Multi-Servo Dynamixel Library

Taksch A. Dube

M. Arnett

Nathan C. Moder

Capstone Project

Kent State University

May 2021

Table of Contents

Project Proposal

Description

Servo motors provide a more accurate and programmable way to control the joints of a robot. A simple implementation of a robotic manipulator relies on a series of connected servos that must simultaneously receive commands and provide feedback to the controller. To accomplish this, a control library must be implemented to simplify the programming of the control of such robots and issues of latency, data throughput, and scalability (number of servos), must be addressed.

A current fork of a servo control library is available from the ATR Lab. The library currently allows multi-servo communication, but requires further work towards scaling above 10 servos. Further work is required towards developing a user interface to manage the servos.

Development Scope

- Architecting and modularising the library
- Optimising messages/data sent in parallel
- Designing and implementing an interface that allows for testing of servos
- Unit testing, and integration testing

End Goals and Benefits

The end goal of this project is to fully open-source our implementation, allowing researchers to leverage our library/application and or its core components. Furthermore, students will have the opportunity to write and publish an academic research paper as leading authors. Other tangible benefits include workshops and training on software design, architecture, and development. As well as training on the technologies listed below (as needed).

Risk

One of the risks involved is with how niche this type of product would be. Mainly regarding any sort of troubleshooting and QA testing that would be needed. Though, one of these solutions would be to spend more time fully comprehending what the technology does; and as well with reaching out to members of the ATR Lab for further resources.

Prospective Technologies

- Robot Operating System (ROS)
- ElectronJS
- SocketIO
- React JS
- Material UI
- MongoDB
- typicode/lowdb

Project Timeline

Jan 21: Beginning of Capstone and team formation.

Jan 25: The client, ATR Lab, pitches the 'Multi-Servo Dynamixel Library Project' and the team subsequently accepts it..

Jan 26: A objective state for the project was drafted and the first round of elicitation was conducted.

Jan 29: The second round of elicitations were conducted and finalised.

Feb 2: The project proposal was finalised and submitted.

Feb 5: Two sets of three servo motors were acquired from the ATR Lab for testing purposes.

Feb 6: First presentation meeting with the stakeholders covering the wireframe of the Graphical User Interface.

Feb 11: Established the Gantt chart, Project Schedule, and drafted the Elicitation details.

Feb 12-17: Work was done towards drafting the functional and non-functional requirements. Require meeting schedules were established and documentation was conducted more vigorously.

Feb 18: The functional and non-functional requirements were submitted.

Feb 19-22: The team started work on system architecture and software design.

Feb 23-26: The system architecture and software design were finalised a week prior to the submission date.

Mar 3: First in-class presentation on March 2, 2021 covering the project proposal, elicitation, functional and non-functional requirements, system architecture, and software design.

Mar 4: Architecture and software design was submitted. The software was named MINT.Box and accompanying back-end library was named MINT.Patch.

Mar 5-Apr 3: Development period.

Mar 25: Second in-class presentation on March 25, 2021 covering the first demo of MINT.Box and the design decisions for MINT.Patch.

Mar 31 or Apr 1: Development period.

Apr 4: Rough draft of software and full documentation was submitted.

Apr 5-Apr 20: Revised documentation and software was submitted. Final presentation were prepared.

Apr 27-May 4: Final presentations was presented.

Project Elicitation

I. What is involved in the development of the library?

The Advanced Tele-robotics (ATR) Lab at Kent State University currently possesses a fork of the Dynamixel library that allows the use of different models of the Dynamixel servos, concurrently. However, the current library is slow and limited in the number of servo motors that it can accommodate, hence the need for optimisation and improvement.

II. What are the specific optimisations and improvements?

Specifically, the group's job is two-fold. Primarily, we would like to optimise the library to remove the time-lag that the current version suffers from and also improve it so that it can handle more than ten servos at once. Secondly, our job is to create a Graphical User Interface (GUI) for the library.

III. What does the GUI have to do?

The GUI's primary function is to create an environment to facilitate the testing of the library's functionality, as well as provide a visual interface to monitor the servos simultaneously.

IV. What does the GUI test?

The GUI tests the rotation, initial position, final position, status, and temperature of the servo motors. It also provides an interface for inputting commands to the servo motors to test the library.

V. What tools are we going to use for prototyping the architecture of the GUI?

We are using industry tools for wire-framing our GUI designs. We are mainly concentrating on utilising Figma for this purpose.

VI. What programming languages and libraries will the GUI be coded in?

After much contemplation, we're planning on using Javascript + HTML/CSS for the GUI by leveraging the React.js and Electron.js libraries to allow us to create a desktop application.

VII. How will we interact with the C++/Python library using a JavaScript GUI?

For the purposes of binding JavaScript to C++, we will be leveraging Node.js which is the parent library to both Electron.js and React.js. We can also utilise SpiderMonkey by Mozilla for JavaScript to C/C++ binding and Python binding. The library with then bind the C/C++ and Python code to ROS, to operate the motors.

VIII. What are the design objectives surrounding the GUI?

The primary objective of the GUI is to provide an environment to facilitate testing and monitoring of the code in a modular fashion. Our objective is to accomplish our primary goals while making it visually intuitive for new users.

IX. Who are the end users of the GUI?

The GUI is primarily meant for robotics researchers who are fluent with the existing library. However, we would like to make it accessible to new learners of robotics as well, therefore once the initial requirements are met, we will be focusing our attention to increasing visual readability of the interface and simplifying for ease of learning and usage.

X. Does the library need to support an infinite numbers of servos concurrently?

While the goal of the library is to support as many servos as possible, due to the limited memory of each servo motor there is a limit to the amount of motors that can work simultaneously. The current library can run 8-10 motors with a high degree of lag; hence, our primary objective to support up to 10 servos concurrently with minimal time-lag. Then, we will try to improve the servo capacity as much as we can.

XI. What models of Dynamixel servos does the library intend to support?

MX-64 and H54-200-S500-R are the models that we are working with for the development of the library.

XII. Will the library support future releases by Dynamixel?

Yes, the library will keep being updated with each major release by ROBOTIS. Our goal is to extend support for all the Dynamixel releases.

XIII. How will support for these future releases be implemented?

Currently, we will be manually writing code for each of the major servos with differing standards. This will need to be done for each new release that follows a newer standard of encoding registers on the servo motors. Our goal in the future would be to come back and polish the code so that it can be generalised better for changing standards.

XIV. Will the library be compatible with non-Dynamixel servo motors?

For the moment there are no plans to support non-Dynamixel servos, however it would be an ultimate goal to provide direct support or providing a paradigm for introducing further support.

XV. Should developers be able to add support for non-Dynamixel servos?

Yes, it would be a primary objective to make the library extremely accessible to third party.

XVI. Are we going to create a platform that aids such open-source development?

If we are able to satisfy the primary objectives of the project, then we will certainly add support for future improvement and expansion of the library. This will ensure the longevity of our contribution to the field of robotics.

Functional and Non-Functional Requirements

Functional Requirements:

I. The software needs to be able to add and remove servo motors from the list of actively displayed servo motors.

II. The software needs to interact with the servo motors in a controlled and comprehensive way.

III. The software needs to display the temperature, current, angle, speed, and I/O state of the servo motor(s) that it is currently interacting with.

IV. The software needs to be able to interact with more than ten concurrent servo motors at one moment.

Non-Functional Requirements:

I. The software should operate with all the past and future releases of Dynamixel servo motor models by Robotis Co.

II. The software needs to be optimised to run with the least amount of lag possible, while accommodating the maximum number of concurrent servo motors.

