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

The 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. 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.

Development Scope

- Archeting and modularizing the library
- Optimizing 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: The team for the project was assembled.

Jan 25: The project idea, proposed by the ATR lab, was accepted.

Jan 26: A modified version of the ATR lab's problem statement was produced.

Jan 29: The group met with the stakeholder and performed the primary elicitation.

Feb 2: The project proposal was finalised and submitted.

Feb 5: All necessary supplies were obtained from the ATR lab.

Feb 6: Meeting with the stakeholder, initial functional requirement communication began.

Feb 11: First edition of the Gantt chart was created. First timeline was created. Elicitations were documented. All three were submitted.

Feb 12-17: There will be development of functional requirements through use case diagrams and context diagrams. Upon their completion the architecture for the two parts of the system will begin.

Feb 18: The functional requirements will be submitted.

Feb 19-22: Intense work will occur on architecture design and presentation planning.

Feb 23 or 25: In-class presentation will occur.

Feb 26-Mar 3: Work will continue on architecture. Actual coding will likely begin.

Mar 4: Architecture and software design will be submitted.

Mar 5-Apr 3: The development of project and planning of presentations will last a month.

Mar 31 or Apr 1: Progress report presentation will occur.

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

Apr 5-Apr 20: Revise software, polish documentation, and finalise presentation.

Apr 27-May 4: Final presentations occur.

Project Elicitation

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

The Advanced Telerobotics (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.

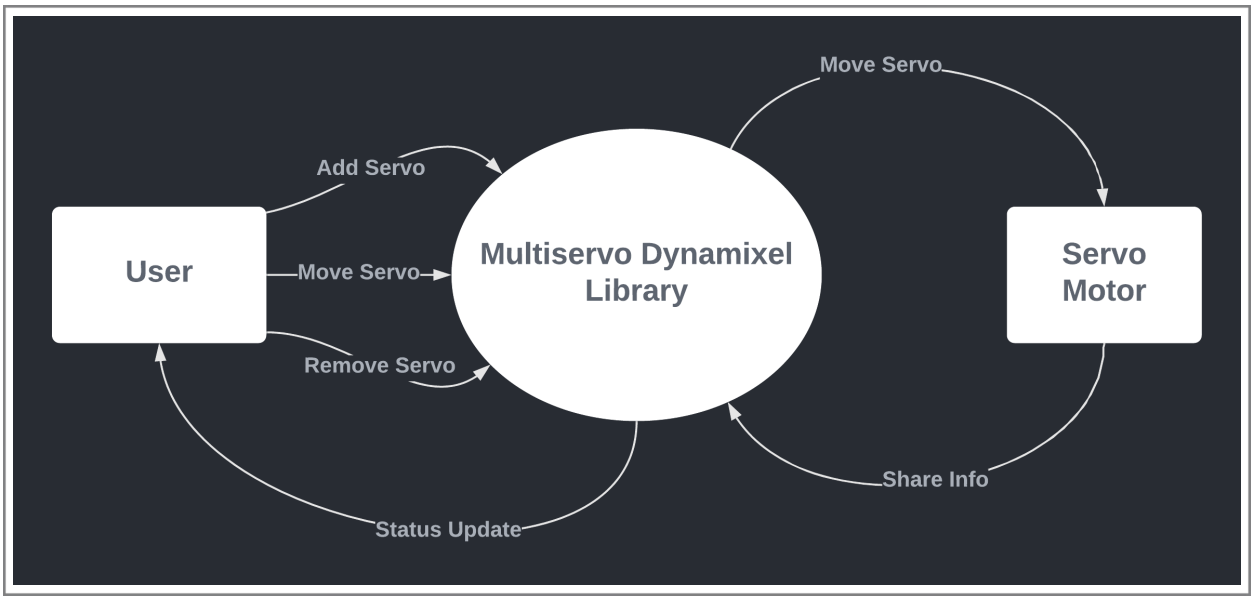
IV. The software needs to be able to interact with more than ten concurrent servo motors at one moment.

