J.U.M.P Box: The Jukebox Updated for Modern Play

Andre Jennings, Julian Vazquez, Jacob Riesterer, Jack Wilson

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The J.U.M.P Box modernizes the classic jukebox by automating the selection, retrieval, and playback of 12” vinyl LP records using a robotic gantry system, an intelligent tonearm actuator, and a connected mobile app. This senior design project integrates mechanical design, embedded systems, and software engineering into a cohesive device. This paper summarizes the technical development and integration of each subsystem, highlighting the innovation, engineering tradeoffs, and system performance in a compact and automated vinyl playback system.**

***Index Terms* — Automation, embedded systems, gantry robot, mobile interface, tonearm actuator, vinyl playback.**

I. Introduction

In an age of digital streaming, vinyl records maintain their charm through analog audio and tactile interaction. The J.U.M.P Box (Jukebox Updated for Modern Play) reimagines vinyl playback by merging modern automation and software control with classic media. Our project was inspired by both the limitations of current jukeboxes and the opportunity to make vinyl more accessible through automation. The major goal of the project is to automate the storing and playing of 12” vinyl records. This includes a mechanical system that allows for the physical transfer of records from storage to the turntable, as well as electronic control systems that enable taking user input, playing audio, and interfacing with robotic components.

Our priority for this project is to emphasis usability and convenience. We recognize that the target audience for the J.U.M.P Box consists of music fanatics who will go above and beyond for their listening experience. As such, we place emphasis on having a responsive and streamlined system. This allows us to confidently claim our project as a modern upgrade to an older technology.

II. Core Systems

The J.U.M.P Box consists of three core systems: robotic, audio, and control. This section provides a brief overview of each core system and its role. This section is expanded upon in Section III, which provides more detail on some of the mentioned subsystems.

A. Robotic System

The robotic system consists of all electro-mechanical components. This includes all motors, motor drivers, limit-switches, and any mechanical driving hardware. This system is further divided into a robotic gantry which provides linear movement in two axes (X and Z) and an end-effector, which provides two rotational joints (Roll and Pitch) as well as a clamping motion to grab the records.

B. Audio System

The audio system consists of any audio-interacting devices within the J.U.M.P Box. The primary component here is the turntable, which generates line-level audio signals by reading the vinyl records. These signals are then passed into a digital potentiometer for volume control, and then into an amplifier and speaker.

C. Control System

This is the brain of the J.U.M.P Box. The control system is headed by a duo: a master computer and a slave microcontroller. The computer handles all user inputs from both the touchscreen interface and the mobile app, and delegates commands to the microcontroller. The microcontroller handles all signal-level communication to the motor drivers, limit switches, and turntable controls.

III. Subsystem Overview

The J.U.M.P Box is comprised of many subsystems and components, which each combine into the overall ‘jukebox’ system. This section will introduce each subsystem and some of its details such as its components, purpose, and specifications.

A. Robotic Gantry

The robotic gantry transports vinyls between storage and the turntable. It uses three NEMA 23 stepper motors (model 23HS32-4004S)—two for the horizontal (X) axis and one for the vertical (Z) axis. These steppers were chosen for their high torque (approximately 2.4 Nm) and precise positioning.

They are controlled via TB6600 stepper drivers that support up to 32nd microstepping for smooth motion. Integrated limit switches define the travel endpoints, ensuring reliable and repeatable operation.

B. Servo-Based End Effector (Gripper)

The end effector is responsible for picking up and maneuvering the vinyls. It is attached to the robotic gantry. This end effector consists of three DS-3235-SG servos linked together to accomplish the required motion. These servos were selected due to their high torque (35 kg/cm) and low operating voltage (5-8.4 V).

These servos are controlled using varied pulse widths. As we are utilizing multiple servos, we elected to use a driver circuit to offload the burden of control from the microcontroller. We selected the PCA9685 PWM Driver for this purpose. The PCA9685 boasts 16 channels with a 12-bit PWM resolution at various frequencies. It is controlled via I^2C, which allows for the use of only two GPIOs to power and control up to 16 servos.

