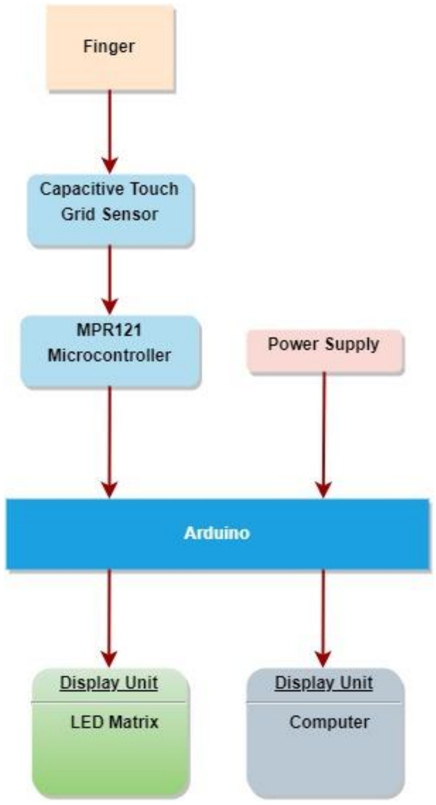
Akhil Bonela (abonela2) and Jizheng He (jizheng4)

Capacitive Touch Numpad Final Report

1. Introduction

We are attempting to make a number pad addon based on capacitive touch sensors for computers without one. Number pads are frequently used in scenarios from 3D Modelling, CAD, and other complex software to easy-access numerical input. Many computers are restricted by size and design methods, so they're not manufactured with this convenient tool. Developing a "keyless" number pad addon with capacitive touch sensors makes for a fantastic project, as well as an excellent learning experience.

2. Design



The general idea of the system is to take an input sensed by the touch sensor grid and show it on two display units. The MPR121 microcontroller must constantly scan the capacitance values for every row and column to sense a touch input. A touch input means that the capacitance values of the row and column that intersect at the input will increase. It must be noted that the output of the MPR121 microcontroler is inversely proportional to capacitance. If the capacitance values pass a certain threshold, the Arduino will determine which key was "pressed" from the location of the touch signal. The key's value will be displayed on the computer, and the corresponding LED underneath the key will light up.

Capacitive Touch Grid

A capacitor is a device that stores electrical energy within an electric field by charging and discharging. It's composed of two conductive surfaces (called electrodes) with a non-conductive layer (a dielectric) in between them. In a capacitive sensor, the sensor itself is the positive electrode of the capacitor, the dielectric is the air, and the grounded electrode is a finger. The current design is a self capacitence sensor. This means that each electrodeworks independent to the others and capacitance will increase as proximity increases due to the formula:

$$C = \frac{\varepsilon A}{d}$$

Where C is capacitance, ε is the dielectric constant, A is the electrode surface area, and d is the distance between the electrodes.

The number pad design is a 4x5 grid which requires 20 touch sensors. Rather than building 20 sensors with 20 individual connections, a capacitive touch grid with only 9 connections was implemented to save space and create a more efficient number pad (see Appendix B). A capacitive touch grid works like an individual sensor. When a point is touched, only the capacitance for both the row and column that intersect at that point increases.

In the current design, a clear ITO (Indium Tin Oxide) coated PET plastic is used as the positive electrode. Thin scratches were made in a grid pattern to scrape off some of the ITO and create 20 isolted sensors. Since the ITO was scratched away, there needed to be a conductive material like one-side conductive copper tape (inverted) to connect all the sensors. One end of each row and column is connected to the capacitive touch detector MPR121 microcontroler, which, in turn, is connected to the Aurduino.

LED Matrix:

The purpose of the LED matrix is to notify the user that the capacitive touch grid is sensing their input. In the design, the matrix is located underneath the capacitive touch grid. When a key is touched on the grid, the corresponding LED (the one right below it) will light up until the touch input ends. To accomplish this, a PCB was devised (see Appendix C) that utilizes th concept of multiplexing for increased efficiency and to reduce the number of connections required. A proof of concept was made on a breadboard at a smaller scale.

Arduino Numpad Program

We use MPR121, the 12-input-pin capacitive touch detector chip. The chip uses I2C to communicate with Arduino. Instead of messing around with I2C protocols, we can utilize the MPR121 library written by Adafruit [4]. Note that all wires connected to the input pins form a grid in the capacitive num pad, thus decreasing the capacitance. Therefore, we cannot directly use the auto-generated boolean value to determine whether a touch is in place; we need to use the raw data and define thresholds for each pin by ourselves.

We also need to emulate the keyboard using Serial outputs. A basic keystroke contains a buffer of 8 bytes. The third byte is our main concern that identifies which key is being pressed. The keycode table for a keyboard can be found in [6]. For each touch detected, we will send a buffer with the keycode in its third byte and send an empty buffer after ten milliseconds (emulate key releasing) into the Serial.

