Programmable Trackpad

Taylor Barnes, Jonah Halili, Brian Modica, Bradley Vanderzalm

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

***Abstract* — The programmable trackpad is a device that assists the productivity of PC users by implementing shortcut macro keys and rotary encoders in a trackpad device which serves as a replacement for a programmable mouse. This device gives users a method for performing common actions and opening frequently-used applications with a single button press. Users can program these shortcuts themselves with a simple computer application that requires no specialized knowledge. Now, PC users who prefer a trackpad can take advantage of the same convenience one would receive from a programmable mouse.**

***Index Terms* — Embedded systems, mice, microcontrollers, system-on-chip, tactile sensors**

I. Introduction

On the market today, there exists a subset of a common computer peripheral meant to boost a basic end-user's productivity. This product is the programmable mouse. The concept is simple; it is a computer mouse that contains several extra buttons that the user can map to any shortcut or command which he/she chooses. The added convenience of these extra buttons means that a user can save valuable time when performing common, repetitive tasks.

Currently, this type of device only exists for the mouse, but not for the trackpad which has millions of users every day. This is where our project comes in. Our purpose in creating the Programmable Trackpad is to bring this type of technology to the trackpad, for users who prefer to use a trackpad over a mouse. Our trackpad will contain a suite of macro keys and rotary encoders, all of which end users can program themselves. With these, trackpad users will be granted the same convenience and functionality as programmable mouse users in a compact and ergonomic package. This device is intended to completely replace the default trackpad on a traditional laptop.

The guiding principle of our development process is this: create a system which, when connected to a PC, can act as a trackpad and a macro keypad. In order to accomplish this, we will create a hardware device with input systems (trackpad and keypad) and a software application which the PC will use to interpret the hardware's output. Additionally, the device's firmware will manage communication between the device and the PC. The requirements for these systems, as well as the technology used to accomplish these tasks, will be expanded upon in further sections.

II. Project Goals

The major goal of this project is to take a task from people's daily use of their PCs and attempt to create an overall convenience and improved efficiency in their work. This project strives to reduce the steps taken in repetitive and common tasks done on the computer, both for work and for personal use. The project aims to create an external trackpad device that exists outside of the computer that is portable, compact, and purposeful.

To reach the goals of this project, there are certain objectives that need to be met based on specific design choices in hardware and software. Table I lays out the general goals that have guided our team's design process, as well as the specific objectives that are a means to reach each goal.

III. Design/System Overview

In order to accomplish the goals of the project, our system will include many interrelated functions. On a high level, these functions can be summarized with a list of the interfaces with which the user will interact. While the inner workings of the device will be expanded upon in further sections, the following figure (Figure 1) is a general look at the device from a user's perspective.

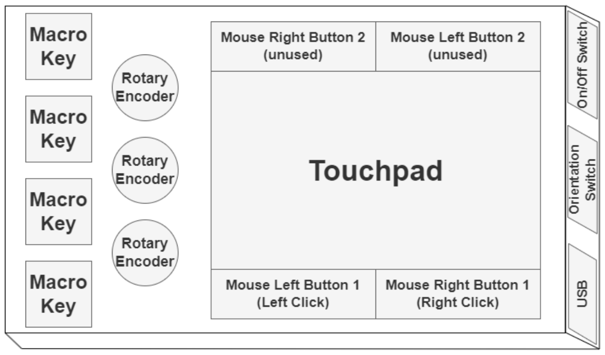

Macro Key
Macro Key
Macro Key
Macro Key
Rotary Encoder
Rotary Encoder
Rotary Encoder
Mouse Right Button 2 (unused)
Mouse Left Button 2 (unused)
Touchpad
Mouse Left Button 1 (Left Click)
Mouse Right Button 1 (Right Click)
On/Off Switch
Orientation Switch
USB

Fig. 1. Trackpad interfaces labeled as the user sees them.

The intended way to use the Programmable Trackpad is for the user to place the device on his/her desk next to the PC's keyboard. For right-handed users, the Programmable Trackpad will be placed on the right of

Table 1
Project Goals and Objectives

Goal	**Objective** (how we achieved said goal)
Reduce common and repetitive tasks	Add buttons with macro key capabilities that are programmable.
Convenient and Ergonomic	**Hardware -** Make the device wireless with Bluetooth capability. **Software** - Run the macros even after termination of the app.
Ergonomic	Support ambidextrous users.
Low Learning Curve	Application with user-friendly interface to program macro keys. Only have to run one executable file, the user doesn't need Python or AutoHotKey installed.
Customizable for user	**Hardware** - Ability to easily remove keys to the user's liking. **Software** - Application should be able to create and store to run on the device

the keyboard, while left-handed users will place the device to the left of the keyboard.