C. Signals and Control PCB

The Signals and Control PCB is a crucial component in the JUMP Box project, serving as the central hub for managing logical operations and ensuring effective communication between the microcontroller and the rest of the system. Built around the ESP32-S3 microcontroller, the board integrates various elements such as Boolean gate logic, manual control buttons, and status indicators to provide both automated and manual control over critical MCU functions. Boolean gates process signals like DTR, RTS, EN, and IO0 to trigger reset and bootloader modes, while the inclusion of a status LED gives visual feedback for debugging and verifying logic operations. This design reduces the need for additional diagnostic tools by making the logic states of important pins immediately visible to developers. Manual buttons supplement this automation, offering an alternative method to control and debug the system during development phases, particularly helpful in prototype testing and troubleshooting.

The board layout places strong emphasis on high signal integrity, effective grounding, and efficient power distribution. USB differential pairs are routed carefully to maintain impedance control and minimize electromagnetic interference (EMI), ensuring robust data transmission. Decoupling capacitors stabilize the power supply, while the strategic layout of GPIO pin headers enhances flexibility for integrating motor drivers, sensors, and other peripherals. Together, these components allow the PCB to support seamless communication between the user interface and motor systems. The modular and scalable design accommodates future enhancements, providing room for added features without a full redesign. The inclusion of both automated and manual controls not only boosts reliability and usability but also ensures that the control system remains responsive and adaptive throughout development and testing.

D. Power Supply System

The power supply system for the J.U.M.P Box has one main purpose: reliably supply and distribute power to every component in the J.U.M.P Box. It accomplishes this with a MEAN WELL LRS-350-36 36 V power supply that directly powers our stepper motors and feeds into three buck converters, which output 12 V for our vinyl player, 7.4 V for our servo motors, and 5 V for our Raspberry Pi 5 and ESP32. All of the buck converters are based on the LM2678 IC from TI and as such, are nearly identical to each other, drastically increasing simplicity.

E. Mobile App

The mobile app is responsible for controlling the record system and managing album data. It is built as an Android application using Flutter and features a similar layout to the touchscreen on the J.U.M.P Box. The app additionally integrates Google Vision's Web Detection module to capture and analyze album cover images, extracting key visual features. It then creates a list of potential album matches, allowing the user to select the most likely result. Following this selection, the app uses the iTunes Search API to retrieve detailed album information—including title, artist, track listings, release date, and genre—which is stored in the app's internal database for later use.

For controlling the record system, users receive an interface similar to the touchscreen's, where they can choose from the four installed records in the player; if a change is desired, the app also provides an option to update the installed records. Additionally, a graphical user interface displays the currently playing album along with its associated information.

F. Control Computer

The main control computer in this project is a Raspberry Pi 5. This single-board computer was selected for its exceptional price to performance in coordination with its ability to run a full operating system. Additionally, its built-in Wi-Fi and Bluetooth features were required for connecting the J.U.M.P Box to the internet and to control the J.U.M.P Box via the mobile app.

The Pi 5 is running Ubuntu 24.04 as our team was familiar with the setup, execution, and networking of that operating system. The Pi 5's network is further facilitated using a mobile router (the GL iNet Mango) which is configured to use the Pi as a gateway with NAT forwarding. This allows all mobile devices to connect directly to the Pi's hosted subnet to set up and control the J.U.M.P Box, regardless of the presence of an external network. The NAT forwarding ensures that connected devices retain internet connectivity for user convenience.

The control computer additionally hosts the graphical user interface (GUI) for local operation. This occurs through the inclusion of a 7” IPS touchscreen. Our group selected a touchscreen interface over a traditional button-based approach to provide more feedback to the user and to add an additional layer of modernization to the J.U.M.P Box.

G. Turntable

The Audio-Technica AT-LP60X was chosen for our automated vinyl project due to its reliability, affordability, and ease of integration. As a fully automatic entry-level turntable, it offers built-in physical controls and an automatic tonearm, making it highly compatible without a robotic system. These features reduce the need for complex modifications, allowing us to focus on automation and system development. Its low power consumption also aligns with our design constraints, being priced at $150, it fits within our budget. Additionally, Audio-Technica provided internal schematics upon request. These schematics prove to be a useful resource for modifications.

