Firmware Design for a Mechatronic Chordophone

Alex Tait 300525521

***Abstract*—Azure Talos is a mechatronic chordophone that is theoretically capable of producing many of the expressive sounds and techniques of a human guitar player. However, despite its potential, Azure Talos was inherited in an incomplete state. To achieve full functionality, an entire firmware architecture is designed, implemented, and evaluated. A User Interface (UI) is then implemented to facilitate operation by non-mechatronic composers. The result is a fully functioning mechatronic chordophone that is capable of levels of expressivity beyond any comparable devices found in the literature.**

I. Introduction

THE development of advanced musical robots has been an area of significant interest and research at Victoria University of Wellington. Among these creations is 'Azure Talos', a unique and expressive mechatronic chordophone. Azure Talos has demonstrated the potential to perform some aspects of human guitar players [1]. However, Azure Talos is inherited as unusable for its intended purpose, and is, instead, dormant. The fundamental issue lies in the lack of documentation or access to the firmware installed on the microcontrollers, consequently blocking access to its actuators and restricting potential improvements. A new firmware design for Azure Talos developed as part of this research unlocks additional expressive capabilities of the robot.

Expressivity, in music, is a term with numerous and diverse meanings. In this project, we use the term to refer to several specific deliberate actions of a human guitar player. Specifically, we refer to the impact of auditory parameters in music performance, including loudness, intensity, tempo, and frequency spectrum.

Sustainability is prioritised within its new firmware design. It achieves this by giving a new purpose to hardware that might otherwise end up as electronic waste. Moreover, the design not only extends the lifespan of the current hardware, but also addresses hardware issues, thereby minimizing the necessity for hardware upgrades. The open-source documentation of the design ensures long-term viability by enabling firmware reuse and adaptation, enhancing product maintainability.

Azure Talos is a six-string mechatronic guitar designed to play the equivalent of the first six frets of a standard guitar (notes ranging from E2 to A4).

It is required that a new firmware design for Azure Talos:

1) Facilitates access to all actuators.
   The inherited system presents non-functional actuators, therefore, accessing these through the firmware to confirm their functionality will allow the system to be used for its intended purpose.
2) Consists of minimal hardware changes.
   With a commitment to sustainability, the existing hardware of Azure Talos is to be utilised to reduce electronic waste.
3) Ensures the system is usable.
   Azure Talos is designed to facilitate expressive musical performances by composers, ensuring not only usability from a composer's standpoint, but also accessibility and versatility for future firmware developers.
4) Implements successful note mapping.
   To produce the composer's instructions when playing a musical score, musical notes must be played accurately.
5) Showcases a range of expressive parameters.
   Azure Talos is a unique device due to its potential to play expressive parameters.

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

Azure Talos was designed to enable a variety of performance modes which can afford this expressivity. The following subsections detail these expressive techniques, including (where feasible) quantifying metrics.

A. Tremolo

Tremolo is the quick and continuous reiteration of a single pitch, often performed by plucking one string over and over [2]. The human ear struggles to distinguish timing differences beyond 100 ms (10 picks per second), detailed in Section II-A3. Therefore, Azure Talos must pick at least 10 times per second to achieve tremolo.

B. Palm Mute

Palm muting is a guitar-playing technique where the strings are partially silenced by the picking hand at the moment of striking [3]. To create this muted effect, the palm is positioned just in front of the guitar's bridge, generating a distinctive tone that is reduced in volume and sustain. For Azure Talos to execute palm muting successfully, a palm-muted pick should produce a note that is sustained for less than the normal pick whilst diminishing its intensity.

C. Ghost Note

A ghost note, also referred to as a 'false note', is a muted or dampened note that has rhythm, but often no discernible pitch [4]. To achieve ghost notes, Azure Talos must produce a pick that is of greater non-harmonic content than a normal pick event.

D. Slide

A slide is a technique where a guitarist physically slides their fretting finger from one note to another. This produces a rapid run of notes without differentiating or accenting each

of the intermediate notes [5]. Therefore, to achieve a slide, Azure Talos must be able to show a continuous increase or decrease in frequency over time, from one note to another, with no interruptions.

E. Vibrato

Vibrato is a wavering variation in pitch produced by a guitarist physically bending a fretted note up and down rapidly [6]. To achieve vibrato, Azure Talos must be able to play a changing audible difference (discussed in Section II-A3).

F. Pitch Bend

A pitch bend is a technique where a guitarist's fretting finger is used to bend the string up or down (towards either side of the neck) [7]. The bend creates an increase of tension on the string and, therefore, an increase in pitch. For Azure Talos to achieve a pitch bend, it must be able to increase its pitch by greater than 3 Hz (detailed in Section II-A3)) when clamping (a term for actuated engagement with the string) in the same position.

These requirements will guide the firmware design for Azure Talos, along with preliminary research.

II. Related Work

Exploring existing literature concerning firmware setups for mechatronic chordophones will provide the necessary context and insights to inform the design and development of the new firmware for the inherited machine. Understanding the hardware limitations of the machine itself will further help design the final firmware implemented.

A. Literature Review

1) Introduction

Mechatronic chordophones are musical instruments that combine stringed instruments with electronic and mechanical components. This study examines firmware structures, musical theory, communication protocols, and existing mechatronic chordophone systems.

2) Firmware Structures

Firmware structures determine how the software operates and interacts with hardware. Selecting the appropriate firmware structure for Azure Talos can assist in meeting the system's functional performance requirements, outlined in Section I).

Azure Talos is currently configured as a decentralized machine, wherein each string unit operates independently with its own microcontroller, outlined in Figure 1.

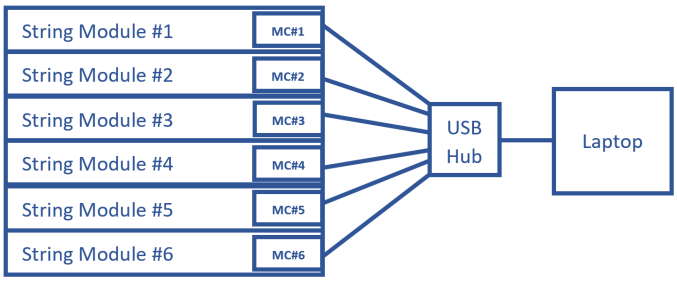

Fig. 1. Decentralized setup of Azure Talos, where 6 string modules have individual Teensy 3.5 microcontrollers

Decentralized architectures provide the advantage of module independence, ensuring that if one module encounters a problem, the others can continue functioning [8]. Expanding the system with additional modules is straightforward, as it involves replicating existing configurations. However, managing decentralized systems can become intricate when modules require different sets of information.

Alternatively, Azure Talos could be configured as a centralized machine, like the setup displayed in Figure 2, where all string modules are interconnected through a central server, for example, a motherboard. [8].

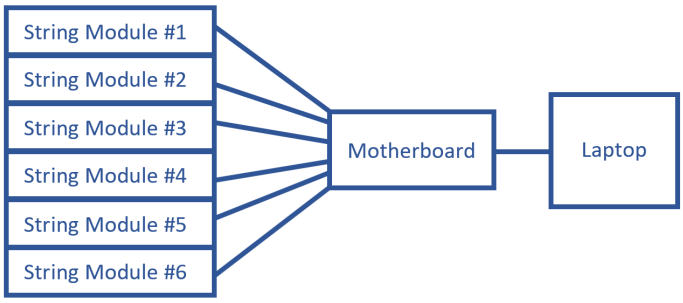

Fig. 2. Alternative centralized setup of Azure Talos

Centralized architectures simplify management and facilitate easier updates since all of the components are consolidated. However, the downside is that centralization introduces a single point of failure, meaning that if the motherboard experiences any issues, it could disrupt the operation of all connected string modules [8].

The choice of machine configuration influences the determination of the firmware structure for Azure Talos. To meet the usability requirement stated in Section I, it must play the incoming instructions with minimal delays to produce the intended output from the composer.

Possible firmware structures for real-time systems include sequential structures, state machines, and event-driven architectures.

A sequential firmware structure in a real-time system refers to the organisation of the system's firmware in a way that prioritises a sequential execution of tasks or processes [9]. Within the main loop, tasks are organised and prioritised based on their importance and timing requirements. Real-time tasks often have deadlines and need to be executed within specific time intervals. The sequential structure ensures that higher-priority tasks are executed before lower-priority ones. In a

sequential firmware structure, tasks can be implemented as non-blocking or blocking tasks, where non-blocking tasks do not cause the main loop to wait for their completion.