For left-handed users, the same device can be used slightly differently by rotating the Programmable Trackpad 180 degrees. One thing to note is that the mouse click buttons that were unused in the right-handed configuration become the primary mouse click buttons in the left-handed configuration and vice versa.

The following sections detail the various subsystems of the device: hardware, firmware, and software. Figure 2 offers a basic overview of how the systems interact.

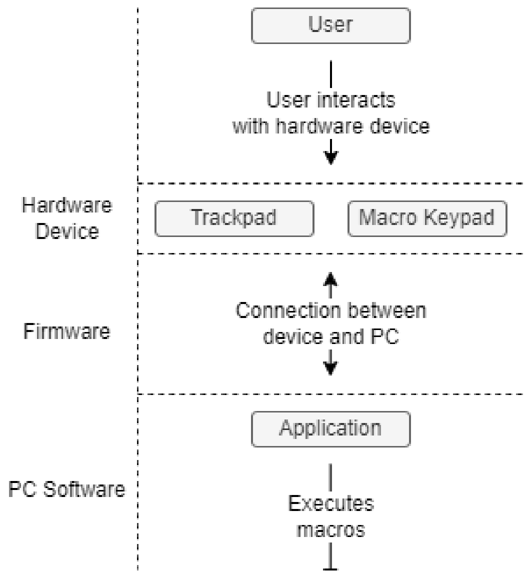


Fig. 2. High-level systems overview.

A. Hardware

The device hardware includes a circuit board, input units, and a 3D-printed plastic chassis. On the circuit board, there is a power system that controls a lithium-polymer battery, a microcontroller, and circuitry to connect these devices to the system's interfaces. Each of these components will be expanded upon in Section IV. All of the electronics are enclosed in the chassis. Figure 3 is an overview of the electronics on the circuit board and how they interact with each other.

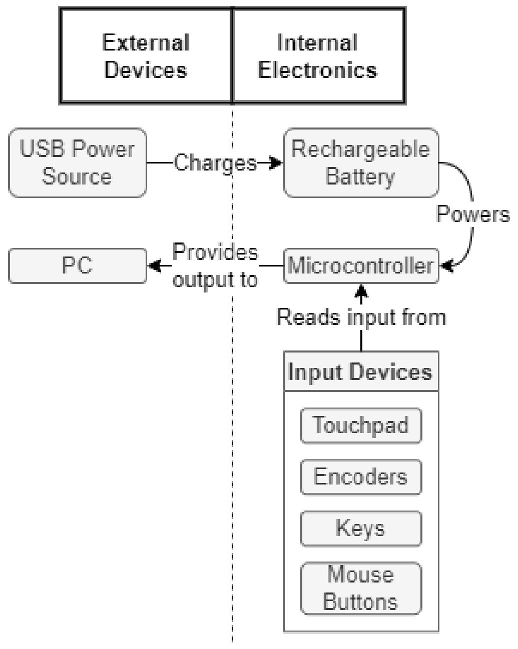


Fig. 3. Hardware block diagram.

B. Firmware

The hardware of the Programmable Trackpad includes multiple computing devices which are responsible for processing input and output. The design challenges associated with such devices include writing code for programmable devices and selecting communication protocols for the devices to use. When the device is powered on, it immediately begins searching for a Bluetooth device with which to pair. The user must configure this Bluetooth connection on the PC. As long as the device stays on and the PC does not sever the connection, then the Programmable Trackpad will stay paired with the PC.

The input/output functionality of the Programmable Trackpad is handled by an nRF52840 microcontroller. Its responsibilities include processing input from input devices, translating input into meaningful output, and routing output to Bluetooth or USB. Each input device is electrically connected to one or more general purpose input/output (GPIO) pins on the microcontroller. The firmware translates each GPIO pin to a particular mouse or keyboard function. For the touchpad, the GPIO pins are used to accept an X and a Y coordinate describing the location that the device is being touched. These coordinates are fed through an algorithm to generate the appropriate mouse movement.

