Improving Usability of a Mechatronic Drum Robot

Jamie U

***Abstract*—This report details the development of a graphical user interface for DrumBot to improve ease of use. DrumBot is a mechatronic robotic system capable of playing complex MIDI compositions on arbitrary drum kit setups. In order to prepare DrumBot for a performance, the user must first run the accompanying command line or graphical application and follow through a setup and configuration process. But when this project inherited DrumBot, several significant usability issues were made apparent after testing the graphical user interface. This included requiring text input where one would expect a button or selection input, unexpected hangs in the software, and users possessing little fidelity over certain configuration settings, which made the configuration process confusing and tedious. It was particularly difficult to navigate the setup process for the first time if one was unfamiliar with the inner workings of the system. This project addressed these issues by producing a new graphical application that is more intuitive than the previous, using standard usability heuristics for user interface design to evaluate. Development involved designing a new user interface, refining the process of configuring the various DrumBot settings, facilitating communications between the host computer and DrumBot, and navigating the challenges of designing an application that reliably interfaces with a real-time embedded system. This new application was designed to better guide the user to navigate the DrumBot setup process, and gives the user greater control over configuration, making DrumBot more usable overall.**

***Index Terms*—Drum Robot, Electrical and Electronic Engineering, Software Engineering, Mechatronic Robotic System**

I. Introduction

DRUMBOT is a mechatronic robotic system capable of playing complex MIDI compositions on a drum kit. It consists of three robotic arms, a self-actuated kick pedal, and a self-actuated hi-hat pedal. A user can send a stream of MIDI (a commonly used protocol for describing electronic music notes) notes over a serial connection to the motherboard, which then sends commands out to the arms and pedals instructing them to move and play the notes.

Each component has a daughterboard which connects to the central motherboard. The motherboard receives commands over a serial connection and sends instructions out to the components. The command contains the ID of the component that the command is for, so only the associated component will respond. For example, the user sends a command via the user interface instructing DrumBot to move Arm 1 to the snare drum. The command is received by the motherboard, which sends the command out to all the arms. The board of the matching arm – Arm 1 in this case – responds and moves the arm to the position of the snare. Other DrumBot commands include instructions for calibrating the limits of the arm joints and configuring the positions of the drums.

This project was supervised by Dale Carnegie (primary), and Jim Murphy.

Fig. 1. DrumBot setup

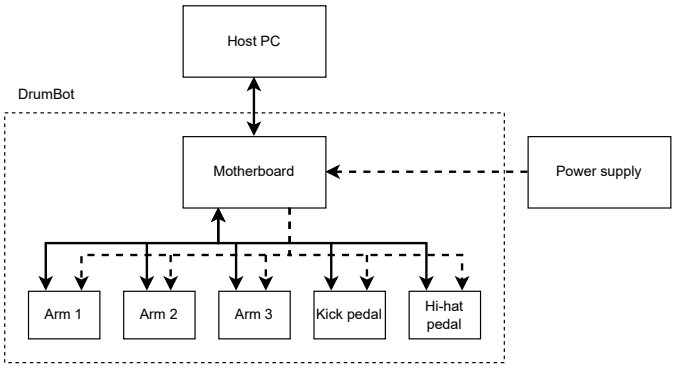

Fig. 2. DrumBot high-level structure – solid lines represent serial communication and dashed lines represent power

For a user to communicate with DrumBot, they have the option to use the CLI (command line interface) or GUI (graphical user interface). The DrumBot CLI is more robust than the GUI, but is less comfortable to use, especially for a user less experienced using a CLI. This needs to be considered because DrumBot's target users are musicians who wish to use DrumBot for performance, and musicians generally are not familiar with CLIs. This makes the GUI the preferred application to use.

The GUI provides a more user-friendly experience by making user input easier to provide via buttons, and not overwhelming the user with a text dump of information messages. But while it is an improvement from the CLI application in terms of accessibility, the GUI possesses several significant usability issues, which were identified at the beginning of this project. These issues include requiring text input where a clickable selection or button confirmation would be expected, and being unable to cancel certain actions – issues which ultimately make the GUI only a slight improvement over the

more "primitive" CLI. For both the CLI and GUI, it is also extremely difficult for a user who is not familiar with DrumBot to figure out how to configure the system due to the lack of documentation.

Addressing these usability issues is important for achieving the greater goal of orchestrating a mechatronic band – one of the overall goals for DrumBot is to coordinate with other mechatronic instruments MechBass (bass guitar) and Azure Talos (6-string guitar) to form a heterogeneous mechatronic ensemble which can outperform other mechatronic ensembles. To make this goal feasible, DrumBot needs to be accessible to musicians who may have little engineering knowledge and understanding about the inner workings of DrumBot, and this is what this project achieved.

This project developed a new GUI, *DrumSet*. DrumSet is developed with the aim of making a fully graphical UI that utilises UI design conventions and fulfils usability heuristics such as those outlined by Norman Nielsen [1]. Key improvements over the old GUI include the elimination of text input areas and replacing them with buttons, lists, and comboboxes, providing the user with more guidance via non-intrusive helpful messages, implementation of more error messages to inform the user of the system status, and stopping the system from unexpectedly hanging. DrumSet not only improves user experience and accessibility, but also improves code extensibility. Features can be more easily added onto the codebase, and DrumSet's GUI is built with PyQt5 allowing developers to use the Qt Designer tool which enhances the development process. By developing a new and improved GUI, DrumBot has become more accessible to users, allowing musicians to configure DrumBot with ease and use it in a performance with other mechatronic instruments.

II. Related Work

A. Robotic Music Ensembles

This section briefly covers several other robotic musical ensembles that exist to help give context to the overall goals of DrumBot.

There are several examples of robotic instruments and robotic ensembles in the world (such as Compressorhead, Z-Machines, The Trons [2], and The Machine Orchestra [3]), but not all are the same. Some of these instruments are designed to play alongside human players in a robot-human collaboration of sorts, while others are fully automated and do not require human input during performance.

The Machine Orchestra is an example of a robot-human ensemble that incorporates robotic instruments and human musicians [3]. Many of the robotic instruments involved in the performances require the inputs of musicians to produce music. For example, a percussion robotic instrument could be connected to a custom sitar and responds to the player's inputs into the sitar [4]. In a sense, the percussion robot is "playing alongside" the sitar player. There are also other robotic instruments in the ensemble that receive input from musicians, but not via another traditional instrument. In summary, The Machine Orchestra puts on a unique experience of robots and humans playing alongside each other in harmony.

Compressorhead differs in comparison to The Machine Orchestra in that it is a rock band consisting of only robots [2]. The robots are humanoid in appearance and play their electric and acoustic instruments in a human-like manner, giving a unique visual aspect to performances. They are controlled via a MIDI sequencer [2] much like DrumBot. The lineup consists of a lead guitarist, bassist, drummer, drummer's assistant (for operating the hi-hat cymbal), a vocalist, and a rhythm guitarist [2], imitating a complete rock band. Compressorhead's combination of musicality and visual appeal makes it highly entertaining for audiences.

As discussed previously, one of DrumBot's greater goals is to form a mechatronic ensemble with other mechatronic instruments MechBass and Azure Talos. This ensemble would differ to the previously discussed robotic bands and offer its own unique experience that is both musically impressive and visually intriguing. However, if DrumBot is not easily accessible to its target users (musicians), then its use becomes limited. Thus, this project sought to address DrumBot's usability issues by developing a new GUI.

B. Evaluating User Interface Design

This section discusses the evaluation methods used in this project.

Norman Nielsen's 10 Usability Heuristics for User Interface Design [1] are widely used heuristics for evaluating user interfaces. They are not strict guidelines, but instead general principles to follow when designing and evaluating user interfaces. These heuristics were proposed by Jakob Nielsen. The 10 heuristics are below, with a short summary. They are described in more detail in [1].

1) Visibility of system status – Keep the user informed of what is going on.
2) Match between system and the real world – Use words, phrases, and concepts familiar to the user, follow real-world conventions and natural and logical organisation.
3) User control and freedom – Allow users to quickly back out of unwanted actions.
4) Consistency and standards – Follow platform and established industry conventions.
5) Error prevention – Prevent errors from happening in the first place rather than always relying on error messages.
6) Recognition rather than recall – Minimize the user's memory load by making elements, actions, and options visible. Help should be readily available instead of relying on the user to remember information.
7) Flexibility and efficiency of use – Make shortcuts available to experienced users who are familiar with the interface. Provide personalisation and customisation.
8) Aesthetic and minimalist design – Essential information should not have to compete with irrelevant or rarely needed information for attention.
9) Help users recognize, diagnose, and recover from errors – Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.