A state machine is a firmware structure based on distinct states. Each state represents a specific mode of operation. The advantages of using state machines include predictable behaviour, modularity, and real-time responsiveness [9]. However, managing numerous states and transitions can become complex, limiting flexibility in dynamic scenarios.

Event-driven structures revolve around responding to events or interrupts. These events, triggered by hardware inputs, dictate the firmware's execution flow, making it responsive to external inputs [9]. Event-driven architectures offer advantages like quick responsiveness, scalability, and power efficiency. However, their complexity can pose challenges in terms of sequencing and debugging.

The firmware structure chosen for Azure Talos will affect its real-time performance response, therefore, the selection should allow for timing constraints to be considered.

3) Music Theory

In musical evaluation, two key concepts are the Just-Noticeable Difference (JND) and cents. The JND represents the smallest change in a musical attribute that the human ear can perceive at least half the time [10]. For example, in terms of pitch, the JND typically amounts to ±3 Hz, or 10 cents, marking the threshold for perceiving changes in pitch [11]. One cent equals 1/100th of a semitone, indicating the smallest detectable shift in pitch. Similarly, in the case of sound amplitude, the JND is approximately 1 dB, enabling the detection of even minor volume variations. These perceptual thresholds can be affected by factors such as frequency content and controlled conditions. In terms of the JND for the speed perception between two notes, humans begin to struggle to distinguish a difference of 100 ms, or 10 notes per second [10]. Consequently, these nuances must be considered to ensure Azure Talos' performance aligns with standards of human auditory perception. Furthermore, these concepts can assist in evaluating the quality assessment of Azure Talos' performance.

The position at which each note will be clamped on Azure Talos follows the same ideology as frets on a guitar. In standard E guitar tuning, each fret on the guitar's neck represents a half step, or semitone, in the chromatic scale. As a guitar player progresses up the fretboard, the pitch of the notes played increases, with each fret representing a higher note in the musical alphabet. As pitch increases, the distance between the notes becomes narrower due to the logarithmic increase in frequency within each octave. This same relationship will assist in the note mapping calculations of each string unit, where the distance between the frets/clamping positions will get smaller and smaller as the clamp travels up the chassis. This will assist in achieving the accurate note-mapping requirement highlighted in Section I

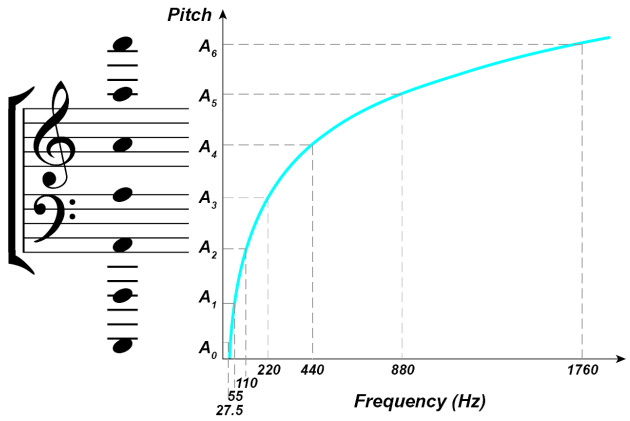

Fig. 3. Logarithmic increase in pitch and frequency, source [12]

4) Protocols

A communication protocol allows devices to transmit data to each other [13]. Understanding protocols that could be used for Azure Talos is needed as this is the communication pathway that will allow access to the actuators of the system, a requirement outlined in Section I). The ideal protocol for Azure Talos should include features that allow the expressive parameters, stated in Section I), to be communicated.

Open Sound Control (OSC) and Music Instrument Digital Interface (MIDI) are two commonly used protocols in the realm of digital music communication.

a) OSC

OSC is a protocol utilised for communication between multimedia devices and software. Its messages, written in plain text, include addresses and arguments to transmit information such as control parameters, synchronisation cues, and multimedia events. It provides higher precision and resolution for data transmission compared to MIDI and supports a range of flexible data types, including custom data types, which go beyond data transmission of traditional musical parameters. OSC is designed for network communication, making it well-suited for distributed systems and remote-control applications where devices communicate over local networks or the internet [14]. However, there are also disadvantages to consider. While OSC has gained popularity in the music domain, it lacks standardisation which can lead to compatibility issues. OSC messages can have high processing overhead in comparison to MIDI which can impact network bandwidth and latency, which needs to be considered for a real-time machine. Implementing and understanding OSC can also be complex as it requires knowledge of network protocols and programming techniques specific to OSC.

b) MIDI

MIDI is a well-established standard for transmitting musical data between electronic instruments and offers many advantages. It provides musical-specific features (like note-on/off messages) making it more suitable for tasks related to music communication in comparison to OSC. Another feature includes Control Change (CC) messages, which enable controllable functions, like volume (CC 7). This is a beneficial feature that could enable expressive parameters, such as those required in Section I.

MIDI has low overhead in comparison to OSC, enabling efficient transmission of musical data with minimal latency [14]. Additionally, MIDI has achieved widespread standardisation, ensuring compatibility between different MIDI-enabled devices, software, and interfaces including Digital Audio Workstations (DAWs).

A DAW is a software platform that acts as a UI for MIDI controls. An example of this is Ableton Live. A MIDI device can be connected to a computer and the Ableton Live application allows for a MIDI track to be created and communicated to the connected MIDI channel. MIDI's capability to work with a DAW can simplify the process of connecting and integrating various musical instruments, controllers, and software, which can enhance usability, a further requirement stated in Section I. MIDI is also commonly used by the target user group for this project. As OSC is predominantly used in an academic and experimental context, MIDI is a more practical choice for widespread usability [14].

However, there are also disadvantages to the MIDI protocol. Its 31.25 Kbaud baud rate is slow compared to modern serial protocols, limiting its suitability for transmitting large amounts of data [15]. MIDI messages are not error-checked and are not verified during transmission, making them susceptible to undetected errors which can result in missed or corrupted messages. This can, however, be overcome by using a USB device that incorporates error-checking features. Moreover, MIDI's CC messages have a limited resolution, typically ranging from 0 to 127 which can restrict the control precision.

Overall, the communication protocol for Azure Talos should be chosen based on what can achieve the requirements specified in Section I. This can be investigated through other expressive mechatronic chordophone systems.

5) Mechatronic Chordophones

An analysis of existing expressive mechatronic chordophones helps determine how this project is to be approached. The inherited system is designed for expressivity, but there are expressive techniques that are non-functional. Among the well-documented examples in this domain, Mechbass and Guitarbot are examined.

Mechbass is a mechatronic bass guitar designed to surpass human capabilities and consistency [16]. With its numerous degrees of freedom, Mechbass executes expressive techniques such as pitch bends and damping control. However, its pitch shifter's reliance on a non-rolling clamper limits its ability to perform sliding techniques or play microtonal pitches. It utilises a MIDI interface, compatible with DAWs or music programming environments like ChucK [16].

Guitarbot (created by LEMUR) is a mechatronic slide guitar consisting of four independent string modules. It offers a range of expressive possibilities, allowing control over playing speed and the creation of polyrhythms (simultaneously playing contrasting rhythms). However, it is limited when performing techniques that rely on applying pressure to the strings (like vibrato, and pitch bends) due to the design of its slide which cannot be disengaged from the strings. Guitarbot also employs the MIDI protocol. This allows for individual or ensemble playing options, through MIDI keyboards and other compatible devices, without additional software or specialised equipment [17].

Surveying these mechatronic chordophones highlights the popularity of the MIDI protocol within the realm of mechatronic instruments, and reveals that there is a higher rate of mechanical innovation for these devices, rather than software or firmware innovation. This project can showcase the unique opportunity to create firmware that can simplify access to playing basic notes, whilst allowing the ability to access expressivity.

6) Conclusion

To meet the requirements specified in Section I, including minimising hardware alterations, this project will maintain Azure Talos as a decentralized machine. When it comes to structuring the system, there are inherent trade-offs to consider in the context of a real-time machine. While various approaches such as sequential, state-based, and event-driven designs can be explored, the use of a state-machine design stands out as the most conducive to fulfilling the majority of requirements.

Mechbass and Guitarbot are examples of mechatronic chordophones that showcase some expressivity. These instruments incorporate the MIDI protocol into their systems, which demonstrates that MIDI is an effective communication protocol for an expressive mechatronic chordophone, and OSC has had no such up-take.