The following flowchart (Figure 4) shows the behavior of the device's firmware from the time it is turned on.

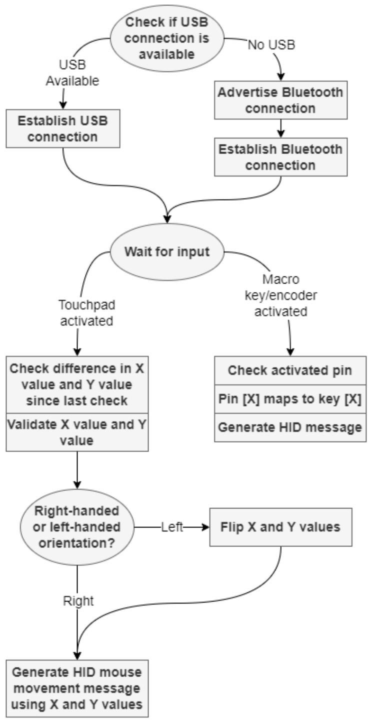

Check if USB connection is available
USB Available
No USB
Establish USB connection
Advertise Bluetooth connection
Establish Bluetooth connection
Wait for input
Touchpad activated
Macro key/encoder activated
Check difference in X value and Y value since last check
Validate X value and Y value
Check activated pin
Pin [X] maps to key [X]
Generate HID message
Right-handed or left-handed orientation?
Left
Flip X and Y values
Right
Generate HID mouse movement message using X and Y values

Fig. 4. Firmware flowchart.

Once the microcontroller's code is written and compiled, it is loaded onto the chip using the Arduino IDE. Before it can be loaded in this way, the Arduino bootloader must be installed on the chip using its SWD interface, represented by two pins on the circuit board. Once the bootloader and device firmware are loaded onto the chip, it need never be programmed again. It is not intended for users to ever reprogram the hardware of the device; therefore, the mouse and keyboard functionalities of the system are handled within firmware.

C. Application Software

The application software was created to be user-friendly, efficient, and sleek, with a modern design. The UI allows the user to create, delete, edit, search, and run macros all on the PC. The macros can then be triggered using the keypads or encoders selected within the application. The frontend displays these features in a simple way where the user is able to select preset macros in a dropdown list and search by name or macro type in the search bar.

The application intends to help users that might not be very familiar with automating tasks and creating their own shortcut macros. Our Python application was built with a TKinter GUI frontend base. It integrates with AutoHotkey, an open-source macro program, to compile and execute scripts running at the Windows OS level to automatically create different macros accessible by the different keys and encoders.

Running in the background, the Python code takes these selected macro objects which are then written and transferred to an .ahk script. Running this script will take the function keys associated with the keypads and encoders and remap them to a new macro automation shortcut feature.

Our application was designed with the user's experience in mind. Hence, our goals and objectives focused on the aspect of making the macro creation process as easy and repeatable as possible. For example, once the AutoHotkey script is running and remapping the function keys, the GUI application doesn't have to be open for the device's inputs to function properly.

Along with this, the data and information gathered through the program are stored in text files upon termination of the app. These files aren't necessary to run the application, but they remember quality-of-life data such as custom presets the user created, the macros that are currently selected that are represented on the device, and the application color theme (light, dark, or system).

With the application being built in Python and utilizing the different components of the AutoHotkey scripting language, all of the code was able to be converted into an executable file. Therefore, a blank Windows 10 machine can successfully run our application without having Python or AutoHotkey installed. The block diagram in Figure 5 shows how the application functions.

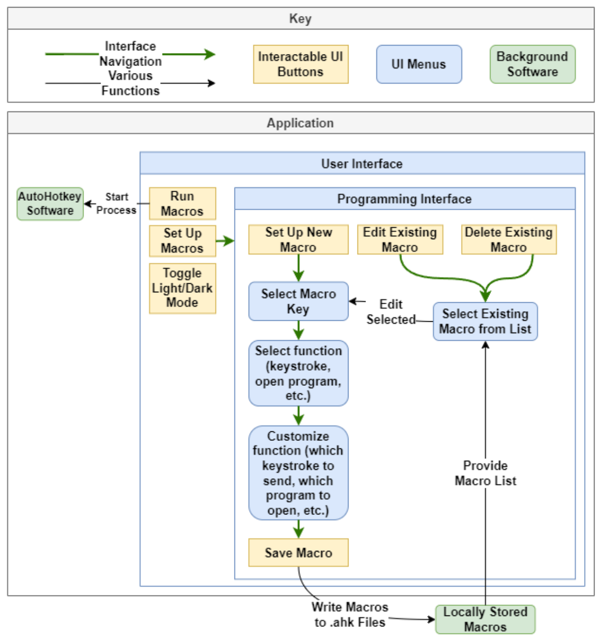


Fig. 5. Application block diagram.

