

# PC BUTTONPAD

By

Adit Umakanth

Stephanie Jaster

Yicong Dong

Final Report for ECE 445, Senior Design, Fall 2019

TA: Mengze Sha

11 December 2019

Project No. 12

## Abstract

Macro buttons are not new and exist on some high-end keyboards and specialized keypads on the market. Workflow of multiple actions are also existing features of many operating systems. One of our team members has experience using the stock Automator application of macOS, and it is still only on the software side with no hardware dedicated to make it easily accessible. In addition, the number of functions is limited and mostly only available on the stock applications.

Our project is different because it promises to enable users to customize the button functionality instead of relying on forced presets, spend a much lower cost, and still keep their favorite keyboards on the desk while having our solution in parallel with the existing solutions.

## Table of Contents

1 Introduction.....	1
1.1 Purpose.....	1
1.2 Functionality.....	1
1.3 Subsystem Overview .....	2
2 Design.....	4
2.1 Software Driver and User Interface .....	4
2.2 Control Unit.....	6
2.3 Sensors as User Interface .....	6
2.4 Additional Hardware User Interface .....	9
2.5 USB Power Supply .....	9
3 Design Requirements and Verification.....	10
3.1 Software Driver and User Interface .....	10
3.2 Control Unit.....	10
3.3 Sensors as User Interface .....	11
3.4 Additional Hardware User Interface .....	12
3.5 USB Power Supply .....	12
4 Cost and Schedule .....	13
4.1 Cost Analysis.....	13
4.2 Schedule .....	15
5 Conclusion .....	16
5.1 Accomplishments .....	16
5.2 Uncertainties.....	16
5.3 Ethical Considerations .....	16
5.4 Future Work .....	18
References.....	19
Appendix A Requirement and Verification Tables.....	20

# 1 Introduction

## 1.1 Purpose

Keyboards and mice work well with simple tasks but are not the quickest way to interact with computers. There are some actions people perform on a regular basis that take many mouse clicks or keyboard button presses, which are not only tedious but also tiring and time consuming to perform but could be condensed down into one simple press of a button. Such actions include but are not limited to opening certain pages altogether in a browser, quickly adjusting volume and brightness of the PC, taking screenshots and sharing to social media with one click, etc. In order to increase productivity, decrease the repetitiveness of working on a computer, and enhance the experience and fluidity of working on a computer, we strongly recommend the engineering of more user-friendly hardware designs of the mentioned characteristics.

We therefore propose to make a compact and portable unit with sensors, buttons and LEDs that can be connected and used on the three major operating systems: Windows, macOS, and most distributions of Linux. These buttons will perform whatever repeated actions the user assigns it, and the pad does not need other power supply except the output from the Universal Serial Bus (USB). Sensors and LEDs shall provide more advanced features that further assist the user with work.

## 1.2 Functionality

The high-level requirements are as follows:

1. PC Buttonpad and accompanying software driver should work on the three current versions of mainstream PC operating systems: Windows 7+, macOS 10.11+, and recent versions of popular Linux distributions.
2. Button functions should be easily customizable without any specific knowledge on coding or hardware. Someone who only uses a computer for word documents/web browsing should have no problem using the driver software.
3. The rotary encoder, touch sensors and ultrasonic sensors should work responsively while the user intends to interact with them while staying idle otherwise, i.e. not give erroneous inputs.

*High-level Requirement 1* guarantees usability across a wide variety of computers. Most existing macro keyboards have software limited to only one operating system. This is highly inconvenient for users who do not use a mainstream operating system or users that use multiple operating systems.

*High-level Requirement 2* ensures that a wide variety of users can benefit from the PC Buttonpad. Some existing solutions require domain-specific knowledge and alienates a large percentage of users who do not have the required knowledge to use those solutions.

*High-level Requirement 3* safeguards the user from erroneous input and potentially unsafe errors. An input device with erroneous sensors has the potential to cause a lot of damage to the user (e.g. accidentally deleting files, sending incomplete emails, etc.).

## 1.3 Subsystem Overview

### 1.3.1 Block Diagram

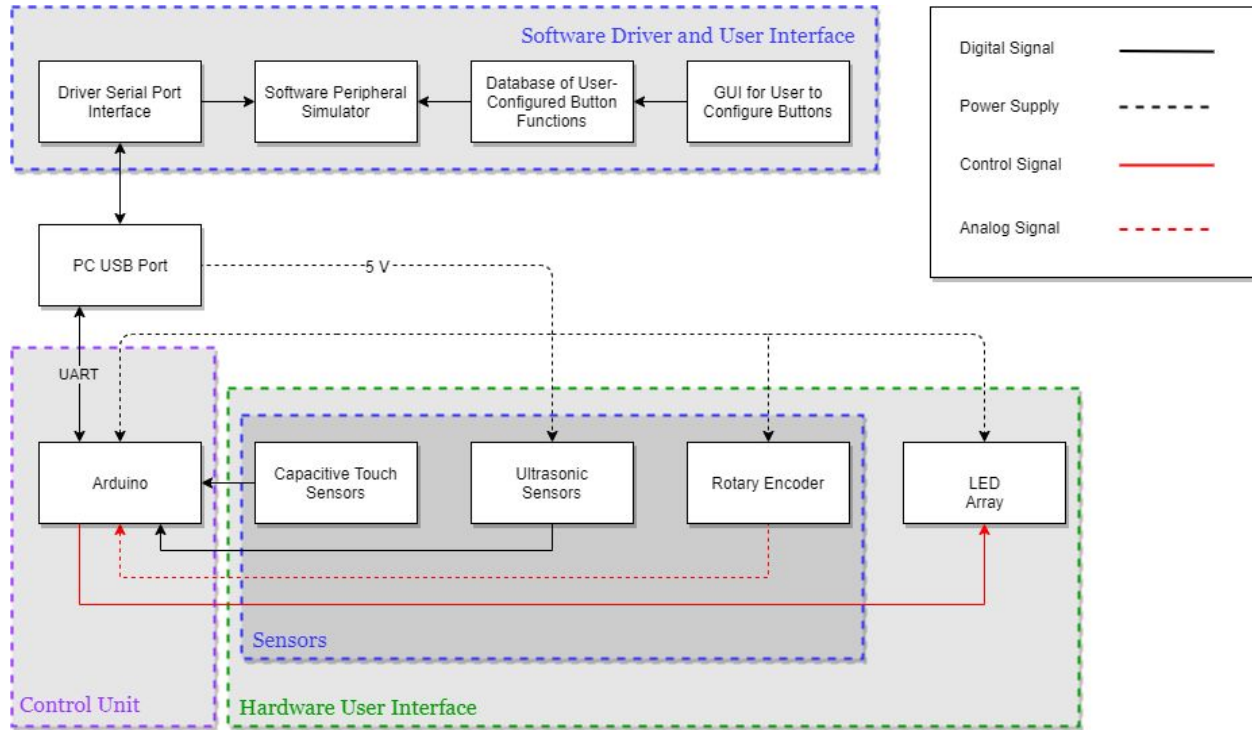


Figure 1. Block diagram of PC Buttonpad.

