of a custom networking library based on ESP-NOW that enables direct communication between any ESP32-based microcontrollers, which are referred to as Smart Modules. The library introduces several features, including an address book system, a message structure with different types and subtypes, and the ability to send larger data packets. Building on this, a custom Smart System Platform add-on library has been created to provide specific commands and handlers to control and configure the Smart Modules. The Smart System Platform has been designed to support and enable access to these networking capabilities, serving as an overarching educational robotics system architecture built around the networking capabilities. Based on this platform, a minimum viable product has been developed in the form of a number of platform components, including hardware component concepts, software, documentation and development support sites such as the GitHub page, guides and website, and a suite of networking-focused development and management tools, including a custom web-based integrated development environment.

The developed networking approach and platform were tested using various methods, including robustness tests, experiments with range and received signal strength indicators, and two hackathons with college students to assess their utilisation of the capabilities provided and the application concepts they devised. The developed networking approach has also been adopted by students and incorporated into other ongoing research projects, such as the Smart Playground project. The outcomes of this research demonstrate the viability of the peer-to-peer communication system and its potential for educational applications. However, the research

also highlights shortcomings in the usability and accessibility of the support tools developed. The research further proposes areas for future development, such as improving the quality and accessibility of support tools and materials. This work contributes to the field of educational robotics by providing a flexible, accessible networking solution and support platform that can be built upon to enhance interactive learning environments. The integration of peer-to-peer communication between educational robotics modules and the facilitating Smart System Platform can support and enable novel innovative approaches to STEM education, as well as the development of more complex, collaborative and engaging learning experiences.

Contents

Nomenclature

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
MIT	Massachusetts Institute of Technology
Tufts	Tufts University
UCI	University of California, Irvine
CEEO	Tufts University Center for Engineering Education and Outreach
D-MAVT	Department of Mechanical and Process Engineering
GHE	Global Health Engineering

Ack	Acknowledgement
AI	Artificial Intelligence
CAD	Computer-aided design
Cmd	Command
CPU	Central processing unit
ESP IDF	Espressif IoT Development Framework
GUI	Graphical user interface
HTML	Hypertext Markup Language
IDE	Integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
Inf	Information
IoT	Internet of things
IRQ	Interrupt request
ISTE	International Society for Technology in Education
ISM	industrial, scientific, and medical

ISO	International Standards Organisation
JSON	JavaScript Object Notation
LLM	Large language model
MC	Microcontroller
MCB	Microcontroller-Board, in this work specifically referring to the Seeed Studio XIAO ESP32C3 and C6 development boards
ML	Machine Learning
msg	Message
NGSS	The Next Generation Science Standards
OSI	Open Systems Interconnection
PCB	Printed circuit board
PEP 8	Python Enhancement Proposal 8
REPL	Read Evaluate Print Loop
RSSI	Received signal strength indicator
SDGs	The Sustainable Development Goals
std	Standard-deviation
STEM	Science, technology, engineering and mathematics
STEAM	Science, technology, engineering, arts and mathematics
SoC	System on Chip
TSMC	Taiwan Semiconductor Manufacturing Company Limited
Qmd	Quarto
Rmd	R Markdown
UART	Universal Asynchronous Receiver-Transmitter
UI	User interface
UN	The United Nations
UX	User experience

GLOSSARY

Term	Definition
Arduino	Italian open-source hardware and software company, with a focus on microcontrollers
Arduino BASIC	C-based Arduino programming language
Arduino IDE	Arduino BASIC integrated Development Environment (IDE)
C++	High-level, C-based, general purpose programming language
Chrome	Web-browser developed by Google
CSS (Cascading Style Sheets)	Style sheet language for styling of documents written in markup languages, such as HTML
Dahal Board	Specific custom printed circuit board for Smart Motors
DeepL	Translation and Language Artificial Intelligence (AI) Platform
Espressif Systems	Semiconductor company
ESP32	Family of system-on-chip microcontrollers
ESP32C3	Series of ESP32-family system-on-chip microcontroller
ESP32C6	Series of ESP32-family system-on-chip microcontroller
Gimp	Open-source graphic editor
GitHub	Cloud-based development, hosting and Git-enabled version control platform

Grove Ecosystem	Seeed Studio's modular ecosystem of electronic components using standardised connectors
HTML (Hypertext Markup Language)	Standard markup language for web content, such as websites
Internet of things	Concept of digitally interconnected computing devices and technology
ISTE Standards	Standards for the use of technology in teaching and learning
JetBrains	Software development company specialised on IDEs
JetBrains PyCharm	Python IDE
JetBrains Rider	General code IDE
K-8	Kindergarten to 8th grade
K-12	Kindergarten to 12th grade
LEGO	Trademark and brand name of the LEGO Group's interlocking brick building system
The LEGO Group	Danish toy manufacturer
LEGO Education	The LEGO Group's education division, focused on educational methods and technology
LEGO Mindstorms	Educational robotics kit developed by the LEGO Group in co-operation with the MIT Media Lab and Tufts CEEO
MAC Address	Fixed unique address assigned to a network interface (such as Wi-Fi or Ethernet)
Massachusetts Institute of Technology	Technical university in Massachusetts
Microcontroller	Small computer on a single integrated circuit
MicroPython	Implementation of Python for microcontrollers
MIT Media Lab	Multidisciplinary research laboratory at MIT

Next Generation Science Standards	Description
Overleaf	Web-based LaTeX editor
OpenAI ChatGPT	Large language model (LLM)
Perplexity AI	Consolidated LLM and AI assistant
positCloud	Web-based RStudio IDE
Python	High-level, general purpose programming language
PyScript	Web-based development platform for web-applications using Python
Quarto	File format for dynamic generation of output mixing code and text
R	Programming language for statistical analysis
R Markdown	File format for dynamic generation of output mixing code and text
RStudio	R IDE
Seeed Studio	Open source hardware company
Seeed Studio XIAO ESP32C3	A specific ESP32C3-series system-on-chip microcontroller-based development board
Seeed Studio XIAO ESP32C6	A specific ESP32C6-series system-on-chip microcontroller-based development board
Smart Motors	Trainable educational robotics system
Sustainable Development Goals (SDGs)	17 goals of the United Nations 2030 Agenda for Sustainable Development
Thonny	Python IDE

1 Introduction

1.1 Background and Motivation

1.1.1 Science, STEM and STEAM Education

Science education has become an integral component of modern general education (Atkin & Black, 2003), and, according to the National Research Council *et al.* (1995) and National Research Foundation and National Science Board (2022), is widely acknowledged as crucial for cultivating a scientifically literate population, capable of comprehending the governing principles of our universe and engaging with our increasingly intricate science- and technology-driven world (Bybee, 2013). A scientifically literate populace further promotes the contribution to national progress in a variety of areas, such as in economic and scientific development. (Atkin & Black, 2003)

> "Science is more than a body of knowledge; it is a way of thinking." - Carl Sagan (Sagan, 1995)

Historically, the primary focus of K-12 science education has been the imparting of knowledge in various scientific subjects, such as biology, chemistry, maths, and physics, in an isolated and purely factual manner. However, in recent times, there has been an increasing emphasis on cultivating critical thinking skills, problem-solving abilities, and the capacity to discern the interconnected relationships between diverse scientific disciplines. (Atkin & Black, 2003; National Research Council *et al.*, 1995; "Next Generation Science Standards", 2013)

> "The principle goal of education in the schools should be creating men and women who are capable of doing new things, not simply repeating what other generations have done; men and women who are creative, inventive and discoverers, who can be critical and verify, and not accept, everything they are offered." - Jean Piaget (Duckworth, 1964)

Examples of this line of thinking are initiatives such as STEM (Science, Technology, Engineering, and Mathematics) education, which build on top of general science education. STEM

education goes beyond teaching scientific subjects in an isolated fashion, instead emphasises interdisciplinary applications and real-world problem-solving, as outlined by Abdi *et al.* (2024), Blackley and Howell (2015), Bybee (2013), and Xie *et al.* (2015). DeVille (2024) further states that STEM is fundamentally connected to everything in our society, and that science education must help students see those connections. More recently, STEAM (Science, Technology, Engineering, Arts, and Mathematics) education has aimed to take STEM a step further by also including the arts. This holistic approach further focuses on the importance of creativity and design thinking in innovation, and the pedagogy thereof (Bequette & Bequette, 2012; Connor *et al.*, 2015; Dolgopolovas & Dagienė, 2021; Marín-Marín *et al.*, 2021). Research, such as that by Jamali *et al.* (2023), Samsudın *et al.* (2020), and Yakman and Lee (2012) has shown that both STEM and STEAM approaches can significantly enhance learning, as measured by an improvement in soft skills such as creativity, critical thinking, problem-solving, collaboration and communication, and create an increased interest in STEM fields, and as such can be said to improve the quality of education.

1.1.2 Education Technology and SDG 4

Educational frameworks, methods, and technology have been constantly evolving, as seen by the aforementioned STEM and STEAM education approaches and reflected in the Next Generation Science Standards (NGSS) ("Next Generation Science Standards", 2013). In terms of methodology, there has been a shift from basic factual learning to more engaging and interactive approaches (Atkin & Black, 2003; "Next Generation Science Standards", 2013). Promising examples of such approaches include hands-on learning (Satterthwait, 2010; Vesilind & Jones, 1996), project-based learning (Kokotsaki *et al.*, 2016; Markula & Aksela, 2022; "Project-Based Learning", 2014; Samsudın *et al.*, 2020), learning through play (Parker *et al.*, 2022; Weisberg *et al.*, 2013; Zosh *et al.*, 2017), as well as group-based collaborative learning (Brennan *et al.*, 2023; Tonkal *et al.*, 2024), which have all shown to have a positive impact on learning, and consequently on quality of education. Engineering technology, and educational robotics in particular, have played a major role in the development and implementation of many of these methods, as outlined by Khine (2017). In the context of educational robotics, research has generally demonstrated an enhancement in learning outcomes, especially in the domain of STEM-related concepts. (Afari & Khine, 2017; Benitti, 2012; LEGO Education, n.d.).

As described by Sapounidis and Alimisis (2020), educational robotics first emerged out of Papert's theory of learning called constructionism (Papert & Harel, 1991), based on Piaget's ideas on constructivism (Von Glasersfeld, 1982), which can be summarised as learning-by-making and connects the concepts of learning and play. This formed the basis for further educational

robotics technology, such as the LEGO Mindstorms Robotic Invention Kit (Mindell *et al.*, 2000), which was developed in cooperation by the MIT Media and the LEGO Group's education division (LEGO Education, formerly LEGO Dacta). A more recent example is "LEGO Education Science" (n.d.), an NGSS-based K-8 hands-on science curriculum that provides both lesson plans and the educational technology, in form of LEGO bricks and a new educational robotics kit, to support science education. (LEGO Education, n.d.)

Such initiatives align closely with the United Nations Sustainable Development Goals (SDGs) (United Nations Department of Economic and Social Affairs, n.d.-a), specifically SDG 4 - Quality Education (United Nations Department of Economic and Social Affairs, n.d.-b), which is part of the 17 SDGs adopted by all UN member states in 2015 as part of the 2030 Agenda for Sustainable Development United Nations Department of Economics and Social Affairs (n.d.). The International Society for Technology in Education (ISTE) has also developed standards that align with SDG 4, which further emphasises the role of technology in achieving quality education for all (ISTE, n.d.; "Technology in education", 2023).

However, while the emergence of educational initiatives, such as STEM and STEAM, as well as the implementation of new methods and technologies, such as educational robotics, have resulted in advancement in educational quality, there still remains considerable work to be done to ensure the widespread accessibility and inclusivity of these technologies for all (Sapounidis & Alimisis, 2020), in accordance with all tenets of SDG 4. Particularly in the case of educational robotics. Accessibility factors such as cost, usability and high entry barriers pose challenges, as stated by Dahal (2024), Dahal *et al.* (2023), and Johnson (2012).

1.1.3 The Smart Motors Project

In response to the recognised need for affordable robotic technologies in the educational sector (Khine, 2017), Smart Motors, a low-cost, open-source solution for educational robotics, has been developed by Dahal (2024) at the Tufts Center for Engineering Education and Outreach (CEEO). As the name suggests, the concept is centred around a motor, though other forms of outputs, such as visual, audio, haptic etc., could also be utilised. A further component is the user interface, enabling interaction with the motor, which in the latest version of the motor includes a screen, three buttons and a potentiometer. Another component of the system is the input, in the latest version of the Smart Motor this is either a built in accelerometer or some other analogue sensor that can be plugged into the system. All these components are connected to and by a microcontroller. The Smart Motor is a self-contained, trainable educational robotics system for STEM classrooms. It has been designed to be trained and programmed directly on the

unit itself, thereby eliminating the need for additional software and hardware to program the motor. This makes it intuitively usable out of the box without the need for any other supporting software or hardware, and independent of any infrastructure, with the exception of electricity to charge the battery. Its design is intended to be as accessible and intuitive as possible, with the objective of reducing its acquisition and production cost and lowering the entry barriers for use. The motor has two modes of operation, in the training mode, the user can program and train the motor output based on certain sensor input. The result of this training is then used to determine the relationship between input and output in the second mode, the play mode. Dahal (2024, p. 3) characterises them as a "Trainable Motor for Storytelling: in response to the challenge of introducing robotics to a classroom, especially with limited access to computers and Wi-Fi". The Smart Motor was designed for Elementary and Middle School students (K-8). The primary teaching objective being the instruction of reinforcement learning to help teach Machine Learning (ML) and Artificial Intelligence (AI) using a hands-on and playful learning approach enabled by the Smart Motors. In the current approach, each motor is used as an individual self-contained unit. Though given the incorporation of general-purpose components and open-source firmware and software, the system possesses a plethora of inherent capabilities, including Bluetooth and Wi-Fi networking, and support for various other inputs and outputs. Furthermore, with a microcontroller at its core, the system can be easily reprogrammed to implement different logic or programs, and adapted to the use of various additional hardware components, among other advantages. This versatility underscores the system's considerable potential and the prospect of expanding its range of applications. (Dahal, 2024)

1.2 Justification, Goals and Research Questions

The potential and untapped capabilities of Smart Motors are explored, particularly in the context of implementing a networking approach that facilitates communication and interaction between Smart Motors and other modules. The aim is to develop a networking approach and supporting material, to enable and explore the application of these modules for educational projects and the ways in which educational researchers can use these technologies to develop new innovative ideas for hands-on learning, learning through play and collaborative learning. The main focus of the thesis lies on the networking, tool and system development and testing.

In this thesis the Smart Systems Platform (SSP) is introduced, a networking-enabled engineering education platform, based on Smart Motors. With the development of networking at its core, the platform is designed to support, enable and make this capability accessible for easy use and development. The platform is a minimum viable product (MVP) that demonstrates the

Smart Motors capabilities and how they can be leveraged into a more broad and general educational robotics system. The platform's overarching system design encompasses a platform architecture approach, which considers the overall system design, nomenclature, components (hardware and software), supporting systems and documentation, usability in form of the of development tools and management suite, and more to facilitate interaction, usability and further reduce entry barriers. The platform also includes a roadmap for future work, while considering the core motivation of the Smart Motors project of being a low cost and accessible and open source educational resource. The advantages of such a system could be numerous. It would serve to simplify the development of new educational projects and approaches by researchers and educators. Furthermore, it could play a pivotal role in enhancing the quality and accessibility of education. This is due to its potential to function as a foundation for an educational robotics kit and as a pedagogical instrument in the classroom, where the networking capabilities specifically might enable new and novel approaches. Additionally, it would contribute to general accessibility on an institutional or regional level in terms of cost and sourcing, thanks to its low-cost and open-source nature.

The research questions are:

1. How can we design and implement a peer-to-peer networking approach for Smart Motor-based devices to enable inter-module communication?
2. How can these networking capabilities be made accessible? What framework or architecture is necessary for a Smart System Platform to support and enable access and effective use of these capabilities?
3. Given access to this networking capability and its support Smart System Platform framework, how do students and educational researchers utilise it, and what do they come up with?

The central objective of the thesis is the development of networking capabilities for Smart Motors. In addition, the support, enablement and ease of use of this capability is examined, which has resulted in the creation and development of the Smart System Platform as a MVP. Furthermore, the developed system and capabilities are then tested with a small focus group of college students, examining how they can use the developed capabilities and platform.

2 Methodology

The research in this thesis can be divided into three distinct phases, Section 2.1 Phase 1 consists of the analysis of the Smart Motor system, the literature and technology review, and the design of the networking and platform architecture, including a framework and guiding principles. Section 2.2 Phase 2 describes the development of the networking approach and the supporting Smart System Platform, including some of its key features such as the GitHub page, the website, and various development and management tools. Section 2.3 Phase 3 outlines the methodology for testing, validating and deploying the designed platform and networking capabilities.

2.1 Phase 1: Analysis and Design

In a first step, detailed in Section 2.1.1, the existing Smart Motor project, its system design and the technology used are analysed and dissected. In Section 2.1.2 all components of the Smart Motor v3 are examined. Additionally, other systems and technologies are also examined, particularly with regard to networking solutions, focusing on, but not limited to, the components examined as part of the Smart Motors. Based on this analysis, a network design concept was developed, which is described in Section 2.1.3. Furthermore, based on considerations of how to support and provide accessibility to the system and its networking capabilities, a concept for an overarching system architecture was designed and is presented in Section 2.1.4.

2.1.1 Smart Motor Technology Analysis and Review

As defined by Dahal (2024), the main components of the Smart Motor concept and how they interact are shown in Figure 2.1. They consist of the microcontroller (MC) as the brain of the system in the centre, a sensor input, a motor output and a user interface (UI) that provides both input and output and enables user-system interaction.

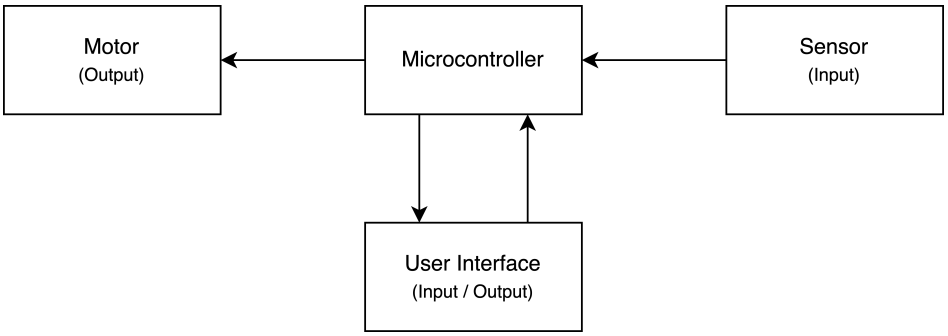


Figure 2.1. General schematic of a Smart Motor system, adapted from Dahal (2024, p. 19)

There are two different modes of operation, *Training* and *Playing*. In the training mode, the user matches a given sensor input S_n to a manually set motor output (either position or speed) P_n using the UI, where n is the index of the input data pair. In play mode, the motor determines the output of the motor P_{output} based on the current sensor input $S_{current}$ using a simplified nearest-neighbour algorithm described in Equation 2.1 with the Machine learning (ML) training data provided by the user during the training mode, if $n \geq 1$.

$$P_{\text{output}} = P_{arg\ min_{x \in 1,\ldots,n} |S_{\text{current}} - S_x|} \tag{2.1}$$

The simple algorithm compares the current sensor input $S_{current}$ with all n stored sensor inputs S_n from training and solves for which S_x the Euclidean distance $|S_{current} - S_x|$ is minimal. As motor output P_n and sensor input S_n are always entered in pairs and therefore linked, the motor output P_x is linked to the sensor input S_x and is then used to set the output of the motor accordingly.

This section focuses on the design, technology and function of the Smart Motor, with particular emphasis on the latest iteration of the Smart Motor, the Smart Motor v3 (Dahal, 2024, p. 38).

Hardware Design

Physically, the Smart Motor v3 is cube-shaped, with its various components contained within the shell, with only the various physical interfaces visible on the surface. An OLED screen, a button and a potentiometer are located on the front face. There are two more buttons on the left, a Grove-compatible sensor port (Seeed Studio, 2023) on the right, and a USB-C port on the bottom. The motor, either a servo or a continuous motor, extrudes on the top. The shell is a mix of 3D printed and laser-cut parts, with the newer version being fully 3D printed. To power the system, a rechargeable lithium polymer (LiPo) battery is contained within the shell.

Figure 2.2. Smart Motor v3 displayed from each side

There are two mechanical interfaces for the Smart Motor v3. One is the motor output, which protrudes from the top of the Smart Motor and is designed to mimic the connection of a LEGO Education motor, allowing LEGO pegs to be connected to the motor output of the Smart Motor. The other is the back of the Smart Motor shell, which is designed to accept LEGO connector pins, allowing components of the LEGO system in play to be attached to the motor or, conversely, the motor to be attached to a LEGO component-based structure. While the system is not directly compatible with LEGO bricks, the dimensions of the shell are approximately the size of a 6 x 6 x 5 brick. Specifically, the cube-shaped shell of the Smart Motor v3 is 47 *mm* x 47 *mm* x 47 *mm* (W x B x H), making it partially compatible with the LEGO system in terms of pure dimensions. However, in newer iterations, the extrusions added to the side buttons extend beyond this frame, as do the button and potentiometer on the front and the motor on the top of the cube-shaped Smart Motor.

Electronics Component and Schematics

The electronical schematics of the Smart Motor v3 closely resembles the general Smart Motor schematic outlined in Figure 2.1. At the heart of the system is a custom printed circuit board (PCB) called the Dahal Board, which houses a Seeed Studio XIAO ESP32C3 development board (Seeed Studio, 2024b), powered by a ESP32C3-series System on Chip (SoC) MC (Espressif Systems, 2024a). For simplicity sake, the Seeed Studio XIAO ESP32C3 will be referred to as the microcontroller-board (MCB). The MCB includes a USB-C connector, and the custom PCB provides various connection options, such as a Grove-compatible sensor port, two connections

for a motor or other analogue outputs, as well as two I2C ports and a battery connection. A variety of components can be connected to the board, although in the case of the Smart Motor, the only fixed connections are to the motor, the screen, which is connected to one of the I2C ports and the battery. The board contains a built-in accelerometer which is used as the default sensor, although it is possible to connect and use other types of sensor to the external Grove-compatible sensor port. The board also contains other types of user interface such as an OLED screen, two small buttons, one large button and a potentiometer to allow and enable user interaction directly on the system. A comprehensive list of the Smart Motor v3 components and their price are listed in Table 2.1.

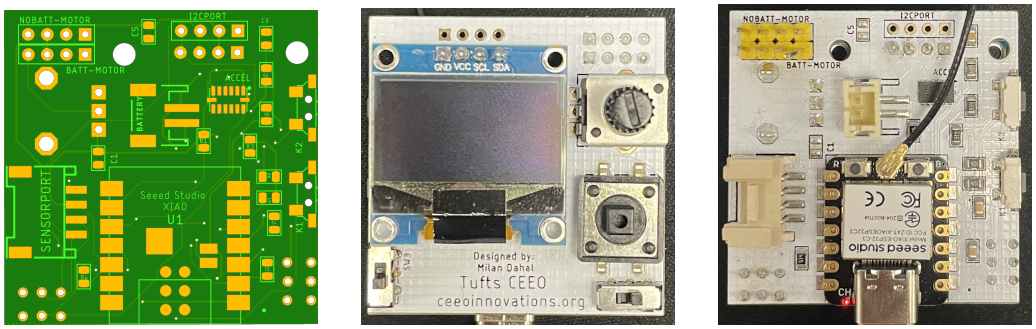


Figure 2.3. Schematic (left) and image from top (middle) and bottom (right) of the Smart Motor v3 custom PCB (Dahal Board)

Name	Description	Price (USD)
Dahal Board	Custom PCB, incl. MC, three buttons, a potentiometer, screen, on-board accelerometer and multiple connectors	16.42 $
Seeed Studio XIAO ESP32C3	SP32C3-based MCB	4.99 $
Miuzei MG90S 9G	Servo motor	2.76 $
Adafruit 4236	LiPo Battery	6.95 $

Table 2.1. Comprehensive list of all Smart Motor v3 electronic components (Dahal, 2024)

Software Architecture

The Smart Motor v3 runs on a slightly adapted version of the MicroPython firmware for ESP32-based MCs, such as the ESP32C3, which allows Python-based code to be run on the chipset. The use of MicroPython brings the advantage of extensive libraries and an active support community, making it easy to use and learn, which in turn makes the system and its code accessible to anyone with basic programming skills and access to a computer and a USB cable.

The software that enables the Smart Motor logic to run includes the main code and libraries to enable and support specific hardware components. All code files and their purpose are listed in Table 2.2.

Name	Purpose
boot.py	empty
main.py	Main program
data.py	File with saved data points
files.py	Library with logic to read and save to files
icons.py	Library with icons for screen
log.py	Log output file
prefs.py	Preferences file
sensors.py	Sensor support library
servo.py	Servo support library
ssd1306.py	Screen support library
version.py	File with version number of all files

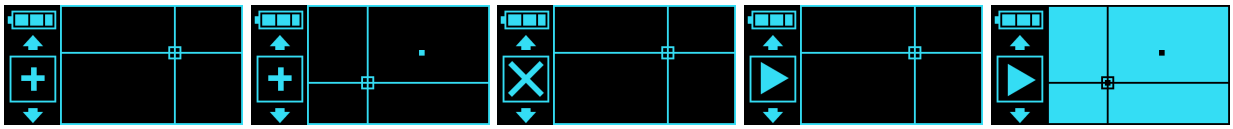
Table 2.2. List of all code files contained on the Smart Motor v3 and their purpose.

The main program contains all the logic necessary for the Smart Motor to function as intended. It sets up and configures all the pins of the ESP32C3 MCB and initialises all the connected components, including the sensors, the motor and the user interface, which includes the screen, the main and two side buttons and the potentiometer. In the case of sensors, it detects whether an external sensor is present and, if not, defaults to using the on-board accelerometer. It also defines the different modes required, training and playing, and sets up the UI logic. It further sets up the button-press handlers and their logic and links the potentiometer to the motor output in the case of training mode. It also defines the ML principle and hosts the nearest neighbour algorithm that determines the motor output based on the current sensor input and training data for the game mode. The further code files host various libraries and supporting code for the different components.

User Interface and Interaction Design

One of the key design features of the Smart Motor concept is for the Smart Motor to used as a stand-alone unit, hence be trainable directly on the device, without the need of any supporting material. To this extent Dahal (2024) has come up with an interaction design and UI, enabling interaction of a user with the Smart Motor system. In the case of the Smart Motor v3, the

UI includes a screen to display information in a graphic way, the two side buttons, used for navigation, the main button, used as a select button and a potentiometer, which in the training mode is linked to the motor, and used to set the motor position or speed.



(a) Training screen: Add data-point

(b) Training screen: Add data-point, One added point

(c) Training screen: Remove data-point

(d) Training screen: Start playing mode

(e) Playing screen: Two data-points

Figure 2.4. Screen UI for training- (a-d) and playing-mode (e) (from Dahal, 2024, p. 40-41)

The design of the visual information on the screen, as shown in Figure 2.4, includes a battery indicator in the top left, an action indicator in the middle to bottom left, and spanning the rest of the screen is a two-dimensional plot showing the current motor position on the X-axis and the current sensor reading on the Y-axis in the form of lines. There is a small crosshair where these two lines meet. In training mode, which the SM v3 enters per default when starting up, there are three possible actions, with the current action being displayed by the action indicator. Switching between the actions is possible using the side button, while confirming the action can be done using the select button.

1. Adding Data-point:
   Training data, in the form of data points (linking a particular sensor reading to a particular motor position) can be added using the Select button, which adds a point at the location of the crosshairs using the current value of the motor and sensor, as shown in Figure 2.4 (a). An added data point is displayed as a point on the graph at the position where the motor position value on the X-axis meets the sensor value on the Y-axis, as seen in Figure 2.4 (b). These points represent the training data for the Smart Motor, which is used in play mode.

2. Removing Data-point:
   Added data points can be removed using the remove action, shown in Figure 2.4 (c), with data-points being removed in the reverse order of them being added.

3. Enter Playing Mode:
   With the third action playing mode can be entered, as displayed in Figure 2.4 (d). The playing mode is displayed as shown in Figure 2.4 (e). To exit the playing mode, the select button can be used to return to the screen in Figure 2.4 (d) or by directly using the side buttons to select a different action, which also exits playing mode.

2.1.2 Technology and Literature Review

Most of the hardware, with the exception of the custom PCB and the Smart Motor v3 shell, are off-the-shelf components. As such, these components inherently offer greater capabilities and more potential applications due to their general-purpose design. This is particularly true for the ESP32C3-based MCB, which is examined in detail. ESP32C3 SoC-enabled networking options are also explored, such as Bluetooth and Wi-Fi, including previous work and solutions developed in this field.

ESP32 System-on-Chip Family

ESP32 refers to a family of SoCs designed and developed by Espressif Systems, starting with the ESP32 series of SoCs, originally manufactured by Taiwan Semiconductor Manufacturing Company Limited (TSMC) using their 40 *nm* process (Espressif Systems, 2025). Further series have been developed and released over time, including the ESP32C3 and ESP32C6 series, which Seeed Studio has used to develop the corresponding XIAO development boards. The first of these, the ESP32C3-based MCB, is of particular interest due to its use in the Smart Motor v3, though consideration will also be given to the newer ESP32C6-based MCB. Although newer series of the SoC, such as the ESP32C5 and ESP32C61, have been announced (Espressif Systems, n.d.-a), their respective datasheets have not yet been released, nor is there any indication of development of the respective development boards by Seeed Studio at the time of writing.

Seeed Studio XIAO ESP32C3

The Seeed Studio XIAO ESP32C3 is a compact development board based on the ESP32C3 SoC. This MCB is designed to leverage the key features of the ESP32C3, in particular its wireless connectivity capabilities. At its core, the board uses a 32-bit RISC-V CPU and supports both IEEE Standard 802.11b, -g and -n (Wi-Fi 4) and Bluetooth Low Energy (BLE) of the Bluetooth 5.0 standard. In terms of interfaces, the chip provides four serial interfaces, including two UART, one I2C and one SPI, as well as eleven GPIO pins (which can be used for PWM), four ADC pins and one JTAG bonding pad interface. The board's design incorporates a small form factor with a single-sided surface mount layout, which is how it is mounted on the Dahal board. The board also includes a Hirose U.FL antenna interface for its wireless functionality. For user interaction, the MCB includes a small reset button and a bootloader mode button, as well as an LED indicator for the USB-C interface. The MCB's layout, pin-out and images are shown in Figure 2.5, while a summary of the MCB's and MC's specifications can be found in Table 2.3. (Espressif Systems, 2024a; Seeed Studio, 2024b)

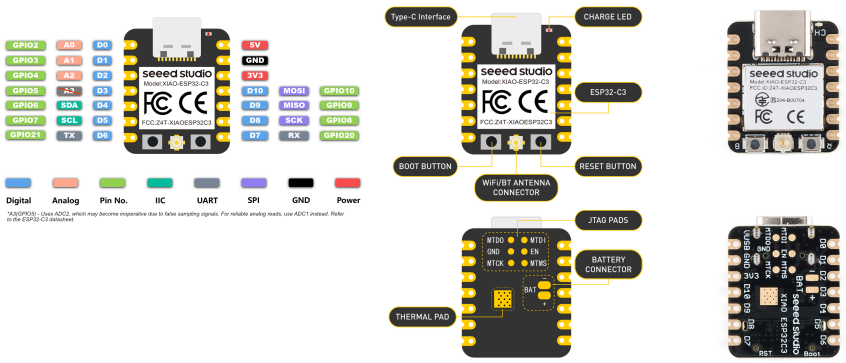


Figure 2.5. Seeed Studio XIAO ESP32C3 pin-out schematic (left), layout (middle) and image(right) (adapted from Seeed Studio, 2024b)

Seeed Studio XIAO ESP32C6

The Seeed Studio XIAO ESP32C6 is a compact development board based on the ESP32C6 SoC. Same as the ESP32C3, this MCB is also designed to leverage the key features of its underlying chip, the ESP32C6, in particular its wireless connectivity capabilities. At its core, the board uses two 32-bit RISC-V CPUs and supports the IEEE Standard 802.11ax (Wi-Fi 6), BLE of the Bluetooth 5.3 standard and IEEE Standard 802.15.4 (Zigbee and Thread). In terms of interfaces, the chip provides four serial interfaces, including one UART, one I2C, one LP I2C and one SPI, as well as eleven GPIO pins (which can be used for PWM), seven ADC pins and one SDIO interface. The board's design incorporates a small form factor and also includes an onboard antenna, as well as a Hirose U.FL antenna interface for its wireless functionality. For user interaction, the MCB includes a small reset button and a bootloader mode button, as well as an LED indicator for the USB-C interface. The MCB's layout, pin-out and images are shown in Figure 2.6, while a summary of the MCB's and MC's specifications can be found in Table 2.3. (Espressif Systems, 2024b; Seeed Studio, 2024c)

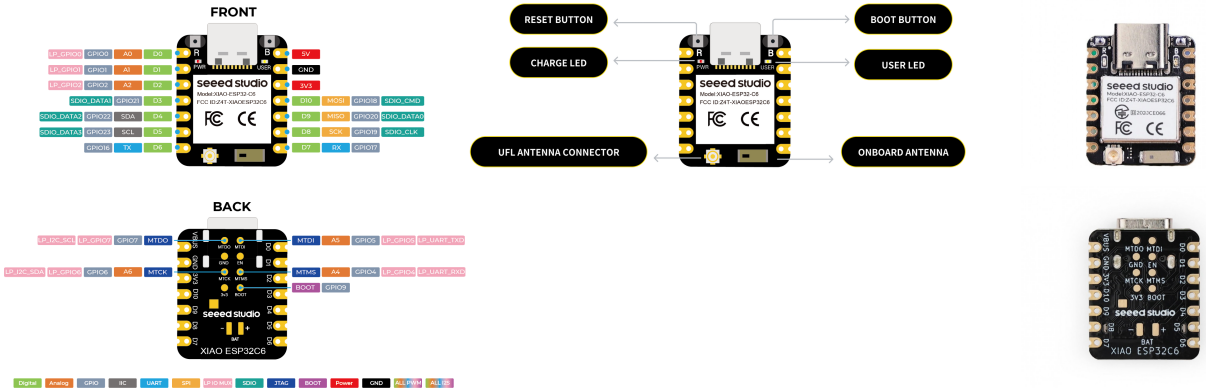


Figure 2.6. Seeed Studio XIAO ESP32C6 pin-out schematic (left), layout (middle) and image (right) (adapted from Seeed Studio, 2024c)

Products		XIAO ESP32C6	XIAO ESP32C3
Processor		Espressif ESP32-C6 SoC	Espressif ESP32-C3 SoC
		Two 32-bit RISC-V processors, with the high-performance one running up to 160 MHz, and the low-power one clocking up to 20 MHz	RISC-V single-core 32-bit chip processor with a four-stage pipeline that operates at up to 160 MHz
Wireless	Wi-Fi	2.4 GHz Wi-Fi 6 subsystem	2.4 GHz Wi-Fi 4 subsystem
	BLE	Bluetooth 5.3, Bluetooth Mesh	Bluetooth 5.0, Bluetooth Mesh
	Other	Zigbee, Thread, IEEE 802.15.4	-
	Antenna	U.FL interface with ext. antenna	On-board antenna + U.FL interface
On-chip Memory		512kB SRAM & 4MB Flash	400kB SRAM & 4MB Flash
Interface		1x UART, 1x LP_UART, 1x I2C, 1x LP_IIC, 1x SPI, 11x GPIO (PWM), 7x ADC, 1x SDIO	2x UART, 1x I2C, 1x SPI, 11x GPIO (PWM), 4x ADC
		1x Reset button, 1x Boot button	
Dimensions		21 x 17.8 mm	
Power	Input voltage	Type-C: 5 V BAT: 4.2 V	
	Circuit operating Voltage	USB: 5 V @ 9 mA BAT: 3.8 V @ 9 mA	
	Charging battery current	100 mA	350 mA
Power Consumption (Supply Power: 3.8 V)	Modem-sleep Model	∼30 mA	∼24 mA
	Light-sleep Model	∼2.5 mA	∼3 mA
	Deep Sleep Model	∼15 µA	∼44 µA
Working Temperature		-40°C ∼ 85°C	

Table 2.3. Specification comparison of the ESP32C3 SoC- and ESP32C6 SoC-based MCB (adapted from Seeed Studio, 2024b; 2024c)

Networking

Both the ESP32C3 SoC- and ESP32C6 SoC-based MCBs come with built in Bluetooth and Wi-Fi wireless networking capabilities, the latter of the two MBCs even includes additional hardware and support for the IEEE Standard 802.15.4 ("IEEE Standard 802.15.4-2024", 2024) for low-rate wireless networks. Both MCBs come with an antenna connector, with the latter even including a small on-board antenna.

Bluetooth and Bluetooth Low Energy

Bluetooth is a short-range wireless transmission technology standard, operating in the 2.4 — 2.485 *GHz* industrial, scientific, and medical (ISM) radio band, with a radio class-dependant range of approximately 1 meter, 10 meters and 100 meters for Class 1, Class 2 and Class 3 respectively (see "Basics | Bluetooth Technology Website", 2012). The standard was developed and is maintained by the Bluetooth Special Interest Group, which includes various companies as members. While initial specifications of Bluetooth were adopted by the Institute of Electrical and Electronics Engineers (IEEE) as part of the IEEE Standard 802.15 in 2002 (Bluetooth 1.1, "IEEE Standard 802.15.1-2002", 2002) and 2005 (Bluetooth 1.2, "IEEE Standard 802.15.1-2005", 2005), these standards have since been withdrawn, with them instead maintained and released directly by the Bluetooth Special Interest Group. In addition to the classic Bluetooth protocol, with the release of the Bluetooth 4.0 standards, Bluetooth Low Energy (BLE) was introduced as a separate entity. As the name suggests, BLE is intended as a low-energy version of the classic Bluetooth for internet of things (IoT) applications ("Bluetooth Low Energy | Bluetooth Technology Website", 2017). While both classic Bluetooth and BLE maintain full backwards compatibility with earlier versions, the two protocols are not compatible with each other, though they can co-exist on the same device. ("Bluetooth Technology Website", n.d.)

Feature	**ESP32C3**	**ESP32C6**
Bluetooth version	Bluetooth 5.0 Low Energy	Bluetooth 5.3 Low Energy
Maximum data rate	2 Mbps	
Frequency band	2.4 GHz	
Frequency range	2402-2480 MHz	
Range	up to 240 m	
Advertising channels	3	

Table 2.4. Specifications of Bluetooth standards found on the ESP32C3 and ESP32C6 SoCs ("Bluetooth Core Specification 5.0 (amended)", 2024; "Bluetooth Core Specification 5.3 (amended)", 2024)

The ESP32C3 supports BLE standards up to version Bluetooth 5.0 ("Bluetooth Core Specification 5.0 (amended)", 2024; Espressif Systems, 2024a; Seeed Studio, 2024b), while the ESP32C6 supports BLE standards up to version Bluetooth 5.3 ("Bluetooth Core Specification 5.3 (amended)", 2024; Espressif Systems, 2024b; Seeed Studio, 2024c). An overview of the two technologies is listed in Table 2.4.

ESP-BLE-MESH

The ESP32C3 and ESP32C6 SoCs' support for BLE adds support for Bluetooth Mesh. Bluetooth Mesh is a device mesh networking protocol based on BLE ("Bluetooth Mesh Model", 2023; "Bluetooth Mesh Protocol", 2023). Building on Bluetooth Mesh, Espressif Systems has developed ESP-BLE-MESH, a Bluetooth Mesh implementation specifically for ESP32 based devices. Once formed, the ESP-BLE-MESH can become a large scale (1000+) interconnected peer-to-peer mesh. Nodes are connected to all other nodes within their range, with the topology of the network determined by heartbeat messages sent by nodes and received by other nodes in their vicinity. A mesh does not form automatically, but requires provisioning: to add nodes to the mesh, they must be provided with the necessary information and cryptographic key via a provisioning device, such as a smartphone. In terms of message transmission, ESP-BLE-MESH uses a controlled flooding approach, which means that a node sends a message to all other nodes to which it is connected. Specially designated gateway nodes forward the message to all their connected nodes, and so on. As the ESP-BLE-MESH is a fully interconnected mesh, there are many overlapping connections and routing possibilities, so in a controlled flooding approach a message is received by sheer brute force, with the possibility of nodes receiving the same message more than once, creating redundancy at the cost of inefficient routing and high network load. The ESP-BLE-MESH implementation is available for use with ESP32-based devices using the Espressif IoT Development Framework (ESP IDF) and standard ESP AT firmware, but there is currently no library implementation for ESP32 devices using MicroPython firmware. ("Bluetooth Mesh Model", 2023; "Bluetooth Mesh Protocol", 2023; Espressif Systems, n.d.-b)

Wi-Fi

Wi-Fi is a family of wireless transmission technologies that enable wireless local area networks (WLANs), based on the original IEEE Standard 802.11 and its amendments ("IEEE Standard 802.11-2024", 2024), and certified for interoperability by the Wi-Fi Alliance ("Certification Process Overview", 2020). The Wi-Fi Alliance, like the Bluetooth Special Interest Group, is a membership-based group of companies that includes most of the manufacturers of IEEE Standard 802.11-based technology. ("Wi-Fi Alliance", n.d.) Wi-Fi operates in the 2.4 GHz ISM band,

the same band used by Bluetooth, although newer versions also use the 5 *GHz* and 6 *GHz* frequency bands. Range and transmission bandwidth are highly dependent on the specific IEEE standard used, which usually corresponds to the Wi-Fi generation. Wi-Fi generations are mostly backwards compatible with earlier generations and standards.