IV. Device Components

As stated previously, the Programmable Trackpad hardware is comprised of a chassis, a circuit board, and various electronic components mounted on the circuit board. The following is a breakdown of each of the electronic components found on the board, how each one is used, and why each one was selected.

A. Mechanical Switches

An important concern of the project is to provide the consumer with options to change aesthetics and convenience. There is a currently existing market for custom keycaps, rotary knobs, switches, etc. and we wanted to reach out to those communities. One way to do this was to implement hot-swappable sockets. This allows the consumer to switch up the mechanical keys with ease, eliminating the pain of soldering each switch. Another aspect that comes with the implementation of mechanical switches is being able to switch up the keycaps.

The common part used as a mechanical switch for keys on PCBs is known as the Kailh hot-swappable socket. Since this is standard, we chose to implement it in our device.

For the default switches, we opted for Boba U4T switches with a 62 gram spring. These offer a tactile feel for the user while also having a quiet sound level. In the case the user is not happy with the default switch, because of the hot-swappable design, users can change the switches depending on their preferences. We chose these switches because, for the general public with little to no knowledge of the variety of mechanical switches, they provide a good balance between tactile feedback and a quiet experience offering the best of both worlds.

B. Rotary Encoders

There are not many options to choose from when designing a system with rotary encoders, since they all serve the same purpose and do not have many features to improve upon. The option we selected was a 24-pulse encoder with detents and haptic feedback. This option allows us to know the current position of the encoder through the microcontroller by measuring the number of clicks it has moved from its initial position.

If we were attempting to determine its rotational position, then a potentiometer would be a better option in this case, but, since the rotary encoder could be mapped to many different things the user might choose, it would most likely be the better option to use a rotary encoder, which does not have the rotational restriction of a potentiometer. The main difference between the two is that rotary encoders have a fully continuous rotation in either direction using digital signals whereas a potentiometer has a set direction in a clockwise or counter-clockwise direction using analog signals.

A typical use case for rotary encoders is changing a system's volume. Turning the encoder clockwise corresponds to an increase in volume; turning the encoder counter-clockwise corresponds to a decrease in volume. On a PC, there are also software controls for volume, so it makes sense to use a digital device (rotary encoder) rather than a rigid analog device (potentiometer).

C. Mouse Buttons

In order to function fully as a mouse, the device needs buttons for left and right clicking. Common convention for trackpad design is to implement these buttons directly beneath the touchpad. Our device uses four such buttons: two above the touchpad and two below. The two pairs of buttons exist to serve both left-handed and right-handed users, since the two different groups are intended to orient the trackpad differently.

The buttons themselves are simple push buttons mounted directly on the circuit board. In order to activate the buttons, the user presses down on metal strips that are anchored to the chassis. These metal strips are attached to standoffs that click the buttons when pressed down. This allows the user to have the clicking feedback of the buttons without needing the precision required for depressing the small buttons.

D. Orientation Switch

One other input device included on the Programmable Trackpad is the orientation switch. This is a physical switch that the user can toggle to switch between right-handed mode and left-handed mode. Any simple toggle switch can be used for this purpose. On our device, the switch is located on the outside of the hardware, so that the user can access it at any time. The switch is connected to the microcontroller, which handles the logic of switching from one orientation to another. This logic requires inverting the input on the touchpad.

E. Touchpad

The touchpad used in our device is the Adafruit Resistive Touch Screen. This touchpad is appealing because of its convenient size and low price. As the name implies, it is a resistive touchpad, which means that it senses pressure applied to its surface. This touchpad has a functional area of 3.7 inches diagonal, which is comparable to other trackpads on the market. In terms of power drain, this touchpad will drain the least amount of power on a battery operated system. It interfaces via a flat flex cable (FFC) with 4 pins. To integrate this module into our device, we use an FFC socket that is soldered directly onto the circuit board.