As illustrated in Figure 1, the capacitive touch sensor detects button presses and sends a signal to the connected computer through an Arduino. The connected computer has drivers installed which decodes these signals and performs the user-assigned actions. The USB port of the user's computer is expected to power the whole circuit. Additional sensors serve as part of the user interface and provide extended features and functionalities.

### 1.3.2 Software Driver and User Interface

The software driver and user interface, as the name implies, is a subsystem that exists entirely on the software. It reads in signals sent by the microcontroller to perform actions and provides an interface to the user to freely customize the actions and the buttons that they are mapped to.

### 1.3.3 Control Unit

Since our project is a blend of hardware and software, the main control unit will be the Arduino on the hardware side and the driver program on the software side. These two modules communicate with each other through the USB Serial Port and a Universal Asynchronous Receiver-Transmitter (UART) to convey data like battery levels of the connected computer and the buttons pressed.

### **1.3.4 Sensors as User Interface**

The sensors are a core part of the PC Buttonpad and will be what the user interacts with on a frequent basis. The capacitive touch sensors under the buttons serve to detect when the user presses a button, and ultrasonic sensors are used to detect hands-free gestures that can switch profiles or provide additional functionality. A rotary encoder used as a turn-dial provides further functionality and ease-of-use by allowing the user to change things like volume and screen brightness right on the PC Buttonpad.

### **1.3.5 Additional Hardware User Interface**

In addition to the sensors mentioned above as part of the user interface, we also include an LED array to indicate the battery level of the PC.

### **1.3.6 USB Power Supply**

We will be using the 5 V power supply from the USB port to power the Arduino, which then provides power to all the other subsystems.

## 2 Design

### 2.1 Software Driver and User Interface

All the software modules were implemented in Electron [1] (a JavaScript framework for building desktop applications), Vue.js [2] (JavaScript front-end framework for building reactive user interfaces), and Python [3] (a powerful cross-platform scripting language with many third-party modules). The Electron and Vue.js code is packaged together and forms the main part of the interface, but it also calls Python processes to perform low-level tasks such as I/O on the serial port and handling keyboard shortcuts.

The software has been named PC Buttonpad Manager, or PCBM for short.

#### 2.1.1 Driver Serial Port Interface

This is the piece of software that allows the Arduino to communicate with the user's computer. A press of the button or trigger of an ultrasonic sensor is passed on to the Arduino and sent to the USB Serial Port of the PC, which is listened to by this piece of software. It carries out the relevant actions depending on the signals from the hardware.

The driver serial port had many different implementation possibilities. The two main options were Node SerialPort [4] and pySerial [5]. Node SerialPort would run within the main Electron application while pySerial works in Python and would have to be launched as a separate process. Initial tests with SerialPort gave lots of errors on opening communication with serial ports, so we used pySerial instead since it satisfied the requirements.

#### 2.1.2 Software Peripheral Simulator

Instead of having the hardware mimic the signals of a keyboard or mouse, we use the serial port interface to read a custom signal, and then use this Software Peripheral Simulator module to simulate the click of a mouse or typing on a keyboard. This offloads all the limitations of storing information for each button onto the computer instead of our hardware. It also allows further customizations beyond keyboard and mouse interactions such as running shell commands or specific libraries for certain software if required. We will use an open-source library for this module instead of creating this functionality from scratch.

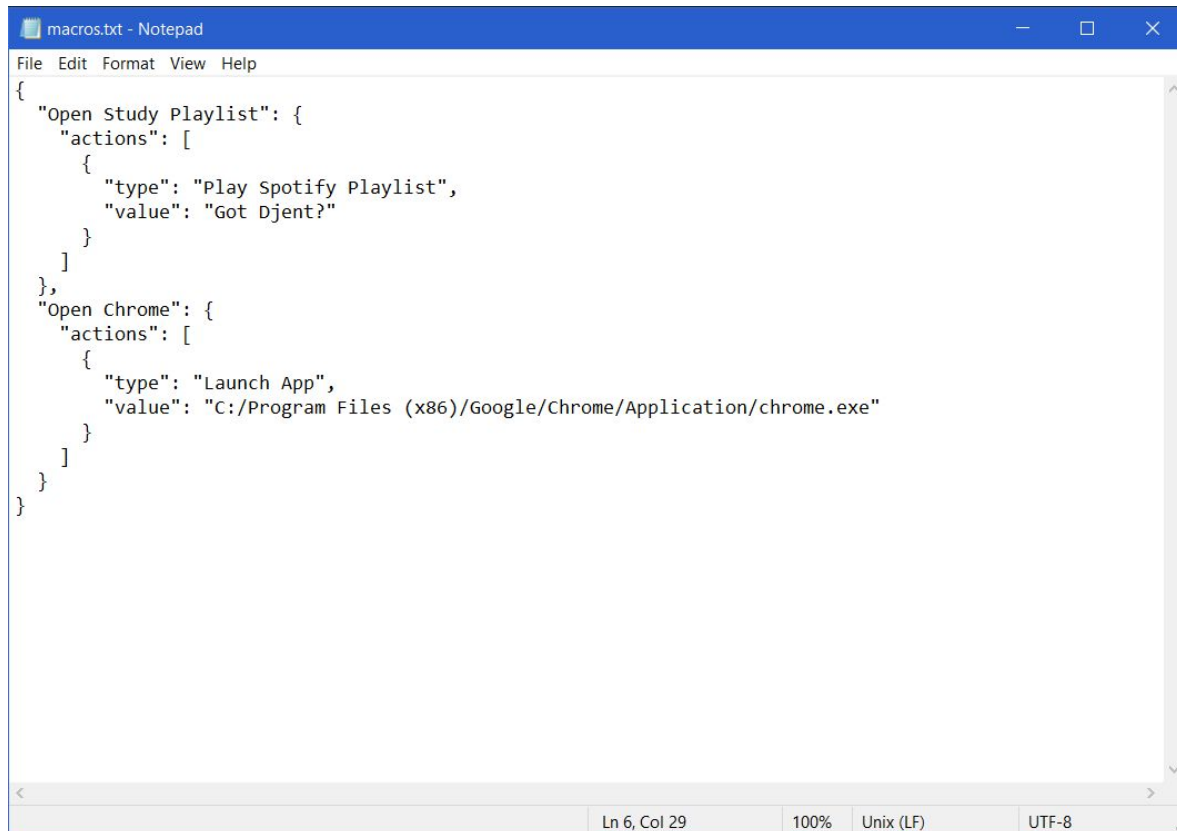
As with the driver serial port, there were a few options. Node.js has a library called kbm-robot [6] that provides the required functionality. However, during testing, we found that this module required a lot of other dependencies, some of which violated the cross-platform requirement. Therefore, we used keyboard [7], a Python module that works across the required operating systems and provides identical functionality.

#### 2.1.3 Database of User-Configured Buttons

This is where all user-saved keyboard and mouse shortcuts are stored. Whenever a signal is received through the serial port, this database is checked to find out what needs to be executed by the software

peripheral simulator. The user does not have direct access to this database but can freely and easily make changes to it through the GUI for User to Configure Buttons.