Despite its advantages, the AT-LP60X does have limitations. Its plastic build lacks the durability and vibration resistance of higher-end models, and it doesn’t include features like anti-skate or adjustable feet, which can impact playback quality. The tonearm is non-adjustable, and the cartridge is not upgradable, limiting performance improvements. While these shortcomings may disappoint audiophiles seeking customization and precision, the AT-LP60X remains a practical and efficient solution for an automation-focused vinyl project.

IV. System Concept

To be understood as a complete system, a flowchart and state machine of operation are helpful. Figure 1 depicts how the J.U.M.P Box manages vinyl playback from selection to completion. Once a user selects a record—via the local interface or mobile app—the robotic arm retrieves it from storage and places it on the turntable. Playback then begins and continues until side A finishes. If side B is also scheduled to play, the system automatically flips the record without user input, allowing the album to continue seamlessly. If no further playback is required, the record is returned to storage. This design provides a hands-free vinyl experience by automating retrieval, placement, and flipping, while still allowing the user to control which album is played.

Figure 2 presents the overall block diagram of major components and details the flow of information as indicated by the arrow direction. Please note that while some connnections are duplex, the arrows indicate the primary direction of communication.

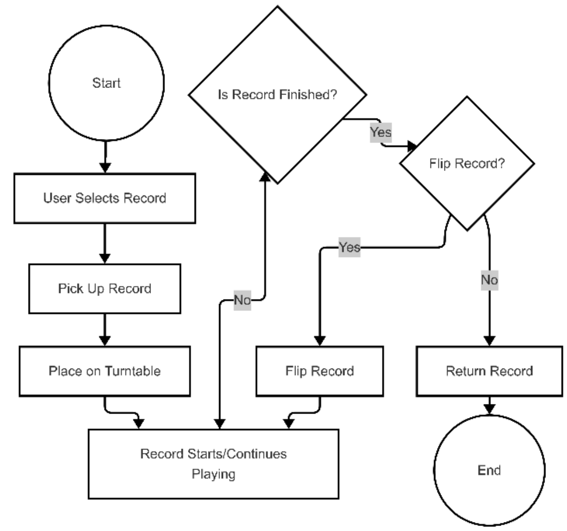


Fig. 1. Complete system flowchart for the record player system. Shows actions taken to ensure proper playback.

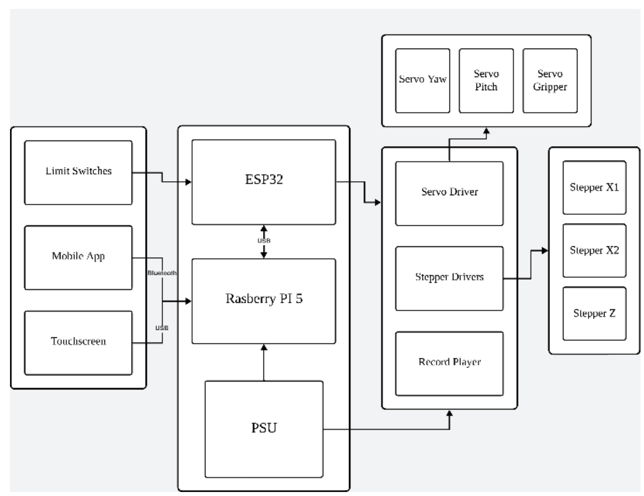


Fig. 2. Block diagram presenting major system components.

V. Hardware Detail

This section details the implementation of the J.U.M.P Box’s key hardware. This includes technical details previously excluded from other sections.

A. Gantry

The gantry is designed with two parallel horizontal rails for X-axis motion, each driven synchronously by dedicated stepper motors to ensure even and balanced movement. The X-axis motion utilizes GT2 timing belt pulleys, which required careful tensioning and precise alignment to maintain accuracy and prevent slippage or backlash. Initial attempts to mount components using laser-cut MDF wood resulted in material bending and alignment issues, negatively impacting the performance and accuracy of the gantry. This issue was resolved by reordering and replacing these mounts with precision-cut metal plates, significantly improving rigidity and performance.