F. Battery

The device runs on battery power so that it can operate wirelessly. The battery selected is a 4.2-volt lithium-polymer (LiPo) battery. This type of battery is very common in embedded systems design, particularly in cell phones. We chose it for this project because of its low profile and built-in circuit protection.

Because the battery's fully-charged voltage is 4.2 volts, it does not interface directly with the device's other electronics, all of which operate at 3.3 volts. Instead, the battery's voltage is stepped down to 3.3 volts using an LM3671 buck converter. This chip was chosen because it was designed with 4.2-volt LiPo batteries in mind. The buck converter is used as the voltage source for the microcontroller and input units.

G. Battery Management System

One key advantage of the LiPo battery is that it is rechargeable. In order to charge the battery, there is an MCP73831 chip soldered onto the circuit board. This chip is a battery management system (BMS), which means that it can provide a constant current source to the battery until it is fully charged, at which point, it will cut off the constant current.

In our design, the BMS receives its power from the USB port. It converts the 5 volts from USB into a usable 4.2-volt source, which then charges the battery. Similar to the aforementioned LM3671, we chose the MCP73831 because it was designed with LiPo batteries in mind. As such, there is thorough documentation on how to set up the chip as a USB battery charger for such batteries. For our implementation, we adapted an electrical schematic that was found in the part's datasheet [1]. The following block diagram shows how the battery and battery management system interacts with the rest of the device electronics.

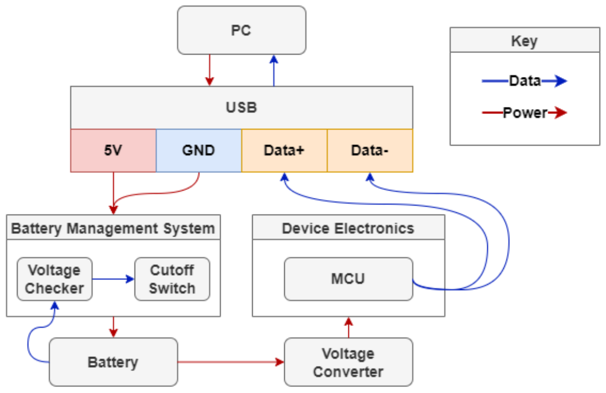


Fig. 6. Power system block diagram.

H. Microcontroller

The centerpiece of the circuit board is the microcontroller (MCU), an nRF52840 system-on-chip. This controller was chosen because it offers a robust variety of functionality across many pins, and also because it has native USB and Bluetooth support.

Each of the input units is connected to one or more of the GPIO pins on the MCU. Most of them are configured as digital inputs using the chip's built-in pull-up resistors, but the touchpad pins are configured as analog inputs. Various firmware libraries allow the MCU to generate Human Interface Device (HID) messages (cursor movements, key presses, and mouse clicks), and its native USB and Bluetooth interfaces can propagate those messages to the connected PC.

Because the nRF52840 supports several different methods of power input, we used the chip's datasheet [2] to design our circuit around the premise that it would receive 3.3 volts directly from a battery source. Figure 7 shows the schematic that we used for the microcontroller including the power options that we used from the datasheet as well as each of the input units on our device.

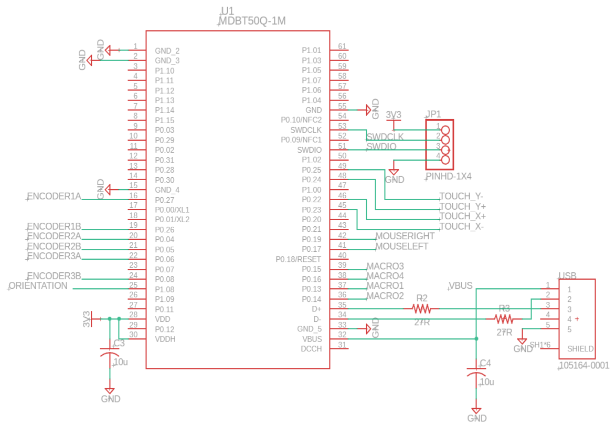


Fig. 7. Microcontroller schematic.

