What is capacitive touch sensing?

Simply put, its an input method that uses the change in capacitance that occurs when a conductive body comes close to a conductive surface, usually a small area of copper on a pcb.

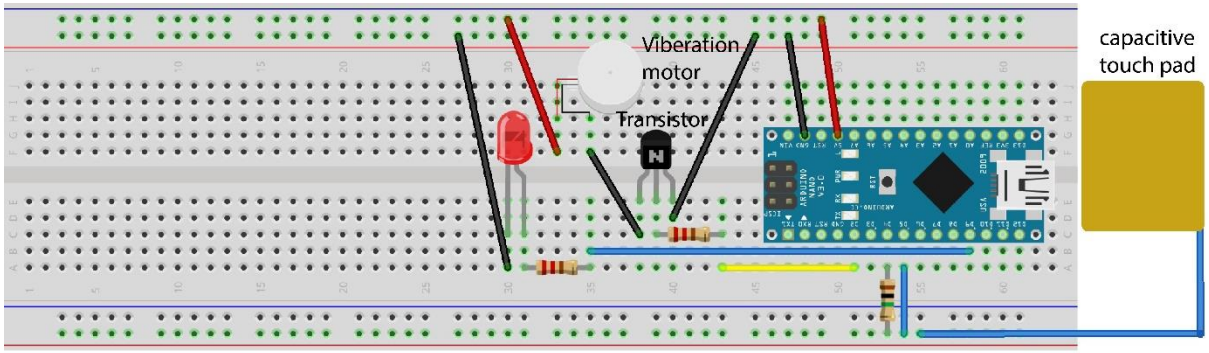
The most significant electrical change that occurs when the capacitance in a circuit changes is frequency, so most capacitive touch setups use some form of timing to detect the change in capacitance, the Arduino capacitive touch library does the same, it sends data from one pin to the other through a resistor and records the time it takes to receive the data on the second pin.

How to connect it

Connecting the Arduino for capacitive touch input is just as simple as connecting a button to the Arduino, all you need is a resistor (1M – 5M recommended), and a wire connected to the conductive surface you intend to use as your touch pad, in the example, I am using a small copper board with a flat piece of plastic on it to act as a separator; your finger should not be making direct contact with the conductor.

The resistor should be connected between the send and receive pins, the conductive surface should be connected to the receive pin.

The test setup in the image also includes an LED and a transistor switching circuit for controlling a vibration motor.



Arduino capacitive touch with viberation feedback (test setup)

How to detect a touch

Although the capacitive touch connection is just as simple as connecting a push button, the same cannot be said for the detection. Unlike a push button where you always have a discrete signal (high or low) , capacitive touch sensing on the Arduino yield continuous and very often noisy signals, So a simple if statement will not suffice if you want to build a robust capacitive touch control with the Arduino. This is where my Advanced Capacitive touch library comes in.

