

GNG2101 Deliverable J

Submitted by

C1

Madison Dorion, 300059625

Elodie Drew, 300027471

Kevin Ooi, 300064997

Aidan Pledge, 300067480

Dennis Sun, 300111813

April 2, 2020

University of Ottawa

Abstract

This document covers the construction, features, user instructions and troubleshooting guidelines of our one handed game controller, designed and built for our client as part of the GNG2101 class. Our client has always loved gaming and uses it as a way to connect with his family, however, he lost mobility in his right side due to several strokes and since then has not been able to pursue his passion. Our controller has unique features allowing our client to return to gaming with his family, while only using his left hand. Its design is based on a miniature keyboard with an incorporated joystick mouse. The construction of our prototype is split into three main components. The first one is the circuit, followed by the coding and case of the remote. This document will explain in depth how each of these components were designed and built, including helpful tips for anyone trying to reproduce our work. Some safety precautions need to be taken while building this prototype, however, it's proper use should not pose any risk. Step by step instructions for the setup and use of the controller are provided for anyone who would simply like to use the product. Troubleshooting guidelines are outlined for both the construction of the prototype as well as its use.

Table of contents

Abstract	2
List of Figures	4
Introduction	5
Main Body	5
Features	5
Circuit	5
Code	7
Case	10
Health and safety guidelines	12
Setup	12
Troubleshooting	13
Conclusion	15
Bibliography	16

List of Figures

Fig 1. Keyboard Matrix Circuit with Pin Assignments

Fig 2. Keyboard map, set up using the keypad library

Fig 3. Keypad library and Keyboard library used to press or release active keys

Fig 4. readAxis function in the mouse code

Fig 5. 3D model of the controller

Fig 6. Printed keyboard grill and main cover

Introduction

Our client is a 44 year old father who loves gaming. It is a place where he can spend quality time with his daughter and his friends. It is a place where he can escape from reality and create his own universe and destiny. Unfortunately that was cut from him when he got two strokes that left him unable to use his right arm and leg. Being that his main console is the playstation 4 and that the main remote for that is a two handed remote, he is unable to play anymore. Our client has been trying different types of remotes so that he can play again with his daughter and friends and have his virtual world back, for several years now but has not had any success. The remotes he has tried either did not work, he did not like or were not adaptable enough for him to be able to use them. We must design and build a fully functional one handed game controller that is compatible with the playstation 4.

Having a remote that is one handed and also left handed creates the possibility for people who have lost mobility. It also creates an opportunity for gamers who would like to try something new. Most one handed remotes are for people who are right handed so this also reaches out to the smaller percentage of games who are left handed.

Most of the remotes in the market right now are hand held and for right handed people. There is the Razer Tartarus gaming pad that is similar to ours but has some differences that make ours more beneficial. It is ergonomically designed for comfort so that the user can play for prolonged periods of time. It is fully customizable so that the user can choose which buttons do what action and have everything set up for his comfort. The remote uses the XIM APEX adapter to connect to the playstation. Although our client is using the playstation, having the XIM APEX as the adapter makes it so that it is compatible with a wide variety of consoles.

Main Body

Features

- Features 22 customizable keys, each producing different sets of functions in regards to how the character moves and interacts when playing video games. The customizable keys are designed for the client to decide the location of the controls based on his playing style and where he is most comfortable.
- Features 1 joystick and is placed where the thumb is set, and the function is for the joystick to control camera movement for the character.
- The controller is compatible with the XIM APEX, meaning the keyboard controller can be played on any type of device such as the PS4 and PC.
- The hand support is designed for the client to rest their hand on top of the support, and can be played upon for hours at a time without any discomfort.

Circuit

The main circuitry of the controller can be separated into 3 components; the mouse joystick controlled by the thumb, the keypad controller by the other four fingers including

another joystick paddle controlled by the thumb and the arduino connections. Since our controller mimics the use of a keyboard and mouse setup, we use one arduino for the keyboard functions and another arduino for just the mouse function. This also allows for more space for connections since using two arduinos separately allows for more space on each arduino to use necessary pins, and decreases the likelihood of running out of pin space on each arduino.