10) Help and documentation – If necessary, provide documentation to help users understand how to complete their tasks.

These usability heuristics were used to evaluate DrumBot's preexisting GUI (Section III), define the requirements for DrumSet (Section IV-A), and evaluate DrumSet (Section VI). Using these set of usability principles to evaluate the UIs means that they are being assessed against industry accepted standards.

III. Preliminary Testing

This section details the usability issues identified in the old DrumBot GUI, research into existing solutions, and specifications for the new configuration software. For context, the original goal of this project was to create a software to coordinate mechatronic robot instruments – one of which is DrumBot – to play together as an ensemble. However, upon testing and experimenting with DrumBot and its configuration software, several significant usability issues were made apparent, making the software unsuitable for use by a musician. This prompted a change of project direction to one that is more DrumBot-focused.

Taking time to gain familiarity with DrumBot was necessary in order to understand how use it properly and assess how it could be integrated into an orchestration software. As mentioned, several usability issues with DrumBot's configuration software were identified during this analysis. Indeed, it was the assessment of DrumBot's configuration software that prompted the change of project direction. This section will walk through the features of the configuration software, discuss the issues identified, and suggest improvements.

DrumBot has a CLI (command line interface) and GUI (graphical user interface) which are used to firstly configure the system (configuration mode) and secondly receive and process MIDI data (performance mode).

A. Command Line Interface

The CLI is programmed in the Arduino language which is a variant of C++, as well as regular C++. This is programmed directly into DrumBot's motherboard, which sends and receives data to a host PC through a serial connection.

The lack of documentation for DrumBot made it challenging to figure out how access the CLI, especially without prior personal experience with Arduino. To access the CLI, the serial connection between the host PC and DrumBot must be monitored. For this, the Arduino programming environment was used, but any program with a serial monitor can be used, such as Virtual Studio Code .

When the program starts, the software begins its startup procedure and declares what components (arms, kick, hi-hat) are connected, as well as configured drums and latency. Then, the user is presented with a menu of configuration options that can be selected by entering a number from 0 to 9. If an invalid input is entered, the options are displayed again. The configuration options are as follows:

0 - Ping all possible arms
1 - Calibrate arm limits
2 - Add Drum MIDI position
3 - Clear all calibrated drum locations
4 - Review latency and PID coefficient info
5 - Change the proportional control coefficients
6 - Move to drum corresponding to MIDI note
7 - Run through automated positional latency detection
8 - Enter performance mode
9 - Tighten/Loosen grips
10 - Disable servos

Several of these options, such as *2 - Add Drum MIDI position*, require additional user input after selecting an option. For example, when adding a drum MIDI position, the user is required to enter the character 'y' when the selected arm is in the desired position.

Some configuration options output additional information to the terminal. In particular, when configuring an option that involves the movement of a robot arm, i.e. *1 - Calibrate arm limits* and *2 - Add Drum MIDI position*, the program will continuously output the current motor value to the console. When the robot is put into performance mode using *8 - Enter performance mode*, the program will output the MIDI notes received and additional information such as whether they were played on time or skipped.

Due to lack of help documentation, some configuration settings were unclear as to what is needed to be done by the user. For example, in the *Calibrate arm limits* option, it is unclear what such "limits" should be (they should be the furthest the arms can extend without colliding with objects such as other arms). Overall, it took an immense amount of trial and error to determine how to properly configure DrumBot. In addition, there is no way to cancel a process mid-configuration – if the user makes a mistake, then the configuration process needs to be restarted from the beginning.

There is also a usability issue specific to the *2 - Add Drum MIDI position* configuration option. It is impossible to tell when an arm is outside of its configured range until after the MIDI position has been selected. The current state of the software involves much back and forth and reconfiguration by the user. A possible improvement is displaying to the user the current motor values of the selected robot arm, and its set minimum and maximum values.

Overall, considering the inherent limitations of a CLI, the DrumBot CLI is functional, aside from a few issues. The major issues specific to the CLI being that when 'y' is input it is not "stripped" of whitespace, and that the list of configuration options are not displayed again after a configuration process is complete. To work around these issues, the user must configure that their input has no appended whitespace in the serial monitor, and they may enter an invalid configuration option number to display the list of options again. Other issues with the software exist, but they exist in the GUI interface as well (more issues to be described in III-B).

Obviously, an improved user interface, especially one for a musician without engineering knowledge, would be graphical,

because interactive graphical elements such as buttons are much easier to use and prevent invalid inputs. The next section walks through the original GUI solution as inherited at the start of this project.

B. Graphical User Interface

This section refers to figures in Section 1 of the Appendix [5].

The preexisting DrumBot GUI was built with the aim of making DrumBot easier to use for users such as musicians who have less engineering knowledge [6].

Due to the program being written in Python, the required libraries needed to be manually installed in order to run it. However, this led to the assumption that the program would work with the latest versions of the libraries – this was not the case. Customtkinter changed some of the formats of its components, such as the combobox, meaning old code would not work. It was not known as to what version of the library the DrumBot GUI was written with, so the code had to be updated to comply with the latest version.

Nevertheless, the purpose of creating a GUI for DrumBot is to have a UI that it is easier to use compared to the CLI. The DrumBot GUI attempts to fulfil this in several ways:

- Display the configuration options as buttons that can be clicked. The most critical options are also highlighted for the user.
- Implement a "back" button which allows the user to exit out of the current configuration option.
- Not display "debug" information, i.e. motor values and received MIDI notes, to the user.

A manual evaluation was performed on the DrumBot GUI to assess its usability. During this, several difficulties were experienced while using the software to configure DrumBot, and a list of usability issues were identified. It was surprising that these issues were not caught in previous user testing during its development [6]. The evaluation conducted during this project is detailed below.

1) Connecting: When DrumBot is turned on and connected to the user's computer via serial USB, and the DrumBot GUI is open, then the user must select which port the DrumBot is connected to from a combo box (see Fig. 1 [5].), which is a good design pattern (a design pattern in software and user experience design is a practice used to solve a commonly occurring problem). The list of ports can be refreshed in case DrumBot is connected *after* launching the GUI. It is also clear what actions the user should take here, that is, to connect DrumBot and select the port.

However, one issue was identified. If the selected port is invalid and the user selects a configuration option, the software hangs. It would be better if the program was halted and an error message displayed to the user, informing that there is a connection error.

2) Option Menu: The DrumBot configuration options are displayed as large buttons that are easy to read. The most critical options are highlighted for the user, guiding them as to what should be configured before entering performance mode (Fig. 2 [5]).

An issue identified with the usability is one that sometimes occurs when selecting an option. A pop-up appears, as shown in Fig. 3 [5]. This sort of "error" never occurs in the CLI version of the configuration software. It is also unclear what should be done when this pop-up shows – it seems that you need to continuously keep selecting the option until it works. But sometimes, this is an indication of a connection error and the software needs to be restarted. In any case, there is no indication of what the error is. This violates the usability heuristic of *visibility of system status* [1]. The wording of the pop-up also implies that the user needs to "wait" for the pop-up to go away, when that is not the solution.

3) Using The Configuration Options: The main issue with the DrumBot GUI is that despite it being graphical, it largely functions the same as the CLI version of the software, and also hides away information that could be useful for debugging. The GUI does not fully utilise graphical features, for example requiring text input for confirmation for everything, instead of using dropdowns or buttons. Also, the messages displayed to the screen exactly mirror those of the command line prompts with no amendments to typos and grammatical errors. This means that the GUI essentially behaves in the same way as the CLI, but with additional issues (to be described) and less information being shown to the user (motor values when configuring arms, and information on MIDI notes received are not shown to the user).

a) Software hangs when pinging arms: Upon clicking *Ping all possible arms*, there is a delay where the software seems to "hang". After pinging is complete, the user is shown a screen that says which arms are connected. However, the "hang" is rather startling and violates the usability heuristic of *visibility of system status* [1]. It would be better if there was no hang and the screen changed instantaneously, *then* pinged, or there was some sort of indication that the software was loading and not hanging. (e.g. animated loading icon).

b) Software hangs when disabling servos: The interface also seems to "hang" when disabling servos.