	ESP32C3	**ESP32C6**
Wi-Fi Generation	Wi-Fi 4	Wi-Fi 6
IEEE Standard	802.11n	802.11ax
Maximum data rate	600 Mbps	9.6 Gbps
Frequency band	2.4 GHz	
Frequency range	2402-2482 MHz	
Channels	13, 11	
Channel width	20, 40 MHz	
MIMO streams	Up to 4	Up to 8
Modulation	Up to 64-QAM	Up to 1024-QAM

Table 2.5. Specifications of Wi-Fi standards as implemented on the ESP32C3 and ESP32C6 SoCs ("IEEE Standard 802.11ax-2021", 2021; "IEEE Standard 802.15.5-2009", 2009)

The ESP32C3 SoC supports the IEEE Standard 802.11b, -g and -n, which correspond to Wi-Fi 1[1], Wi-Fi 3[2] and Wi-Fi 4 respectively (Espressif Systems, 2024a; "IEEE Standard 802.11b-1999", 2000; "IEEE Standard 802.11g-2003", 2003; "IEEE Standard 802.15.5-2009", 2009; Seeed Studio, 2024b; Wi-Fi Alliance, 2023), while the ESP32C6 SoC supports the IEEE Standard 802.11ax, which corresponds to Wi-Fi 6 (Espressif Systems, 2024b; "IEEE Standard 802.11ax-2021", 2021; Seeed Studio, 2024c; Wi-Fi Alliance, 2023). In both cases, the frequency band used is the 2.4 *GHz* ISM band due to hardware restrictions. An overview of the two generations used by the two SoCs is given in the Table 2.5.

ESP-Wi-Fi-Mesh

Based on standard Wi-Fi protocols, Espressif Systems has developed ESP-WIFI-MESH, a mesh networking implementation specifically for ESP32-based devices. Traditionally, in a Wi-Fi network, all peers are connected directly to the AP in a hub-and-spoke fashion. With ESP-WIFI-MESH, nodes outside the range of an AP, but within range of the root node connected to the

[1] Wi-Fi 0, Wi-Fi 1, Wi-Fi 2 and Wi-Fi 3 were names not used in proper Wi-Fi nomenclature, but are retrospectively inferred to the respective IEEE standards with the introduction of the IEEE Standard 802.11n, which was named Wi-Fi 4 by the Wi-Fi Alliance, with which official Wi-Fi generation numbering began. (Wi-Fi Alliance, 2023)

[2] See Footnote 1

Wi-Fi, or an intermediate node connected to the root node either directly or via another intermediate note, can be connected to the Wi-Fi. This is done by creating multiple Wi-Fi networks, taking advantage of the fact that the ESP32 devices have two Wi-Fi interfaces (one Wi-Fi and one AP), allowing nodes to host their own network while being connected to another network. The mesh network hence forms a tree-like structure with a maximum depth of four layers, starting from a root node that is either manually selected or automatically selected based on the RSSI strength of beacon frames from a Wi-Fi router or access point (AP). The network operates autonomously, is self-organising and has self-healing capabilities should a node become inoperable. To form the network, connected nodes, starting with the root node, send out Wi-Fi beacon frames, which inform nodes in their vicinity of their presence. Upon receiving one or more such signals, the surrounding node then connects to a possible parent node, first considering the depth of the parent node, meaning how many intermediary node connections away that node is, referred to as layers, choosing the one closest to the root node, followed by the number of children of two similarly deep parent nodes, choosing the one with fewer children. Since the topography of the network is defined, not intertwined, and the root and parent nodes know their respective subnets (in the form of a table or table of tables of MAC addresses), nodes can forward messages directly to the respective child or intermediate child node if the MAC address is contained in their table. If the MAC address cannot be found, i.e. it is not part of that parent node's subnet, the node sends the message to its parent node, and so on until it reaches the root node, which has information about the entire mesh network. As the MCs have two MAC addresses, one for the AP and one for the Wi-Fi, the Wi-Fi MAC address is used as an identifier. The ESP-WIFI-MESH implementation is available for use with ESP32-based devices using the Espressif IoT Development Framework (ESP IDF) and standard ESP AT firmware, but no implementation is yet available for the MicroPython firmware. (Espressif Systems, n.d.-d)

PainlessMesh

The PainlessMesh library is an open source project written in C++ for ESP32-based devices using Arduino, and enables the creation of ad-hoc, decentralised, stand-alone, Wi-Fi-based network meshes. The project is written using native ESP32 SDK libraries. The mesh is self-organising, self-healing and decentralised in structure, meaning that there is no central root node, although a central root node can be set if required. During formation, a node first considers which AP to join based on the list of other nodes to which it is connected (either directly or indirectly), thus avoiding loops in the network, and then the second choice is based on the RSSI value. This is possible because the topology of the mesh is known to all nodes and is constantly updated. As such, the structure of the mesh contains no entanglement, as nodes can only connect to one AP, but multiple nodes can connect to the same AP, although there is

a risk of multiple separate mesh networks being formed. To prevent the formation of multiple smaller networks, nodes disconnect and randomly connect to different nodes to form a single coherent mesh, although this process is random and time consuming. Message transmission is simple as there is only one route between any two nodes and all nodes are aware of the entire network topology. The ESP32 chip ID is used as a unique identifier and for readability, messaging is JSON based. PainlessMesh is designed to work with Arduino and is available for use with ESP32-based devices using Arduino firmware, but there is currently no library implementation for ESP32 devices using MicroPython firmware. (van Leeuwen & Franzyshen, 2019)

ESP-NOW

ESP-NOW is connectionless communication protocol, created by Espressif Systems, which uses Wi-Fi management frames to transmit small messages in a peer-to-peer or peer-to-all fashion. A Wi-Fi Management Frame is a message structure defined by the "IEEE Standard 802.11-2024" (2024), used to manage and control Wi-Fi connections. To this end, since management and control of a connection is necessarily independent of an actual connection, they can be transmitted to and received by any Wi-Fi enabled device, even if it is not connected to any network. The receiver filters messages based on the address MAC-header, and receives all messages that are addressed to its MAC address or a broadcast MAC address (*xff:0xff:0xff:0xff:0xff:0xff*) in the MAC-Header. The specific management frame used in ESP-NOW is the Vendor-Specific Action Frame (category code 127), the structure of which is shown in Table 2.6. The Organisation Identifier is set to *0x18fe34* for Espressif systems. The Vendor Specific Content, shown in Table 2.7, is of particular interest as it provides space in the body field for custom content that ESP-NOW uses to transmit its message. The space available is 250 $bytes$, which is the maximum message length of ESP-NOW v1.0, although in a newer version, ESP-NOW v2.0, this is increased to 1490 $bytes$. Looking at the Open Systems Interconnection (OSI) reference model (Zimmermann, 1980), ESP-NOW operates at a low level, just above the physical hardware layer and the data link layer.

MAC-Header	**Category Code**	**Organization Identifier**	**Random Values**	**Vendor Specific Content**	**FCS**
...	*127*	*0x18fe34*	*random*	*see Table 2.7*	...
24 $bytes$	1 $byte$	3 $bytes$	4 $bytes$	7−255 $bytes$	4 $bytes$

Table 2.6. Structure of Management Frame data packet, which with category code *127* becomes a Vendor-Specific Action Frame (adapted from Espressif Systems, n.d.-c)

Element-ID	Length	Organization Identifier	Type	Version	Body
221	*length*	*0x18fe34*	*4*	*ESP-NOW version*	...
1 $byte$	1 $byte$	3 $bytes$	1 $byte$	1 $byte$	$0-250$ $bytes$

Table 2.7. Vendor Specific Content structure for ESP-NOW packet (adapted from Espressif Systems, n.d.-c)

In terms of usability, ESP-NOW runs on an initialised Wi-Fi or AP interface as it uses Wi-Fi management frames to transmit its messages. With an initialised ESP-NOW service, a device will be able to receive all messages addressed to it, while for sending, the MAC address of the peer must first be added to the ESP-NOW, and only then can messages addressed to that MAC address be sent. However, the broadcast MAC address (*xff:0xff:0xff:0xff:0xff:0xff:0xff*) can also be added and then used to send to any device in range with an initialised ESP-NOW, that is set to the same Wi-Fi channel. The protocol also comes with receipt confirmation functionality, which confirms that the transmitted message has been received by all designated peers (except when using the broadcast MAC address). The buffer for receiving messages is 20, at which point older messages will be discarded if they have not been called. In any case, proper handling of received messages to prevent loss of received messages is required, based on an interrupt handler (IRQ) running with high priority Wi-Fi tasks. ESP-NOW can use any of the 14 available Wi-Fi channels to transmit and receive messages. In the case of the ESP32C3 and ESP32C6, since both chips have separate Wi-Fi and AP interfaces, both can be used to send and receive ESP-NOW messages, which theoretically also allows messages to be transmitted and received on two channels simultaneously. For the ESP32C3, which includes an external antenna, the claimed range of ESP-NOW is up to 220 $meters$ with a data rate of 1 $Mbps$. ESP-NOW v1.0 and v2.0 are available for use with ESP32-based devices using standard ESP firmware and Arduino firmware. Furthermore there is also an implementation of ESP-NOW v1.0 within the ESP32 MicroPython firmware, which is used on the Smart Motors. (Eridani *et al.*, 2021; Espressif Systems, n.d.-c; MicroPython, 2025a)

ESP-NOW MicroPython Wi-Fi Mesh

As part of a Master's thesis at the Brno University of Technology and a subsequently published conference paper, a dynamic, autonomous and self-healing mesh network was designed, developed and tested using ESP32 MCs running MicroPython firmware, using both ESP-NOW and Wi-Fi to create a mesh network. The mesh topology is based on a tree structure and data transmission, which includes topology updates, is achieved using Wi-Fi connection between the

modules, with the modules using both Wi-Fi and AP modules, to host its own Wi-Fi network and also connect to a parent nodes Wi-Fi, similar to the ESP Wi-Fi mesh. As the whole topology of the mesh is known by every node and connections are not redundant, transmission is achieved using a routing model. ESP-NOW, on the other hand, is used to create and manage the network, including finding and adding new nodes to the network using a flooding approach. The approach was validated and tested, finding that a mesh of six nodes works well, but when seven or more nodes were used, memory problems began to occur, causing the mesh network to become unstable. (Šesták, 2022a; 2022b)

IEEE 802.15.4

In addition to Wi-Fi and BLE, the ESP32C6 also includes an additional wireless baseband and MAC compliant with the IEEE 802.15.4 protocol. This protocol outlines low-rate wireless personal area networks with low complexity, low data rates and very long durations, dealing with the OSI physical and data link layers. The standard is the basis for protocols such as Zigbee and Thread, both of which are supported by the ESP32C6. (Espressif Systems, 2024b; "IEEE Standard 802.15.4-2024", 2024; "IEEE Standard 802.15.5-2009", 2009; Seeed Studio, 2024a; 2024c)

2.1.3 Networking Requirements and Concept Design

As a first step, a number of requirements for the development of the networking concept were identified and set out here:

- Compatible with Smart Motor hardware and software
- Decentralised peer-to-peer application (no central root or hub)
- Easy module discovery and interaction
- Incorporation of simple command and response logic, data transmission and simple message validation

Based on the technology studied and the requirements outlined above, ESP-NOW was chosen as the basis for the networking approach because of its low-tech, peer-to-peer, non-connected functionality and its ready availability on the hardware and firmware used for the SM. A further advantage is that ESP-NOW is easy to set up, it works out of the box and does not require any configuration or pairing for transmissions to take place, the setup can be done in a few simple lines of code and once initialised runs in the background, allowing a main program to run while still allowing networking based on IRQ handling, making it compatible with most

main module code.

In a second step, using the ESP-NOW protocol as a base, a more detailed networking concept was developed, taking into account the ESP-NOW capabilities and limitations. This included the design of a common message structure, providing for different types of messages, allowing for predetermined handling based on identifiers and codes, and measures to identify the message and check its validity. In order to support this concept of command type messages, the following basic message types have been defined, which should also be reflected in the message structure: Command (*cmd*), Information (*inf*) and Acknowledgement (*ack*). *Cmd* messages require some sort of handling or action to be performed by the receiving device. *Inf* messages are the standard type of message, containing some sort of information or data, such as sensor data, that could be used by the MC's main programme. *Ack* message are responses, either to commands or to information messages, which acknowledge or confirm the command, or may be the direct result of a command returning an information message, such as the proposed *ping* message (*cmd* type) which should initiate a *pong* response (*ack* type). To allow for a variety of these base message types, they should be able to contain different sub-types. As ESP-NOW uses the MAC address of the receiver to address and send messages, which may not be known to the device, a way of identifying the surrounding modules must be implemented. To this end, the library should include an address book to store the MAC addresses as well as associated information such as name, configuration and the Wi-Fi channel used for transmission.

It should also attempt to find a way around the 20 peer limit of the ESP-NOW peer buffer (for unencrypted messages), as well as examining the 250 $byte$ message limit and considering a way to send longer messages. The library should also be designed to be adaptable and usable as a base for custom applications, allowing the definition of custom message subtypes and commands and their respective handling logic. In addition, coding standards and best practices such as the Python Enhancement Proposal 8 (PEP 8) outlined in the Style Guide for Python Code (Rossum *et al.*, 2001) should be applied. The following design principles were outlined for consideration in the development of the networking approach:

- Accessibility (in terms of usability)
- Simplicity (in terms of comprehension, usability and accessibility)
- Flexibility (in terms of application)
- Adaptability (in terms of flexibility and application)

2.1.4 Platform Architecture and Design

In order to support the networking capability, and given the evolution of the Smart Motor, a broader framework has been designed to enable access and support use, interaction and development, called the Smart System Platform.

The platform is divided into three parts, as shown in Figure 2.7, consisting of the hardware, which includes modules, called Smart Modules, with their different designs and components, the software, which includes the code and libraries for the different functionalities, and at its core, the Networking Library, and the documentation and development part, which includes various resources, such as the GitHub page, a website containing guides and support material, as well as the development and management tools designed to support the use of the Networking Library.

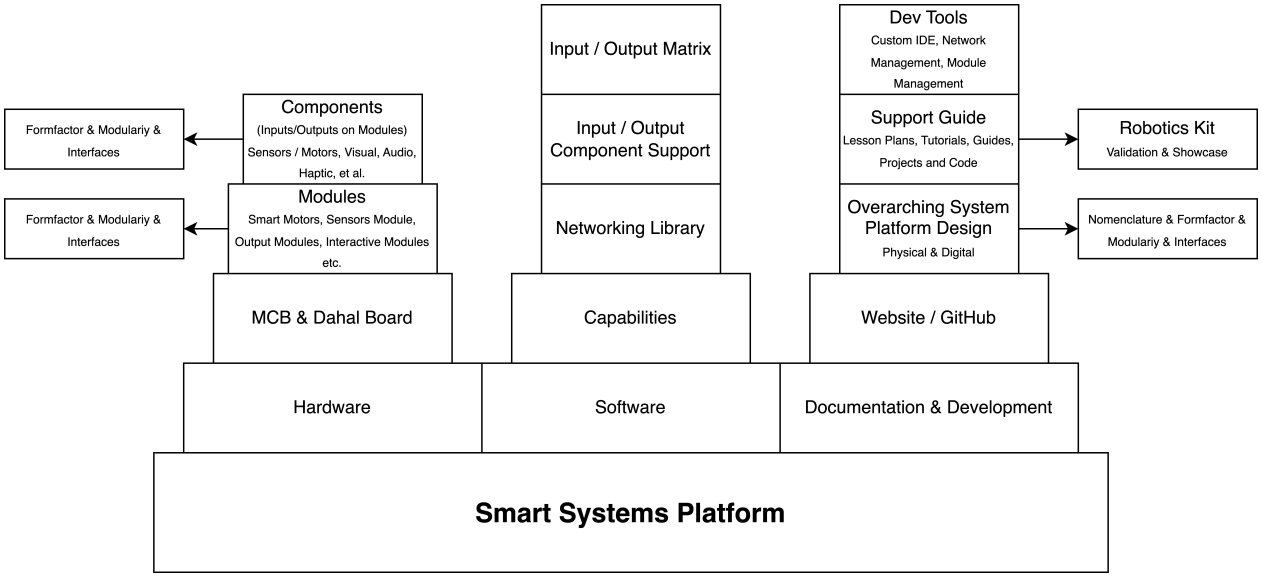


Figure 2.7. Platform Architecture

The main objective of the platform or framework is to provide access to the developed networking approach and to enable its users, and perhaps in a further step different stakeholders such as educational researchers and educational tool developers or even educators and students, to use the platform, to enable a quick ideation, creation and testing using the provided hardware and capabilities, with no entry barriers. In an effort to clarify the concept and the used terms, the SSP-specific nomenclature outlined in Table 2.8.

Term	Description
Hardware	Physical parts of the Smart System Platform
Module	Stand alone modules, such as the Smart Motor, also called Smart Modules
Component	Hardware contained in the module, such as sensors, motors, etc.
Input	Component that provides data input, such as a sensor
Output	Component that provides output, such as a motor
UI	Components that are used to allow user interaction
Software	Code files and library saved and running on a Module
Library	A file to enable a component or capability to work on the module
Networking	The Networking Library
SSP Networking	The SSP Add-on library
Firmware	Firmware running on a Module
Documentation	Documentation for the Smart System Platform, its modules incl. the website, the development and management tools and the GitHub
Website	The Website, hosting information and the dev suite, part of documentation
GitHub	The GitHub page of the Smart System Platform hosting code and all the files
Network Management and Module Configuration Tool	PyScript-based web-application, which allows network management and module control and configuration
Integrated Development Environment	PyScript-based custom web-IDE
Module Management Tool	PyScript-based web-application, which allows modules to be updated to the newest version, based on its configuration
AI Coding Assistant	ChatGPT-based Large Langue Model (LLM), primed for coding and development assistant using the SSP
AI Module Configuration Assistant	ChatGPT-based LLM, primed for configuring SSP modules for interaction using the Network Management and Module Configuration Tool

Table 2.8. Nomenclature for the Smart System Platform

In line with the principles of the networking approach and the ideas outlined in Dahal (2024) and Section 1.2, certain guiding principles were used in the design of the platform architecture:

- Simplicity:
  The platform was designed to be intuitively understandable, trying to reduce components and parts to a minimum in an effort to lower the learning curve and allow easy access.
- Accessibility:
  Website, accessibility, information. Free to use, all information publicly available, hosted on GitHub, including website and development tools and guides.
- Modularity and Interoperability:
  The various components are designed to be modular, interoperable and compatible with each other, both in terms of hardware and software.

2.2 Phase 2: Development

2.2.1 Approach

An iterative approach has been used in the development of the Networking and the components of the Smart System Platform. Much of the work on the different parts and components has been done in parallel with several interrelated iterations, but for ease of reading the different components are described separately in their own subsections.

2.2.2 Networking Development

Given the the requirements and concept for the networking approach defined in Section 2.1.3, the Networking Library is split into two parts, the base Networking Library, which contains the basic networking logic and the more sophisticated SSP Add-on Library, which contains more SSP specific networking logic.

Networking Library

The custom networking library, from here on referred to as the Networking Library, is a generic, all-purpose, customisable networking library built on top of ESP-NOW protocol, that introduces the basic networking functionality and aims to enhance and exploit the capabilities of ESP-NOW.

As outlined in Section 2.1.3, ESP-NOW uses MAC addresses to address messages, however, these addresses might not be known beforehand. To this end an address book is included in the Networking Library, which is automatically populated when a message is sent to a new MAC address or received from an unknown MAC address. In order to be able to discover and gather

information about the surrounding modules, the *ping* command message outlined before, contains information about the sender, such as name and configuration. This *ping* command can be sent out to the general broadcast address and when received by any device running the Networking Library, prompts that devices to return a *pong* message to the sender, which in turn contains information about the responder. In this scenario, both devices have now added each other to their respective address books, which includes the MAC address, name and configuration. This address book, however, is not persistent and the information is lost upon reboot of the device. In addition, an *echo* command, which prompts the receiver to return the same message content to the sender, and a *boop* command, which is intended to calculate and return the stored RSSI values of the last message of every MAC address a message was received from, were also designed as key commands to be included in the core Networking Library. Exception catching and handling is included throughout the code to prevent an error in the message transmission or handling process from interrupting the main code.

The library also includes time-stamped information, debug and error statement printing in the REPL and logging to a local *log.txt* file. These three levels of printing and logging can be individually enabled or disabled. Information print statements are placed only when necessary, such as when sending or receiving, while debug logs are placed in every function and additionally at every critical point in the code, and error logs are logs of caught exceptions.

Message Structure

The message structure developed for the Networking Library is shown in Table 2.9.

Header (1 $byte$)	Type (1 $byte$)	Sub-Type (1 $byte$)	Timestamp (4 $bytes$)	Data Type (1 $byte$)	Data (241 $bytes$)	Checksum (1 $byte$)
Identifier 0x2a	Number (1-3)	Number (0-255)	Number $time.ticks\_ms()$	Number (0-6)	**Payload** ...	Number $sum()\%256$

Table 2.9. Networking message structure

1. The message starts with a *header* ($0x2a$, 1 byte) which is used to identify that the message has been sent by a friendly Networking Library.

2. This is followed by the message *type*, which currently identifies the message as one of the three different message types ($0x01$ for *cmd*, $0x02$ for *inf* and $0x03$ for *ack*, 1 $byte$) defined in Section 2.1.3.

3. The type is followed by the message *subtype*, which identifies the specific subtype of the

message (*cmd*: $0x10$ for *ping*, $0x13$ for *boop* and $0x15$ for *echo*; *inf*: $0x20$ for *boop response*, $0x21$ for *data* and $0x22$ for *message*; *ack*: $0x10$ for *pong* and $0x15$ for *echo*), although only a few subtypes are defined in the core Networking Library. However, the library has provisions to allow the user to define their own message subtypes and their respective handling logic, as is done in the more application specific SSP Add-on Library.

4. The *timestamp* is populated at the time of sending with `time.ticks_ms`, which is the time in milliseconds since the device was started. `time.ticks_ms` returns a $32-bit$ unsigned integer, ensuring that it always stays under 4 $bytes$ by rolling over when it reaches its maximum value. In the case of a $32-bit$ integer, the maximum would be $2^{32}-1\ ms = 4'294'967^{2}95\ ms = 49.71\ days$. However, according to the MicroPython documentation (MicroPython, 2025c), the exact maximum size of `time.ticks_ms` in MicroPython is opaque, although in the case of the MicroPython implementation for ESP32s, the maximum size appears to be only 30 bits, resulting in a maximum of $2^{30}-1\ ms = 1'073'741'823\ ms = 12.43\ days$. As such, the timestamp is only used to calculate relative durations or differences, for example in the *ping* command, where the timestamp is returned to the original sender, which then calculates the ping time, the time it took for the message to travel back and forth.

5. The *data type* field is used to identify the encoding of the payload, as the Networking Library allows the following data types to be encoded: *integer*, *string*, *float*, *list*, *dictionary*, *bytes* and *bytearray*. In addition, the data type is also used to identify long messages, i.e. when a payload longer than 241 $bytes$ is split and sent using multiple messages. In this case the payload type is set to the long message code, where the payload contains its own structure, namely the part number of the message, the total number of parts of the long message and the payload type of the message. Taking into account these three additional meta-informations, each of which occupies one byte per message, the remaining payload is 238 $bytes$. With one byte allocated to the part number of the message, the maximum number of parts is 256 as the part number is stored in 1 $byte$, giving a theoretical payload limit of $256 * 238\ bytes = 60'928\ bytes$. The logic of long message handling is further defined in Section 2.9. With eight options, the data type field takes 1 $byte$.

6. This is followed by the *data* field containing the payload. This payload is encoded depending on the type of data, with the field holding a maximum of 241 $bytes$.

7. Finally, the message structure contains a 1 $byte$ *checksum* to confirm the integrity of the message. The checksum is calculated by taking the sum of all the preceding fields and calculating the modulus of the sums divided by 256, ensuring a number smaller than 256 and therefore fitting into the 1 $byte$ space allocated to it.

However, the message structure does not include the sender's MAC address, as this is already passed by ESP-NOW and included in the surrounding Wi-Fi management frame.

Send Logic

The send logic of the Networking Library is outlined in Figure 2.8. The structure is similar to a funnel, with the pipeline able to be called in a number of ways, either by directly calling one of the implemented command functions, which will format the content and send the appropriate *cmd* or *inf* type message, such as the *ping*, *echo* and *boop* commands, as well as normal message or data type messages. There is also a command to send custom type of msg, which requires the message type and message subtype to be specified. These functions can be called manually by users or by code using the library, although there are some other internal functions, which start the send pipeline, which can only be automatically called from within the networking code. This is the case for sending most *ack* message types such as *pong*, *confirmation*, *success* or *fail*. The various functions for sending specific command types contain logic to look up the appropriate code and format the payload as required by the custom handling logic, and then call the custom send command in the network. Depending on the specific commands, different inputs are required, at least the MAC address or a list of MAC addresses if the message needs no content, as in the case of *ping*, or the MAC and the message or data, as in the case of message or data type messages. And as mentioned above, the type and subtype are also required for the custom send commands. These functions then call the compose function, which attempts to encode the content into payloads.

After encoding the content into the payload, the code checks the length of the payload and, if necessary, splits the message into several parts to ensure a size of 250 $bytes$ or less for each message. The payload is then structured into the message or messages according to the msg structure. To avoid confusion with the message type of the same name, these encoded messages are referred to only as msg. The code then loops through the list of MAC addresses, attempts to add them to the ESP-NOW buffer, then loops through all the msgs and attempts to send each corresponding message. This function will attempt to resend the message three times if an error occurs or if a receipt is not received from the recipient. After sending, the corresponding MAC address is immediately removed from the ESP-NOW buffer to avoid a buffer overflow and to circumvent the 20 peer limit imposed by ESP-NOW. Exception handling is included throughout the code to prevent an error in the sending process from interrupting the main code, including but not limited to catching invalid MAC addresses, invalid message content, or when interfacing with the ESP-NOW library.

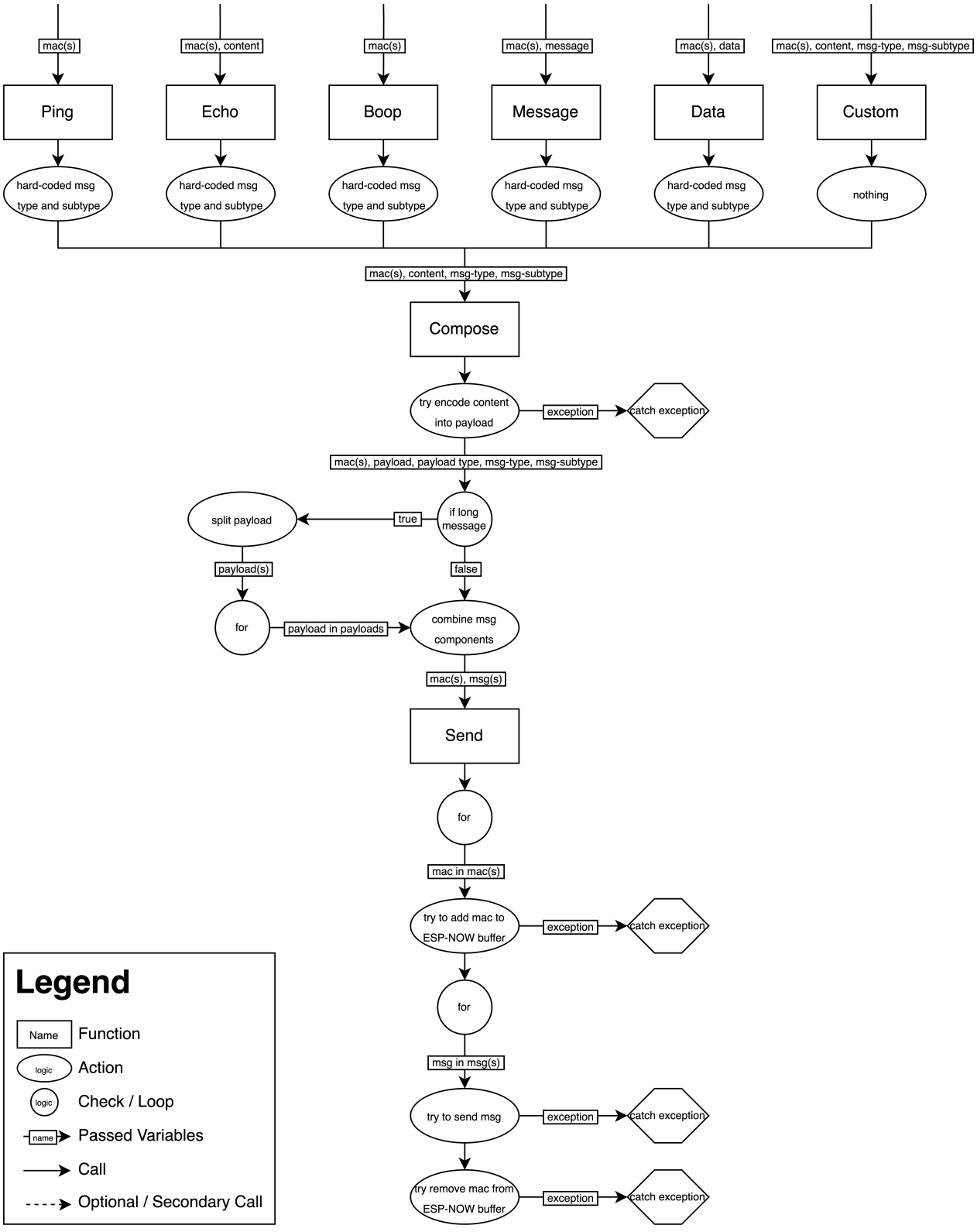


Figure 2.8. Networking message send logic

Receive Logic

The receive logic starts as a linear pipeline since the function to start it, can only be called by a interrupt request (IRQ) of ESP-NOW, which is called when ESP-NOW has received a new msg. The logic follows a tree shape and splits into various branches to handle the different message types, as shown in Figure 2.9.

In a first step the msg is received by the ESP32 and is then handled by ESP-NOW. It then calls the IRQ function, which checks if there is a msg in the ESP-NOW buffer, and then calls the receive function. The receive function loops and gets msg and sender MAC from the ESP-NOW msg buffer as long as there are msgs in the ESP-NOW msg buffer. For each msg it receives, it calls the process function, which attempts to add the sender's MAC address to the Networking Library address book. It then calls the decode function which tries to split the msg according to its structure. If the msg has the wrong structure, does not start with our identifier, or the checksum is incorrect, the msg is discarded. If the msg is split successfully, the payload type is checked, and if the payload type indicates a long message, the msg is sent to the long msg handlers, which decode the payload, collect its number, the total part of msgs and the actual payload type, and add it to the payload buffer. If the payload buffer now contains all parts of the msg, the payload is assembled and passed on. There it attempts to decode the message according to the data type specified by the payload type variable. It is then passed to the handler function, which sorts the msg by type, passes it to the appropriate handler, sorts it by subtype, and calls the appropriate code or logic. In the case of a *ping cmd* msg, this will result in a *pong ack* msg being sent back to the sender. In the case of a message or data type *inf* msg, the content is added to the message or data buffer, while *ack* msg types usually do nothing with the message, except in the case of the *pong* msg, the information encoded in the content of the msg is used to add further information about the peer to the Networking Library address book. If the subtype of the message is not found, the custom subtype handler, if set, is called, where custom handlers are checked. This is the case for the additional SSP Add-on Library, which defines a number of additional msg subtypes and their respective handling. In any case, any received msg that has made it to the handler will be logged as level information. Finally, custom IRQ functions can be called for different message types, if set, and a general custom IRQ function is called after all messages in the ESP-NOW msg buffer have been processed.

Exception handling is included throughout the code to prevent an error in the receiving process from interrupting the main code, including but not limited to catching msgs with an invalid structure, msgs with an invalid payload encoding, or invalid content in terms of handling logic, or any errors that may occur when interfacing with the ESP-NOW library.

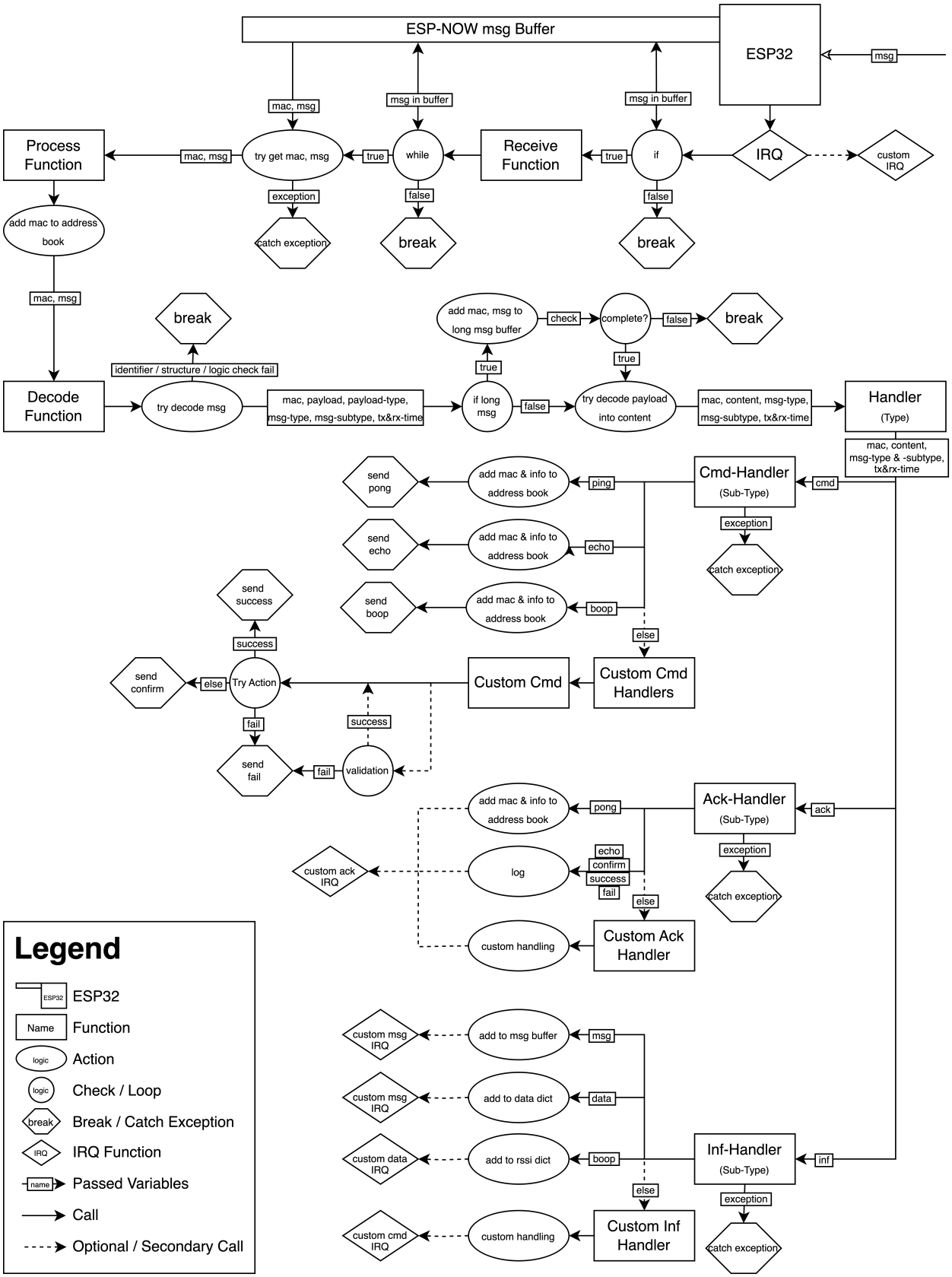


Figure 2.9. Networking message receive éogic

SSP Networking Add-on Library

Building on the Networking Library and using the designed interfaces, an SSP-specific Networking Library add-on has been designed, from here on referred to as the SSP Add-On Library , which defines a variety of SSP specific commands and message types and subtypes, as well as the corresponding custom command handlers for use with different Smart Modules for different applications. The SSP Add-on Library is specifically designed to work with Smart Motors and other Smart Modules and requires both the base Networking Library and a Smart Module *config.py* configuration file to properly function, as shown in the Figure 2.10.

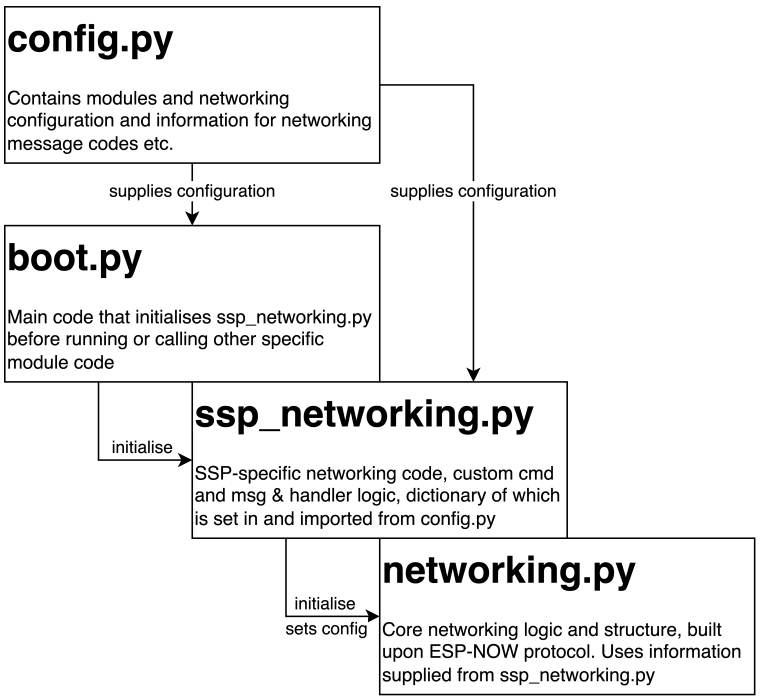


Figure 2.10. Networking code structure

With the aim of future-proofing the code design and making it easy to use the code provided for future projects, the SSP Add-on Library has been designed with a variety of applications and functionalities in mind, which is reflected in the large number of custom commands and message types. Figure 2.11 shows the detailed class and function structure of the two libraries, which illustrates the variety of custom send commands available in the SSP Add-on Library. These include all the basic networking commands and some variations thereof, which are made accessible in the SSP Add-on Library, but call the underlying basic networking functions. In addition, the add-on includes commands to reboot, configure modules to interact, download files, enable AP, connect to Wi-Fi, get the MCBs file directory, change the configuration or configuration file of another module and so on. It also includes an authentication check that only executes certain commands from admin modules whose MAC address is in the configuration's

whitelist, or which use a sudo bypass[3]. The handling of all the custom commands isn't split into different functions, but is contained within the three specific custom type handlers.