Arduino Uno is not recognized as a HID device. This means we need to reflash the bootloader with a hex file that identifies the Arduino as a keyboard HID device. First, we will upload our program and test it thoroughly (since we cannot load programs in keyboard mode). We will then wipe the original bootloader by shorting the leftmost two pins of the top-left six pins on the Arduino and update the driver so that our flashing software, Flip, can successfully recognize it. Finally, we will flash the Arduino with the hex file and reconnect Arduino to the computer. Detailed descriptions and hex files can be found in [3].

3. Results:

We successfully created a num pad addon for computers without one. We used capacitive touch detection instead of traditional pressing & releasing of keys. We successfully translated touch signals into emulated keystroke signals and made the system plugin-to-go. We also completed a LED Matrix PCB design and implemented a proof-of-concept prototype.

4. Problems and Challenges:

There are some manufacturing errors with the capacitive touch grid. Only 6 out of the 20 sensors detect touch. The cause of this problem is the tape binding the inverted copper tape loosening. Since the design relies on the copper tape to be pulled tight against the PET plastic for the sensors to work, loosened bindings pose a great threat to the integrity of the capacitive touch grid.

During the programming process, some pins also have extremely close capacitance values with and without touch. It also took dozens of trials and errors to find out the correct driver and steps to flash the Arduino Uno, including testing out multiple different hex files with the same filename in blogs, etc.

Due to time constraints, the LED matrix PCB could not be manufactured.

5. Future Plans:

As of right now it is uncertain whether this project will be continued, however there are a number of issues and improvements to focus on if it will be. The most important issue is developing a new and more durable capacitive touch grid where all the sensors work. Perhaps, as an improvement, a the grid could use mutual capacitance instead of self (can detect multiple touches). An fun improvement to make would be to create multiple keyboard layouts in addition to the number pad (like one specifically designed for gaming). This will alow the project to be used for a greater variety of purposes.

6. References:

[1] Electronoobie, "ESP8266: Controlling a LED matrix with the 74HC595 ICs - techtutorialsx," *techtutorialsx.com*. https://techtutorialsx.com/2016/09/17/esp8266-controlling-a-led-matrix-with-the-74hc595-ics/ (accessed Dec. 10, 2021).

[2] Texas Instruments, "SNx4HC595 8-Bit Shift Registers With 3-State Output Registers," *ti.com*. https://www.ti.com/lit/ds/symlink/sn74hc595.pdf (accessed Dec. 10, 2021)

[3] "Tutorial: How to Use Arduino Uno as HID | Part 1: Arduino Keyboard Emulation," *Tutorial*, Jul. 14, 2020.

https://techtotinker.blogspot.com/2020/07/tutorial-how-to-use-arduino-uno-as-hid.html?m=1 (accessed Dec. 10, 2021).

[4] “Adafruit MPR121 Library,” *GitHub*, Nov. 07, 2021. https://github.com/adafruit/Adafruit_MPR121/blob/master/examples/MPR121test/MPR121test.ino.

[5] “Capacitive Sensing for Dummies,” *Instructables*. https://www.instructables.com/capacitive-sensing-for-dummies/.

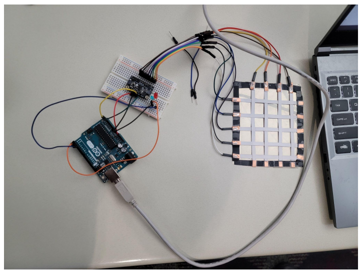
[6] “USB HID Usages,” *Freebsddiary.org*, 2021. https://www.freebsddiary.org/APC/usb_hid_usages (accessed Dec. 11, 2021).

[7] Bare Conductive, “Our hardware uses Capacitive Sensing. What is it?,” *bareconductive.com* 2021. https://www.bareconductive.com/blogs/blog/the-touch-board-uses-capacitive-sensing-what-is-it (Accessed 10 December 2021).