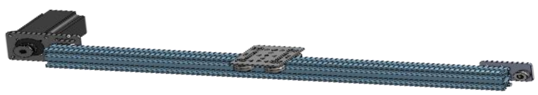


Fig. 3. X-axis CAD drawing made up of steppers, pulley, and v-slot rail with the gantry cart mounted to it.

The Z-axis motion is managed by a single vertical rail system driven by a large lead screw, which offers precise and stable vertical positioning. The Z-axis and X-axis assemblies are securely connected using multiple metal L-bracket connectors along with additional V-slot aluminum framing to ensure rigidity and alignment. The design strategically excludes a Y-axis by positioning the entire system's operation along the central axis of the end effector. This approach simplifies the mechanical complexity and reduces the number of motors and control elements required, effectively streamlining both the control and mechanical design while maintaining full operational functionality.

Fig. 4. Z-axis CAD drawing made up of stepper, lead screw, and v-slot rail with the gantry cart mounted to it.

B. End Effector

The end-effector for the J.U.M.P Box consists of three DS 3235 SG servos. Two of these are used for rotational joints which allow for maneuvering discs from an upright position to a horizontal position, as well as a full rotation which allows for both A and B sides of records to be played. The third servo is used inside the gripper assembly to provide a holding force to actively hold a record.

The gripper assembly itself is based on a dual rack and pinion design which allows for equal pressure and ensures that contact surfaces always meet the records at parallel angles. This ensures that the gripper does not damage the delicate vinyl records while maintaining maximum surface area and contact with the discs. The contact surface of the gripper is felted with wool to ensure that enough friction is present to secure the record without slipping while also being soft enough to prevent damage to the record’s surface.

C. Power System & Layout

There are several components in the J.U.M.P Box that each have their own unique power requirements. The main components and their power requirements are outlined in the table below.

Component	Power Requirement	Quantity	Wattage
Servo Motor	2.3 A, 7.4 V	3	51.06
Stepper Motor	4.0 A, 36 V	3	432
Raspberry Pi 5	5 A, 5 V	1	25
ESP32	0.5 A, 3.3 V	1	1.65 (from Pi)
Vinyl Player	0.083 A, 12 V	1	1
Total:			509.06

Based on this table, we should have a power supply that can output at least 509 W, but this is unrealistic, as the table assumes maximum current draw from every single component, all at the same time, which is far from how the J.U.M.P Box will operate in practice. At maximum, only 2 stepper motors and 1 servo motor would move at once, while the Raspberry Pi 5 and the vinyl player are still operating. Still assuming maximum current draw from each component, the power consumption comes out to be 331 W. Although that estimate is still unrealistic, building a power supply system around it will ensure a margin of safety.

For the J.U.M.P Box, running it off power from the wall makes the most sense. It is not intended to be portable, so batteries introduce unnecessary complexity and even risk.

The next challenge is outputting the correct voltage for each of our components. Several voltages are needed: 36 V, 12 V, 7.4 V, 5 V, and 3.3 V. One voltage can be obtained by choosing it to be the output of our power supply. The output voltage could be any of these voltages, but 36 V was chosen for a critical reason: the stepper motors use 36 V. The stepper motors are by far the most power-consuming component in the J.U.M.P Box, and as such, we want to avoid supplying the voltage for it through a buck converter, as that will introduce more inefficiencies. This also saves us from designing a PCB that would handle the large current draw of the stepper motors. Naturally, the rest of the voltages are to be obtained by buck converters that will take 36 V as their input. The ESP32 requires 3.3 V, but to save ourselves one less buck converter, the ESP32 is powered as a peripheral of the Raspberry Pi 5, converting the 5 V output voltage of the Pi to 3.3 V with an on-board linear regulator. That leaves just 3 buck converters for the rest of the components: 12 V, 7.4 V, and 5 V. The overall power architecture can be observed in figure 5.