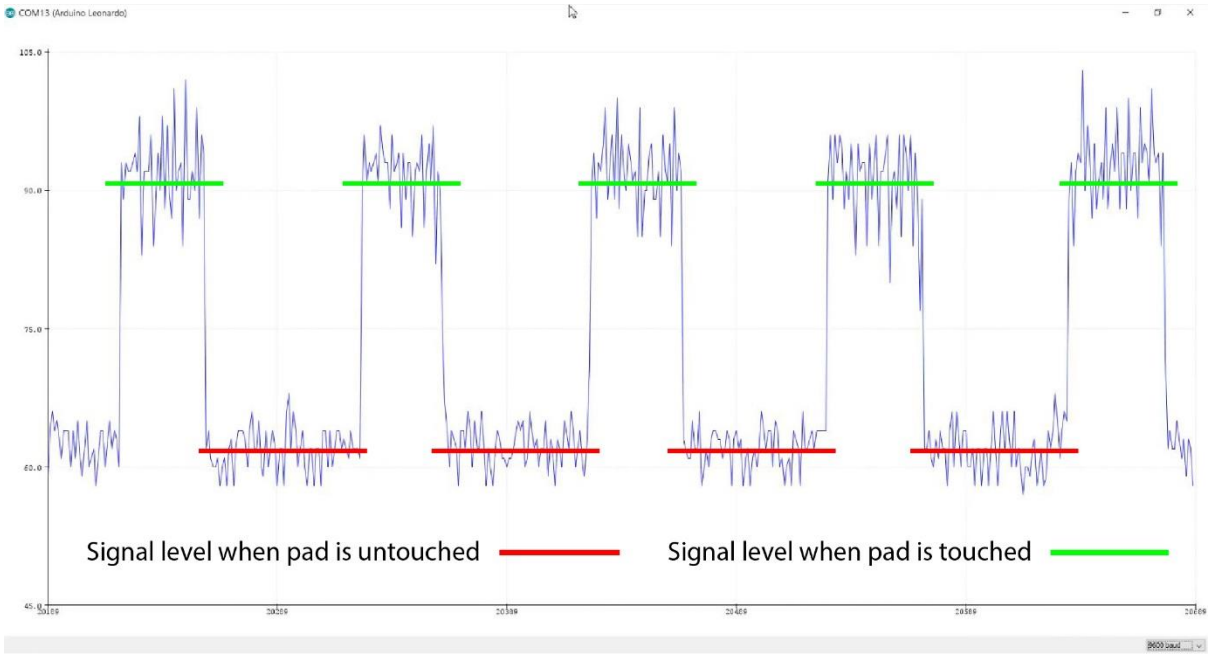
AdvCapTouch Library.

These capacitive touch detection library is written as an extension of the Arduino capacitive touch library, it offers two detection algorithms that solves two of the main problems associated with detecting capacitive touch on the Arduino. The library also allows advanced touch detections; such as, Double tap, Short press and Long press. Also featured in the library is a haptics controller, this allows

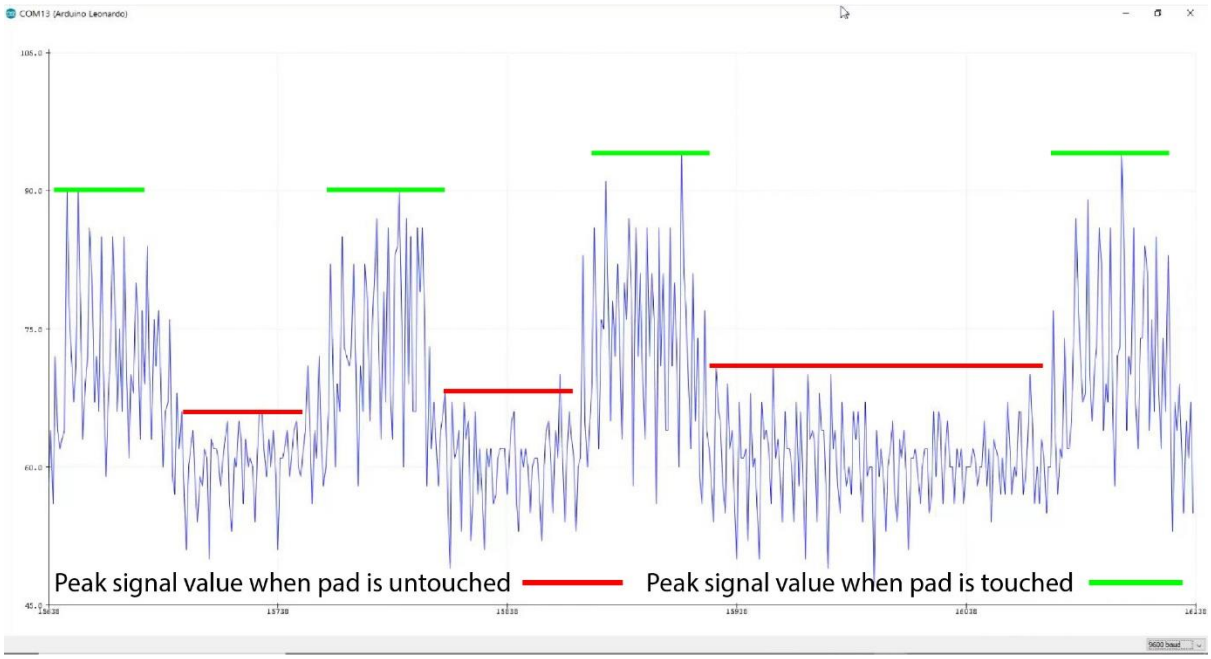
you to connect and control a vibration motor as a feedback when you interact with the capacitive touch surface.