Azure Talos utilises the Teensy 3.5 microcontroller system to control each string module. One notable advantage of the Teensy is its ability to function as a USB MIDI device, which is supported by Arduino Teensy MIDI libraries (detailed in Section II-B4). Given the MIDI-supported features of this microcontroller, and the MIDI CC feature that can be used to implement the expressive parameters required, the design of the Azure Talos interface will use the MIDI protocol.

Azure Talos surpasses Mechbass and Guitarbot in terms of expressivity, partly due to its greater number of actuators. However, it cannot be optimized to play to its full potential without access to its firmware.

B. Tools and Methodology

Upon reviewing approaches to mechatronic chordophone systems in Section II-A, Section II-B helps refine the necessary tools to help fulfill the requirements of Section I.

1) Firmware

Firmware, for a mechatronic chordophone, refers to the specialised software embedded in the instrument's microcontroller that enables low-level access to the actuators, allowing them to be controlled and manipulated via a higher-level UI. The firmware needs to support the MIDI communication protocol for interfacing with other devices or systems. Additionally, the firmware needs to provide a means of interacting with the instrument's hardware components. The firmware has to recognise incoming MIDI messages and translate these into accurate actuator movements. To develop the firmware effectively, it is important to understand the available hardware components.

2) Exploration of Hardware

a) System Overview

Azure Talos comprises six separate string modules that must communicate in parallel. To facilitate this, these modules are interconnected via a USB hub, as depicted in Figure 4, and are recognised by a laptop as separate MIDI channels.

To interact with this mechatronic instrument and meet the usability requirement stated in Section I, users need to establish a means of transmitting input commands to the system. Typically, this can be achieved through MIDI hardware or a DAW like Ableton Live.

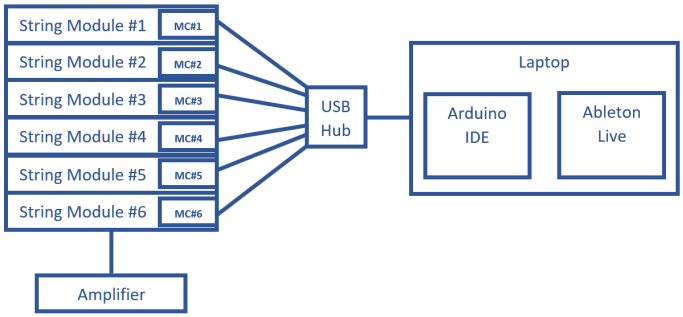

Fig. 4. Final setup of Azure Talos

b) Actuators

The Azure Talos hardware consists of four parts that incorporate actuators for specific functionalities.

Firstly, the picking mechanism uses a NEMA23 stepper motor to rotate the pickwheel. Manual alignment of the pickwheel is necessary during the startup process of Azure Talos as it requires a reference point to accurately monitor its position during performance.

An additional NEMA23 stepper motor enables the raising and lowering of the pickwheel. This motor incorporates a limit switch which can be utilised in the firmware upon startup to establish its home position.

The robot arm relies on a Dynamixel servomotor to move the carriage along the chassis. The carriage holds the clamping mechanism which is controlled by an MKS DS95i Micro Tail Rotor Servo. This servo rotates the clamp to adjust the clamping rod or damping sleeve pressure against the string.

The palm muting mechanism also uses an MKS DS95i Micro Tail Rotor Servo. This servo rotates the silicone damper from its idle position (parallel to the string), to firmly pressing on the string.

Prior to developing firmware functionality for the actuators, the physical limitations must be defined. For the Dynamixel, the position is printed on a continuous loop, and the attached carriage is manually moved to both ends of the string to determine its maximum limits. This is to ensure the Dynamixel does not exceed its safety boundaries or extend beyond its physical capabilities, which could lead to component damage or overextension.

TABLE I
DYNAMIXEL POSITIONS AT THE MAXIMUM CARRIAGE DISPLACEMENT ALONG THE CHASSIS

String	Nut-end Max	Bridge-end Max	Difference
E	259	1077	818
G	222	1051	829
D	95	922	827
A	198	993	795
E	345	1142	797

The test results, shown in Table I, reveal that despite identical setups for each string module, the Dynamixel actuators yield varying position readings at the same physical distances. The relationships between these maximum positions are also inconsistent, which indicates varying note position calculations for each string module. This additionally suggests that each string module's Dynamixel may not be the only actuator requiring different setpoint values to achieve equivalent actions. Consequently, the nuances in the firmware versions will extend beyond just string notes.

It is essential to maintain usability, as this is a requirement outlined in Section I. Declaring these altering values as global variables will streamline the process for future firmware developers, allowing them to make changes in a centralized location, rather than having to search through the entire program to adapt to each string module.

Furthermore, printing the Dynamixel's position in a loop revealed occasional null values as the output from some actuators, which will need to be considered when interpreting the current position of the Dynamixel in the firmware.

c) Pin Mapping

To enable communication with the actuators, pin mapping is required. However, due to the absence of documentation for Azure Talos, as previously noted in Section I, with no information available regarding the printed circuit board (PCB) or its corresponding pins, a manual alignment process is undertaken. This procedure involves the use of an unpopulated PCB, shown in Figure 5, to manually trace each connection from the microcontroller pins to the actuator pins. This method allows access to all of the actuators within Azure Talos, thereby addressing another requirement highlighted in Section I.

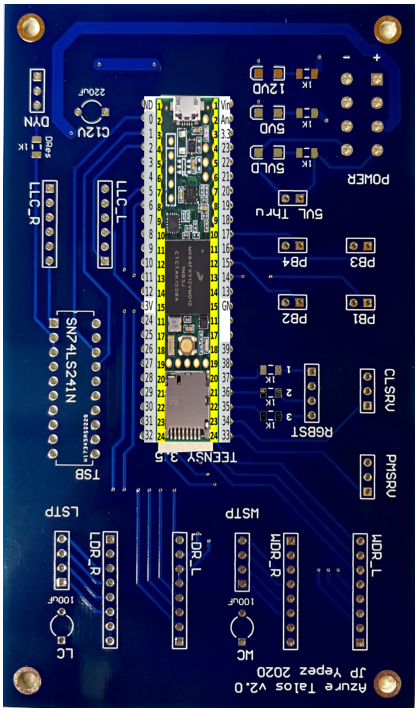

Fig. 5. Unpopulated PCB used for manual pin mapping

TABLE II
RESULTING MICROCONTROLLER PINS FOR THE ACTUATORS

Teensy Pin	Actuator
2	Dynamixel
4 - 9	Lift
14	Limit Switch
25 - 30	Pickwheel
33	Palm Mute Damper
34	Clamper

The firmware needs to coordinate the initialised actuators of Azure Talos to produce the desired musical tones, considering factors such as timing, velocity, and dynamics, along with being able to implement the various expressive techniques stated in Section I. To write the firmware, an appropriate development environment has to be chosen.

3) Development Environment

The development environment needs to support the Teensy 3.5 microcontroller on each string module, provide a dedicated workspace for the firmware development tasks, ensure compatibility with necessary tools (like a DAW), and support version control. This is achieved through the Arduino IDE with the Teensyduino add-on, which is set up on a Windows operating system and stored on the ECS Mechatronics Gitlab.

4) Software Libraries

Programming languages are often set up with a range of built-in libraries and functions to improve programming efficiency. The firmware can utilise various libraries to assist the control over Azure Talos' actuators. The Arduino IDE has a Teensduino add-on available to assist with communicating MIDI instructions, and may be implemented when retrieving MIDI inputs [18]. The DynamixelSerial Library may be used to control the Dynamixel servos on each string module as this allows specialised data to be read directly from the motor [19]. The AccelStepper Library may be used to control the stepper motor's acceleration, speed, and position control [20]. Additionally, the Bounce Library can enable the activation of the limit switch for the pickwheel lift [21].

Understanding the work surrounding Azure Talos provides a foundation for the development and design of the firmware. Azure Talos was inherited unusable with no firmware documentation, so its design is realised without a preexisting foundation to build upon.

III. DESIGN AND IMPLEMENTATION

Mechatronic instruments need a process to convert input commands into specific actions for playing music. The instrument's firmware handles several critical tasks:

1) Starting up the system and initialising actuator control.
2) Receiving and interpreting input commands (MIDI messages).
3) Processing input data and mapping it to output data.
4) Driving the actuators to produce actions.

