VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Swept Frequency Capacitive Sensing with Arduino

by

Christo Joby Antony (21BCE1635)

Suseel Mathusoothanan (21BCE1645)

Neelam Naga Saivenkata Suryavenu (21BCE1660)

A project report submitted to

Dr. E.Sathish

School of Electronics Engineering

for the fulfillment of Digital Assignment – 3 for the course of

BECE204P – Microprocessor and Microcontrollers

in

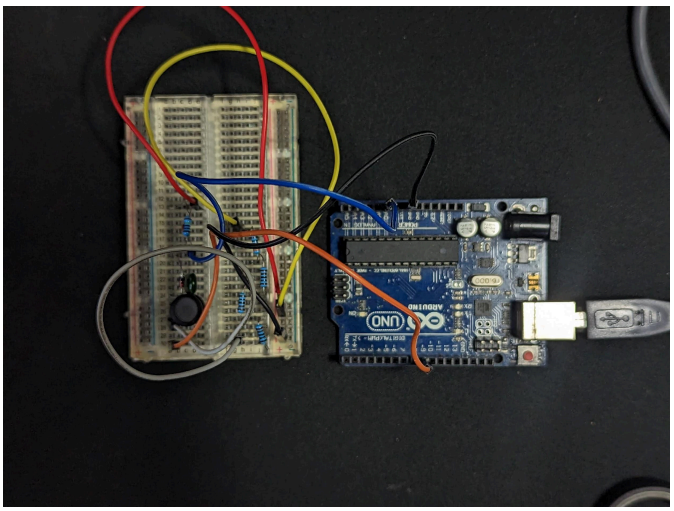
B.Tech Computer Science Engineering

Introduction

Our project introduces an innovative and accessible Swept Frequency Capacitive Sensing method that goes beyond traditional touch event detection. This cutting-edge technique not only detects touch events, but also has the ability to recognize intricate configurations of the human hands and body, providing contextual information that greatly enhances touch interactions across various applications.

From conventional touchscreens to unconventional contexts and materials, we have extended the capabilities of capacitive touch sensing. For instance, in our research, we have successfully added touch and gesture sensitivity to organic objects and even liquids, opening up new possibilities for touch-based interactions.

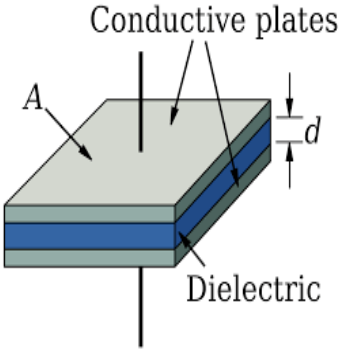
Our implementation aims to be cheaper, accessible while producing comparable results to advanced implementations that use dedicated electrical wave generating instruments. Therefore we aim to make this technology more affordable and easier to implement and scale.



Understanding Touch Capacitance

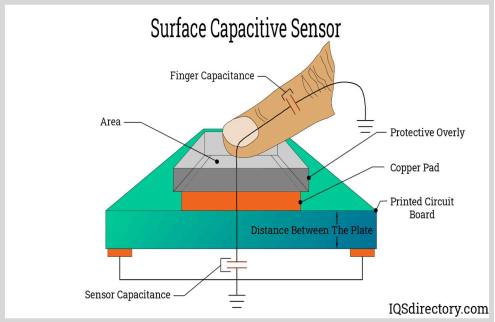
A quick recap in capacitance:

Capacitance is a fundamental concept in physics and electrical engineering that refers to the ability of a capacitor or a system of conductors to store electrical charge. It is a measure of the amount of electric charge that can be stored per unit voltage. They consist of two conductive plates separated by an insulating material, known as a dielectric. When a voltage is applied across the capacitor, charge accumulates on the plates, creating an electric field between them.