Context Diagram

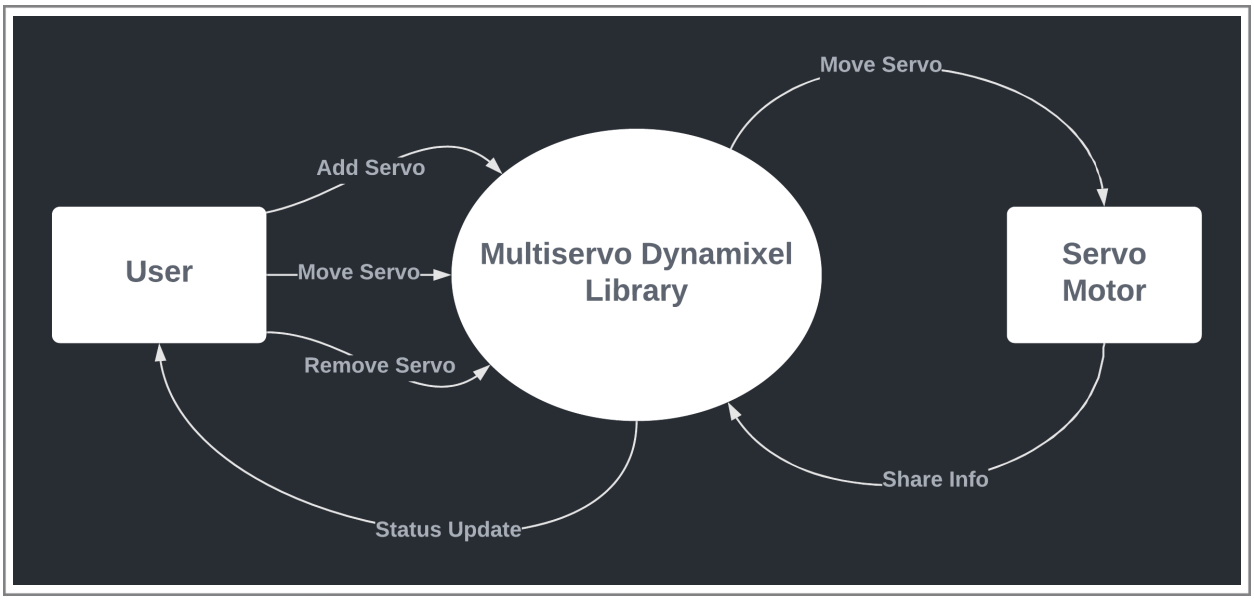

Figure 12.1 - Context Diagram

Description

MINTBOX will act as an intermediary between a user and a number of servo motors. In Fig. 12.1, the broad needs of this interaction are listed, as well as which parts of the system they interact with.

Use-Case Diagrams

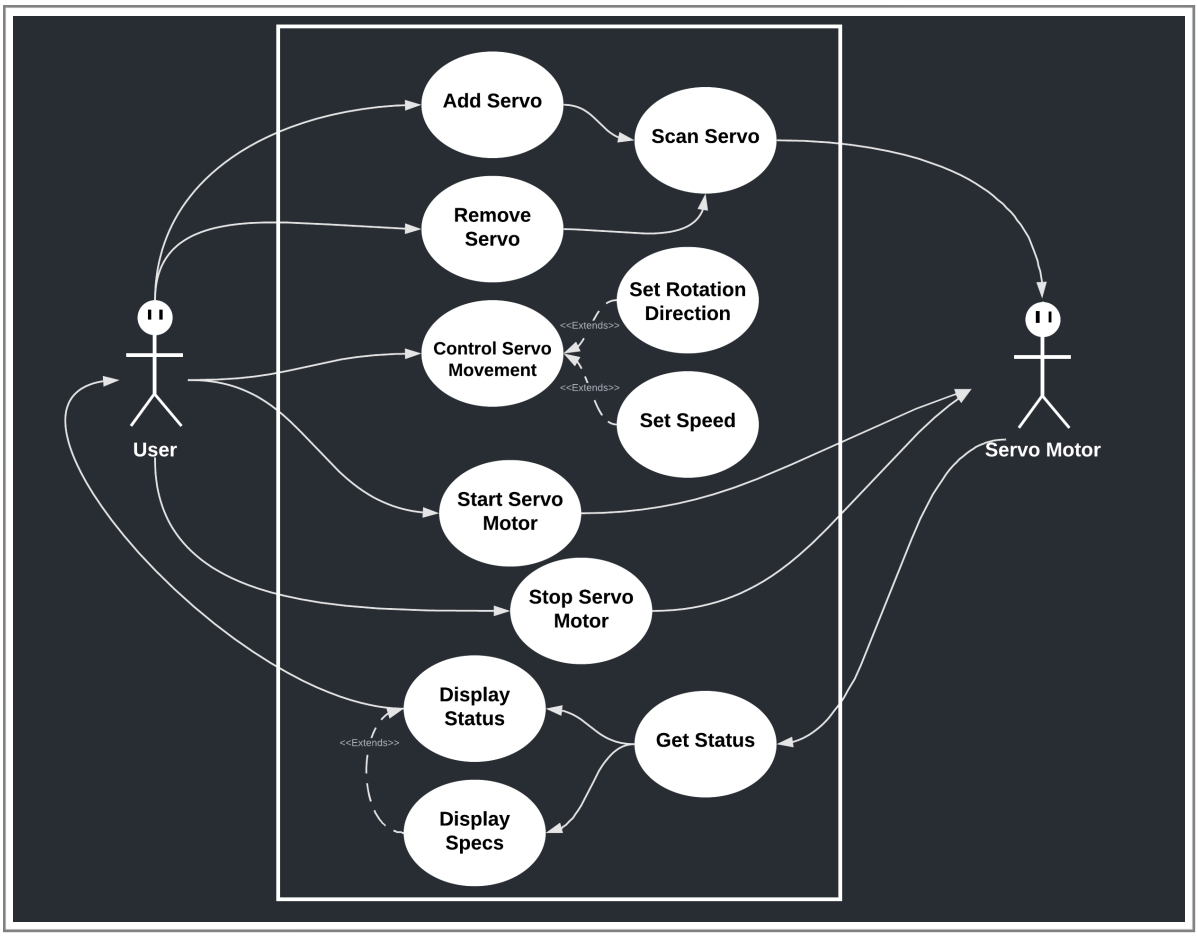

Figure 13.1 - Full Use Case Diagram

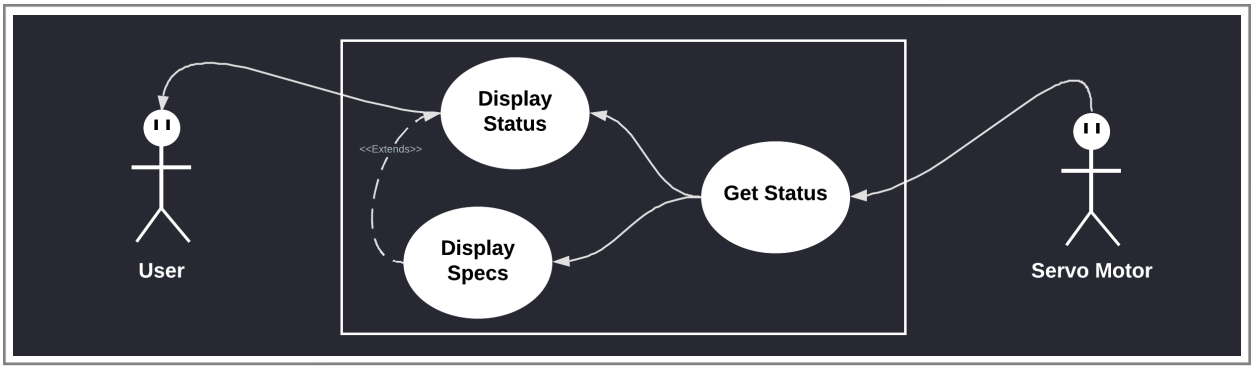

Figure 13.2 - Servo Diagnosis

Description

Fig. 13.2 describes the diagnostic function of MINTBOX. Since this is a testing environment meant to be used by robotics engineers, detailed information about the status and capabilities of the servo motors must be communicated. The system will frequently request updates through the library, displaying the information returned from the servos.

