A

Mini Project Report

On

**Micro-controller based Advanced Touch Sensing**

**Naman Kalkhuria (IEC2010065)**

**Prathak Rastogi (IEC2010093)**

**Prashant Mishra (IEC2010096)**

**B.Tech. - 6th Semester (E.C.)**

Under the Guidance of:

**Dr. Ajay Singh Raghuvanshi**

**I.I.I.T.-Allahabad**



**CERTIFICATE**

This is to certify that the project work entitled, “**Micro-controller based Advanced Touch Sensing**” submitted by

NAMAN KALKHURIA - IEC2010065

PRATHAK RASTOGI - IEC2010093

PRASHANT MISHRA - IEC2010096

Towards partial fulfillment of the requirements of B.Tech. (E.C.) embodies the work done by them under my supervision.

Signature of the Signature of

Chairperson Supervisor

Dated:

**Indian Institute of Information** **Technology Allahabad**

## Table of Contents:

|  |  |
| --- | --- |
| **Content** | **Page Number** |
| 1. Acknowledgement | 4 |
| 1. Abstract | 5 |
| 1. Motivation | 5 |
| 1. Scope | 5 |
| 1. Introduction | 6 |
| 1. Various hardware and software components used 2. Micro-Controller Board 3. Software Used | 6 to 11 |
| 1. Circuit Diagram | 12 |
| 1. Proposed Mechanism 2. Sensing 3. Graph Generation and Analysis Using Computer | 12 to 15 |
| 1. Working Architecture | 15 to 16 |
| 1. Work till NOW | 17 |
| 1. Timeline for future work | 17 |
| 1. References and Suggestions | 18 |

## Acknowledgement

This project could not have been accomplished without **Dr. Ajay Singh Raghuvanshi** who not only served as our supervisor but also encouraged us throughout project. He guided us through the debugging process, never accepting less than our best efforts. We thank him for all his help.

Yours Sincerely,

Naman Kalkhuria (IEC2010065)

Prathak Rastogi (IEC2010093)

Prashant Mishra (IEC2010096)

**Abstract**

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is a micro-controller board widely used in various embedded systems.

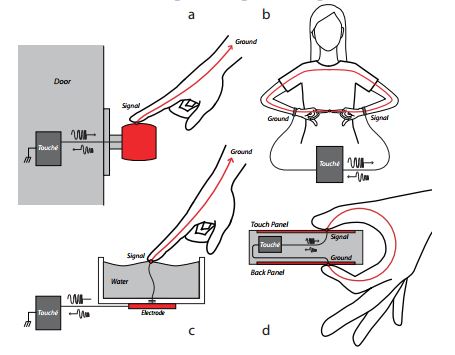
In this project we are using Arduino board as a micro-controller unit for our project and then we are programming it work as PWM generator and ADC to analyze the input signal in advanced touch sensing technique. For this we require the Arduino IDE as a programmer and USB board. Programming on Arduino platform using processing language which is an extension on C/C++ languages.

The signal received using the Arduino board is then analyzed using the graphical capabilities of the Processing programming language.

**Motivation**

In our day-to-day task we come in contact with various types of machines and we always yearn that these machines work according to our will without even telling them to do so. So to make this happen we need a type of interaction with these types of machines.

The most common type of interaction we make with objects or machines is touch. So if we can easily make an object sensitive to our touch and make them understand the way we touch; this could help in automating various tasks. By this thought this project came into our mind.

**Scope**

The scope of our project is far reaching as described by these pictures:

* For detecting the touch on a door knob.
* For detecting configuration of fingers touching to each other.
* For detecting touch on liquid surfaces.
* For detecting the touch and configuration of touch on normal touch sensitive screens.