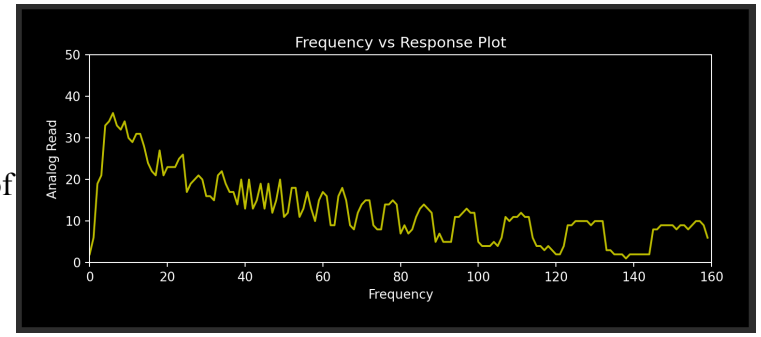
Capacitance in modern touch screens:

The touch screen panel is made up of a grid of conductive electrodes separated by an insulating layer . These electrodes are then excited by a fixed frequency signal. When a finger or a conductive object touches the screen, it creates a change in the electric field of the capacitor formed by the electrode and the object, leading to a measurable change in capacitance. This change in capacitance is then detected by the touch screen controller, which interprets it as a touch event and determines the position and characteristics of the touch, such as location, pressure, and movement.



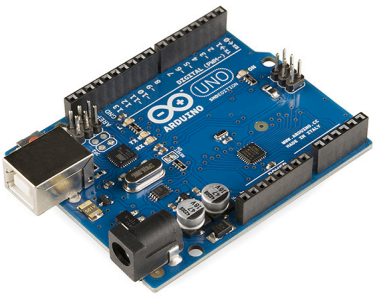
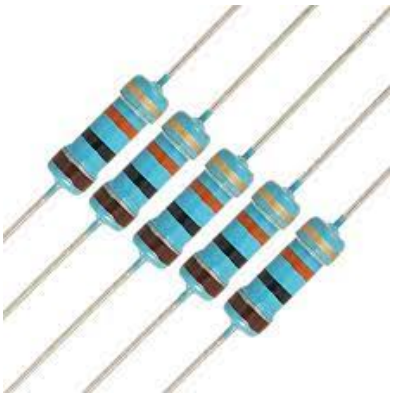
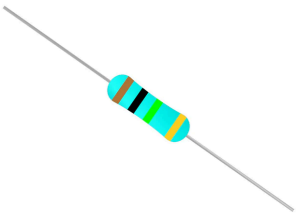
Swept Frequency Capacitive Sensing:

In SFCS we monitor the response to capacitive human touch over a range of frequencies. In contrast to fixed frequency excitation in traditional touch sensing. Objects excited by an electrical signal respond differently at different frequencies. Thus, instead of measuring a single data point for each touch event, we measure a multitude of data points at different frequencies. Then we can use ML techniques to extract rich interaction context. This technique is traditionally implemented using a wave generator, that wave is then amplified and applied to the object. Then the response wave is recorded using an ADC