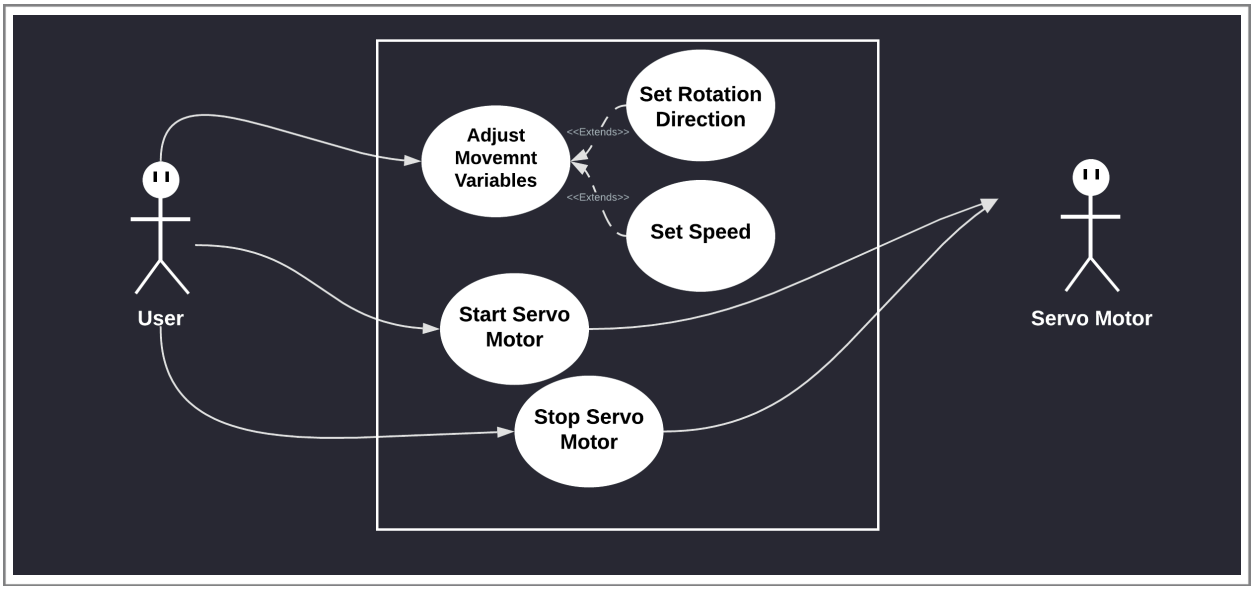

Figure 14.1 - Servo Controller

Fig. 14.1 describes the role of MINTBOX as a controller of the servos. In order to have a full range of movement available, the user will be able to set the direction of rotation and the speed of rotation, separately, at any time. There will also be a button which can start the movement of the servo, and a button which stops that movement and puts the servo in an idle state. Both buttons will have corresponding operations inside the library.

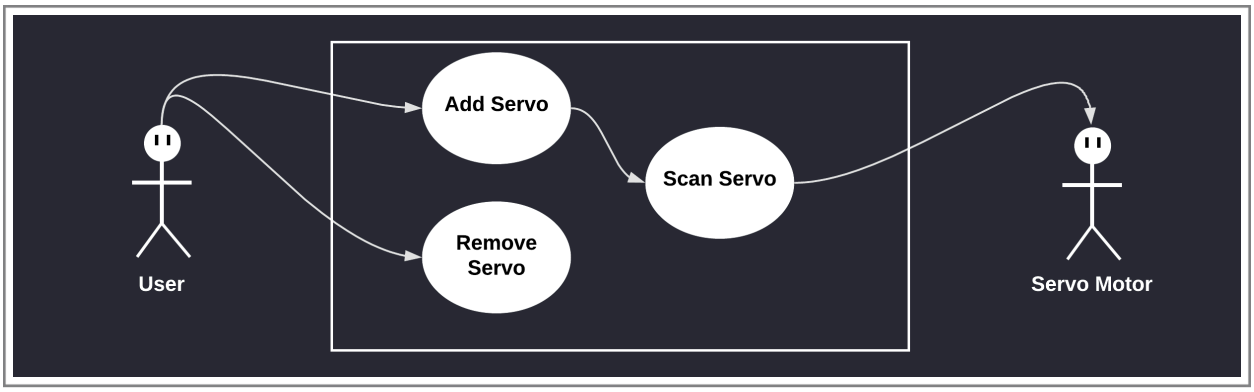

Figure 15.1 - Servo Manager

Fig. 15.1 describes the need to add and remove servos from the User's control. Within the system, there will be a function to scan all servos which are attached to the computer, which will be used to collect a list for the User. The User will also have a standalone ability to remove an individual servo from the list.

System and Software Architecture

System Architecture

Figure 16.1 - System Architecture

Description

The User will interact with a GUI; one containing buttons, one type of text box, and a display of many types of information. As the User interacts with this software, it will send commands to the Drivers located on the Servo Motors. This will serve every needed purpose: gathering information and controlling Servo motion.