V. Development

After conceptualizing the project and researching the associated technology, our team went through multiple phases of prototyping. During the first phase of prototyping, we used development boards that implemented each of the chips we intended to use in the final product. We used this phase to prove that the components we had selected could function together as intended. At this stage, if there were any incompatibilities between parts, we could quickly pivot to another design/implementation of a system.

During the next phase of development, we created a custom breakout board for the microcontroller so that we could experiment with different configurations for its power input and GPIO pinout. The circuitry that we implemented on this breakout board using breadboards eventually was implemented in the final PCB after the breakout board served as a proof of concept. Figure 8 shows the custom board. The two connectors on the board are a USB connector for PC connectivity and a ribbon cable connector for the touchpad.

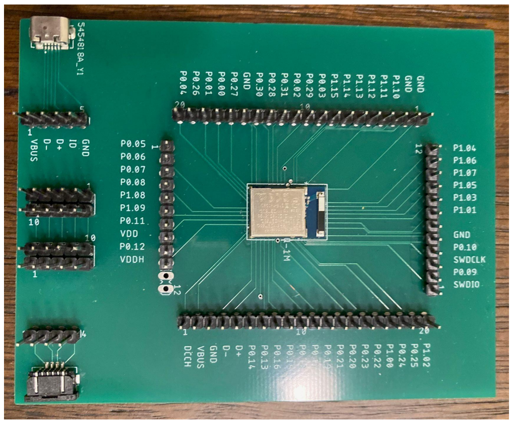


Fig. 8. Custom breakout board for microcontroller prototyping.

Our goal here with this development process was to mainly test the microcontroller and its capabilities. Having tested the microcontrollers on consumer development boards, we wanted to make sure it would work with ours. This included testing and uploading our firmware as well as implementing simultaneous battery charging and connection.

Initially using a third party J-link connected through SWD to our microcontroller was a huge hurdle in our development and ended up bricking two of our testing development boards. We had to look at using official J-link devices to program our microcontroller correctly and we were led to the PCA10056. This, on the other hand, was successful in uploading the bootloader for us to use the microcontroller with either Arduino or CircuitPython. However, we ran into another problem. CircuitPython was our main route we chose for configuring the GPIO pins on our custom board but anything we did to upload the CircuitPython uf2 bootloader onto our custom board, there was no luck into that being successful and with some searching on the internet and forums, this is a pretty common problem that hasn't been solved. This is most likely due to the microcontroller itself, but nevertheless we had to look at different options.

Arduino also has some configurations we could work through and that worked perfectly. The only problem with this route is the implementation of Bluetooth. Using community libraries and the native ones by the Arduino nRF52 Bluetooth, there was no luck in implementing a successful bluetooth connection with our custom development board. Although we had gotten the main functionality to fully work with our testing development board, we had hoped to not run into these problems with both CircuitPython and Arduino with the final design.

After prototyping the entire system using this breakout board, the next phase of development was final design.

VI. Final Design

The final PCB design implements all of the components listed in Section IV and addresses the hardware goals set in Table I. That includes three rotary encoders, four macro keys, a touchpad, left-click and right-click buttons in two orientations, a USB port, an orientation switch, and an on/off switch. The PCB design can be seen in Figure 9.

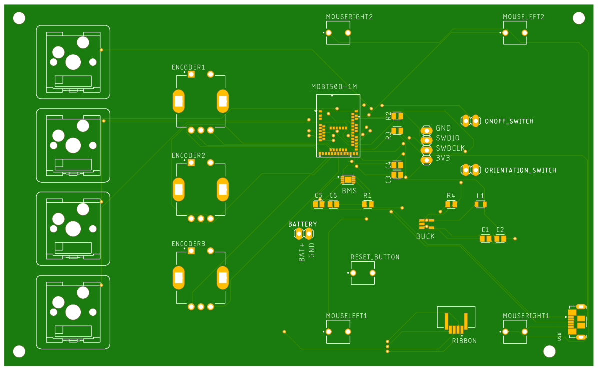


Fig. 9. Final printed circuit board design.

The final design for the device chassis was based on the dimensions of the PCB. It leaves an opening on one side for the encoders and macro keys, while the other side has a flat plane for the touchpad to be mounted. On the side of the chassis, there are holes cut out for the USB connector, orientation switch, and on/off switch. Standoffs are placed in the chassis lined up with the screw holes in the PCB. Figure 10 shows what the chassis model looks like.