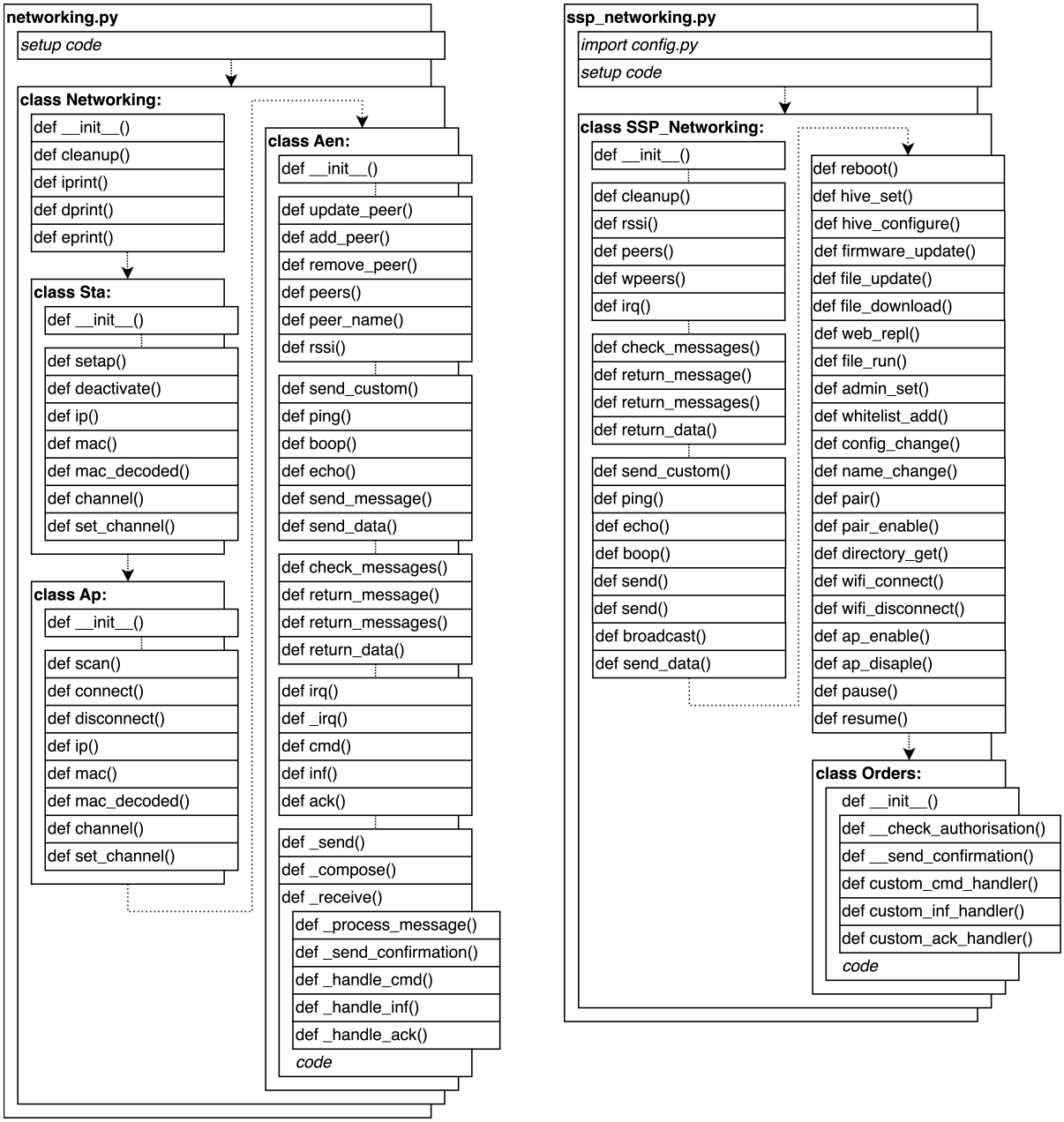


Figure 2.11. Networking Library and SSP Add-on Library code structure

2.2.3 Smart System Platform Development

In addition to the concept and design of the Smart System Platform, described in Section 2.1.4, several components have been developed, which are described in this section. These include

[3]This can be achieved by adding the string "sudo" to a payload list at position [-1].

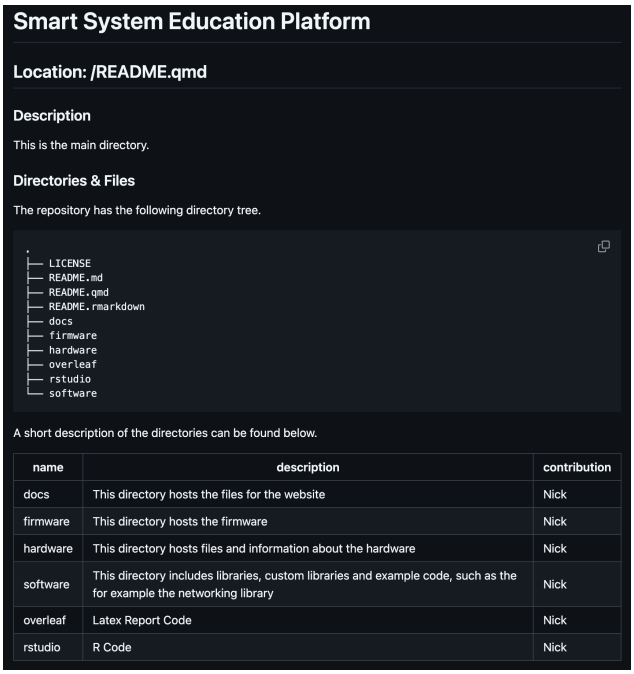
parts of the documentation, such as the GitHub page and the website, including the various networking development and management tools developed in PyScript that are part of the website.

GitHub

As part of the software development, a public GitHub page has been created for this thesis to host the developed code and supporting files. However, in line with the goal of accessibility, the public GitHub page also serves as the main host for all other information and resources of the Smart System Platform, including the website, all data related to this thesis, and more, with the exception of the PyScript-based tools. The site is licensed under the MIT licence.

In terms of organisation, the page is divided into folders, namely *docs*, which contains the website files, *firmware*, which contains the firmware files, *hardware*, which contains the hardware related files and *software*, which contains the code. There are also thesis-specific folders, *overleaf*, which host the thesis report, and *rstudio*, which hosts the draft of the automated README files and any experimental data that has been analysed and their respective plots. There are also two hidden folders, *.github*, which hosts the custom GitHub workflows, and the *.git* folder, which hosts git-specific files. The *main* branch of GitHub is restricted and protected, preventing direct commits to it, requiring a pull request that must be approved before merging. Active development takes place in subversions of the */dev* branch. Development done by the author is hosted in the *dev/nick* branch. This was done in an effort to protect the main release versions of the code from accidental editing.

To help with the development workflow, some automation has been introduced on GitHub. Automated activities include the building of the website, which is automated and triggered whenever there is a push or merge to the *main* branch. In addition to the website being built, there is a custom render and release pipeline that is triggered by merge commits. As the *main* branch is restricted and can't be committed to directly, it requires a pull request to be submitted and approved. The pipeline is triggered when the pull request is approved and merged. It then automatically populates the *software/release* folder with all relevant code files from their various locations, and updates *config.py* with the new version number of those files. The pipeline then renders all the Quarto Markdown README files, updating them to include the latest file structure and information about the respective folders and their files. Finally, the pipeline merges the above changes back into the branch from which the pull request originated.



Smart System Education Platform

Location: /README.qmd

Description

This is the main directory.

Directories & Files

The repository has the following directory tree.

```
.
├── LICENSE
├── README.md
├── README.qmd
├── README.rmarkdown
├── docs
├── firmware
├── hardware
├── overleaf
├── rstudio
└── software
```

A short description of the directories can be found below.

name	description	contribution
docs	This directory hosts the files for the website	Nick
firmware	This directory hosts the firmware	Nick
hardware	This directory hosts files and information about the hardware	Nick
software	This directory includes libraries, custom libraries and example code, such as the for example the networking library	Nick
overleaf	Latex Report Code	Nick
rstudio	R Code	Nick

Figure 2.12. Example of the *README.md* file, which is used to give information about the current directory and its files

Development and Management Tools

In an effort to facilitate interaction with the developed Networking Library and for the sake of accessibility and ease of debugging, various tools have been created to assist in the management and development using the Smart System Platform.

Integrated Development Environment

A read-eval-print loop (REPL) is a simple, interactive interface to a computing device, in the case of MCs using asynchronous serial communication with a universal asynchronous receiver/-transmitter (UART). It provides outputs, can accept user input, execute supplied code and return results to the user. A REPL is also the area where print statements are displayed from code running on a device. The MicroPython REPL prompt is provided by default on the device's serial peripheral UART0, connected to pins GPIO1 for TX and GPIO3 for RX, with a baud rate of 115200 (MicroPython, 2025b). The ESP32C3 MCB also has a USB-to-serial converter that allows direct connection to the REPL via the USB interface. This allows connections to be made directly from an IDE such as Thonny or PyCharm. However, Chrome-based browsers also allow websites to connect to devices via various serial ports, such as USB, which was used by WebReflection (n.d.) to build micro-repl, a web-based REPL, which was used by Rogers (n.d.) to build a web-based IDE using PyScript. The advantage of a web-based IDE is that it requires

no software installation and works out of the box in the Chrome browser. This web-based IDE was used as the basis for the development of the custom SSP IDE, seen in Figure 2.13 and Figure 2.14, specifically designed and specified for use with the Smart System Platform and Networking Library, which was developed using PyScript. PyScript is an open source platform for Python in web-browser, aimed to enable development of web-applications using Python. (Anaconda Inc., n.d.-a; n.d., -b; 2025)

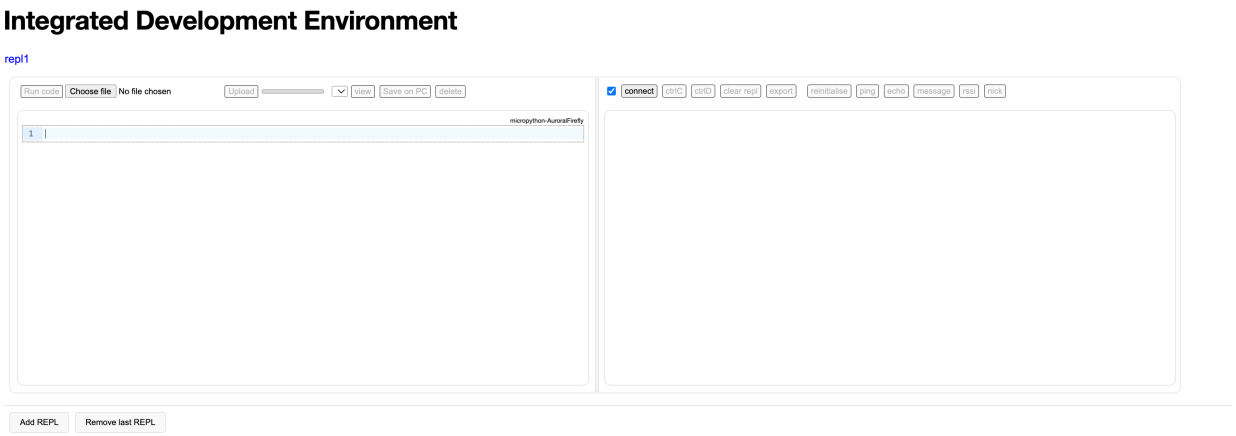


Figure 2.13. Web-based IDE

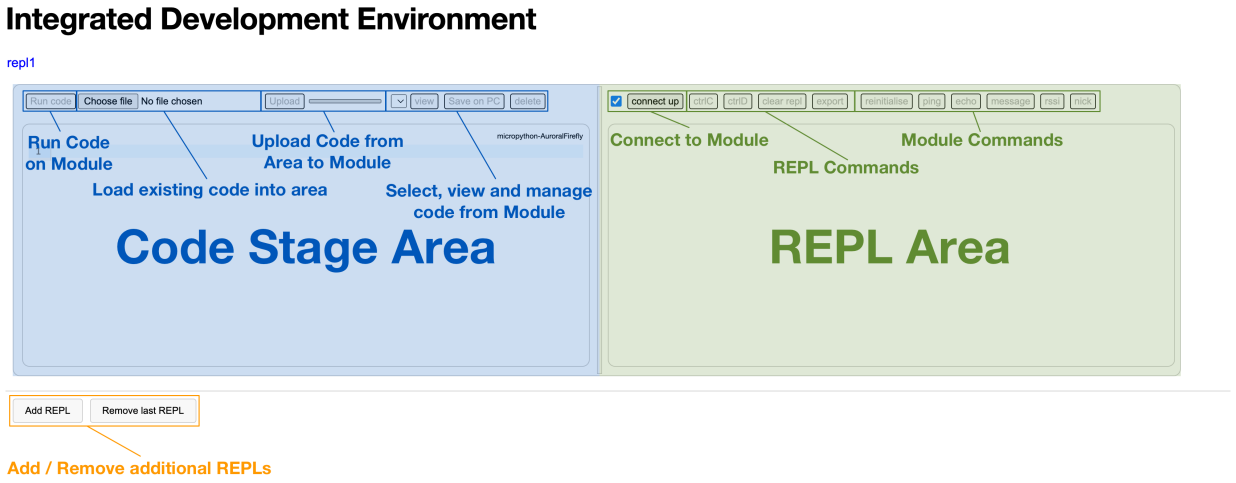


Figure 2.14. Annotated IDE

The REPL area, which represents the connected module, is where the serial REPL is displayed. It also contains REPL and SSP specific command buttons such as *connect* to connect to a module, *ctrl-c* and *ctl-d*, *clear repl* and an *export* button to export the complete REPL text to a local file. The SSP specific module commands are *reinitialise*, which initialises a base code, which initialises networking, *ping*, which sends a *ping cmd* message, *echo*, which sends an *echo* message, *message*, which sends a *message* type msg and *rssi*, which prints the RSSI table of the module. The code stage area, which represents the website/computer side, includes a space

for writing code, as well as various command buttons such an *upload* button that uploads the code in the stage area as a file to the module, a *selection menu* and a *view* button to display code from code files from the device, a *save to PC* button to save the selected file to PC and a *delete* button to delete the selected file.

Additional terminal pages can be added and removed using the *Add REPL* and *Remove last REPL* buttons. This allows multiple terminal pages to be added, which has been done in an effort to facilitate concurrent coding on multiple connected devices or modules, which is especially helpful when using the Networking Library. This capability comes directly from a shortcoming in other IDEs. For example, Thonny, a beginner-friendly Python IDE (Aivar Annamaa, n.d.; 2025; Annamaa, 2015a; 2015b), only allows one window to be open at a time.

AI Code Assistant

In an effort to ease the learning curve and simplify development of code using the Networking Library and other SSP libraries, a Large Language Model (LLM), specifically OpenAI's Chat-GPT, has been primed with instructions, coding documentation and annotated code examples. Based on this, the idea was that the assistant should be able to provide specific code using correct structure and nomenclature, as requested by the user, and help in debugging during development using the Networking Library and SSP.

Network Management and Module Configuration Tool

To be able to manage the network and all the modules in the vicinity, the Network Management and Module Configuration Tool has been developed using PyScript based on the SSP IDE.

The concept of the Network Management and Module Configuration Tool is shown in Figure 2.15. The website uses information provided by the REPL to populate a list of modules and their information. It also introduces several commands that allow commands and messages to be sent to the selected modules from the Admin module. The website simply creates the commands based on the module information and, if applicable, user-provided content, and sends the command to the REPL, where it is received and executed by the Admin Module.

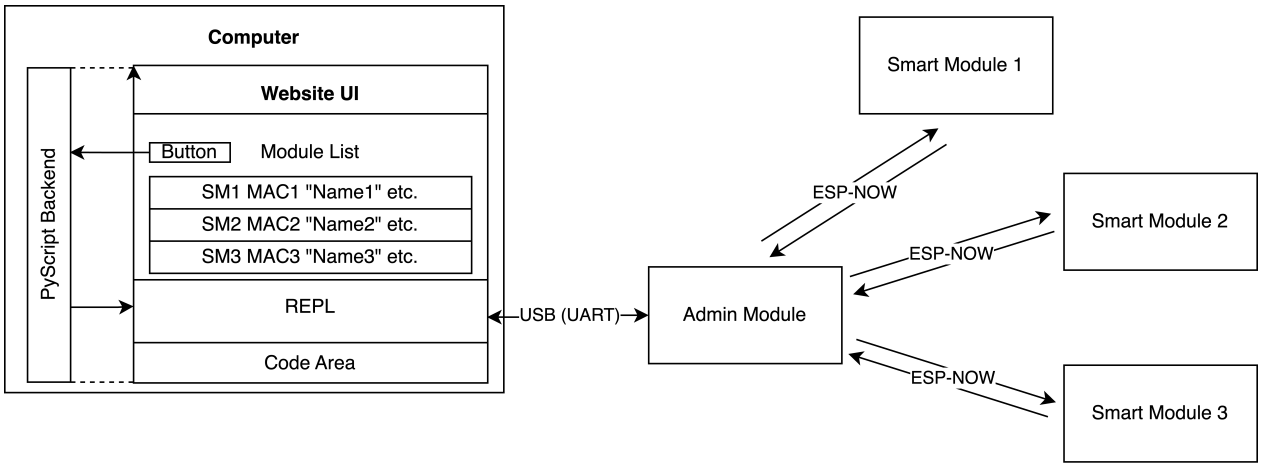


Figure 2.15. Functional concept of the Network Management and Module Configuration Tool

As shown in Figure 2.16 and Figure 2.17, the page is divided into three parts, the bottom part is the code stage area, the middle part contains the REPL area of the associated admin module, while the top part hosts the module management area. The code stage area and the REPL area are almost identical in functionality to the corresponding areas of the IDE.

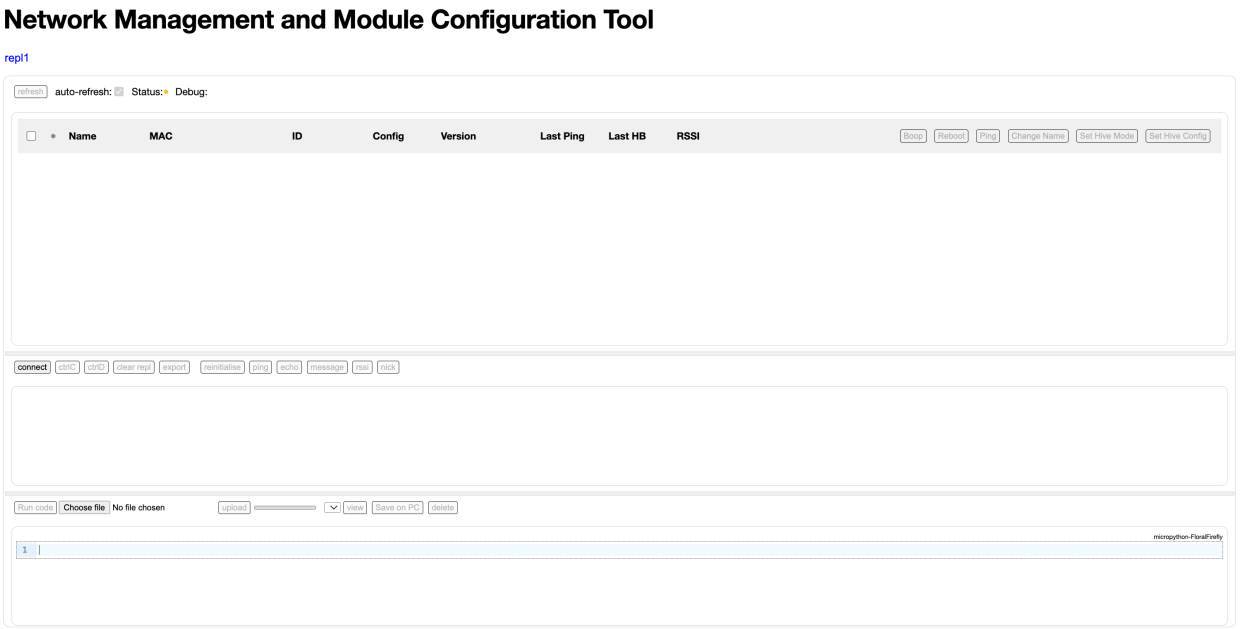


Figure 2.16. Network Management and Module Configuration Tool

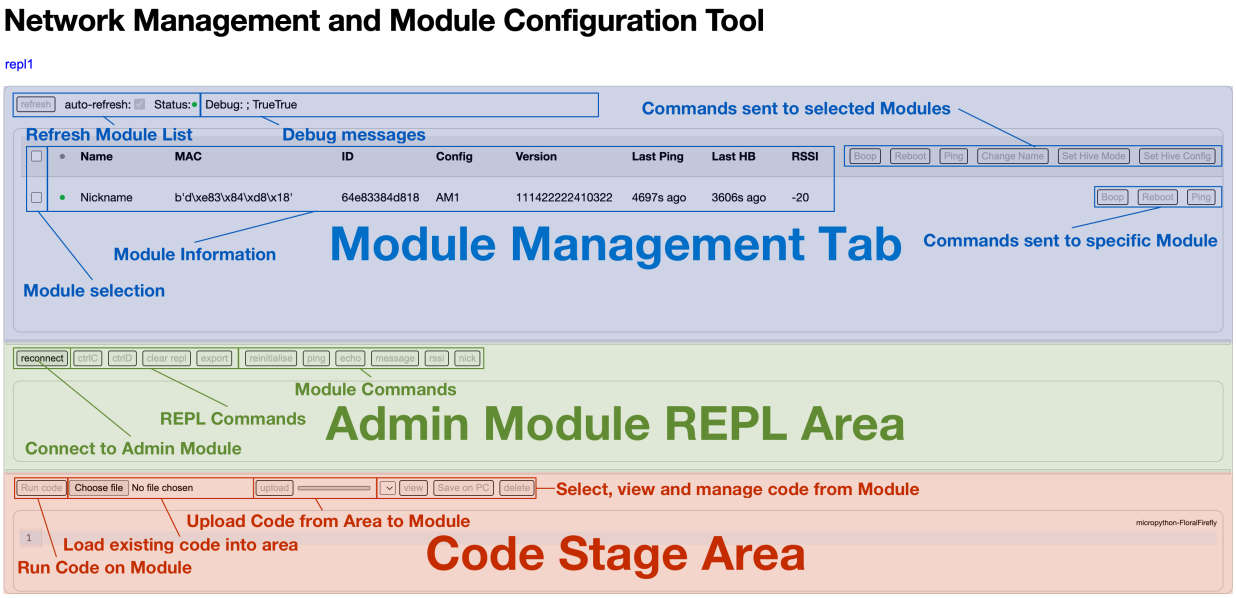


Figure 2.17. Annotated Network Management and Module Configuration Tool

The module management area, as the name suggests, is the area where all the modules that have been in contact with the admin module, i.e. are in the vicinity of the admin module and have responded to a *ping* message, are listed and can be managed directly from the list using various networking commands sent from the admin module to the respective modules. The module management area consists of several parts, the list of modules with their information, such as the status of the module, its name, its MAC address, its chip ID, its configuration, its code version, the time since the last ping and the last message received, as well as the RSSI value of the last transmission from the respective device as received by the Admin Module, and some specific commands to be sent to this module (*boop*, *ping*, *reboot*), a list header with the information titles, and additional commands to be sent to all modules selected by the checkbox, such as the *name change* command, the *hive set* command, which puts a device in hive mode and allows it to use its hive configuration to interact with other modules, and the *set hive configuration* commands, which set the hive configuration of a specific module. The panel also contains a panel header which provides controls for the panel, such as a refresh button which retrieves the latest information from the Admin module, an auto-refresh select button and a debug message box.

Figure 2.18. Network Management and Module Configuration Tool: Drop-down configuration feature

One version of the tool, specifically designed for use with the example *hive mode* program, which will be introduced in Section 2.1.1 and discussed in Section 3.3.1, includes drop-down menus to set specific configuration values, based on which the module will then interact with other modules. The various fields set are which module (MAC address) to send its sensor data to, from which module (MAC address) it should expect to receive sensor data, which specific sensor value to use and in what fashion it should use it(continuous or discrete). The various inputs and outputs are based on the specific module types. The feature is shown in Figure 2.18, and should further simplify the configuration and use of interconnected modules, while at the same time serving as an example and showing the potential of the system.

AI Module Configuration Assistant

In an effort to simplify the configuration of Smart Modules using the *Set Hive Config* command of the Network Management and Module Configuration Tool, an LLM, again using OpenAI's Chat-GPT, has been primed with documentation and instructions to take a description of the available modules, as well as the required interaction configuration, and return the respective values to be provided when sending the *configuration* command. This support feature was not included in the AI Code Assistant to avoid confusion, both on the side of the user and the LLM, given the very specific application and linear pipeline for the module configuration, compared to the more open-ended intended use of the AI Code Assistant.

Module Management Portal

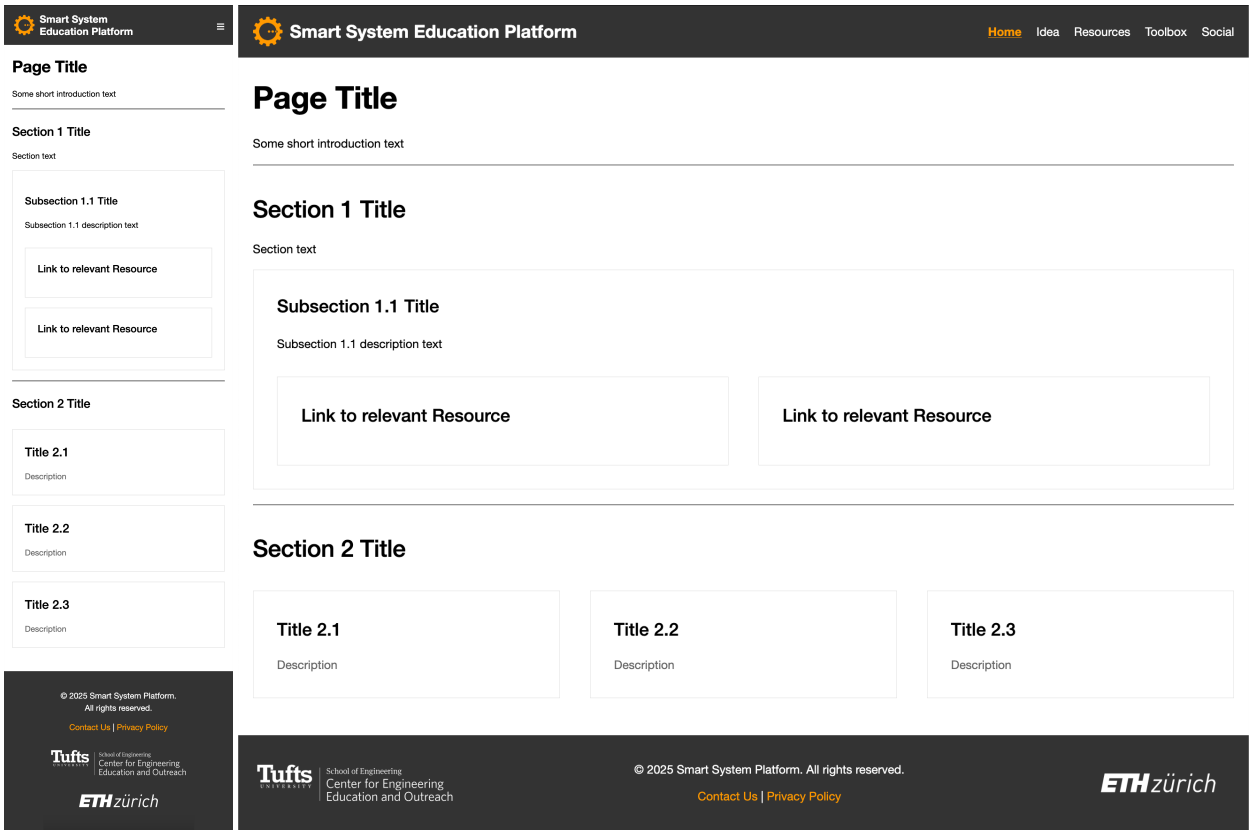
To update the code files on a given module, the module page has been developed using PyScript, which checks the *config.py* file for version numbers against the latest *config.py* file on the release branch of the GitHub page. If a mismatch is detected, the appropriate files are downloaded from GitHub to the webpage, which writes the files to the module, updating the software files on the device to the latest version based on its specific configuration.

Website

In addition to the GitHub, and to better provide an overview and introduction to the concept of the Smart System Platform and its networking capabilities, host guides, documentation, as well as the developed management and development tools, a website was created.

For simplicity sake, the website is hosted as part of the GitHub repository, which allows one

website to be hosted and comes with an automatic release pipeline for the website whenever a change is made to the underlying files in the *docs* folder.



(a) Mobile version (b) Desktop version

Figure 2.19. Website layout

To keep in line with the designing principles of keeping things simple, accessibility and with the main goal of conversation of information and outreach in mind, the minimalistic Swiss design style was chosen to convey the information in a clear and functional fashion, using a grid system, with function dictating form (Hollis, 2006; Müller-Brockmann, 2020). Both a version for desktop and mobile devices was developed, with the design draft of various elements of both being shown in Figure 2.19.

The website is divided into five parts: the *home* page or *index* page, which gives a brief introduction and overview of the Smart System Platform and provides links to the main resources; the *idea* page, which describes the concept of the Smart System Platform, the networking approach and provides technical background; the *resources* page, which contains a guide to getting started with the Smart System Platform, its tools and the Networking Library and links various other resources; the *toolbox* page, which links to the various tools developed, such as the IDE, the

Network Management portal, the Module Management portal and the primed AI chatbots; and the *social* page, which provides background on the involved parties and their contact details.

Software

Various code and libraries have been written and collected as part of the platform, in addition to the Networking Library designed and developed in Section 2.1.3 and Section 2.2.2 respectively. An overview of all SSP-related code and libraries, including their purpose and source where applicable, can be found in Table 2.10.

The various configuration files, such as *prefs.py* and *version.py* from the Smart Motors (see Section 2.1.1), have been merged into *config.py*, which contains other configuration values necessary for the networking and normal operation of the modules, such as the module configuration, id, code version numbers, and the STA and AP Wi-Fi channel on which the devices are operating. Except for the module type and its name, the values are set automatically when the module is powered up. The file also contains dictionaries of network type and subtype keys and their codes, network handshake keys and codes, a whitelist of MAC addresses from which certain commands are accepted, and Wi-Fi network secrets for connecting to a specific Wi-Fi. The file also contains an I2C dictionary of common peripheral hardware I2C addresses and a dictionary of sensors and their respective range of output values, as well as the hive configuration, which includes values for which MAC addresses to send sensor data to, which devices to expect data from and which of their sensor values to use, how to use them, the rate at which data should be sent and output calculated, and more.

A customisable boot program has also been written, which automatically calls the appropriate module main program from within *boot.py*, based on a value in *config.py*. The module's main program will first initialise the network so that the module can be found and receive messages and commands, set up everything based on its configuration, and then either run in hive mode or call the module's stand-alone main program. This allows the module to function as a stand-alone, but still receive and respond to commands using the Networking Library if necessary. These commands can be used to change its configuration and, for example, put it into hive mode. To enable certain hardware inputs and outputs, such as sensors, motors, lights, etc., various libraries are also required to support their functions, which are also included in the code list.

File	Purpose	Credit
config.py	Configuration File	Triebold *et al.* (2025)
boot.py	Loads main program based on module configuration	Triebold *et al.* (2025)
main.py	Empty	
am1.py	Admin Module main code	Triebold *et al.* (2025)
hm3.py	Hive Motor main code	Triebold *et al.* (2025)
sl1.py	Smart Light main code	Dahal and Cross (2025)
sm3.py	Smart Motor main code	Dahal and Cross (2025)
sp1.py	Splat Module main code	Triebold *et al.* (2025)
networking.py	Library to enable ESP-NOW based networking for modules	Triebold *et al.* (2025)
ssp_networking.py	Library with SSP-specific networking add-ons and custom cmd types and handlers	Triebold *et al.* (2025)
adxl345.py	Library to support the built in accelerometer	NanaWangDFR *et al.* (n.d.)
files.py	Library to simplify file saving and reading for Smart Motor	Dahal and Cross (2025)
icons.py	Library with icons for the screen of the Smart Motor	Dahal (2024)
sensors.py	Library to support sensors of the Smart Motor and Dahal Board	Dahal and Cross (2025)
servo.py	Library to support servo of the Smart Motor	Dahal and Cross (2025)
smartlight.py	Library to support the LED light ring of the Smart Light	Dahal and Cross (2025)
splat.py	Library to support the components of the Splat module	Hankin *et al.* (n.d.)
ssd1306.py	Support for the built in OLED screen	Lehmann and Scheller (n.d.)
variableLED.py	Library that powers the variable LED grid	Hankin *et al.* (n.d.)

Table 2.10. Overview of all the software files, including libraries and their purpose

Hardware

During the development of the Networking Library and the Smart System Platform, in terms of hardware, Smart Motors were used, alongside other prototyped derivatives such as the Smart Light, which is a Smart Motor which uses an LED ring instead of a motor as output, and standalone Dahal Boards, as well as certain prototypes from the Smart Playground project such as the Splat Module and the Button Module. The Dahal Board especially, proved to be a useful asset, thanks to its many interfaces, outlined in Section 2.3, allowing plug and play with a variety of sensor inputs and outputs. An overview of the possibilities, based on which any Smart Modules could be developed, can be seen in Figure 2.20.

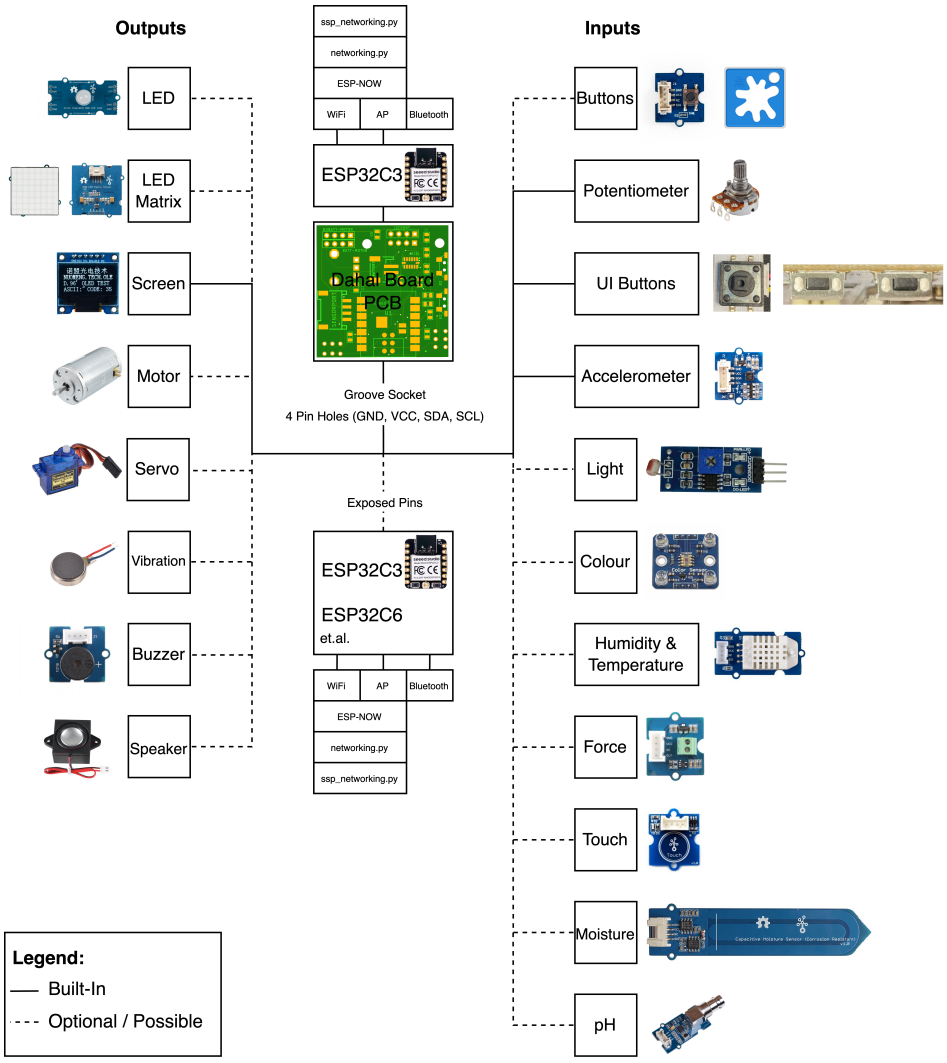


Figure 2.20. Hardware matrix with possible inputs and outputs for Smart Modules

As part of the ongoing Smart Playground project (Blake-West *et al.*, 2025), which will be further introduced in Section 2.3.3, certain modules have been developed that are consistent and

compatible with the Smart System Platform approach, in particular using the developed Networking Library, the use of which is described in Section 3.3.4. The hardware developed for the Smart Playground project, shown in figure 2.21, includes a Button Module and a Splat Module, based on the Splat toy, but both using an ESP32C6 MCB (see Figure 2.6), as well as Plushies and Control Boards, based on the Dahal board, using an ESP32C3 MCB (see Figure 2.5), both introduced in Section 2.1.2.

(a) Smart Light (b) Plushie Module (c) Button Module (d) Smart Splat

Figure 2.21. Various modules developed based on the Smart Motors project (Blake-West *et al.*, 2025) (a) and by the Smart Playground project (Blake-West *et al.*, 2025) (b, c and d)

Firmware

A custom firmware was created for the Smart Motors with additional packages and libraries, and it was briefly considered to do the same for the Smart System Platform and create a custom firmware that include certain libraries, in particular the Networking Library. However, for the sake of accessibility and interoperability the base MicroPython firmware was instead used.

2.3 Phase 3: Testing and Validation

In a final phase, the developed Networking Library was tested. Furthermore, the various components of the Smart System Platform, including the tools and documentation, were evaluated and both the Networking Library and SSP were utilised, both in a test setting and in other projects.

2.3.1 Networking

The following tests were designed during the development of the Networking Library to evaluate, validate, and assess its networking capabilities in relation to the first research question.

Reliability

In an effort to assess and test the reliability and sturdiness of the Networking Library with a larger amount of modules, a program called the *boop-o-meter*[4] was created. The program runs on Dahal-Board hardware and uses a simple interaction design, discovering all modules in its vicinity using the *ping* functionality of the Networking Library, and then lists all the discovered modules in a list on the screen and then allows a message to be sent to the highlighted MAC address in a list of MAC addresses of discovered devices, which also includes the broadcast address, keeping count of received and sent messages. The test was conducted during Hackathon 1 with 18 devices, where the goal was to send and receive 999 messages. The results of this test are outlined in Section 3.1.7 and the code can be found in Appendix E.

Maximum Range

A further test was carried out to determine the maximum range at which messages could still be received, using the same programme as described for the robustness test, the code for which can be found in Appendix E. Given the range requirements for this test, the test was conducted outdoors on a straight road between the CEEO offices and the Tufts campus to ensure an uninterrupted line of sight. Two people, connected via phone call, walked away from each other on the street and periodically sent messages to each other from the MCB. At the point where neither person was receiving messages, their positions were marked and the final distance was calculated using Google Earth. The results of this test are discussed in Section 3.1.2. This test was conducted once in isolation as a reference and to validate the range capabilities outlined in the XIAO ESP32C3 documentation (Seeed Studio, 2024b).

Response Time, RSSI and Packet Loss Rate by Range

To test the reliability of the transmission and to get an idea of the response time, RSSI values and packet loss by range, a test was carried out at different distances. As this code was intended to demonstrate the technical capabilities of the underlying ESP-NOW protocol, a specific code was written for this test, reducing the code and networking to the absolute minimum. The code was designed to send 100 messages to the broadcast address, with the receiving module designed to echo the messages immediately on receipt. Sending was triggered manually and message exchange was monitored with print statements. The various modules logged relevant data, such as the message ID (to calculate the number of dropped messages), the time of reception and transmission to calculate the ping time of a message sent back and forth, and the RSSI value. In a separate step, the logged raw values were then written to a file, and the various

[4]The name is a reference to he Tumblr April 1st joke in 2021, by which the tool was inspired

values of interest were calculated and stored in a separate step at the end. The code was kept to a minimum and the logged data was written after the fact to allow sufficient time for all messages to be received and returned. The test was carried out in two stages: for distances of less than one metre, the test was carried out in the laboratory, while for longer distances, in order to ensure a clear line of sight between the two devices, given the distance requirements, the test was carried out outdoors on the roof of the car park outside the CEEO offices at 200 Boston Avenue, MA, USA. The results of this test are discussed in Section 3.1.3 and the corresponding code can be found in Appendix E.

A further response time test was conducted using the Networking Library, to measure the time taken up by the additional calculation burden imposed by the library, which is discussed in Section 3.1.4.