Non-Functional Requirements:

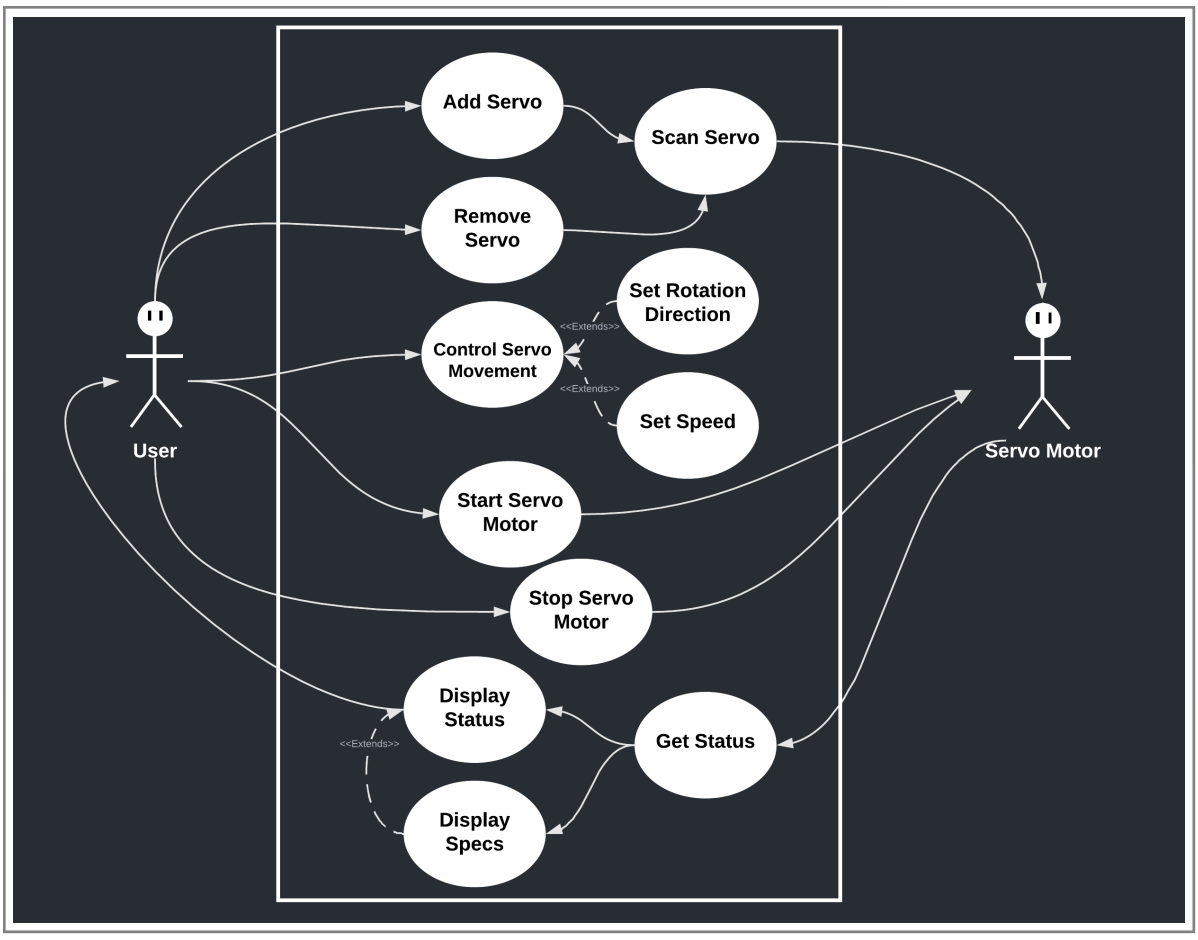
I. The software should operate with all the past and future releases of Dynamixel servo motor models by Robotis Co.