Software Architecture

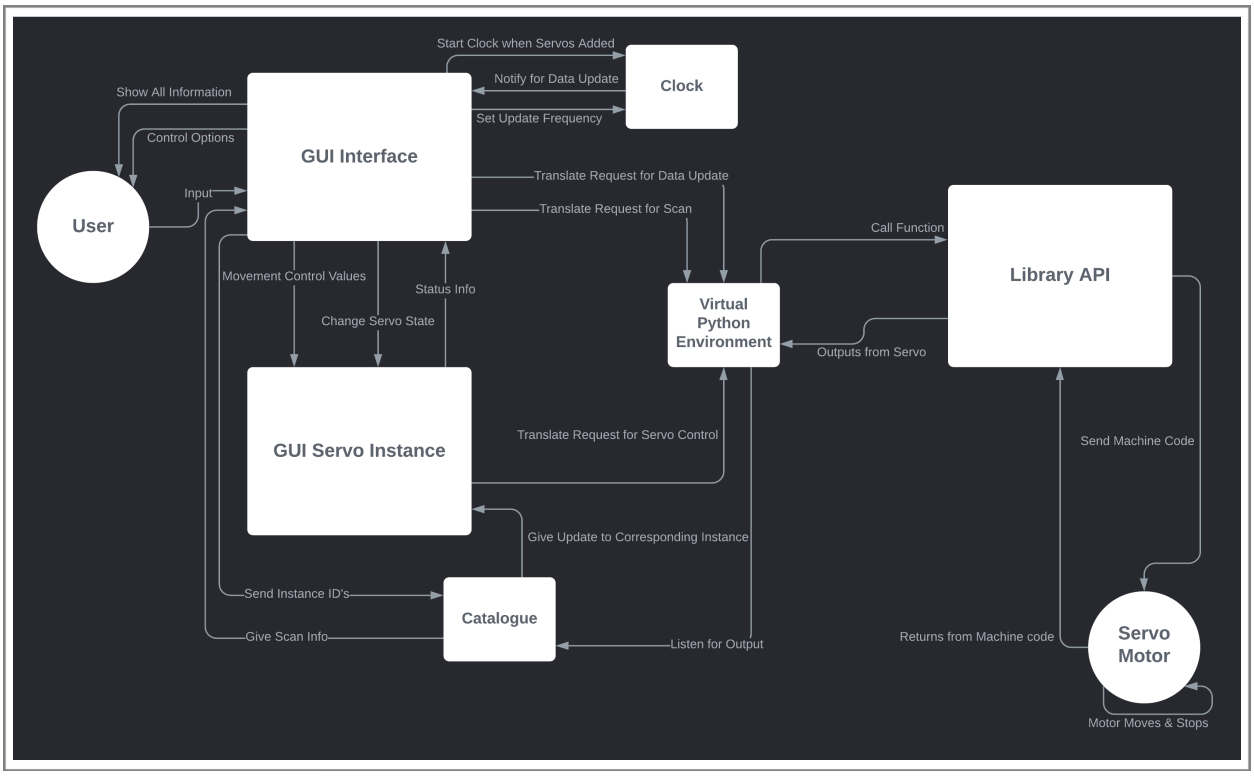

Figure 17.1 - MINT.Box Architecture

Description

The program begins with a blank UI; the only buttons available will be internal options and the “scan” button. This will prompt the GUI to send a python command to the Virtual Python Environment (VPE). The python environment includes the library script, so this will run commands in the library to check which servos are connected to the computer. This information will be output to the Python Environment. The Catalogue, which is reading every line of the Python Environment, will process this information and, seeing that it is new servo info, will send it to the GUI. The GUI will then make Instances with each servo’s ID, and these instances will track everything about the Servo Motor to which they correspond. At the same time, the GUI will start the Clock, which will indicate the regular

increments to get new data from the active servos. This will prompt the GUI to send a python command including all the active servos, which will be processed by the library. The returns for each servo will be sent to the Catalogue, where they will be distributed to the instance objects to be efficiently stored.

At any time, the user should be able to set a direction or speed for servo movement. This is done in the GUI, and the new info is sent to the GUI Instance (where it is verified by the limits of the specific servo). Once verified, it will be displayed in the GUI.

Another always-available functionality should be the starting and stopping of servo motion. This will be performed by pressing a button located on every Instance, which will cause a python command to be fabricated with the info contained in the instance. This command will go through the Python Environment to the Library, effecting the motion of the servo.

Graphical User Interface Design

Initial Screen

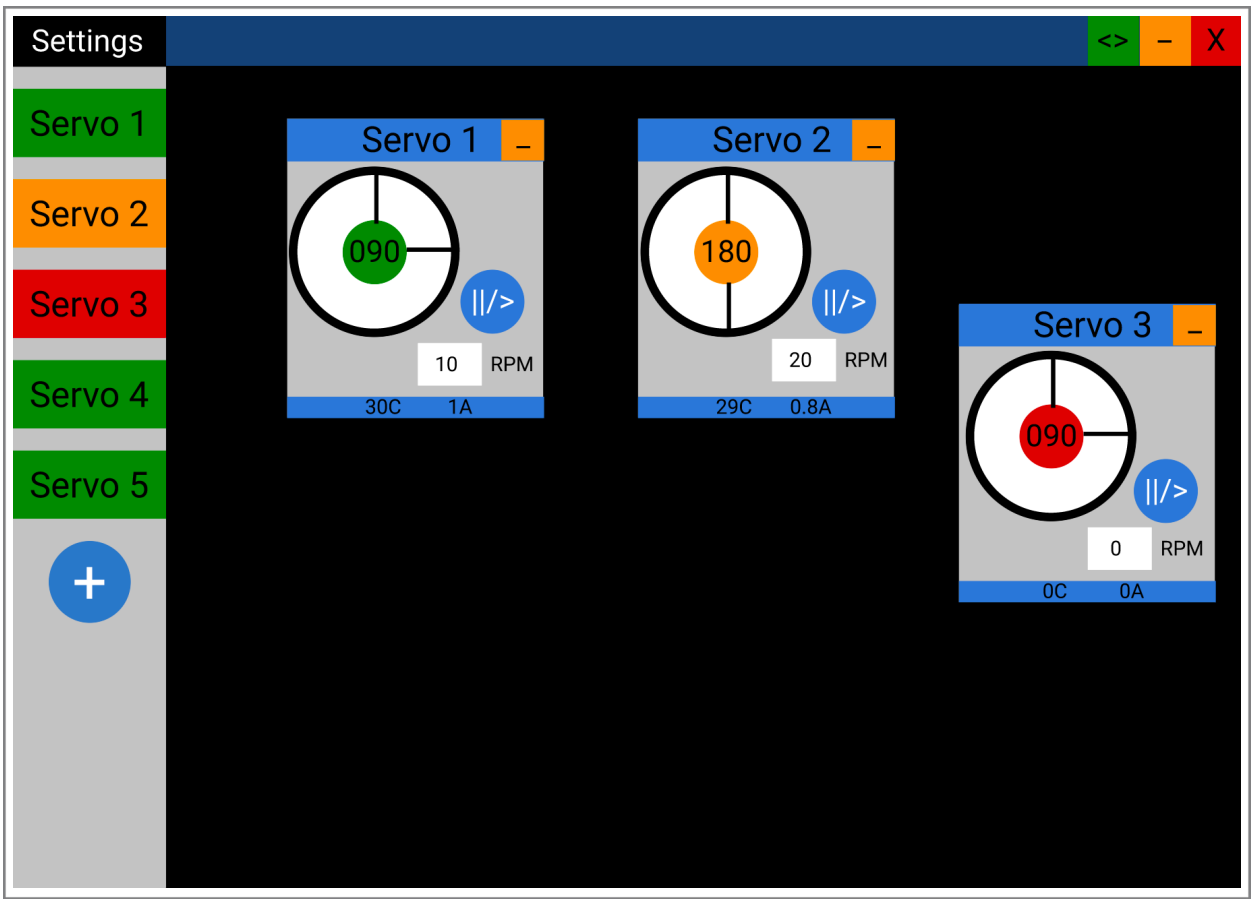

Figure 19.1 - MINT.Box Example Screen Wireframe

The example screen from Fig. 19.1 shows how MINT.Box will look in action. The grey vertical bar is the “Catalogue” which holds the complete list of servo motors connected to the software. They are listed in the order they are connected to the system and the colours indicate their current state. Green means currently moving, orange represents an idle motor that is switched on, and red indicated a powered down motor. The motors themselves can be dragged around the canvas, and more motors can be added by pressing the “+” button in the Catalogue.