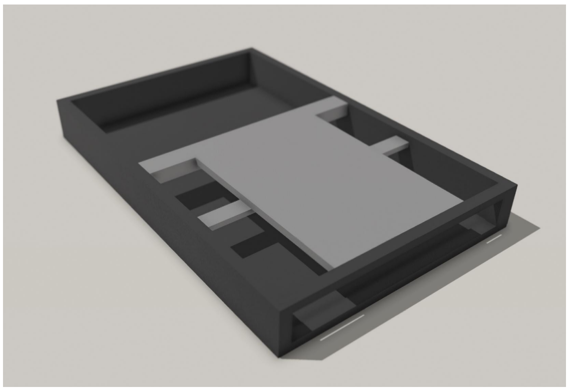
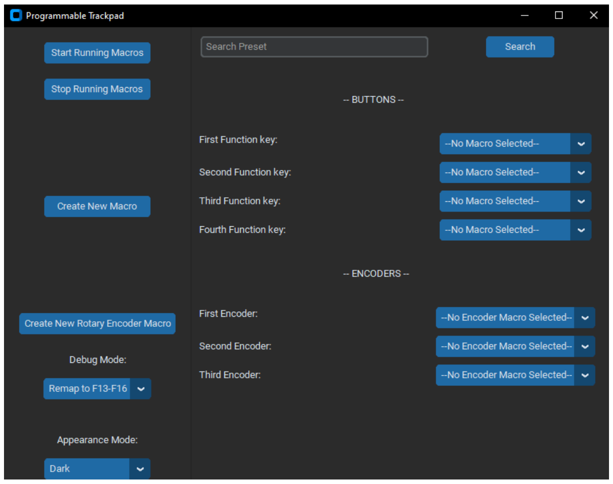


Fig. 10. Final chassis design.

The final application design fulfills all of the software goals set in Table I. It includes functionality for the user to create macros, assign macros to keys and encoders, store those macros on the PC, delete stored macros, and run the code to run macros in the background. Figure 11 shows what the user interface of the application looks like.



-- BUTTONS --

First Function key: --No Macro Selected--

Second Function key: --No Macro Selected--

Third Function key: --No Macro Selected--

Fourth Function key: --No Macro Selected--

-- ENCODERS --

First Encoder: --No Encoder Macro Selected--

Second Encoder: --No Encoder Macro Selected--

Third Encoder: --No Encoder Macro Selected--

Debug Mode: Remap to F13-F16

Appearance Mode: Dark

Fig. 11. Application user interface.

VII. Typical Use

To use the device, there are a few options the user can first explore: swap out the default switches, change the knob on the rotary encoder, or set the orientation of the device. Once these are considered we recommend plugging in the device to make sure the battery is charged. If the battery is fully charged, the user can also opt into using the device through Bluetooth. This can be done by adding the device into the Bluetooth connections on their computer. The user then can open the user application where all the magic happens between the device and the computer. The user must first create new macros for the desired actions where they can then be found in the dropdowns for each macro key and encoder. Once those are all configured, by pressing the “Start Running Macros” button in the top left corner, the application starts and the user will be able to use the device based on their custom configuration.

VIII. Conclusion

With the Programmable Trackpad, we believe that we have covered the market space for a niche product that did not exist before. Once we identified a potential need, we began brainstorming ways to fill that need. We decided that we would create a trackpad that has several macro-keys and rotary encoders that would map to user-defined shortcuts. Based on those and multiple other engineering requirements, we conducted thorough research on each component, and based the final design on the most practical application of each. Once we had settled on specific parts, we created block diagrams for each of our major systems and visualized how they would work together. From here, we tested a few prototype designs before settling on the final design.

This roadmap set us up for the execution of a successful project.

Through Senior Design at UCF, it gave us the opportunity to work as a team and encouraged us to apply the skills and knowledge we learned throughout our years at UCF.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Richard Blair with IDEM Systems for PCB assembly and guidance. We would also like to acknowledge Dr. Samuel Richie and Dr. Lei Wei for their support throughout the past two semesters of Senior Design.

References

[1] "MCP73831." Microchip, https://www.microchip.com/en-us/product/MCP73831

[2] Raytac. "MDBT50Q-1MV2." Raytac, https://www.raytac.com/product/ins.php?index_id=2