The two main factors that can cause problems for your capacitive touch setup are:

- The inconsistency from the values the Arduino spits out – There's really not much you can do about this, plus the resulting signal is usually still very usable. This is mostly caused by the power supply to the Arduino, from my tests, the value appears more consistent with a linear power supply like a battery. Switching power supplies will introduce more noise to the system.
- The inconsistency caused by Electromagnetic interference (EMI) – EMI is usually the cause of most of the noise in your signal. A proper detection algorithm can be used to salvage the touch signal from the noise; as I'll demonstrate later on, only in extreme cases will the signal become truly unusable.



Fairly stable capacitive touch setup



Noisy capacitive touch setup

Depending on the conditions surrounding your capacitive touch setup, you'll have a signal similar to one of the images shown above to work with. Upload the "see noise level" code sample included in the library to determine the type of signal that your capacitive touch setup yields.

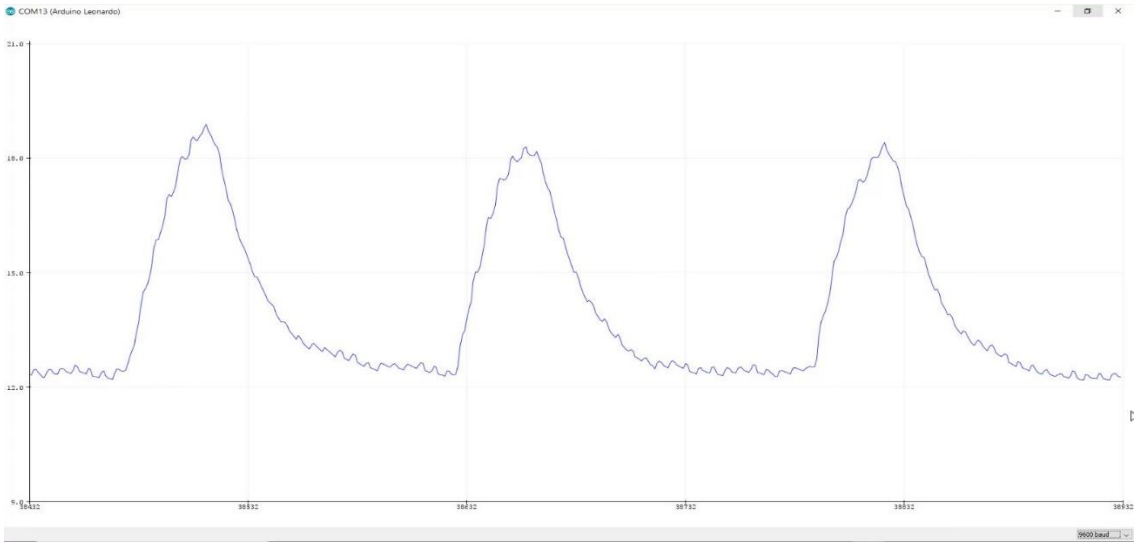
The first signal shows a fairly stable system, where the touch can clearly be seen as well-defined bumps in the signal, in this case you can simply take the average of the signal over time and use that as the base value for comparison.

In this case the code is taking the average of the signal as the bases for detection.

It the second signal, the system is a lot noisier, and there are no clearly defined bumps in the signal, what we do see is a slight change in the peak value, so we can assume that whenever the peak value starts to reach a certain level, a touch as being detected.