Settings Functionality

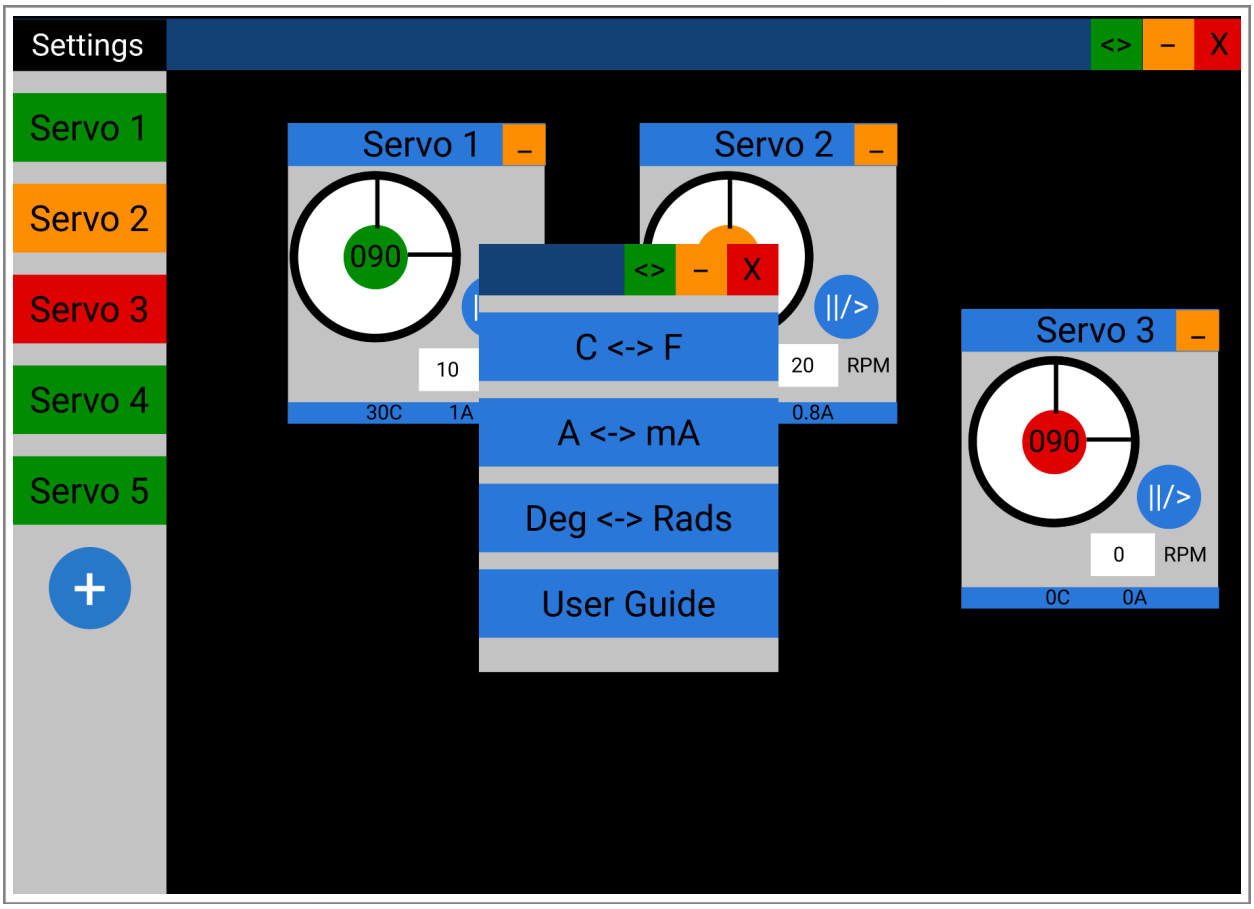

Figure 20.1 - MINT.Box Settings Wireframe

The “Settings” button can be pressed to bring into view the above menu. For the prototype implementation, the setting allows the ability to pull up the user guide for software usage information, and the ability to change units for the Servo’s position, temperature, and current readings.

Dragging Functionality

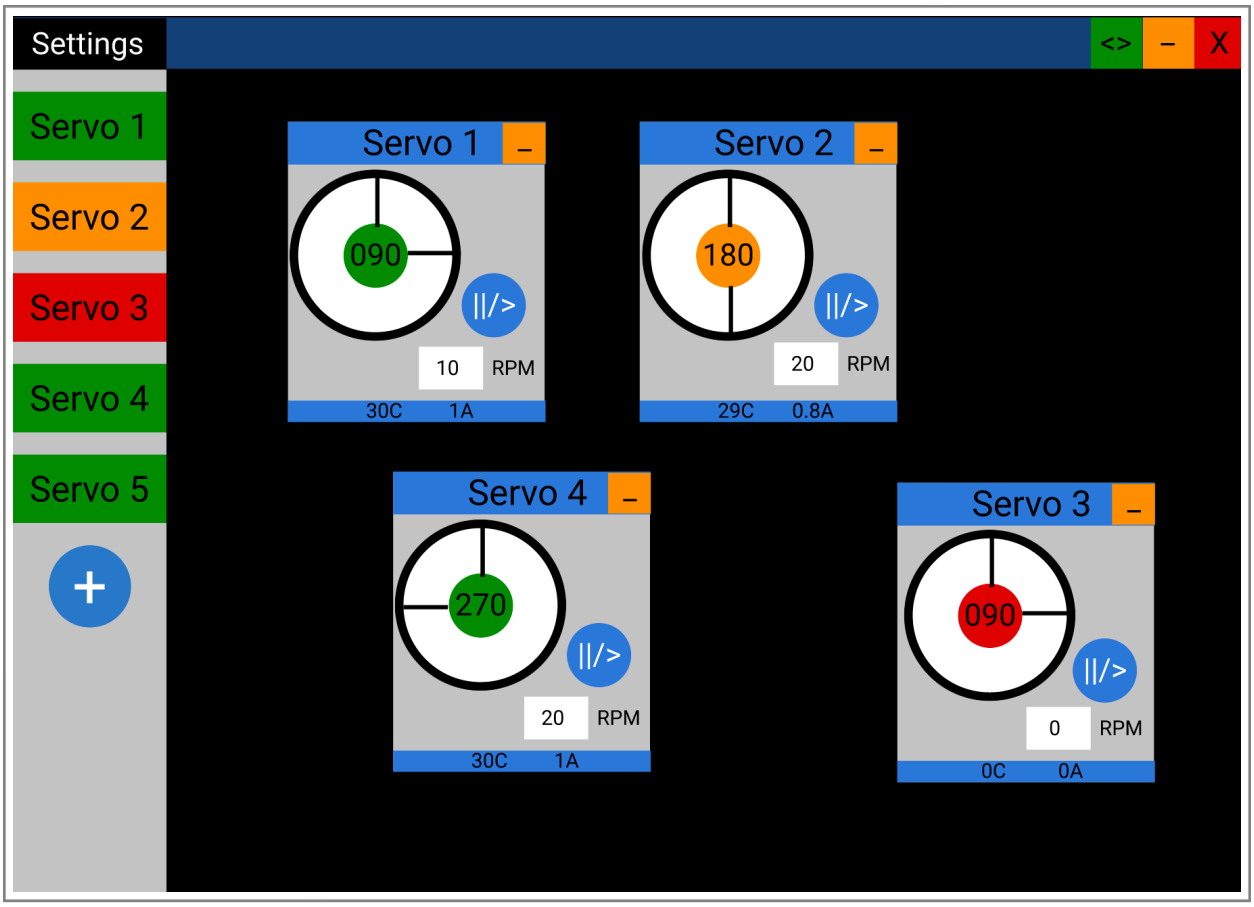

Figure 21.1 - - MINT.Box Dragging Functionality Wireframe

The individual visualisation units of each servo can be dragged around on the “Canvas”. The goal is to make the Canvas infinite with the ability to zoom in and out as the user pleases, as seen above with the repositioning of “Servo 3”. New servos can be dragged onto the Canvas as seen above with the introduction of “Servo 4” to the Canvas.