**Introduction**

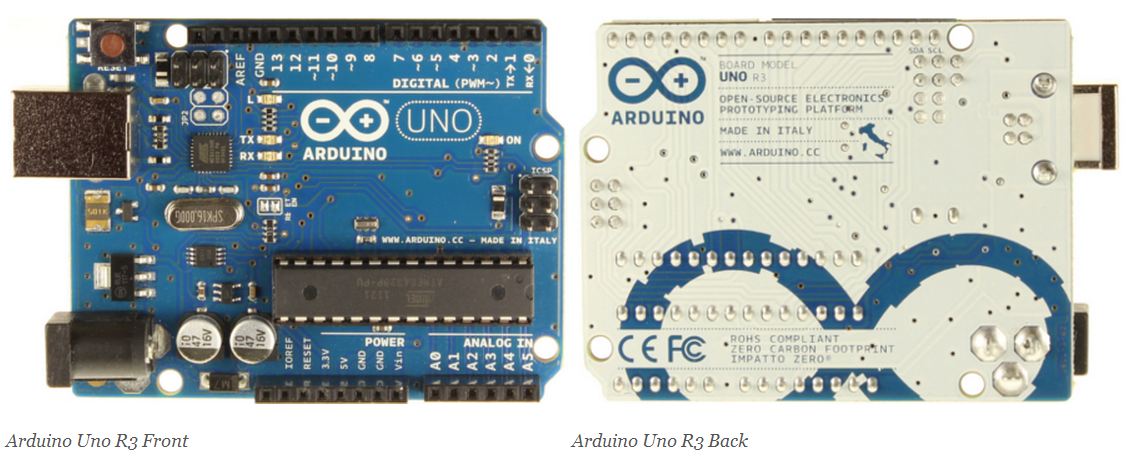
In this project we implement the technique of Swept Frequency Capacitive Touch Sensing. By the help of this technique we can easily detect the touch but we can also detect the configuration of touch. We are implementing this technique using the micro-controller board instead of Digital Signal Processor.

**Various hardware and software components to be use:**

1. **MICRO-CONTROLLER BOARD:-** ARDUINO UNO R3

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

**Overview**

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

### Summary

|  |  |
| --- | --- |
| Microcontroller | ATmega328 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328) of which 0.5 KB used by boot loader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Clock Speed | 16 MHz |

### Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm centre-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

* **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
* **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
* **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
* **GND.** Ground pins.

### Memory

The ATmega328 has 32 KB (with 0.5 KB used for the boot loader). It also has 2 KB of SRAM and 1 KB of EEPROM

### Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode()](http://arduino.cc/en/Reference/PinMode), [digitalWrite()](http://arduino.cc/en/Reference/DigitalWrite), and[digitalRead()](http://arduino.cc/en/Reference/DigitalRead) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

* **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
* **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the [attachInterrupt()](http://arduino.cc/en/Reference/AttachInterrupt) function for details.
* **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite()](http://arduino.cc/en/Reference/AnalogWrite) function.
* **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the [SPI library](http://arduino.cc/en/Reference/SPI).
* **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labelled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the [analogReference](http://arduino.cc/en/Reference/AnalogReference)() function. Additionally, some pins have specialized functionality:

* **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the [Wire library](http://arduino.cc/en/Reference/Wire).

There are a couple of other pins on the board:

* **AREF.** Reference voltage for the analog inputs. Used with [analogReference](http://arduino.cc/en/Reference/AnalogReference)().
* **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the [mapping between Arduino pins and ATmega328 ports](http://arduino.cc/en/Hacking/PinMapping168). The mapping for the Atmega8, 168, and 328 is identical.

### Communication

### The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, [on Windows, a .inf file is required](http://arduino.cc/en/Guide/Windows#toc4). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

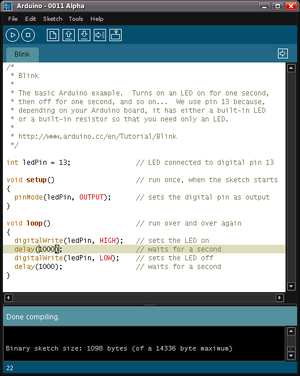
A [Software Serial library](http://www.arduino.cc/en/Reference/SoftwareSerial) allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the [documentation](http://arduino.cc/en/Reference/Wire) for details. For SPI communication, use the [SPI library](http://arduino.cc/en/Reference/SPI).