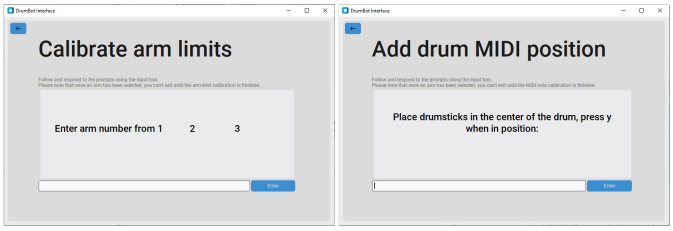

Fig. 3. The GUI requires text input in places it shouldn't, contrary to conventional design

c) Selecting an arm requires text input: When selecting an arm to configure, the user is prompted to enter a number a number from 1 to 3 in a text input box (Fig. 4 [5]). This is poor UI design; the arms should be selectable via buttons, or a dropdown. This doesn't utilise graphical features, making it functionally no different to the CLI.

d) Confirming input requires text input: "Press y when in position" is shown when configuring one of the robot arms (Fig. 5 [5]). It would be more sensible to use a button for confirmation instead.

e) Motor values are hidden during calibration: When calibrating arm limits, it is no longer possible to view the value of the motor being set. In the CLI version, the program would constantly output the motor position value to the serial monitor. These values are intentionally hidden from the user to have a cleaner UI. But this has the adverse effect of hiding what is actually useful information for debugging the system. The current value of the motor should be displayed somewhere.

f) User must exit and re-select configuration option: Upon completion of configuring an arm, the software asks the user to "please exit" (Fig. 6 [5].) Better user experience would involve "resetting" the configuration option so that the user can immediately begin configuring another arm, without having to exit and click back in again.

g) Unable to clear single MIDI positions from arm: It is only possible to clear all MIDI note positions at once, then reconfigure the positions.

h) Unable to cancel action mid-configuration: It is not possible to exit and cancel actions while in the middle of configuring an arm. This affects the usability heuristic of *user control and freedom* [1]. This behaviour is also observed in the CLI.

C. Performance Mode

When in performance mode, the user is presented with a screen with a single message – no other information is communicated. This is actually a downgrade from the CLI program – when the CLI program was in performance mode, the received MIDI notes and details on whether the note was played on time or skipped would be output to the screen. It is also unclear for a new user as to what should be done in performance mode. There is a lack of documentation explaining that DrumBot should be connected to a DAW (digital audio workstation) such as FL Studio, Ableton Live, or Logic Pro X, like a regular instrument. A simple solution for this is to add additional help messages to the UI.

Overall, while the GUI makes it a little easier for a user without engineering experience to understand and use the software, it doesn't do a good job of utilising graphical features to improve the user experience. The user is still required to *frequently* switch between the mouse and keyboard when configuring DrumBot, and there is no additional indication of what "valid" input is for some configuration options, such as *Move to drum corresponding to MIDI note*.

There is also a lack of documentation on how to configure the software properly, for both the CLI and GUI programs. For example, in the *Calibrate arm limits* option, it is unclear what such "limits" should be, as mentioned previously.

Analysing the code reveals that the GUI essentially sits on top of the CLI – the CLI output is buffered to a stream which is read by the GUI. Essentially, the CLI hides underneath the GUI, instead of the being replaced by it. While this solution works to an extent, is not a particularly elegant solution, and isn't very flexible. This is not an ideal solution as the GUI *depends* on the CLI. The GUI directly reflects the CLI, meaning code changes would need to be both the CLI and the GUI. Instead, it is preferable if the CLI and GUI commands were separate. This, as well as selecting an ideal GUI framework that is well supported, would make it easier for developers to build on the software in the future.

IV. Design

A. Requirements

This section discusses the requirements for the new configuration software, DrumSet. These requirements are largely informed by *Norman Nielsen's Usability Heuristics for User Interface Design* [1] as well as the shortcomings of the original DrumBot GUI and CLI as discussed in III. In general, the software should allow, at a minimum, the same functionality offered by the original DrumBot GUI application.

1) The software shall allow the user to ping the arms, kick pedal, and hi-hat – *Pinging the arms involves sending a "ping" request to DrumBot which then sends a request to each component. The components that send a response back are connected. This is necessary for knowing which components are available for configuration.*
   - The software shall inform the user when a ping is in process. (Usability heuristic *Visibility of system status [1]*)
   - The software shall inform the user which components have responded to the ping.
   - The software shall inform the user how long it took for a component to respond.
2) The software shall allow the user to calibrate the limits of the joints
   - The software shall give the user clear instructions to calibrate a joint. (Usability heuristic *Help and documentation [1]*
   - The software shall allow the user to calibrate a specific joint. (Usability heuristic *User control and freedom [1]*)
   - The software shall allow the user to calibrate all joints in one routine.
   - The user shall be able to cancel calibration once started. (Usability heuristic *User control and freedom [1]*)
3) The software shall allow the user to configure MIDI drum positions
   - The software shall allow the user to add a new MIDI drum position.
   - The software shall allow the user to remove a single configured MIDI drum position. (Usability heuristic *User control and freedom [1]*)
   - The software shall allow the user to remove multiple configured MIDI drum positions at once.
4) The software shall allow the user to run through the automated positional latency detection routine
   - The software shall inform the user when the routine is running, and when it has finished. (Usability heuristic *Visibility of system status [1]*)

5) The software shall allow the user to review configured values of DrumBot
6) The software shall allow the user to directly edit the shoulder PID coefficients of the arms
   - The user shall be able to cancel editing the shoulder PID coefficients. (Usability heuristic *User control and freedom [1]*)
7) The software shall allow the user to move an arm to a configured drum position
8) The software shall allow the user to tighten and loosen the grips of the wrists
9) The software shall allow the user to disable the servo-motors of the arms
10) The software shall allow the user to put DrumBot into Performance Mode
    - The software shall display the log of DrumBot system messages to the user while DrumBot is performing. (Usability heuristic *Visibility of system status [1]*)
11) The software shall allow the CLI application to function as normal – *Modifications to DrumBot's code may be made to accommodate the new GUI application, but the user should still be able to use the CLI application.*
12) While the software is completing an action, the software shall prevent user input – *This is to prevent the system from entering an error state. (Usability heuristic Error prevention [1])*
13) The software shall inform the user what each configuration setting does – *(Usability heuristic Help and documentation [1])*

1) Excluded Features: In the DrumBot CLI and original GUI applications, the user is able to modify values for the wrists by accessing *Change the proportional control coefficients*. However, these configured values cannot be reviewed like how the user can review the shoulder PID coefficients. As it is not clear how the proportional control coefficients affect the wrists' movement (lack of documentation), it was decided that it would not be included in DrumSet. However, it can implemented in future work.

B. Choice of GUI Framework

This section discusses the candidates for GUI framework that were explored.

As the DrumBot code was written in Arduino C++, it was initially thought that using a C++ GUI framework would make it easier to interface with DrumBot. Some frameworks considered were Qt (pronounced "cute"), and JUCE. Qt offers tools, including a "design studio" for designing and implementing GUIs. It is a popular choice of UI framework and is guaranteed to look consistent across Windows, Linux, and macOS – important as musicians make use of a wide range of platforms. JUCE is a C++ framework for audio and plug-in development, making it ideal for creating VST plug-ins (virtual instrument plug-ins), but can also be used for making standalone GUI applications. It also comes with additional MIDI support. In fact, JUCE was used for GUI support in Signal for The Machine Orchestra [7]. Some other options considered were GTK and wxWidgets, but GTK is more focused on creating applications for Linux, and wxWidgets is not guaranteed to look consistent across all platforms, so these candidates were not ideal.

Python libraries were also explored, the motivation being that developing GUIs in Python is generally more developer-friendly. Customtkinter and PyQt5 were considered. Customtkinter is a UI library built upon tkinter (a popular Python UI toolkit) but with additional features and more attractive components out of the box. Customtkinter makes making modern-looking GUIs easy, and is also what the original DrumBot GUI was built with. PyQt5 on the otherhand, is a set of Python bindings for the previously mentioned Qt, allowing the use of the Qt C++ framework but in Python. It was also thought that this would make it easier to interface with C++ code.