Right-Click Functionality for Catalogue Elements

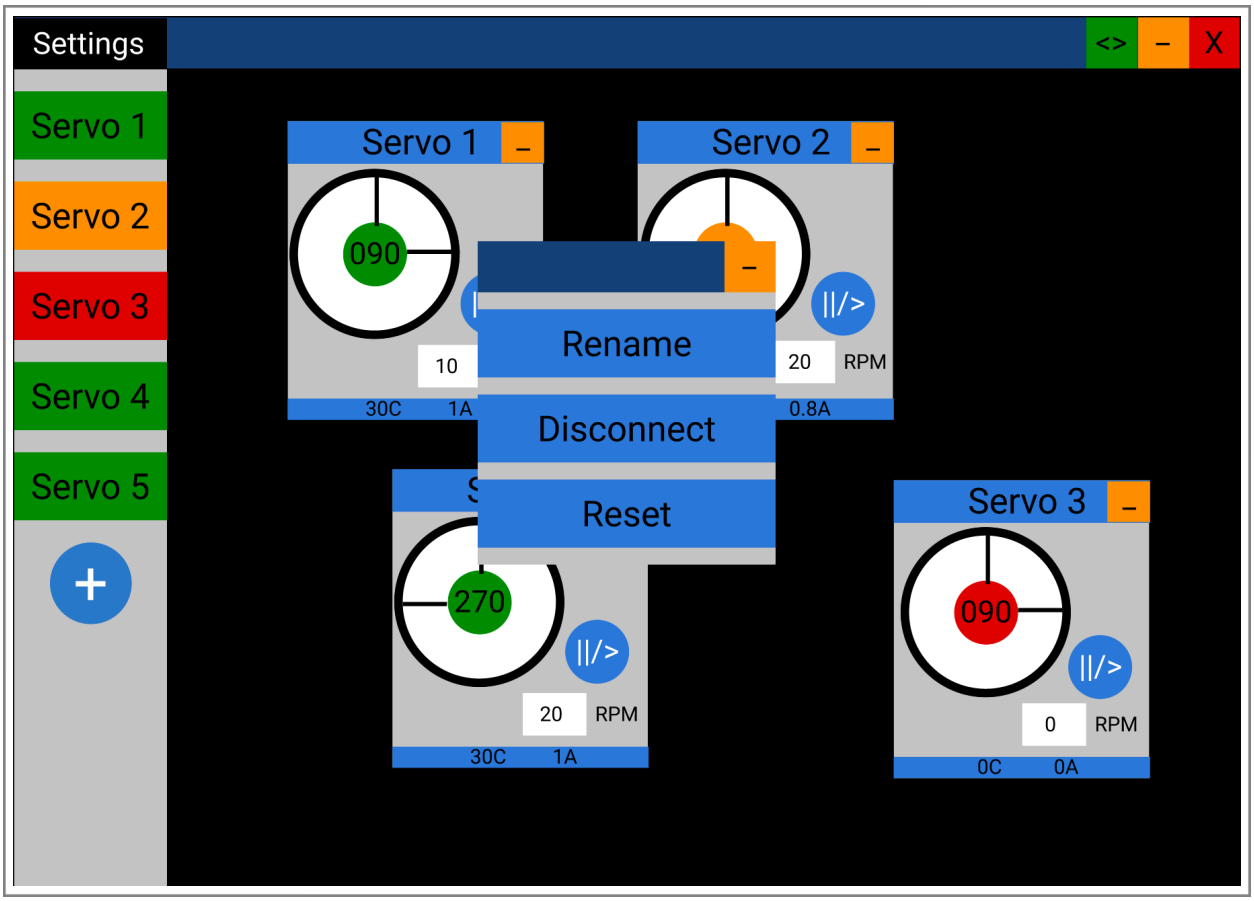

Figure 22.1 - MINT.Box Options Wireframe

Each servo in the Catalogue can be dragged to introduce to the Canvas, left-clicked to do the same, and right-clicked to introduce the above menu that provides options to “Rename” the specific servo - to allow for better readability, “Disconnect” the servo - in order to remove it from the current list of connected components to the system, and “Reset” the servo - which resets the current, temperature, angle, speed, and state of the servo; allowing for better troubleshooting and testing.

Reset Functionality

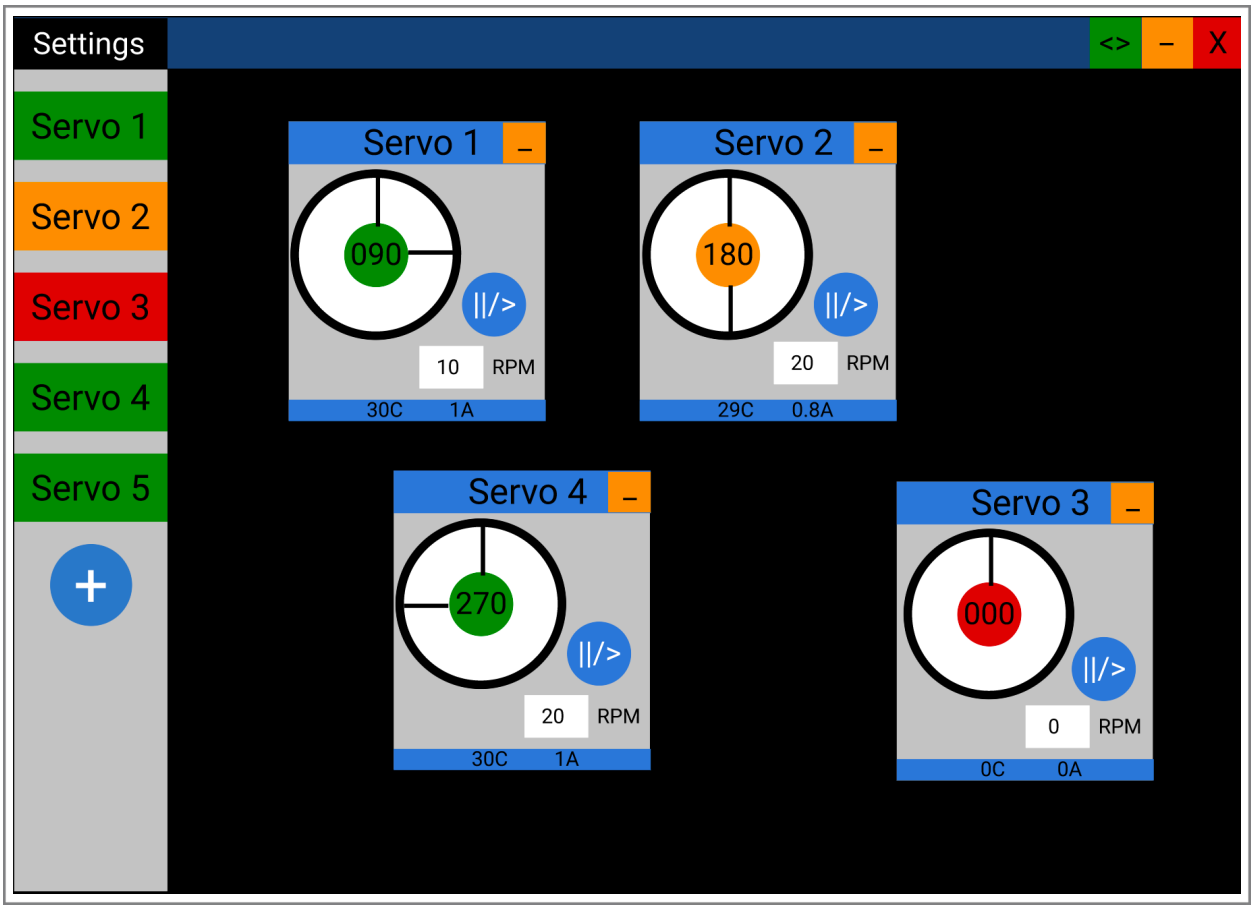

Figure 23.1 - MINT.Box Reset Functionality Wireframe

The “Reset” functionality allows troubleshooting and testing capabilities for the library by enabling the ability to reset the angle of the servo, place it in an “off” state, reset the current travelling through the individual servo and its speed, and wait for the servo temperature to cool down completely.