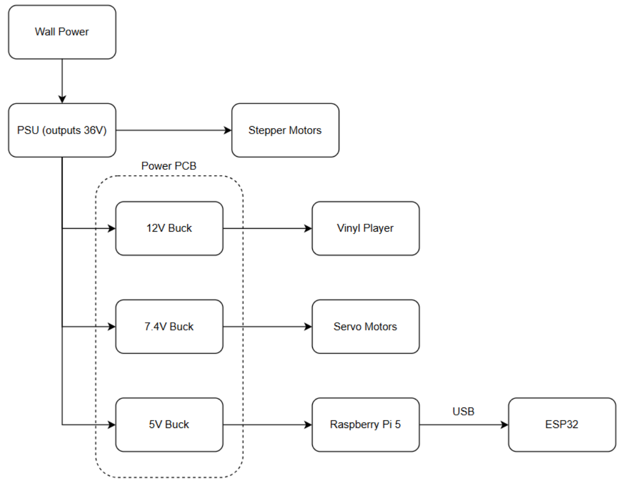


Fig. 5. Power Distribution Flowchart

All the buck converters use a version of the LM2678. The LM2678 was chosen due to its high input voltage, high current output, and low switching frequency. The lower frequency allows a more forgiving PCB layout. The 12 V and 5 V use a fixed output version for their respective voltage, and the 7.4 V uses an adjustable version with a feedback line made with an on-board voltage divider. The inductor also changes on each buck converter. This means that the buck converter PCBs are nearly identical to each other, especially when it comes to board layout. The PCB layout detailed in Figure 6 is the layout for the 5 V buck converter, but it is largely the same for the other buck converters. This was a design decision made on purpose, as components and even some of the boards can be shared across buck converters. In Figure 7 is the schematic for the 12 V buck converter, which is just like the other ones, except for the inductor and the lack of voltage divider.

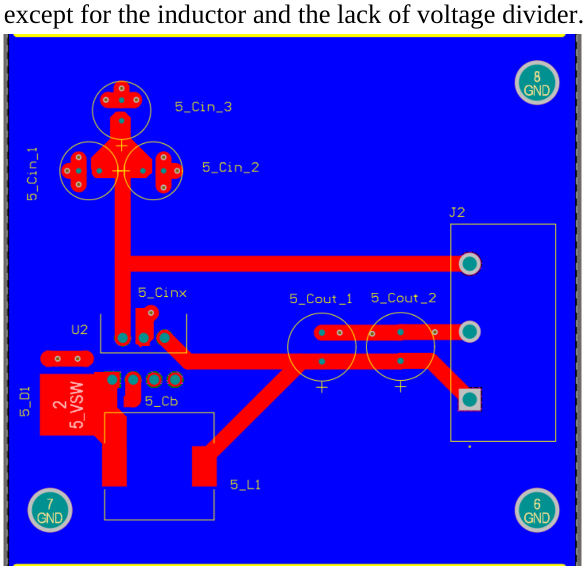


Fig. 6. 5 V Buck Converter PCB Layout

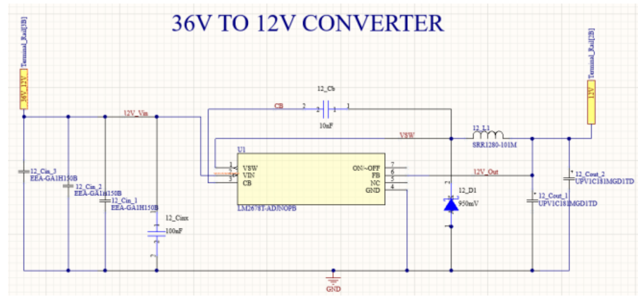


Fig. 7. 12 V Buck Converter Schematic

VI. Control System Detail

This section details the implementation and connectivity of the control system. This includes technical details previously excluded from other sections. The control system can be viewed in abstract layers, as shown in the following figure.

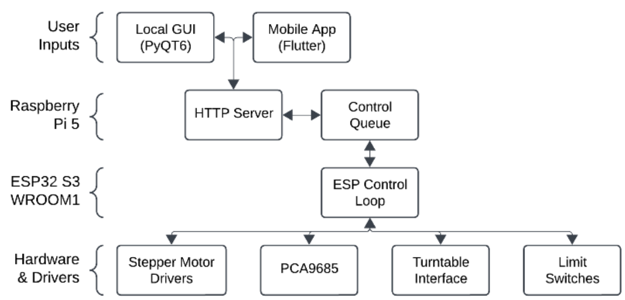


Fig. 8. Abstraction Layers for Control System

With this four-layer concept, we can distinctly define the relationship between each component in the control system and prevent redundant communication or "carry-through" signals. It is important to recognize that we limit communication within layers (with the notable exception of the systems running on the Raspberry Pi) and only allow for communication between a layer and its immediate neighbors. In this way, no communication must pass directly from a hardware component to the Pi, and no user inputs must be accepted by the ESP32. This simplifies our control logic while ensuring that all devices can communicate their requirements through a defined chain of command.