Instead of using a separate database application to keep track of the information, we used a Node.js file parsing library to simply save this information into a text file. Since the information stored is relatively small (up to a few MB and not GB), this method works perfectly, and all the data can be accessed easily. Figure 2 shows an example of macros saved by a user.



```
{
  "Open Study Playlist": {
    "actions": [
      {
        "type": "Play Spotify Playlist",
        "value": "Got Djent?"
      }
    ]
  },
  "Open Chrome": {
    "actions": [
      {
        "type": "Launch App",
        "value": "C:/Program Files (x86)/Google/Chrome/Application/chrome.exe"
      }
    ]
  }
}
```

Figure 2. Screenshot of text file used to hold macros saved by the user.

#### 2.1.4 GUI for User to Configure Buttons

This module is the core of customizing the buttons to the user's liking. An easy-to-use GUI will allow the user to choose specific buttons and assign desired keyboard shortcuts or command line executions. Since it was a user-interface, many design choices had to be taken on how best to make this software as user-friendly as possible.

The interface was split into three different tabs, namely: Home, Manage, and Assign. The Home tab contains an introduction and helpful tips on how to use the software. The Manage tab provides the user with an interface to create macros and actions. These created macros can then be assigned to particular buttons in the Assign tab. Figure 3 shows the interface of the Manage tab that has the macro "Open Chrome" created.



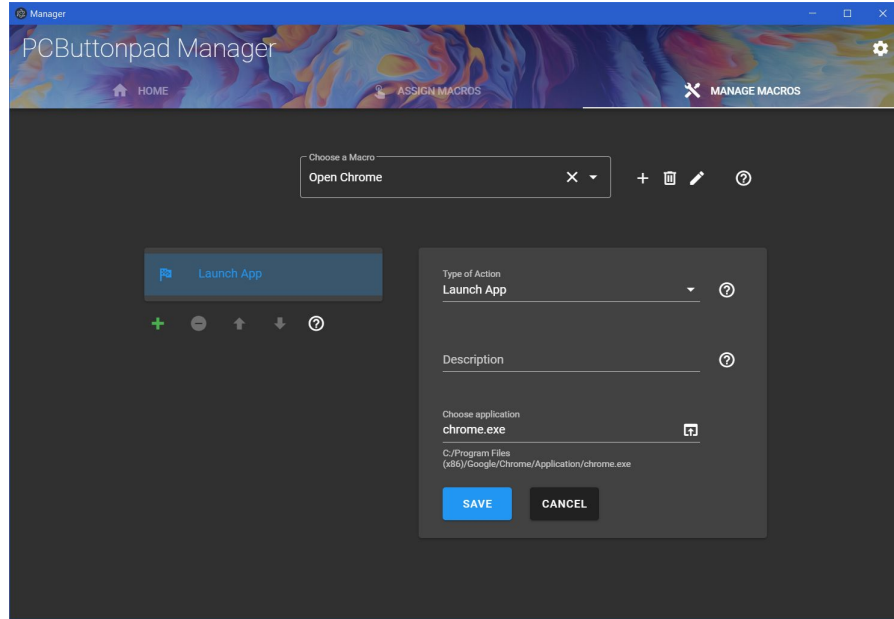


Figure 3. A screenshot of PCBM’s Manage Tab.

## 2.2 Control Unit

The control unit acts as a bridge between the hardware and the software. The microcontroller constantly reads inputs from the different sensors and sends data to the user’s computer via the USB port based on the inputs read from the sensors. The driver serial port interface decodes the data sent by the microcontroller and performs any required actions based on the data.

We decided to use the UART protocol since this is supported by the hardware on the Arduino and can be easily managed in code by using the Serial class. If we were using only an ATmega328, we would need additional hardware such as a FT232R USB UART IC [8] to facilitate the communication between the microcontroller and the computer.

## 2.3 Sensors as User Interface

### 2.3.1 Capacitive Touch Sensors

Since we do not need high resolution for each single button, it is best to build capacitive touch sensors using surface capacitance, where “only one side of the insulator is coated with conductive material. A small voltage is applied to this layer, resulting in a uniform electrostatic field. When a conductor, such as a human finger, touches the uncoated surface, a capacitor is dynamically formed” [9]. We built nine of these sensors and placed them in a 3×3 grid-fashion on the PC Buttonpad. Capacitive touch sensors ensure that there are no mechanical parts in the buttons, making the design simpler and more durable. The capacitive touch sensors were built from scratch including timer chips, second-order low-pass filters, and encoders and comparators to convert analog signals to digital signals. When the finger touches a button, it is equivalent to a capacitor and the RC time constant would increase significantly – this shall detect the touch.

For the analog equivalent solution of the capacitive button, we would be using a “timer circuit to generate a frequency that is inversely proportional to capacitance and then utilize a microcontroller to count pulses within a given period to calculate the frequency.” Figure 4 shows the schematic design [10].

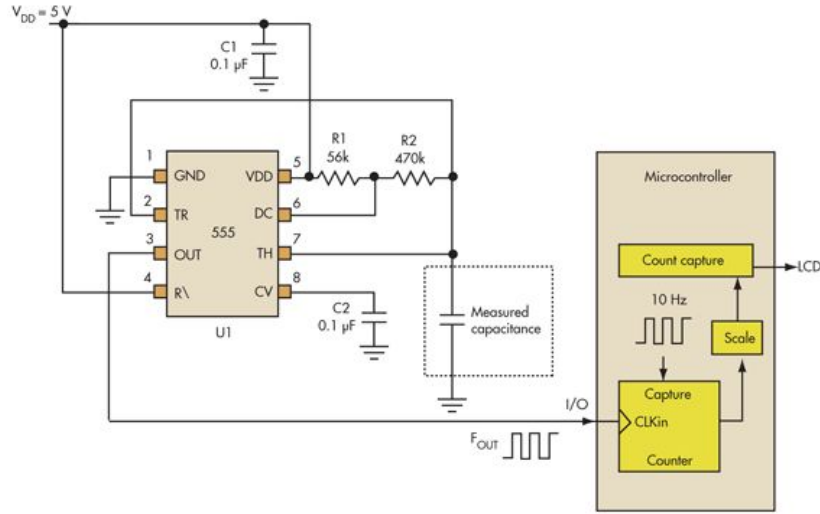


Figure 4. Digitizing the value of a capacitive sensor often involves generating a frequency that is inversely proportional to the capacitance and counting pulses over a fixed period to determine the frequency [10].

The frequency calculation will be performed using Equation (1) as follows [10]:

$$f = (C(R_1 + 2R_2)\ln(2)) - 1 \quad (1)$$

What we have added is basically a second-order low-pass filter at the end of each of the timer outputs to transform the periodic pulse to DC signal. With that DC signal, we use a comparator and a reference voltage to further transform it to high/low digital signal such that the controller knows if we are touching or not touching the button. After testing, we decided that all capacitors used are 1  $\mu$ F; choices of the resistors  $R_1$  and  $R_2$  are both 470 k $\Omega$ ; low-pass filters use 1 k $\Omega$  and 2.2 k $\Omega$  respectively.

Alternatively, we could have chosen to use two digital pins from the Arduino for each button so that the button designs themselves become much simpler. However, as we still need to reserve a lot of pins to the ultrasonic sensors, LEDs, etc., we believe it is safer to use the analog solution and digitize the signals on our own so that each button requires only one digital pin.