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

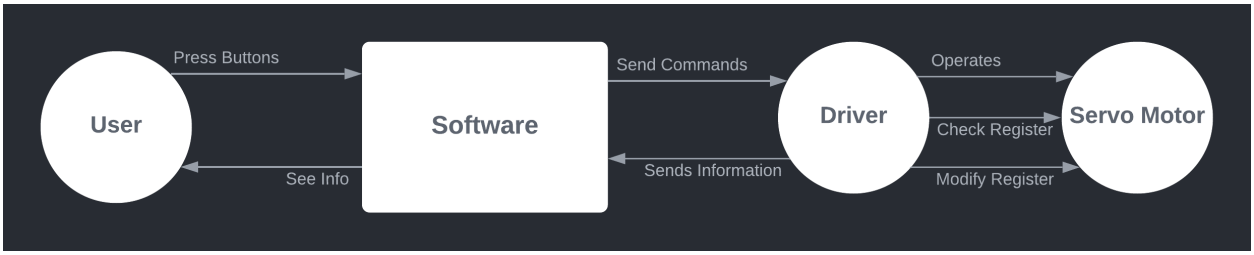


Use-Case Diagram



System and Software Architecture

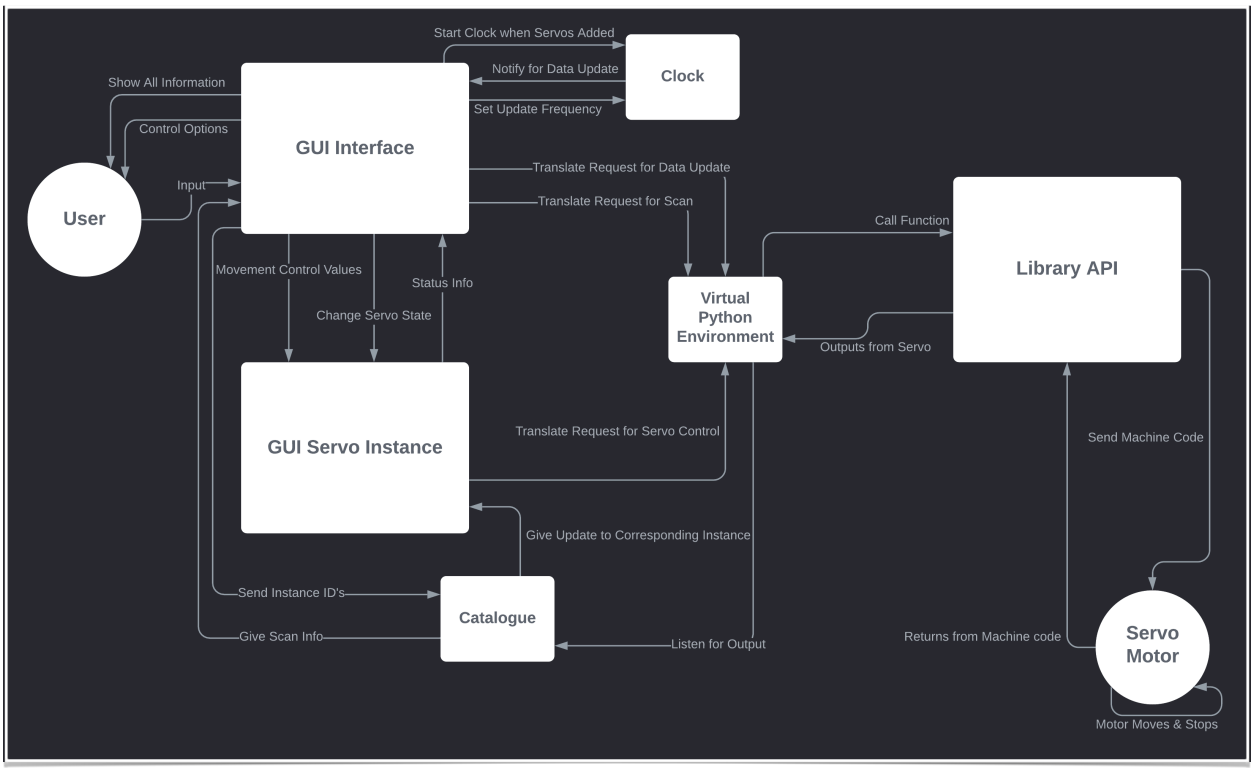
System Architecture



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



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