Antenna Angle Effects on RSSI and Ping

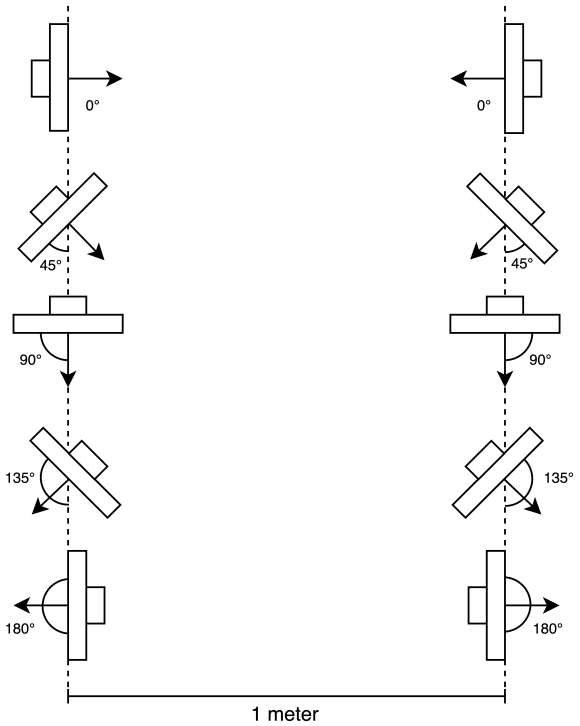


Figure 2.22. The various setups to test antenna angle effects on response time, RSSI and packet loss rate

Based on anecdotal observations from the Smart Playground project using ESP32C3 and ESP32C6 devices with the developed Networking Library and SSP Add-on Library, differences in RSSI values, which were used to approximate range, were observed for different devices. Therefore

effect of the relative antenna angle between two units was tested at a predefined distance of one metre, as outlined in Figure 2.22. The same program used to calculate the response time, RSSI values and packet loss rate for different distances was used for this test, the code for which can be found in Appendix E. The results of this test are presented in Section 3.1.5.

Power Consumption and Battery Endurance

Another test that was prepared after problems with power consumption were discovered by the Smart Playground project when using networking in conjunction with various additional components such as audio output and LED lights, powered by three AA batteries instead of the usual LiPo batteries. After about ten minutes the power was too low to run all three components simultaneously. To this end, a test was designed and carried out to determine the power consumption of the Networking Library when continuously sending and receiving messages. Starting with a full LiPo battery, *echo* type messages were sent every second, while a receiver connected to a laptop continued to log the time since startup when the last message was received. This test was repeated three times with different devices and batteries, specifically using one 3.7 V 350 mAh for the first test and two 3.7 V 420 mAh batteries for the second and third test. The code for this test can be found in Appendix E, while the results are outlined in Section 3.1.6.

2.3.2 Smart System Platform

In addressing the second research question, while an accessibility and design test for the website was conducted using Google Developer's PageSpeed Insights ("PageSpeed Insights", n.d.), a comprehensive validation of the Smart System Platform can only be achieved through continued use, as discussed in Section 2.3.3. To facilitate this, the Networking Library and Smart System Platform were tested and utilised during two hackathons, detailed in Sections 2.3.3 and 2.3.3.

2.3.3 Utilisation and Outcomes

This section explores how the Networking Library and Smart System Platform were utilised in various contexts.

Example Applications

Some example applications were developed by the author during the research, using the developed Networking Library and Smart System Platform and its tools, such as the *boop-o-meter*

program, Smart Motors controlling each other and the *hive mode* program, which describes the Smart Modules set up to be configurable using the Network Management and Module Configuration Tool. The results of this are described in Section 3.3.1.

Hackathon 1

The focus of the first hackathon was to test the robustness of the Networking Library and ESP-NOW and to explore the use and usability of the Networking Library. The hackathon was therefore divided into several parts, starting with an introduction to the research, the Networking Library and the proposed Smart System Platform. This was followed by a robustness test using the *boop-o-meter* program, as described in Section 2.3.1. As a next step, the participants were tasked with brainstorming potential uses and applications of such a networked device, and then using the provided Networking Library and example code to build a small project using Smart Motors hardware. The results are outlined in Section 3.3.2.

Hackathon 2

To address the third research question, the focus of the second hackathon was to test the Smart System Platform as a whole, focusing on the usability of the development tools and the Networking Library, and to find out what students could do with the tools and capabilities provided. The hackathon started with an introductory presentation about the project, the Networking Library and the Smart System Platform approach, as well as a quick overview of the tools. Participants were then tasked in small groups to try out the tools and develop a small interactive project using the Networking Library and Smart System Platform tools. Feedback on the usability as well as the application potential was collected via a questionnaire, the results of which are discussed in Section 3.3.3.

Smart Playground

The Smart Playground project is a project run in cooperation by Tufts CEEO, Boston University and University California, Irvine, which was being developed at the same time as this research. The main goal of the project is to create interactive playground and classroom experiences to teach computational thinking. The approach taken was to use interactive modules, some of which are carried or held individually by children, and others in the form of stations such as buttons or adapted Splats originally developed by Unruly Studios (n.d.), which are a sort of stomp pads (see Section 2.2.3). As such, the project required a way for the different modules to communicate and interact with each other, which was exactly the capability developed as part of this research in the form of the Networking Library. The project also based its hardware on the

Dahal Board and ESP32 MCBs, which are fully compatible and in line with the aims of the Smart System Platform, one of the goals of which is to enable and facilitate such development, leading to some collaboration. The project relied on the Networking Library to develop the software to allow interaction of hardware for its initial tests, and used some of the ideas outlined in Section 2.1.4 and Section 2.2.3 to develop hardware that is compatible and in line with the idea of the idea of Smart System Platform introduced in Section 2.1.4. The use of the Networking Library by the Smart Playground project is further explored and discussed in Section 3.3.4.

3 Results and Discussion

3.1 Networking

As outlined in Section 2.1.3 and Section 2.2.2, and in accordance with research question 1, a networking structure has been conceptualised and implemented for use with the Smart Motors-based hardware.

3.1.1 Concept, Structure and Logic

As outlined in Section 2.1.3 and Section 2.2.2, the networking was designed and developed in two parts: the Networking Library and, using this library as a base, the SSP Add-on Library.

The Networking library has been designed to serve as a general-purpose library, building upon ESP-NOW while incorporating essential structural and functional elements. It introduces address book logic that bypasses ESP-NOW's 20 peer limit, allowing transmission to a theoretically unlimited number of peers, though at the expense of efficiency. The library also introduces the fundamental message structure, including various message types (*cmd*, *inf*, *ack*) and different subtypes (*cmd*: *ping*, *echo*, *boop*; *inf*: *msg*, *data*; *ack*: *pong*, *echo*, *boop*, *confirm*, *success*, *fail*). The library also includes logic to send data larger than 241 $bytes$, the maximum possible payload per message using the message structure defined in Section 2.1.2. This was increased to a theoretical limit of 60′928 $bytes$. In reality, however, the practical limit is lower, with the maximum amount successfully transmitted being about 30 kB due to the memory limitations of the MicroPython firmware running on the ESP32C3. Additionally, the transmission of large messages in chunks entails the risk of some chunks being lost during transmission (see Section 3.1.3), especially over long distances, resulting in the entire message being dropped. The code includes a provision to remedy this issue through the use of a buffer, wherein all components of a long message are stored. In the event that the receiving device fails to receive one or more parts of a multi-part message after a certain duration, it initiates a request for the missing parts. However, this functionality has been disabled to optimise memory usage. The network design

incorporates a range of interfaces for customisation, with the potential for the inclusion of additional custom commands, message types and handling logic, including custom IRQ messages.

The SSP Add-on Library has been developed as an extension of the base Networking Library, introducing a range of additional Smart Module-specific commands, handlers and message types. These enable the effective control and configuration of Smart Modules through the utilisation of designated commands. The architecture of the code for both libraries, along with the full suite of commands available in each, is illustrated in Figure 2.10.

As such, the Networking Library fulfils the requirements outlined in Section 2.1.3. Notably, it is compatible with the Smart Motors hardware and with any ESP32-based hardware that includes Wi-Fi and thus ESP-NOW capabilities, as well as with the MicroPython firmware used. The protocol uses a peer-to-peer communication approach, inherited from ESP-NOW protocol, on which it is based, and requires no central root or hub, nor any configuration to use, other than initialising the library. The library also includes message validation features, employing a common message structure that includes an identifier and checksum. It further facilitates module discovery, using the *ping* command, and interaction via various further defined commands and message types specified in Networking Library and the SSP Add-on Library, thereby satisfying the latter two requirements. In terms of design principles, the networking is split into two parts, allowing the base networking, which contains simple logic and commands, to be used in a variety of custom ways, further facilitated by interfaces for customisation. Finally, the code adheres to the Style Guide for Python Code based PEP 8 (Rossum *et al.*, 2001).

3.1.2 Maximum Range Test

As described in Section 2.3.1, the original range test was conducted using the *boop-o-meter* program on a street near the CEEO offices. The maximum range at which some messages were still being received by each of the two ESP32C3 MCBs was approximately 242 meters, as shown in Figure 3.1.2, as measured using Google Earth.

Figure 3.1. Approximate maximum range test on Boston Avenue, MA

3.1.3 Response Time, RSSI and Packet Loss Rate by Range

The RSSI, ping response and packet loss for various ranges were measured three times using two ESP32C3s, one ESP32C3 and one ESP32C6 and again using two ESP32C6s.

ESP32C3 to ESP32C3

As anticipated, RSSI values, an index of signal strength, decrease for extended ranges, following a logarithmic curve, as shown in Figure 3.1.3. The values for both devices are similar and follow a similar curve, with marginal divergence, likely attributed to material variations.

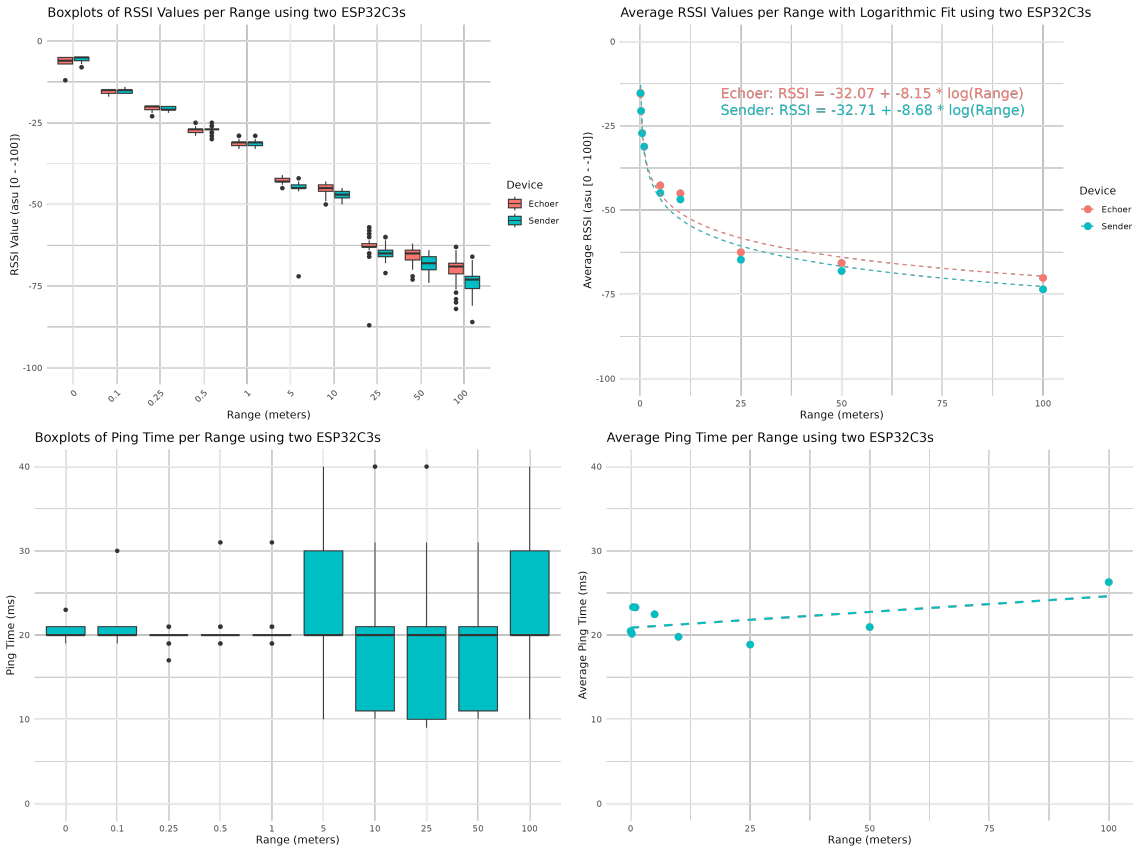


Figure 3.2. RSSI and ping response time depending on range using two ESP32C3s

With regard to ping time, values remain largely unaffected by range, with a consistent median value of 20 *ms* and mean values consistently around 20 *ms*. The increase in travel time from 0 to 100 meters of the radio wave is only approximately 0.0000033356 *ms* (100 *m* / 299′792′458′000 *m*/*ms*), given the travel at the speed of light, which is insignificant compared to the time required to send and process the message by the MCB. Of particular interest is the observation that values are clustered at specific intervals of 10 *ms*, 20 *ms*, and 30 *ms*, which may be attributable to an underlying system clock rate of the MCB.

Range	Packet Loss	Measurement		Values				
[meters]	[%]			mean	std	min	max	median
0 m	0	RSSI 1	[asu]	-6.05	1.14	-12	-5	-6
	0	RSSI 2	[asu]	-5.44	0.67	-8	-5	-5
		Ping	[ms]	20.57	0.61	19	23	20
0.1 m	0	RSSI 1	[asu]	-15.5	0.59	-17	-15	-15
	0	RSSI 2	[asu]	-15.25	0.54	-16	-14	-15
		Ping	[ms]	20.36	1.08	19	30	20
0.25 m	0	RSSI 1	[asu]	-20.59	0.72	-23	-20	-20
	0	RSSI 2	[asu]	-20.58	0.53	-22	-20	-21
		Ping	[ms]	20.16	0.58	17	21	20
0.5 m	0	RSSI 1	[asu]	-27.22	0.73	-29	-25	-27
	0	RSSI 2	[asu]	-27.15	0.77	-30	-25	-27
		Ping	[ms]	23.32	22.42	19	223	20
1 m	0	RSSI 1	[asu]	-31.14	0.93	-33	-29	-31
	0	RSSI 2	[asu]	-31.18	0.74	-29	-33	-31
		Ping	[ms]	23.29	22.11	19	219	20
5 m	0	RSSI 1	[asu]	-42.67	0.97	-45	-41	-43
	0	RSSI 2	[asu]	-44.91	2.87	-72	-42	-45
		Ping	[ms]	22.47	9.34	10	50	20
10 m	0	RSSI 1	[asu]	-45.03	1.23	-50	-43	-45
	0	RSSI 2	[asu]	-46.83	1.01	-50	-45	-47
		Ping	[ms]	19.8	7.52	10	41	20
25 m	3	RSSI 1	[asu]	-62.54	3.17	-87	-57	-63
	3	RSSI 2	[asu]	-64.72	2.04	-71	-60	-65
		Ping	[ms]	18.88	7.46	9	41	20
50 m	12	RSSI 1	[asu]	-65.69	2.38	-73	-62	-65
	12	RSSI 2	[asu]	-68.08	2.38	-74	-64	-68
		Ping	[ms]	20.95	11.22	10	91	20
100 m	32	RSSI 1	[asu]	-70.12	3.61	-82	-63	-69
	34	RSSI 2	[asu]	-73.52	3.27	-86	-66	-73
		Ping	[ms]	26.29	18.17	10	141	20

Table 3.1. RSSI, ping response time and packet loss measurements for various ranges using two ESP32C3s

There are various outliers in terms of ping time, notably for greater distances, though also for the measurements at 1 and 0.5 meters. The outliers observed in these two measurements were the first measurements of the series, both presumably taken after a restart of the used ESP32 MCBs. Consequently, the code had to add the MAC address it had received the message from to its ESP-NOW buffer, which might explain some of the delay and the elevated ping response time for that first measurement. Conversely, subsequent measurements taken at the same distance, for example for the angle tests in Section 3.1.5, did not yield any such outliers. For other test sessions the code was first run as a test of functionality, with data disregarded, which might be the reason why this issue is only apparent for these two specific measurements.

As for packet loss, transmission remains lossless until 25 meters, at which point packets start to become lost. These values increase with distance, with approximately one third of messages lost at a distance of 100 meters, as shown in Table 3.1.

ESP32C3 to ESP32C6

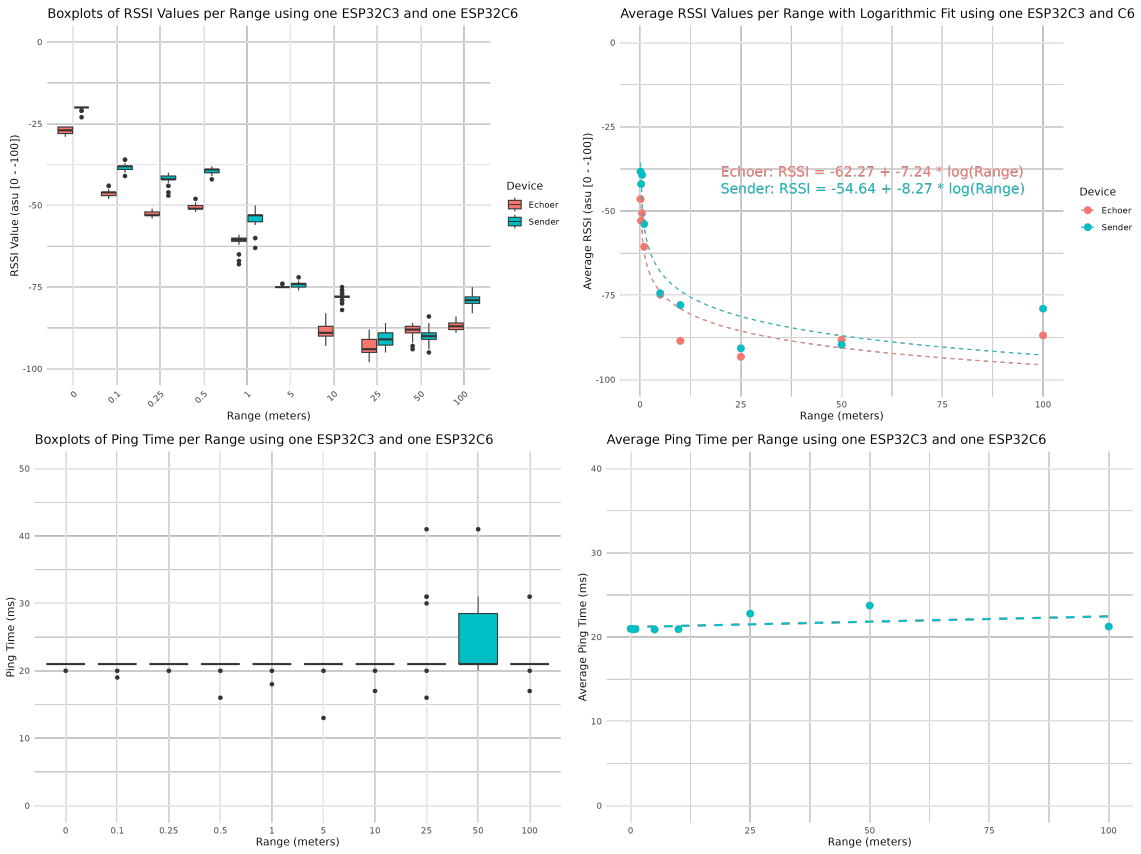


Figure 3.3. RSSI and ping response time depending on range using one ESP32C3 and one ESP32C6

Analogous results were obtained using an ESP32C3 as the sender and an ESP32C6 as the echoer. The RSSI values followed a similar logarithmic trajectory, although in this instance the values were considerably lower for both devices, likely attributed to the smaller on-board antenna used by the ESP32C6. The RSSI values for the sender (ESP32C3) remained significantly higher for most measurements, except for the 5 meter and 50 meter measurements, where the values were found to be similar. Surprisingly, the RSSI values for 100 meters were better than both 50 and 25 meters, with the lowest values for the test occurring at 25 meters. This might be explained by the intermittent fluctuation of RSSI values. While during standard deviation remains small, repeating the same measurement at the same range one a different day, can result in different RSSI values. The underlying cause of this variability remains unclear.

The ping values also exhibited a uniform distribution, with an average of around 20 *ms*, with some minor outliers observed for greater distances.

In terms of packet loss, a parallel trend to RSSI values is observed, with packet loss at the highest for the measurement at a range of 25 meters, with a similar amount of packets lost each direction. Notably, no packets were lost for the measurement at 100 meters.

However, a single, isolated and unexplained inconsistency in the data is observed. Specifically for the 10 meter measurement, the echo device did not receive two packets (number 62 and 72), yet the sender did receive the return echo of these two packets. This discrepancy suggests the presence of an error, yet the raw data collected indicates that packets 62 and 72 were successfully received with a timestamp assigned to the sender. However, there are no records for these packets on the echo device. One possible explanation for this discrepancy is a bug in the test code, such as the dictionary not being cleared or reinitialised properly, retaining previous values. However, this appears improbable, as the dictionary is cleared and reinitialised empty with every new run of the code, as well as when data is written to files. Consequently, the most plausible explanation appears to be an error on the part of the echoer in writing the data to the dictionary. The echo logic and response are based on IRQ functions, so an error would not have interrupted the test and given the lack of logging, such an error may have gone unnoticed.

Range	Packet Loss	Measurement		Values				
[meters]	[%]			mean	std	min	max	median
0 m	0	RSSI 1	[asu]	-6.05	1.14	-12	-5	-6
	0	RSSI 2	[asu]	-27.13	0.93	-29	-26	-27
		Ping	[ms]	20.98	0.14	20	21	21
0.1 m	0	RSSI 1	[asu]	-46.43	1.12	-44	-48	-46
	0	RSSI 2	[asu]	-38.22	1.04	-41	-36	-38
		Ping	[ms]	20.96	0.24	19	21	21
0.25 m	0	RSSI 1	[asu]	-52.86	0.72	-54	-51	-53
	0	RSSI 2	[asu]	-41.97	1.02	-47	-40	-42
		Ping	[ms]	20.97	0.17	20	21	21
0.5 m	0	RSSI 1	[asu]	-50.61	0.72	-52	-48	-51
	0	RSSI 2	[asu]	-39.26	0.66	-42	-38	-39
		Ping	[ms]	20.92	0.52	16	21	21
1 m	0	RSSI 1	[asu]	-60.58	1.39	-68	-59	-60.5
	0	RSSI 2	[asu]	-53.84	1.6	-63	-50	-53
		Ping	[ms]	20.95	0.33	18	21	21
5 m	0	RSSI 1	[asu]	-74.77	0.42	-75	-74	-75
	0	RSSI 2	[asu]	-74.36	0.76	-76	-72	-74
		Ping	[ms]	20.9	0.81	13	21	21
10 m	2	RSSI 1	[asu]	-88.48	2.02	-93	-83	-89
	0	RSSI 2	[asu]	-77.89	0.87	-82	-75	-78
		Ping	[ms]	20.94	0.42	17	21	21
25 m	17	RSSI 1	[asu]	-93.24	2.51	-98	-88	-94
	26	RSSI 2	[asu]	-90.73	1.33	-95	-86	-91
		Ping	[ms]	22.78	4.31	16	41	21
50 m	0	RSSI 1	[asu]	-88.18	1.49	-94	-86	-88
	6	RSSI 2	[asu]	-89.67	1.73	-95	-84	-90
		Ping	[ms]	23.74	4.94	20	41	21
100 m	0	RSSI 1	[asu]	-86.9	1.17	-89	-84	-87
	0	RSSI 2	[asu]	-78.95	1.6	-83	-75	-79
		Ping	[ms]	21.25	1.76	17	31	21

Table 3.2. RSSI, ping response time and packet loss measurements for various ranges using one ESP32C3 and one ESP32C6

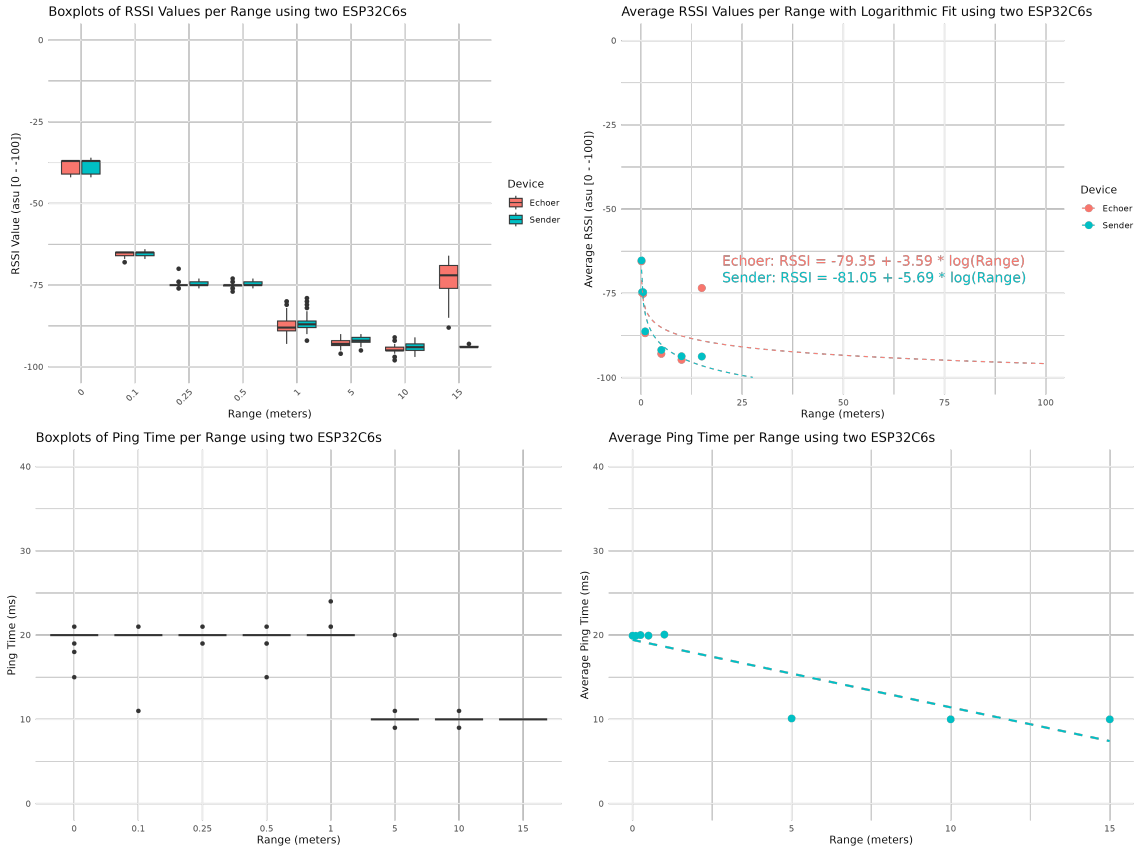


Figure 3.4. RSSI and ping response time depending on range using two ESP32C6s

However, repeating the same test with two ESP32C6s, difficulties were encountered with transmission over a distance exceeding 15 meters. As transmission with an ESP32C3 (sender) and an ESP32C6 (echoer) was achieved up to 100 meters, the experiment was repeated with various ESP32C6 chips, to rule out hardware issues. However, similar results were obtained, though in some trials the maximum distance was even lower. One hypothesis to explain these issues are problems with the antenna, or rather the wrong antenna interface being used. Given that the MCB is equipped with two antennas, a built-in one and a connector, it is hypothesised that it may have been utilising the antenna connector, with which short-range transmission is possible even in the absence of a connected antenna. However, it is important to note that by default, the chips are configured to use the onboard antenna, and this setting appears to be set correctly.. However, upon connecting an antenna to the interface, an increase in RSSI values was observed. Furthermore, the RSSI values measured with the on board antenna / without a connected external antenna exhibited a degree of similarity to those received when using an ESP32C3 without an antenna over short ranges, thereby substantiating this theory. However,

in the prior test, transmission over a range of 100 meters was achieved with an ESP32C6 in connection with an ESP32C3 MCB, and hence it seems unlikely that the wrong antenna interface was used. Hence, a further hypothesis to explain this phenomenon is that the problem is simply caused by the ESP32C6's smaller on-board antenna.

In terms of RSSI values, the overall values were lower in comparison to the test conducted with an ESP32C3 and an ESP32C6, a phenomenon that could be explained by the two smaller antennas. The results also follow the expected logarithmic curve. However, the RSSI values of the receiver at a distance of 15 meters are significantly higher than those at shorter ranges and also compared to the RSSI values of the sender. Though the reliability of this data point is questionable, as it is only compromised by four values due to the high rate of packet loss (96%) at this range.

With regard to the ping values, these were found to be consistent with expectations, albeit at a slightly lower level at higher ranges. This is likely to be a consequence of the reduced computational burden due to the higher packet loss. An examination of packet loss reveals that only a minimal number of messages were lost for 5 and 10 meters, respectively. However, at 15 meters, 1% of packets were lost during outbound transmission, while 95% of packets were lost during return transmission.

Range	Packet Loss	Measurement		Values				
[meters]	[%]			mean	std	min	max	median
0 m	0	RSSI 1	[asu]	-38.27	1.98	-42	-37	-37
	0	RSSI 2	[asu]	-38.14	1.97	-42	-36	-37
		Ping	[ms]	19.93	0.55	15	21	20
0.1 m	0	RSSI 1	[asu]	-65.51	0.74	-68	-65	-65
	0	RSSI 2	[asu]	-65.23	0.66	-67	-64	-65
		Ping	[ms]	19.92	0.9	11	21	20
0.25 m	0	RSSI 1	[asu]	-74.88	0.64	-76	-70	-75
	0	RSSI 2	[asu]	-74.64	0.54	-76	-73	-75
		Ping	[ms]	20	0.14	19	21	20
0.5 m	0	RSSI 1	[asu]	-75.16	0.61	-77	-73	-75
	0	RSSI 2	[asu]	-74.66	0.53	-76	-73	-75
		Ping	[ms]	19.94	0.53	15	21	20
1 m	2	RSSI 1	[asu]	-86.83	2.98	-93	-80	-88
	2	RSSI 2	[asu]	-86.31	2.82	-92	-79	-87
		Ping	[ms]	20.06	0.42	20	24	20
5 m	1	RSSI 1	[asu]	-92.95	1.11	-96	-90	-93
	1	RSSI 2	[asu]	-91.81	1.04	-95	-90	-92
		Ping	[ms]	10.1	1.01	9	20	10
10 m	0	RSSI 1	[asu]	-94.73	1.3	-98	-91	-95
	3	RSSI 2	[asu]	-91	1.31	-91	-97	-94
		Ping	[ms]	10	0.14	9	11	10
15 m	1	RSSI 1	[asu]	-73.42	4.9	-88	-66	-72
	96	RSSI 2	[asu]	-93.75	0.43	-94	-93	-94
		Ping	[ms]	10	0	10	10	10

Table 3.3. RSSI, ping response time and packet loss measurements for various ranges using two ESP32C6s

3.1.4 Networking Library Ping Response Time

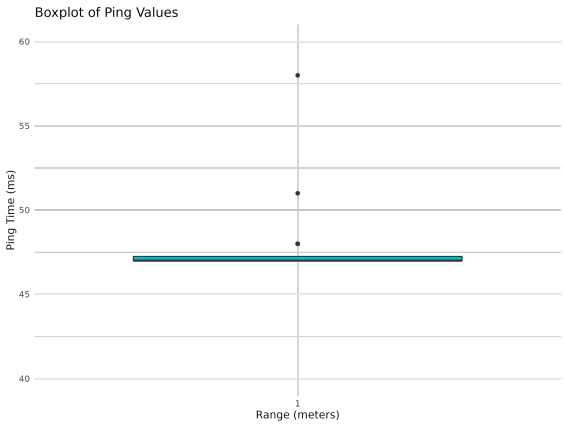


Figure 3.5. Ping response time using two ESP32C3s with the Networking Library and SSP Add-on Library

A ping test using the Networking Library and the SSP Add-on Library was conducted at a fixed range of one meter. The test was only performed at a fixed range, given the lack of variance in ping response time observed in Section 3.1.3 for various ranges. This was done for the purpose of comparison with the ping response time achieved by the test code used for the tests discussed in Section 3.1.3 and Section 3.1.5. The average value of the ping response time is approximately twice as high (47.38 *ms*) compared to the values observed using the basic ESP-NOW code (23.29 *ms* and $20.04 - 20.14$ *ms* respectively). This is likely due to increased processing demands and the necessity to add the peer's MAC address to the ESP-NOW peer buffer before each transmission and remove it afterward. The ping time demonstrates a high degree of consistent, with a standard deviation of only 1.21 *ms*, as shown in Figure 3.5 and Table 3.4.

Range	Packet Loss	Measurement		Values				
[meters]	[%]			mean	std	min	max	median
1 m	0	Ping	[ms]	47.38	1.21	47	58	47

Table 3.4. Ping time and packet loss measurements using the Networking Library and SSP Add-on Library

Using base ESP-NOW with minimal code, with no interrupt handlers and no other additional code, such as to save values for analysis, which might slow processing speed, the minimum value for ping response time achieved at minimal range under optimal conditions was 4 *ms*.

3.1.5 Antenna Angle Effects on RSSI and Ping Response Time

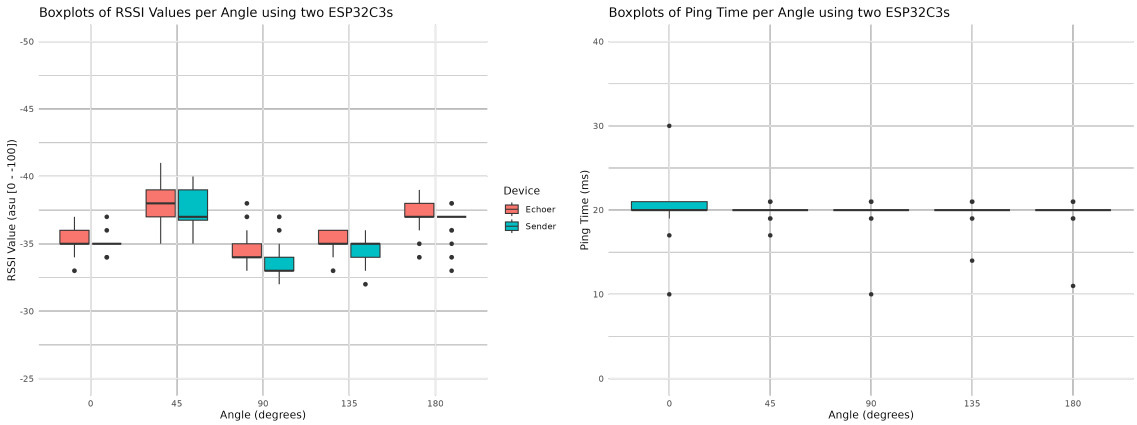


Figure 3.6. RSSI and ping response time depending on antenna angle

Angle	Packet Loss	Measurement		Values				
[°]	[%]			mean	std	min	max	median
0°	0	RSSI 1	[asu]	-35.71	0.6	-37	-35	-36
	0	RSSI 2	[asu]	-34.8	0.51	-37	-35	-35
		Ping	[ms]	20.2	1.53	10	30	20
45°	0	RSSI 1	[asu]	-37.74	1.47	-41	-35	38
	0	RSSI 2	[asu]	-37.47	1.49	-40	-35	-37
		Ping	[ms]	20.14	0.58	17	21	20
90°	0	RSSI 1	[asu]	-34.33	1.18	-38	-33	-34
	0	RSSI 2	[asu]	-33.68	1.15	-37	-32	-33
		Ping	[ms]	20.08	1.12	10	21	20
135°	0	RSSI 1	[asu]	-35.12	0.72	-36	-33	-35
	0	RSSI 2	[asu]	-34.64	0.7	-36	-32	-35
		Ping	[ms]	20.12	0.77	14	21	20
180°	0	RSSI 1	[asu]	-37.01	0.96	-39	-34	-37
	0	RSSI 2	[asu]	-36.73	0.86	-39	-33	-37
		Ping	[ms]	20.04	1.05	11	21	20

Table 3.5. RSSI, ping response time and packet loss measurements depending on antenna angle

This test was conducted on the basis of anecdotal observations from the Smart Playground project, as described in Section 2.3.1, which suspects that device and antenna orientation might

affect RSSI values. As illustrated in Figure 3.6, the ping time values remain largely consistent. While there is some slight variation in RSSI values, these are likely normal RSSI value fluctuations that were observed between different measurements, and therefore appear to be independent of the orientation of the MCBs and their antenna.

3.1.6 Power Consumption and Battery Endurance

As outlined in Section 2.3.1, the endurance of the battery of devices was tested. With a fully charged LiPo battery, the device was able to broadcast messages every second for a duration of a slightly over 4.5 $hours$ (16313723 ms) in the first test using a 3.7 V 350 mAh battery. In the two subsequent tests, a slightly extended runtime of approximately 5 $hours$ (18186793 ms and 18135873 ms) using a 3.7 V 420 mAh battery was achieved.

3.1.7 Reliability

A reliability test was conducted during the first Hackathon, the methodology of which is detailed in Section 2.3.3 and the results of which are discussed in Section 3.3.2. As part of this test, 18 participants used a Dahal Board running the *boop-o-meter* program to send messages to each other. The program kept track and count of the amount of messages sent and received, with the objective being to achieve 999 received and sent messages. During the test, no issues with transmission or reliability of the Networking Library or ESP-NOW were observed. However, given the high volume of messages and their peer-to-peer nature, it was not possible to track and validate every single message. The code used for this usability test, can be found in Appendix E.

3.1.8 Limitations

As outlined in the above results, the Networking Library is subject to certain limitations. One such limitation is inherent to the peer-to-peer connectionless networking model, in which no connection is established, with the system merely transmitting the message, akin to a radio station, with the notable distinction that the messages are directed to specific recipients. While the Networking Library does include a test for successful transmission based on the built-in ESP-NOW receipt confirmation, currently the Library only attempts to resend the message for a total of three attempts without any further failure handling. This is a deliberate choice to minimise the complexity of the Networking approach, but a limitation nonetheless and can result in messages being lost during transmission, as discussed in Section 3.1.3. While the transmission of messages using the library is reliable, there are circumstances and scenarios where this

might be inhibited. Hence, in scenarios in which reliable message delivery is essential, the Networking Library might not be able to deliver the required reliability.

A further limitation of the Networking Library is the increased processing time in comparison to the transmission time. Specifically, the Networking Library requires approximately 47 *ms* for a ping, whereas the fastest back-and-forth transmission achieved using ESP-NOW with MicroPython firmware was 4 *ms*. This discrepancy illustrates a limitations of MicroPython, particularly its speed and memory constraints. However, it also highlights the potential of the technology if it were to be optimised and written in a lower-level language such as C++, which could enhance its efficiency in a resource-constrained environment such as the MCB. This inefficiency can hinder scalability and responsiveness for larger and more complex projects. This represents the trade-off between optimisation, feature richness and accessibility.

During testing, particularly the RSSI, ping response time and packet loss measurement test using two ESP32C6 MCBs, issues were encountered. These tests revealed challenges in achieving a transmission distances greater than 15 meters using two ESP32C6 MCBs, which could be indicative of inconsistencies in the underlying hardware or network transmission performance that could affect overall reliability of the system. These issues, however, were not observed with the ESP32C3, which could transmit up to a maximum range of 240 meters, with reliable transmission possible up to 50 meters, at which point packet loss became an issue.

The development process further revealed some issues with the underlying ESP-NOW implementation for MicroPython. Once the Networking Library had become sufficiently complex, unknown Wi-Fi errors and memory errors were encountered, in addition to errors with Thonny and its interaction with ESP32-based MCBs. The unknown Wi-Fi error, in particular, proved to be a significant challenge in the troubleshooting process, further complicated by the fact that once this error occurred, the initialisation of the networking was no longer possible, necessitating the re-flashing of the firmware onto the MCB. While some reports of this error were available online, they provided no definitive solution. Through rigorous debugging, a cause was identified, with the length of the Networking Library files being the primary factor. It was observed that the error manifested itself once the library exceeded approximately 1000 lines in length. This finding has influenced the decision to divide the library into two parts: the Networking Library and the SSP Add-on Library. Following this decision, no further issues were encountered.

3.2 Smart System Platform

In order to validate the design of the Smart System Platform and to support the core networking capability, certain parts of the platform were developed as a proof of concept, as well as used and validated during the second Hackathon, which is discussed in Section 3.3.3. The various developed parts are presented in this section.