In the end, PyQt5 was chosen as the GUI framework. Coding the application in Python is more developer-friendly than in C++, and at the time it was also believed that PyQt5 should also allow easy interfacing with DrumBot C++ code, making it the preferred option over Customtkinter. PyQt5 was also chosen over JUCE because building components appeared to be simpler, and the additional MIDI support from JUCE was not deemed necessary. Qt is well-documented and is discussed a lot in online forums by the developer community. PyQt5 also comes with the free-to-use Qt Designer tool, which is a graphical application for designing GUIs without using code. This makes creating and maintaining a complex GUI application much easier than developing the GUI from scratch purely with code. Having access to the Qt Designer is a significant advantage of using PyQt5.

It should be noted that while it was initially thought that using a C++ framework would be advantageous for interfacing with the Arduino C++ code, this was a misconception, as the communication between DrumBot's motherboard and the host PC can only be achieved through messages sent over a serial port – the language the GUI application is built in does not matter for interfacing with DrumBot. This was realised after development of the new DrumBot configuration software had began, and a clearer understanding of the existing codebase and how DrumBot works was gained.

C. Design Planning

This section discusses the planned design for DrumSet, as a result of the requirements IV-A.

1) Configuration Settings: It was decided that DrumSet's list of configuration settings would differ from those of the DrumBot CLI and original GUI (III-B). The aim was to make the configuration settings more intuitive by grouping related settings together, which as a result, reduces the number of main menu options and the amount of navigation required by the user.

The original eleven menu options were reduced to eight:

- Calibrate Arm Limits
- Edit MIDI Positions
- Run Through Automated Positional Latency Detection Routine

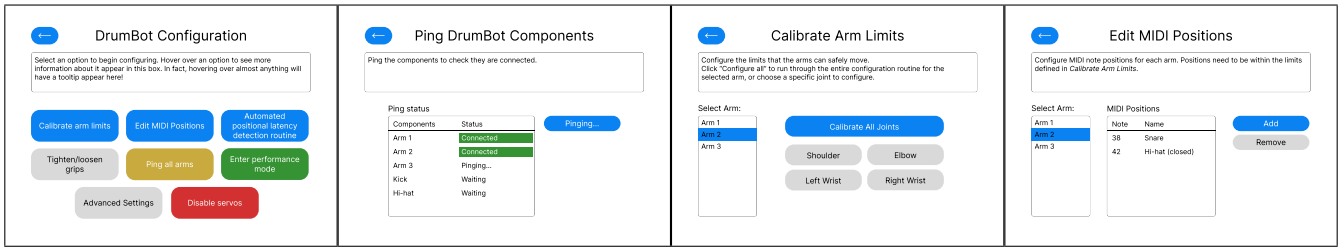

Fig. 4. DrumSet mockup overview, full mockup suite available in Section 2 of Appendix [5]

- Tighten/Loosen Grips
- Ping All Arms
- Enter Performance Mode
- Advanced Settings
- Disable Servos

The *Edit MIDI Positions* option allows the user to add or remove (single or multiple) MIDI drum positions and command an arm to move to a configured position. This intuitively groups together three of DrumBot's configuration settings in one place.

Advanced Settings groups together *Review latency and PID coefficient info* and *Change the proportional control coefficients*, two configuration options that would not normally be accessed by a user with little understanding of the inner workings of DrumBot. By grouping these options under the heading "Advanced Settings", it makes it obvious to a user that they should not be accessing these functions unless they know what they are doing.

2) Mockup: A mockup (Fig. 4) was created to plan and visualise how DrumSet would like (the full suite of wireframes can be viewed in [5]). The final product as discussed later in Section V does not look exactly like this, but comes very close – design decisions were made during implementation as they were seen fit. While the design takes some inspiration from the original DrumBot GUI (e.g. in the main menu), there are several major differences.

Of note, the design does not take any text input from the user as if it were a CLI application, but instead utilises graphical features such as buttons, comboboxes, and list and table selections. At no point is the user required to use their keyboard to submit text input – the user selects the arm they want to configure by clicking the desired arm in the arm selection list, and MIDI positions are selected via a combobox, just to name a few improvements.

The design also has a "hint box" for displaying information about the current page. It also will show information about what the user is hovering over – when the user hovers over a widget (a GUI component) such as a button, the hint box will update its message to describe what the widget does, providing easily accessible in-app help documentation. This design choice was directly inspired by FL Studio's user interface, a popular DAW used by musicians.

a) Calibrate Arm Limits: The user selects the arm they wish to configure, then clicks a button to start configuring a joint. They can configure one specific joint, or all joints in one continuous routine. To advance through the calibration process, the use clicks the *Submit/Save* button. The user may cancel the calibration process at any point via the *Cancel* button.

b) Edit MIDI Positions: The user selects the arm they wish to configure. DrumSet presents the user with the drum MIDI positions the arm has been loaded with. From here, the user can configure a new position by clicking the *Add* button. The user then selects the MIDI drum note they wish to configure from the combobox, then moves the arm's drumsticks to the centre of the desired drum and clicks *Save* when done.

In this menu setting, the user can also remove a configured position by selecting it from the table and clicking *Remove*. Multiple positions can be removed at once by clicking and dragging or holding down the `Ctrl` or `Shift` keys to select. An arm can also be made to move to a position if a single position is selected.

c) Positional Latency Detection Routine: The user clicks the button to begin the routine. While the routine is running, a loading bar is shown to give an indication of progress. A message is displayed when the routine has finished.

d) Tighten/Loosen Grips: The user selects the arm they wish to configure, then the grip (left or right). The user then clicks either the *Tighten* button to tighten the selected grip, or the *Loosen* button to loosen it. As soon as tightening or loosening begins, a *Stop Tightening* button appears. When the user is finished configuring the grip and wants to go back to the arm/grip selection page, they click the *Finish* button.

e) Advanced Settings: The user selects the arm they wish to review or configure. The calibrated limits, loaded MIDI notes, latency, and shoulder PID coefficients. The user can click *Edit* to enable in-place editing of any of the values.

f) Ping Arms: The user is presented with the ping information of the last time the components were pinged. The user clicks the button to ping the arms. While DrumSet is waiting for responses from DrumBot, the interface signifies that pinging is in progress. As soon as a ping response is received from a component, the latency is displayed in the ping status table, and the component is listed as "Connected". If the component did not respond to the ping in the required time frame, the component is listed as "Not Connected". When the pinging process is complete, the time of the last ping is updated.

g) Performance Mode: Upon selecting this menu option, DrumBot is put into Performance Mode. When DrumBot receives a MIDI stream, the output is displayed in the MIDI Log. This information may be useful for debugging.

h) Disable Servos: The user clicks the *Disable Servos* menu option, and a "loading" popup message is shown while