Disconnect Functionality

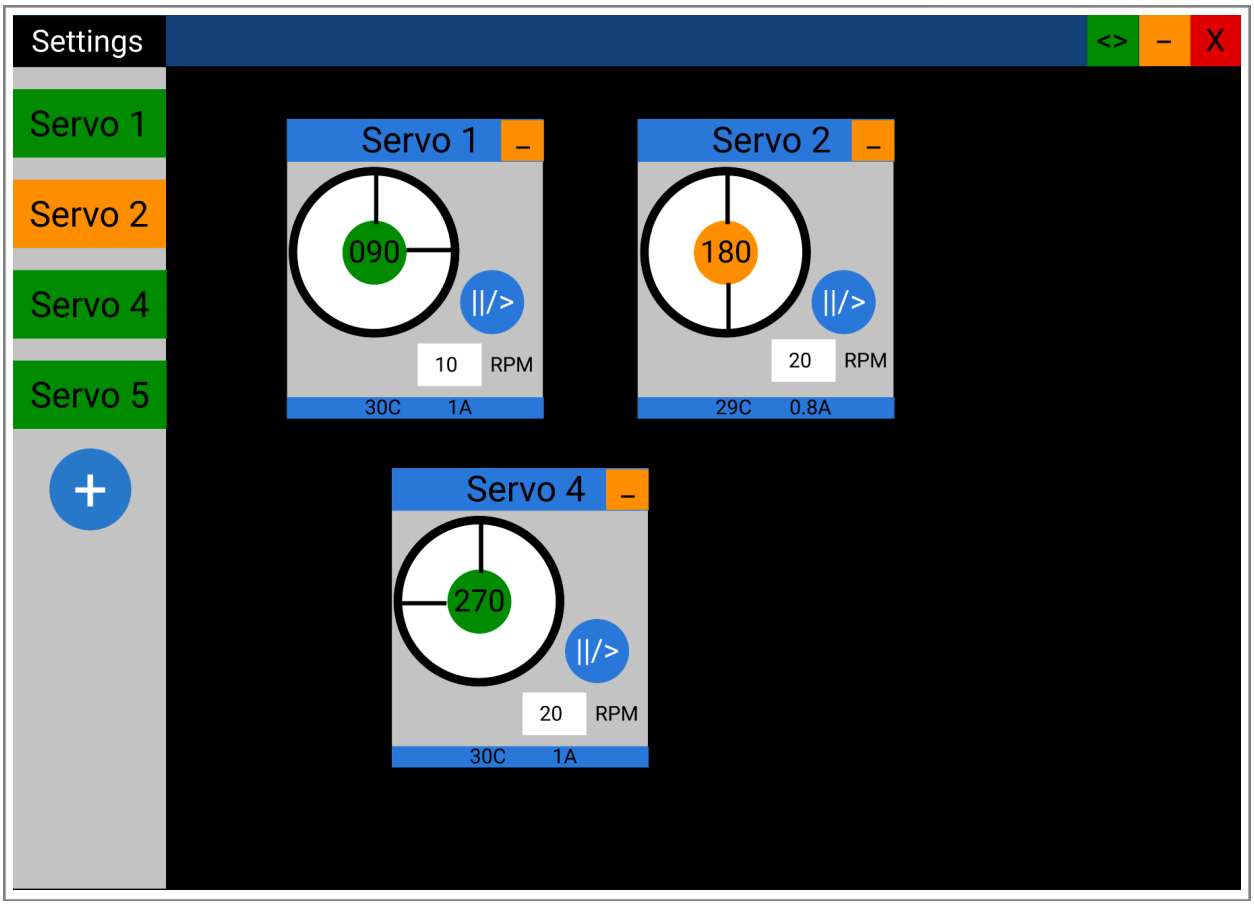

Figure 24.1 - MINT.Box Disconnect Functionality Wireframe

As for the "Disconnect" functionality, it can be seen in Fig. 24.1. "Servo 3" has been completely removed from the environment, hence its absence not just from the Canvas but the Catalogue as well.