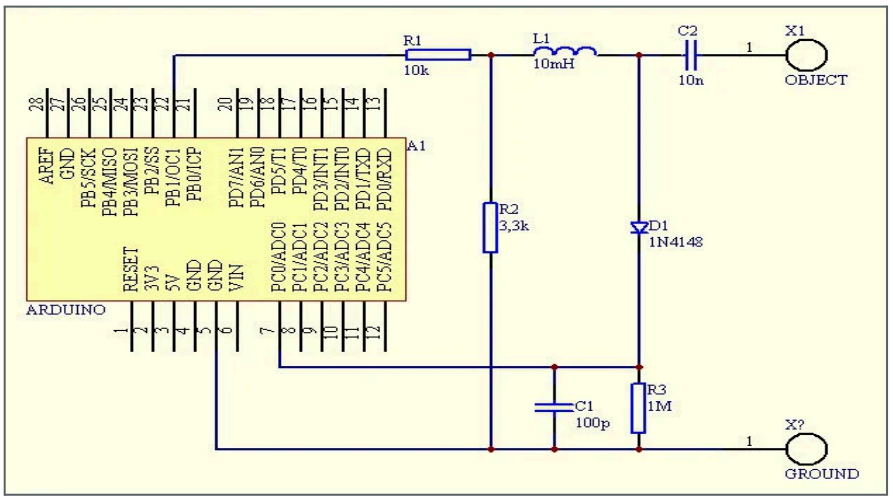
Hardware Implementation

Components utilized

Component	Picture	Quantity	Estimate Price
Arduino		1	₹600
USB Cable		1	₹150
Resistor -10k ohm		1	₹30
Resistor -1M ohm		1	₹25

Resistor -3.3k ohm		1	₹35
Capacitor: - 100pf		1	₹30
Capacitor - 10nf		1	₹60
Diode: 1N4148 diode		1	₹60
Coil / Inductor: 10mH		1	₹45

Circuit Diagram



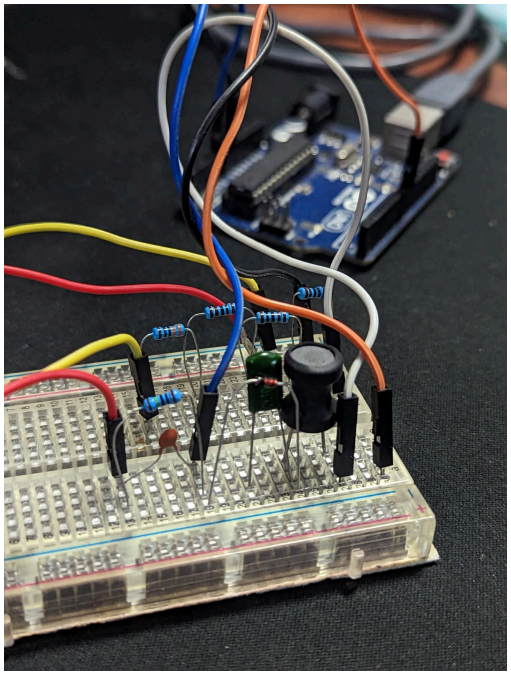
The given circuit diagram illustrates the Swept Frequency Capacitive Sensing with Arduino.
Pin 5, which is connected to the ground, is referred to as the Crown pin .
Pin 22, on the other hand, is connected to port 9 and used for generating precise PWM waves, which are internally connected to timers and produce precise square waves.

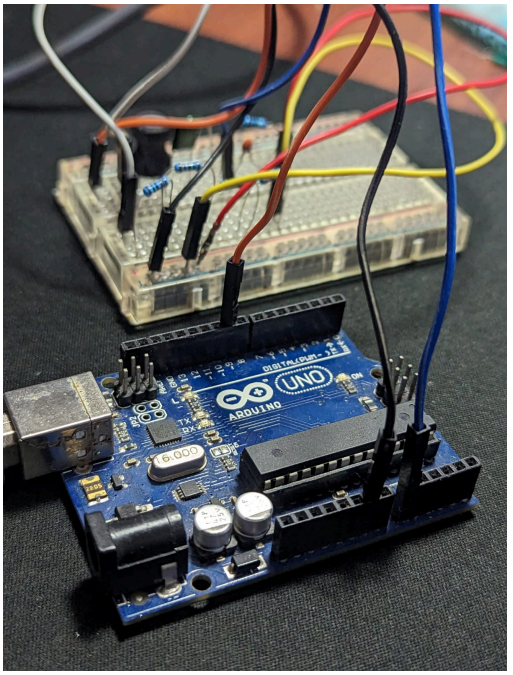
To achieve this, the circuit is connected to an RLC circuit that converts the mere square waves into usable sine waves. When an object comes in contact with the capacitor, it produces a change in voltage that generates energy, flowing in the opposite direction. A diode is used to allow the voltage change to flow in only one direction.