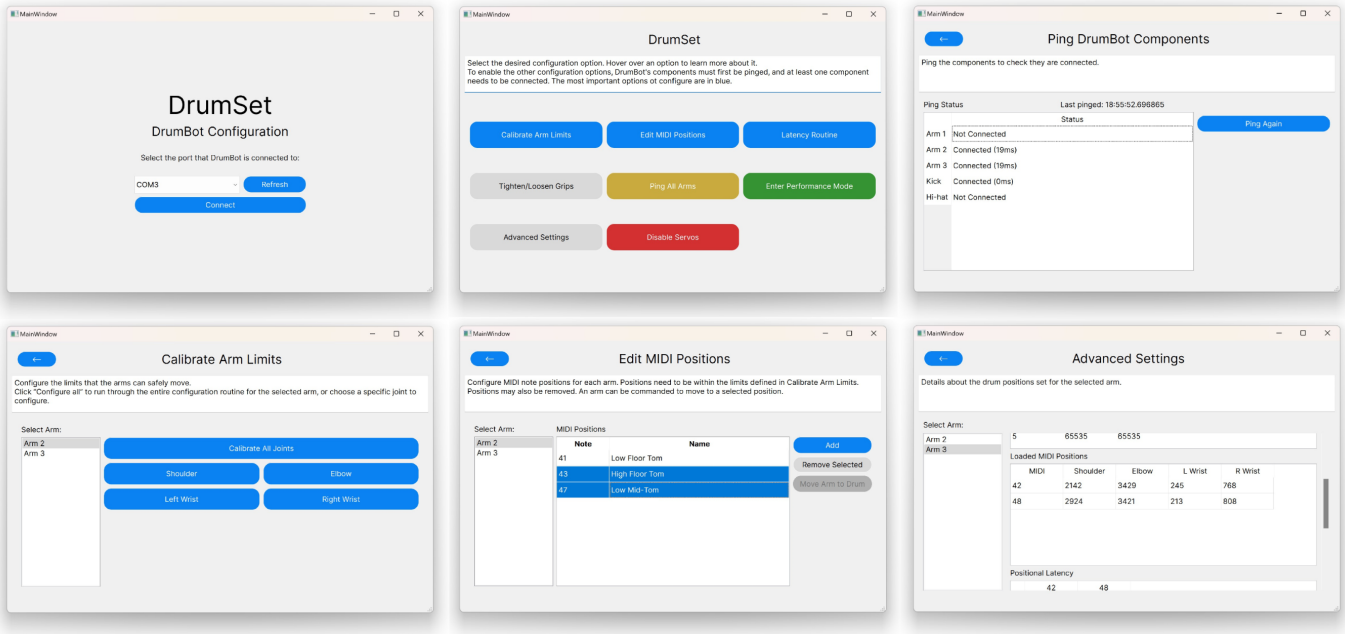

Fig. 5. Overview of DrumSet implementation

DrumBot is disabling the servomotors. A message is displayed when the servos have been disabled.

V. Implementation

This section discusses the realisation of the design, the differences between the design and the implementation, and challenges faced. An overview of the implemented design is shown in Fig. 5.

DrumSet was developed in Python with the PyQt5 library. Because the design of DrumSet has a large number of components, Qt Designer (the design tool PyQt5 comes packaged with) was used to build the GUI. Qt Designer is a tool that allows you to design a GUI in a drag-and-drop manner, making it easier to build and maintain the GUI. Using Qt Designer reduces the amount of code the developer needs to write. The Qt Designer also allows the GUI to be styled with a CSS-like style sheet, which makes it easy to update the look and feel of widgets.

By using PyQt5, it contributes to the sustainability goals of the project, that is, to ensure DrumSet is easy for other developers to extend and maintain in future. The application was also designed such that it does not directly mirror the DrumBot CLI, so that development of the CLI and DrumSet can diverge from each other and have different functionality. This is an improvement from the original GUI application which was not easily extensible due to the fact that it directly mirrored the CLI and used the same commands.

In addition to developing the DrumSet application, DrumBot's code needed to be modified in order to respond to DrumSet. Developing for DrumBot requires the Teensyduino add-on for Arduino. The Arduino IDE was used to modify DrumBot's code, which is written in Arduino C++. Several new functions were developed for DrumBot in order to respond to DrumSet.

One of the requirements for DrumSet is to ensure that the functionality of the DrumBot CLI is maintained. To accomplish this, commands sent to DrumBot from DrumSet were designed to begin with a "!" character. This character signifies to DrumBot that the command should be interpreted differently to a CLI command. CLI commands and DrumSet commands often elicit the same behaviour/response from DrumBot and its arms, but some actions require different behaviour depending on if the command were from the CLI or from DrumSet. For example, to add a new MIDI position using the CLI, the user would submit the number of the arm, then the number of the MIDI position, then submit "y" to complete the configuration – three separate inputs in total. For DrumSet, only one command – containing the arm and the MIDI note – is sent.

To provide a smooth user experience, much of the functionality of DrumSet involves starting up a separate thread to complete the task so that the GUI does not freeze. This is because long-running processes and blocking instructions block GUI updates, so threads are required to run such processes in a separate thread to the GUI.

Many of the actions in DrumSet require some time to complete, such as waiting for a response from DrumBot. Such actions require blocking user input so that processes are not interrupted and DrumSet is not put into an error state (e.g. a state where two threads are reading from the serial port at the same time). This has been accomplished via a "loading message popup" (Fig. 6), where a popup message appears with the message "Loading..." and does not hide until the process is complete. The user is able to close the popup by clicking the x, however; this solution is quite rudimentary, but lays the groundwork for a more robust solution in future.

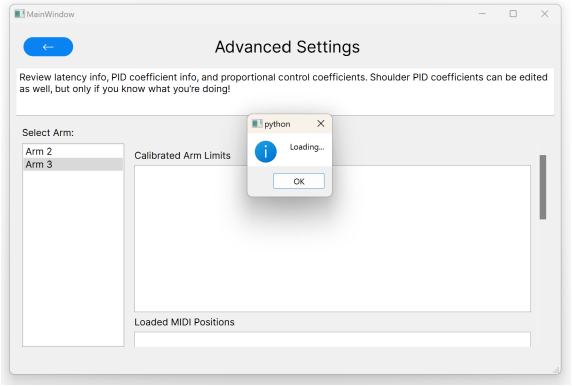

Fig. 6. DrumSet loading message popup in Advanced Settings

A. Features

1) Connection: (See Fig. 5) The process of connecting to DrumBot is largely the same as the original DrumBot GUI. When DrumSet is opened, the user is presented with the list of available ports, selectable in a combobox. The user can refresh the list of ports by clicking the *Refresh* button, which spins up a separate thread to complete the task. While the ports are being retrieved, the *Refresh* button becomes disabled and its text becomes "Refreshing...". The reason a separate thread is necessary for retrieving the list of available ports is because it is a long-running process and would otherwise cause the GUI to hang. Once the user has selected the port DrumBot is connected to, they click *Connect*, and DrumSet sets up the serial connection.

2) Main Menu: The main menu presents the eight menu options to the user (see Fig. 5). From here, the user can select their desired configuration option. If the DrumBot components have not yet been pinged (i.e. when DrumSet has only just been opened), then only the *Ping All Arms* option will be available, with all other disabled (see Fig. 16 in Appendix [5]). This is because DrumSet does not know which of the components are connected, and all other options require at least one component to be available. Specifically, *Performance Mode* requires at least one arm, the kick, or hi-hat to be available. All other options require at least one arm, since they are configuration options for arms. After the user has used the *Ping Arms* option, the software will enable/disable the other configuration options based on what components are connected.

3) Hint Box: A tooltip system was implemented to help users understand what different things mean. At the top of the screen, a box displays information about what the user is currently hovering over, or general information about the page if the user is not hovering over a widget with an associated hint.

To implement this, a map of widget names to hint messages was created in a separate file. Then, in the main file when the GUI is initialised, the program traverses through the widget tree of the GUI and checks if the widget has an associated hint message. If it does, then a hover event listener is installed into the widget, which updates the hint box's text if the mouse hovers over the widget. If the mouse moves off the widget, then the text is set to the default text for the current page.

4) Ping Arms: (See Fig. 23 in Appendix [5]) The *Ping Arms* option allows the user to ping the components of DrumBot and view details about the most recent ping. At initialisation, the components' response latencies are all recorded internally as -2, and displayed as "Not Pinged" in the ping status table. To ping the components, the user clicks the *Ping Again* button which starts up a thread. The thread sends a ping command over serial to DrumBot then waits for a response for each component. If a response is received, the latency is recorded, and the ping status table is updated with "Connected" and the latency in milliseconds. If the latency for a component is 1000 (arm took too long to respond to DrumBot), or if no response is received for 4 seconds (sufficient time to allow the command to be sent and processed, and for the response to be sent and processed), then the component is recorded as -1 internally, and "Not Connected" in the ping status table.

When pinging the components, the table is updated row by row as responses are received, giving a satisfying visual to the user. This visual required that the ping function be threaded because otherwise the GUI would freeze and not update until pinging was completely finished.

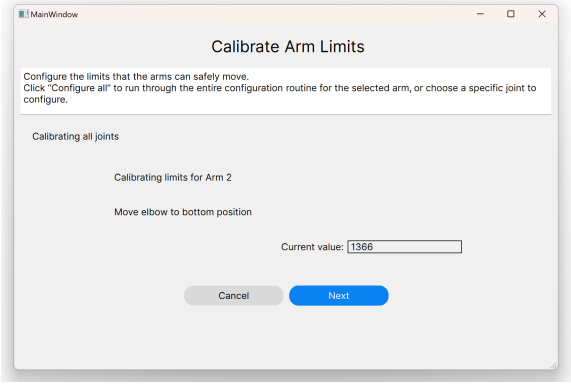

Fig. 7. DrumSet calibrating the limits for elbow of Arm 2

5) Calibrate Arm Limits: (See Fig. 17 in Appendix [5]) The user selects the arm they wish to configure by selecting from the arm selection list, which is only populated with connected arms. Then, the user clicks the button correspond to the joint they wish to configure – the user may configure the limits of just one joint for the selected arm, or configure all joints in one continuous routine.