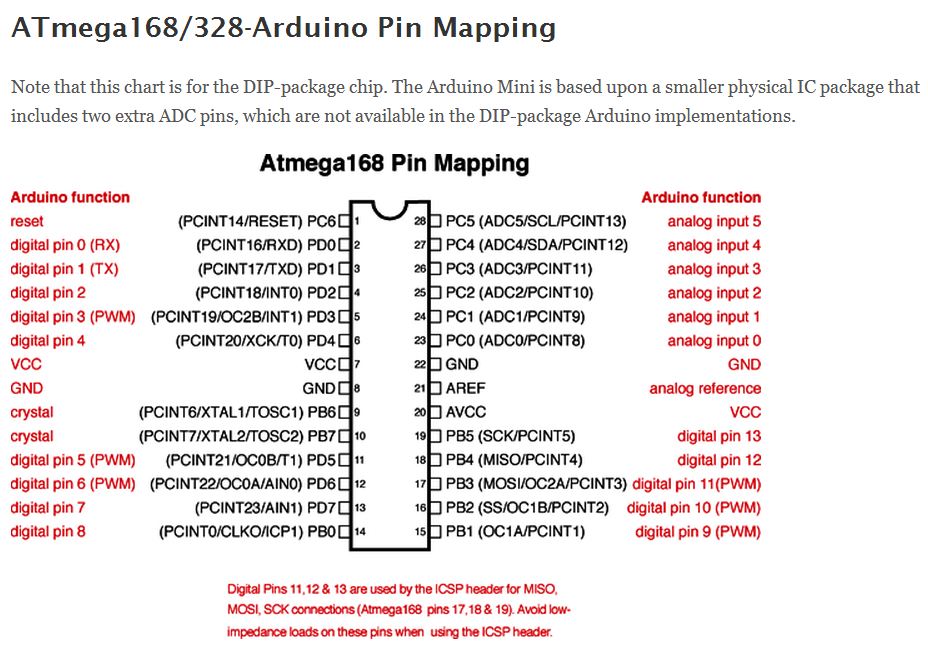
**Programming**

The Arduino Uno can be programmed with the Arduino software.

The ATmega328 on the Arduino Uno comes pre-burned with a [boot loader](http://arduino.cc/en/Tutorial/Bootloader) that allows us to upload new code to it without the use of an external hardware programmer.



* About Micro-controller; ATmega328:
  + High Performance, Low Power AVR® 8-Bit Microcontroller
  + Advanced RISC Architecture
    - 131 Powerful Instructions – Most Single Clock Cycle Execution
    - 32 x 8 General Purpose Working Registers
    - Fully Static Operation
    - Up to 20 MIPS Throughput at 20 MHz
    - On-chip 2-cycle Multiplier
  + High Endurance Non-volatile Memory Segments
    - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
* (ATmega48PA/88PA/168PA/328P)
  + - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
    - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
    - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
    - Data retention: 20 years at 85°C/100 years at 25°C(1)
    - Optional Boot Code Section with Independent Lock Bits
* In-System Programming by On-chip Boot Program
* True Read-While-Write Operation
  + - Programming Lock for Software Security
  + Peripheral Features
    - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
    - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture
* Mode
  + - Real Time Counter with Separate Oscillator
    - Six PWM Channels
    - 8-channel 10-bit ADC in TQFP and QFN/MLF package
* Temperature Measurement
  + - Programmable Serial USART
    - Master/Slave SPI Serial Interface
    - Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
    - Programmable Watchdog Timer with Separate On-chip Oscillator
    - On-chip Analog Comparator
    - Interrupt and Wake-up on Pin Change
  + Special Microcontroller Features
    - Power-on Reset and Programmable Brown-out Detection
    - Internal Calibrated Oscillator
    - External and Internal Interrupt Sources