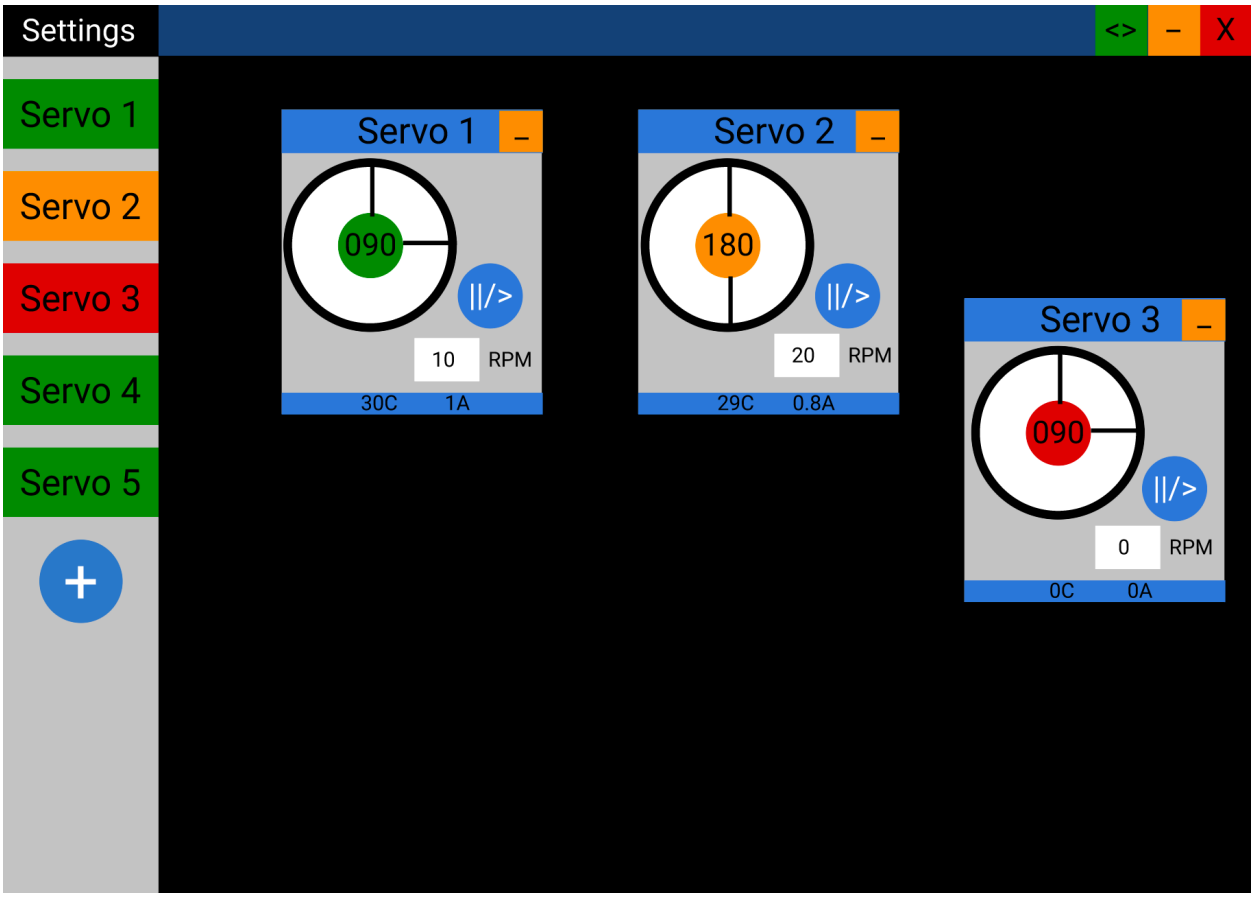
python command including all the active servos, which will be processed by the library. The returns for each servo will be sent to the Catalogue, where they will be distributed to the instance objects to be efficiently stored.

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.