Clicking a button starts up a new thread for communication with DrumBot. The thread sends a command to DrumBot requesting to begin calibration for the selected arm and joint and waits for a response. While waiting for a response, the loading message popup is shown until DrumBot sends a message confirming that it has begun calibration for the joint.

Implementing threading for this feature was particularly challenging because the thread has to submit multiple commands to DrumBot during its lifetime. It has to send the initial command to start a calibration routine, a command to proceed to the next step of calibration, and a command to cancel the

routine. In addition, state has to be maintained to keep track of what joint is currently being calibrated, whether all joints are being calibrated, and the current step of the routine.

The specific details of the implementation for calibrating a single joint versus all joints are detailed below.

a) Calibrating a single joint: After selecting the joint to calibrate, the user is brought to a screen that instructs them to move the joint to its topmost (for the shoulder) or leftmost (for the elbow and wrists) position. The current value of the servomotor's position is streamed from DrumBot (this is part of the original DrumBot CLI code) and displayed in the GUI. When the joint is in position, the user clicks the *Next* button to proceed to the next step – the thread sends a "y" message to DrumBot to command it to go to the next step, then waits for a response (loading message popup is shown). The next step is to calibrate bottommost/rightmost position, in the same way the topmost/leftmost position was calibrated. Clicking *Next* sends the "y" command, and since this is the last step in this calibration routine, the user is brought back to the main *Calibrate Arm Limits* page once DrumBot has responded confirming calibration has completed. The thread is then discarded. At any point during the calibration process, the user can click *Cancel* which will make the thread send a "x" message to DrumBot, telling it to cancel the current routine and do not save the calibrated values.

b) Calibrating all joints: Calibrating all joints follows a similar process, the main technical difference being that there are more steps. DrumSet guides the user through calibration for the shoulder, elbow, left wrist, and right wrist, in the same manner a single joint would be calibrated. But clicking "Next" after calibrating the bottommost/rightmost position doesn't bring the user back to the main *Calibrate Arm Limits* screen, but instead moves onto calibrating the next joint in the order, unless the current joint is the right wrist, which is the last joint in the order. Cancelling the calibration process discards *all* values calibrated during the entire routine.

6) Edit MIDI Positions: This configuration option comprises of several features and has several dedicated threads for completing the actions – the user can view the currently configured drums, add a new position, remove positions, and move an arm to a configured position. The user selects an arm from the arm selection list, and views the configured MIDI drum positions displayed in a table. The table rows are selectable.

a) Viewing the currently configured drums: When the user enters this menu option, a thread is immediately started to fetch the configured drum positions for the currently selected arm. While waiting for DrumBot to respond with the loaded drums, the loading message popup is shown. The positions are then displayed in the table, as shown in Fig. 8. The user can select a different arm via the arm selection list, which will send a new command to DrumBot to fetch the loaded drums.

b) Adding a new position: The user clicks *Add* and is brought to a different screen as shown in Fig. 9. The user is instructed to move the arms to the centre of the drum they want to configure, then select the corresponding MIDI note from the combobox. The user then clicks *Save*, which starts up a new thread that sends a command to DrumBot to save

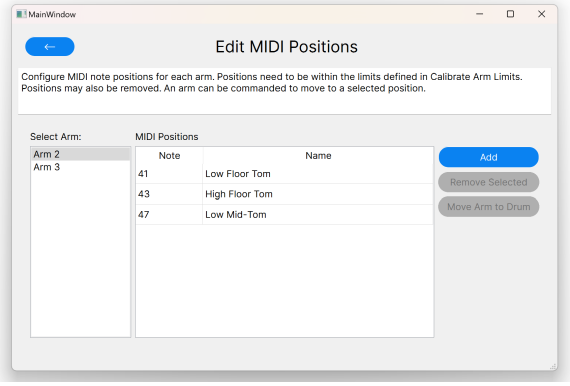

Fig. 8. DrumSet viewing the configured MIDI drum positions for Arm 2

a new drum position. The loading message popup is shown while DrumSet waits for a response. DrumBot validates the position, and if the position is invalid – if the arm is out of range, for example – then DrumBot includes error messages in its response, which DrumSet presents to the user. Otherwise, it confirms that the position was saved, and the user is returned to the main *Edit Midi Positions* page, where the drums are fetched again and displayed in the table.

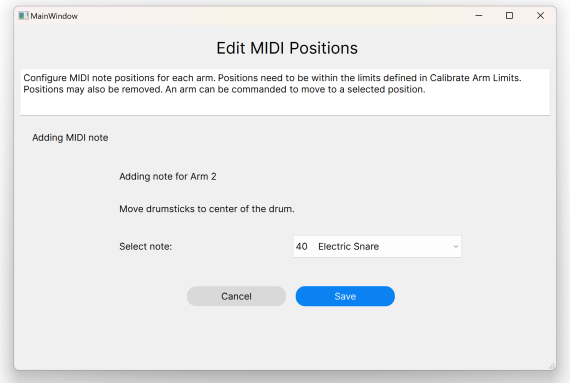

Fig. 9. DrumSet adding a new MIDI Position for Arm 2

c) Removing positions: The user selects a single position in the table, or multiple positions (see Fig. 20 in Appendix [5]) by clicking and dragging or holding down the `Ctrl` or `Shift` keys. The user removes the selected positions by clicking *Remove Selected*, which starts up a thread that sends a command to DrumBot to remove the specified positions. The loading message popup is shown while waiting for a DrumBot response confirming that the positions were removed.

d) Moving an arm to a position: If the user has selected exactly one position in the table, then the *Move Arm to Drum* button will be enabled. Clicking this button starts a new thread to send a move arm command to DrumBot, and the arm moves to the position. If DrumBot returns a response that there was an error in moving the arm, then a popup message alerting the user is displayed.

7) Latency Routine: (See Fig. 21 in Appendix [5]) The user enters the *Latency Routine* page, then clicks the button to begin the automated positional latency detection routine. This starts

up a thread that sends a command to DrumBot to start the latency routine. While the routine is running, a loading bar is displayed, giving an indication of how far along the routine is – when positional latency detection for one arm is complete, the loading bar progresses. Arms with more configured drum positions will take longer to complete, so the loading bar is not completely accurate. However, it is the closest representation that can be achieved with the current state of DrumBot.

8) Tighten/Loosen Grips: (See Fig. 22 in Appendix [5]). The user selects the arm they wish to configure using the arm selection list. Then, they click the button for the grip they wish to adjust. Once selected, the user is presented with *Tighten* and *Loosen* buttons. When the user clicks on a button, a new thread is created to tighten/loosen the grip, which sends the command to DrumBot to tighten/loosen the selected grip. If DrumBot confirms that adjusting has begun, the buttons are then hidden, and a *Stop* button is shown. When clicked, the thread sends "x" to DrumBot, commanding it to stop adjusting the grip. If DrumBot confirms that adjusting has stopped, the Stop button is hidden and the Tighten and Loosen buttons are shown again. The user can click the Finish button to go back to the main Tighten/Loosen Grips screen.

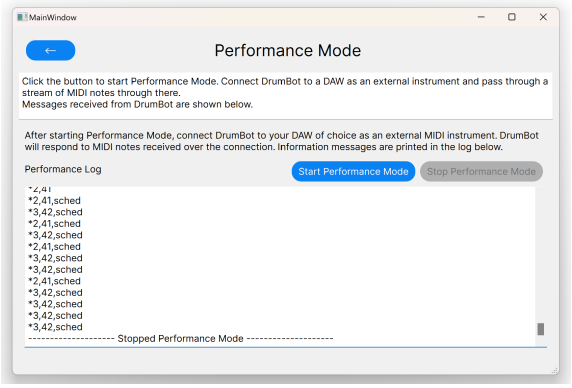

Fig. 10. DrumSet Performance Mode with log messages

9) Enter Performance Mode: (See Fig. 24 in Appendix [5]). The *Performance Mode* page contains a text display which displays the messages sent by DrumBot during Performance Mode. At first, this text display is empty. The user is instructed to start Performance Mode and connect DrumBot to a DAW (digital audio workstation) as an external MIDI instrument, which will allow DrumBot to receive and process MIDI notes.