1. **SOFTWARE USED:**

**Processing Programming Language:**

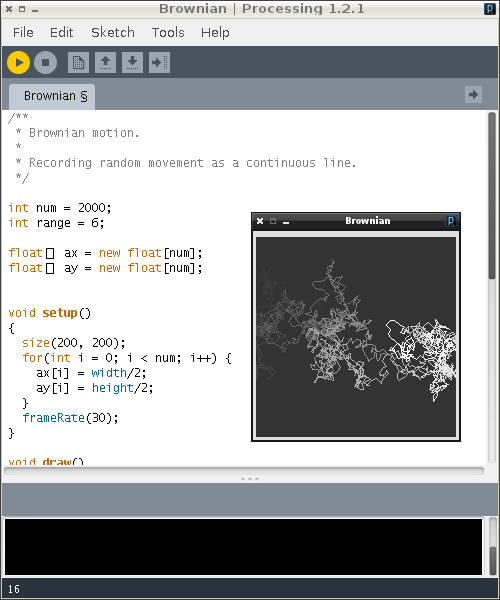
Processing is for writing software to make images, animations, and interactions. The idea is to write a single line of code, and have a circle show up on the screen. Add a few more lines of code, and the circle follows the mouse. Another line of code, and the circle changes colour when the mouse is pressed. We call this sketching with code. You write one line, then add another, then another, and so on. The result is a program created one piece at a time.

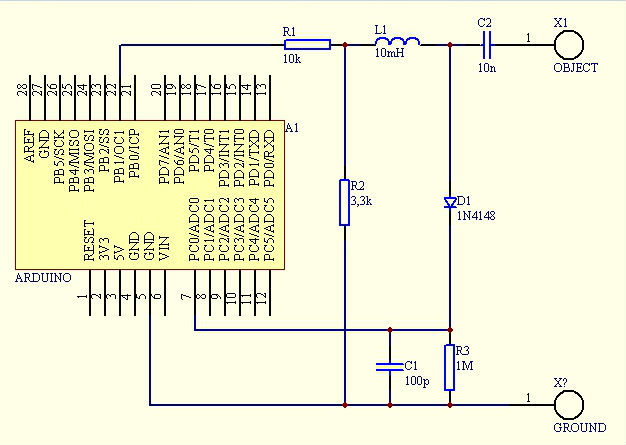
Processing includes a *sketchbook*, a minimal alternative to an [integrated development environment](http://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for organizing projects.

Every Processing sketch is actually a subclass of the [PApplet](http://processing.googlecode.com/svn/trunk/processing/build/javadoc/core/processing/core/PApplet.html) Java class which implements most of the Processing language's features.

When programming in Processing, all additional classes defined will be treated as [inner classes](http://en.wikipedia.org/wiki/Inner_class) when the code is translated into pure Java before compiling. This means that the use of static variables and methods in classes is prohibited unless you explicitly tell Processing that you want to code in pure Java mode.

Processing also allows for users to create their own classes within the PApplet sketch. This allows for complex data types that can include any number of arguments and avoids the limitations of solely using standard data types such as: int (integer), char (character), float (real number), and color (RGB, ARGB, hex).



**CIRCUIT DIAGRAM:**

**PROPOSED MECHANISM:**

The mechanism is divided into two parts:

1. Sensing.
2. Graph generation and analysis using Computer.

**SENSING:-**

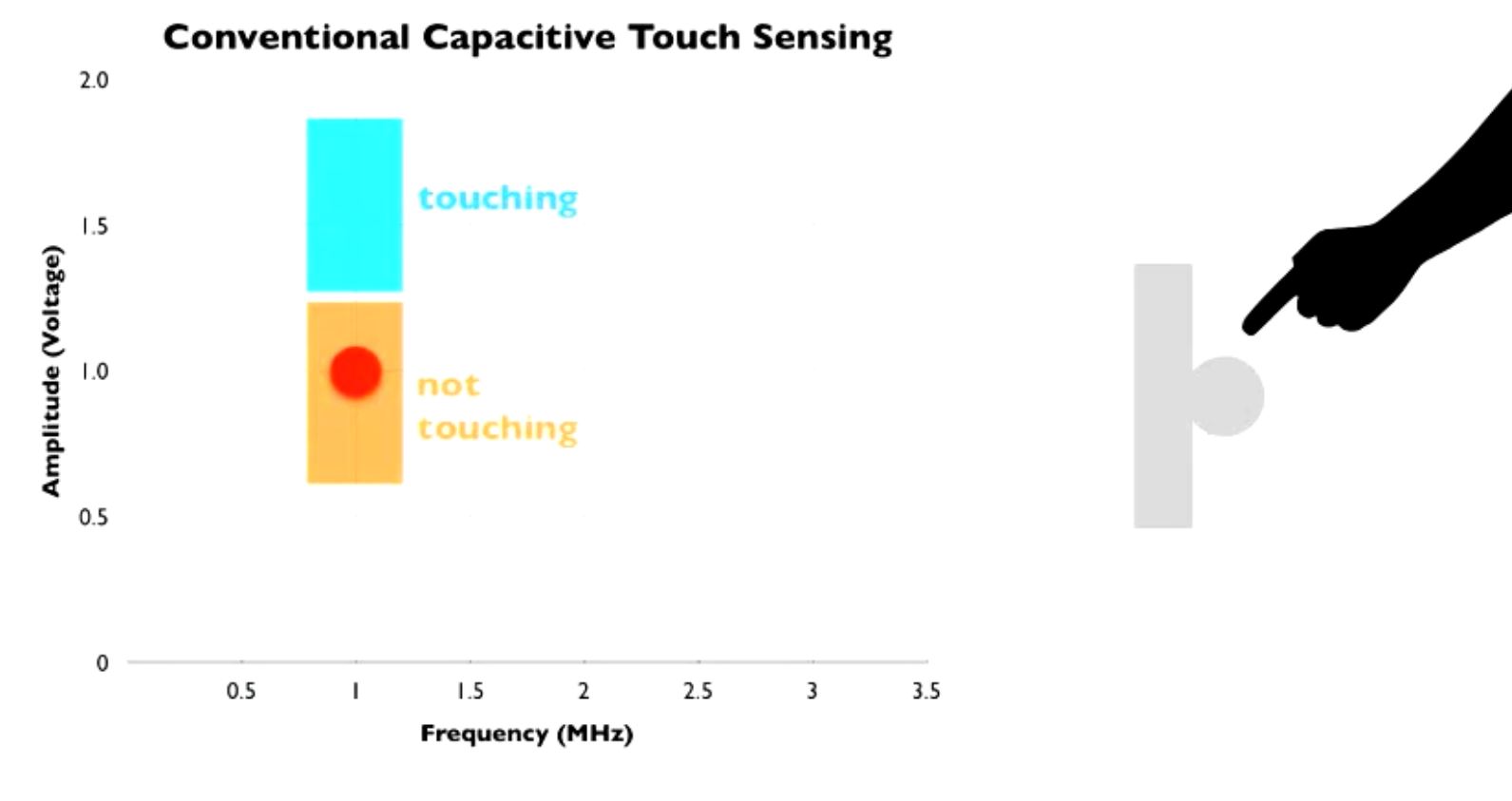
For sensing, we will be using the method of ***Swept Frequency Capacitive Sensing*** technique that can not only detect a touch event, but also recognize complex configurations of the human hands and body.

In a typical capacitive touch sensor, a conductive object is excited by an electrical signal at a fixed frequency. The sensing circuit monitors the return signal and determines touch events by identifying changes in this signal caused by the electrical properties of the human hand touching the object.