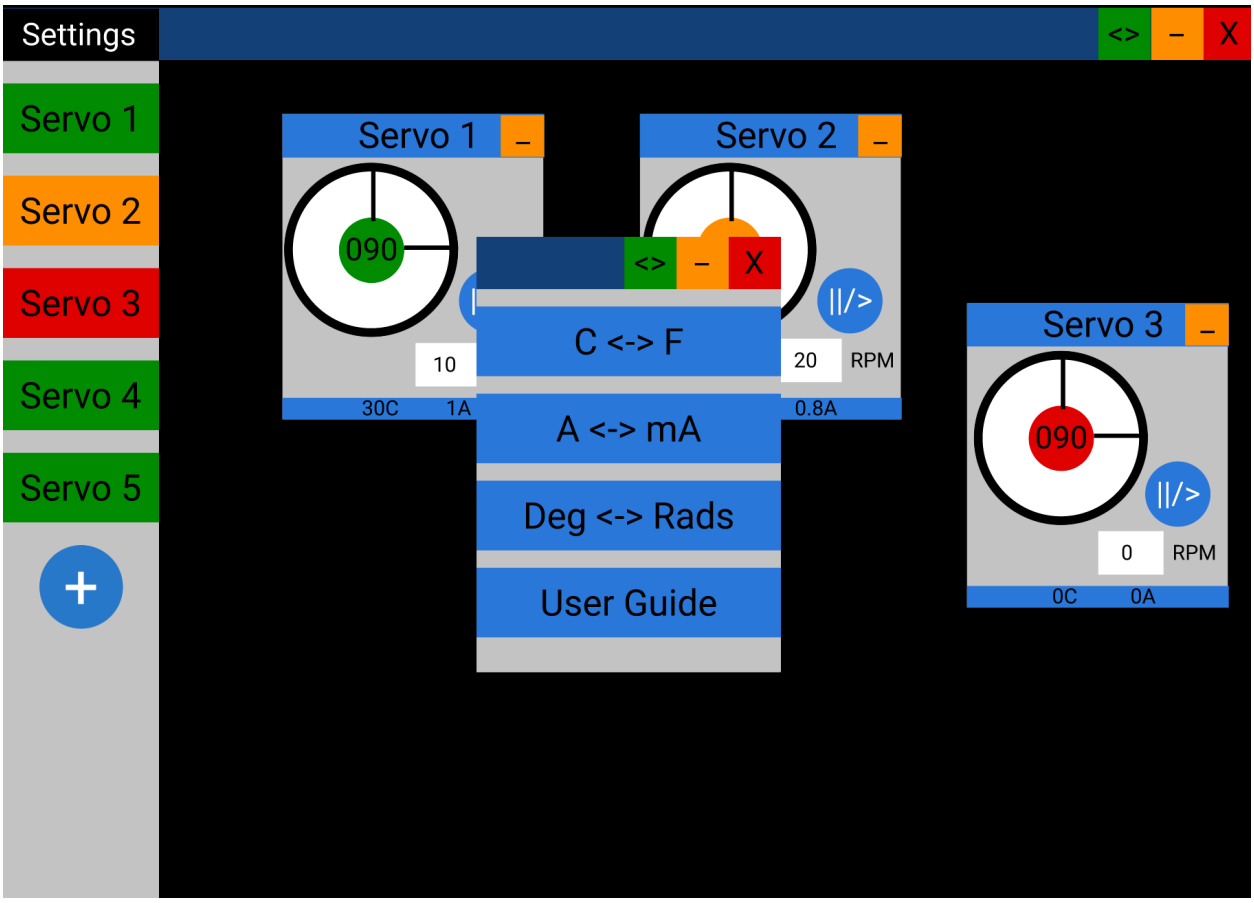
Graphical User Interface Design

Initial Screen



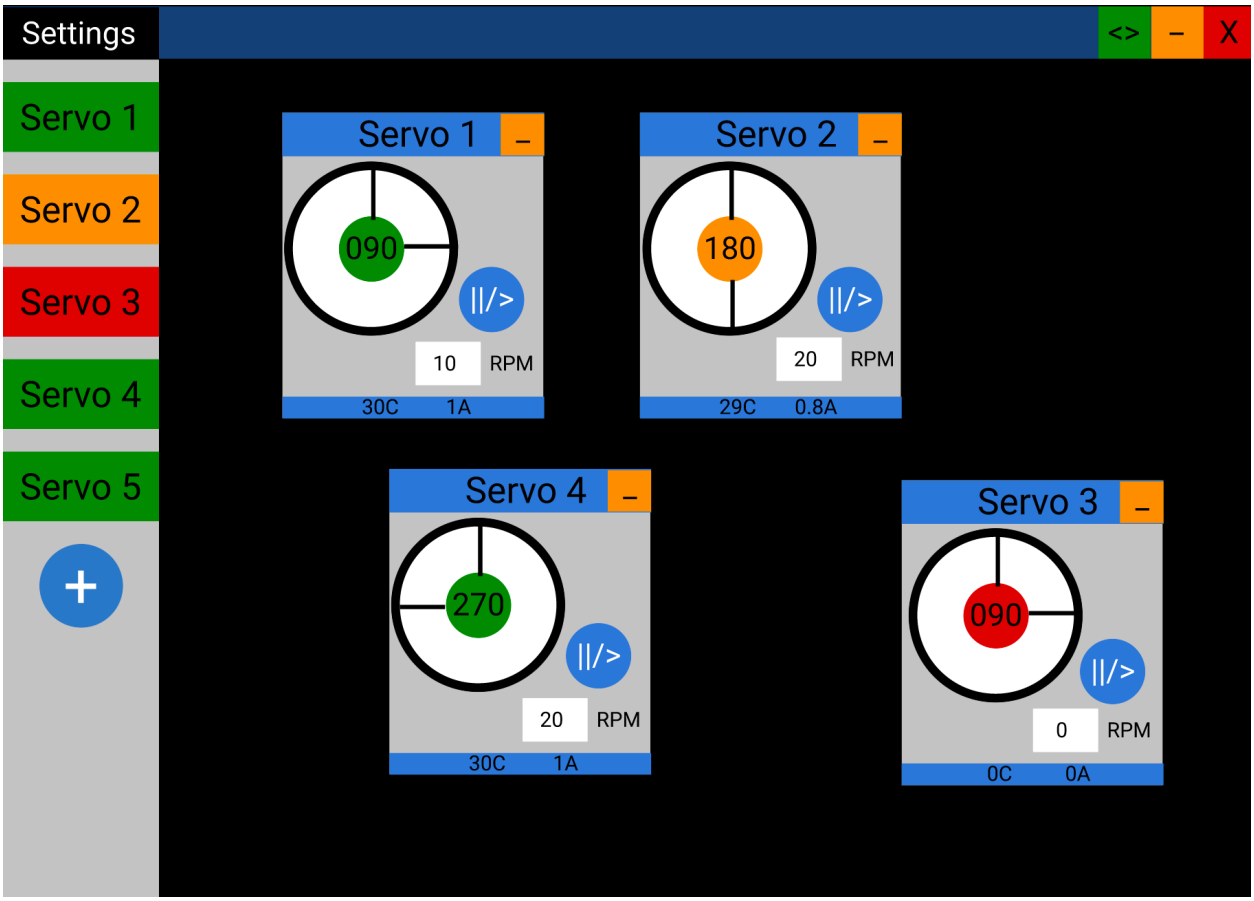
This is an example screen showing how the software will look in action. The grey vertical bar is the “Catalogue” which holds the complete list of servo motors connected to the software. They are listed in the order they are connected to the system and the colours indicate their current state. Green means 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