In this case the code is taking the peak value of the signal as the bases for the detection. This method is a little slower, since the program has to take a small sample from the signal, find the peak from said sample and then damping the oscillations in order to produce a more defined signal.

Here's what the signal looks like after the library processes it.



Processed signal

The actual signal levels and detection thresholds are determined by the thickness of the material separating the conductive surface and your finger, the size of the conductive surface, the value of resistor used, level of interference and the type of power supply used. This means your setup will be unique to you, you just need to play around with the values to see what works for your specific needs.

Using the library.

Initializing the library.

```
#include <AdvCapTouch.h>


AdvCapTouch samplepad  = AdvCapTouch();  //create a new captouch object

int touchtype;


void setup() {

 samplepad.set_inputTypeThresholds(20, 40, 70, 150); // set the thresholds for the four input types  (singletap, shortpress, longpress, doubletapspeed)

 samplepad.set_detectionThreshold(400, 100);  //set touch sensitivity in the form of detection, rejection thresholds values

 samplepad.set_capTouchPins(3,4,0,0,0);   //set arduino pins associated with the pads (sendpin, receivepin1, receivepin2, receivepin3, receivepin4) this example uses just one pad.

 samplepad.initialize_capTouch(1);

 samplepad.set_haptics(6,40,255);  //set haptic feedback variables (arduino pwm pin, duration of haptics(ms), pwn strength from 0-255)

}
```

The code snippet above shows how you would typically setup the library.

Line 2 is used to create a new capacitive touch object; the object is named “samplepad” but you can give any other name.

In the setup section of the code, the first line calls the function set_inputTypeThresholds(), this function is used to set the different detection levels of the four detectable capacitive touch types

(single tap, short press, long press and double tap). Think of the numbers as how long your finger must be on the pad for specific touch types. For example, the code above sets 20 as the single tap threshold; this means any touch input that yields a count less than 20 will be categorized as a single tap, if the touch yields a count greater than 20 but less than 40, the touch is categorized as a short press, the same logic applies to the remaining touch types.

The second line set_detectionThreshold(); is used to set the sensitivity of the capacitive touch surface. The function accepts two integers, the touch detection threshold and the touch rejection threshold, the recommended ratio of detection to rejection threshold is 3:1. The larger the detection thresholds, the lower the pad sensitivity. The sensitivity can also be set to auto with the function set_adaptiveSensitivity(0, true), the "0" represents the capacitive touch pad designation, "true" turns on auto sensitivity on the designated pad.

Line three is used to configure the Arduino pins connected to the resistor and the touch surfaces. The code sample uses just one pin i:e one touch pad, which is why the last three digits are zeros. You can have up to 4 pads connected, all of which will share the same send pin.

Line four is used to initialize the capacitive touch, the function initialize_capTouch() accepts an integer from 1 – 4, representing the number of pads your working with.

Line five is optional, only use this function when you have a haptics circuit setup.

Detecting touch in the main program loop.

```
void loop() {

touchtype = samplepad.detect_touch(0);  //function return 1-4 based on the input detected, 1 = singletap, 2 = doubletap, 3 = shortpress, 4 = longpress

samplepad.update_basevalue(0);

}
```

There are two main functions for running the touch detection algorithm, the first is the detect_touch() function. This function should be used for a low noise touch setup, the function will return an integer value from 0 – 4.

- 0 means no touch was detected.
- 1 means a single tap was detected.
- 2 means a double tap was detected.
- 3 means a short press was detected.
- 4 means a long press was detected.

The function detect_touch() should always be called with a integer value from 0 – 3, this value represents the position/designation of the capacitive touch pad you want to run a touch detection on.

For noisy touch setups, use the detect_touchFromNoise() function instead, this function is more reliable and works in the exact same way as detect_touch(), but because this algorithm involves signal sampling, its just a little bit slower than the alternative.

The second function update_basevalue() or update_basevalueFromNoise() for noisy signals is used to continuously update the nominal value of the touch pad. This function is integral to making the touch detection adaptable across various touch conditions.