The thumb-controlled portion of the controller is a simple concept and execution; one joystick is used to control the mouse and a second joystick placed below controls a paddle that acts as a button. The mouse joystick is a simple joystick that is seen on most standard controllers - it has movement in all directions in an x-y coordinate system and a controllable sensitivity, as well as a button that can be activated by pressing down on the joystick. This is accomplished simply with 5 wires that recognize the X coordinate, Y coordinate and push button, as well as provide a ground and vcc. While our controller does not have a function that allows for a change in sensitivity built-in, it functions with the XIM APEX adapter, allowing for a real-time sensitivity adjustment from the client's phone while playing a game. While this joystick is one of two joysticks used, it is the only part of the controller that gives function to the mouse, and therefore is the only component connected to the mouse arduino.

A second joystick is used with only 1 dimension in a coordinate system, giving function when moved in a single direction only. Since this joystick functions as a paddle-shaped button, it does not have pressure sensitivity like the first joystick and only recognizes whether or not the paddle is receiving an input. The paddle is placed below the mouse joystick and is meant to be pressed down with the thumb, giving it a similar feel to the spacebar on a keyboard. Aside from the absence of a single dimension and a push button when pushing the joystick in, the circuitry for this joystick contains all the same components as the mouse joystick and is connected to the keyboard arduino as it functions as a keyboard button being pressed.

The keypad portion of the controller contains 20 buttons organised in 4 rows and 5 columns. Since an arduino micro as we used does not contain enough pins to control every button alongside the paddle controlling the spacebar, a more efficient circuit must be used to prevent the technical and financial complications included in using a third arduino. Directly wiring the buttons to the arduino would take up one pin each, so in order to make this more efficient we used a keyboard matrix fitted to match the 4x5 matrix our keyboard consists of. Using a wire that runs vertically at each column and another wire that runs horizontally along each row can provide each button with a combination of [Row]x[Column]. This means for instance, that button A is denoted by [1]x[1], B denoted by [1]x[2], and so on until button T in position [4]x[5]. Since every button can now be recognized by a combination of a row and a column, assigning each column as an input and each row as an output can effectively lower the number of pins needed from 20 to 9 since we only need to use 4 rows and 5 columns. However, this does provide another issue due to each row being connected to 5 buttons and each column being connected to 4 buttons. This compact style makes it possible for ghosting to occur, where pressing a combination of 2 or more buttons can result in buttons being recognized as pressed

when they are not. To give an example on how something like this could occur, consider that buttons A [1]x[1], B [1]x[2], F[2]x[1], and G [2]x[2] are adjacent to one another in a square shape. Since A and G are diagonal from one another, the pressing of these two buttons would require inputs 1 and 2, and outputs 1 and 2. The combination of these inputs and outputs would function as pressing A and G as expected, except since B and F are also a combination of these inputs and outputs, they would be pressed in the process regardless of whether or not they are actually being pressed. This is called ghosting as mentioned before, and is caused by the fact that an output row from another key can travel back up another button's output. If the input is also being pressed on this button, the arduino will recognize this as the button being pressed. This problem can be simply solved however, with the use of diodes pointing out from each button before connecting to the output wire. The diodes prevent the current from travelling back into the button and thus prevent ghosting with any combination of button. These inputs were then named 6-10 and outputs were named 2-5 to match the pins each row or column would be connected to on the arduino, resulting in a functioning 20 button keyboard in addition to a spacebar paddle on the keyboard arduino.

A default configuration and use of these keys is built-in due to the fundamental structure of the code, however the client can change the layout of these keys in real time easily for each game he plays using the XIM APEX adapter with which this controller can connect.

As seen below in figure 1, rows 2-5 are outputs denoted by a black line, and columns 6-10 are inputs denoted by a red line. Switches are placed in the location of each button and triangles display the location and orientation of the diodes used.