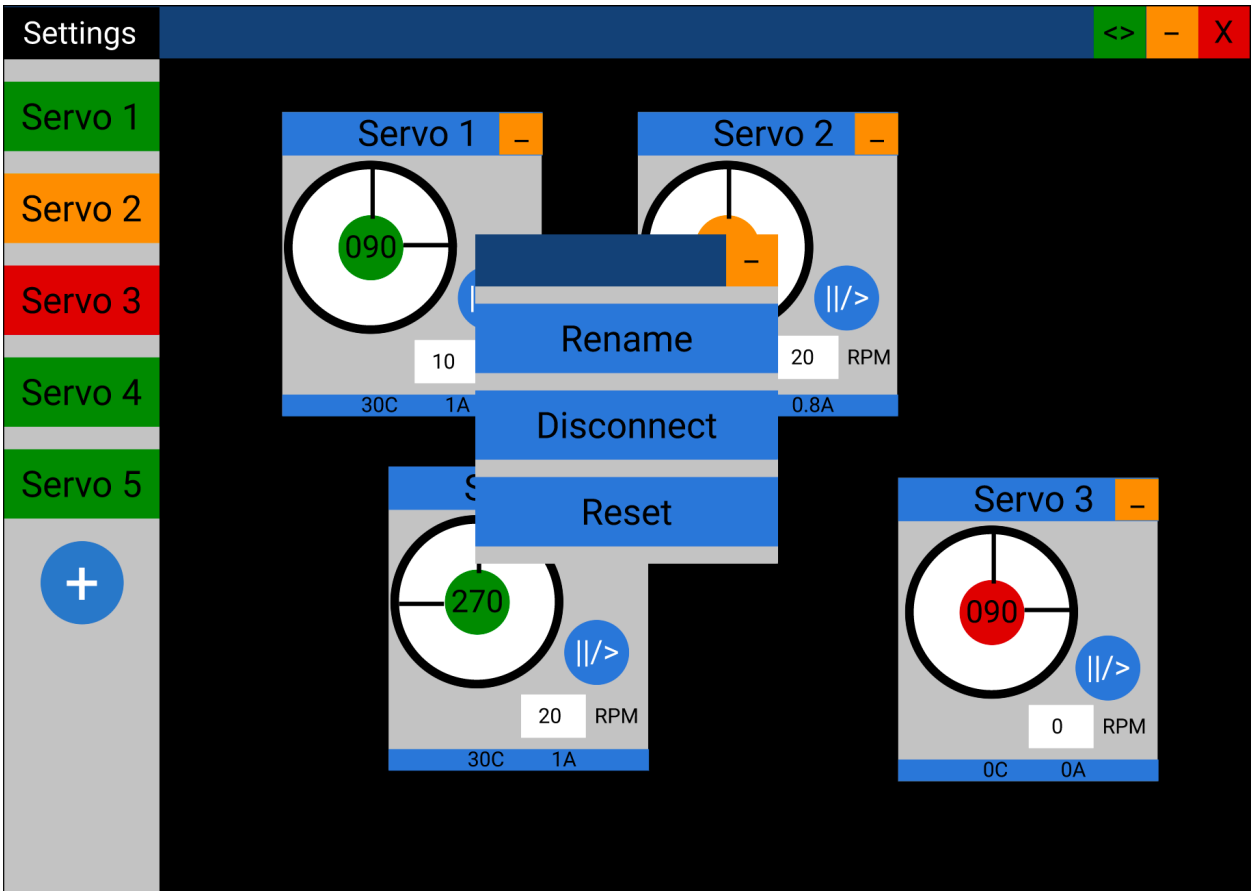
The “Settings” button can be pressed to bring into view the above menu. For the prototype implementation, the setting allows the ability to pull up the user guide for software usage information, and the ability to change units for the Servo’s position, temperature, and current readings.