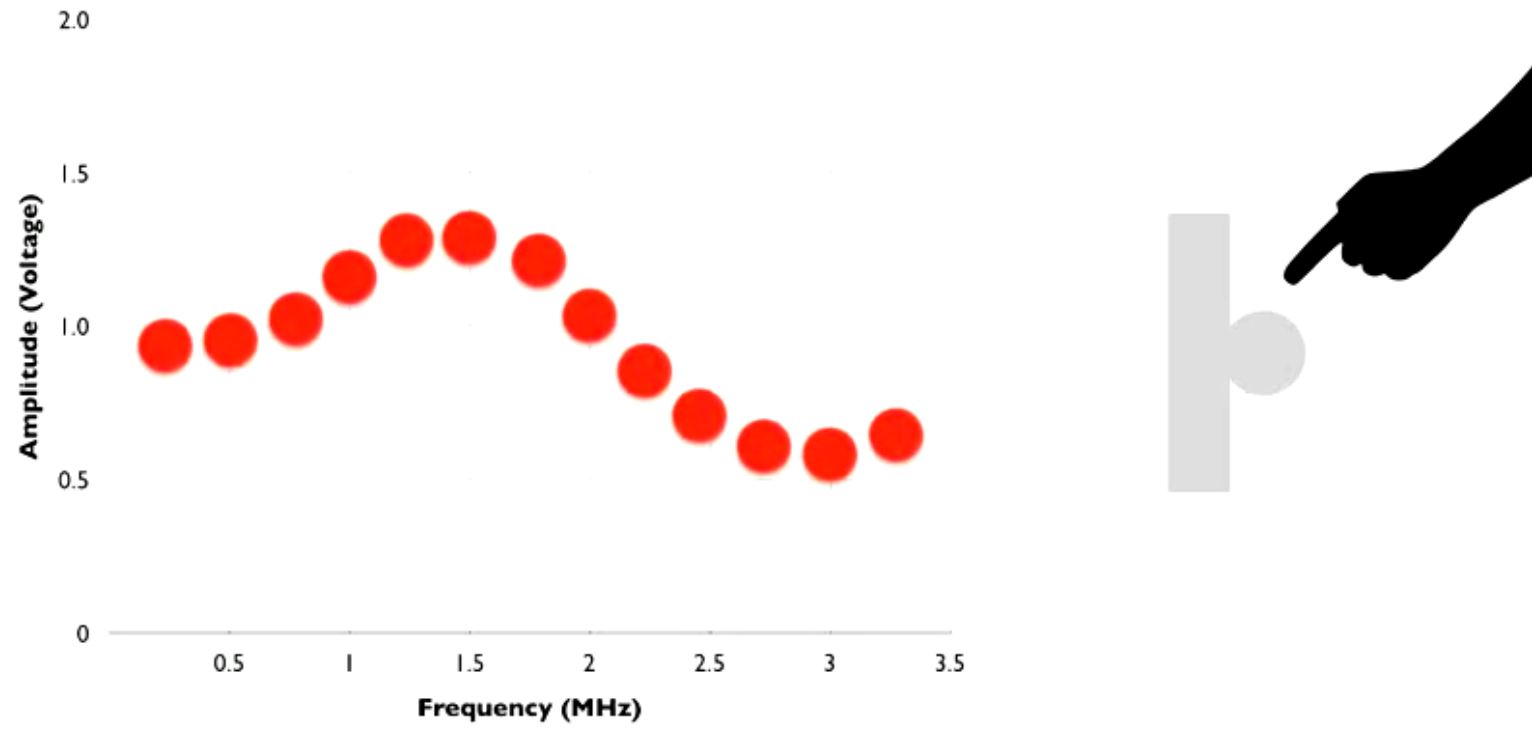
In SFCS, on the other hand, we monitor the response to capacitive human touch over a range of frequencies. Objects excited by an electrical signal respond differently at different frequencies, therefore, the changes in the return signal will also be frequency dependent. Thus, instead of measuring a single data point for each touch event, we measure a multitude of data points at different frequencies. Not only can we determine that a touch event occurred, we can also determine how it occurred. Importantly, this contextual touch information is captured through a single electrode, which could be simply the object itself.