The user clicks the *Start Performance Mode* button, which starts a new thread to send the command to DrumBot. The user stops Performance Mode by clicking *Stop Performance Mode*. When starting and stopping Performance Mode, DrumSet waits for a confirmation from DrumBot. If the confirmation is not received, then an error message is displayed.

10) Advanced Settings: (See Fig. 26 in Appendix [5]). When the user enters the *Advanced Settings* page, a new thread is created to retrieve the review information for the selected arm. The popup loading message is shown while the information is being retrieved. The user can review details for another arm by selecting it from the arm selection list, which starts up a new thread. The information is parsed and displayed in the corresponding tables in the GUI – the calibrated arm limits, loaded MIDI positions, positional latency, and shoulder PID coefficients.

It should be noted that this feature is highly dependent on DrumBot formatting the review data in a certain way. DrumSet expects a certain number of columns (except for the latency data) and cannot infer table headers. This means that if DrumBot had an update which changed the format of the review data, this would cause DrumSet to crash.

a) Editing the shoulder PID coefficients: (See Fig. 26 in Appendix [5]). Typically, a user would not want to edit the shoulder PID coefficients, but a user may wish to if they want to fine-tune the arm movements. To edit the coefficients, the user clicks on the *Edit* button, which enables the values in the table to be edited. The user clicks *Save* when they are finished editing, or *Cancel* to reset their changes. Clicking *Save* creates a new thread to send the update to DrumBot.

It should be noted that in the original plan, DrumSet would also allow the user to edit the values for calibrated arm limits, latency, and configured MIDI positions. Due to time constraints, this was not implemented. Cutting out these features is not a problem though, as the CLI and original GUI did not allow this functionality in the first place.

11) Disable Servos: (See Fig. 27 in Appendix [5]). Clicking *Disable Servos* creates a new thread to send the disable servos command to DrumBot. A popup message appears, confirming that the command was sent. If the command was correctly processed on the DrumBot side, then then the torque of all of the servomotors are disabled and the arms relax.

B. Challenges

There were several technical challenges in implementing DrumSet. These challenges were unforeseen due to limited personal experience with Arduino, but they were able to be overcome.

One was the challenge of safely communicating with DrumBot in a way that doesn't lead to error states. DrumBot is an embedded system and does not respond instantaneously to serial input, and during the delay between sending a command to DrumBot and receiving a response, unexpected input from DrumSet can often result in errors. Since threading is required to complete many tasks, DrumSet is put into an error state if it has two threads active and reading from the serial buffer at the same time. Therefore, mechanisms were put in place in the code to ensure only one thread is active at a time, and that while a thread is running, user input is prevented (with the loading message popup).

A challenge that also stems from the fact that DrumBot cannot send back an instantaneous response, is that DrumSet cannot always assume DrumBot's state, since its configured values could have changed since the last time the values were retrieved – for example, DrumBot's loaded MIDI positions could have been changed by the CLI application while DrumSet was still open. Having inaccurate data about DrumBot's state can lead to error states. So, to be on the safer side, DrumSet sends a new request to DrumBot every time a page is visited, such as when viewing the *Advanced Settings* information for an arm.

Overall, implementing DrumSet was challenging because it is required to interface with an external software-hardware system which is limited in how fast it can process input and return a response, making instantaneous – or even just fast – communication infeasible. DrumSet also does not directly control DrumBot (it just sends messages over serial), so extra care needed to be taken to avoid error states.

C. Conclusions

Overall, DrumSet was implemented very close to the design that detailed in Section IV. The required features planned in the design were able to be implemented, and styling was applied to make the GUI have a modern look and feel. Particular noteworthy features of DrumSet are *Calibrate Arm Limits* and *Edit MIDI Positions*, which are significant user experience upgrades from their original counterparts. While there are some aesthetic differences to the mockup, DrumSet possesses all required functionality. Technical challenges due to the embedded nature of the project were faced, but solutions were found and implemented. DrumBot's code lays a good foundation for further improvement of these solutions in future.

VI. Evaluation

This section discusses whether DrumSet met the requirements outlined in IV-A, and whether the usability issues in the original GUI were solved.

A. Meeting the Requirements

The requirements outlined in IV-A are reiterated here and treated like a checklist.

1) The software shall allow the user to ping the arms, kick pedal, and hi-hat – *Complete*.
   - The software shall inform the user when a ping is in process. – *Complete*. DrumBot displays the text "Pinging..." and disables the *Ping* button while a ping is in progress.
   - The software shall inform the user which components have responded to the ping. – *Complete*. DrumBot updates the ping status table as ping responses are received. Text next to each component informs the user whether the component is "Connected" or "Not Connected". If components have not been pinged, then the text "Not Pinged" will be displayed.
   - The software shall inform the user how long it took for a component to respond. – *Complete*. Next to the "Connected" text, the latency is displayed in milliseconds.
2) The software shall allow the user to calibrate the limits of the joints – *Complete*.
   - The software shall give the user clear instructions to calibrate a joint. – *Complete*. Instructional messages are displayed to guide the user.
   - The software shall allow the user to calibrate a specific joint. – *Complete*. The user may select "Shoulder", "Elbow", "Left Wrist", or "Right Wrist" to calibrate that joint only.
   - The software shall allow the user to calibrate all joints in one routine. – *Complete*. The user may select "Calibrate All Joints" and DrumSet will go through calibration for all the joints.
   - The user shall be able to cancel calibration once started. – *Complete*. The user may click the *Cancel* button to cancel the current calibration routine. This is possible when calibrating a single joint or when calibrating all joints in one routine. Cancelling calibration discards any values calibrated during the routine; calibrated values are not saved.
3) The software shall allow the user to configure MIDI drum positions – *Complete*.
   - The software shall allow the user to add a new MIDI drum position. – *Complete*. The user clicks *Add*, moves the drum sticks of the selected arm to the desired position, selects the MIDI note from the combobox, then clicks *Save*.
   - The software shall allow the user to remove a single configured MIDI drum position. – *Complete*. The user may click to select a MIDI position in the table, then click *Remove Selected* to remove the position.
   - The software shall allow the user to remove multiple configured MIDI drum positions at once. – *Complete*. The user may select multiple positions at once and remove them.
4) The software shall allow the user to run through the automated positional latency detection routine – *Complete*.
   - The software shall inform the user when the routine is running, and when it has finished. – *Complete*. DrumSet shows a message and loading bar while the routine is running. A message is shown when the the routine is complete.
5) The software shall allow the user to review configured values of DrumBot – *Complete*. Calibrated arm limits, configured MIDI positions, positional latency, and shoulder PID values can be viewed under *Advanced Settings*.
6) The software shall allow the user to directly edit the shoulder PID coefficients of the arms – *Complete*.
   - The user shall be able to cancel editing the shoulder PID coefficients. – *Complete*. The user many cancel editing the shoulder PID coefficients by clicking *Cancel*.
7) The software shall allow the user to move an arm to a configured drum position – *Complete*. In *Edit MIDI Positions*, the user may select a single configured MIDI position and click *Move Arm to Drum*.
8) The software shall allow the user to tighten and loosen the grips of the wrists – *Complete*. The user can do this in *Tighten/Loosen Grips*.
9) The software shall allow the user to disable the servomotors of the arms – *Complete*.

10) The software shall allow the user to put DrumBot into Performance Mode – *Complete*. The user starts and stops Performance Mode by clicking the buttons.
    - The software shall display the log of DrumBot system messages to the user while DrumBot is performing. – *Complete*. Serial output from DrumBot is displayed while Performance Mode is running.
11) The software shall allow the CLI application to function as normal – *Complete*. The DrumBot CLI application functions as normal. Its functionality is not affected by the implementation of DrumSet.
12) While the software is completing an action, the software shall prevent user input – *Partially Complete*. As discussed in Section V, a loading message pop appears while DrumSet is completing an action, preventing user input. However, this popup can easily be closed. A more robust solution should be implemented in future.
13) The software shall inform the user what each configuration setting does – *Complete*. Hint messages have been implemented for each configuration setting, shown when the setting is hovered over.