Figure 6 illustrates the firmware structure of Azure Talos. The system receives MIDI messages from the computer, comprising channel data, note data, and velocity data. The firmware must interpret this MIDI data and generate actions for the actuators of each string module.

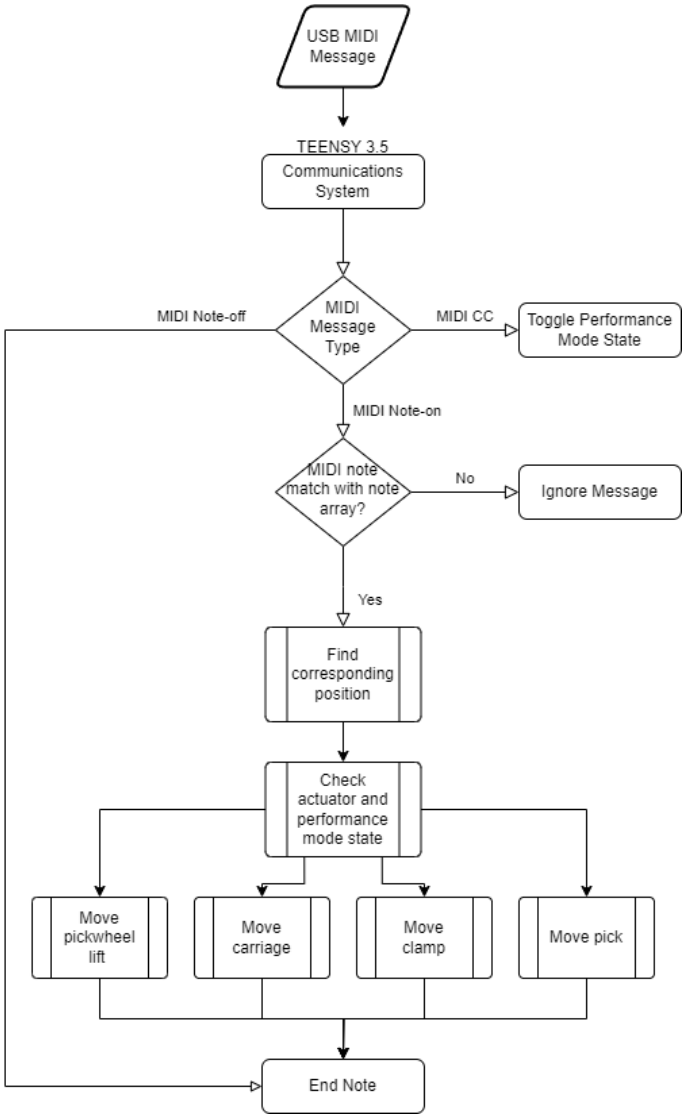

Fig. 6. Azure Talos' final firmware flowchart

The initial firmware design followed a sequential structure. This allowed for the incorporation of each performance mode, fulfilling the requirement in Section I, along with the implementation of playing random notes within the boundaries of the string as a MIDI note-on message was received. This approach was initially selected due to its expediency and simplicity in ensuring access to all actuators. While it met requirements and allowed parallel operation across each string module, it introduced delays, posing a challenge when handling fast musical compositions with incoming MIDI note-on messages before the sequential routine was completed. This trade-off impacted usability, a further requirement. Consequently, the firmware's structure was transitioned to a state machine model (with some event driven functions to receive the MIDI data). This decision offered both advantages and drawbacks, which are elaborated on in Section II-A2, but was ultimately made to better meet the usability requirements due to the reduction in latency.

A. Task One: System Startup

Upon system startup, the actuators are initialised and their limits are set, the note array for the string module is retrieved, the pickwheel lift is homed with the limit switch, detailed in Figure 7, and the carriage is moved to the middle of the string module.

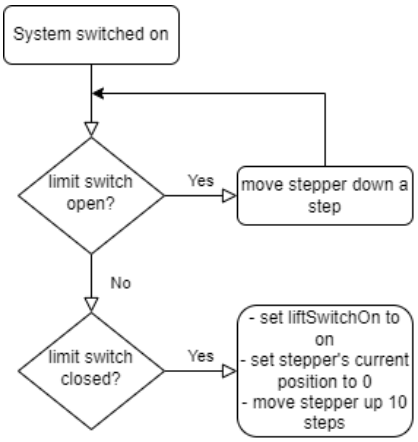

Fig. 7. Flowchart of the pickwheel lift startup

B. Task Two: MIDI Note-on

As a MIDI note-on message is received, the note value and velocity are stored using the `setHandleNoteOn()` function within the Arduino MIDI Library [22]. The note position is then retrieved from an array, detailed in Section IV-B

1) Control Change

Azure Talos utilises MIDI CC messages to enable its expressive performance modes. As displayed in Table III, performance modes with on/off functionality can be triggered by sending a non-zero value with the MIDI CC message and can be switched off with a value of zero. Performance modes based on intensity can be adjusted using the MIDI CC range of 0 – 127.

TABLE III
AZURE TALOS' PERFORMANCE MODES

Performance Mode	MIDI CC	Type
Tremolo	20	On/Off
Palm Mute	21	On/Off
Ghost Note	22	On/Off
Slide	23	Range (0–127)
Vibrato Strength	24	Range (0–127)
Vibrato Speed	25	Range (0–127)
Pitch Bend	26	Range (0–127)

The MIDI CC data is retrieved using the `setHandleControlChange()` function within the Arduino MIDI Library [22], consisting of three bytes; channel, control, and value. The `midiCC()` function uses these three bytes to update the state of each performance mode within the firmware.

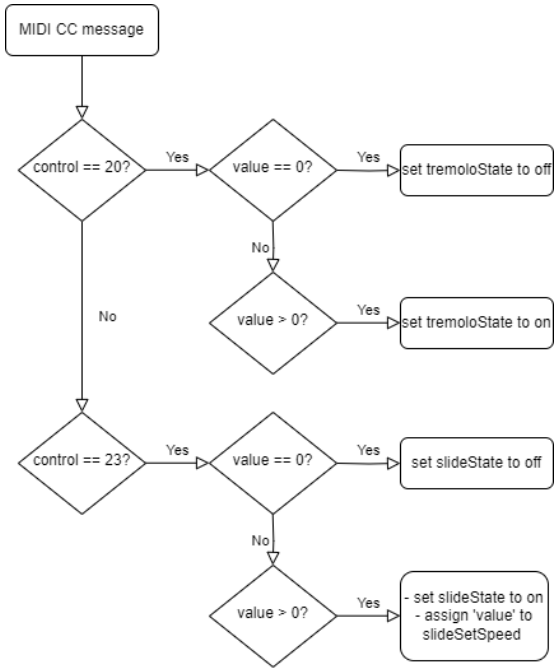

Fig. 8. Flowchart of MIDI CC message check

Figure 8 illustrates the processing of incoming MIDI CC message data. The initial check represents the process for the on/off performance modes (MIDI CC 20, 21, and 22). The latter check is the process for intensity-dependent performance modes (MIDI CC 23, 24, 25, and 26). During this step, the value data (0-127) is temporarily stored before it undergoes a mapping process.

C. Task Three: Mapping

Once the CC data has been received, the 0 - 127 input range must be mapped to the output required by the actuators. This is achieved through the Arduino `map()` function. This is only relevant for the intensity-based performance modes as the 'value' byte precipitately controls the data magnitude. An input of zero results in the selected performance mode being switched off, therefore, the range of 1 - 127 is mapped.

For the slide performance mode (CC 23), this range is mapped to 1 - 400. This range was chosen due to the speed tests conducted in Section IV-C.

The vibrato strength, controlled by CC 24, is represented on a scale from 0 to 6, with 0 indicating maximum string pressure, and 6 indicating a light touch. These limits are determined through manual testing to ensure that the clamper maintains contact with the string within this range.

For the vibrato speed (CC 25), this range is mapped from 20 and limited to 300 for mechanical noise issues.

The pitch bend amount (CC 26) is mapped within a range from 10 to 0. In this mapping, zero represents the maximum position at which the clamping action can secure the string, while 10 positions the clamp gently against the string. This extended range, compared to vibrato strength, is due to the pitch bend action being less hardware-intensive than vibrato, making it a safer option for achieving more extreme string bends with little mechanical noise.

Additionally, the mapping function can be used when setting the velocity of each incoming MIDI note. This is also a 0 - 127 input range from the third byte of the incoming MIDI message. This range is mapped from 1000 - 0, which is the set range of the lift due to physical hardware constraints.