Each cross-layer connection follows a single point of contact to relay information between the two layers. This section will detail which communication protocols are used, how they are used, and why they were selected.

A. User Inputs to Raspberry Pi Communication

To allow the Raspberry Pi to receive user inputs, we elected to host a local HTTP server on the Raspberry Pi. This server is contactable from the Raspberry Pi's local network on its private static IP. This configuration was used to allow for HTTP traffic in all locations regardless of any existing infrastructure. As the Pi 5 only has a single built-in Wi-Fi interface, we were required to use its ethernet interface to host the private subnet. This was accomplished by connecting the Pi 5 to a mobile smart router (the GL iNet Mango) and configuring the Pi to act as the gateway for the router. The following figure details this network configuration.

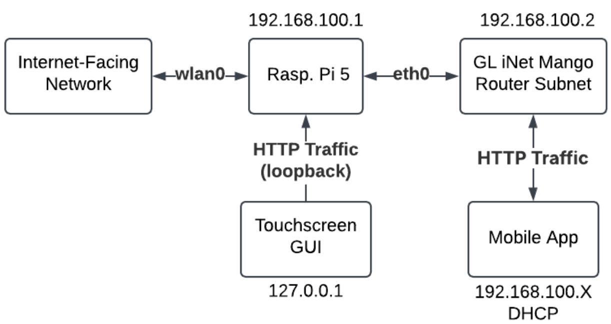


Fig. 9. Raspberry Pi Network Configuration and Topography

Both the touchscreen and the mobile app communicate with an HTTP server running on the Pi. The touchscreen is running on a separate process within the Pi, and as such, communicates to the HTTP server over the loopback address. The mobile app is required to use the private subnet provided by the router to access the Pi's server.

B. Raspberry Pi to ESP32 Communication

The HTTP server receives commands but does not perform any decision making. It parses and passes received commands to the main control loop, which is responsible for implementing command error-checking, command priority and deconfliction, and command issuing. Commands are issued via serial communication with the main signal PCB. This is accomplished via a USB connection. The serial connection runs with a baud of 115200 with a standard 8N1 configuration (8 data bits, 1 stop bit, with no parity). This baud rate is more than enough for issuing our ~16-character commands. The ESP32 additionally reports any pertinent statuses to the control loop on the upstream serial wire, which includes informing the Pi of the current disc being moved/played, current volume setting, the microcontroller's busy status, and the state of the limit switches when required.

C. ESP32 to Hardware and Drivers Communication

The communication between the ESP32 and the hardware is the most complicated link between layers. While all other layers have a single communication schema, the ESP32-to-hardware link must be specialized for each hardware type. These primarily are operated through direct signal wires, but the protocols vary by device. For example, the TB6600 stepper drivers are run through a three-wire configuration using the AccelStepper motor driver library. This relies on Pulse, Direction, and Enable signals. The enable signal turns the motor driver on and off, with a default value of enable low. The direction signal sets the stepping direction (either clockwise or counterclockwise). To inform the driver to take a step, the pulse wire is driven high, and a step is taken at the rising edge. Micro-stepping and current limiting are handled by mechanical switches, meaning that those configurations are not changed during operation.

For the PCA9685 Servo driver, all communication signals are sent via I2C. This communication is facilitated by Adafruit's PWM Servo Driver Library. Our communication is based on the PCA9685's default address of 0x40. These commands are then run simultaneously on the PCA9685, leaving the ESP32 available for other tasks instead of spending cycles operating PWM loops or interrupts. This PWM driver operates all servos, including those in the servo-based end effector and on the turntable's mechanical switches.

Communication with the limit switches occurs using lines pulled up to 3.3V through pull-up resistors. The limit switches are normally closed, and when triggered, tie the line down to low (~0) volts. This is then read by the ESP32's main loop as a digital read input.