3.2.1 Platform and Framework Design

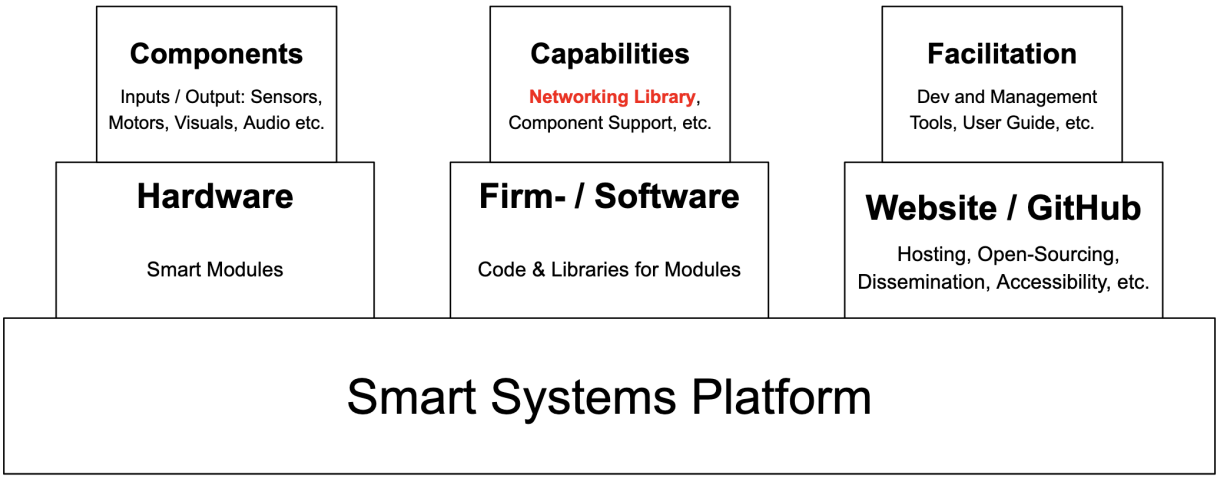


Figure 3.7. Simplified Smart System Platform architecture

In response to the second research question, which sought to ascertain methods for enhancing usability and accessibility of the developed networking capability for an educational robotics system such as the Smart Motors, the Smart System Platform was designed and partially developed. The platform is a basic framework, centred around the networking capability and is designed to be usable as a platform for the creation and development of network-enabled educational robotics projects.

The design of the Smart System Platform is outlined in Section 2.1.4 and depicted in a simplified version in Figure 3.7, is divided into three parts:

- Hardware:
  This component encompasses the supported hardware, designated modules, which can be any MCB based on an ESP3 SoC running MicroPython firmware. It also includes the

supported components, inputs and outputs, such as sensors and motors. The support is primarily based on the design of the Dahal board, which is the central component of the Smart Motors.

- Software:
  This component is situated at the core of the platform and includes the various capabilities that the platform has been designed around and to enable, namely the Networking Library. The software part also includes various libraries and components for component support.

- Documentation and Development:
  The third part of the platform is concerned with documentation and development, and includes the various tools and resources necessary to support the accessibility of the main parts of the platform, mainly its software, its use and its development. This third component includes the GitHub site, the central repository for any code or data related to the SSP project, a website that serves as an outreach platform and also as a base for hosting the various network development and management tools and user guides.

3.2.2 Website

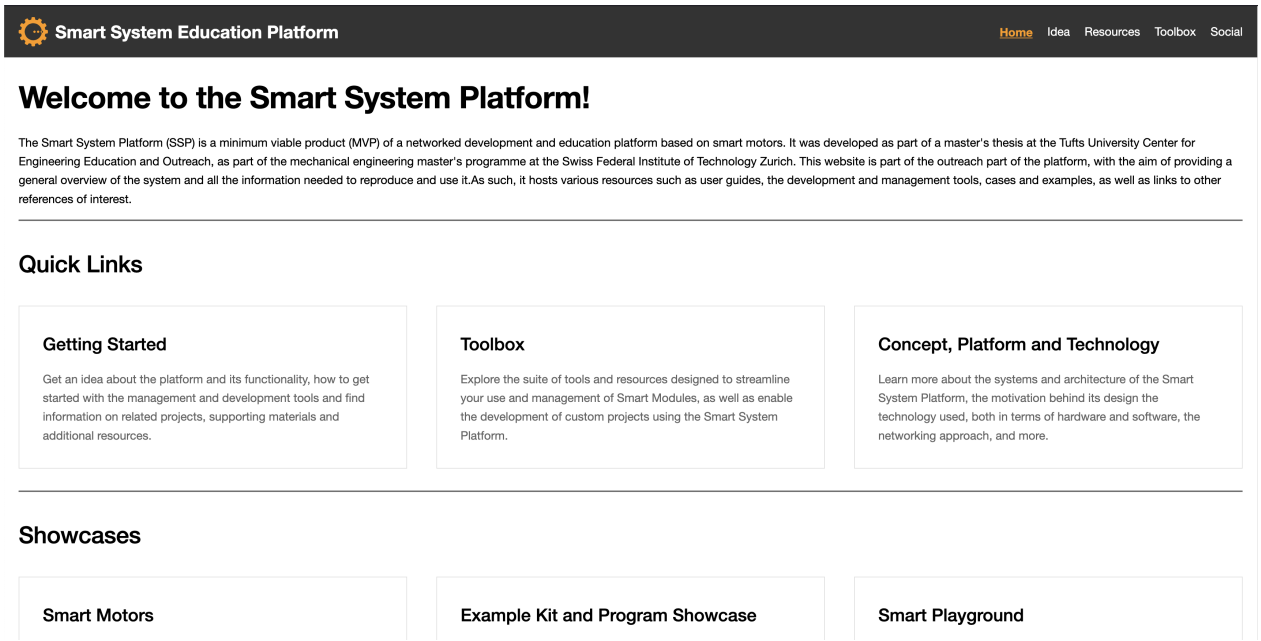


Figure 3.8. Smart System Platform website landing page

The website consists of various pages, including an introduction page, information about the project and involved parties, as well as a user guide. It further hosts links to the various tools and provides background information on the concept of the Smart System Platform. The website's design aligns with the Swiss design style outlined in Section 2.2.3, as referenced by Hollis (2006) and Müller-Brockmann (2020) in their work on the Swiss design style. The primary objective of the website's design is to ensure accessibility and the facilitate the effective dissemination of information. The website was tested using the Google for Developers PageSpeed Insights ("PageSpeed Insights", n.d.), which resulted in an overall accessibility and usability score of 100 out of 100 for both mobile and desktop version of the webpage.

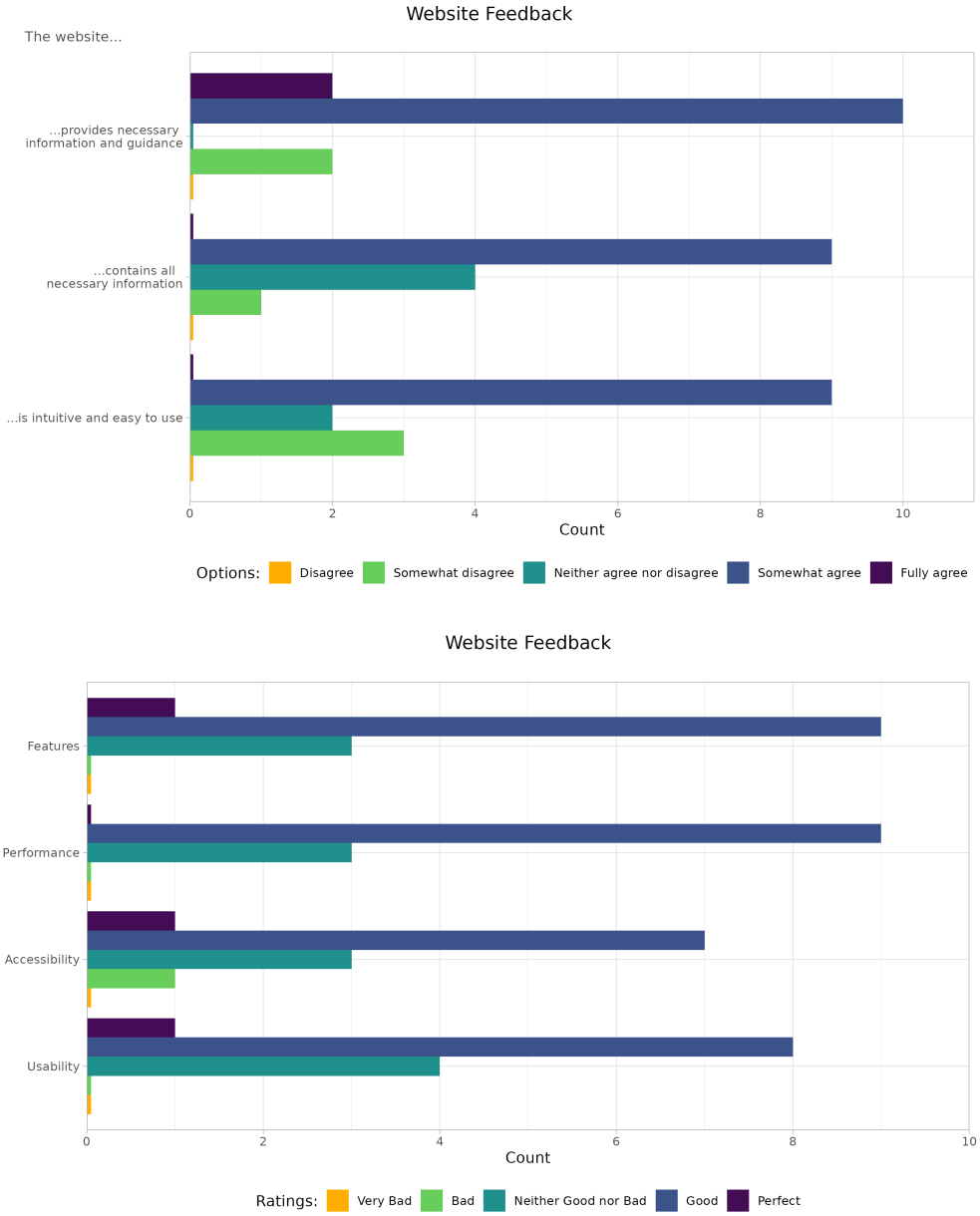


Figure 3.9. Hackathon 2: Website ratings

The ratings given to the website by the participants of Hackathon 2, shown in Figure 3.9, are predominantly positive, particularly with regard to content and features. Nevertheless, the ratings and the especially the individual comments suggest there are potential areas for enhancement, especially with regard to accessibility, the intuitiveness and the design of the website, as some technical difficulties and navigation issues were encountered by some users.

3.2.3 Development and Management Tools

One of the website's and the Smart System Platform's main components, are the various networking development and management tools. The purpose of these tools is to facilitate the utilisation and advancement of the core networking functionality. The feedback gathered after the second Hackathon is shown in Figure 3.10. The following section will thus examine and discuss these tools.

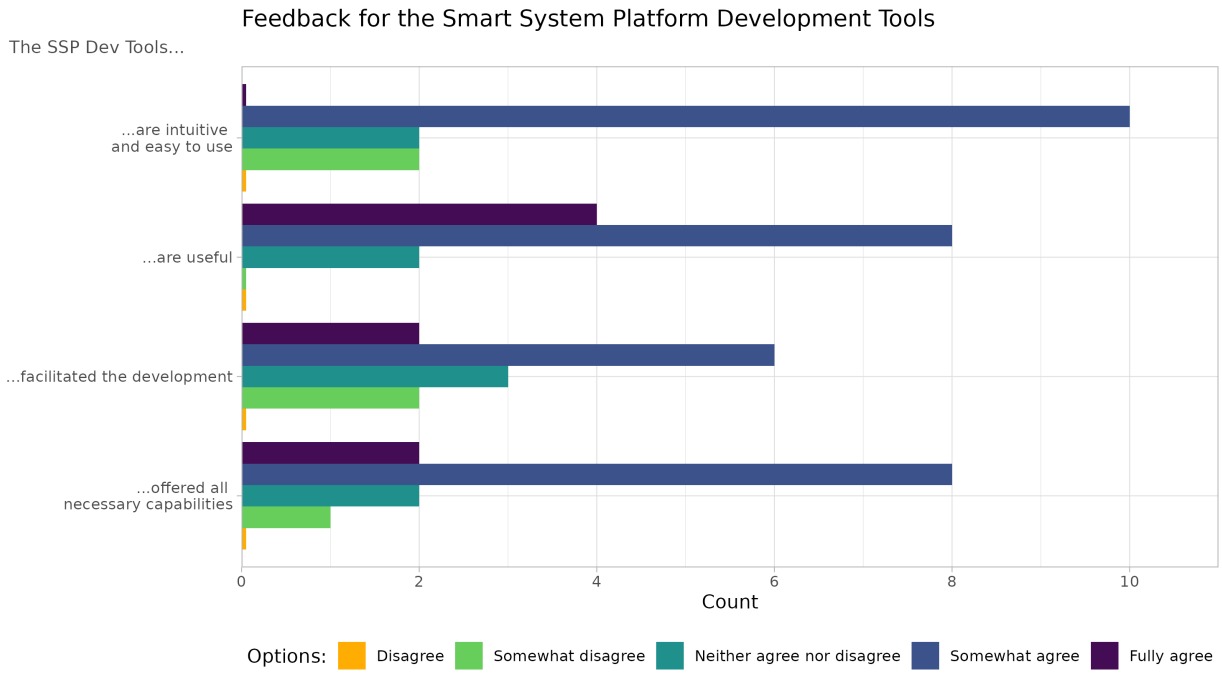


Figure 3.10. Hackathon 2: Smart System Platform Development Tools Rating

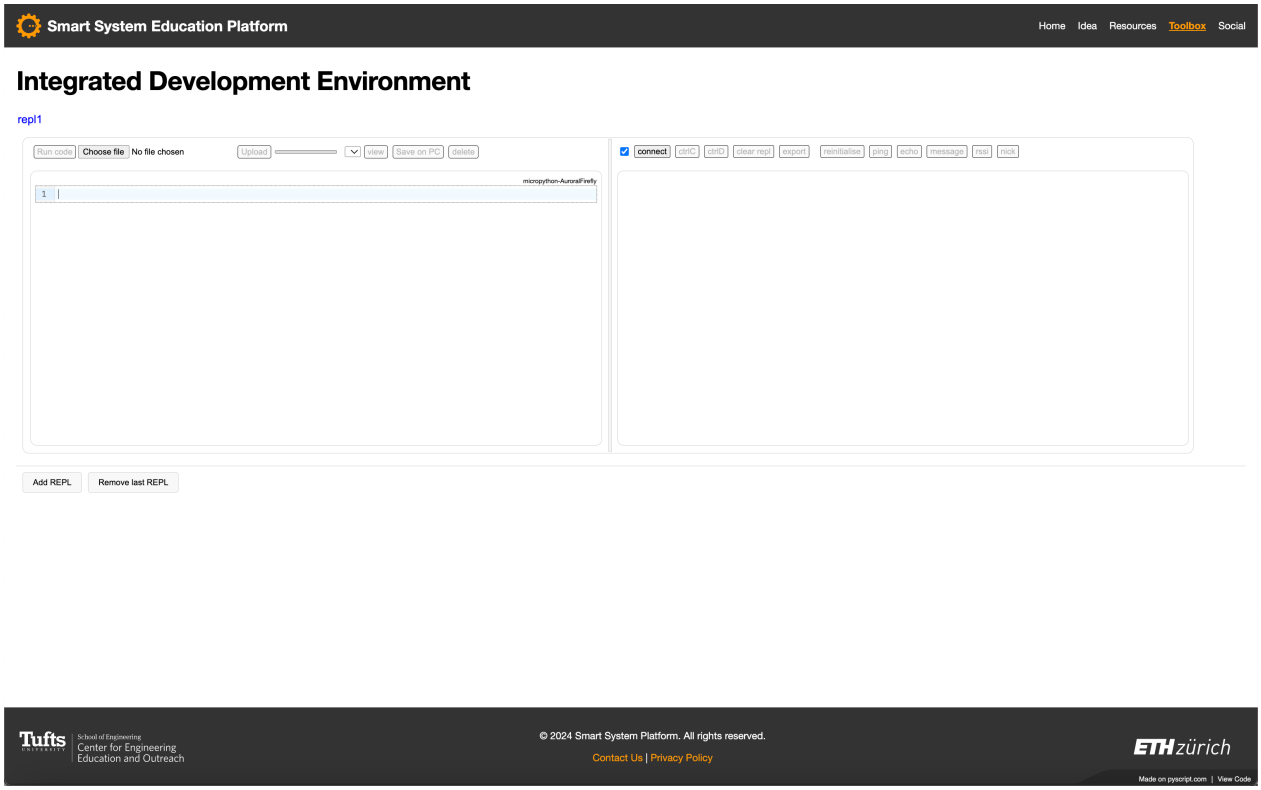


Figure 3.11. Custom Web IDE

The custom Web-IDE was specifically developed for use with the Smart System Platform and the developed Networking Library, as described in Section 2.2.3, and features a variety of custom features. The most notable feature being the multiple development tabs that can be added to the page. These tabs permit simultaneous development and REPL control of multiple devices, which is advantageous for networking development. Furthermore, the IDE facilitates the initiation of networking and the transmission of networking commands via pre-coded buttons within the REPL area. User feedback on the IDE has been largely positive, with suggestions for improvement relating to accessibility, intuitiveness and design, as well as performance, based on specific user feedback.

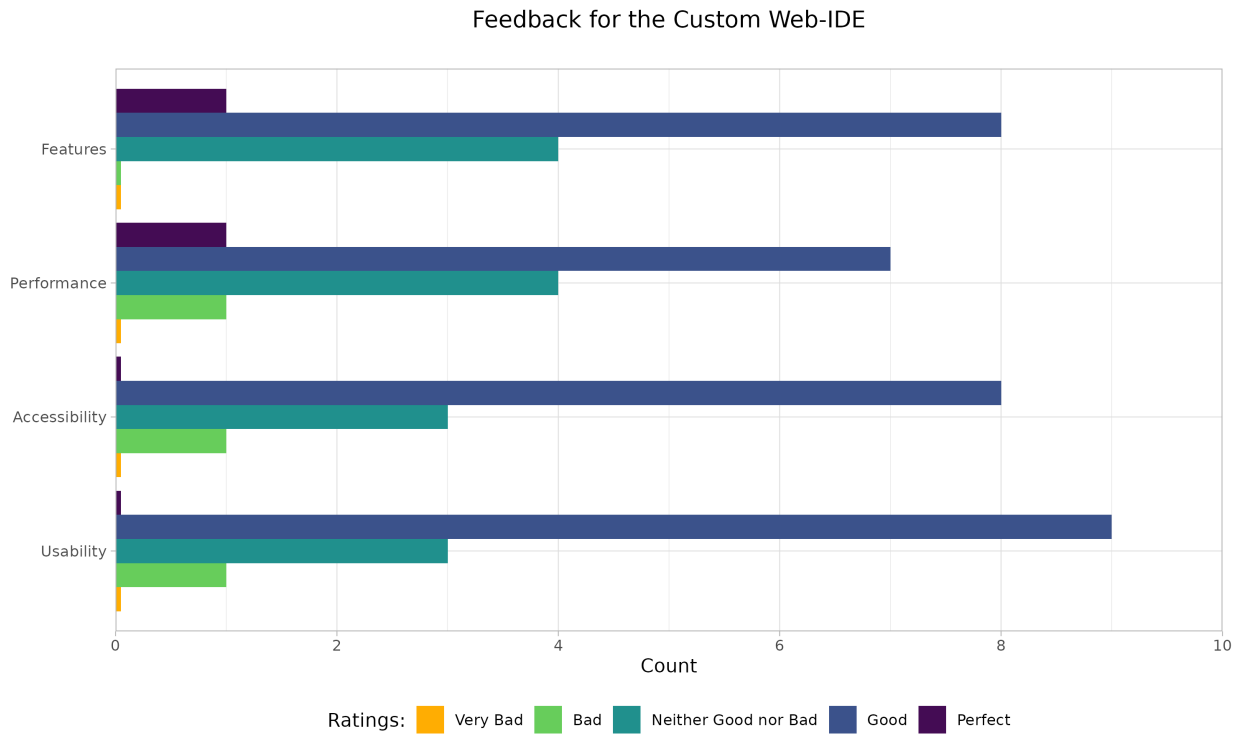


Figure 3.12. Hackathon 2: Custom Web-IDE ratings

AI Code Assistant

In an effort to facilitate development with the Networking Library and the Smart System Platform more accessible, a LLM was primed with the code library, example code and the provided guides, as outlined in Section 2.2.3. However, the primed LLM was not directly tested and no structured feedback was gathered from users using it. Consequently, the presented feedback is of an anecdotal nature. While the LLM has been primed with knowledge of the Smart System Platform, all of its libraries, sample code and other files, including the Networking Library for networking, and mostly returns correct structure, nomenclature and commands, students reported mistakes being made in the example syntax and structure for certain commands provided by the assistant, which significantly limits the usefulness as users likely to use the AI Code Assistant are not familiar with the code and as such may not notice the incorrect code and the support provided to them.

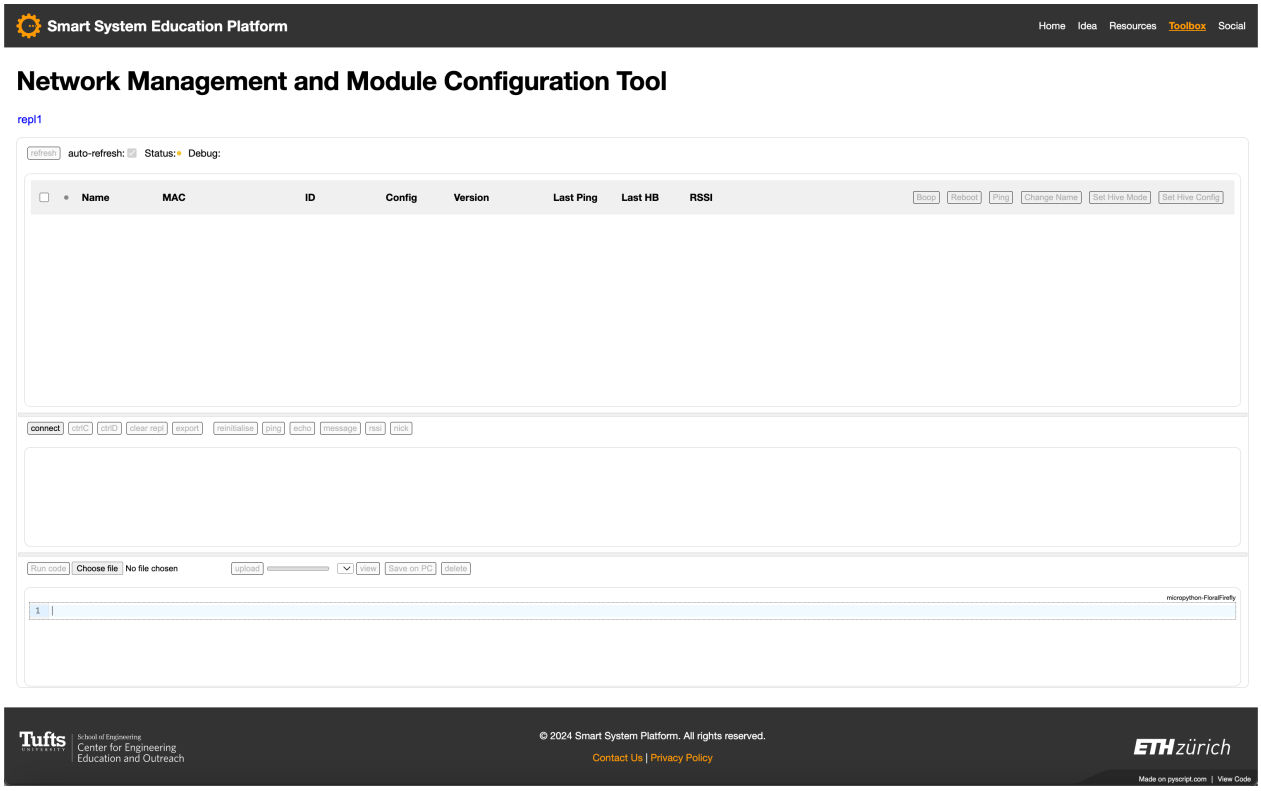


Figure 3.13. Network Management and Module Configuration Tool

The Network Management and Module Configuration Tool has been developed to allow and facilitate management of the networking operations and associated modules using the Networking Library. The tool hence has a number of features, which include the ability to find any Smart Module with an initialised network class, retrieve information from the module such as name, configuration and more, and to send commands to the module directly from the web page. The web page has also been designed to work with the hive mode program written for some sample modules, which allows modules to be configured to send their data to specific MAC addresses, and to use data sent from specific other modules in a specific way. Further details and more in-depth discussion of that program and functionality can be found in Section 3.3.1.

Figure 3.14. Network Management and Module Configuration Tool: Drop-down configuration feature with input options

In order to further simplify the use of the Network Management and Module Configuration Tool in connection with the *hive mode* program, drop-down based configuration had been developed, which is shown in Figure 3.14. Instead of the arduous task of manually inputting the variables into the command prompt, the configuration command can be sent with the press of a button, using the values set via the drop-down menus, which are populated with the information of all the other operational modules found in the vicinity. This feature is a result of feedback gathered during the hackathons and was subsequently developed and not tested in the hackathons.

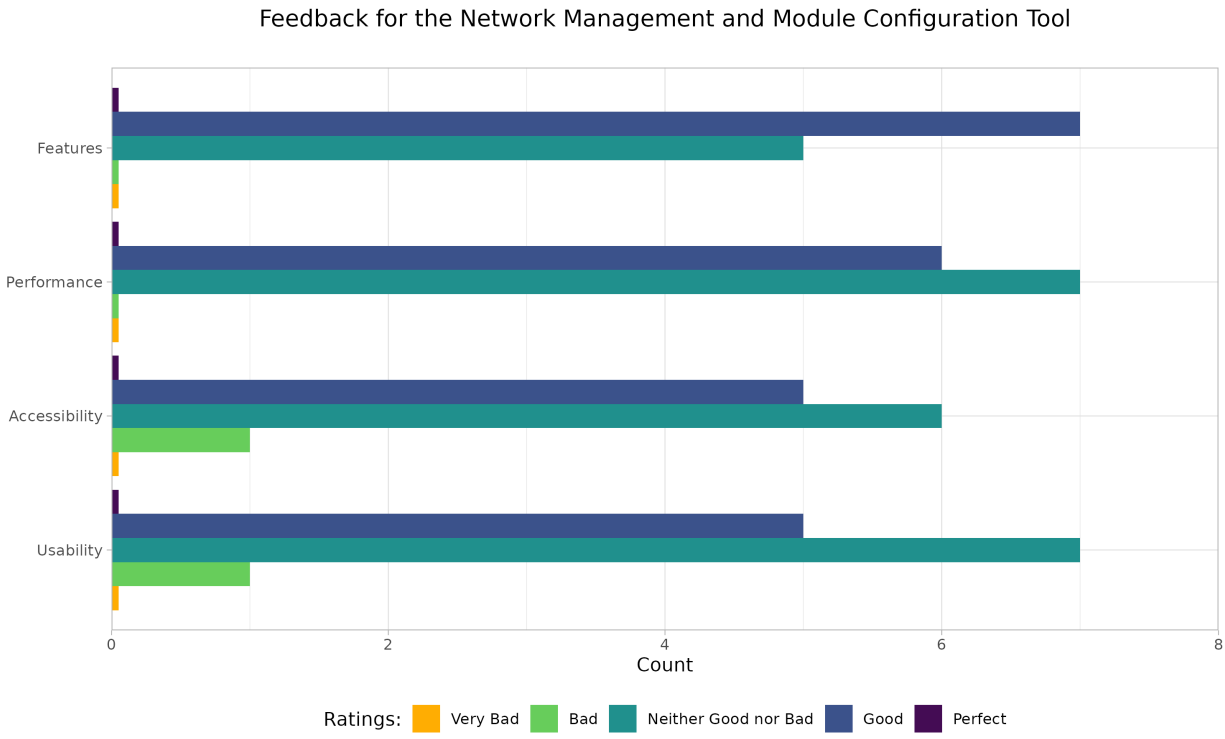


Figure 3.15. Hackathon 2: Network Management and Module Configuration Tool ratings

This webpage incorporates all the necessary technical capabilities and could serve as the back-end for a graphical user interface (GUI). A notable limitation of the tool is that ESP-NOW-based networking can not be accessed directly from a computer or a website, necessitating the connection of a Smart Module with an initialised network library to the computer via USB to serve as a gateway.

Based on the evaluation gathered in Hackathon 2, shown in Figure 3.15, the ratings for this tool are mediocre in all areas, with only the features receiving a slightly positive rating. Despite the existence of a user guide provided on the website, it appears that the tool was too complex and not intuitive enough to be considered accessible. This assertion is supported by the participants' comments. Although the tool had all the necessary features and capabilities in terms of functionality and technology, the complexity of the GUI and the tool itself acted as a barrier to its usability. Users therefore found it difficult to make constructive use of the tool.

AI Module Configuration Assistant

As outlined in Section 2.2.3, an additional LLM was primed to assist in the hive configuration to make modules interact. However, while the tool was useful in principle, when it worked and provided accurate information, similar to the problems encountered with the AI Code Assistant, this LLM also exhibited issues with the accuracy of the output it provided. Specifically, the LLM was expected to provide the necessary configuration values for the configuration fields, however, it frequently parsed the provided data incorrectly. For example, it confused the MAC address of the MCB with the chip ID, returning the latter instead of the MAC address for the required values for the hive configuration command. In these cases the usefulness of this tool was diminished significantly.

Module Management Tool

As described in Section 2.2.3, this tool has been developed to facilitate updating the software of a Smart Module. This is achieved by checking the version number in the configuration file of the motor against the latest release version to determine if an update is required. Based on the selected configuration, the latest software files are downloaded from the *software/release* folder of the Smart System Platform GitHub and loaded onto the module, updating the *config.py* file accordingly. This process is automated, thereby minimising the need for user intervention.

3.3 Utilisation and Outcomes

3.3.1 Example Applications

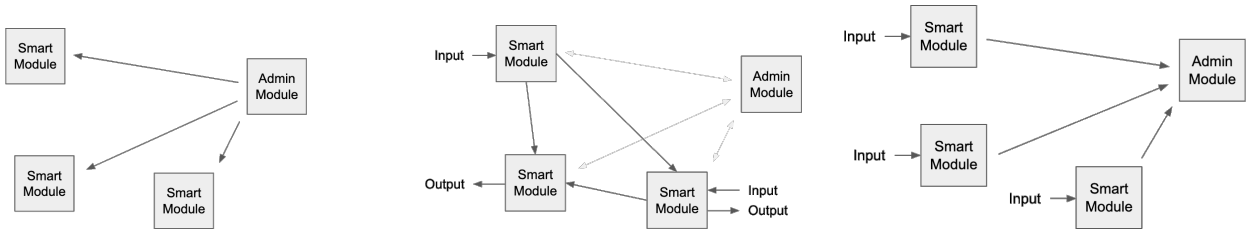
In the course of the research for this thesis, a number of example programs were developed, a selection of which is introduced and described here:

- **boop-o-meter:**
  The boop-o-meeter program was developed during the early stages of the Networking Library's development, with the objective to evaluate and demonstrate the direct peer-to-peer and peer-to-all messaging capabilities of the ESP-NOW-based networking approach. The program runs on a Dahal board, and makes use of the included user interface in the form of the screen to display information and buttons for interaction. In addition, it includes the capability to scan for nearby devices, which will sends out a *ping* command to the broadcast address. Based on the *pong* returned by other devices in the vicinity, a list of the discovered devices' MAC address, in addition to the general broadcast MAC address, is populated and displayed on the screen. By selecting one of the entries a message is sent to the selected recipient. The program tracks the number of sent and received messages, with a maximum set at 999 for each.

- **Hive Motors:**
  In order to demonstrate the networking capabilities of Smart Motors, a showcase was prepared in which two Smart Motors were hard-coded to transmit their sensor data streams to each other using the developed Networking Library. Rather than using their own sensor input to be mapped against the output for the standard Smart Motor program, they use the other Smart Motor's sensor, thereby transforming the program into a collaborative task, with each motor exerting control over the other. This configuration was adopted to demonstrate the data transmission capabilities of the networking approach.

- **Hive Mode Program:**
  In an effort to facilitate interaction of different Smart Modules without the necessity of manually hardcoding, an approach was devised to allow configuration of the module interaction using networking commands. The software has been adapted to run a *boot.py* file, which determines the configuration based on the entry in *config.py* and runs the corresponding main program. This program initialises the networking and other required libraries and then, by default, runs the main stand-alone logic of the module, if available. For example, in the case of Smart Motors, the aforementioned *boot.py* initialises the SSP Add-on Library, which in turn initialises the Networking Library, and checks if the mod-

ule is in hive mode, and if it is not, runs the main stand-alone Smart Motor program. However, with the networking initialised, the module becomes discoverable, and is able receive messages and commands, and can be managed by the Network Management and Module Configuration Tool. The second part of the name of the tools derives from the ability to send configuration commands to the respective modules, a functionality that has been added for this specific mode of operation. Upon receiving a hive configuration command, the module saves the transmitted configuration values into the configuration file. This configuration consists of the MAC addresses of the modules to which it should send its data, the MAC addresses of the module it should expect data from and which sensor data should be used to determine the output, the rate at which messages should be sent, and the mode of interaction. The module then reboots and enters hive mode, in which it operates according to its hive configuration. The concept is illustrated in the Figure 3.16.



(a) Admin module sends command with configuration to set modules into hive mode

(b) Modules in hive mode sending, receiving and processing the data as configured

(c) Alternatively, the modules can be set up to send all of their sensor data to the Admin Module

Figure 3.16. Hive mode operation illustrations

3.3.2 Hackathon 1

The first hackathon was designed to evaluate the Networking Library and gather ideas for potential applications. It also involved conducting the reliability test using the *boop-o-meter* program, the result of which are discussed in Section 3.1.7. While users demonstrated engagement with both the Networking Library and the concept of the Smart System Platform, the feedback from participants revealed significant challenges in getting started with the library, particularly for those with no prior coding experience. The absence of structured onboarding materials rendered the initial setup difficult, and the provided *example.py* file was insufficient as a starting point. Groups encountered issues with fundamental tasks, such as trying to find their MAC address. Additionally, the high volume of network messages, due to every participant broadcasting to all others on the same channel, resulted in confusion and made meaningful project development challenging. Furthermore, a majority of IDEs used by the groups, it was

only possible to connect and code one module at a time, which rendered the development of networking applications, which required coding on both the sender and the recipient side challenging when only using a single computer.

These insights underscored the need for improved onboarding support, including structured documentation, enhanced example code, and development tools to streamline accessibility and flatten the learning curve. Consequently, these results have directly led to the development of the various development and management tools with the aim of improving accessibility, which were introduced and used during Hackathon 2.

During the ideation phase, teams generated a broad array of creative concepts, ranging from rudimentary communication tools to more complex interactive experiences. Some basic ideas included sending messages, smiley faces, custom images, and poking interactions between modules. Others explored game-based applications, such as hide and seek, tag, and a silent assassin-style game, where the modules used RSSI to help players locate their targets. More advanced concepts emerged as well, such as *Boop Among Us*, a game inspired by social deduction mechanics, as well as virtual data ball throwing and catching, which utilises accelerometer data. Teams also proposed practical and educational applications, including mesh data logging for school science curricula, wireless collaborative reward/punishment-based training using networking, and a collaborative, sensor-driven navigation game. These ideas showcase the versatility of the Networking Library and its potential for playful, collaborative and educational use. Nonetheless, the majority of the proposed applications were linear in nature, with few proposals where behaviour and interaction depended on multiple interconnected Smart Modules.

3.3.3 Hackathon 2

During the second hackathon, teams were again challenged to brainstorm ideas and develop functional prototypes using the Networking Library and the SSP development tools. The primary objective of this activity was to observe what the participants would come up with, and to ascertain the efficacy of the tools in enabling the rapid design and development of networked prototypes. The focus remained on exploring the potential of the Smart System Platform to facilitate efficient development and experimentation, enabling teams to build and iterate on their projects with minimal setup time.

The teams proposed a variety of creative ideas, including a game utilising the hive mode of Smart Modules, where motors were configured to respond to button presses, kicking a ball

towards an opponent, a motor simulating a carnival ride, which could be controlled with a networked module in terms of speed, rotations, and state (on/off), as well as a motion-based red-light green-light game where a Smart Splat displayed either a red or green light, while an accelerometer sensor on another module tracked player movement, enforcing the movement or no movement phases.

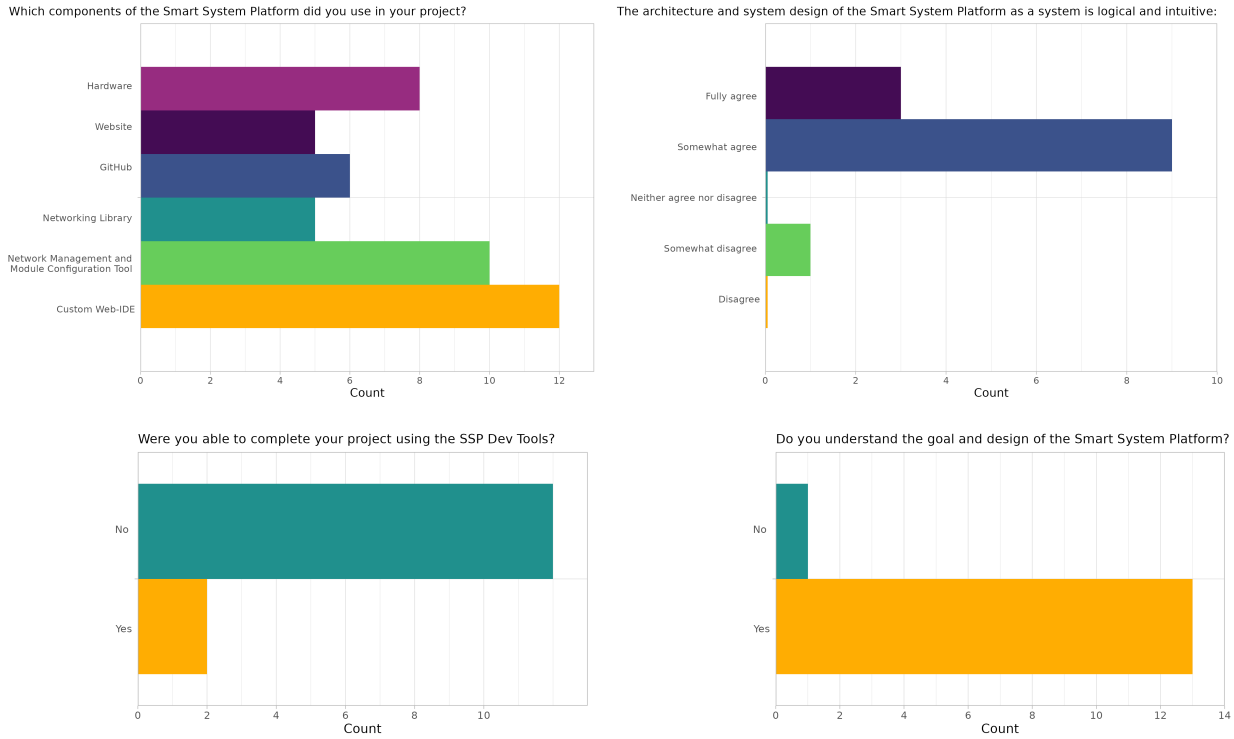


Figure 3.17. Hackathon 2: Questionnaire results

As discussed in Section 3.2.3, the tools received mostly positive feedback. However, this is contrasted by the fact that a majority of teams were unable to develop a fully functional prototype during the hackathon, as shown in Figure 3.17. The limited time available was cited as the primary challenge, which prevented participants from completing their projects, followed by the complexity and usability of the tools provided. Conversely, the participants expressed that they understood the goal and design of the Smart System Platform, with a majority concurring that the system is both logical and intuitive. The key conclusion of the hackathon was that the entry barrier for using the provided tools was too high, especially given the limited amount of time available. Participants found the tools complex and unintuitive, making it difficult to prototype efficiently within the given time-frame. These insights highlighted the need for more user-friendly onboarding materials and an even more simplified development workflows to enhance accessibility in future iterations.