In summary, DrumSet fulfils most of the requirements outlined in IV, except for requirement 12. Preventing user input while DrumSet is busy is an important as to avoid the application going into an error state. Thus, it is paramount that a more robust solution is implemented in future. A possible solution for this is to have a dark, translucent overlay cover the GUI, preventing user input. Another solution is to implement a popup box, but remove the buttons to close it, and always ensure that it sits on top of the main GUI. Improvements such as these can easily be integrated into the code, as the current solution lays the groundwork.

B. Evaluation against Original GUI

DrumSet was evaluated against the original DrumBot GUI to assess whether it is an improvement and if it addresses the usability issues in the original GUI discussed in Section III.

1) Software hangs when pinging arms and disabling servos – *Addressed*. DrumSet does not hang when pinging the components or disabling the servos because it starts a separate thread to run these tasks. A separate thread is used for any long-running task or task that requires writing to or reading from the serial buffer.
2) Selecting an arm and confirming input requires text input – *Addressed*. DrumSet does not require any text input (unless they are editing the shoulder PID values), and instead follows design conventions and receives user input via buttons, comboboxes, list selections, and table selections. Receiving user input via graphical options where appropriate is more sensible for a GUI, and sets it apart from a CLI.
3) User must exit and re-select configuration option – *Addressed*. After finishing configuring a setting for an arm, DrumSet automatically brings the user back the the main page for that configuration setting. This reduces the number of clicks the user has to make, making the process more efficient.
4) Motor values are hidden during calibration – *Addressed*. During a calibration routine, the current positional value of the joint being calibrated is displayed to the user in a small text box. Although users will not generally need this information, it is now available if a user does need it, such as when debugging the system.
5) Unable to clear single MIDI positions from arm – *Addressed*. DrumSet allows the user to select single MIDI positions to remove, instead of having to clear all positions. The user can also select multiple positions to remove at once.
6) Unable to cancel action mid-configuration – *Addressed*. It is now possible for a user to cancel actions that were previously not able to be cancelled. For example, a user can now click *Cancel* to back out of calibrating limits for an arm. This has also been extended to arm calibration in the CLI application, and by extension the original DrumBot GUI, since it mirrors the CLI. Now, users have greater freedom of control.

All the issues identified in the original DrumBot GUI have been addressed in DrumSet. This shows that DrumSet does indeed improve the usability of DrumBot.

C. Evaluation against Sustainability Goals

One of the goals of this project was to ensure DrumBot is easily extensible by other developers. To achieve this, PyQt5 was chosen as the GUI framework because it is well-supported with resources and comes packaged with the Qt Designer studio, a tool which greatly assists development. This not only contributes to DrumSet's extensibility, but also the ease to maintain it, as it is usually easier to make modifications with a GUI than in code. DrumSet's design is also modular. Configuration features do not interfere with each other, ensuring that features can be updated or added without affecting other features. Each DrumSet feature that requires a thread uses a separate thread, creating separation of responsibilities. The GUI is easy to update to accommodate new functionality as well, thanks to Qt Designer.

Good documentation is essential for enabling other developers to work on DrumBot and DrumSet in the future. In fact, the lack of documentation at the start of this project made it difficult to learn how to configure DrumBot and understand the codebase, which hindered progress. Unfortunately, this project was not able to develop extensive documentation for DrumSet within the time constraints, so DrumSet's codebase is currently not easily understandable. However, some code comments have been written to guide developers and clearly outline what each part of the code is for. As discussed previously, a "hint box" was implemented for DrumSet to guide users in configuring DrumSet as well. In the near future, extensive documentation should be written for DrumSet and DrumBot.

In summary, DrumSet has addressed all the usability issues identified in the original DrumBot GUI. It also satisfies all of its design requirements, except for one where only a partial solution was implemented, but lays the groundwork

for it to be expanded upon on in future. DrumSet addresses software sustainability issues regarding code extensibility and maintainability, but more documentation should be written in the near future to ensure the project is easily picked up by future developers. Overall, DrumSet is an improvement of the original DrumBot GUI, therefore fulfilling the project goals of improving usability of DrumBot, making it fit for use by a musician for performance.

VII. Conclusion

DrumBot is a mechatronic robotic system capable of playing complex MIDI compositions on an arbitrary arrangement of drums. It consists of three robotic arms for playing the drums, and two self-actuated pedals for controlling a kick and hi-hat. For DrumBot to perform a composition, it must first be put through a setup process to configure the limits of the movement of the arms, set the positions of the drums, and calculate the latency when moving arms between drums. A user can do this using DrumBot's CLI or GUI programs which communicate with DrumBot over a serial connection. However, several significant usability issues were made apparent in DrumBot's GUI, making it unsuitable for use by a musician.

An evaluation was carried out on DrumBot's GUI to assess its usability, and several significant issues were identified. Many of the issues stem from the fact that the GUI goes against design conventions and does not utilise graphical features in its implementation, making it essentially behave in the same way as the CLI program. DrumBot also lacks documentation, making it particularly difficult for a new user to understand how to configure DrumBot. To address the issues of DrumBot's preexisting GUI, a new software, *DrumSet*, was developed. Developing a new solution instead of building upon the original DrumBot GUI was required because the original GUI is difficult to extend and maintain long term.

DrumSet gives the user the same essential functionality provided by the original GUI, with additional features and improvements to user experience. DrumSet's design was informed by widely used usability heuristics and the shortcomings of the original GUI. DrumSet is is easy to navigate, provides flexible functions like allowing the user to calibrate a singular joint or remove a single MIDI position, and guides the user by providing helpful messages. With DrumSet, DrumBot is now more accessible and usable by musicians for performance.

A. Future Work

Some areas of potential future work have been identified for both DrumBot and DrumSet.

1) Improvements to DrumSet: DrumSet's solution for preventing user input while waiting for a response from DrumBot can be improved upon. Some suggestions are made in Section VI-A, and DrumBot's code lays the groundwork for such a solution to be implemented.

Future work could also involve implementing editing and reviewing of the wrist proportional control coefficients (as discussed in IV-A1), as well as direct editing of the values of the arm limits, MIDI positions, and positional latency.

2) Orchestration Software: Now that DrumBot is in a state where it can be used by a musician for a performance, the greater goal of orchestrating a mechatronic band can be revisited. This would likely involve developing a software program that can interface with DrumBot, MechBass, and Azure Talos, and synchronise them with consideration to their individual latencies.

3) Finer Control of Movement: DrumBot and DrumSet could be extended to allow the user to have finer musical control of DrumBot. The "pitch bend" property of MIDI notes could be used to communicate positional information, such as playing more to the center or more to the edge of a given drum. This would extend the musical capabilities of DrumBot.

This project overcame the challenges of implementing a GUI that interfaces with a real-time embedded system, and produced a new software application that enables a user to configure DrumBot with ease. DrumSet has addressed the usability issues present in the original DrumBot GUI and has made DrumBot more accessible to users who are not familiar with the inner workings of DrumBot. By developing DrumSet, this project has achieved its goals in improving the usability of DrumBot, and as a result, DrumBot is now in a state where it can be used in a performance by a musician. DrumSet is also in a position where it can easily be extended by other developers in future.

Acknowledgments

I would like to thank Dale Carnegie and Jim Murphy for their support, guidance, and feedback on this project. They have been excellent supervisors and I have benefit from their expertise in mechatronics. I would also like to thank electronics technician Tim Exley, for his help when DrumBot needed repairs. Lastly, I would like to thank those in my cohort for their friendship and camaraderie.

References

[1] J. Nielsen, *10 usability heuristics for user interface design*, Apr. 1994.

[2] R. Bogue, "The role of robots in entertainment," *Industrial Robot*, Mar. 2022.

[3] A. Kapur, M. Darling, D. Diakopoulos, *et al.*, "The machine orchestra: An ensemble of human laptop performers and robotic musical instruments," *Computer Music Journal*, 2011.

[4] *The machine orchestra*, https://youtu.be/3spspyK28E4, [Online; accessed 29-May-2023], Sep. 2010.

[5] J. U, "Final report appendix," Appendix for Final Report.

[6] F. Olley, "The development of a novel mechatronic percussion system," Victoria University of Wellington, Kelburn, Wellington, New Zealand, Tech. Rep., Oct. 2022.

[7] O. Vallis, D. Diakopoulos, J. Hochenbaum, and A. Kapur, "Building on the foundations of network music: Exploring interaction contexts and shared robotic instruments," *Organised Sound*, vol. 17, 2012.