### 2.3.2 Ultrasonic Sensors

This sensor really comes down to measuring how far an object is. Using the time it takes for sound to travel to the object and reflect to get received, it can measure the distance through a simple multiplication. In our case, we allow a certain range of distance ( $\leq 15$  cm) that the user can wave his/her hand passing by two ultrasonic sensors, and this shall allow the controller to figure out if the movement is to the left or to the right. With this movement, the profile of the PC Buttonpad switches and other functionalities can be

activated. These sensors will be properly arranged at the upper end of the PC Buttonpad such that using the buttons on the design would not trigger the ultrasonic sensors mistakenly. Since the ultrasonic sensors output digital signals of the distance information directly, no encoders or any other circuitry is needed.

Figure 5 displays the timing diagram that each ultrasonic sensor follows in order to collect data. A trigger signal of 10  $\mu$ s is sent to the trigger pin of the ultrasonic sensor and a timer is started. Once the timer is started, eight 40 kHz ultrasonic bursts are sent out of the transmitting end of the ultrasonic sensor. Each of these ultrasonic bursts is reflected to the receiving end of the ultrasonic sensor once an object is detected. If an object is not detected, the ultrasonic bursts will reflect off an object in the background and noted as a maximum distance. Once the echo is received, the timer stops and the time measured is converted into a distance value as shown in Equation (2). This cycling period for each ultrasonic sensor can be easily adjusted in the Arduino code.

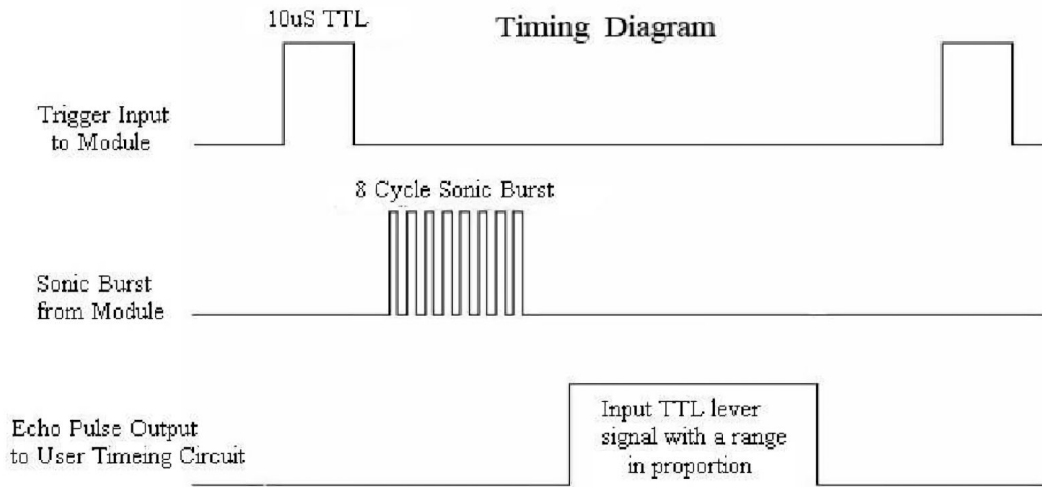


Figure 5. Timing diagram for each ultrasonic sensor [14].

The distance ( $D$ ) is calculated using the following equation, where *time* refers to the time measured after each timer is stopped [14]:

$$D = (\text{speed of sound in air} \times \text{time})/2 \quad (2)$$

Once both ultrasonic sensors were serially connected, Arduino code was written to allow for swipe detections. To do this, the distances calculated for each ultrasonic sensor were compared. If the distances were the same or within 3 cm of each other, no swipe was detected. If one distance was greater than the other, than the swipe detection process would begin until the opposing ultrasonic sensor has a greater distance. For multiple cycling periods, the distances would be measured and compared until the distance of the opposing ultrasonic sensor is larger. From the order that these gaps in distance are detected, a left swipe or right swipe can be concluded.

During the integration period of our project, the ultrasonic sensors were unable to perform at the same level. We think that combining the ultrasonic sensor code with the rest of the software and hardware code

threw off the timing of the ultrasonic sensors and caused them to time out. Another potential cause for the malfunction may be the amount of current being supplied to the ultrasonic sensors after they were connected to the other sensors. Originally, the ultrasonic sensors were directly connected to the 5 V pin of the Arduino and ground. In the final design, the capacitive touch sensors were connected between the power and ultrasonic sensors. The capacitive touch sensors had changing capacitance values when touches were detected, so the amount of current going to the ultrasonic sensors would frequently change. A design alternative could require a separate power supply for the ultrasonic sensors that does not connect to other sensors. This can ensure a steady current flow to each of the ultrasonic sensors. Additionally, the Arduino code could be changed in a way that the ultrasonic sensor timing is not affected by the rest of the design implementation.

### **2.3.3 Rotary Encoder**

This device has one ground pin and two signal pins disconnects to the ground when it is turned. As we produce two periodic pulses through the device, a turn in either direction is going to modulate the phases of the two periodic pulses differently, which tells the direction of turning to the Arduino.

## **2.4 Additional Hardware User Interface**

As for the number of LEDs, we used four green LEDs and linearly indicate the battery percentage value to the closest proportional ceiling. For example, for four LEDs, 60% battery level would have three LEDs on and one off as it is between 50% and 75%. We planned to use logic gates to decode the bits from the controller to decide on which LED(s) to light up, but during the testing, some of the gates smoked and burned down due to too much current going through them to the LEDs. In the end, we decided to connect the LEDs with resistors directly to the Arduino and that would take four digital pins total.

## **2.5 USB Power Supply**

Since we are using the power supply directly from the USB port, there is not much to design for this module except the choice of port shape. Most laptops use a USB Type-A port, so we decided to use Type-A, also for convenience of getting cables for the Arduino.

Alternatively, we could have used a battery array to power up the system, but since only a small amount of power consumption are expected for the buttons, LEDs, etc. In that sense, we think the 5 W power supply from a standard USB 3.0 port should be much more than enough.

Another alternative is for us to use 3.3 V because of stability issues, but we planned for a 5 V microcontroller at the beginning and did not put the regulator into our PCB design as a result.

## 3 Design Requirements and Verification

The requirements and verification tables for each component can be found in Appendix A Tables 5–14.

### 3.1 Software Driver and User Interface

#### 3.1.1 Driver Serial Port Interface

Table 5 describes the requirements and verifications for the driver serial port interface. Since we used a pre-built Arduino, the verifications were straightforward and involved writing simple code using the Serial library on the Arduino and the pySerial library for Python on the computer to decode the data sent from the Arduino.

#### 3.1.2 Software Peripheral Simulator

Requirement 1 in Table 6 was easy to verify and involved writing a short program in Python with some test shortcuts. The tested shortcuts were “Ctrl+W” to close browser tab, “Ctrl+T” to open a new browser tab, “Alt+Tab” to switch windows, and “Ctrl+Alt+Delete” to open task manager. All these shortcuts carried out the desired action when run through the software simulator.

Requirement 2 was somewhat harder to verify, as it required a Windows, Linux, and a macOS computer. Our group did not have a Linux computer, so we loaded our software onto a virtual machine running Manjaro 18.1.4 to test the peripheral simulator. All shortcuts worked except “Ctrl+Alt+Delete”, which is Windows-specific so not expected to work on Linux.