3.3.4 Smart Playground

The Smart Playground project successfully integrated the Networking Library, thereby enabling communication between various interactive modules. Some of these modules were based on the Dahal Board and the ESP32C3 MCBs, while others were custom-built and based on the ESP32C6. The various modules have been introduced in Section 2.20 and are shown in Figure 2.21. Two game modes were designed:

- **Sequence Game:**
  Smart Buttons, each assigned a number and an assigned colour, are distributed across the playground area. At a central control board, one of the handheld stuffed animal modules is held in close proximity of the control board, and is assigned the coder role. The individual who is assigned this role then goes around the playground with their module, collecting a sequence. This is achieved by pressing the button on the Smart Button module, which sends out a *ping* command that detects which module is closest based on the RSSI values. By pressing the button on the handheld stuffed animal module, the sequence step is added to their module. The step is represented in the form of a coloured dot on the module's LED matrix. Once a complete sequence has been collected, the coder returns to the control board, where the sequence is sent to the control board. The sequence of colours is then displayed on the control board and up to four players are then tasked to reproduce the sequence in the correct order. This is done by retracing the steps of the coder and going to the various button modules and adding the steps to their stuffed animal modules, while the control board keeps track of their progress. A player has completed the game once they have collected all the steps of the given sequence.

- **Music Game:** The music game functions in a similar manner to the sequence game. However, in lieu of Smart Buttons distributed around locations, the Smart Splats are kept in a central location and are assigned a sound and colour. The user can then collect these sounds on their modules by stomping on the Smart Splats. Once the notes have been collected, they can then send them to the control board, which in turn will play the sequence of notes collected.

The interaction and wireless communication, as well as proximity detection based on RSSI, are both facilitated by the Smart System Platform Networking Library. In addition, all the developed hardware is compatible and in line with the concept of the Smart System Platform. The collaboration between Tufts CEEO and Boston University highlights the potential for real-world educational applications using the networking capabilities and tools of the Smart System Platform.

While the system successfully fulfilled the communication and networking requirements of that project, feedback and observations during use highlighted areas for further refinement. In particular, user feedback from various applications led to targeted tests and improvements in the Networking Library and Smart System Platform throughout their development. Notably, battery performance and RSSI angle tests were directly influenced by anecdotal reports from users working with the system.

4 Conclusions

The research in this thesis builds on the Smart Motors project (Dahal, 2024) by developing a peer-to-peer networking approach using ESP-NOW for communication between different MCs. To support this networking capability, the Smart System Platform framework was designed to improve accessibility and usability. A minimum viable product (MVP) of the platform and its components was developed to ensure that networking between modules was not only possible, but also practical and easy to implement. In addition to developing and testing the networking library, way how the platform could be used in practice were also investigated. Various components of the Smart System Platform were designed and tested, and their usability was evaluated through trials with college students. These evaluations provided valuable insights into how educators and students interact with the system, how they might use it, and what kinds of concepts they might develop.

The research findings demonstrate that the developed networking approach is both viable and functional, thus highlighting its significant potential for practical applications. The peer-to-peer communication system functions reliably in both controlled test environments and independent projects by different users. The networking framework successfully enables direct communication between Smart Modules, thus rendering it suitable for a wide range of applications, from collaborative robotics to interactive educational experiences. Despite being in its early stages, the Smart System Platform serves as a functional proof of concept, demonstrating the potential of networked educational robotics and a structured approach to implementation. While the Smart System Platform was designed to provide an accessible means for users to engage with the networking capabilities, some usability challenges were identified. The MVP successfully demonstrated the feasibility of both the networking system and the platform-based approach. However, further refinements are required to enhance the accessibility, usability and intuitiveness of the platform's supporting material. The accessibility and usability of the supporting tools and materials pose challenges, particularly due to the initial learning curve, though the core concept remains robust.

A key finding of this thesis is that networked educational robotics could be viably used, from a

technological perspective, for interactive and collaborative learning approaches. Educational use cases, such as the Smart Playground project (Blake-West *et al.*, 2025), provide an example of how the Smart System Platform and its networking capabilities can be used to support and facilitate the development of creative and engaging educational experiences. The results of the hackathons further highlight the innovative concepts and ideas that emerge when students and educators are given access to this technology. The range of concepts developed in hackathons and external projects reinforces the viability of the approach and to demonstrate the creative potential of networked modules in education and research.

Peer-to-peer networking for the Smart Modules is both feasible and valuable, particularly in the context of education and robotics. Despite remaining challenges, this technology has great potential. By refining the tools, improving accessibility and fostering a community of users and developers, the Smart System Platform could become a transformative tool for networked robotics, STEM education and interactive learning. With further development and refinement, the Smart System Platform could become a highly effective educational tool, enabling students, educators, and researchers to explore new frontiers in connected robotics and collaborative learning.

5 Future Work

The Smart System Platform, as a proof of concept, was established and introduced in this thesis, demonstrating its potential as a tool for networking and interactive system development. However, future work could extend beyond a technical prototype, further developing the networking capability and its support framework. The following areas of interest and key questions were identified for further exploration:

- **Continued development of SSP-based projects and activities**
  Of the numerous ideas and concepts proposed using the Networking Library during the two hackathons, which could be developed into a game or educational experience? Furthermore, how could a collaborative educational experience be developed, fully leveraging the established networking approach?

- **Developing educational applications**
  How can the Smart System Platform and its developed networking capabilities be used to improve usability and accessibility to educational robotics, and what novel functionalities or educational opportunities could be enabled by this enhanced connectivity? It would be interesting to develop a lesson plan centred on the Smart Motors and their networking capability, and to explore how such a networking approach could be integrated into science education.

- **Further development of software and tools**
  As outlined in Chapter 3, there are several areas for improvement, especially in the area of development and management tools. The Networking Library and the SSP Add-on Library could be optimised and rewritten in C++. Furthermore, building on the basic concept, how could the tools be designed to be intuitive and accessible? For example, a LabVIEW inspired drag-and-drop GUI could be developed for wireless configuration of modules and their interactions.
  Additionally, would it be feasible to fully integrate an LLM to enable full AI-driven configuration of the various modules, based on natural text or image input only? This could greatly simplify and streamline the configuration process and enhance the accessibility

for non-technical users.

- **Integration with other Technologies**
  How is it possible to connect a mobile phone to the designed network? Could the phone's sensors be used to provide data input to the modules? In addition, an app version of some tools could be developed to program, configure and interact with the system from a mobile device, further increasing accessibility.
  What other technologies could the system be integrated with?

- **Hardware and component support development**
  In light of the explored possibilities of sensors and outputs, the development of support libraries using the hardware could be continued. Furthermore, the development of additional modules and capabilities could be pursued and the integration or use of novel hardware explored. And in a second step, based on the modules developed, what could a networked educational robotics kit look like? What components could be further used?

- **Enlarging the user base**
  Although the MVP was principally designed to support the CEEO and educational researchers, the system could be expanded and made widely available to other stakeholders such as teachers and students. How could the platform be integrated with external platforms such as data analysis tools or educational software? In order to facilitate its accessibility and utility for teachers, what further developments would be necessary? How could educators quickly develop and deploy custom or off-the-shelf systems to support STEM education?

- **Community building & open source development**
  How can an open source community be fostered to support ongoing development? What strategies could be implemented to encourage contributions from students, teachers and developers? What tools or platforms would most effectively facilitate collaboration and sharing of knowledge?

List of Tables

List of Figures

Bibliography

Abdi, A. I., Omar, A. M., Mahdi, A. O., Asiimwe, C., & Osman, M. A. (2024). Tracing the evolution of STEM education: A bibliometric analysis. *Frontiers in Education*, *9*. https://doi.org/10.3389/feduc.2024.1457938

Afari, E., & Khine, M. S. (2017). Robotics as an Educational Tool: Impact of Lego Mindstorms. *International Journal of Information and Education Technology*, *7*(6), 437–442. https://doi.org/10.18178/ijiet.2017.7.6.908

Aivar Annamaa. (n.d.). Thonny, Python IDE for beginners. Retrieved February 9, 2025, from https://thonny.org/

Aivar Annamaa. (2025, February). Thonny GitHub [original-date: 2019-01-01T10:29:50Z]. Retrieved February 9, 2025, from https://github.com/thonny/thonny

Anaconda Inc. (n.d.-a). PyScript | Code and Share Python in the Browser. Retrieved February 9, 2025, from https://pyscript.com/

Anaconda Inc. (n.d.-b). Pyscript.net. Retrieved February 9, 2025, from https://pyscript.net/

Anaconda Inc. (2025, February). PyScript GitHub [original-date: 2022-02-21T04:55:20Z]. Retrieved February 9, 2025, from https://github.com/pyscript/pyscript

Annamaa, A. (2015a). Thonny,: A Python IDE for Learning Programming. *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, 343. https://doi.org/10.1145/2729094.2754849

Annamaa, A. (2015b). Introducing Thonny, a Python IDE for learning programming. *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, 117–121. https://doi.org/10.1145/2828959.2828969

Atkin, J. M., & Black, P. (2003, January). *Inside Science Education Reform: A History of Curricular and Policy Change* [Google-Books-ID: BCv93qPg_7wC]. Teachers College Press.

Basics | Bluetooth Technology Website. (2012, October). Retrieved February 2, 2025, from https://web.archive.org/web/20121028194747/http://www.bluetooth.com/Pages/basics.aspx

Benitti, F. B. V. (2012). Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, *58*(3), 978–988. https://doi.org/10.1016/j.compedu.2011.10.006

Bequette, J. W., & Bequette, M. B. (2012). A Place for Art and Design Education in the STEM Conversation. *Art Education*, *65*(2), 40–47. https://doi.org/10.1080/00043125.2012.11519167

Blackley, S., & Howell, J. (2015). A STEM Narrative: 15 Years in the Making. *Australian Journal of Teacher Education*, *40*(7). https://doi.org/10.14221/ajte.2015v40n7.8

Blake-West, J., Rogers, C., Cross, J., Hanking, S., Battel, K., Triebold, N., Bers, M., Dahal, M., Pimentel, I., Wolfe, L., Corsi, V., Hannes, L., Bayraktar, D., Bustamante, A., Ahn, J., Garcia, L., Seccia, I., Cervera, K., Romano, D., & Hays, V. (2025). Smart Playground.

Bluetooth Core Specification 5.0 (amended). (2024, June). Retrieved February 3, 2025, from https://www.bluetooth.com/specifications/specs/core-specification-amended-5-0/

Bluetooth Core Specification 5.3 (amended). (2024, June). Retrieved February 3, 2025, from https://www.bluetooth.com/specifications/specs/core-specification-amended-5-3/

Bluetooth Low Energy | Bluetooth Technology Website. (2017, March). Retrieved February 3, 2025, from https://web.archive.org/web/20170310111443/https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy

Bluetooth Mesh Model. (2023, September). Retrieved February 3, 2025, from https://www.bluetooth.com/specifications/specs/mesh-model-1-1/

Bluetooth Mesh Protocol. (2023). Retrieved February 3, 2025, from https://www.bluetooth.com/specifications/specs/mesh-protocol/

Bluetooth Technology Website. (n.d.). Retrieved February 2, 2025, from https://www.bluetooth.com/

Brennan, R., Tonkal, M., & Rogers, C. B. (2023). Implementing Systems Engineering with Elementary School Students. Retrieved January 27, 2025, from https://peer.asee.org/implementing-systems-engineering-with-elementary-school-students

Bybee, R. W. (2013). *The Case for STEM Education: Challenges and Opportunities* [Google-Books-ID: gfn4AAAAQBAJ]. NSTA Press.

Certification Process Overview. (2020, November). https://www.wi-fi.org/system/files/Wi-Fi_Alliance_Certification_Process_Overview_v3.6.pdf

Connor, A. M., Karmokar, S., & Whittington, C. (2015). From STEM to STEAM: Strategies for Enhancing Engineering & Technology Education. *International Journal of Engineering Pedagogy (iJEP)*, *5*(2), 37–47. https://doi.org/10.3991/ijep.v5i2.4458

Dahal, M. (2024). *Designing and Evaluating Smart Motors: Trainable Educational Robotics System for STEM Classrooms* [Ph.D.]. Tufts University [ISBN: 9798384091523]. Retrieved January 19, 2025, from https://www.proquest.com/docview/3104983984/abstract/686BC60B0DCB4999PQ/1

Dahal, M., & Cross, J. (2025, January). Smart Motors GitHub [original-date: 2022-11-09T19:21:34Z]. Retrieved February 9, 2025, from https://github.com/tuftsceeo/SmartMotors

Dahal, M., Kresin, L., Peres, A., Bento Pereira, E., & Rogers, C. (2023). International Collaboration to Increase Access to Educational Robotics for Students [ISSN: 2377-634X]. *2023 IEEE Frontiers in Education Conference (FIE)*, 1–5. https://doi.org/10.1109/FIE58773.2023.10343494

DeVille, K. (2024, February). STEM vs. STEAM - What's the difference? [Section: News & Articles]. Retrieved January 21, 2025, from https://stemeducationguide.com/stem-v-steam/

Dolgopolovas, V., & Dagienė, V. (2021). Computational thinking: Enhancing STEAM and engineering education, from theory to practice. *Computer Applications in Engineering Education*, *29*(1), 5–11. https://doi.org/10.1002/cae.22382

Duckworth, E. (1964). Piaget rediscovered. *The Arithmetic Teacher*, *11*(7), 496–499. Retrieved January 22, 2025, from https://www.jstor.org/stable/41186862

Eridani, D., Rochim, A. F., & Cesara, F. N. (2021). Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy

Usage and Barrier Resistance. *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)*, 322–328. https://doi.org/10.1109/iSemantic52711.2021.9573246

Espressif Systems. (n.d.-a). ESP SoCs. Retrieved February 2, 2025, from https://www.espressif.com/en/products/socs

Espressif Systems. (n.d.-b). ESP-BLE-MESH Documentaiton [ESP32-C3 - ESP-IDF Programming Guide v5.4 documentation]. Retrieved February 3, 2025, from https://docs.espressif.com/projects/esp-idf/en/v5.4/esp32c3/api-guides/esp-ble-mesh/ble-mesh-index.html

Espressif Systems. (n.d.-c). ESP-NOW Documentation [ESP32-C3 - ESP-IDF Programming Guide v5.4 documentation]. Retrieved January 19, 2025, from https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html

Espressif Systems. (n.d.-d). ESP-WIFI-MESH Documentation [ESP32-C3 - ESP-IDF Programming Guide v5.4 documentation]. Retrieved January 19, 2025, from https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html

Espressif Systems. (2024a). ESP32-C3 Series Datasheet. Retrieved January 19, 2025, from https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf

Espressif Systems. (2024b). ESP32-C6 Datasheet. Retrieved January 23, 2025, from https://www.espressif.com/sites/default/files/documentation/esp32-c6_datasheet_en.pdf

Espressif Systems. (2025). ESP32 Datasheet. Retrieved January 19, 2025, from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Hankin, S., Cross, J., Triebold, N., & Art, N. (n.d.). Smart Playground GitHub. Retrieved February 9, 2025, from https://github.com/shanki01/Playground-Code

Hollis, R. (2006). *Swiss Graphic Design: The Origins and Growth of an International Style, 1920-1965*. Laurence King Publishing.

IEEE Standard 802.11-2024 [IEEE Approved Draft Standard for Information Technology – Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks – Specific Requirements - Part 11: Wireless Local Area Network (LAN) Medium Access Control (MAC) and Physical Layer (PHY) Specifications]. (2024). *IEEE P802.11-REVme/D7.0, August 2024*, 1–6213. Retrieved January 28, 2025, from https://ieeexplore.ieee.org/document/10638490

IEEE Standard 802.11ax-2021 [IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN]. (2021). *IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020)*, 1–767. https://doi.org/10.1109/IEEESTD.2021.9442429

IEEE Standard 802.11b-1999 [IEEE Standard for Information Technology - Telecommunications and information exchange between systems - Local and Metropolitan networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher Speed Physical Layer (PHY) Extension in the 2.4 GHz band]. (2000). *IEEE Std 802.11b-1999*, 1–96. https://doi.org/10.1109/IEEESTD.2000.90914

IEEE Standard 802.11g-2003 [IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Con-

trol (MAC) and Physical Layer (PHY) Specifications: Further Higher Data Rate Extension in the 2.4 GHz Band]. (2003). *IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11, 1999 Edn. (Reaff 2003) as amended by IEEE Stds 802.11a-1999, 802.11b-1999, 802.11b-1999/Cor 1-2001, and 802.11d-2001)*, 1–104. https://doi.org/10.1109/IEEESTD.2003.94282

IEEE Standard 802.15.1-2002 [Conference Name: IEEE Std 802.15.1-2002]. (2002). *IEEE Std 802.15.1-2002*, 1–473. https://doi.org/10.1109/IEEESTD.2002.93621

IEEE Standard 802.15.1-2005 [IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN)]. (2005). *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, 1–700. https://doi.org/10.1109/IEEESTD.2005.96290

IEEE Standard 802.15.4-2024 [IEEE Standard for Low-Rate Wireless Networks]. (2024). *IEEE Std 802.15.4-2024 (Revision of IEEE Std 802.15.4-2020)*, 1–967. https://doi.org/10.1109/IEEESTD.2024.10794632

IEEE Standard 802.15.5-2009 [Conference Name: IEEE Std 802.15.5-2009]. (2009). *IEEE Std 802.15.5-2009*, 1–166. https://doi.org/10.1109/IEEESTD.2009.4922106

ISTE. (n.d.). ISTE Standards & U.N. Sustainable Development Goals. Retrieved January 20, 2025, from https://iste.org/iste-standards-and-unesco

Jamali, S. M., Ale Ebrahim, N., & Jamali, F. (2023). The role of STEM Education in improving the quality of education: A bibliometric study. *International Journal of Technology and Design Education*, *33*(3), 819–840. https://doi.org/10.1007/s10798-022-09762-1

Johnson, C. C. (2012). Implementation of STEM Education Policy: Challenges, Progress, and Lessons Learned. *School Science and Mathematics*, *112*(1), 45–55. https://doi.org/10.1111/j.1949-8594.2011.00110.x

Khine, M. S. (Ed.). (2017). *Robotics in STEM Education*. Springer International Publishing. https://doi.org/10.1007/978-3-319-57786-9

Kokotsaki, D., Menzies, V., & Wiggins, A. (2016). Project-based learning: A review of the literature [Publisher: SAGE Publications Ltd]. *Improving Schools*, *19*(3), 267–277. https://doi.org/10.1177/1365480216659733

LEGO Education. (n.d.). STEM & STEAM Solutions for the Classroom. Retrieved January 20, 2025, from https://education.lego.com/en-us/

LEGO Education Science. (n.d.). Retrieved January 21, 2025, from https://education.lego.com/en-us/lego-education-science/

Lehmann, S., & Scheller, A. (n.d.). MicroPython SSD1306 Driver GitHub. Retrieved February 9, 2025, from https://github.com/stlehmann/micropython-ssd1306/tree/master

Marín-Marín, J.-A., Moreno-Guerrero, A.-J., Dúo-Terrón, P., & López-Belmonte, J. (2021). STEAM in education: A bibliometric analysis of performance and co-words in Web of Science. *International Journal of STEM Education*, *8*(1), 41. https://doi.org/10.1186/s40594-021-00296-x

Markula, A., & Aksela, M. (2022). The key characteristics of project-based learning: How teachers implement projects in K-12 science education. *Disciplinary and Interdisciplinary Science Education Research*, *4*(1), 2. https://doi.org/10.1186/s43031-021-00042-x

MicroPython. (2025a, February). MicroPython ESP-NOW Documentation. Retrieved January 19, 2025, from https://docs.micropython.org/en/latest/library/espnow.html

MicroPython. (2025b, February). MicroPython Quick Reference for the ESP32. Retrieved February 9, 2025, from https://docs.micropython.org/en/latest/esp32/quickref.html#general-board-control
MicroPython. (2025c, February). MicroPython time & time related functions Documentation. Retrieved February 6, 2025, from https://docs.micropython.org/en/latest/library/time.html
Mindell, D., Beland, C., Chan, W., Clarke, D., Park, R., & Trupiano, M. (2000). LEGO Mindstorms. *The Structure of an Engineering (R) evolution}*.
Müller-Brockmann, J. (2020). *Grid systems in graphic design: A visual communication manual for graphic designers, typographers and three dimensional designers* (16th edition). Niggli.
NanaWangDFR, bluewings, mengbishu, black5heep, & gboldwang. (n.d.). MicroPython ADXL345 Library GitHub. Retrieved February 9, 2025, from https://github.com/DFRobot/micropython-dflib/tree/master
National Research Council, Division of Behavioral and Social Sciences and Education, Board on Science Education, & National Committee on Science Education Standards and Assessment. (1995, December). *National Science Education Standards* [Google-Books-ID: s2pXgQ5tMFQC]. National Academies Press.
National Research Foundation & National Science Board. (2022). Science & Engineering Indicators. https://www.nsf.gov/nsb/sei/one-pagers/K-12-Indicator-2022.pdf
Next Generation Science Standards. (2013, April). Retrieved January 19, 2025, from https://www.nextgenscience.org/
PageSpeed Insights. (n.d.). Retrieved February 4, 2025, from https://developers.google.com/speed/docs/insights/v5/about
Papert, S., & Harel, I. (Eds.). (1991). *Constructionism* [Pages: xi, 518]. Ablex Publishing.
Parker, R., Thomsen, B. S., & Berry, A. (2022). Learning Through Play at School – A Framework for Policy and Practice [Publisher: Frontiers]. *Frontiers in Education*, *7*. https://doi.org/10.3389/feduc.2022.751801
Project-Based Learning. (2014). In R. K. Sawyer, Krajcik, Joseph S., & Shin, Namsoo (Eds.), *The Cambridge Handbook of the Learning Sciences* (2nd ed., pp. 275–297). Cambridge University Press. https://doi.org/10.1017/CBO9781139519526
Rogers, C. (n.d.). Serial Test. Retrieved February 9, 2025, from https://pyscript.com/@chrisrogers/nick-esp-now/latest?files=README.md
Rossum, G. v., Warsaw, B., & Coghlan, A. (2001, July). Python Enhancement Proposals 8 – Style Guide for Python Code. Retrieved February 6, 2025, from https://peps.python.org/pep-0008/
Sagan, C. (1995, January). *The Demon-Haunted World: Science as a Candle in the Dark* [Google-Books-ID: Yz8Y6KfXf9UC]. Random House Publishing Group.
Samsudın, M. A., Jamalı, S. M., Zaın, A. N. M., & Ebrahım, N. A. (2020). The effect of STEM project based learning on self-efficacy among high-school physics students. *Journal of Turkish Science Education*, *17*(1), 94–108. Retrieved January 22, 2025, from https://tused.org/index.php/tused/article/view/876
Sapounidis, T., & Alimisis, D. (2020). Educational Robotics for Stem: A Review of Technologies and Some Educational Considerations.
Satterthwait, D. (2010). Why are 'hands-on' science activities so effective for student learning? [Publisher: Australian Science Teachers Association]. *Teaching Science: The Journal of the Australian Science Teachers Association*, *56*(2), 7–10. Retrieved January 23, 2025, from

https://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=52539907&site=ehost-live
Seeed Studio. (2023, March). Grove Ecosystem Introduction. Retrieved March 5, 2025, from https://wiki.seeedstudio.com/Grove_System/
Seeed Studio. (2024a, May). Seeed Studio XIAO ESP32C6 Zigbee Documentation. Retrieved February 2, 2025, from https://wiki.seeedstudio.com/xiao_esp32c6_zigbee/
Seeed Studio. (2024b, August). Seeed Studio XIAO ESP32C3 Documentation. Retrieved January 19, 2025, from https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/
Seeed Studio. (2024c, August). Seeed Studio XIAO ESP32C6 Documentation. Retrieved February 2, 2025, from https://wiki.seeedstudio.com/xiao_esp32c6_getting_started/
Šesták, J. (2022a). *Dynamic Mesh Network Implemented in MicroPython on top of ESP-NOW Protocol* [Master's thesis, Brno University of Technology].
Šesták, J. (2022b). Dynamic Mesh network in Micropython on ESP32.
Technology in education: GEM Report 2023 | Global Education Monitoring Report. (2023, July). Retrieved January 20, 2025, from https://www.unesco.org/gem-report/en/technology
Tonkal, M., Wu, A., & Rogers, C. (2024). Exploring the Impact of Systems Engineering Projects on STEM Engagement and Learning. *2024 IEEE Frontiers in Education Conference (FIE)*.
Triebold, N., Art, N., Hankin, S., & Dahal, M. (2025, February). Smart System Platform GitHub [original-date: 2024-10-04T23:40:06Z]. Retrieved February 9, 2025, from https://github.com/tuftsceeo/Smart-System-Platform
United Nations Department of Economic and Social Affairs. (n.d.-a). The 17 Goals | Sustainable Development. Retrieved January 19, 2025, from https://sdgs.un.org/goals
United Nations Department of Economic and Social Affairs. (n.d.-b). Goal 4 Quality Education | Sustainable Development. Retrieved January 19, 2025, from https://sdgs.un.org/goals/goal4
United Nations Department of Economics and Social Affairs. (n.d.). Sustainable Development Goals. Retrieved January 23, 2025, from https://unosd.un.org/content/sustainable-development-goals-sdgs
Unruly Studios. (n.d.). Unruly Splats. Retrieved March 5, 2025, from https://www.unrulysplats.com/
van Leeuwen, E., & Franzyshen, S. (2019, June). PainlessMesh Technical Documentation. Retrieved January 19, 2025, from https://gitlab.com/painlessMesh/painlessMesh/-/wikis/home
Vesilind, E. M., & Jones, M. G. (1996). Hands-On: Science Education Reform [Publisher: SAGE Publications Inc]. *Journal of Teacher Education*, *47*(5), 375–385. https://doi.org/10.1177/0022487196047005007
Von Glasersfeld, E. (1982). An Interpretation of Piaget's Constructivism [Publisher: Revue Internationale de Philosophie]. *Revue Internationale de Philosophie*, *36*(142/143 (4)), 612–635. Retrieved January 23, 2025, from https://www.jstor.org/stable/23945415
WebReflection. (n.d.). Micro-repl. Retrieved February 9, 2025, from https://www.jsdelivr.com/package/npm/micro-repl
Weisberg, D. S., Hirsh-Pasek, K., & Golinkoff, R. M. (2013). Guided Play: Where Curricular Goals Meet a Playful Pedagogy. *Mind, Brain, and Education*, *7*(2), 104–112. https://doi.org/10.1111/mbe.12015
Wi-Fi Alliance. (n.d.). Retrieved February 2, 2025, from https://www.wi-fi.org/

Wi-Fi Alliance. (2023). Generational Wi-Fi User Guide. https://www.wi-fi.org/system/files/Generational_Wi-Fi_User_Guide_202304.pdf

Xie, Y., Fang, M., & Shauman, K. (2015). STEM Education. *Annual Review of Sociology*, *41* (Volume 41, 2015), 331–357. https://doi.org/10.1146/annurev-soc-071312-145659

Yakman, G., & Lee, H. (2012). Exploring the Exemplary STEAM Education in the U.S. as a Practical Educational Framework for Korea [Publisher: The Korean Association for Science Education]. *Journal of The Korean Association For Science Education*, *32*(6), 1072–1086. https://doi.org/10.14697/jkase.2012.32.6.1072

Zimmermann, H. (1980). OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection [Conference Name: IEEE Transactions on Communications]. *IEEE Transactions on Communications*, *28*(4), 425–432. https://doi.org/10.1109/TCOM.1980.1094702

Zosh, J. M., Hopkins, E. J., Jensen, H., Liu, C., Neale, D., Pasek, K. H., Solis, S. L., & Whitebread, D. (2017). *Learning through play: A review of the evidence*. Retrieved January 23, 2025, from https://www.ucviden.dk/en/publications/learning-through-play-a-review-of-the-evidence

Appendix A - Links and References

In the interest of open data, a copy of the submitted thesis and most relevant files, including a non-curated collection of all code, data, images and videos, as well as various other additional data, can be accessed using the following link:

http://gofile.me/3EjjZ/EdmlpGXHm

The following GitHub repository hosts all the data, information and references of the of the Smart System Platform and can be access using the following link:

https://github.com/tuftsceeo/Smart-System-Platform

The Smart System Platform webpage, which hosts general information about the Smart System Platform, its idea, the goal, description, guides, how to get started, the development suites, examples and references, can be accessed using the following link:

https://tuftsceeo.github.io/Smart-System-Platform

The Smart System Platform development pages (such as the IDE, networking management etc.), which are hosted by pyscript and embedded in the aforementioned webpage, can be accessed directly using the following link:

https://pyscript.com/@nickart/collections/153973b3-25be-4ffa-a525-c3e9cd134d4a

APPENDIX B - TOOL USE DECLARATION

Tool Name	**Description**	**Use**
Arduino IDE	Arduino BASIC IDE	Arduino BASIC coding
DeepL	Language AI platform	Translation, writing and proof-reading
GitHub	Cloud-based development, hosting and Git-enabled version control platform	Project and website hosting, as well as code and version management
JetBrains PyCharm PE	Python IDE	Python and MicroPython coding
JetBrains Rider	General code IDE	HTML, CSS and R coding
Microsoft PowerPoint	Presentation program	Presentations and figure design
Microsoft Word	Word processing program	Writing and title page design
Overleaf	Web-based LaTeX Editor	Report writing
OpenAI ChatGPT	Large Language Model (LLM)	Coding support
Perplexity AI	Consolidated LLM and AI assistant	Coding support
Pixelmator	Graphic Editor	Graphic design for figures and report
positCloud	Web-based RStudio IDE	Coding of R Markdown and Quatro for GitHub automation, data analysis and figures
PyScript	Web-based development platform for web-applications using Python	Coding of Smart System development tools (Module- and Network Management, IDE)
Thonny	Python IDE	Python and MicroPython coding
Qualitrics	Survey Research Company	Hackathon survey

Appendix C - Module Code and Libraries

boot.py:

```
from config import config
import time

print("Running boot.py")

try:
    time.sleep(3)
except Exception as e:
    print(f"Error {e}")

module_name = config["configuration"].lower()
try:
    with open(module_name + ".py") as f:
        code = f.read()
    exec(code)
except Exception as e:
    print(f"Error running {module_name}: {e}")
```

config.py:

```
config = {"name": "Nickname", "configuration": None, "id": None, "version": None, "sta_channel": 1, "ap_channel": 1}
version = {'adxl345.py': 3,
            'splat.py': 1,
'sp1.py': 3,
'hm3.py': 3,
            'am1.py': 1,
'ssp_networking.py': 6,
            'files.py': 2,
            'icons.py': 2,
            'prefs.py': 2,
            'sensors.py': 4,
            'servo.py': 2,
            'ssd1306.py': 2,
            'sm3.py': 1,
            'sl1.py': 1,
            'smartlight.py': 1,
'networking.py': 9,
            'boot.py': 0
            }
mysecrets = {"SSID": "Tufts_Robot", "key": ""}
msg_codes = {"cmd": 0x01, "inf": 0x02, "ack": 0x03}
msg_subcodes = {
    "cmd": {
        "Reboot": 0x00,
        "Firmware_Update": 0x01,
        "File_Update": 0x02,
        "File_Download": 0x03,
        "File_Run": 0x05,
        "Set_Admin": 0x06,
```

```
        "Whitelist-Add": 0x07,
        "Config-Change": 0x08,
        "Name-Change": 0x09,
        "Ping": 0x10,
        "Pair": 0x11,
        "Set-Pair": 0x12,
        "RSSI/Status/Config-Boop": 0x13,
        "Directory-Get": 0x14,
        "Echo": 0x15,
        "Resend": 0x16,
        "Hive-Set": 0x17,
        "Hive-Configure": 0x18,
        "Web-Repl": 0x20,
        "WiFi-Connect": 0x21,
        "WiFi-Disconnect": 0x22,
        "AP-Enable": 0x23,
        "AP-Disable": 0x24,
        "Pause": 0x25,
        "Resume": 0x26,
    },
    "inf": {
        "RSSI/Status/Config-Boop": 0x20,
        "Data": 0x21,
        "Message": 0x22,
        "Directory": 0x23,
    },
    "ack": {
        "Pong": 0x10,
        "Success": 0x11,
        "Fail": 0x12,
        "Confirm": 0x13,
        "Echo": 0x15,
    },
}
networking_keys = {
    "default_handshake_key": "handshake",
    "handshake_key1": "handshake1",
    "handshake_key2": "handshake2",
    "handshake_key3": "handshake3",
    "handshake_key4": "handshake4",
    "default_ap_key": "password",
    "default_wifi_key": "password"
}
whitelist = [b'd\xe83\x84\xd8\x18', b'd\xe83\x84\xd8\x19',
             b'd\xe83\x85\xd3\xbc', b'd\xe83\x85\xd3\xbd',
             b'd\xe83\x84\xd8\x18', b'd\xe83\x84\xd8\x19']  #each ESP32 has two MAC addresses
i2c_dict = {
    "0x3C": ["pca9685", 0, "screen"],
    "0x53": ["ACCEL", 1, "accelerometer"]
}  #key is i2c address: ["device name", Output (0) or Input (1), "Description"]
sensor_dict = {"sensor": [0,4095], "potentiometer": [0,180], "select": [0,1], "up": [0,1], "down": [0,1], "button": [0,1], "sw1": [0,1],
       "sw2": [0,1],  "sw3": [0,1],  "sw4": [0,1]}
hive_config = {"hive": False, "refreshrate": 0, "recipients": [], "sender_sensor_list": [], "mode": None}
```