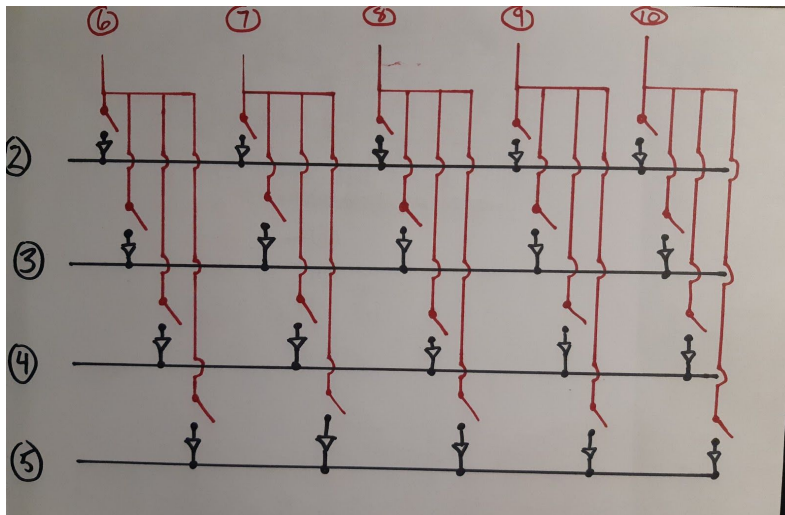


Fig 1. Keyboard Matrix Circuit with Pin Assignments

Code

One of the key parts of this project is the code. The first step of this process is selecting a microcontroller. The Arduino Pro Micro is a good choice as it is small enough to fit in the

controller case and can be recognized as a mouse or keyboard. Seeing as we are using two separate Arduino boards we also needed two codes. One code controls the keyboard whereas the other controls the joystick mouse. We chose this configuration for an easy conversion for the XIM APEX which is configured to allow the use of a keyboard and mouse setup with a console

The keyboard code (Appendix A) uses two libraries; the Keypad library (1) and Keyboard library (2). The keypad library allows for very simple coding of the keyboard matrix. It sets up the rows as outputs and columns as inputs in the background and checks every column and row automatically. It also allows for a simple and intuitive keyboard map and is capable of storing the state of up to ten keys simultaneously. The code then uses a loop that detects all “active” keys, in other words keys that have been pressed, are being held or were just released. The keyboard library then uses the `Keyboard.press(val)` or `Keyboard.release(val)` to either “presses” or “releases” the key value associated with it. Once a key has been “pressed” in the program, it remains in that state until it is released by the keyboard. Any value found on a typical keyboard can be assigned to a key, from alphanumeric values to modifiers. Alphanumeric values can be assigned directly whereas modifiers are generally assigned using the ASCII code, however a complete list of modifier codes can be found on the Arduino website (3).

```
const byte ROWS = 4; //four rows
const byte COLS = 5; //five columns

//set up the map of values to assign to every position in the keyboard matrix
char keys[ROWS][COLS] = {
  {'a', 'b', 'c', 'd', 'e'},
  {'f', 'g', 'h', 'i', 'j'},
  {'k', 'l', 'm', 'n', 'o'},
  {'p', 'q', 'r', 's', 't'},
};
byte rowPins[ROWS] = {5, 4, 3, 2}; //connect to the row wires of the keyboard
byte colPins[COLS] = {10, 9, 8, 7, 6}; //connect to the column wires of the keyboard

//initialize the keypad map
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

Fig 2. Keyboard map, set up using the keypad library


```

if (kpd.getKeys())
{
  for (int i=0; i<LIST_MAX; i++) // Scan the whole key list.
  {
    if ( kpd.key[i].stateChanged ) // Only find keys that have changed state. Also prevents unnecessary loops
    {
      switch (kpd.key[i].kstate) { // Report active key state : IDLE, PRESSED, HOLD, or RELEASED
        case PRESSED:
          Keyboard.press(kpd.key[i].kchar); //press the key corresponding to the i position in the key map
          break;
        case RELEASED:
          Keyboard.release(kpd.key[i].kchar); //release the key corresponding to the i position in the key map
      }
    }
  }
}

```

Fig 3. Keypad library and Keyboard library used to press or release active keys

The mouse code (Appendix A) uses the mouse library (4) to translate joystick values into mouse movement. A joystick is usually composed of two potentiometers and a momentary push button. One potentiometer controls the x axis whereas the other controls the y axis. The momentary push button can be configured as any mouse click. In this code, it is configured as a left click `Mouse.press(MOUSE_LEFT)`. Seeing as the analog values returned by a potentiometer range from 0 to 1023, they need to be converted to a range appropriate for mouse movement, with a center value of zero. In this case the code uses a range of twelve, from negative six to positive six. The conversion is done in a separate function in the loop called `readAxis(int)` and the value returned is used in the `Mouse.move(xVal, yVal, wheel)` function.