#### 3.1.3 Database of User-Configured Buttons

The requirements as described in Table 7 were easy to verify once the software was written. The JavaScript object was successfully parsed from the text file and executed by the Python process.

Requirement 2 was verified by simply looking at the JavaScript object limit that is ~50 MB, which is far more than required to hold a sequence of <100 keys.

#### 3.1.4 GUI for User to Configure Buttons

Table 8 has requirements that were verified once the software was completed. The GUI allows the user to choose a sequence of actions and keyboard shortcuts within the 100-key limit.

Requirement 2 can be verified by running the software, creating a sequence with the Manage tab editor, and ensuring that the sequence has been written to the macros file on persistent storage.

### 3.2 Control Unit

The requirements on Table 9 became far easier to verify since forgoing the ATmega328 IC for the Arduino instead. The Arduino has far more in-built functionality and allowed us to verify Requirement 1 simply by writing some code on the Arduino and on pySerial. Requirement 2 took longer to verify because it needed the capacitive sensor to be built, but once that was done, the requirement was verified by connecting the sensors to the Arduino and running code that read from those sensors and output the values to ensure that they were within the expected ranges.

### 3.3 Sensors as User Interface

#### 3.3.1 Capacitive Touch Sensors

For the selected resistors and capacitors, we tested both on breadboard and PCB. On PCB, the analog voltage after the second-order low-pass filter without touching is 4.06 V; if touched and released, the voltage is  $< 3.7$  V; if kept pressed, the voltage is 3.43 V. Likewise, on the breadboard, the voltage when touching reads to be 4.3 V while the voltage when not touching reads to be 3.4 V. We also found the best reference voltage to the comparator to be 3.81 V generated from 5 V voltage division using a 10 k $\Omega$  and 33 k $\Omega$  resistor combination. Oscilloscope graphs of an earlier working capacitive button testing (not the resistor combinations described above) are attached here as Figure 6. for reference. The voltage stays constant unless we physically touch the buttons, so it is consistent with the requirements in Table 10.

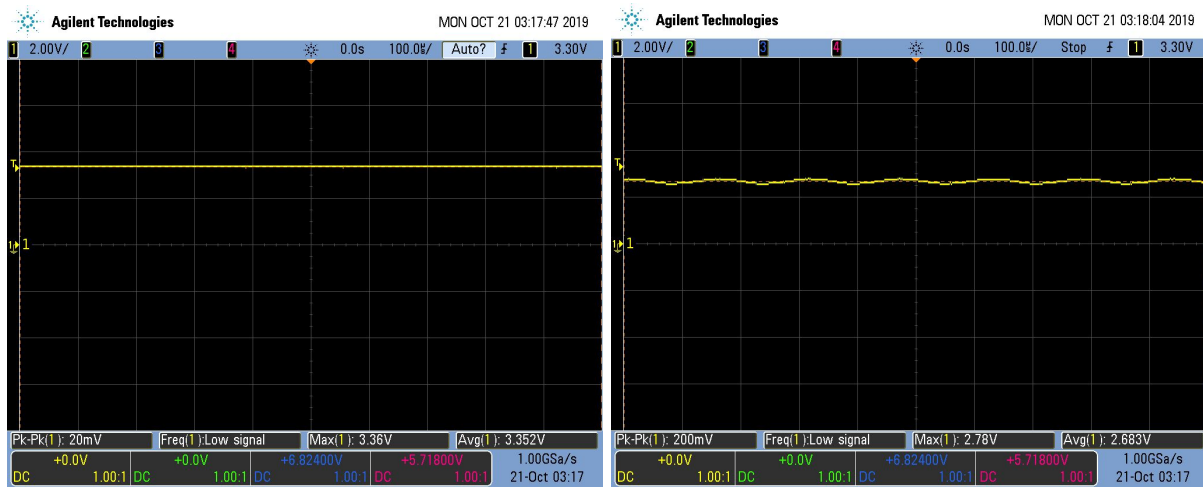


Figure 6. Timer output signals with second-order low-pass filter untouched (left) and touched (right).

#### 3.3.2 Ultrasonic Sensors

For both ultrasonic sensors used, there were four requirements with corresponding verifications as listed in Table 11. The ultrasonic sensors were required to measure distances between 2 cm and 1 m to make sure that signals were not interfering with each other. The distances of both ultrasonic sensors were measured individually and then together. When combined, the distances were able to reach up to 170 cm before the data started to become inaccurate. For the application of the PC Buttonpad, a distance range between 2 cm and 15 cm was necessary – anything above 15 cm was not measured. The ultrasonic sensors also required an angle of measurement of 15°, as explained in the datasheet [14]. The angle of measurement could not be tested to the degree, but the point of this requirement was to make sure the angle of measurement was not wide. If the angle was wide, then a user's hand could be detected when touching a button below the ultrasonic sensors. We found that our hand must be placed directly above the ultrasonic sensors in order to detect motion and confirmed that the angle of measurement is narrow. Another requirement was that both ultrasonic sensors should work when placed adjacently. Table 1 below displays the test results when connecting both ultrasonic sensors together. Through these tests, we were

able to learn that the ultrasonic sensors must be connected in series in order to collect proper distance measurements. This means that the data of the first ultrasonic sensor must be collected before the second ultrasonic sensor can start to collect data. Once this connection was achieved, we were able to test the swipe functions.

Table 1. Ultrasonic sensor pin tests.

Trial #	trigPin1	echoPin1	trigPin2	echoPin2	distance1	distance2	Notes
1	9	10			Correct		One sensor
2	9	10	5	6	Correct	0	Two sensors in parallel
3	9	10	3	5	Correct	0	Two sensors in parallel
4	5	6	9	10	Correct	0	Two sensors in parallel
5	3	5	9	10	Correct	0	Two sensors in parallel
6			9	10		Correct	First sensor disconnected
7			5	6		Correct	First sensor disconnected
8	9	10	5	6	Correct	Correct	Two sensors in series

The final requirement was that the cycling period for each ultrasonic sensor must be greater than 60 ms. The minimum cycling period value could be controlled in the Arduino code, so we set the minimum cycling period to 75 ms to avoid overlapping signals of different cycling periods.

### 3.3.3 Rotary Encoder

By connecting the rotary encoder to the Arduino directly taking three pins, we printed the number in positive or negative increments to indicate the direction, and we also managed to store the number cumulatively if needed for any other purposes. Each turn only registered one number, and the signal is never mistaken by the Arduino, which agrees with Table 12.

## 3.4 Additional Hardware User Interface

The LEDs lit up successfully without burning down according to the requirements found in Table 13. The illumination is not verified because we do not have the equipment in the lab to make such measurement, and in some ways the LEDs are clearly visible to naked eyes already.

## 3.5 USB Power Supply

We tested the power of the entire circuit using the power supply in the lab according to Table 14, and the power consumption read on the supply is way less than 2.5 W with the voltage very stable at 5 V. We did not take note of the current, but it was definitely no more than 10 mA, and USB 3.0 could not have any problem with providing such power.

## 4 Cost and Schedule

### 4.1 Cost Analysis

Outlined in Tables 2–3 are the cost of parts used in the final PC Buttonpad design and total cost of parts purchased throughout the semester. Labor costs and the total costs of the PC Buttonpad project are also included in this section.