D. Task Four: Actions

When a MIDI note-on message is received, Azure Talos executes a set of actions performed by its actuators. These actions are executed based on the state of the actuators, following a state machine firmware architecture.

1) Velocity Pick

The first action involves adjusting the velocity at which the string is to be picked. This is accomplished by adjusting the pickwheel lift to the specified dynamic level when the `moveLift()` function is activated in response to a MIDI note-on signal. The position is updated in the variable, `currentLiftPos`, to keep track of the lift's current position. This provides continuous monitoring of the motor's location within the 0 - 1000 range. It is unnecessary to calculate if the lift must move upward or downward based on its previous position, as the `runToNewPosition()` function moves the lift according to its position within the designated boundary range, rather than considering the positional difference. This enhances the usability of the machine as the user does not need to manually interfere with the pickwheel's height.

2) Move-Clamp-Pick

When there are no performance modes enabled, Azure Talos follows the move-clamp-pick routine.

- Move: The robot arm slides the clamping carriage to the target position on the string.
- Clamp: The clamping mechanism applies force upon the string.
- Pick: The pickwheel plucks the string to create sound.

a) Move

The `moveDyna()` function is called when the lift is at its target position. This uses the DynamixelSerial `moveSpeed()` function, which moves the arm to the note target position at the specified speed, discussed in Section IV-C.

b) Clamp

The `moveClamp()` function is called when the carriage has reached its target position. If the ghost note performance mode is enabled (CC 22), it rotates the damping sleeve in contact with the string. If either of the vibrato performance modes (CC 24 and 25) are enabled, the `doVibrato()` function is called which uses the MillisTimer `millis()` function, so it can execute in parallel to other actions. For the duration of the MIDI note-on message, the clamp angle (mapped by vibrato strength) switches from position zero (clamping the string) to the desired strength/rotation at the velocity mapped by vibrato speed.

If CC 22, 24, and 25 are not enabled, or are at a value of zero, the clamper is rotated to the zero position, where the clamping rod is in contact with the string.

c) Pick

The `movePick()` function is called when the clamper is in the engaged position. If the tremolo performance mode (CC

20) is enabled, the pickwheel moves at a rate faster than 10 picks per second for the duration of the note-on message. The pickwheel position is updated to ensure realignment of the pick after a tremolo pick.

If CC 20 is not enabled, or is at a value of zero, the pickwheel rotates forward to pick the string once.

3) Slide

If the slide performance mode is enabled (CC 23), the actions detailed in Section II-B2b are executed, but in a clamp-pick-move routine, rather than a move-clamp-pick routine. This produces a continuous increase or decrease in frequency from one note position to another, which refers back to the original requirement stated in Section I.

4) Palm Mute

When the palm mute performance mode (CC 21) is enabled, the palm mute servo is rotated to its engaged position where the damper comes in contact with the string. When CC 21 is disabled (set to a value of zero), the palm mute servo returns to its idle position where the damper is not in contact with the string. As this instruction does not rely on the state of any other actuator, the action is executed within the `midiCC()` function.

5) MIDI Note-off

When a MIDI note-off message is received, the clamp is rotated to a dampened position where the damping sleeve makes light contact with the string. This is to stop the sustain to keep to the composer's intended duration of the note. Additionally, the pickwheel motor is turned off due to its high-pitched mechanical noise when waiting for an instruction.

E. Note Mapping

The clamping mechanism is expected to enable playing within the JND of ±10 cents of a target pitch as outlined in Section II-A3.

Common guitar tuning encompasses its six strings tuned to Standard E tuning, as illustrated in Figure 9. This figure visually represents the E2 to A4 range that Azure Talos is designed to cover. In MIDI notation, this range translates to notes 40 through 71. Within this range, each string offers a different set of notes.

The firmware has been designed to accommodate the playing of these notes by tuning the instrument to Eb, which is one semitone below the standard E tuning. This adjustment is necessary because of the hardware challenges associated with the high E note on its respective string module. While a typical guitar neck falls within the length range of 609.6 mm to 647.7 mm [23], the Azure Talos chassis design extends the string over 889 mm. This increased distance over which the string is stretched creates additional tension, often leading to the high E string snapping. As a solution, the instrument is tuned to Eb, which helps alleviate the tension on this string. Consequently, the first clamped note becomes the major note rather than the open (unclamped) note. It is worth noting that the current firmware does not support open notes, although the possibility of incorporating them is discussed further in Section V-A.

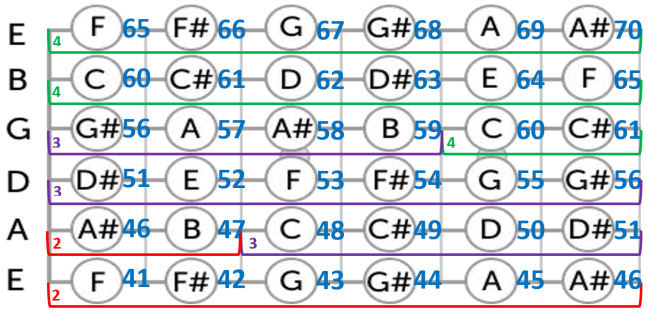

Fig. 9. Expected MIDI notes for each string module, represented as guitar strings

Azure Talos' pitch shifter relies on a robotic arm to slide the clamping carriage along the string. The carriage's position, denoted as X_t, is determined by both the arm's length, L, and the servomotor's angle, θ. This relationship is expressed as:

$$X_{\mathrm{t}} = 2L\cos(\theta) \tag{1}$$

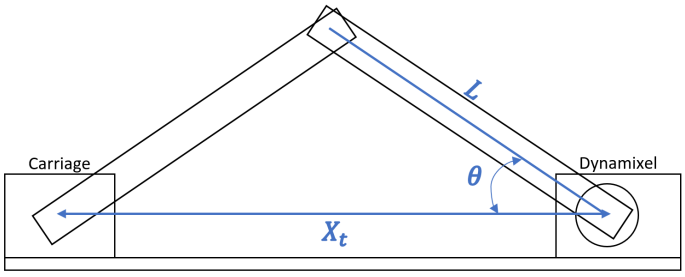

Fig. 10. Carriage movement variables for Eq. 1

As the Dynamixel rotates, it moves the carriage to different positions (X_t) along the string. Adding an extra distance of 11 mm to X_t accounts for the additional engaged clamp's displacement from the carriage. This results in the final angles required for the Dynamixel to stop at its various "fret" positions on the string, similar to a standard guitar, outlined in Section II-A3. This relationship is inherently non-linear, leading to variations in the rate of carriage displacement along the length of the string.

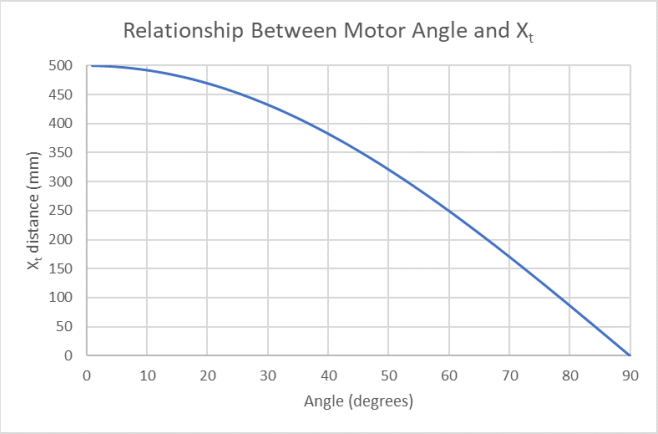

Fig. 11. The non-linear relationship between the Dynamixel angle and the resulting carriage distance

A fret position calculator [24] is a tool that calculates these theoretical fret positions based on the string length from the nut to the bridge. This is employed to determine the Dynamixel positions which are derived from the distances (X_t) for each individual string module.

The system maintains individual arrays dedicated to each string module to store the resulting note positions. For each firmware version running on various microcontrollers, the incoming MIDI note is traversed through the respective string module's input note array. If a match is found, it is used as an index to retrieve the corresponding note position of the output array, as seen in Figure 12.