```

int readAxis(int thisAxis) {
// read the analog input:
int reading = analogRead(thisAxis);

// map the reading from the analog input range to the output range:
reading = map(reading, 0, 1023, 0, range);

// if the output reading is outside from the rest position threshold, use it:
int distance = reading - center;

if (abs(distance) < threshold) {
  distance = 0;
}

// return the distance for this axis:
return distance;
}

```

Fig 4. readAxis function in the mouse code

Both of these codes need to be used with caution due to the use of the keyboard and mouse libraries. While using the keyboard library, if a key is pressed but is never released in the code, the value associated with it will appear on the screen in an uncontrollable fashion as the

keyboard library can override your own keyboard. While using the mouse library, if the coordinates returned by the `readAxis` function are incorrect, you can lose complete control over your own mouse. Both of these situations can lead to difficulties in updating your code at the very least. Through considerable trial and error, we have established a few tricks to help:

- Always have access to a reset button, either on the board or wired to the board.
- Always check your code for logic, even if it compiles properly and it is very simple.

While using the keyboard library:

- Always place your cursor behind comment bars before testing a new circuit, it is less intrusive to your code.
- Triple check your code for a release function after every press function, no matter how simple the code.

Case

All of the pieces for the remote were designed on Solidwork. A lot of the pieces had to be redesigned multiple times for them to all fit together and be comfortable for the user to use for prolonged periods of time. Once everything was designed, the files were transferred to Ultimaker Cura so that they could then be 3D printed. The main cover, the key grill and the base were all printed with a 4.0mm nozzle on the Ultimaker 2+. The key caps were also printed on the Ultimaker 2+ but with a 0.25mm nozzle, as they are smaller components and need to be done very precisely for them to fit on the buttons.

Every time there was a component printed we would test it to see if there were some adjustments to be made. The key grill was adjusted multiple times. The first one was too big so we made it smaller and changed the radius of the curve. The second one was too small but we liked the new radius and the way it was shaped so we just made it a bit bigger. Once we got the general shape down, we decided to go with a thickness of about 2mm for it. We then reprinted it and used that as our final piece for it. The file for the final keygrill is “keygrill2.stl”.

The key caps took the longest to print. We did the first print on a 0.6mm nozzle because it was fast but they turned out really bad so we then tried the 0.25mm nozzle and it worked much better. The pieces were very sharp so we added some fillets to them for them to have a curve and be less sharp. With the new fillet, the prints kept failing no matter what we did so we had to make the radius of the fillet a bit smaller and it worked perfectly after. The thickness of the walls were also too thin at first which kept making the prints fail but once the thickness was adjusted the key caps printed with no issues. Because of the time limit with prints in makerspace, we decided to print five key caps at a time and repeated that four times to get our 20 caps. The final files for the key caps are “keycap3_(2).stl” and “Keycap3large.stl”. There are 16 of the keycap file that need to be printed and 4 of the keycaplarge file that need to be printed. The large buttons are the ones that are placed on the far left row where the pinky finger of the user would be.

The biggest piece that needed to be printed was the base of the remote. The base is what would connect all the pieces together. This was the last piece that was designed as it was