#### 4.1.1 Parts Cost

Cost of parts purchased that actually went into our project can be found below in Table 2. Table 3 displays the total cost of all parts purchased throughout the semester.

Table 2. PC Buttonpad parts cost.

Description	Manufacturer	Digikey Part #	Unit Price (Prototype)/\$	Unit Price (Bulk)/\$	Quantity	Proto Sum/\$
<b>LED</b> GREEN DIFFUSED T-1 3/4 T/H	Lite-On Inc.	160-1706-ND	0.36	0.04416	9	3.24
IC GATE <b>OR</b> 1CH <b>2-INP</b> SOT23-5	Texas Instruments	296-1093-1-ND	0.33	0.07107	26	8.58
IC GATE <b>OR</b> 2CH <b>4-INP</b> 14SOIC	Texas Instruments	296-25954-5-ND	0.49	0.1624	6	2.94
IC GATE <b>AND</b> 1CH 2-INP SC70-5	Texas Instruments	296-8743-1-ND	0.33	0.07107	9	2.97
IC <b>INVERTER</b> 1CH 1-INP SC70-5	Texas Instruments	296-1090-1-ND	0.33	0.07107	5	1.65
IC OSC SGL <b>TIMER</b> 2.1MHZ 8-SOIC	Texas Instruments	296-1336-1-ND	0.75	0.31971	10	7.5
IC OSC <b>TIMER DUAL</b> 2.1MHZ 14-SOIC	Texas Instruments	296-1338-1-ND	1.32	0.59792	10	13.2
IC <b>COMPARATOR</b> LP DUAL 8-SOIC	STMicroelectronics	497-1593-1-ND	0.37	0.09228	5	1.95
IC DIFF <b>COMP SINGLE</b> SOT23-5	Texas Instruments	296-10168-1-ND	0.58	0.22403	3	1.74
<b>ROTARY ENCODER</b> MECHANICAL 20PPR	TT Electronics/BI	987-1185-ND	0.99	0.47600	2	1.98
<b>IC EXTRACTOR</b>			3.08		1	3.08
CMOS <b>TIMER</b> TLC555CP	Texas Instruments		0.87		2	1.74
<b>Total</b>						50.57



Table 3. Total parts cost.

Description	Manufacturer	Digikey Part #	Unit Price (Prototype)/\$	Unit Price (Bulk)/\$	Quantity	Proto Sum/\$
<b>LED GREEN DIFFUSED T-1 3/4 T/H</b>	Lite-On Inc.	160-1706-ND	0.36	0.04416	9	3.24
<b>ULTRASONIC SENSOR DISTANC US-100</b>	Adafruit Industries LLC	1528-2789-ND	6.95		2 (lab)	0
<b>USB-C BREAKOUT</b>	SparkFun Electronics	1568-1958-ND	4.5		2	9
<b>IC GATE OR 1CH 2-INP SOT23-5</b>	Texas Instruments	296-1093-1-ND	0.33	0.07107	26	8.58
<b>IC GATE OR 2CH 4-INP 14SOIC</b>	Texas Instruments	296-25954-5-ND	0.49	0.1624	6	2.94
<b>IC GATE AND 1CH 2-INP SC70-5</b>	Texas Instruments	296-8743-1-ND	0.33	0.07107	9	2.97
<b>IC INVERTER 1CH 1-INP SC70-5</b>	Texas Instruments	296-1090-1-ND	0.33	0.07107	5	1.65
<b>IC OSC SGL TIMER 2.1MHZ 8-SOIC</b>	Texas Instruments	296-1336-1-ND	0.75	0.31971	10	7.5
<b>IC OSC TIMER DUAL 2.1MHZ 14-SOIC</b>	Texas Instruments	296-1338-1-ND	1.32	0.59792	10	13.2
<b>IC MCU 8BIT 32KB FLASH 28DIP</b>	Microchip Technology	ATMEGA328P-PU-ND	2.14	1.78	2	4.28
<b>IC REG LINEAR 3.3V 2A 20HTSSOP</b>	Texas Instruments	296-18155-1-ND	10.81	6.13071	2	21.62
<b>SWITCH ROTARY 12POS 2.5A 125V</b>	C&K	CKN10200-ND	5.97	3.85429	2	11.94
<b>IC COMPARATOR LP DUAL 8-SOIC</b>	STMicroelectronics	497-1593-1-ND	0.37	0.09228	5	1.95
<b>IC DIFF COMP SINGLE SOT23-5</b>	Texas Instruments	296-10168-1-ND	0.58	0.22403	3	1.74
<b>ROTARY ENCODER MECHANICAL 24PPR</b>	TT Electronics/BI	987-1399-ND	0.81	0.39200	2	1.62
<b>ROTARY ENCODER MECHANICAL 20PPR</b>	TT Electronics/BI	987-1185-ND	0.99	0.47600	2	1.98
<b>IC EXTRACTOR</b>			3.08		1	3.08
<b>CMOS TIMER TLC555CP</b>	Texas Instruments		0.87		2	1.74
Adafruit <b>FTDI Friend</b> + Extras	Adafruit Industries		14.87		1	14.87
<b>Arduino</b>					3	0
<b>Grand Total</b>						113.9

### 4.1.2 Labor Cost

Undergraduates are usually paid somewhere above 10 US Dollars/hour. Assume a 20 US Dollars/hour salary for each of the team members, and approximately 10 hours of work per week for each person. For a total of 16 weeks in the semester, the total labor cost would be 9,600 US Dollars.

### 4.1.3 Total Cost

As mentioned in Table 3, the total cost for parts of this project is 113.90 US Dollars while the labor cost is 9,600 US Dollars which would equal 9,713.90 US Dollars total for this project. However, as design components changed for this project some of the parts listed in Table 3 were not used for the final prototype. The final PC Buttonpad prototype cost would be 50.57 US Dollars, as seen in Table 2, and 9,650.57 US Dollars including the labor cost.

## 4.2 Schedule

The schedule that the team planned for the semester is found below in Table 4.

Table 4. Schedule with work distribution.

Week	Adit	Stephanie	Yicong
10/07/19	Working on GUI and assisting with hardware prototyping	Hardware prototyping; figure out ultrasonic sensor spacing issues	
10/14/19	Refining the GUI and assisting with Eagle CAD	Complete Eagle CAD design for early PCB order	
10/21/19	Choosing and setting up database depending on requirements and constraints of different DB engines	Continue testing hardware components, finalize CAD files for 3D printing, submit button requirements to Machine Shop	Continue testing hardware components and order parts
10/28/19	IPR	3D print physical design and modifications; IPR	Test PCB with components; IPR
11/04/19	Link GUI to database and start creating functionality of software peripherals	Complete third version of Eagle CAD design	
11/11/19	Prepare for mock demo		
11/18/19	Write USB serial code for connection between USB port and microcontroller	Solder components to PCB and test	
11/25/19 (Break)			
12/02/19	Final stages of software and hardware integration testing		
12/09/19	Prepare for final presentation and complete final report		

## 5 Conclusion

### 5.1 Accomplishments

The PC Buttonpad ended up having many successful features that have been shown to fellow classmates, TAs, and professors. Major accomplishments include the establishment of cross-platform software, capacitive sensing buttons, data communication through an Arduino, and the customization of functions performed. The software GUI was able to run on Windows, macOS, and Linux operating systems as outlined in the high-level requirements. Additionally, the capacitive sensing circuit built from scratch along with second-order low-pass filters designed for optimal voltage drop detection was able to perform with no touch detection disruptions. As a main component of the PC Buttonpad, the capacitive touch sensors were crucial for the success of the PC Buttonpad. Part of this success comes from the use of an Arduino as a microcontroller that connected through USB to the PC in use. The Arduino could properly read the data sent by the hardware components and forward their signals to the PC to perform accordingly. The Arduino also supplied 5 V power through the USB to the rest of the hardware system. The achievement that gives the PC Buttonpad its purpose is the customization of button functions. Several actions of the user's choice could be assigned to any button and performed immediately by the touch of a capacitive touch button.