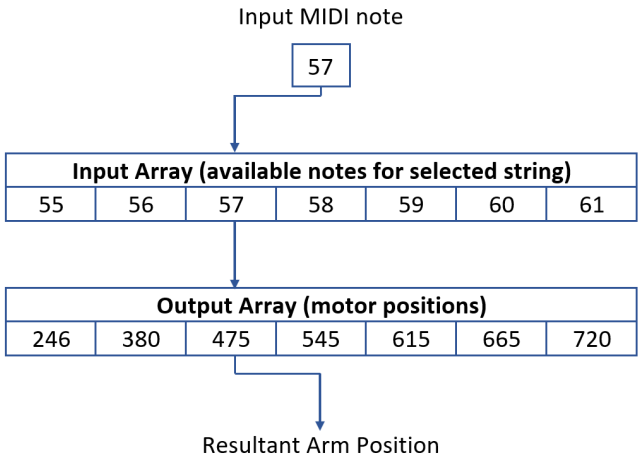

Fig. 12. MIDI note to Dynamixel position arrays

If a match is not found, the instruction gets ignored and the carriage stays in its previous position. Therefore, the user must compose each string module based on the notes within the designated boundaries displayed in Figure 9.

To maintain consistency across the decentralized machine, identical code is implemented for every string module, and altering values are declared as global variables. One of these variables is declaring the string number.

TABLE IV
STRING NUMBER DECLARATION FOR NOTE ARRAYS

String	String Number
High E	1
B	2
G	3
D	4
A	5
Low E	6

This number is checked against an if statement upon startup to retrieve the corresponding set of note and position arrays.

The advantage of a decentralized machine, in this case, lies in its ability to address scenarios where two strings may produce the same note (e.g., string B and G both play a C4). To address this overlap, note sets are stored as separate arrays to ensure the matching position corresponds to the intended string. This aligns with the decentralized setup, detailed in Section II-A2, as each string module only considers its mapped set based on the initial string identification. In contrast, a centralized machine would need to deal with multiple strings playing the same note. This is a trade-off which could be beneficial in efficiency-focused systems rather than user choice.

IV. Evaluation

To meet the specifications outlined in Section I, evaluations of the expressive performance modes are conducted. Additionally, the note mapping designed in Section III-E is tested. Lastly, speed tests are performed to determine the optimal carriage movement speed between the mapped notes.

A. Performance Modes

Each performance mode was executed on Azure Talos and the subsequent audio was recorded.

1) Tremolo

Section I specifies that to achieve tremolo picking, a rate of at least 10 picks per second is necessary.

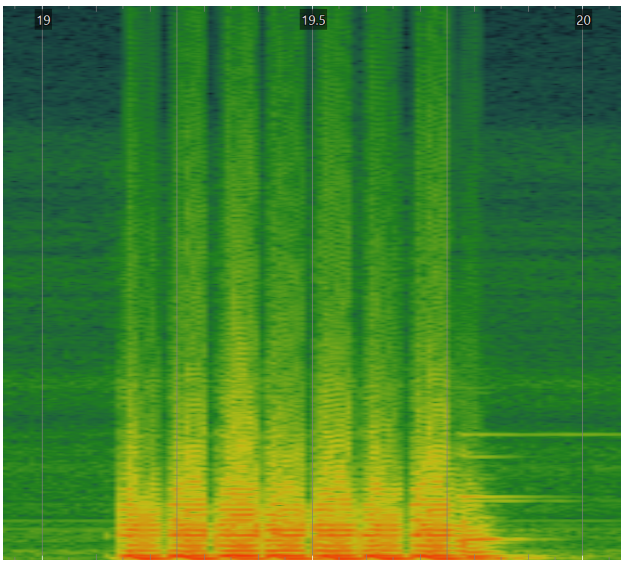

Fig. 13. Spectrogram of tremolo pick

Figure 13 shows a spectrogram recording of the tremolo performance mode[1]. Seven picks can be seen within a 0.63-second time frame, equivalent to a rate of 11 picks per second. This exceeds the required speed for tremolo picking, confirming the successful execution of the technique.

2) Palm Mute

Section I specifies that to achieve palm muting, the amplitude and sustain in comparison to a normal pick must decrease.

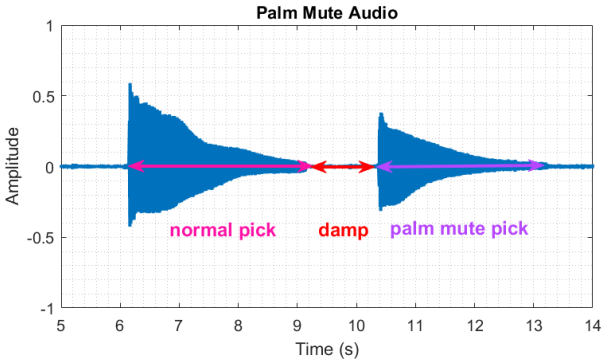

Fig. 14. Audio analysis of normal pick vs palm mute pick

Figure 14 displays that the palm mute pick[2] is diminished before the damper comes in contact with the string, showing it sustains quicker than a normal pick. It is also at only 70 percent of the amplitude of a normal pick, therefore meeting both of these requirements.

3) Ghost Notes

As specified in Section I, the ghost note pick must produce a pick that is of more non-harmonic content than a normal pick event. Unlike a palm mute pick, which still has a distinguishable pitch, a ghost note results in a percussive sound. This reemphasises the importance of having access to all of Azure Talos' actuators, as those with similar features (both have a silicone damper) can result in different expressive sounds.

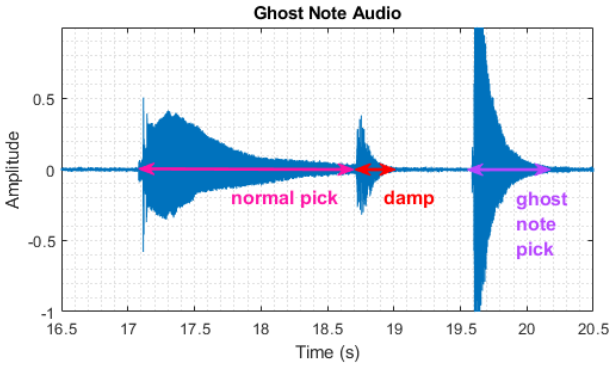

Fig. 15. Audio analysis of normal pick vs ghost note pick

[1]Link to tremolo video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Tremolo

[2]Link to palm mute video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Palm-Mute

Figure 15 shows an audio recording of a normal pick, a damp of this pick, and a ghost note pick[3]. The normal pick is sustained over 1.1 seconds, whereas the ghost note is dissipated within 0.5 seconds and is over double the amplitude. This result shows the percussive sound outlined in Section I, therefore resulting in a successful ghost note.

4) Slide

Section I specifies to achieve a slide, there must be a continuous increase or decrease in frequency over time, from one note to another, with no interruptions. Figure 16 displays a slide[4] performed by Azure Talos, where the frequency continuously increases from approximately 220 Hz to 250 Hz, successfully achieving a slide.

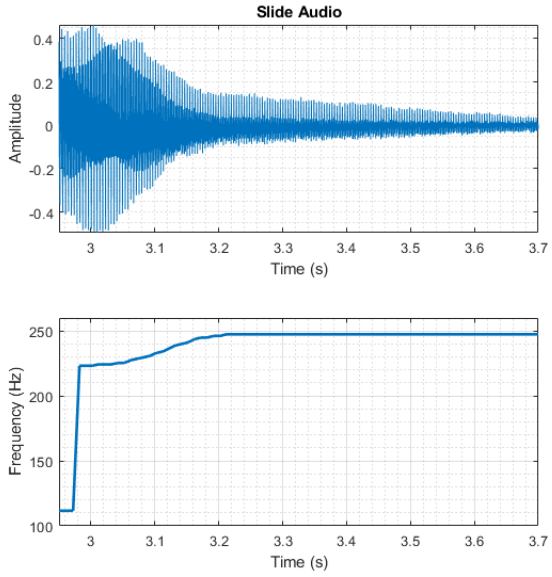

Fig. 16. Audio analysis of a slide

5) Vibrato

Section I specifies to achieve vibrato[5], a wavering pitch difference of over 3 Hz must occur as this surpasses the JND. Figure 17 shows a wavering pitch difference between 90 Hz and 280 Hz, which exceeds well beyond the 3 Hz requirement.