Scan Functionality

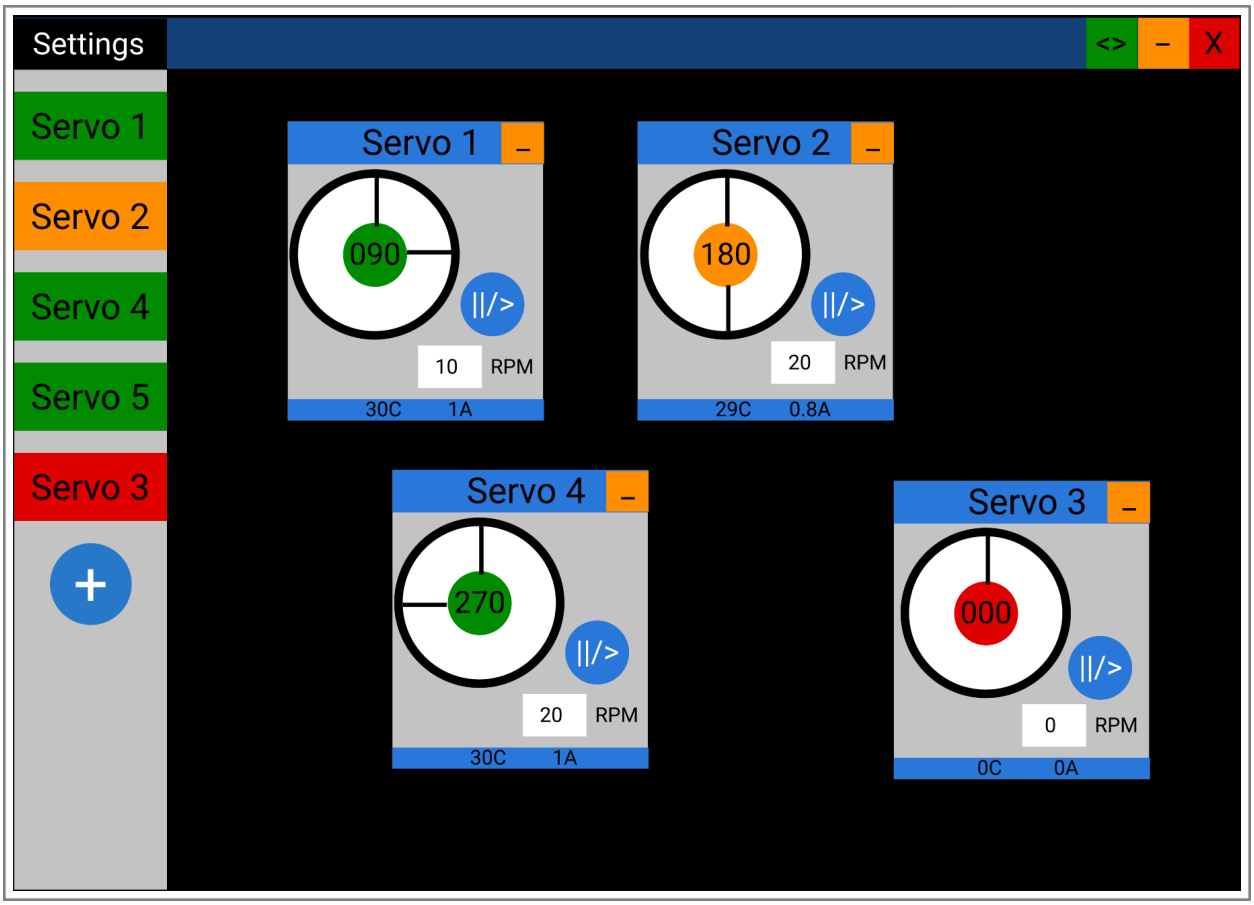

Figure 25.1 - MINT.Box Scan Functionality Wireframe

Fig. 25.1 shows the visualisation of the effects of the “Scan” functionality. The previously disconnected “Servo 3” has been re-added. This was done by pressing the “+” button or the “Scan” button, which adds all connected motors to the Catalogue. The added servos are named numerically by default, but this name can be changed.

Alpha Feature Set	Beta Feature Set
Name	Name
Initial Angle	Initial Angle
Current Angle	Current Angle
Current Speed	Goal Angle
Electrical Current	Current Speed
Current Temperature	Max Speed Limit
	Current Torque
	Max Torque Limit
	Current Electrical Current
	Current Electrical Voltage
	Current Temperature

Figure 26.1 - Attribute List

Library Design: MINT.Patch

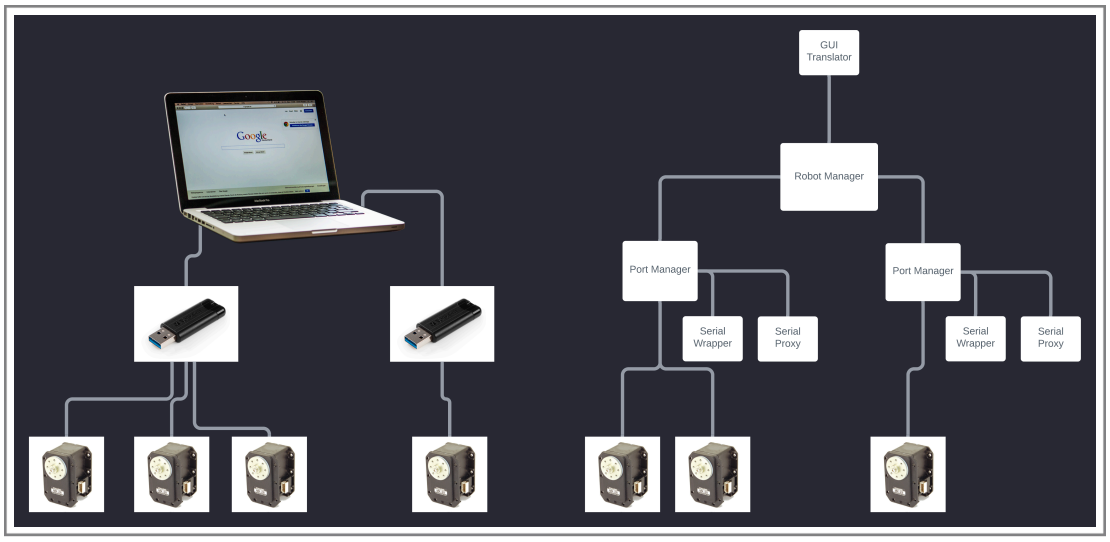

Figure 27.1 - Physical System Diagram Diagram

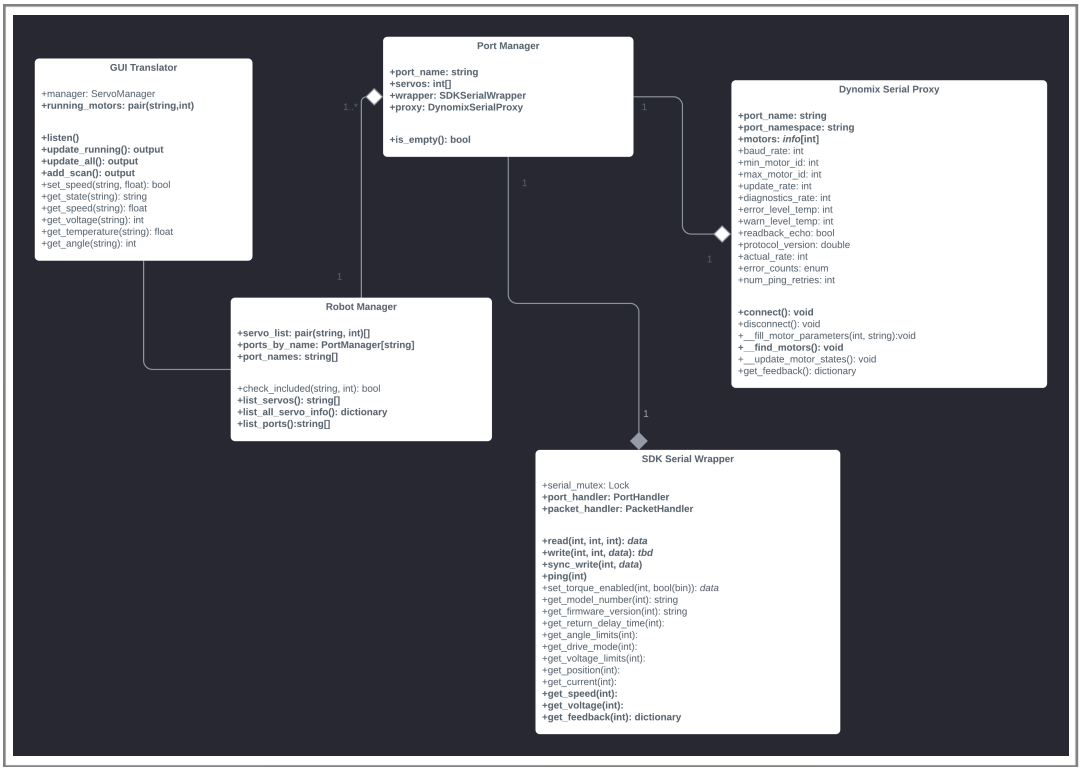

Figure 27.2 - MINT.Patch Class Diagram

The architecture of the library aims to efficiently connect a single entity with a number of servo motors. Because of this, we decided to create a tree that mirrors the organisation of the physical items, demonstrated in Fig. 27.1. On the left, any number of the USB ports on the computer may have a special device attached which connects it to several motors. The motors on one specific port must have the same type of cord connecting them, so it is not feasible to host every motor for the system on one port. On the right, we have created a similar functionality in the code. There are Port Managers which hold not only a list of every motor attached to a specific port, but also the Serial Proxy and Serial Wrapper, tools of our own design for interacting with the motors. These tools create handlers based on the port they are attached to, so they need to be unique to the Port Manager. A list of all the Port Managers is held by the Robot Manager, which is expanded on in Fig. 27.2. Using this list, the Robot Manager also constructs a list of the servos attached to the system. Utilising these lists, a GUI Translator can access the Serial Wrapper and Proxy while implementing a communication protocol we decide on with MINT.Box.