Dragging Functionality



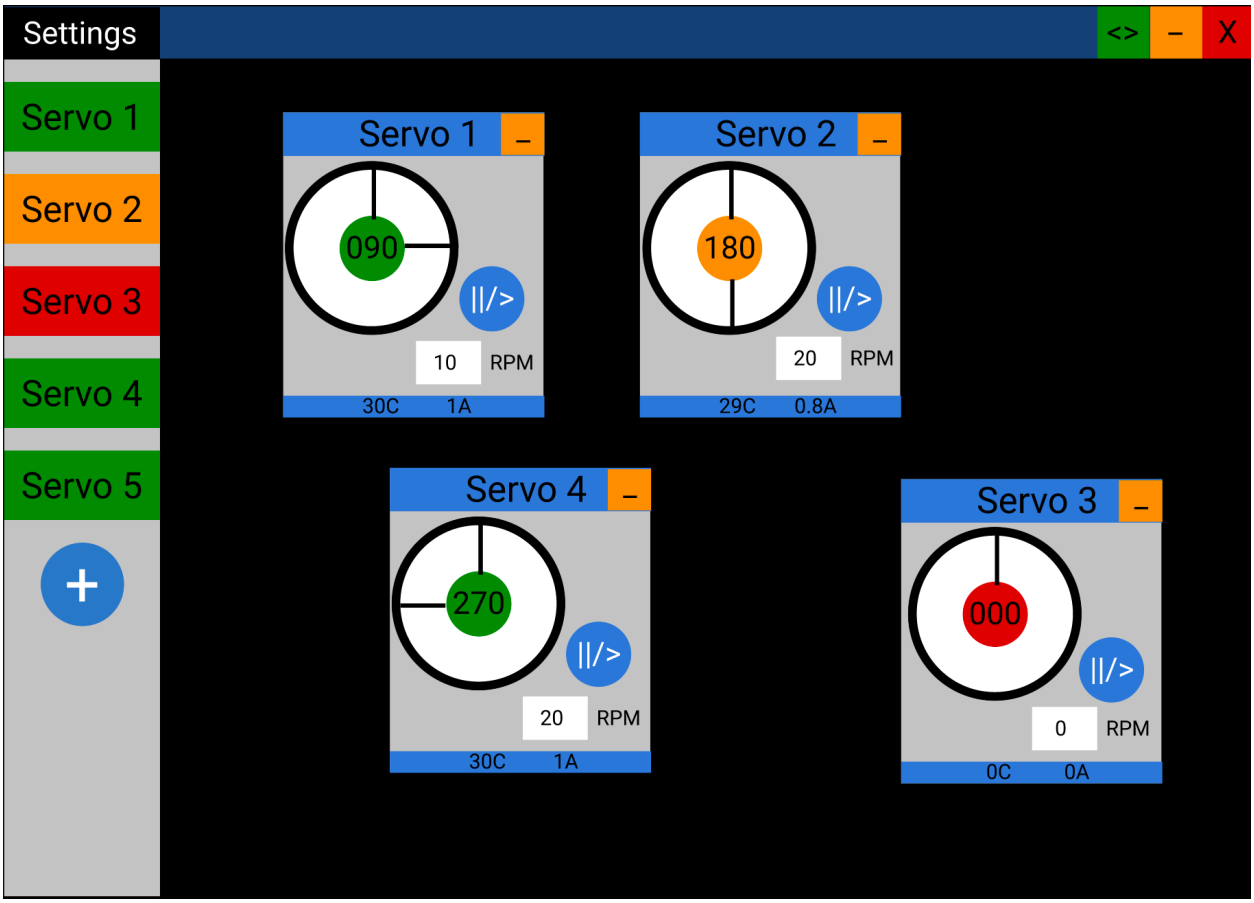
The individual visualisation units of each servo can be dragged around on the “Canvas”. The goal is to make the Canvas infinite with the ability to zoom in and out as the user pleases, as seen above with the repositioning of “Servo 3”. New servos can be dragged onto the Canvas as seen above with the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