The change in voltage is caused by the resistance of the capacitor, and the resulting signal is sent to resistor R3, which has a very high resistance of 1 million ohms. This high resistance ensures that even the smallest change in voltage produced by the capacitor can be detected by the circuit. A filtering capacitor, connected in parallel with R3, helps filter out noise and disturbances in the signal, resulting in a final analog signal. This analog signal is then sent to analogue port 7 on the Arduino chip.

In summary, this circuit diagram demonstrates the Swept Frequency Capacitive Sensing with Arduino, which uses PWM waves, an RLC circuit, and a high-resistance resistor to detect changes in voltage caused by contact with a capacitor. The resulting analog signal is then filtered and sent to the analogue port on the Arduino chip.

The Hardware Setup





Arduino board - The Arduino board is a microcontroller board that serves as the brain of the system. It is programmed to control the Swept Frequency Capacitive Sensing shield (we have implemented this shield ina prototyping board) and collect and process data from the capacitive sensing electrodes.

Swept Frequency Capacitive Sensing Shield - This shield is an add-on board that can be connected to the Arduino board.It contains the necessary circuitry to generate a frequency sweep and measure the capacitance of the electrodes at each frequency. It also includes signal conditioning circuits and amplifiers to improve the quality of the data collected from the electrodes.

Capacitive Sensing Electrodes - These are conductive elements that are used to detect changes in capacitance. The electrodes can be any shape or size, but they must be conductive and isolated from the environment. When a finger or other object comes close to the electrode, it changes the capacitance, which is detected by the Swept Frequency Capacitive Sensing circuit. Connecting wires - These are used to connect the capacitive sensing electrodes to the Swept Frequency Capacitive Sensing shield and the Arduino board. The wires should be shielded to minimize noise and interference.

Power source - The Arduino board and the Swept Frequency Capacitive Sensing shield require power to operate. This can be provided by a USB cable connected to a computer. The power source should be capable of providing sufficient current to power the system.

Once the hardware is assembled and connected, the Arduino board runs a program that controls the frequency sweep and processes the signals received from the capacitive sensing electrodes. The program can be written in Arduino programming language and uploaded to the board using the Arduino Integrated Development Environment (IDE). The data collected from the capacitive sensing electrodes can be used for various applications, such as touch-sensitive interfaces, proximity sensors, and object detection.

Software Implementation

The complete code and implementation is hosted under the github repository [ChristoJobyAntony/ArduinoAdvnacedTouchSensor (github.com)]

We have included snippets of code relevant to the generation of frequency wave and the high speed transmission of the recorded values through serial

Arduino Code for Sensing Gestures

```
#define SET(x, y) (x |= (1 << y))    //-Bit set/clear macros
#define CLR(x, y) (x &= (~(1 << y))) // |
#define CHK(x, y) (x & (1 << y))     // |
#define TOG(x, y) (x ^= (1 << y))    //-+

#define N 160 // How many frequencies

float results[N]; //-Filtered result buffer
float freq[N];    //-Filtered result buffer
int sizeOfArray = N;

void setup()
{

    //  Timer/Counter1 Control Register A
    //  Bit7:6 10 -> Clear OC1A/OC1B on compare match, set OC1A/OC1B
    //  Bit5:4 00 -> OC1B is disconnected
    //  Bit1:0 10 -> Wave form Gerneation mdoe
    TCCR1A = 0b10000010; //-Set up frequency generator

    // Timer/Counter1 Control Register B
    // Bit2:0 Clock Select: (001) CLK 1 i/o no prescaling
    // Bit4:3 Waveform generation mode(11) (1110) : Fast PWM
    TCCR1B = 0b00011001; //-+
                         // The input capture is updated with the counter (TCNT1) value each time
an event occurs on the ICP1 pin (or optionally on the
                         // analog comparator output for Timer/Counter1). The input capture can be
used for defining the counter TOP value
    ICR1 = 110;
    // Output Compare register 1
    OCR1A = 55;

    pinMode(9, OUTPUT); //-Signal generator pin
    pinMode(8, OUTPUT); //-Sync (test) pin

    Serial.begin(115200);

    for (int i = 0; i < N; i++) //-Preset results
        results[i] = 0;         //-+
}

void loop()
{
    unsigned int d;
```

```
    int counter = 0;
    for (unsigned int d = 0; d < N; d++)
    {
        int v = analogRead(0); //-Read response signal
        CLR(TCCR1B, 0);        //-Stop generator
        TCNT1 = 0;             //-Reload new frequency
        ICR1 = d;              // |
        OCR1A = d / 2;         //-+
        SET(TCCR1B, 0);        //-Restart generator

        results[d] = results[d] * 0.5 + (float)(v)*0.5; // Filter results

        freq[d] = d;

        //    plot(v,0);              //-Display
        //    plot(results[d],1);
        delayMicroseconds(1);
    }

    PlottArray(1, freq, results);
    TOG(PORTB, 0); //-Toggle pin 8 after each sweep (good for scope)
}
```