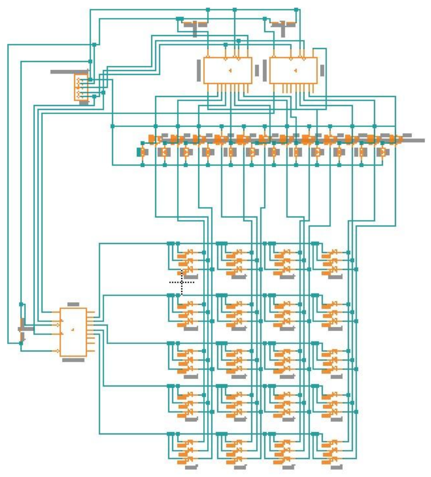
Appendix A. Arduino Code

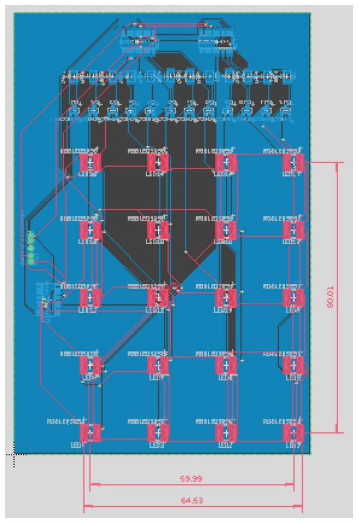
```
////////// INCLUDES //////////

#include <Wire.h>
#include <Adafruit_MPR121.h>

////////// CAPACITIVE TOUCH SENSOR //////////

#ifndef _BV
#define _BV(b) (1 << (b))
#endif

Adafruit_MPR121 cap = Adafruit_MPR121();
uint16_t prev_touch = 0;
uint16_t curr_touch = 0;

#define prev_touched(i) (prev_touch & _BV(i))
#define curr_touched(i) (curr_touch & _BV(i))

// Capacitance min threshold when not touching
const int kThreshold[9] = {10, 10, -1, -1, -1, 10, 10, -1, 50};

// Mapping from capacitive grid to key
const int kKeyMapping[5][4] = {
  {KEY_1, KEY_2, -1, KEY_3},
  {KEY_4, KEY_5, -1, KEY_6},
  {-1, -1, -1, -1},
  {-1, -1, -1, -1},
  {-1, -1, -1, -1}
};

////////// KEYCODES //////////

#define KEY_1 0x59
#define KEY_2 0x5a
#define KEY_3 0x5b
#define KEY_4 0x5c
#define KEY_5 0x5d
#define KEY_6 0x5e
#define KEY_7 0x5f
#define KEY_8 0x60
#define KEY_9 0x61
#define KEY_0 0x62
#define KEY_PERIOD 0x63
#define KEY_PLUS 0x57
#define KEY_MINUS 0x56
#define KEY_MUL 0x55
#define KEY_DIV 0x54
#define KEY_NUMLOCK 0x53
#define KEY_ENTER 0x58

void press_key(int key) {
  uint8_t buf[8] = {0};
  buf[2] = key;
  Serial.write(buf, 8);
  buf[2] = 0;
  Serial.write(buf, 8);
}

////////// PROGRAM //////////
// #define DEBUG

void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);

  Serial.begin(9600);
  while (!Serial) delay(10);

  digitalWrite(3, HIGH);
#ifdef DEBUG
  Serial.println("Adafruit MPR121 Capacitive Touch sensor test");
#endif
  if (!cap.begin(0x5A)) { // Default addr of I2C
#ifdef DEBUG
    Serial.println("MPR121 not found, check wiring?");
#endif
    while (1);
  }
  digitalWrite(3, LOW);
#ifdef DEBUG
  Serial.println("MPR121 found!");
#endif
  delay(1000);
}

void loop() {
  // Get the currently touched pad
  curr_touch = 0;

  for (uint8_t i=0; i<9; i++) {
    int data = cap.filteredData(i);
#ifdef DEBUG
    Serial.print("[");
    Serial.print(i);
    Serial.print("] ");
    Serial.print(data);
    Serial.print("\t");
#endif

    if (data <= kThreshold[i]) curr_touch |= _BV(i);
  }
#ifdef DEBUG
  Serial.println();
#endif

  int x = -1, y = -1;
  for (int i = 0; i < 5; i++) {
    if ((!prev_touched(i) && curr_touched(i)) || (prev_touched(i) &&
!curr_touched(i))) {
      x = i;
    }
  }
  for (int j = 5; j < 9; j++) {
    if ((!prev_touched(j) && curr_touched(j)) || (prev_touched(j) &&
!curr_touched(j))) {
      y = j;
    }
  }

  if (x != -1 && y != -1) {
    if (!prev_touched(x) && curr_touched(x)) { // new touch
      press_key(kKeyMapping[x][y-5]);
#ifdef DEBUG
      Serial.print(kKeyMapping[x][y-5]);
      Serial.print(" pressed at ");
#endif
      digitalWrite(2, HIGH);
    } else { // release
      digitalWrite(2, LOW);
#ifdef DEBUG
      Serial.print("Key released at ");
#endif
    }
#ifdef DEBUG
    Serial.print("(");
    Serial.print(x);
    Serial.print(", ");
    Serial.print(y);
    Serial.println(")");
#endif
  } else if (x != -1 || y != -1) {
    curr_touch = prev_touch;
  }

#ifdef DEBUG
  Serial.println("========================================");
#endif

  prev_touch = curr_touch;

  // put a delay so it isn't overwhelming
  delay(100);
}
```

Appendix B. Capacitive Touch Grid



Appendix C. LED Matrix




80.01
59.99
64.63