### 5.2 Uncertainties

Some of the less successful components of this project included the ultrasonic sensor functionality and the LED logic in the PCB design. The ultrasonic sensors ultimately did not work in the final product due to timing alterations with the rest of the codes. The ultrasonic sensors worked properly by detecting swipes in the ultrasonic sensor Arduino code, but could not achieve the same performance when connected to the rest of the project. Ultrasonic sensors are very time-specific to the millisecond and we believe that, with other codes running, the sensors could have timed out due to the amount of lag created with the rest of the software. This was the main integration failure of our project. The LED array worked with the Arduino but was originally supposed to operate solely through the PCB. On the second and final PCB designs we created, we included the LED array and logic. When this was tested, two of the smaller logic gates smoked – too much current was being passed through the gates. Some of the uncertainties in this project could have been easily fixed with additional time: we could have found a library that could set PC functionalities to the rotary encoder and used larger gates for the LED logic that could withstand larger current flow.

### 5.3 Ethical Considerations

Abiding to IEEE ethics, we are responsible for implementing and maintaining professional and ethical standards by which we follow throughout our design process and afterwards. As a team, we have ensured that all data and communications are truthful, following #3 of the IEEE Code of Ethics [11]. This includes communications between each other, professors, TAs, classmates, and future users. It is with utmost importance that we are honest and straightforward with all affected parties, especially regarding safety as

that is our highest priority in this process. Clarity and guidance will result if a party misunderstands the makeup of our design, technical and physical, the purpose of our product or its safety concerns.

To create a safe working environment for our team and future users, we were aware of our individual technical skillset and did not attempt technical tasks that we did not feel qualified to complete. Assistance was required in these cases to avoid any unsafe practices that would violate #6 of the IEEE Code of Ethics [11].

Our team values our compliance with the IEEE Code of Ethics through honesty, safety, and our technical work output. We have upheld #7 of the IEEE Code of Ethics to stay true to each other and our individual accountability and contributions within the project [11]. We are each responsible for our individual work and would have faced proper consequences for discrediting another's ideas or causing extreme setback for our team's progression.

We are also in agreement with the ACM Code of Ethics and Professional Conduct. While this code similarly reflects that of the IEEE Code of Ethics, it points out leadership responsibilities that we as a team must display. Our top priority is to ensure the safety and well-being of all users as listed in #3.1 of the ACM Code of Ethics and Professional Conduct [12]. As leaders, we take complete ownership of our product and strive toward creating a safe and educational environment.

Our continuous display of commitment to the ethical and professional standards outlined will remain after our project design and implementation.

Our project contains safety hazards that are important to the awareness of all users. The user interface includes several sensors which should be kept dry at all times. Moisture and other types of liquid have the potential to leak into the interior of the PC Buttonpad and could cause damage to the circuitry inside. Corrosion, mechanical failure, electrical shortages, or the creation of fire may occur in this instance. If any liquid leak is suspected, immediately shut off the PC and unplug the device from the PC. To prevent this from happening, we created a physical model that will seal any gaps between the interior circuitry and exterior of the device.

The metallic buttons are connected to capacitive touch sensors which can malfunction if pressed with too much pressure or if excess layers of dirt, grease, etc. are accumulated on the surface. Excessive forces placed upon these buttons can result in damages, disconnections, and loss in overall functionality of this product. To prevent the occurrence of these instances, the buttons are placed on a ledge within the 3D model for additional support. However, this does not completely avoid the damaging of our product through forcible touch – disconnections are still possible. The capacitive touch sensors rely on the detection of capacitance change measured across the metallic buttons. If these buttons accumulated thin layers of dirt on their surfaces, the capacitive difference would decrease and eventually make it hard to tell the difference between a touch and no touch. Standard hygiene measures are recommended for the user, such as washing hands, to maximize the button efficiency. In the case that buttons do get damaged, do not disassemble the device, as one incorrect wire placement could lead to overheating elements.

Overheating could occur if 5 V of power was not provided to the design components. This could also happen if more than 5 V of power is provided through the USB port to the PCB. Each module of our device is designed around a 5 V power input from the USB. Since the device relies on PC power, there is no worry that the voltage through the USB port will exceed 5 V. It is important for the user to plug the device into the correct USB port at 5 V when connecting it to the PC monitor.

The accompanying software will be made open-source, and anyone will have access to the code. This provides a guarantee to users or potential users that there is no malicious code or data collection. It is also likely that users with a programming background can bring awareness to any unintentional security issues or bad programming practices, making the software safer and more stable for everyone else.

Our team successfully completed the Lab Safety Training required for access to any lab. Following campus policy #RB-13, Campus Environmental Health and Safety, we have been responsible for maintaining a healthy and safe environment for ourselves and the rest of the University of Illinois at Urbana-Champaign community [13].

## 5.4 Future Work

As with any short-term project, there are many things to improve upon and features that could be added. The software can be made more user-friendly by adding tutorials and examples on how to use macros and actions.

There are few main improvements that can be made to the hardware. Moving away from the Arduino prototype and using only an ATmega328 microcontroller instead would dramatically decrease the cost and reduce the physical dimensions of the circuitry. Using an integrated circuit for the capacitive sensors could also reduce the area of the PCB and allow us to reduce the dimensions of the PC Buttonpad to increase portability. Integrating the ultrasonic sensors with the other sensors and adding additional functionality to the swipe gestures is another improvement that can be made.