connecting all the other components together and the dimensions of the keyboard and cover were needed to successfully design the base. We first tried to print it with a 0.6mm nozzle but the print failed. We were gonna reprint it with a 0.4mm nozzle with different speed settings but due to the Covid-19 pandemic, Makerspace closed and we did not have access to the 3D printers anymore to print it. This was the last component that needed to be printed. The file for the final base is “keyboard_base_3.stl”.

The cover/mouse part of the remote was the first piece that was designed and printed. It was reprinted multiple times. The first one was a dome. We didn't like the shape of it and wanted the shape to be more like a computer mouse. We then redesigned it to make it bigger and make it more like a dome and not a half sphere. We also added a stick looking thing that came out of the cover. After printing the new design we decided to go back to just the dome shape and take out the branch. With this design, the joystick will be connected directly to the dome. The final file for the cover is “cover3.stl”.

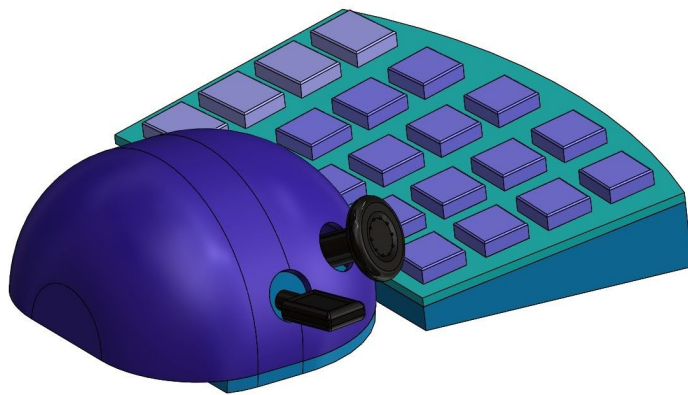


Fig 5. 3D model of the controller

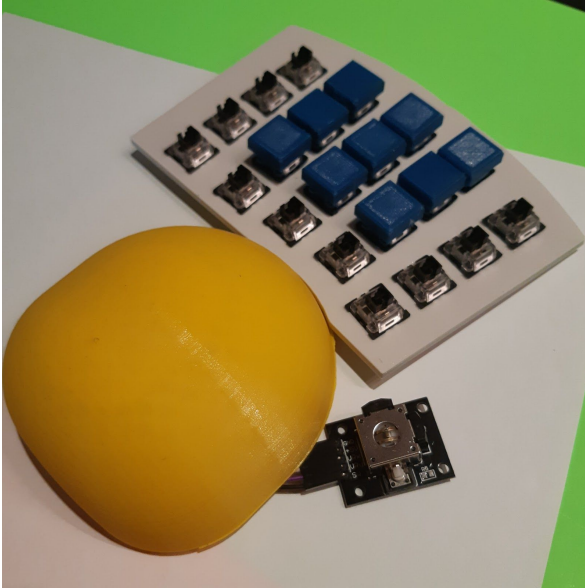


Fig 6. Printed keyboard grill and main cover

Health and safety guidelines

In terms of health and safety, there are a couple precautions that need to be taken. While soldering, a vent is needed to clear the toxic fumes from the solder. It is also recommended to wear safety goggles. The tip of the iron is usually around 600-700°F so it is important that you do not touch the iron.

There are also a couple precautions that need to be taken while 3D printing. The nozzle is extremely hot, as it needs to melt the PLA for it to print properly. It is important that you do not touch the nozzle or the base plate as it is also heated so that the print can stick to it.

Setup

In order to use our controller, the use of a XIM APEX or similar product is required and the following steps should be followed:

1. Connect the XIM APEX to your console
2. Connect a controller designed for the console to port 3 on the XIM APEX via USB (PS4 controller for PS4)
3. Connect the USB cord for the mouse (denoted by) to port 1 and the USB cord for the keypad (denoted by) to port 2 on the XIM APEX on the RPGenius controller.
4. Connect the XIM APEX to the XIM APEX manager app by opening the app and pressing the button on the XIM APEX while the manager is searching.
5. Choose the proper game that matches the one being played on the app to set up a configuration custom for that game.

6. Change the sensitivity in-game to maximum in order for best control over the mouse sensitivity.
7. Press the “Edit Config” button in the top left corner of the app to create a custom ideal configuration of keys as well as change the sensitivity.

Troubleshooting

During the construction of our controller we noticed several common problems that occurred. They are listed below with the solution and steps to fix it. The troubleshooting guide is split into two sections, one for building the controller and one for operating the controller.

Building:

1. Buttons not functioning
 - a. Diode orientation is incorrect - If the diode is placed in the wrong orientation the buttons will not work as the diode impedes current in one direction. To fix this, switch the direction the diode is facing.
 - b. Incorrect pin assignment - Check if the button is connected to the correct pin on your arduino. If the pin is incorrect either reconnect the switch to the correct pin or change the pin assignment in your code.
 - c. Switch is defective - If the two previous steps do not work it is possible the switch is defective. Connect the switch directly to the arduino and check if there is a current with a multimeter.
2. Joystick drifting
 - a. Center is improperly defined - Ensure the center is properly set and calibrated in your code.
 - b. Dead zones not set or not large enough - Increase the size of the dead zone. Most joysticks are not perfectly centered. The deadzone is a certain distance away from the center before the joystick registers movement.
3. Keystrokes Causing Frame Drops
 - a. Using `keyboard.print` instead of `keyboard.press` - Make sure your code is using `keyboard.press` and not `keyboard.print`. `Keyboard.print` prints a string instead of the ascii character.

4. 3D prints failing.