In the setup() function, the Timer/Counter1 is configured for generating a Fast PWM signal, with a clock source of I/O clock with no prescaling (i.e., running at the maximum frequency of the microcontroller). The waveform generation mode is set to generate a signal that clears the output on compare match and sets it at the TOP value. The TOP value is set to 110 using the ICR1 register, which defines the counter TOP value for the PWM signal. The output compare register OCR1A is set to half the value of the TOP register, which defines the duty cycle of the PWM signal.

After the Timer/Counter1 is configured, the main loop() function starts. Inside the loop, a for loop sweeps through the frequency range defined by N. At each frequency point, the Timer/Counter1 is stopped, the frequency registers are updated with the new frequency values, and the Timer/Counter1 is restarted with the new frequency. The response signal from the analog sensor is then read and filtered using a simple moving average filter. The frequency and response values are stored in the freq and results arrays, respectively.

Arduino Code for Sending Serial Data

```
byte yMSB = 0, yLSB = 0, xMSB = 0, xLSB = 0, zeroByte = 128, Checksum = 0;

void SendData(int Command, unsigned int yValue, unsigned int xValue)
{

    /* >=================================================================<
        y = 01010100 11010100    (x & y are 2 Byte integers)
             yMSB      yLSB      send seperately -> reciever joins them
      >=================================================================<  */

    yLSB = lowByte(yValue);
    yMSB = highByte(yValue);
    xLSB = lowByte(xValue);
    xMSB = highByte(xValue);

    /* >=================================================================<
       Only the very first Byte may be a zero, this way allows the computer
       to know that if a Byte recieved is a zero it must be the start byte.
       If data bytes actually have a value of zero, They are given the value
       one and the bit in the zeroByte that represents that Byte is made
       high.
       >=================================================================< */

    zeroByte = 128; // 10000000

    if (yLSB == 0)
    {
        yLSB = 1;
        zeroByte = zeroByte + 1;
    } // Make bit 1 high
    if (yMSB == 0)
    {
        yMSB = 1;
        zeroByte = zeroByte + 2;
    } // make bit 2 high
    if (xLSB == 0)
    {
        xLSB = 1;
        zeroByte = zeroByte + 4;
    } // make bit 3 high
    if (xMSB == 0)
    {
        xMSB = 1;
        zeroByte = zeroByte + 8;
    } // make bit 4 high

    /* >=================================================================<
       Calculate the remainder of: sum of all the Bytes divided by 255
       >=================================================================< */

    Checksum = (Command + yMSB + yLSB + xMSB + xLSB + zeroByte) % 255;

    if (Checksum != 0)
    {
```