The basic principles of operation in most common capacitive sensing techniques are quite similar: A periodic electrical signal is injected into an electrode forming an oscillating electrical field. As the user’s hand approaches the electrode, a weak capacitive link is formed between the electrode and conductive physiological fluids inside the human hand, altering the signal supplied by the electrode. This happens because the user body introduces an additional path for flow of charges, acting as a charge “sink”. By measuring the degree of this signal change, touch events can be detected.

There is a wide variety of capacitive touch sensing techniques. One important design variable is the choice of signal property that is used to detect touch events, e.g., changes in signal phase or signal amplitude can be used for touch detection. The signal excitation technique is another important design variable. The choice of topology of electrode layouts, the materials used for electrodes and substrates and the specifics of signal measurement resulted in a multitude of capacitive techniques, including charge transfer, surface and projective capacitive, among others.

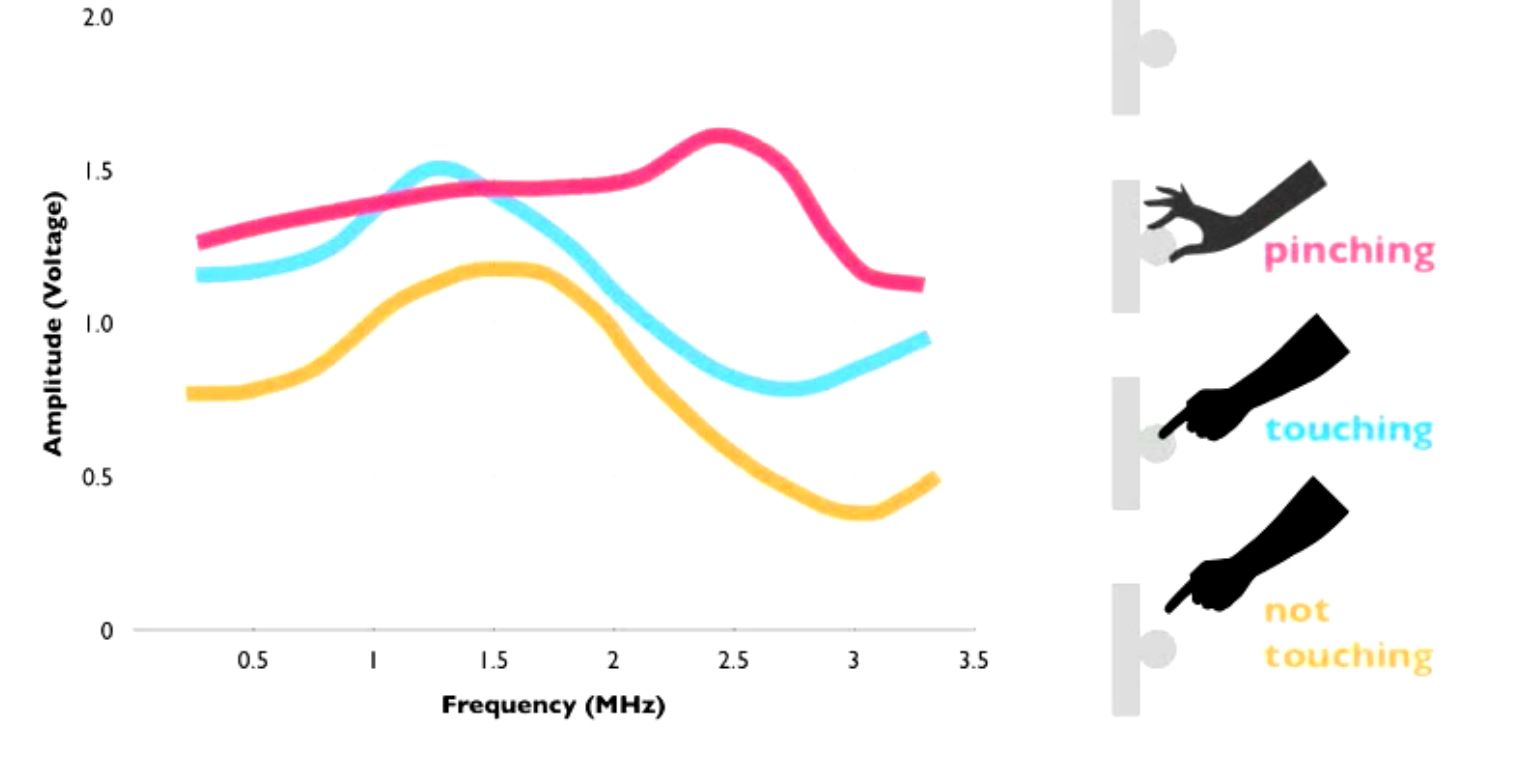


Capacitive sensing is a malleable and inexpensive technology – all it requires is a simple conductive element that is easy to manufacture and integrate into devices or environments. Consequently, today we find capacitive touch in millions of consumer device controls and touch screens. It has, however, a number of limitations. One important limitation is that capacitive sensing is not particularly expressive – it can only detect when a finger is touching the device and sometimes infer finger proximity. To increase the expressiveness, matrices of electrodes are scanned to create a 2D capacitive image. Such space multiplexing allows the device to capture spatial gestures, hand profiles or even rough 3D shapes. However, this comes at the cost of increased engineering complexity, limiting its applications and precluding ad hoc instrumentation of our living and working spaces. Current capacitive sensors are also limited in materials they can be used with. Typically they cannot be used on the human body or liquids.