Right-Click Functionality for Catalogue Elements



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

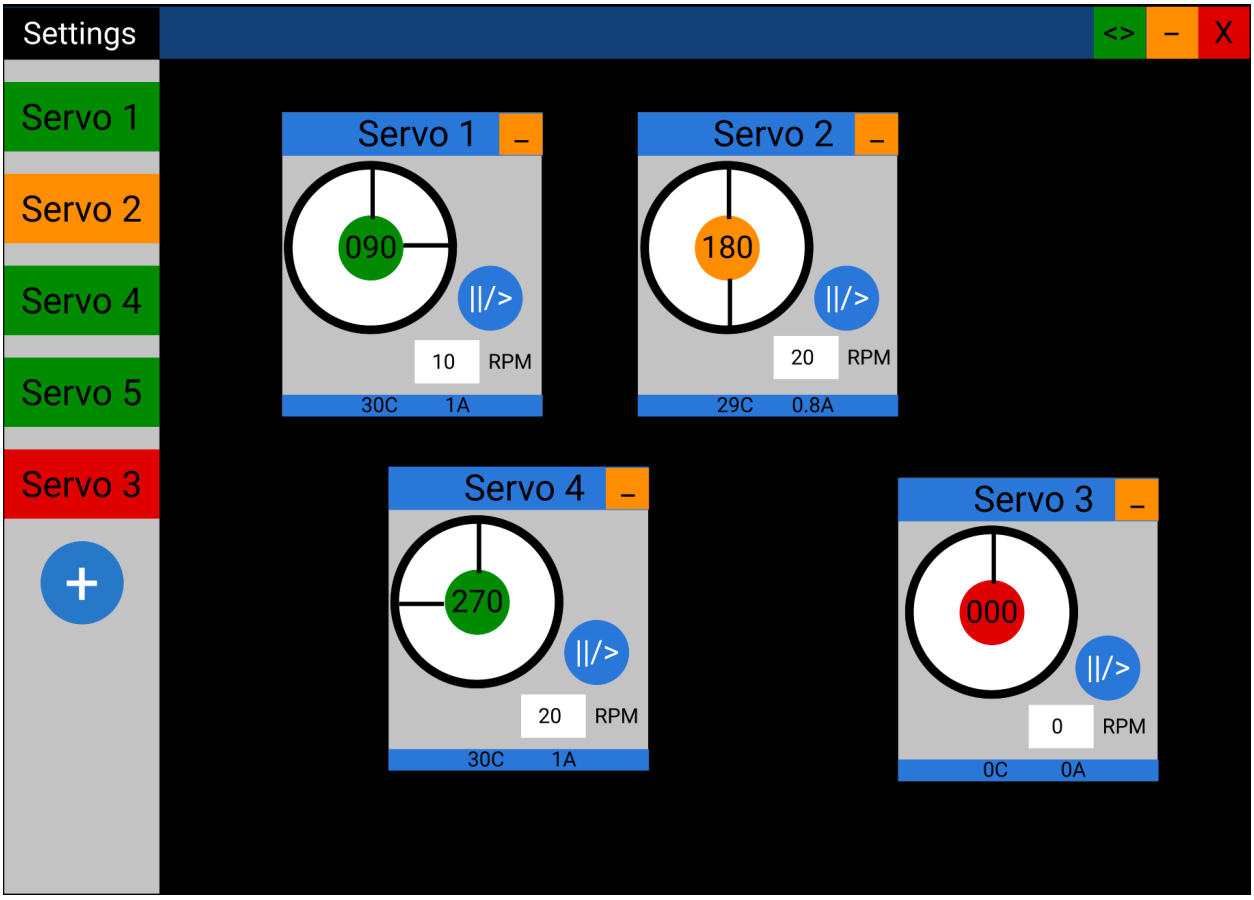


As discussed above, the "Reset" functionality allows troubleshooting and testing capabilities for the library by enabling the ability to reset the angle of the servo, place it in an "off" state, reset the current travelling through the individual servo and its speed, and wait for the servo temperature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

Disconnect Functionality

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

Scan Functionality



Above is 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.