 - a. Decrease print speed - Takes longer but less material is extruded at once and
 - b. Decrease layer height - If the layer height is too large then there is an increases risk of plastic dripping
 - c. Enable z-hop - Prevents the nozzle from hitting other parts of your project.
 - d. Increase baseplate temperature
 - e. Change nozzle size

5. Controller is not detected as a mouse or keyboard
 - a. Ensure you are using an arduino pro micro or an equivalent board with HID compliance.

Operation

1. Failed to connect to PS4
 - a. Check controller is set to USB and not bluetooth - It need to be set to USB to communicate with the XIM APEX
 - b. Ensure the controller is off when you turn on the PS4
2. Joystick sensitivity too low
 - a. Make sure the joystick or mouse you connect is set to its maximum sensitivity. You can then change it to your preferred sensitivity through XIM APEX.

3. Button not detected
 - a. Restart the PS4 and controller - The XIM APEX sometimes will fail to detect and set up a button on wakeup. Restarting your console will resolve this issue

If at any time an unexpected error occurs simply restart the controller and console. This will resolve most issues.

Lessons Learned and Future Work

Over the course of the semester, we have learned many valuable lessons that have helped us improve our product and work together as a team. With the unexpected occurrences that have happened over the semester, it was important that our team have various contingency plans to adapt to any situation. It was also important that we not only meet every deadline on time, but to complete the task in a sufficient time frame. Due to unforeseen circumstances, we have failed to complete our prototype, and part of it was because we did not give ourselves enough time to complete each task. In the future, we should be more prepared for any type of scenario to occur, and plan ahead for each task to be completed weeks ahead. Another lesson to be learned is that

we need to ensure that the problem is well defined, and we should be more prepared to find a solution to every problem. In the future, we should create a more detailed task list, highlighting the important deadlines and detailed problems so everyone stays on track.

For our future work, our group wants to continue to collaborate with our client and continue to create a fully functional prototype in the near future. We want to continue working on the general comfort of the prototype, ensuring that the ergonomics is fully maximized and our client can be ready to use the product for hours at a time without feeling any discomfort. We also wanted to focus on optimizing our client's positioning by first measuring the length of his hand and adjusting accordingly based on the size of his hand. Then, ensuring that his hands have space to maneuver on the keyboard, and that his fingers and thumb can reach every control with ease. Also, we want to exceed the budget and use our own money, improve on the material used, and create a well-designed and high-quality product for our client. Finally, we wanted to add aesthetics and lightning on the keyboard. This will add a little decoration and improve the quality of the product for our client.

Conclusion

In conclusion, the goal of this project was to design and create an adaptive game controller for a client that can only use his left hand. To obtain this goal, our group underwent a various amount of ideas, concepts, and possible solutions were generated, all from previous deliverables. Through this whole process and every deliverable, we were able to produce our final product in the best of our efforts. In the end, we were able to describe the key functions, circuitry, coding, health and safety, and the case behind our product. We then troubleshooted and tested our product to test its functionality. Finally, we evaluated our progress and decided on our next step.

Bibliography

1. Chris. "Chris--A/Keypad." *GitHub*, 26 Dec. 2017, github.com/Chris--A/Keypad.
2. Arduino-Libraries. *Arduino-Libraries/Keyboard*. 2 Mar. 2020, github.com/arduino-libraries/Keyboard.
3. "KeyboardModifiers." *Arduino*, www.arduino.cc/en/Reference/KeyboardModifiers.