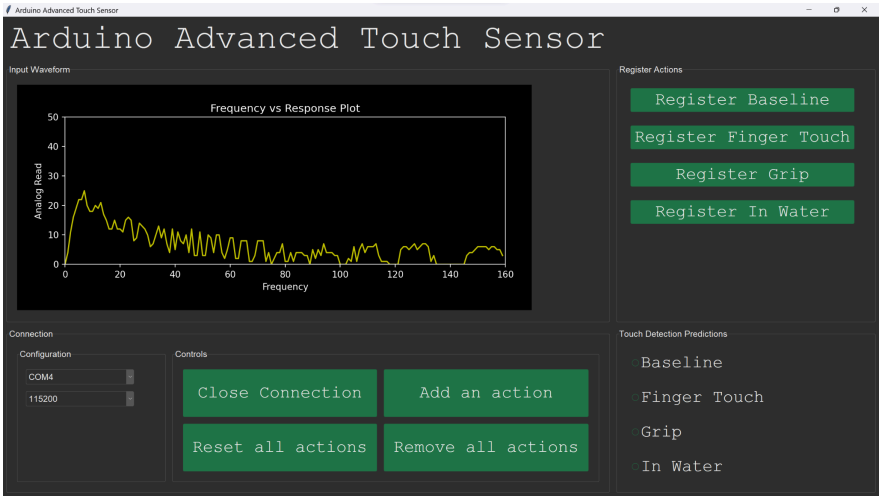
```
        Serial.write(byte(0));       // send start bit
        Serial.write(byte(Command)); // command eg: Which Graph is this data for

        Serial.write(byte(yMSB)); // Y value's most significant byte
        Serial.write(byte(yLSB)); // Y value's least significant byte
        Serial.write(byte(xMSB)); // X value's most significant byte
        Serial.write(byte(xLSB)); // X value's least significant byte

        Serial.write(byte(zeroByte)); // Which values have a zero value
        Serial.write(byte(Checksum)); // Error Checking Byte
    }
}

void PlottArray(unsigned int Cmd, float Array1[], float Array2[])
{
    //      CMD 0 sets the beginnign of an array
    SendData(Cmd + 1, 1, 1); // Tell PC an array is about to be sent
    delay(1);
    //      CMD 1 flags the data being sent
    for (int x = 0; x < sizeOfArray; x++)
    { // Send the arrays
        SendData(Cmd, round(Array1[x]), round(Array2[x]));
        // delay(1);
    }
    //      CMD 2 flags the end of the array
    SendData(Cmd + 2, 1, 1); // Confirm arrrays have been sent
}
```

Python Code for Reading Serial Data

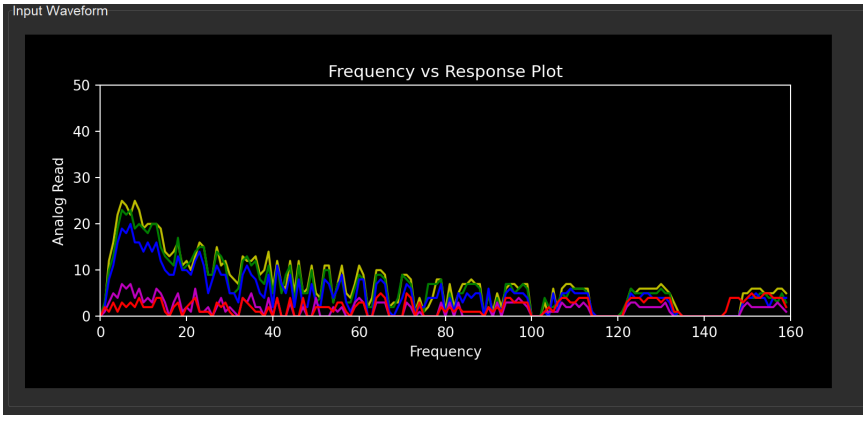
```
from typing import List, Tuple
import serial as s

ARRAY_SIZE  = 160

def get_from_serial (port: s.Serial) -> Tuple[List, List]:
    x_axis = []
    y_axis = []
    flag = 0

    n = 0
    while True:
        dat = port.read(1).hex()
        if dat == "00" :
            n += 1
            ar = []
            for j in range(7):
                ar.append(port.read(1).hex())
            e = int(ar[6], base=16)
            # check for command word
            cmd = int(ar[0], base=16)

            if (cmd == 2):
                # print("Start of array")
                x_axis = []
                y_axis = []
                flag = 1
                continue
            elif (cmd == 3):
                # print("End of array")
                if flag == 1:
                    return (x_axis, y_axis)

            # run checksum:
            # add all the bytes except the checksum
            check_sum = sum([int(d, base=16) for d in ar[0:6]])%255

            # if the checksums dont match, exit
            if (check_sum != e) :
                print("Error in Checksum ! ", check_sum)
                flag = 0
                continue

            z = int(ar[5], base=16)
            # parsing the zero byte and updating zeroes
            if (z & 1 != 0): ar[2] = "00"
            if (z & 2 != 0): ar[1] = "00"
            if (z & 4 != 0): ar[4] = "00"
            if (z & 8 != 0): ar[3] = "00"

            y = int(ar[1]+ar[2], base=16)
            x = int(ar[3]+ar[4], base=16)
            x_axis.append(x)
            y_axis.append(y)

            # print(f"y:{y} x:{x}")

if __name__ == "__main__":
    port = s.serial_for_url("COM4",115200)
    print(get_from_serial(port))
```