## References

- [1] “Electron,” *Electron*. [Online]. Available: <https://electronjs.org/>. [Accessed Dec. 10, 2019]
- [2] “The Progressive JavaScript Framework,” *Vue.js*. [Online]. Available: <https://vuejs.org/>. [Accessed Dec. 10, 2019]
- [3] “Welcome to Python.org,” *Python*. [Online]. Available: <https://www.python.org/>. [Accessed Dec. 10, 2019]
- [4] “Node SerialPort,” *Node SerialPort*. [Online]. Available: <https://serialport.io/>. [Accessed Dec. 10, 2019]
- [5] “pySerial Documentation,” *pySerial*. [Online]. Available: <https://pythonhosted.org/pyserial/>. [Accessed Dec. 10, 2019]
- [6] “GitHub - kbm-robot,” *Kyle Paulsen*. [Online]. Available: <https://github.com/kylepaulsen/kbm-robot>. [Accessed Dec. 10, 2019]
- [7] “GitHub - boppreh,” *boppreh*. [Online]. Available: <https://github.com/boppreh/keyboard>. [Accessed Dec. 10, 2019]
- [8] “FT232R USB UART IC,” *Future Technology Devices*. [Online]. Available: [https://cdn-shop.adafruit.com/datasheets/DS\\_FT232R.pdf](https://cdn-shop.adafruit.com/datasheets/DS_FT232R.pdf). [Accessed Dec. 10, 2019]
- [9] “Capacitive sensing,” *Wikipedia*, Aug. 16, 2019. [Online]. Available: [https://en.wikipedia.org/wiki/Capacitive\\_sensing](https://en.wikipedia.org/wiki/Capacitive_sensing). [Accessed Sep. 16, 2019]
- [10] “Use Analog Techniques to Measure Capacitance in Capacitive Sensors,” *ElectronicDesign*. [Online]. Available: <https://www.electronicdesign.com/analog/use-analog-techniques-measure-capacitance-capacitive-sensors>. [Accessed Oct. 2, 2019]
- [11] “7.8 IEEE Code of Ethics,” *IEEE*. [Online]. Available: <https://www.ieee.org/about/corporate/governance/p7-8.html>. [Accessed Sep. 15, 2019]
- [12] “ACM Code of Ethics and Professional Conduct,” *Code of Ethics*. [Online]. Available: <https://www.acm.org/code-of-ethics>. [Accessed Sep. 15, 2019]
- [13] “Campus Environmental Health and Safety,” *Campus Administrative Manual*. [Online]. Available: <https://cam.illinois.edu/policies/rp-13/>. [Accessed Sep. 19, 2019]
- [14] “HC-SR04 Datasheet,” *Maker Pro*. [Online]. Available: <https://maker.pro/custom/tutorial/hc-sr04-ultrasonic-proximity-sensor-datasheet-highlights>. [Accessed December 8, 2019]

## Appendix A Requirement and Verification Tables

Table 5. Driver serial port interface requirements and verifications.

Requirements	Verifications	Verified?
1. The driver should be able to receive 8-bit data values at a rate of 9600 baud.	1A. Set up an Arduino to send a sequence of bytes at a pseudo-random interval (to simulate the human press of a button) and ensure that the driver receives those same sequence of bytes.	Y
2. Receiving a byte should trigger a function depending on what byte is received.	2A. The same Arduino can be used to send the sequence of bytes and some dummy functions can be tested on the driver side to ensure that they are executed.	Y

Table 6. Software peripheral simulator requirements and verifications.

Requirements	Verifications	Verified?
1. This module should be able to simulate single keypresses and keyboard shortcut combinations.	1A. Attempt to simulate the press of individual letters and ensure they are output. 1B. Attempt to simulate commonly used keyboard shortcuts and ensure that their effect is visible (e.g. close window, open a new tab, etc.).	Y Y
2. Simulating keypresses should work on the stated operating systems: Windows 10, macOS, and mainstream Linux distributions.	2A. Run the above tests on all three of the operating systems to ensure smooth operation.	Y

Table 7. Database of user-configured button functions requirements and verifications.

Requirements	Verifications	Verified?
1. The database can hold key combinations and sequences such that it can be read and executed by the software peripheral simulator module.	1A. Store dummy key sequences in the database and attempt to use the software peripheral simulator to read and execute those sequences.	Y
2. Multiple key sequences must be set up to execute sequentially if required, but within reason (< 100 keys).	2A. Test a dummy key sequence of around 100 keys to ensure that this can execute without any glitches or crashes.	Y

Table 8. GUI requirements and verifications.

Requirements	Verifications	Verified?
1. The GUI should allow the user to choose any key combination and sequence of keys, within the reasonable 100-key limit.	1A. Ensure that the GUI allows for easy editing and setting up of the key sequences.	Y
2. A selected key sequence must also be written to the database for further use by the software peripheral simulator.	2A. Set a key sequence using the GUI and then check that the software peripheral simulator can correctly execute that key sequence from the database.	Y



Table 9. Control unit requirements and verifications.

Requirements	Verifications	Verified?
1. The Arduino should be able to send a sequence of bytes through the USB serial output.	1A. Use the driver serial interface to intercept bytes sent by the Arduino and check for validity.	Y
2. It should be able to receive valid inputs from the ultrasonic sensor and the capacitive sensor.	2A. Check that the Arduino receives the distance measurement from the ultrasonic sensors encoded as a number.	Y
	2B. Ensure that the Arduino receives different numbers for different buttons pressed on the PC Buttonpad.	Y

Table 10. Capacitive touch sensor requirements and verifications.

Requirements	Verifications	Verified?
1. Must detect touching situations only, i.e. would not trigger unless the finger is $\leq 0.5$ mm away from the button.	1A. Pretest this using Arduino, coins, foils, etc. on a breadboard, and place finger as close to the metal as possible without touching. Observe how the reading changes as we do so. If not desirable, change resistor value.	Y

Table 11. Ultrasonic sensor requirements and verifications.

Requirements	Verifications	Verified?
1. Distance range for motion detection must be between 2 cm and 1 m.	1A. Test accuracy of data around lower distance threshold at 2 cm and ensure that accuracy is maintained up to 1 m. Data collected between 2 cm to 12 cm will be most important for the uses of this design.	Y
2. Angle of measurement is within 15°.	2A. Test hand placement above sensor at several angles, including 15°, and compare the data. If 15° is too tight of an angle for motion detection, the physical design will be modified to slightly angle the ultrasonic sensor more towards the user.	Y
3. Two adjacent ultrasonic sensors must individually collect data to perform the profile swipe function.	3A. Take and record data from a single ultrasonic sensor. Test two ultrasonic sensors, side by side, to see if their signals interfere or cause any discrepancy in the data compared to the data of a single ultrasonic sensor.	Y
4. Measurement cycle period should exceed 60 ms.	4A. We will set the cycling period to 70 ms in the Arduino code before sending the following 10 $\mu$ s TTL pulse.	Y

Table 12. Rotary encoder requirements and verifications.

Requirements	Verifications	Verified?
1. Must register each turn as either +1 or -1 in the signal print out.	1A. Print on screen using Arduino.	Y

Table 13. LED array requirements and verifications.

Requirements	Verifications	Verified?
1. Illumination of at least 10 mcd at the designed angle of 50° for our LEDs chosen, which translates to about half of the 19 mcd specification.	1A. Make sure that the current through the LEDs is at least more than half of the current limit, i.e. $\geq 10$ mA for each LED is expected such that the brightness will be somewhere more than 10 mcd. This can be measured using a multimeter when a 3.3 V power supply goes to PIN1 and PIN2 from the schematics above, and all four LEDs are on at the same time, drawing the largest amount of current.	N
2. Does not burn out due to exceeding the tested power limit, in this case $\leq 2.6$ V, 20 mA is the limit we would aim to attain.	2A. Make sure that the current through the LEDs is strictly less than 20 mA, which again can be measured with a multimeter when 0 V goes to PIN1 and PIN2, and only one LED is on.	Y

Table 14. USB power supply requirements and verifications.

Requirements	Verifications	Verified?
1. The output voltage is no less than 4.7 V, and the capability of outputting current is no less than 500 mA.	1A. Use a multimeter to measure the output from the regulator when connected with/without load.	Y