We advocate a different approach to enhance the expressivity of capacitive sensing – by using frequency multiplexing. Instead of using a single, pre-determined frequency, we sense touch by sweeping through a range of frequencies. We refer to the resulting curve as a capacitive profile and demonstrate its ability to expand the vocabulary of interactive touch without increasing the number of electrodes or the complexity of the sensor itself.

**GRAPH GENERATION AND ANALYSIS USING COMPUTER:-**



For generation of the graph we sense the input from the circuit in different frequencies thus and plotted the amplitude at different output frequencies.

For analysing, we will be using the method of storing the values of the graph in 2D array in which we can analyse the peak of the graph i.e. the point where the amplitude is maximum. By storing the range of values for which we can differentiate different configurations we can easily distinguish between different types of configuration of touching.

**WORKING ARCHITECTURE**

Code for generation of PWM and ADC:

#define SET(x,y) (x |=(1<<y))

#define CLR(x,y) (x &= (~(1<<y)))

#define CHK(x,y) (x & (1<<y))

#define TOG(x,y) (x^=(1<<y))

#define N 160

float results[N];

float freq[N];

int sizeOfArray = N;

void setup()

{

TCCR1A=0b10000010;

TCCR1B=0b00011001;

ICR1=110;

OCR1A=55;

pinMode(9,OUTPUT);

pinMode(8,OUTPUT);

Serial.begin(115200);

for(int i=0;i<N;i++)

results[i]=0;

}

void loop()

{

unsigned int d;

int counter = 0;

for(unsigned int d=0;d<N;d++)

{

int v=analogRead(0);

CLR(TCCR1B,0);

TCNT1=0;

ICR1=d;

OCR1A=d/2;

SET(TCCR1B,0);

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

freq[d] = d;

// plot(v,0);

// plot(results[d],1);

// delayMicroseconds(1);

}

PlottArray(1,freq,results);

TOG(PORTB,0);

}

**Work till now: (***till 5th March, 2013***)**

Learned about the working of the circuit

Learned about working of timers and ADCs in ATmega

Tried to program the PWM generation and ADC program

Learned the Processing programming language

Tried to work on the GUI for the computer

Collected the circuit components

**Timeline for future Work**

Will try to implement the circuit

Try to complete the GUI

Will complete the task of sensing

Try to complete the task of distinguishing and will try to implement that

**References:**

* <http://www.instructables.com/id/Touche-for-Arduino-Advanced-touch-sensing/?ALLSTEPS>
* <http://www.youtube.com/watch?v=E4tYpXVTjxA>
* <http://arduino.cc/blog/2012/06/01/touche-with-arduino/>

**Suggestions:**