[3]Link to ghost note video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Ghost-Note

[4]Link to slide video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Slide

[5]Link to vibrato video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Vibrato

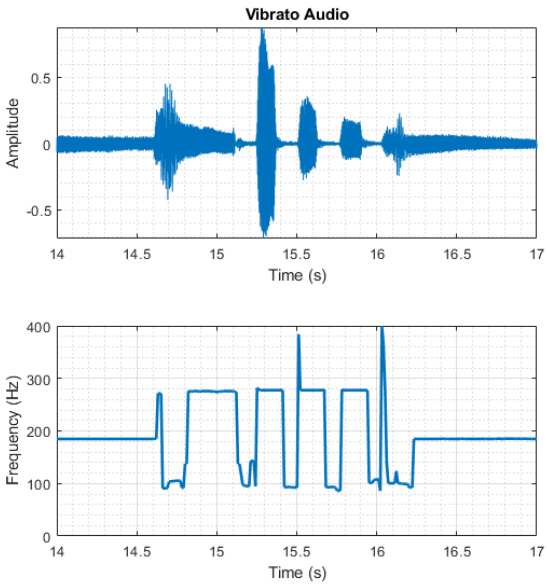

Fig. 17. Audio analysis of vibrato

6) Pitch Bend

Section I specifies to a play successful pitch bend[6], a pitch difference of at least 3 Hz must be achievable when clamping in the same location.

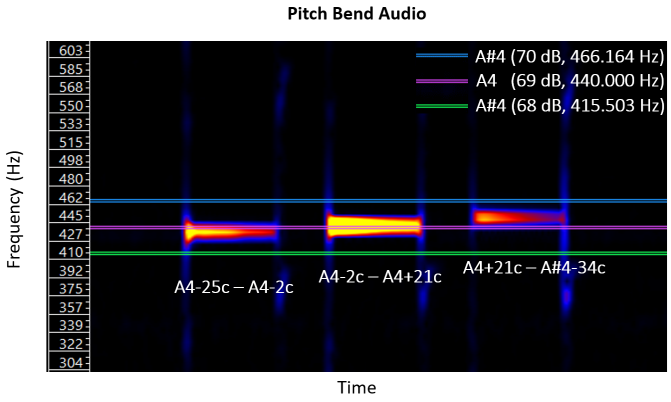

Fig. 18. Melodic range spectrogram of a pitch bend pluck at 3 different intensities

Figure 18 illustrates three picks where the clamper is exerting varying levels of tension on the string. It is evident that as greater pressure is applied to the string, the pitch rises by more than 3 Hz, and therefore creates a noticeable change in pitch, meeting the pitch bend requirement.

These audio analyses prove the successful implementation of the expressive techniques stated in Section I.

In addition to playing expressive techniques, Azure Talos must be able to play musical scores.

B. Note Mapping

The estimated note positions were calculated in Section III-E. The resulting pitches of these locations were evaluated to ensure the accuracy of each note was within the JND of ± 10 cents.

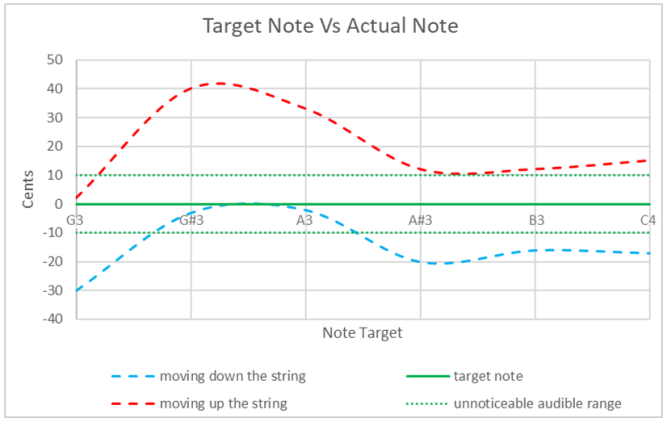

Fig. 19. Note accuracy of calculated positions

Figure 19 shows the results from testing the calculated string positions, where the audio was recorded and analysed by Ableton Live's tuner application. During this test, it was revealed that the same position instructions in the firmware would result in different physical positions from the hardware depending on whether the carriage was travelling up or down the string.

Further investigation revealed the presence of mechanical backlash. When the Dynamixel believed it had reached its intended position and froze, manually pushing the carriage demonstrated a backlash. As a result, when the carriage moved up the string, it stopped below the pitch of the target, and when it moved down the string, it exceeded the pitch of the target. These pitch discrepancies fell outside the JND range, necessitating firmware adjustments.

To address this issue, an if statement was introduced within the `myNoteOn()` function to compare the incoming MIDI note value with the previous one, determining the direction of carriage movement. Given a backlash discrepancy of 13 (Dynamixel position), the new note target would displace the carriage by half of this value. Consequently, when the carriage moved upward, the new target position became the original array value plus 7.5 (Dynamixel position), and when moving downward, the new target position became the array value minus 7.5. This approach established a dual note-mapping system for each direction of the string module, eliminating the need to create six new note arrays in the programming.

After the note target was updated, the same test was performed.

Figure 20 illustrates the improved note accuracy achieved after addressing the issue of backlash, resulting in all six notes of the G string falling within JND of the target note.

This same process was used to map the remaining string modules of Azure Talos, with note positions updated based on the carriage displacement relationships noted in Section II-B2a.

This fulfills the requirement of no hardware changes, as the firmware could account for hardware issues. This further plays into the sustainability of the machine as firmware adaptations

[6]Link to pitch bend video: https://gitlab.ecs.vuw.ac.nz/sonics/azure-talos/-/tree/main/Videos/Pitch-Bend

to the machine are more sustainable than ordering or designing new hardware to account for the backlash.

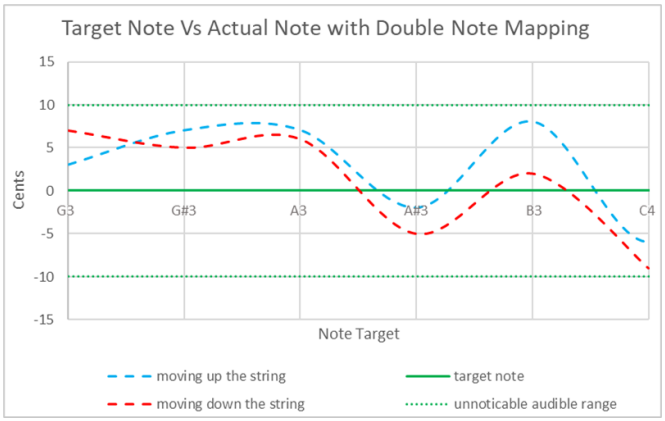

Fig. 20. Note accuracy after backlash compensation

C. Carriage Speed

To determine how fast a user could compose Azure Talos to play, the carriage was tested travelling at different speeds to define a balance between minimizing delays, overshoot of the target position, and mechanical noise. As a composer for Azure Talos would use a DAW to send musical data, Ableton Live is used for testing.

Incorporating the existing firmware structure of a performance mode, MIDI CC 27 was introduced during the evaluation phase to adjust the speed data transmitted to the Dynamixel. The system was testing at speeds of 100 rpm, 200 rpm, 300 rpm, and 400 rpm, resulting in a mapping of the input range from 1 to 127 to an output range of 1 to 400. This mapping resulted in equivalent Dynamixel speed levels of 1, 43, 86, and 127. This reemphasises the value of the previously developed firmware in the evaluation process. The use of MIDI CC offers enhanced expressivity and overall usability, aligning with the system's requirements.

The test involved transmitting two MIDI note-on messages, each lasting 0.25 seconds, as illustrated in Figure 21. This test aimed to gauge the carriage's capabilities in high-speed situations, such as quickly traversing the string to play a fast-paced melody. Additionally, it ensured that the system could perform incoming instructions, allowing quarter notes to transition to 1/8 notes at the commonly used tempo of 120 bpm. To accurately emulate this, the Ableton Live project was configured at a tempo of 60 bpm, equating seconds to musical time.

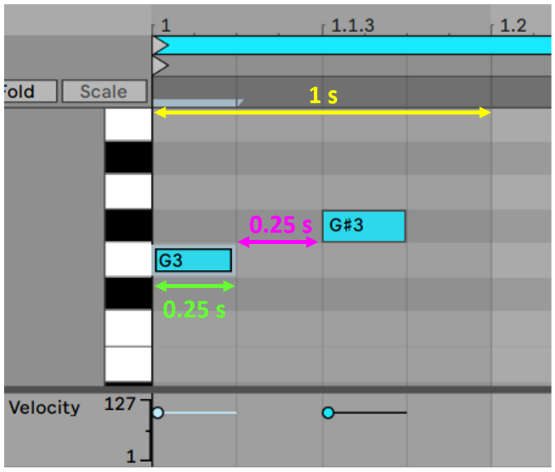

Fig. 21. Able Live program testing notes 1 and 2 0.25 s apart

These messages were evaluated at different positions along the string, spanning from note position one to seven. In relation to this note range, four distances were examined: 0 to 0.25 (encompassing notes 1 to 2), 0 to 0.5 (encompassing notes 1 to 3), 0 to 1 (encompassing notes 1 to 7), and 0.25 to 0.75 (encompassing notes 2 to 6).