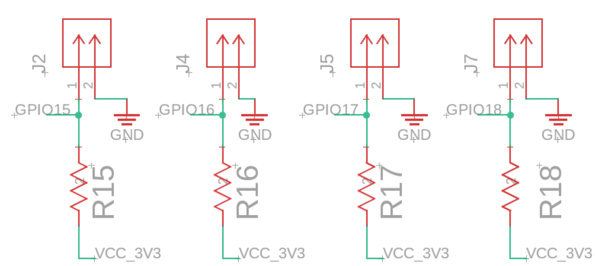


Fig. 10. Schematic of Limit Switch Connectors on Signals and Controls PCB

D. ESP32 S3 Signals PCB

To manage all hardware connections while keeping the system modular and reliable, a custom Signals PCB was developed. This board features the ESP32-S3 and serves as a central hub for connecting stepper drivers, servo controllers, limit switches, and status LEDs. It has an addressable RGB status LED, built-in Boolean logic for signal control, and buttons for manual testing. This PCB layout ensures that all wiring is neat, accessible, and functional while also providing an easy avenue for potential upgrades later on.

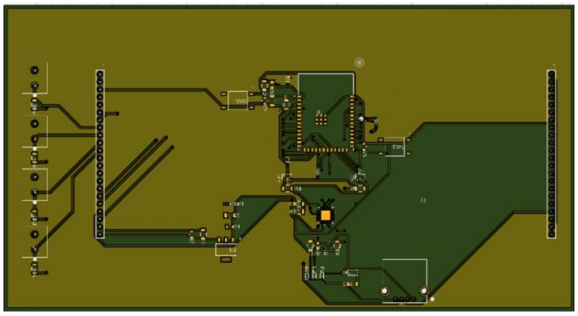


Fig. 11. Completed design of the Signals and Controls PCB

VII. Software Detail

This section details the implementation of the J.U.M.P Box's key software. This includes technical details previously excluded from other sections, as well as important snippets of code and pseudo-code to facilitate understanding.

A. Raspberry Pi Software

Our system integrates a mobile application and a Python server to create an intelligent, automated vinyl playback solution. The core of our Python backend is encapsulated in app.py, a file that orchestrates various system functions by leveraging several key libraries. At its foundation, Flask is employed to create a RESTful API that serves as the communication bridge between the mobile application and the backend services. Flask-Cors is used alongside to facilitate secure cross-origin resource sharing, ensuring that the Flutter-based mobile app can interact with the server seamlessly. A critical aspect of the system is image processing, achieved by integrating the google cloud vision library. This module sends uploaded images to the Google Vision API to extract descriptive labels, which then inform subsequent music searches. Finally, the serial library is pivotal for direct hardware control; it enables the app.py server to send formatted commands over a serial connection to an ESP microcontroller, thereby interfacing with physical components such as motors. Together, these libraries make app.py the linchpin of our project, ensuring smooth data flow, reliable storage, advanced image analysis, and precise hardware control—all of which are essential to the overall operation of our automated vinyl playback system.

In our system's image analysis module, the function detect_web employs the Google Cloud Vision API to extract descriptive annotations from an input image. The function opens the image file in binary mode, constructs an image object from the binary data, and then submits this object to the Vision API's web detection service. If the response contains best guess labels, these labels are printed; if an error is encountered, an exception is raised. The functionality can be represented in pseudocode as follows:

```
function detect_web(image_path):
    binary_data = read image_path in binary mode
    image_object = create image from binary_data
    response = VisionAPI.web_detection(image_object)
    if response contains best_guess_labels:
        for each label in best_guess_labels:
            print(label)
```

Fig. 12. Pseudo code for detect_web function, which uploads image to google vision for identification.

This process is essential as it automates the extraction of keywords from album cover images, which are subsequently used to query the iTunes API for detailed album information.