The Project Demonstration

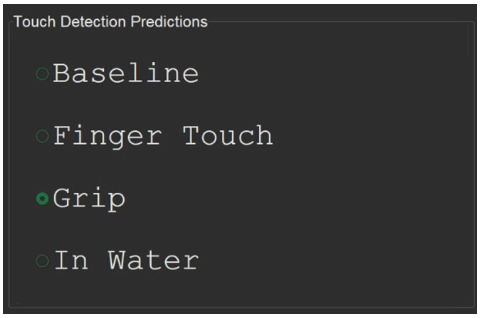
The Graphical User Interface

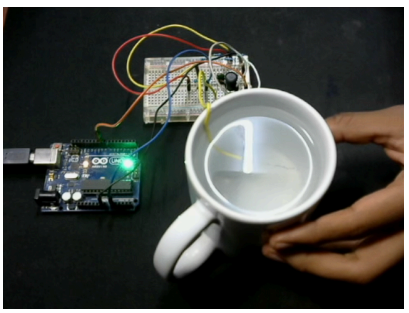


Setting up different actions



Predicting touch actions





Applications of Advanced Touch Sensing

Wearable Technology

Our Project could be used to create touch-sensitive interfaces for wearable technology such as smartwatches, fitness trackers, and other wearables. This could allow users to control the device with a simple touch, without the need for buttons or switches.

Industrial Applications

Our Project could be used in industrial settings to create touch-sensitive interfaces for machinery and equipment. This could improve safety by reducing the need for physical buttons and switches, and also increase efficiency by allowing for faster and more intuitive control of the equipment.

Gaming

Our Project could be used to create touch-sensitive game controllers for video games and mobile games. This could provide a more immersive gaming experience and allow for more intuitive control of the game.

Musical Instruments:

The Touche for Arduino project could be used to create advanced touch sensing capabilities in musical instruments such as keyboards, guitars, and drums. This would allow musicians to create more complex and nuanced sounds by varying their touch on the instrument.

Conclusion

The proposed Swept Frequency Capacitive Sensing technique is a significant breakthrough in touch interaction technology that can greatly enhance the user experience in a variety of applications. The ability to detect and recognize complex configurations of the human hands and body provides contextual information that can greatly expand the range of possibilities for touch interaction beyond traditional touchscreens. This novel technique has the potential to revolutionize the way we interact with technology and create new opportunities for innovation in various fields.

References

Touche for Arduino: Advanced Touch Sensing. : 5 Steps (with Pictures) - Instructables

Touché: Enhancing Touch Interaction on Humans, Screens, Liquids, and Everyday Objects

2 — Experimenting with Touché. Swept Frequency Capacitive Touch… | by Dhvanil Shah | Ground Up Arduino for Interactive Systems | Medium