networking.py:

```
import network
import time
import ubinascii
import espnow
import gc
import struct
import json
import machine


class Networking:
    def __init__(self, infmsg=False, dbgmsg=False, errmsg=False, admin=False, inittime=0):
        gc.collect()
        self.inittime = inittime
        if infmsg:
            print(f"{(time.ticks_ms() - self.inittime) / 1000:.3f} Initialising Networking")
```

```
        self.master = self
        self.infmsg = infmsg
        self.dbgmsg = dbgmsg
        self.errmsg = errmsg
        self.admin = admin
        self.config = {"Name": None,
                       "Configuration": None,
                       "id": None,
                       "Version": None,
        } #Just as an example, can include more information here, but for interoperability purposes please keep the basic four
              attributes

        self._staif = network.WLAN(network.STA_IF)
        self._apif = network.WLAN(network.AP_IF)

        self.sta = self.Sta(self, self._staif)
        self.ap = self.Ap(self, self._apif)
        self.aen = self.Aen(self)

        if infmsg:
            print(f"{(time.ticks_ms() - self.inittime) / 1000:.3f} Networking initialised and ready")

    def cleanup(self):
        self.dprint(".cleanup")
        self.aen.cleanup()
        self._staif.active(False)
        self._apif.active(False)

    def log(self, text):
        if self.admin:
            with open("log.txt", "a") as f:
                f.write(f"{text}\n")

    def iprint(self, message):
        if self.infmsg:
            try:
                text = f"{(time.ticks_ms() - self.inittime) / 1000:.3f} Networking Info: {message}"
                print(text)
                self.log(text)
            except Exception as e:
                print(f"Error printing Networking Info: {e}")
        return

    def dprint(self, message):
        if self.dbgmsg:
            try:
                text = f"{(time.ticks_ms() - self.inittime) / 1000:.3f} Networking Debug: {message}"
                print(text)
                self.log(text)
            except Exception as e:
                print(f"Error printing Networking Debug: {e}")
        return

    def eprint(self, message):
        if self.errmsg:
            try:
                text = f"{(time.ticks_ms() - self.inittime) / 1000:.3f} Networking Error: {message}"
                print(text)
                self.log(text)
            except Exception as e:
                print(f"Error printing Networking Error: {e}")
        return

    class Sta:
        def __init__(self, master, _staif):
            self.master = master
            self._sta = _staif
            self._sta.active(True)
            self.master.iprint("STA initialised and ready")

        def scan(self):
            self.master.dprint("sta.scan")
            scan_result = self._sta.scan()
            if self.master.infmsg:
                for ap in scan_result:
                    self.master.iprint(f"SSID:%s BSSID:%s Channel:%d Strength:%d RSSI:%d Auth:%d " % ap)
```

```
        return scan_result

    def connect(self, ssid, key="", timeout=10):
        self.master.dprint("sta.connect")
        self._sta.connect(ssid, key)
        stime = time.time()
        while time.time() - stime < timeout:
            if self._sta.ifconfig()[0] != '0.0.0.0':
                self.master.iprint("Connected to WiFi")
                return
            time.sleep(0.1)
        self.master.iprint(f"Failed to connect to WiFi: {self._sta.status()}")

    def disconnect(self):
        self.master.dprint("sta.disconnect")
        self._sta.disconnect()

    def ip(self):
        self.master.dprint("sta.ip")
        return self._sta.ifconfig()

    def mac(self):
        self.master.dprint("sta.mac")
        return bytes(self._sta.config('mac'))

    def mac_decoded(self):  # Necessary?
        self.master.dprint("sta.mac_decoded")
        return ubinascii.hexlify(self._sta.config('mac'), ':').decode()

    def channel(self):
        self.master.dprint("sta.channel")
        return self._sta.config('channel')

    def set_channel(self, number):
        self.master.dprint("sta.set_channel")
        if number > 14 or number < 0:
            number = 0
        self._sta.config(channel=number)
        self.master.iprint(f"STA channel set to {number}")

class Ap:
    def __init__(self, master, _apif):
        self.master = master
        self._ap = _apif
        self._ap.active(True)
        self.master.iprint("AP initialised and ready")

    def set_ap(self, name="", password="", max_clients=10):
        self.master.dprint("ap.set_ap")
        if name == "":
            name = self.master.config["name"]
        self._ap.active(True)
        self._ap.config(essid=name)
        if password:
            self._ap.config(authmode=network.AUTH_WPA_WPA2_PSK, password=password)
        self._ap.config(max_clients=max_clients)
        self.master.iprint(f"Access Point {name} set with max clients {max_clients}")

    def deactivate(self):
        self.master.dprint("ap.deactivate")
        self._ap.active(False)
        self.master.iprint("Access Point deactivated")

    def ip(self):
        self.master.dprint("ap.ip")
        return self._ap.ifconfig()

    def mac(self):
        self.master.dprint("ap.mac")
        return bytes(self._ap.config('mac'))

    def mac_decoded(self):
        self.master.dprint("ap.mac_decoded")
        return ubinascii.hexlify(self._ap.config('mac'), ':').decode()

    def channel(self):
```

```
        self.master.dprint("ap.channel")
        return self._ap.config('channel')

    def set_channel(self, number):
        self.master.dprint("ap.set_channel")
        if number > 14 or number < 0:
            number = 0
        self._ap.config(channel=number)
        self.master.iprint(f"AP channel set to {number}")

class Aen:
    def __init__(self, master):
        self.master = master
        self._aen = espnow.ESPNow()
        self._aen.active(True)

        self._peers = {}
        self._received_messages = []
        self._received_messages_size = []
        self._long_buffer = {}
        self._long_buffer_size = {}
        self.received_sensor_data = {}
        self.received_rssi_data = {}
        self._irq_function = None
        self.boop_irq = None
        self.data_irq = None
        self.msg_irq = None
        self.ack_irq = None
        self.custom_cmd = None
        self.custom_inf = None
        self.custom_ack = None
        self.boops = 0
        self.ifidx = 0  # 0 sends via sta, 1 via ap
        # self.channel = 0

        self._aen.irq(self._irq)

        self.master.iprint("ESP-NOW initialised and ready")

    def cleanup(self):
        self.master.iprint("aen.cleanup")
        self.irq(None)
        self._aen.active(False)
        # add delete buffers and stuff

    def update_peer(self, peer_mac, peer_config=None, channel=None, ifidx=None):
        self.master.dprint("aen.update_peer")
        if peer_mac == b'\xff\xff\xff\xff\xff\xff':
            return
        if peer_mac in self._peers:
            try:
                if peer_config is not None:
                    self._peers[peer_mac].update(peer_config)
                if channel is not None:
                    self._peers[peer_mac].update({'channel': channel})
                if ifidx is not None:
                    self._peers[peer_mac].update({'ifidx': ifidx})
                self.master.dprint(f"Peer {peer_mac} updated to channel {channel}, ifidx {ifidx} and name {self.peer_name(peer_mac)
                    }")
            except OSError as e:
                self.master.eprint(f"Error updating peer {peer_mac}: {e}")
            return
        self.master.iprint(f"Peer {peer_mac} not found")

    def add_peer(self, peer_mac, peer_config=None, channel=None, ifidx=None):
        self.master.dprint("aen.add_peer")
        if peer_mac == b'\xff\xff\xff\xff\xff\xff':
            return
        if peer_mac not in self._peers:
            try:
                self._peers[peer_mac] = {}
                if channel is not None:
                    self._peers[peer_mac].update({'channel': channel})
                if ifidx is not None:
                    self._peers[peer_mac].update({'ifidx': ifidx})
                if peer_config is not None:
```

```
                self._peers[peer_mac].update(peer_config)
            self._peers[peer_mac].update({'rssi': None, 'time': None, 'last_ping': 0})
            self.master.dprint(f"Peer {peer_mac} added with channel {channel}, ifidx {ifidx} and name {self.peer_name(peer_mac)
                }")
        except OSError as e:
            self.master.eprint(f"Error adding peer {peer_mac}: {e}")
    else:
        self.master.dprint(f"Peer {peer_mac} already exists, updating")
        self.update_peer(peer_mac, peer_config, channel, ifidx)

def remove_peer(self, peer_mac):
    self.master.dprint("aen.remove_peers")
    if peer_mac in self._peers:
        try:
            del self._peers[peer_mac]
            self.master.iprint(f"Peer {peer_mac} removed")
        except OSError as e:
            self.master.eprint(f"Error removing peer {peer_mac}: {e}")

def peers(self):
    self.master.dprint("aen.peers")
    rssi_table = self._aen.peers_table
    for key in self._peers:
        self._peers[key].update({'rssi': rssi_table[key][0]})
        self._peers[key].update({'time': rssi_table[key][1]-self.master.inittime})
    return self._peers

def peer_name(self, key):
    self.master.dprint("aen.name")
    if isinstance(key, list):
        return "..."
    if key in self._peers:
        if 'name' in self._peers[key]:
            return self._peers[key]['name']
        else:
            return None
    else:
        return None

def rssi(self):
    self.master.dprint("aen.rssi")
    return self._aen.peers_table

# Send cmds
def send_custom(self, msg_code, msg_subcode, mac, payload=None, channel=None, ifidx=None):
    self.master.dprint("aen.send_custom")
    self._compose(mac, payload, msg_code, msg_subcode, channel, ifidx)
    gc.collect()

def ping(self, mac, channel=None, ifidx=None): #Ping
    self.master.dprint("aen.ping")
    if bool(self.ifidx):
        send_channel = self.master.ap.channel()
    else:
        send_channel = self.master.sta.channel()
    self.send_custom(0x01, 0x10, mac, [send_channel, self.ifidx, self.master.config], channel, ifidx)  # sends channel, ifidx
        and name
    if isinstance(mac, list):
        for key in mac:
            self._peers[key].update({'last_ping': time.ticks_ms()})
    elif mac == b'\xff\xff\xff\xff\xff\xff':
        for key in self._peers:
            self._peers[key].update({'last_ping': time.ticks_ms()})
    else:
        self._peers[mac].update({'last_ping': time.ticks_ms()})

def boop(self, mac, channel=None, ifidx=None): #"RSSI/Status/Config-Boop"
    self.master.dprint("aen.boop")
    self.send_custom(0x01, 0x15, mac, None, channel, ifidx)

def echo(self, mac, message, channel=None, ifidx=None):
    self.master.dprint("aen.echo")
    if len(str(message)) > 241:
        try:
            self.master.iprint(f"Sending echo ({message}) to {mac} ({self.peer_name(mac)})")
        except Exception as e:
```

```
            self.master.eprint(f"Sending echo to {mac}, but error printing message content: {e}")
    else:
        self.master.iprint(f"Sending echo ({message}) to {mac} ({self.peer_name(mac)})")
    self.send_custom(0x01, 0x15, mac, message, channel, ifidx)
    gc.collect()

def send_message(self, mac, message, channel=None, ifidx=None):
    self.send(mac, message, channel, ifidx)

def send(self, mac, message, channel=None, ifidx=None):
    self.master.dprint("aen.send_message")
    if len(str(message)) > 241:
        try:
            self.master.iprint(
                f"Sending message ({str(message)[:50] + '... (truncated)'}) to {mac} ({self.peer_name(mac)})")
        except Exception as e:
            self.master.eprint(f"Sending message to {mac} ({self.peer_name(mac)}), but error printing message content: {e}")
            gc.collect()
    else:
        self.master.iprint(f"Sending message ({message}) to {mac} ({self.peer_name(mac)})")
    self._compose(mac, message, 0x02, 0x22, channel, ifidx)
    gc.collect()

def send_data(self, mac, message, channel=None,ifidx=None):  # message is a dict, key is the sensor type and the value is the
      sensor value, fix name!!!!!!!
    self.master.dprint("aen.message")
    try:
        self.master.iprint(f"Sending sensor data ({message}) to {mac} ({self.peer_name(mac)})")
    except Exception as e:
        self.master.eprint(f"Sending sensor data to {mac} ({self.peer_name(mac)}), but error printing message content: {e}")
    self._compose(mac, message, 0x02, 0x21, channel, ifidx)

def check_messages(self):
    self.master.dprint("aen.check_message")
    return len(self._received_messages) > 0

def return_message(self):
    self.master.dprint("aen.return_message")
    if self.check_messages():
        self._received_messages_size.pop()
        return self._received_messages.pop()
    return None, None, None

def return_messages(self):
    self.master.dprint("aen.return_messages")
    if self.check_messages():
        messages = self._received_messages[:]
        self._received_messages.clear()
        self._received_messages_size.clear()
        gc.collect()
        return messages
    return [(None, None, None)]

def return_data(self):
    self.master.dprint("aen.return_data")
    return self.received_sensor_data

def _irq(self, espnow):
    self.master.dprint("aen._irq")
    if self.master.admin:
        try:
            self._receive()
            if self._irq_function and self.check_messages():
                self._irq_function()
            gc.collect()
            return
        except KeyboardInterrupt:
            self.master.eprint("aen._irq except KeyboardInterrupt")
            self.master.cleanup() #network cleanup
            raise SystemExit("Stopping networking execution. ctrl-c or ctrl-d again to stop main code")  # in thonny stops
                  library code but main code keeps running, same in terminal
    else:
        self._receive()
        if self._irq_function and self.check_messages():
            self._irq_function()
        gc.collect()
```

```
        return

    def irq(self, func):
        self.master.dprint("aen.irq")
        self._irq_function = func

    def cmd(self, func):
        self.master.dprint("aen.cmd")
        self.custom_cmd = func

    def inf(self, func):
        self.master.dprint("aen.inf")
        self.custom_inf = func

    def ack(self, func):
        self.master.dprint("aen.ack")
        self.custom_ack = func

    def _send(self, peers_mac, messages, channel, ifidx, long_msg=False):
        self.master.dprint("aen._send")
        if isinstance(peers_mac, bytes):
            peers_mac = [peers_mac]
        for peer_mac in peers_mac:
            try:
                if channel is None:
                    if peer_mac in self._peers:
                        if 'channel' in self._peers[peer_mac]:
                            channel=self._peers[peer_mac]['channel']
                    else:
                        channel = 0
                elif ifidx is None:
                    if peer_mac in self._peers:
                        if 'ifidx' in self._peers[peer_mac]:
                            ifidx=self._peers[peer_mac]['ifidx']
                    else:
                        ifidx=self.ifidx
                self._aen.add_peer(peer_mac, channel=channel, ifidx=ifidx)
                self.master.dprint(f"Added {peer_mac} to espnow buffer with channel {channel} and ifidx {ifidx}")
            except Exception as e:
                self.master.eprint(f"Error adding {peer_mac} to espnow buffer: {e}")
            for m in range(len(messages)):
                i = 0
                while i in range(3):
                    i += i
                    ack = False
                    try:
                        ack = self._aen.send(peer_mac, messages[m], True)
                        self.master.dprint(f"Receipt confirm: {ack}")
                        if ack:
                            break
                    except Exception as e:
                        self.master.eprint(f"Error sending to {peer_mac}: {e}")
                self.master.dprint(f"Sent {messages[m]} to {peer_mac} ({self.peer_name(peer_mac)}) with {ack}")
                gc.collect()
            try:
                self._aen.del_peer(peer_mac)
                self.master.dprint(f"Removed {peer_mac} from espnow buffer")
            except Exception as e:
                self.master.eprint(f"Error removing {peer_mac} from espnow buffer: {e}")

    def __send_confirmation(self, msg_type, recipient_mac, msg_subkey_type, payload=None, error=None):
        if msg_type == "Success":
            self._compose(recipient_mac, [msg_subkey_type, payload], 0x03, 0x11)
        elif msg_type == "Fail":
            self._compose(recipient_mac, [msg_subkey_type, error, payload], 0x03, 0x12)
        else:
            self._compose(recipient_mac, [msg_subkey_type, payload], 0x03, 0x13)

    def _compose(self, peer_mac, payload=None, msg_type=0x02, subtype=0x22, channel=None, ifidx=None):
        self.master.dprint("aen._compose")

        if isinstance(peer_mac, list):
            for peer_macs in peer_mac:
                if peer_macs not in self._peers:
                    self.add_peer(peer_macs, None, channel, ifidx)
        elif peer_mac not in self._peers:
```

```
            self.add_peer(peer_mac, None, channel, ifidx)

        payload_type, payload_bytes = None, None
        self.master.dprint("aen.__encode_payload")
        if payload is None:  # No payload type
            payload_type, payload_bytes = b'\x00', b''
        elif isinstance(payload, bytearray):  # bytearray
            payload_type, payload_bytes = b'\x01', bytes(payload)
        elif isinstance(payload, bytes):  # bytes
            payload_type, payload_bytes = b'\x01', payload
        elif isinstance(payload, bool):  # bool
            payload_type, payload_bytes = b'\x02', (b'\x01' if payload else b'\x00')
        elif isinstance(payload, int):  # int
            payload_type, payload_bytes = b'\x03', str(payload).encode('utf-8')
        elif isinstance(payload, float):  # float
            payload_type, payload_bytes = b'\x04', str(payload).encode('utf-8')
        elif isinstance(payload, str):  # string
            payload_type, payload_bytes = b'\x05', payload.encode('utf-8')
        elif isinstance(payload, dict) or isinstance(payload, list):  # json dict or list
            json_payload = json.dumps(payload)
            payload_type, payload_bytes = b'\x06', json_payload.encode('utf-8')
        else:
            raise ValueError("Unsupported payload type")

        messages = []
        identifier = 0x2a
        timestamp = time.ticks_ms()
        header = bytearray(8)
        header[0] = identifier
        header[1] = msg_type
        header[2] = subtype
        header[3:7] = timestamp.to_bytes(4, 'big')
        long_msg = False
        if len(payload_bytes) < 242:  # 250-9=241=max_length
            header[7] = payload_type[0]
            total_length = 1 + 1 + 1 + 4 + 1 + len(payload_bytes) + 1
            message = bytearray(total_length)
            message[:8] = header
            message[8:-1] = payload_bytes
            message[-1:] = (sum(message) % 256).to_bytes(1, 'big')  # Checksum
            self.master.dprint(f"Message {1}/{1}; Length: {len(message)}; Free memory: {gc.mem_free()}")
            messages.append(message)
        else:
            self.master.dprint("Long message: Splitting!")
            max_size = 238  # 241-3
            total_chunk_number = (-(-len(payload_bytes) // max_size))  # Round up division
            lba = b'\x07'
            header[7] = lba[0]  # Long byte array
            if total_chunk_number > 256:
                raise ValueError("More than 256 chunks, unsupported")
            for chunk_index in range(total_chunk_number):
                message = bytearray(9 + 3 + min(max_size, len(payload_bytes) - chunk_index * max_size))
                message[:8] = header
                message[8:10] = chunk_index.to_bytes(1, 'big') + total_chunk_number.to_bytes(1, 'big')
                message[10] = payload_type[0]
                message[11:-1] = payload_bytes[chunk_index * max_size: (chunk_index + 1) * max_size]
                message[-1:] = (sum(message) % 256).to_bytes(1, 'big')  # Checksum
                self.master.dprint(message)
                messages.append(bytes(message))
                self.master.dprint(
                    f"Message {chunk_index + 1}/{total_chunk_number}; length: {len(message)}; Free memory: {gc.mem_free()}")
                gc.collect()
                long_msg = True
        gc.collect()
        self._send(peer_mac, messages, channel, ifidx, long_msg)


    def __remove_long_message_from_buffer(self, timer, key):
        self.master.dprint("aen.__remove_long_message_from_buffer")
        if key in self._long_buffer:
            del self._long_buffer[key]
        if key in self._long_buffer_size:
            self._long_buffer_size[key]
        gc.collect()

    def _receive(self):
```

```
self.master.dprint("aen._receive")



def __process_message(self, sender_mac, message, receive_timestamp):
    self.master.dprint("aen.__process_message")
    if message[0] != 0x2a:  # Unique Message Identifier Check
        self.master.dprint("Invalid message: Message ID Fail")
        return None
    if len(message) < 9:  # Min size
        self.master.dprint("Invalid message: too short")
        return None

    msg_type = int.from_bytes(message[1:2], 'big')
    subtype = subtype = int.from_bytes(message[2:3], 'big')
    send_timestamp = int.from_bytes(message[3:7], 'big')
    payload_type = bytes(message[7:8])
    payload_bytes = message[8:-1]
    checksum = message[-1]
    self.master.dprint(f"{type(msg_type)}: {msg_type}, {type(subtype)}: {subtype}, {type(send_timestamp)}: {send_timestamp},
          {type(payload_type)}: {payload_type},  {type(payload_bytes)}: {payload_bytes},  {type(checksum)}: {checksum}")

    # Checksum
    if checksum != sum(message[:-1]) % 256:
        self.master.dprint("Invalid message: checksum mismatch")
        return None

    if sender_mac not in self._peers:
        self.add_peer(sender_mac)

    payload = None

    if payload_type == b'\x07':
        self.master.dprint("Long message received, processing...")
        part_n = int.from_bytes(payload_bytes[0:1], 'big')
        total_n = int.from_bytes(payload_bytes[1:2], 'big')
        payload_type = bytes(payload_bytes[2:3])

        # Create a key as a bytearray: (msg_type, subtype, timestamp, payload_type, total_n)
        key = bytearray()
        key.extend(message[1:2])
        key.extend(message[2:3])
        key.extend(message[3:7])
        key.extend(payload_type)
        key.extend(payload_bytes[1:2])
        key = bytes(key)
        self.master.dprint(f"Key: {key}")

        payload_bytes = payload_bytes[3:]

        # Check if the key already exists in the long buffer
        if key in self._long_buffer:
            # If the part is None, add the payload
            if self._long_buffer[key][part_n] is None:
                self._long_buffer[key][part_n] = payload_bytes
                self._long_buffer_size[key] = self._long_buffer_size[key] + len(payload_bytes)
                self.master.dprint(
                    f"Long message: Key found, message added to entry in long_message_buffer, {sum(1 for item in self.
                          _long_buffer[key] if item is not None)} out of {total_n} packages received")
                # If there are still missing parts, return
                if any(value is None for value in self._long_buffer[key]):
                    gc.collect()
                    return
            else:
                return
        else:
            # Initialize the long message buffer for this key
            payloads = [None] * total_n
            payloads[part_n] = payload_bytes
            self._long_buffer[key] = payloads
            self._long_buffer_size[key] = len(payload_bytes)
            self.master.dprint(
                f"Long message: Key not found and new entry created in long_message_buffer, {sum(1 for item in self.
                      _long_buffer[key] if item is not None)} out of {total_n} packages received")

            while len(self._long_buffer) > 8 or sum(self._long_buffer_size.values()) > 75000:
```

```
                self.master.dprint(
                    f"Maximum buffer size reached: {len(self._long_buffer)}, {sum(self._long_buffer_size.values())} bytes;
                        Reducing!")
                oldest_key = next(iter(self._long_buffer))
                del self._long_buffer[oldest_key]
                del self._long_buffer_size[oldest_key]
            gc.collect()
            timer = machine.Timer(0)
            timer.init(period=10000, mode=machine.Timer.ONE_SHOT, callback=lambda t: self.__remove_long_message_from_buffer(
                t, key))
            return

        # If all parts have been received, reconstruct the message
        if not any(value is None for value in self._long_buffer[key]):
            payload_bytes = bytearray()
            for i in range(0, total_n):
                payload_bytes.extend(self._long_buffer[key][i])
            del self._long_buffer[key]
            del self._long_buffer_size[key]
            self.master.dprint("Long message: All packages received!")
        else:
            self.master.dprint("Long Message: Safeguard triggered, code should not have gotten here")
            gc.collect()
            return

    payload_size = len(payload_bytes)

    self.master.dprint("aen.__decode_payload")
    if payload_type == b'\x00':  # None
        payload = None
    elif payload_type == b'\x01':  # bytearray or bytes
        payload = bytes(payload_bytes)
    elif payload_type == b'\x02':  # bool
        payload = payload_bytes[0:1] == b'\x01'
    elif payload_type == b'\x03':  # int
        payload = int(payload_bytes.decode('utf-8'))
    elif payload_type == b'\x04':  # float
        payload = float(payload_bytes.decode('utf-8'))
    elif payload_type == b'\x05':  # string
        payload = payload_bytes.decode('utf-8')
    elif payload_type == b'\x06':  # json dict or list
        payload = json.loads(payload_bytes.decode('utf-8')) #use eval instead? #FIX
    elif payload_type == b'\x07':  # Long byte array
        payload = bytes(payload_bytes)
    else:
        raise ValueError(f"Unsupported payload type: {payload_type} Message: {payload_bytes}")

    # Handle the message based on type
    if msg_type == 0x01:  # Command Message
        msg_key = "cmd"
        __handle_cmd(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key)
    elif msg_type == 0x02:  # Informational Message
        msg_key = "inf"
        __handle_inf(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key)
    elif msg_type == 0x03:  # Acknowledgement Message
        msg_key = "ack"
        __handle_ack(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key)
    else:
        self.master.iprint(f"Unknown message type from {sender_mac} ({self.peer_name(sender_mac)}): {message}")

def __handle_cmd(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key):
    self.master.dprint(f"aen.__handle_cmd")
    if (msg_subkey := "Ping") and subtype == 0x01 or subtype == 0x10:  # Ping
        self.master.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.peer_name(sender_mac)})")
        self.add_peer(sender_mac, payload[2], payload[0], payload[1])
        if bool(self.ifidx):
            channel = self.master.ap.channel()
        else:
            channel = self.master.sta.channel()
        response = [channel, self.ifidx, self.master.config, send_timestamp]
        self._compose(sender_mac, response, 0x03, 0x10)
    elif (msg_subkey := "RSSI/Status/Config-Boop") and subtype == 0x03 or subtype == 0x13:  # RSSI/Status/Config Boop
        self.boops = self.boops + 1
        self.master.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.peer_name(sender_mac)}),
            Received total of {self.boops} boops!")
        try:
```

```
            self._compose(sender_mac, [self.master.config, self.master.version, self.master.sta.mac, self.master.ap.mac,
                  self.rssi()], 0x02, 0x20)  # [ID, Name, Config, Version, sta mac, ap mac, rssi]
        except Exception as e:
            self.master.aen.__send_confirmation(self, "Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    elif (msg_subkey := "Echo") and subtype == subtype == 0x02 or subtype == 0x15:  # Echo
        self.master.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.peer_name(sender_mac)}): {
              payload}")  # Check i or d
        self._compose(sender_mac, payload, 0x03, 0x15)
    else:
        if self.custom_cmd:
            self.master.iprint(f"Checking custom cmd handler library")
            self.custom_cmd([sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key])
        self.master.iprint(f"Unknown command subtype from {sender_mac} ({self.peer_name(sender_mac)}): {subtype}")

def __handle_inf(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key):
    self.master.dprint("aen.__handle_inf")
    if (msg_subkey := "RSSI/Status/Config-Boop") and subtype == 0x00 or subtype == 0x20:  # RSSI/Status/Config-Boop
        self.master.iprint(f"{msg_subkey} ({subtype}) data received from {sender_mac} ({self.peer_name(sender_mac)}): {
              payload}")
        self.received_rssi_data[sender_mac] = payload
        # self.master.aen.__send_confirmation(self, "Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload) #confirm
              message recv
        if self.boop_irq:
            self.boop_irq()
    elif (msg_subkey := "Data") and subtype == 0x01 or subtype == 0x21:  # Sensor Data
        payload["time_sent"] = send_timestamp
        payload["time_recv"] = receive_timestamp
        self.master.iprint(f"{msg_subkey} ({subtype}) data received from {sender_mac} ({self.peer_name(sender_mac)}): {
              payload}")
        if sender_mac in self.received_sensor_data:
            self.received_sensor_data[b'prev_' + sender_mac] = self.received_sensor_data[sender_mac]
        self.received_sensor_data[sender_mac] = payload
        # self.master.aen.__send_confirmation(self, "Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload) #confirm
              message recv
        if self.data_irq:
            self.data_irq()
    elif (msg_subkey := "Message") and subtype == 0x02 or subtype == 0x22:  # Message / Other
        self.master.iprint(f"{msg_subkey} ({subtype}) received from {sender_mac} ({self.peer_name(sender_mac)}): {payload}")
        self._received_messages.append((sender_mac, payload, receive_timestamp))
        self._received_messages_size.append(payload_size)
        while len(self._received_messages) > 2048 or sum(self._received_messages_size) > 20000:
            self.master.dprint(f"Maximum buffer size reached: {len(self._received_messages)}, {sum(self.
                  _received_messages_size)} bytes; Reducing!")
            self._received_messages.pop(0)
            self._received_messages_size.pop(0)
        # self.master.aen.__send_confirmation(self, "Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload) #confirm
              message recv
        if self.msg_irq:
            self.msg_irq()
    else:
        if self.custom_inf:
            self.master.iprint(f"Checking custom inf handler library")
            self.custom_inf([sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key])
        self.master.iprint(f"Unknown info subtype from {sender_mac} ({self.peer_name(sender_mac)}): {subtype}")

def __handle_ack(self, sender_mac, subtype, send_timestamp, receive_timestamp, payload, payload_size, msg_key):
    self.master.dprint("aen.__handle_ack")
    if (msg_subkey := "Pong") and subtype == 0x10:  # Pong
        self.add_peer(sender_mac, payload[2], payload[0], payload[1])
        self.master.iprint(f"{msg_subkey} ({subtype})  received from {sender_mac} ({self.peer_name(sender_mac)}), {
              receive_timestamp - payload[3]}")
    elif (msg_subkey := "Echo") and subtype == 0x15:  # Echo
        self.master.iprint(f"{msg_subkey} ({subtype})  received from {sender_mac} ({self.peer_name(sender_mac)}), {payload
              }")
        self.last_echo = payload
    elif (msg_subkey := "Success") and subtype == 0x11:  # Success
        # payload should return a list with a cmd type and payload
        self.master.iprint(f"{msg_subkey} ({subtype}) received from {sender_mac} ({self.peer_name(sender_mac)}) for type {
              payload[0]} with payload {payload[1]}")
        # add to ack buffer
    elif (msg_subkey := "Fail") and subtype == 0x12:  # Fail
        # payload should return a list with a cmd type, error and payload
        self.master.iprint(f"{msg_subkey} ({subtype}) received from {sender_mac} ({self.peer_name(sender_mac)}) for type {
              payload[0]} with error {payload[1]} and payload {payload[2]}")
        # add to ack buffer
    elif (msg_subkey := "Confirm") and subtype == 0x13:  # Confirmation
```

```
                    # payload should return a list with message type and payload
                    self.master.iprint(f"{msg_subkey} ({subtype}) received from {sender_mac} ({self.peer_name(sender_mac)}) for type {
                        payload[0]} with payload {payload[1]}")
                    # add to ack buffer
                else:
                    if self.custom_ack:
                        self.master.iprint(f"Checking custom ack handler library")
                        self.custom_ack([sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key])
                        self.master.iprint(f"Unknown ack subtype from {sender_mac} ({self.peer_name(sender_mac)}): {subtype}, Payload: {
                            payload}")
                        # Insert more acknowledgement logic here and/or add message to acknowledgement buffer
                if self.ack_irq:
                    self.ack_irq()

            if self._aen.any():
                timestamp = time.ticks_ms()
                for mac, data in self._aen:
                    self.master.dprint(f"Received {mac, data}")
                    if mac is None:  # mac, msg will equal (None, None) on timeout
                        break
                    if data:
                        if mac and data is not None:
                            # self._received_messages.append((sender_mac, data, receive_timestamp))#Messages will be saved here, this is
                                only for debugging purposes
                            __process_message(self, mac, data, timestamp)
                    if not self._aen.any():  # this is necessary as the for loop gets stuck and does not exit properly.
                        break

# message structure (what kind of message types do I need?: Command which requires me to do something (ping, pair, change state(update,
    code, mesh mode, run a certain file), Informational Message (Sharing Sensor Data and RSSI Data)
# | Header (1 byte) | Type (1 byte) | Subtype (1 byte) | Timestamp (ms ticks) (4 bytes) | Payload type (1) | Payload (variable) |
    Checksum (1 byte) |
```

ssp_networking.py:

```
from config import mysecrets, config, whitelist, i2c_dict, version, msg_codes, msg_subcodes, networking_keys
from networking import Networking
import ubinascii
import machine
import time
import os
import gc
import webrepl


class SSP_Networking:
    def __init__(self, infmsg=False, dbgmsg=False, errmsg=False, admin=False, inittime=0):
        if infmsg:
            print(f"{(time.ticks_ms() - inittime) / 1000:.3f} Initialising Smart System Platform Networking")
        self.networking = Networking(infmsg, dbgmsg, errmsg, admin, inittime)
        config["id"] = ubinascii.hexlify(machine.unique_id()).decode()
        config["version"] = ''.join(str(value) for value in version.values())
        config["ap_mac"] = self.networking.ap.mac_decoded()
        config["sta_mac"] = self.networking.sta.mac_decoded()
        self.networking.config = config
        self.config = self.networking.config
        self.version = version
        self.orders = self.Orders(self)
        self.inittime = self.networking.inittime
        try:
            if self.config["ap_channel"] is not None:
                self.networking.ap.set_channel(self.config["ap_channel"])
            if self.config["sta_channel"] is not None:
                self.networking.sta.set_channel(self.config["sta_channel"])
        except Exception as e:
            self.networking.eprint(f"Error: {e}")
        try:
            file_path = "config.py"
            with open(file_path, "r") as f:
                lines = f.readlines()
            with open(file_path, "w") as f:
                for line in lines:
```

```
                if line.startswith(f'config = '):
                    f.write(f'config = {self.config}\n')
                else:
                    f.write(line)
    except Exception as e:
        self.networking.eprint(f"Error: {e}")

def cleanup(self):
    self.networking.cleanup()

def rssi(self):
    return self.networking.aen.rssi()

def peers(self):
    return self.networking.aen.peers()

def wpeers(self):
    self.networking.iprint(f"time.ticks_ms(): {time.ticks_ms()}")
    original_dict = self.networking.aen.peers()
    #decoded_dict = {ubinascii.hexlify(key, ':').decode(): value for key, value in original_dict.items()}
    networking_peer_info = f"networking_peers_info_start{original_dict}networking_peers_info_end"
    print(networking_peer_info)

def irq(self, func):
    self.networking.aen.irq(func)

def check_messages(self):
    return self.networking.aen.check_messages()

def return_message(self):
    return self.networking.aen.return_message()

def return_messages(self):
    return self.networking.aen.return_messages()

def return_data(self):
    return self.networking.aen.return_data()

def send_custom(self, msg_subkey, mac, payload=None, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("aen.send_custom")
    if sudo and isinstance(payload, list):
        payload.append("sudo")
    elif sudo and payload is None:
        payload = ["sudo"]
    else:
        payload = [payload, "sudo"]
    if (msg_key := "cmd") and msg_subkey in msg_subcodes[msg_key]:
        if isinstance(mac, list):
            self.networking.iprint(f"Sending {msg_subkey} ({bytes([msg_subcodes[msg_key][msg_subkey]])}) command to {mac}")
        else:
            self.networking.iprint(f"Sending {msg_subkey} ({bytes([msg_subcodes[msg_key][msg_subkey]])}) command to {mac} ({self.
                networking.aen.peer_name(mac)})")
        self.networking.aen.send_custom(msg_codes[msg_key], msg_subcodes[msg_key][msg_subkey], mac, payload, channel, ifidx)
    else:
        self.networking.iprint(f"Command {msg_subkey} not found")
    gc.collect()

def ping(self, mac, channel=None, ifidx=None):
    self.networking.dprint("net.cmd.ping")
    self.networking.aen.ping(mac, channel, ifidx)

def echo(self, mac, message, channel=None, ifidx=None):
    self.networking.dprint("net.cmd.echo")
    self.networking.aen.echo(mac, message, channel, ifidx)

def boop(self, mac, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.boop")
    self.networking.aen.boop(mac, channel, ifidx)

def send(self, mac, message, channel=None, ifidx=None):
    self.networking.dprint("net.cmd.message")
    self.networking.aen.send(mac, message, channel, ifidx)

def broadcast(self, message, channel=None, ifidx=None):
    self.networking.dprint("net.cmd.broadcast")
    mac = b'\xff\xff\xff\xff\xff\xff'
```

```
        self.send(mac, message, channel, ifidx)

    def send_data(self, mac, message, channel=None, ifidx=None):  # message is a dict, key is the sensor type and the value is the
          sensor value
        self.networking.dprint("net.cmd.message")
        self.networking.aen.send_data(mac, message, channel, ifidx)

    def reboot(self, mac, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.reboot")
        self.send_custom("Reboot", mac, None, channel, ifidx, sudo)

    def hive_set(self, mac, hive_bool, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.hive_set")
        self.send_custom("Hive-Set", mac, hive_bool, channel, ifidx, sudo)

    def hive_configure(self, mac, hive_config, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.hive_configure")
        self.send_custom("Hive-Configure", mac, f"{hive_config}", channel, ifidx, sudo)

    def firmware_update(self, mac, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.firmware_update")
        self.send_custom("Firmware-Update", mac, None, channel, ifidx, sudo)

    def file_update(self, mac, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.file_update")
        self.send_custom("File-Update", mac, None, channel, ifidx, sudo)

    def file_download(self, mac, link, file_list=None, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.file_download")
        self.send_custom("File-Download", mac, [link, file_list], channel, ifidx, sudo)

    def web_repl(self, mac, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.web_repl")
        self.networking.ap.set_ap(ap_name := self.networking.config["name"], password := networking_keys["default_ap_key"])
        self.send_custom("Web-Repl", mac, [ap_name, password], channel, ifidx, sudo)
        # await success message and if success False disable AP or try again

    def file_run(self, mac, filename, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.file_run")
        self.send_custom("File-Run", mac, filename, channel, ifidx, sudo)

    def admin_set(self, mac, new_bool, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.admin_set")
        self.send_custom("Admin-Set", mac, new_bool, channel, ifidx, sudo)

    def whitelist_add(self, mac, mac_list=None, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.whitelist_add")
        if mac_list is not None:
            mac_list = [self.networking.sta.mac_decoded, self.networking.ap.mac_decoded]
        self.send_custom("Whitelist-Add", mac, mac_list, channel, ifidx, sudo)

    def config_change(self, mac, new_config, hardcode=False, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.config_change")
        self.send_custom("Config-Change", mac, [new_config, hardcode], channel, ifidx, sudo)

    def name_change(self, mac, new_name, hardcode=False, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.name_change")
        self.send_custom("Name-Change", mac, [new_name, hardcode], channel, ifidx, sudo)

    def pair(self, mac, key=networking_keys["handshake_key1"], channel=None, ifidx=None):
        self.networking.dprint("net.cmd.pair")
        self.send_custom("Pair", mac, key, channel, ifidx)

    def pair_enable(self, mac, pair_bool, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.pair")
        self.send_custom("Set-Pair", mac, pair_bool, channel, ifidx, sudo)

    def directory_get(self, mac, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.directory_get")
        self.send_custom("Directory-Get", mac, None, channel, ifidx, sudo)

    # resend cmd

    def wifi_connect(self, mac, name, password, channel=None, ifidx=None, sudo=False):
        self.networking.dprint("net.cmd.wifi_connect")
        self.send_custom("Wifi-Connect", mac, [name, password], channel, ifidx, sudo)
```

```
def wifi_disconnect(self, mac, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.wifi_disconnect")
    self.send_custom("Wifi—Disconnect", mac, None, channel, ifidx, sudo)

def ap_enable(self, mac, name, password, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.ap_enable")
    self.send_custom("AP—Enable", mac, [name, password], channel, ifidx, sudo)

def ap_disable(self, mac, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.ap_disable")
    self.send_custom("AP—Disable", mac, None, channel, ifidx, sudo)

def pause(self, mac, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.pause")
    self.send_custom("Pause", mac, None, channel, ifidx, sudo)

def resume(self, mac, channel=None, ifidx=None, sudo=False):
    self.networking.dprint("net.cmd.resume")
    self.send_custom("Resume", mac, None, channel, ifidx, sudo)


class Orders:
    def __init__(self, master):
        self.master = master
        self.master.networking.dprint("net.cmd.orders")
        self._whitelist = whitelist

        self._pause_function = None

        # Flags
        self._pairing_enabled = True
        self._pairing = False
        self._paired = False
        self._paired_macs = []
        self._running = True

        def __check_authorisation(sender_mac, payload):
            return sender_mac in self._whitelist or payload == "sudo" or payload[-1] == "sudo"

        def __send_confirmation(msg_type, recipient_mac, msg_subkey_type, payload=None, error=None):
            self.master.networking.dprint("net.order.__send_confirmation")
            self.master.networking.aen.__send_confirmation(msg_type, recipient_mac, msg_subkey_type, payload, error)

        def custom_cmd_handler(data):
            self.master.networking.dprint("net.order.custom_cmd_handler")
            sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key = data
            if (msg_subkey := "Reboot") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Reboot
                self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
                    .aen.peer_name(sender_mac)})")
                if __check_authorisation(sender_mac, payload):
                    __send_confirmation("Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload)
                    machine.reset()
                else:
                    __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
            elif (msg_subkey := "Hive—Set") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Hive—Set
                self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
                    .aen.peer_name(sender_mac)})")
                if __check_authorisation(sender_mac, payload):
                    try:
                        from config import hive_config
                        hive_config["hive"] = payload[0]
                        file_path = "config.py"
                        with open(file_path, "r") as f:
                            lines = f.readlines()
                        with open(file_path, "w") as f:
                            for line in lines:
                                if line.startswith(f'hive_config = '):
                                    f.write(f'hive_config = {hive_config}\n')
                                else:
                                    f.write(line)
                        __send_confirmation("Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload)
                        machine.reset()
                    except Exception as e:
                        self.master.networking.eprint(f"Error: {e} with payload: {payload}")
                        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
```

```
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Hive—Configure") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Hive—Configure
    self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
        .aen.peer_name(sender_mac)})")
    if __check_authorisation(sender_mac, payload):
        try:
            #from config import hive_config
            #hive_config.update(payload[0])
            hive_config = payload[0]
            file_path = "config.py"
            with open(file_path, "r") as f:
                lines = f.readlines()
            with open(file_path, "w") as f:
                for line in lines:
                    if line.startswith(f'hive_config = '):
                        f.write(f'hive_config = {hive_config}\n')
                    else:
                        f.write(line)
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
            machine.reset()
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Firmware—Update") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Firmware—Update
    self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
        .aen.peer_name(sender_mac)})")
    if __check_authorisation(sender_mac, payload):
        try:
            # Insert update logic here
            self.master.networking.iprint("no update logic written just yet")
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "File—Update") and subtype == msg_subcodes[msg_key][msg_subkey]:  # File—Update
    self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
        .aen.peer_name(sender_mac)})")
    if __check_authorisation(sender_mac, payload):
        try:
            # Insert update logic here
            self.master.networking.iprint("No update logic written just yet")
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "File—Download") and subtype == msg_subcodes[msg_key][msg_subkey]:  # File—Download
    self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
        .aen.peer_name(sender_mac)})")
    # should return a list with a link and the list of files to download
    if __check_authorisation(sender_mac, payload):
        try:
            # import mip
            # base = payload[0]
            # files_to_copy = payload[1]
            # if files_to_copy is None:
            #     mip.install(base)
            # else:
            #     for f in files_to_copy:
            #         mip.install(base + f)
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Web—Repl") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Web—Repl
    self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking
        .aen.peer_name(sender_mac)})")
    # should be a list with name and password
```

```
    if __check_authorisation(sender_mac, payload):
        try:
            # add logic to connect to Wi-Fi and set up webrepl
            webrepl.start()
            self.master.sta.connect(payload[0], payload[1])
            link = "webrepl link"
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", link)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "File-Run") and subtype == msg_subcodes[msg_key][msg_subkey]:  # File-Run
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        try:
            file_name = payload[0]
            if not file_name.endswith(".py"):
                file_name += ".py"
            with open(file_name) as f:
                code = f.read()
            __send_confirmation("Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload)
            exec(code)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Set-Admin") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Set Admin Bool
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        self.master.networking.iprint(f"Received Set-Admin command: self.admin set to {payload[0]}")
        try:
            self.master.admin = payload[0]
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Whitelist-Add") and subtype == msg_subcodes[msg_key][
    msg_subkey]:  # Whitelist-Add - Add Admin macs to _whitelist
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        self.master.networking.iprint(
            f"Received add admin macs to _whitelist command, added {payload[0]} and {payload[1]}")
        try:
            self._whitelist.append(ubinascii.unhexlify(payload[0].replace(':', '')))
            self._whitelist.append(ubinascii.unhexlify(payload[1].replace(':', '')))
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Config-Change") and subtype == msg_subcodes[msg_key][msg_subkey]:  # change config
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        try:
            self.master.config.update(payload[0])
            if payload[1]:
                file_path = "config.py"
                with open(file_path, "r") as f:
                    lines = f.readlines()
                with open(file_path, "w") as f:
                    for line in lines:
                        if line.startswith(f'config = '):
                            f.write(f'config = {self.master.config}\n')
                        else:
                            f.write(line)
```

```
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Name-Change") and subtype == msg_subcodes[msg_key][msg_subkey]:  # change name
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        try:
            self.master.config["name"] = payload[0]
            if payload[1]:
                file_path = "config.py"
                with open(file_path, "r") as f:
                    lines = f.readlines()
                with open(file_path, "w") as f:
                    for line in lines:
                        if line.startswith('    "name": '):
                            f.write(f'    "name": "{payload[0]}"\n')
                        else:
                            f.write(line)
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "Ping") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Ping
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    self.master.networking.aen.add_peer(sender_mac, payload[2]["name"], payload[2]["id"], payload[2]["configuration"],
                payload[2]["version"], payload[0], payload[1])
    if bool(self.master.networking.aen.ifidx):
        channel = self.master.ap.channel()
    else:
        channel = self.master.sta.channel()
    response = [channel, self.master.networking.aen.ifidx, self.master.config, send_timestamp]
    self.master.networking.aen.send_custom(0x03, 0x10, sender_mac, response)
#                 elif (msg_subkey := "Pair") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Pair #add something
    that checks that the messages are from the same mac # BREAKS NETWORK
#                     self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self
    .peer_name(sender_mac)})")
#                     if self._pairing_enabled and networking_keys["handshake_key_1"] == payload:
#                         self._pairing = True
#                         self.pair(sender_mac, networking_keys["handshake_key_2"])
#                         # some timer for if key 3 is not received to reset states
#                     elif self._pairing_enabled and self._pairing and networking_keys["handshake_key_2"] == payload:
#                         self._paired = True
#                         self._paired_macs.append(sender_mac)
#                         self.pair(sender_mac, networking_keys["handshake_key_3"])
#                         # some timer that sets to false if key 4 is not received
#                     elif self._pairing_enabled and self._pairing and networking_keys["handshake_key_3"] == payload:
#                         try:
#                             # Insert pairing logic here do a reverse handshake
#                             self._paired = True
#                             self._paired_macs.append(sender_mac)
#                             self.pair(sender_mac, networking_keys["handshake_key_4"])
#                             self.master.networking.iprint("no pairing logic written just yet")
#                             __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
#                         except Exception as e:
#                             self.master.networking.eprint(f"Error: {e} with payload: {payload}")
#                             __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
#                     elif self._pairing_enabled and self._pairing and networking_keys["handshake_key_3"] == payload:
#                         self._paired = True
#                         # remove timer from before
#                     else:
#                         __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", "Pairing disabled",
    payload)
#                 elif (msg_subkey := "Set-Pair") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Enable pairing
    mode
#                     self.master.networking.iprint(f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self
    .peer_name(sender_mac)})")
#                     if __check_authorisation(sender_mac, payload):
#                         try:
#                             self._pairing_enabled = payload[0]
```

```
#                           __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
#                       except Exception as e:
#                           self.master.networking.eprint(f"Error: {e} with payload: {payload}")
#                           __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
#                   else:
#                       __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised
    ")
elif (msg_subkey := "RSSI/Status/Config-Boop") and subtype == msg_subcodes[msg_key][
    msg_subkey]:  # RSSI/Status/Config Boop
    self.boops = self.boops + 1
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            }), Received total of {self.boops} boops!")
    try:
        self.master.networking.aen.send_custom(0x02, 0x20, sender_mac,[self.master.config, self.master.version, self.
            master.sta.mac, self.master.ap.mac, self.master.networking.aen.rssi()])  # [ID, Name, Config, Version, sta
             mac, ap mac, rssi]
    except Exception as e:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
elif (msg_subkey := "Directory-Get") and subtype == msg_subcodes[msg_key][
    msg_subkey]:  # Get List of Files # BREAKS NETWORK
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    try:
        def ___list_all_files(path):
            result = []
            entries = os.listdir(path)
            for entry in entries:
                full_path = path + "/" + entry
                try:
                    if os.stat(full_path)[0] & 0x4000:
                        result.extend(___list_all_files(full_path))
                    else:
                        result.append(full_path)
                except OSError:
                    # Handle inaccessible files or directories
                    continue
            return result

        start_path = '.'
        all_files = ___list_all_files(start_path)
        self.master.networking.aen.send_custom(0x02, 0x20, sender_mac, all_files)
    except Exception as e:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
elif (msg_subkey := "Echo") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Echo
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            }): {payload}")  # Check i or d
    self.master.networking.aen.send_custom(0x03, 0x15, sender_mac, payload)
elif (msg_subkey := "Resend") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Resend lost long messages
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    self.master.networking.iprint("Long_sent_buffer disabled due to memory constraints")
elif (msg_subkey := "WiFi-Connect") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Connect to Wi-Fi
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        try:
            self.master.sta.connect(payload[0], payload[1])
            __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
        except Exception as e:
            self.master.networking.eprint(f"Error: {e} with payload: {payload}")
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
    else:
        __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
elif (msg_subkey := "WiFi-Disconnect") and subtype == msg_subcodes[msg_key][
    msg_subkey]:  # Disconnect from Wi-Fi
    self.master.networking.iprint(
        f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
            })")
    if __check_authorisation(sender_mac, payload):
        try:
            self.master.sta.disconnect()
```

```
                __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
            except Exception as e:
                self.master.networking.eprint(f"Error: {e}")
                __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
        else:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
    elif (msg_subkey := "AP-Enable") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Enable AP
        self.master.networking.iprint(
            f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
                })")
        if __check_authorisation(sender_mac, payload):
            try:
                ssid = payload[0]
                if ssid == "":
                    ssid = self.master.config["name"]
                password = payload[1]
                self.master.ap.setap(ssid, password)
                __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
            except Exception as e:
                self.master.networking.eprint(f"Error: {e} with payload: {payload}")
                __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
        else:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
    elif (msg_subkey := "AP-Disable") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Disable AP
        self.master.networking.iprint(
            f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
                })")
        if __check_authorisation(sender_mac, payload):
            try:
                self.master.ap.deactivate()
                __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
            except Exception as e:
                self.master.networking.iprint(f"Error: {e}")
                __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
        else:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
    elif (msg_subkey := "Pause") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Set Pause
        self.master.networking.iprint(
            f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
                })")
        if __check_authorisation(sender_mac, payload):
            try:
                self.master.networking.iprint(f"Received pause command: {payload[0]}")
                self._running = False
                __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
                if self._pause_function:
                    self._pause_function()  # calls the custom set pause function to display a screen
                while not self._running:
                    time.sleep(0.5)
            except Exception as e:
                __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
        else:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
    elif (msg_subkey := "Resume") and subtype == msg_subcodes[msg_key][msg_subkey]:  # Set Continue
        self.master.networking.iprint(
            f"{msg_subkey} ({subtype}) command received from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)
                })")
        if __check_authorisation(sender_mac, payload):
            try:
                self.master.networking.iprint(f"Received continue command: {payload}")
                self.master._running = True
                __send_confirmation("Success", sender_mac, f"{msg_subkey} ({subtype})", payload)
            except Exception as e:
                __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, e)
        else:
            __send_confirmation("Fail", sender_mac, f"{msg_subkey} ({subtype})", payload, "Not authorised")
    else:
        self.master.networking.iprint(
            f"Unknown command subtype from {sender_mac} ({self.master.networking.aen.peer_name(sender_mac)}): {subtype}")

def custom_inf_handler(data):
    self.master.networking.dprint("net.order.custom_inf_handler")
    sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key = data
    if (msg_subkey := "Directory") and subtype == msg_subcodes[msg_key][msg_subkey]:  # File Directory
        self.master.networking.iprint(f"{msg_subkey} ({subtype}) data received from {sender_mac} ({self.master.networking.
            aen.peer_name(sender_mac)}): {payload}")
```

```
        # __send_confirmation("Confirm", sender_mac, f"{msg_subkey} ({subtype})", payload) #confirm message recv

def custom_ack_handler(data):
    self.master.networking.dprint("net.order.custom_ack_handler")
    sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key = data
    # data contains [sender_mac, subtype, send_timestamp, receive_timestamp, payload, msg_key]


self.master.networking.aen.cmd(custom_cmd_handler)
self.master.networking.aen.inf(custom_inf_handler)
self.master.networking.aen.ack(custom_ack_handler)
```