The results presented in Table V illustrate various timing intervals (0 seconds, 0.25 seconds, and 0.5 seconds) between two played notes. The objective is for the system to execute the move-clamp-pick routine when transitioning from opposite ends of the string within a window of 0.25 seconds, ensuring no instructions are missed. This capability enables the composer to play four notes per second on a single string module, regardless of the note played. To further enhance this efficiency, predictive positioning, as discussed in Section V-B, can be employed. This approach eliminates the need for the carriage to wait for a MIDI note-on message, which is presently a source of latency.

TABLE V
SPEED TESTING IF AZURE TALOS PLAYS BOTH THE GIVEN NOTES

Speed	Note Relationship	0 s	0.25 s	0.5 s
100	1 to 2	N	Y	Y
100	1 to 3	N	N	N
100	1 to 7	N	N	N
100	2 to 6	N	N	N
200	1 to 2	Y	Y	Y
200	1 to 3	Y	Y	Y
200	1 to 7	N	N	Y
200	2 to 6	Y	Y	Y
300	1 to 2	Y	Y	Y
300	1 to 3	Y	Y	Y
300	1 to 7	N	Y	Y
300	2 to 6	Y	Y	Y
400	1 to 2	Y	Y	Y
400	1 to 3	Y	Y	Y
400	1 to 7	Y	Y	Y
400	2 to 6	Y	Y	Y

A speed of 400 rpm was initially deemed sufficient to meet the timing constraints, albeit with a minor issue of overshooting before reaching the target position. In the single case when travelling from note 7 to note 1, the proximity of the first note to the chassis with this overshoot caused

the carriage to collide with the frame. To address this, the speed was adjusted to 380 rpm, which still maintained the timing precision and note accuracy, while eliminating excess overshoot. This modification also resulted in a reduced level of mechanical noise from the carriage hitting the frame. This speed optimisation adjustment enhances the machine's usability by improving its real-time performance, consequently expanding the range of musical scores it can accurately reproduce.

Evaluating the requirements for the Azure Talos firmware design shows the successful implementation of expressivity, usability, note accuracy, and no hardware changes. This firmware design lays the necessary foundations for future work to be explored.

V. Future Work

The evaluation of Azure Talos has highlighted several areas where further research and development are warranted in order to enhance its capabilities, which are now feasible due to the new accessible and documented firmware design.

A. Advanced Firmware Design

The existing firmware has laid the foundation for Azure Talos, but there is room for refinement and expansion. Future work should focus on optimizing the firmware to achieve an even greater speed response and accuracy. Additionally, incorporating open notes to be played would increase the melodic range Azure Talos could play.

B. Intelligent Positioning and String Selection

Improving the system's capacity to intelligently determine the appropriate string module for playing the next note in scenarios where notes may overlap, as detailed in Section III-E, could significantly boost the system's performance. Further investigation into AI-driven solutions, such as the potential utilization of Markov models [25] or similar techniques, should be pursued. Furthermore, integrating AI-driven score following techniques or predictive positioning could empower the system to play alongside real musicians or improvise itself.

C. Pitch Extraction and Auto-Tuning

Developing pitch extraction algorithms and implementing auto-tuning capabilities can significantly improve the quality of Azure Talos' sound output. This will enable users to maintain perfect pitch of the system.

D. UI Enhancements

A designated UI for Azure Talos would allow for easy interaction with the system without needing prior knowledge of a DAW.

VI. Conclusion

In conclusion, the new firmware design for Azure Talos not only renews the robot's potential for expressive musical performances, but also places a strong emphasis on sustainability and usability, ensuring the longevity of this unique mechatronic chordophone. The project's objectives were successfully achieved, laying the foundation for future advancements.

References

[1] Victoria University of Wellington, "Machine Music", wgtn.ac.nz, https://www.wgtn.ac.nz/news/2022/02/machine-music, (accessed June. 02, 2023).

[2] O. M. Dictionary, "tremolo." https://dictionary.onmusic.org/terms/3663-tremolo. Accessed: Sep 2023.

[3] G. Scholar, "Palm muting." https://www.guitarscholar.co.uk/lessons/palm-muting.php. Accessed: Sep 2023.

[4] O. M. Dictionary, "false note." https://dictionary.onmusic.org/terms/1377-false_note. Accessed: Sep 2023.

[5] O. M. Dictionary, "slide." https://dictionary.onmusic.org/terms/3211-slide. Accessed: Sep 2023.

[6] G. Scholar, "Guitar dictionary - vinrato." https://www.guitarscholar.co.uk/dictionary/index.php?t=vibrato. Accessed: Sep 2023.

[7] D. Darling, "Encyclopedia of music - pitch bend." https://www.daviddarling.info/encyclopedia_of_music/B/bend.html. Accessed: Sep 2023.

[8] "Centralized vs. decentralized digital networks: Key differences." https://cointelegraph.com/explained/centralized-vs-decentralized-digital-networks-key-differences. Accessed: Sep 2023.

[9] M. M. Ramanjaneyulu, "Lecture notes on embedded systems design," 2023.

[10] D. Povel, "A theoretical framework for rhythm perception," *Psychol*, vol. 45, p. 315–337, 1984.

[11] B. Kollmeier, T. Brand, and B. Meyer, *Perception of Speech and Sound*, pp. 61–82. 01 2008.

[12] MUTOR, "Unit 5: Pitch, Intervals and Key Areas," Available: https://mutor-2.github.io/ScienceOfMusic/units/05/ [Accessed Sep 2023].

[13] CompTIA, "What Is a Network Protocol, and How Does It Work?", comptia.org, https://www.comptia.org/content/guides/what-is-a-network-protocol: :text=Communication

[14] NYU, "Open Sound Control", itp.nyu.edu, https://itp.nyu.edu/networks/explanations/open-sound-control/, (accessed June. 02, 2023).

[15] Sparkfun, "Serial Communication", sparkfun.com, https://learn.sparkfun.com/tutorials/serial-communication/all, (accessed June. 02, 2023).

[16] J. McVay, D. Carnegie and J. Murphy, "An overview of MechBass: A four string robotic bass guitar," 2015 6th International Conference on Automation, Robotics and Applications (ICARA), Queenstown, New Zealand, 2015, pp. 179-184, doi: 10.1109/ICARA.2015.7081144.

[17] E. Singer, J. Feddersen, C. Redmon, B Bowen, "LEMUR's Musical Robots", International Conference on New Interfaces for Musical Expression, NIME 2004.

[18] "Teensyduino - Add-on for Arduino IDE to use Teensy USB development board," www.pjrc.com. https://www.pjrc.com/teensy/teensyduino.html (accessed June. 02, 2023).

[19] "Arduino Dynamixel Library – Savage Electronics Blog." https://savageelectronics.com/arduino-y-biblioteca-dynamixel-ax-12a/ (accessed June. 02, 2023).

[20] "AccelStepper Arduino Library, connecting Stepper Motors to Teensy," www.pjrc.com. https://www.pjrc.com/teensy/td-libs-AccelStepper.html (accessed June. 02, 2023).

[21] Arduino, "Bounce library." https://www.pjrc.com/teensy/td_libs_Bounce.html. Accessed: Sep 2023.

[22] Arduino, "Midi library." https://www.arduino.cc/reference/en/libraries/midi-library/.

[23] M. Guitar, "Acoustic guitar scale length." https://www.martinguitar.com/blog-categories/from-the-factory/blog-101123-scale-length.html#:~:text=Types%20of%20Acoustic%20Guitar%20Scale%20Length&text=They%20suit%20genres%20like%20blues,fret%20spacing%2C%20and%20tonal%20characteristics. Accessed: Sep 2023.

[24] Stewmac, "Fret position calculator." https://www.stewmac.com/fret-calculator/. Accessed: Sep 2023.

[25] P. Brans, "Markov model." https://www.techtarget.com/whatis/definition/Markov-model, 2022. Accessed: Sep 2023.