In the backend, the music_search function plays a pivotal role in converting user input into detailed album information retrieved from the iTunes API. When a request is received, the function extracts the album title and, if provided, an artist name from the query parameters. It then constructs an HTTP GET request to the iTunes search endpoint using these parameters, ensuring that the search is confined to albums. If the initial search yields results, and an artist name is specified, the function filters these results to retain only those that match the artist criteria. Once a suitable album is identified, the function retrieves the collection ID and uses it to issue a second HTTP GET request to the iTunes lookup endpoint. This lookup fetches comprehensive details, including the album's track listing. The entire process is encapsulated in a try-catch block to handle errors gracefully, and the final result is returned in

```
function music_search():
    album = get query parameter "album"
        return error response
    search_response = HTTP GET to iTunes search with album as term
    lookup_response = HTTP GET to iTunes lookup with collection_id
    parse lookup_response into lookup_data
    return lookup_data as JSON
```

Fig. 13. Pseudo code for music_search function, which makes requests to the itunes search api.

JSON format. The following pseudocode summarizes the logic of music_search:

This function is essential as it bridges the output of the image analysis process with the retrieval of precise music metadata, thereby enabling the mobile application to present users with accurate and detailed album information based on their initial input.

B. ESP32 Software

The microcontroller is programmed using the Arduino IDE. Our software runs in one main control loop with non-blocking command execution. This format is required as we need to be able to accept commands and provide feedback while still having precise control of the motors and all hardware devices. The following flowchart displays the main loop.

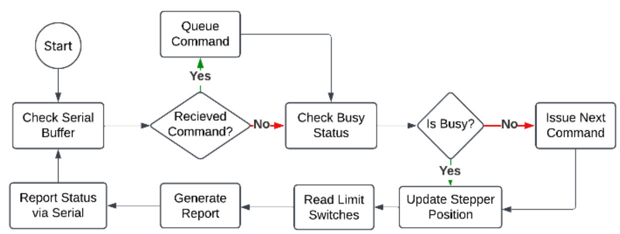
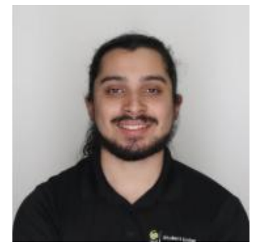


Fig. 14. ESP32 Main Control Loop for Non-Blocking Operation

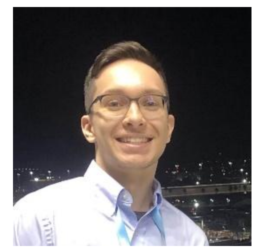
It is essential to note that the stepper positions operate through the AccelStepper library, which allows for the setting of target locations and associated max speeds and acceleration values for stepper motion. Once these values are set, the move() command can be ran within the main loop to update the stepper's positions every cycle. This allows for multiple steppers to be driven by the ESP32 simultaneously while still allowing enough time per cycle to check the serial buffer and generate reports.

The PCA9685 logic has been purposely left out of the main control loop as it houses some specific and intricate logic. Each command is sent via I2C to the controller. The controller operates at a variable frequency between 24 and 1526 Hz with a 12-bit resolution. Our servos operate between 50-330 Hz with pulse widths between 500 and 2500 microseconds. To achieve a high resolution and maximum range of operation, we map 0 degress to 500 microseconds and 270 degrees to 2500 microseconds. In terms of the 12 bit duty cycle at 50 hertz, values between 102 and 512 (of 4096) can be used to control the servos. We increased this value by setting our servos to be driven at 300 Hz instead, with minimum pulses of 614 and 3072 cycles of 4096. This provides far more resolution than the 50 Hz cycle, at the cost of some holding torque.

The Engineers



Andre Jennings is a 22-year-old Electrical Engineering student. Jack intends to continue his education by pursuing his master's degree in engineering management at the University of Central Florida.



Julian Vazquez is a 22-year-old Electrical Engineering student. Julian will be taking a job at Lockheed Martin in Orlando, FL as an Integration and Test Engineer after graduation.

Jacob Riesterer is a 22-year-old Computer Engineering student. Jacob intends to continue his education by pursuing his master's degree in Cybersecurity and Privacy at the University of Central Florida.

Jack Wilson is a 21-year-old Computer Engineering student. Jack intends to continue his education by pursuing his master's degree in computer engineering at Virginia Polytechnic Institute and State University.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Lei Wei, Dr. Wei Sun, Dr. Mohsen Rakhshan, and Mr. Don Harper; University of Central Florida.