Appendix D - GitHub Automation Code

GitHub Actions Code:

```
name: release-render-pipeline

on:
  pull_request:
    types:
      - closed
    branches:
      - main

jobs:
  release-render:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v4
      with:
        ssh-key: ${{ secrets.RENDER_KEY }}

    - name: Set up SSH for Git
      run: |
        mkdir -p ~/.ssh
        echo "${{ secrets.RENDER_KEY }}" > ~/.ssh/id_rsa
        chmod 600 ~/.ssh/id_rsa
        ssh-keyscan github.com >> ~/.ssh/known_hosts
        eval "$(ssh-agent -s)"
        ssh-add ~/.ssh/id_rsa

    - name: Parse File List from release/README.qmd
      id: parse
      run: |
        # Extract the table section and parse the file paths
        raw_file_list=$(awk '/\| name/ {f=1; next} /\|---/ {next} f && /\|/ {print $2}' software/release/README.qmd | tr -d ' ')

        file_list=$(echo "$raw_file_list" | sed 's/^/software\//')

        echo "Files to monitor:"
        echo "$file_list"

        echo "file_list<<EOF" >> $GITHUB_OUTPUT
        echo "$file_list" >> $GITHUB_OUTPUT
        echo "EOF" >> $GITHUB_OUTPUT

    - name: Copy Files to Release Folder
      run: |
        mkdir -p software/release  # Ensure the release folder exists
        echo "${{ steps.parse.outputs.file_list }}" | while read file; do
          # Skip the config.py file
          if [[ "$(basename $file)" == "config.py" ]]; then
            echo "Skipping config.py"
            continue
          fi
```

```
      # Copy the file, overwriting if it already exists
      cp "$file" software/release/  # Overwrite files directly to the root of the release folder
    done

- name: List Files in Release Folder
  run: |
    echo "Files copied to the release folder:"
    find software/release -type f

- name: Update Version in Config.py
  run: |
    # Get the list of changed (staged) files using git diff
    changed_files=$(git diff --name-only)

    new_files=$(git ls-files --others --exclude-standard)

    all_changed_files=$(echo "$changed_files $new_files" | tr ' ' '\n')

    echo "${{ steps.parse.outputs.file_list }}" | while read file; do
      base_file=$(basename "$file")

      if echo "$all_changed_files" | grep -q "$base_file"; then
        if grep -q "$base_file" software/release/config.py; then
          current_version=$(sed -n "s/^.*'$base_file': \([0-9]*\),.*$/\1/p" software/release/config.py)

          new_version=$((current_version + 1))

          sed -i "s/^.*'$base_file': $current_version,.*$/'$base_file': $new_version,/g" software/release/config.py

          echo "Updated $base_file version from $current_version to $new_version"
        else
          sed -i "/^version = {/a \ \ \ '$base_file': 1," software/release/config.py

          echo "Added $base_file with version 1"
        fi
      else
        echo "No change detected for $base_file"
      fi
    done

- name: Set up R
  uses: r-lib/actions/setup-r@v2

- name: Install Required R Packages
  run: |
    Rscript -e 'if (!requireNamespace("rmarkdown", quietly = TRUE)) install.packages("rmarkdown", repos = "https://cloud.r-project.
        org/")'
    Rscript -e 'if (!requireNamespace("knitr", quietly = TRUE)) install.packages("knitr", repos = "https://cloud.r-project.org/")'

- name: Set up Quarto
  uses: quarto-dev/quarto-actions/setup@v2

- name: Find and Render README.qmd Files
  run: |
    find . -name "README.qmd" | while read qmd_file; do
      quarto render "$qmd_file" --to gfm
    done

# --- Step 4: Commit and Push All Changes Once ---
- name: Commit and Push All Changes
  run: |
    # Configure Git
    git config user.name "github-actions[bot]"
    git config user.email "github-actions[bot]@users.noreply.github.com"

    # Check if there are changes to commit (release folder or rendered README files)
    git diff --exit-code || (git add software/release/ && git add . && git commit -m "Release and Render")

    # Push the changes if there were any
    git push origin main || echo "No changes to push"

- name: Set up Pull Request from main to original branch
  env:
      GH_TOKEN: ${{ github.token }}
  run: |
    # Ensure the latest changes from remote are fetched
```

```
git fetch origin

# Get the original branch from the pull request
original_branch="dev/nick"

# Check if the original branch exists
if git show-ref --verify --quiet "refs/remotes/origin/$original_branch"; then
  # Create a pull request from main to the original branch
  pr_url=$(gh pr create --base "$original_branch" --head "main" --title "Sync main with $original_branch" --body "Automated sync
        from main branch" --fill || echo "")

  # Merge the pull request if created
  if [[ -n "$pr_url" ]]; then
    pr_number=$(echo "$pr_url" | grep -oP '\d+$')
    gh pr merge "$pr_number" --merge --admin || echo "Failed to merge PR $pr_number"
  else
    echo "No new pull request was created."
  fi
else
  echo "Error: The branch '$original_branch' does not exist remotely."
fi
```

Example Quatro README File:

```
---
format: gfm
execute:
  echo: false
  message: false
  warning: false
editor: visual
editor_options:
  chunk_output_type: console
---

# Smart System Education Platform

```{r setup, include=FALSE}
file_path <- fs::path_real("README.qmd")
path_components <- strsplit(file_path, "/")[[1]]
shortened_path <- fs::path_join(path_components[-c(2:6)])
```

Location: `r shortened_path`

Description

Example description of the directory.

Directories & Files

The repository has the following directory tree.

```{r}
fs::dir_tree(recurse = 1)
```

A short description of the directories can be found below.

name	description	contribution
example	Example description	Example Contributor
```

Appendix E - Test Code

Maximum Range Test and Reliability Test Code

boop-o-meter Code:

```
from machine import Pin, SoftI2C, PWM, ADC

#custom libraries
from networking import Networking
from secrets import codes

#libraries
import espnow
import asyncio
import time
import random
import ubinascii

#hardware support
import ssd1306
import servo
import machine


#Initialisation
#define buttons , sensors and motors
#nav switches
switch_down = Pin(8, Pin.IN, Pin.PULL_UP)
switch_select = Pin(9, Pin.IN, Pin.PULL_UP)
switch_up= Pin(10, Pin.IN, Pin.PULL_UP)

i2c = SoftI2C(scl = Pin(7), sda = Pin(6))

# Initiate display
oled = ssd1306.SSD1306_I2C(128, 64, i2c)
oled.fill(0)
oled.show()
oled.text("boop-o-meter 300",0, 28, 1)
oled.show()
print("boop-o-meter 300")
print("Initialising!")

# Scan I2C devices
devices = i2c.scan()
for device in devices:
    print("I2C device found at address:", hex(device))

#servo
s = servo.Servo(Pin(2))

#unique name
ID = ubinascii.hexlify(machine.unique_id()).decode()

# Example usage
```

```
networking = Networking()
peer_mac = b'\xff\xff\xff\xff\xff\xff'
networking.aen.add_peer(peer_mac)
#networking.aen.send(peer_mac, b'Boop')
message = b'Boop'
networking.aen.send(peer_mac, message)
networking.aen.send(peer_mac, message, 0x01, 0x10)
#networking.aen.remove_peer(peer_mac)

async def main():
    last_check_time = 0
    while True:
        await asyncio.sleep(5)
        #if networking.aen.received_messages:
        #    for sender_mac, data, timestamp in networking.aen.received_messages:
        #        if timestamp > last_check_time:
        #            print(f"Processed received message from {sender_mac}: {data}")
        #    last_check_time = time.time_ns()
        networking.aen.send(peer_mac, message, 0x01, 0x10)
        #print(f" RSSI Table at {time.ticks_ms()}: {networking.aen.get_rssi_table()}")

# Initialize the event loop
def loop():
    loop = asyncio.get_event_loop()
    loop.create_task(main())
    loop.run_forever()

async def clear_message(current_id, delay=2.5):
    await asyncio.sleep(delay)
    if current_id == message_id:
        oled.fill_rect(0, 10, 128, 9, 0)
        oled.show()

# Menu management
def show_count():
    oled.fill_rect(0, 0, 128, 9, 0)
    if boopedn > 999 and boopn > 999:
        oled.text(f"Rx: uwu; Tx: wow", 0, 0, 1)
    elif boopn > 999:
        oled.text(f"Rx: {boopedn}; Tx: wow", 0, 0, 1)
    elif boopedn > 999:
        oled.text(f"Rx: uwu; Tx: {boopn}", 0, 0, 1)
    else:
        oled.text(f"Rx: {boopedn}; Tx: {boopn}", 0, 0, 1)
    oled.show()

def show_message(message):
    global message_id
    message_id += 1
    oled.fill_rect(0, 10, 128, 9, 0)
    oled.text(message, 0, 10, 1)
    oled.show()
    asyncio.create_task(clear_message(message_id))

def show_list(display_list, hindex):
    oled.fill_rect(0, 20, 128, 1, 1)
    oled.fill_rect(0, 21, 128, 44, 0)
    pos = 23
    show_count()
    max_displayed_items = 4

    if hindex >= 4:
        oled.text("^", 0, 23, 1)
    if hindex/4 < len(display_list)//4:
        rotated_caret = [0b00000000,
                         0b00011000,
                         0b00111100,
                         0b01100110,
                         0b00000000,
                         ]
        for row in range(len(rotated_caret)):
            for col in range(8):
                if rotated_caret[row] & (1 << col):
                    oled.pixel(0 + col, 57 + (3 - row), 1)
        #oled.text("v", 0, 53, 1)  # Arrow for items below
```

```
    start_index = max(0, (hindex//4)*4)
    for index in range(start_index, min(len(display_list), start_index+4)):
        if index == hindex:
            oled.fill_rect(8, pos-1, 128, 9, 1)  # Highlight selected item
            oled.text(str(display_list[index]), 8, pos, 0)  # Display selected item
        else:
            oled.text(str(display_list[index]), 8, pos, 1)  # Display unselected item
        pos += 10  # Move position down for the next item
    oled.show()

def is_cooldown(cooldown):
    global last_press_time
    print(time.time_ns()//1000000-last_press_time)
    return (time.time_ns()//1000000 - last_press_time) < cooldown #prevent the scan function from being spammed

def update_last_press_time():
    global last_press_time
    last_press_time = time.time_ns()//1000000 #ms for accuracy
    print(last_press_time)

def up(pin):
    if is_cooldown(100):
        return
    update_last_press_time()
    global hindex, clist
    hindex = (hindex - 1) % len(clist)  # Wrap around
    show_list(clist, hindex)

def down(pin):
    if is_cooldown(100):
        return
    update_last_press_time()
    global hindex, clist
    hindex = (hindex + 1) % len(clist)  # Wrap around
    show_list(clist, hindex)

def action(pin):
    if is_cooldown(100):
        return
    global hindex, clist, mindex, boopn
    if not (mindex == 0 and hindex == 0):
        update_last_press_time()
    if mindex == 0:
        if hindex == 0:
            if is_cooldown(1100):
                return  #Prevent Scan from being spammed
            update_last_press_time()
            show_message(" Scanning...")
            pairing(1,32)
            show_message(f" {len(peer)} peers found!")
            asyncio.create_task(clear_message(message_id))
        elif hindex == 1:
            clist = peer[:] + ["Back"]
            mindex = 1
            hindex = 0
            show_list(clist, hindex)
    elif mindex == 1:
        if hindex == len(clist) - 1:
            print("Going Back")
            clist = mainmenu[:]
            mindex = 0
            hindex = 0
            show_list(clist, hindex)
        else:
            selected_peer = clist[hindex]
            print(f" Booping: {selected_peer}")
            e.send(selected_peer, "Boop")
            boopn += 1
            show_list(clist, hindex)
            show_message(f"Booping {selected_peer}!")
            asyncio.create_task(clear_message(message_id))

# Main code
time.sleep(3)
#Initialisation
print("Finding peers...")
```

```
pairing(1, 32)

# Set up interrupt handlers for button presses
switch_up.irq(trigger=Pin.IRQ_FALLING, handler=down)
switch_down.irq(trigger=Pin.IRQ_FALLING, handler=up)
switch_select.irq(trigger=Pin.IRQ_FALLING, handler=action)

clist = mainmenu
show_list(clist, hindex)

loop()
```

Ping Response Time Code

```
import network
import espnow
import time

#Set up the network and espnow
staif = network.WLAN(network.STA_IF)
staif.active(True)
aen = espnow.ESPNow()
aen.active(True)
#add the send to all mac address to esp now (maximum of 20 peers can be added)
peer = b'\xff\xff\xff\xff\xff\xff'
aen.add_peer(peer)

start = time.ticks_ms()

durations = {}

#callback function that retrieves the last message from the message buffer
def irq_receive(aen):
    tpl = aen.irecv() #returns a tuple of mac and message
    mac, msg = tpl
    try: #This only works in a non message saturated environment as I'm only ever retrieving the last received message
        numb = int.from_bytes(msg, 'big')
        dur = time.ticks_ms()-numb
        tim = f"{(time.ticks_ms() - start) / 1000:.3f}"
        print(f"At {tim} from: {mac}: {dur}") #prints the time from since the message was originally sent until the echo has been
              received back
        durations[tim] = dur
    except Exception as e:
        print(f"Error: {e}")

aen.irq(irq_receive)#IRQ Trigger which is called as soon as as possible after a message is received

while True and time.ticks_ms()-start < 10000:
    num = time.ticks_ms()
    aen.send(peer, num.to_bytes(4, 'big'))
    time.sleep(1)

print(durations)
```

Response Time, RSSI and Packet Loss Rate Depending on Range Test Code

```
import network
import espnow
import time

#Set up the network and espnow
staif = network.WLAN(network.STA_IF)
staif.active(True)
aen = espnow.ESPNow()
```

```
aen.active(True)
#add the send to all mac address to esp now (maximum of 20 peers can be added)
peer = b'\xff\xff\xff\xff\xff\xff'
aen.add_peer(peer)

start = time.ticks_ms()

durations = {}

#callback function that retrieves the last message from the message buffer
def irq_receive(aen):
    tpl = aen.irecv() #returns a tuple of mac and message
    mac, msg = tpl
    try: #This only works in a non message saturated environment as I'm only ever retrieving the last received message
        numb = int.from_bytes(msg, 'big')
        dur = time.ticks_ms()-numb
        tim = f"{(time.ticks_ms() - start) / 1000:.3f}"

        print(f"At {(tim)} from: {mac}: {aen.peers_table[mac]}") #prints the time from since the message was originally sent until the
            echo has been received back
        durations[tim] = dur
    except Exception as e:
        print(f"Error: {e}")

aen.irq(irq_receive)#IRQ Trigger which is called as soon as as possible after a message is received

while True and time.ticks_ms()-start < 10000:
    num = time.ticks_ms()
    aen.send(peer, num.to_bytes(4, 'big'))
    time.sleep(1)

print(durations)
```

Power Consumption and Battery Endurance Test Code

```
import network
import espnow
import time

#Set up the network and espnow
staif = network.WLAN(network.STA_IF)
staif.active(True)
aen = espnow.ESPNow()
aen.active(True)
#add the send to all mac address to esp now (maximum of 20 peers can be added)
peer = b'\xff\xff\xff\xff\xff\xff'
aen.add_peer(peer)

start = time.ticks_ms()

durations = {}

#callback function that retrieves the last message from the message buffer
def irq_receive(aen):
    tpl = aen.irecv() #returns a tuple of mac and message
    mac, msg = tpl
    try: #This only works in a non message saturated environment as I'm only ever retrieving the last received message
        numb = int.from_bytes(msg, 'big')
        dur = time.ticks_ms()-numb
        tim = f"{(time.ticks_ms() - start) / 1000:.3f}"

        print(f"At {(tim)} from: {mac}: {aen.peers_table[mac]}") #prints the time from since the message was originally sent until the
            echo has been received back
        durations[tim] = dur
    except Exception as e:
        print(f"Error: {e}")

aen.irq(irq_receive)#IRQ Trigger which is called as soon as as possible after a message is received

while True and time.ticks_ms()-start < 10000:
    num = time.ticks_ms()
    aen.send(peer, num.to_bytes(4, 'big'))
```

```
        time.sleep(1)

    print(durations)
```

Appendix F - First Hackathon Results

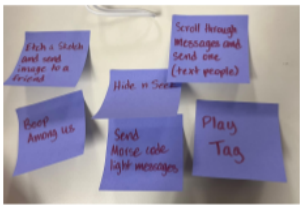
Team 1 -

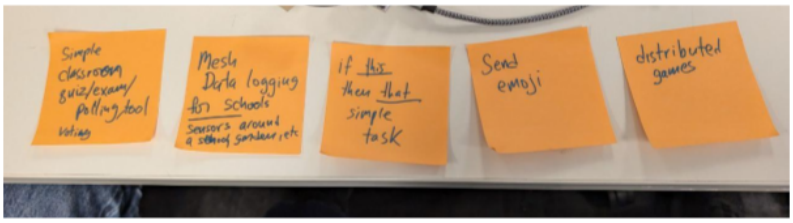
Ideas

- Messaging service
  - Ut enim ad minim veniam, quis nostrud exercitation.
  - Tempor incididunt incididunt ut labore.
-

ETH zürich Tufts 30.09.24 10

Team 2:

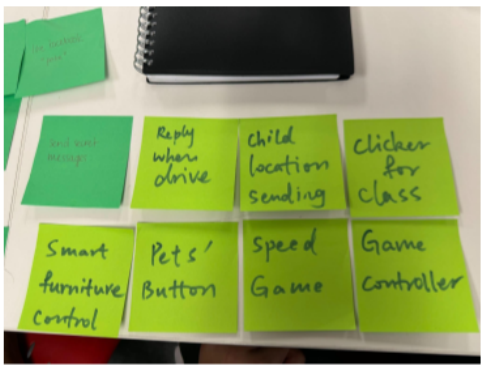


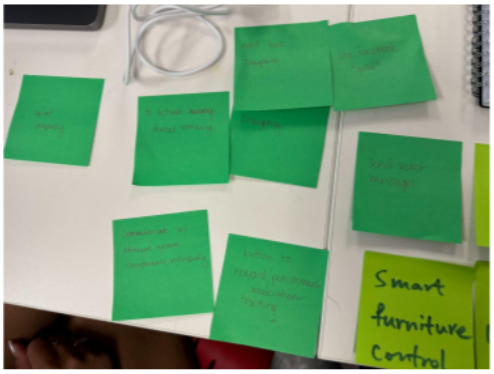


We spent time trying to find our mac address and recognize it (written in bytes) in the list.

ETH zürich Tufts 30.09.24 11

Team 3 -





Team 4:

Boop Assassin Game

- Circle of people, each one has one target (like in the game “Assassin”)
  - You know the codename of your target and can boop them (or anyone else) and get their distance from you
  - If you boop them from within 2 meters, they are dead and you get their target
  - Last person standing wins
- A team car game where you can ping another car and you can receive sensor information from that car

Useful things to add in examples:

print("My MAC ID is")

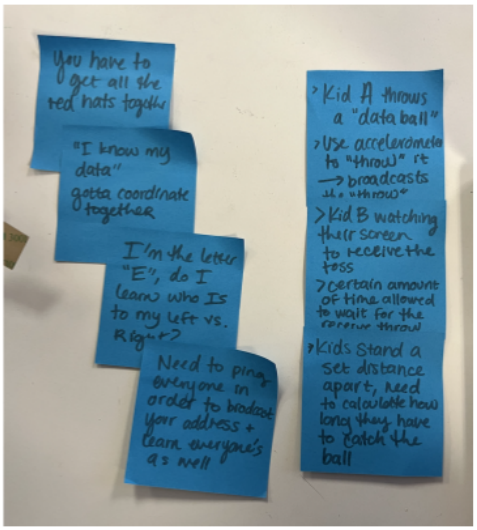
print(networking.mac())

networking.set_channel(4) # can pick a channel 0-14 to communicate (less noisy)

Team 5 -

Digital Catch

- Student A “throws” a data ball with the accelerometer
- Student B watches screen for the “throw”
- Student B has a limited amount of time to click button to “catch” the ball.
  - If the throw was fast, the student has less time
  - If the throw was slow, the student has more time
- Student B catches ball successfully, you win!



Appendix G - Second Hackathon Questionnaire and Results

Questionnaire

Tufts UNIVERSITY

Greetings, I am a mechanical engineering student from ETH Zurich, Switzerland currently writing my Master's thesis here at Tufts University in Massachusetts, the United States.

As part of my master's thesis I am developing the Smart System Platform (SSP), which is building up on the existing Smart Motors Project and aims to develop it further for more wide-spread application. The goal is to develop an educational platform which can not only be used to teach certain science and engineering principles intuitively out o the box, but also to enable educators and academia to build lessons anc prototype using said platform.
As you are either participating in the SSP-Hackathon or have used some of the developed tools in a different context, I would like to ask you some questions for my research to gather feedback on said tools for future improvement.
The survey by itself will take about five minutes of your time. If you are participating in the Hackathon, you will first be given a brief introduction about the SSP, you will then be asked to split into groups and develop a small project using the SSP hardware and development tools, after which you will then present your project to the group and share some feedback. Afterwards you will then be given some time to filling out this survey. The Hackathon including this survey will take 60 minutes of your time.

It's your decision, and there are no consequences to not participating. I further don't anticipate any major risks to participation. If at any time during the form you do not want to answer a question, you can skip it. If at any time you want to stop participating, or you don't want to start at all, you may feel free to quit the survey. Before submitting the survey, none of the information will be saved.

The content of this survey may be used in publications or presentations, however identifiable information about you will not be shared beyond the research team. Your privacy is our top priority, but there is always a slight chance that someone could find out about your participation in this survey. People responsible for monitoring this research may be able to access the data. This includes the Tufts University IRB. Below you can find my contact information and the contact information of the research oversight board at Tufts, the Tufts SBER IRB, if you need to get in touch about this research at any point in the future.

If you agree and would like to consent to participate in this Hackathon and/or questionnaire regarding the experience using the Smart System Platform, please indicate so below.

Do you consent to participate?

◯ I consent

For questions or concerns about the research study or procedures, or if you need to notify someone of a complaint, please contact the Tufts research team.

PI Contact Info:
Nicolas Triebold
Tufts University
Center for Engineering Education and Outreach
Email: nicolas.triebold@tufts.edu
Phone Number: 617-798-4262

Faculty Advisor Contact Info:
Chris Rogers
Tufts University
Center for Engineering Education and Outreach
Email: chris.rogers@tufts.edu
Phone number: 617-627-2882

If you have questions or concerns about your rights as a research participant, or if you would like to discuss the study with someone outside of the research team, contact the Tufts SBER IRB.

Tufts SBER IRB Contact Info:
Tufts University
Social Behavioral & Educational Research
Institutional Review Board (SBER IRB)
75 Kneeland Street, 8th Floor | Boston, MA 02111
Telephone: 617-627-8804
Email: sber@tufts.edu Website:
http://viceprovost.tufts.edu/sberirb/

Tufts UNIVERSITY

Thank you for your participation!

If you are participating in the hackathon, please focus your attention back to the introductory presentation.
You may continue the survey after the main part of the hackathon, once you are instructed to do so.

→

Tufts UNIVERSITY

We would like to gather some very general information about you and your familiarity with education and technology.

What best describes your current position?

- ○ Undergraduate Student
- ○ Graduate Student
- ○ Post-Doc / Academic Staff
- ○ Faculty
- ○ Other:

How familiar are you with educational science?

- ○ I'm an expert
- ○ I'm advanced
- ○ I'm proficient
- ○ I'm somewhat familiar
- ○ I'm not familiar at all

How familiar are you with educational technology? (i.e. educational tools etc.)

- ○ I'm an expert
- ○ I'm advanced
- ○ I'm proficient
- ○ I'm somewhat familiar
- ○ I'm not familiar at all

How familiar are you with science and technology? (Engineering, Digital, Robotics, Micro-processors and other)

- ○ I'm an expert
- ○ I'm advanced
- ○ I'm proficient
- ○ I'm somewhat familiar
- ○ I'm not familiar at all

Please rate your proficiency in the following scientific fields, topics and technologies

	I'm an expert	I'm advanced	I'm proficient	I'm somewhat familiar	I'm not familiar at all
Educational Science	○	○	○	○	○
Educational Technology	○	○	○	○	○
Science & Technology	○	○	○	○	○
Computer Science	○	○	○	○	○
Digital Technology	○	○	○	○	○
Electrical Engineering	○	○	○	○	○
Mechanical Engineering	○	○	○	○	○
Robotics	○	○	○	○	○
Other:	○	○	○	○	○

What field are you active and/or interested in?

- ○ Please specify:
- ○ Please specify:
- ○ Please specify:
- ○ Please specify:
- ○ Please specify any other:

→

Tufts
UNIVERSITY

This section of question focuses on the hackathon and/or project you have developed with the aid of the Smart System Platform Development Tools and/or other capabilities and components. The question aims to gather feedback on the tools in context with your project/idea/activity.

Did you participate in the Smart System Platform hackathon?

- ○ Yes
- ○ No

Please describe what your project was about:

Which components of the Smart System Platform did you use in your project?

- ☐ Smart Module Hardware (Smart Motor, Dahal Board, etc.)
- ☐ Smart System Website (Guide, Specifications, other support material)
- ☐ Smart System Platform Github page
- ☐ Software (Networking library, etc.)
- ☐ Custom Web-IDE
- ☐ Network Management and Module Configuration Tool

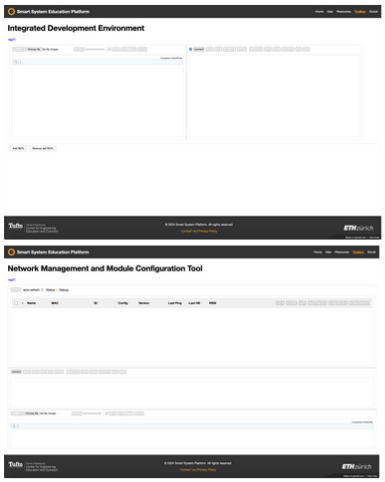
In what way did the above components aid in the development of your project?

Did you have any other ideas on what could be done using the Smart System Platform and its Networking Capabilities?

→

Tufts
UNIVERSITY

The next questions focus on the custom web IDE and the Network Management and Module Configuration Tool, referred to as the Smart System Development and Management Tools, depicted below.



Please rate the following aspects for the custom web IDE:

	Perfect	Good	Neither Good nor Bad	Bad	Very Bad
Usability	○	○	○	○	○
Accessibility	○	○	○	○	○
Performance	○	○	○	○	○
Features	○	○	○	○	○

Please rate the following aspects for the Network Management and Module Configuration Tool:

	Perfect	Good	Neither Good nor Bad	Bad	Very Bad
Usability	○	○	○	○	○
Accessibility	○	○	○	○	○
Performance	○	○	○	○	○
Features	○	○	○	○	○

Please rate the following aspects for the Website in general:

	Perfect	Good	Neither Good nor Bad	Bad	Very Bad
Usability	○	○	○	○	○
Accessibility	○	○	○	○	○
Performance	○	○	○	○	○
Features	○	○	○	○	○

I found the Smart System Development Tools to be an aid/asset in developing an activity/project:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

In what way did you find the development tools helpful or not?

The Smart System Development Tools offered all the necessary capabilities to develop my activity or project.

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

What capabilities or improvements would you like?

The Smart System Development Tools enabled me to develop my project, which I otherwise would not have been able to do as efficiently.

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

Were you able to complete and design your project without issues using the provided tools?

- ○ Yes
- ○ No

What limitations did you encounter?

→

Tufts
UNIVERSITY

This section focuses on feedback for the Smart System Platform development tools in a usability fashion, as well as for the Smart System Platform in general.

Do you understand the goal and design of the Smart System Platform?

- ○ Yes
- ○ No

The architecture and system design of the Smart System Platform as a system is logical and intuitive:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

The design and layout of the Smart System Platform website as a whole is intuitive and easy to use:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

The Smart System Platform website contains all the necessary information:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

Was there any information that you were missing?

The design and layout of the Smart System Platform Development Tool page is intuitive and easy to use:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

The Smart System Platform user guide adequately introduced you to the system and provided all the necessary information and guidance:

- ○ Fully agree
- ○ Somewhat agree
- ○ Neither agree nor disagree
- ○ Somewhat disagree
- ○ Fully disagree

Was there anything missing or that could be improved in the guide?

→

Tufts
UNIVERSITY

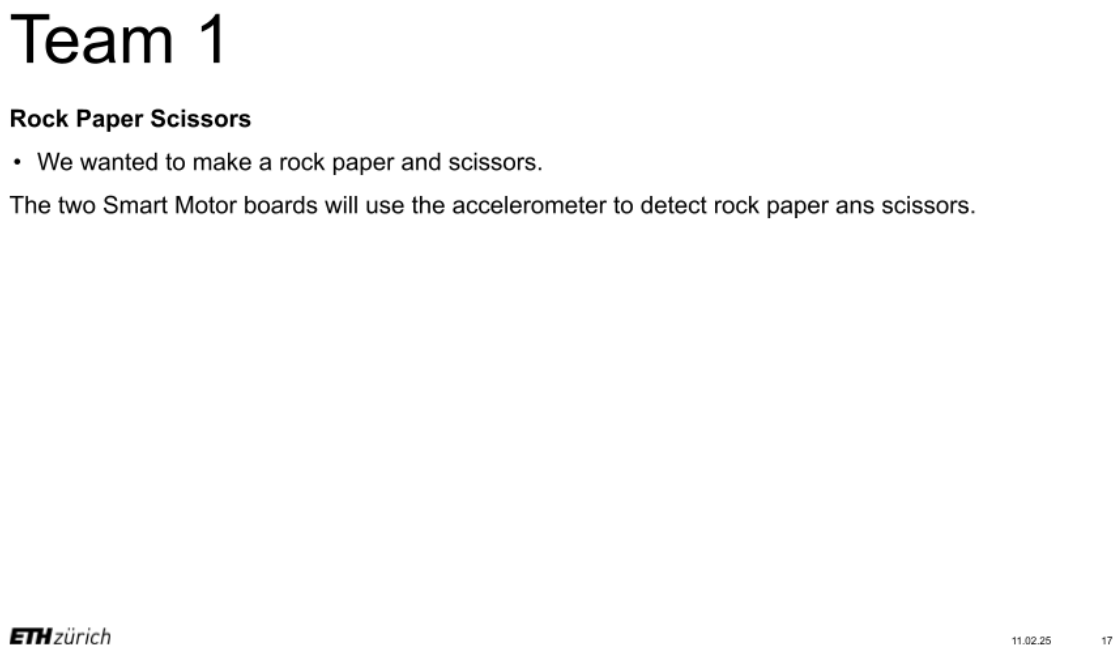
We thank you for your time spent taking this survey.
Your response has been recorded.

Results

Questionnaire Results

Due to data protection rules, the raw questionnaire response data is not included here.

Hackathon Results



Team 3

Carnival Rides

- Using the deconstructed ESP 32, we are controlling ¼ of the analog values to make a motor rotate in one direction once selecting the ESP32's button. The rest of the ¾ values should make the system rotate in the opposite direction after selecting the button. The ¼ values send "A" and the smart motor receives A as the input to then rotate in one dir., and if not sending "A", the motor will rotate in negative direcition.
- We plan to sending messages to the Smart Motors to control our carnival rides
  - In different rotations
  - In different speeds
  - Moving and Stop

ETH zürich 11.02.25 19

Team 4:

RED LIGHT
GREEN LIGHT

SPLAT:
display
- RED LIGHT
or
- GREEN LIGHT

Player 1:
- if RED LIGHT and MOVING: **THEN DIE**
- if GREEN LIGHT: ignore movement

Player 2:
- if RED LIGHT and MOVING: **THEN DIE**
- if GREEN LIGHT: ignore movement

Green Light *or* ***Red Light***
Splat displays color and broadcasts message

Wearable SM: detect IMU motion and move motor if motion detected (during RED)

ETH zürich

CODE FOR SM (pre-networking):

```
from machine import Pin, SoftI2C
import adxl345
import time
import servo
from ssd1306 import SSD1306_I2C  # Assuming an OLED screen is used

# Initialize I2C for accelerometer
i2c = SoftI2C(scl=Pin(7), sda=Pin(6))
adx = adxl345.ADXL345(i2c)

# OLED Setup (if available)
WIDTH = 128
HEIGHT = 64
i2c_display = SSD1306_I2C(WIDTH, HEIGHT, i2c)

# Motor Setup
s = servo.Servo(Pin(2))  # Servo on pin 2
s.write_angle(0)  # Start at 0 degrees

# Constants
MOVEMENT_THRESHOLD = 300  # Adjust as needed
DELAY_AFTER_MOVEMENT = 2  # Seconds to wait after detecting movement

# Simulated function to receive "red" or "green" (replace this with actual game input)
def get_light_status():
    """Simulates receiving 'red' or 'green'. Replace with actual data source."""
    time.sleep(0)  # Simulate time delay between game state changes
    return "red" if time.time() % 10 < 5 else "green"  # Toggle every 5 seconds

def is_moving():
    """Reads accelerometer values and determines if movement is detected."""
    acc_x = adx.xValue
    acc_y = adx.yValue
    acc_z = adx.zValue

    # Calculate total acceleration change
    total_acc = abs(acc_x) + abs(acc_y) + abs(acc_z)
    print(f"Accel: X={acc_x}, Y={acc_y}, Z={acc_z}, Total={total_acc}")

    return total_acc > MOVEMENT_THRESHOLD

def display_message(msg):
    """Displays a message on the OLED screen."""
    i2c_display.fill(0)
    i2c_display.text(msg, 20, 30)
    i2c_display.show()

# Game Loop
while True:
    light_status = get_light_status()
    display_message(f"{light_status.upper()} LIGHT")
    print(f"{light_status.upper()} LIGHT")

    if light_status == "red":
        print("RED LIGHT! Stop moving!")

        # Monitor for movement during red light
        start_time = time.time()
        while time.time() - start_time < 5:  # Check movement for 5 seconds
            if is_moving():
                print("Player MOVED! Eliminated.")
                display_message("ELIMINATED!")

                # Move motor from 0° to 90° to indicate elimination
                s.write_angle(90)
                time.sleep(2)  # Wait before restarting game
                break  # Exit red light phase if player loses

            time.sleep(0.1)  # Check movement every 100ms

        # Reset motor to 0° for the next round
        s.write_angle(0)
```

Networking Code (to be added)

```
import network
import espnow

# Initialize Wi-Fi in station mode (ESP-NOW requires it)
sta = network.WLAN(network.STA_IF)
sta.active(True)

# Initialize ESP-NOW
esp = espnow.ESPNow()
esp.active(True)

# Add a peer (MAC address required if sending, but not for receiving)
# esp.add_peer(b'\xFF\xFF\xFF\xFF\xFF\xFF')  # Replace with sender's MAC if needed

# Variable to store light status
light_status = "green"  # Default to green

def receive_message():
    """Receives an ESP-NOW message and updates the light status."""
    global light_status
    msg = esp.recv()  # Wait for a message
    if msg:
        mac, message = msg  # ESP-NOW returns MAC, message
        message = message.decode("utf-8")  # Convert bytes to string
        print(f"Received: {message}")

        if message in ["red", "green"]:
            light_status = message  # Update game status

# Example loop to keep checking for messages
while True:
    receive_message()
```

ETH zürich 11.02.25 20