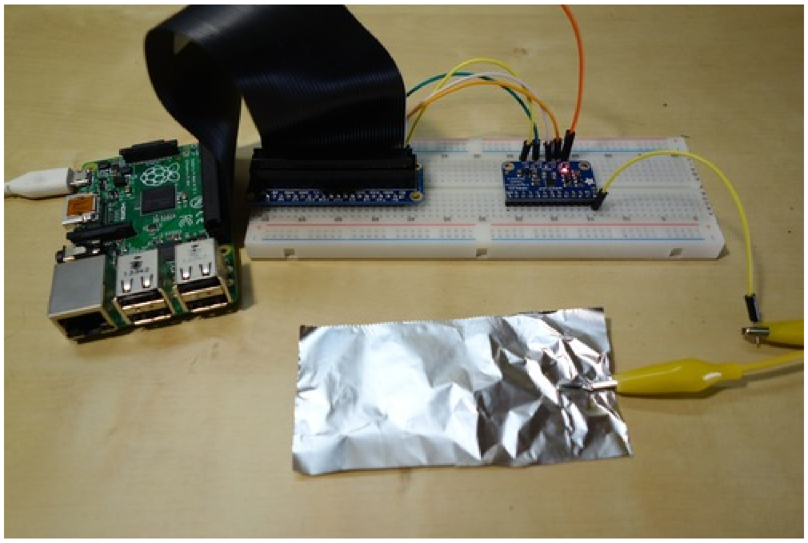
MPR121 Capacitive Touch Sensor on Raspberry Pi & BeagleBone Black

Created by Tony DiCola



Last updated on 2018-08-22 03:45:37 PM UTC

Guide Contents

Guide Contents 2
Overview 3
Hardware 4
Parts 4
Wiring 4
- Raspberry Pi 4
- BeagleBone Black 5

Software 6
Dependencies 6
Installation 6
Usage 7
Example Code 7
Library Usage 7
Raspberry Pi Virtual Keyboard 10
Wiring 10
Dependencies 10
Configuration 10
Usage 11
- Launch In Background 11
- Stop Background Process 12

Electrodes 13

Overview

Reach out and touch someone, or some*thing*, with the MPR121 capacitive touch sensor and a Raspberry Pi or BeagleBone Black! This breakout provides 12 capacitive touch inputs and all the logic to read them over a simple I2C communication bus. For example you can make your own buttons, detect if someone touched a device, create a fun plush game controller (https://adafru.it/ne8), or even a wacky devices like a banana piano (https://adafru.it/eid)!

With the MPR121 python library you can now easily use the MPR121 capacitive touch sensor with a Raspberry Pi or BeagleBone Black. Follow this guide to learn how to connect the MPR121 to a development board, install the python library, and start using the MPR121 with your own programs on the Pi or BeagleBone Black. I'll even show you how to create a 'virtual keyboard' to send key presses when inputs are touch on a Raspberry Pi--great for making a fun controller kind of like the Makey Makey (https://adafru.it/eie)!

Before you get started you'll want to be familiar the MPR121 breakout and connecting to your development board's Linux terminal. Check out the following guides if you need a little more information:

- Adafruit MPR121 12-Key Capacitive Touch Sensor Tutorial (https://adafru.it/dKH)
- Adafruit Raspberry Pi Lesson 6: Using SSH (https://adafru.it/jsE) or SSH To BeagleBone Black Over USB (https://adafru.it/ne9)

In addition you'll want to make sure your Raspberry Pi is running the latest version of the Raspbian (https://adafru.it/dpb) or Occidentalis (https://adafru.it/dvg) operating system, or your BeagleBone Black is running the latest Debian (https://adafru.it/dUI) operating system.

When you're ready to start, continue on to learn what hardware you'll need and how to wire it together.

Hardware

Parts

You'll need the following parts to build this project:

- MPR121 Breakout (https://adafru.it/dKK)
    - If you haven't assembled it yet, follow the MPR121 guide (https://adafru.it/dKH) to solder the breakout board.
- Long wires, aluminum foil, or other surface (https://adafru.it/nea) to use as capacitive touch inputs to the MPR121.
- Raspberry Pi (http://adafru.it/1914) (any model A, B, A+, or B+), or BeagleBone Black (http://adafru.it/1876).
- Breadboard (http://adafru.it/239) and jumper wires (http://adafru.it/1957).

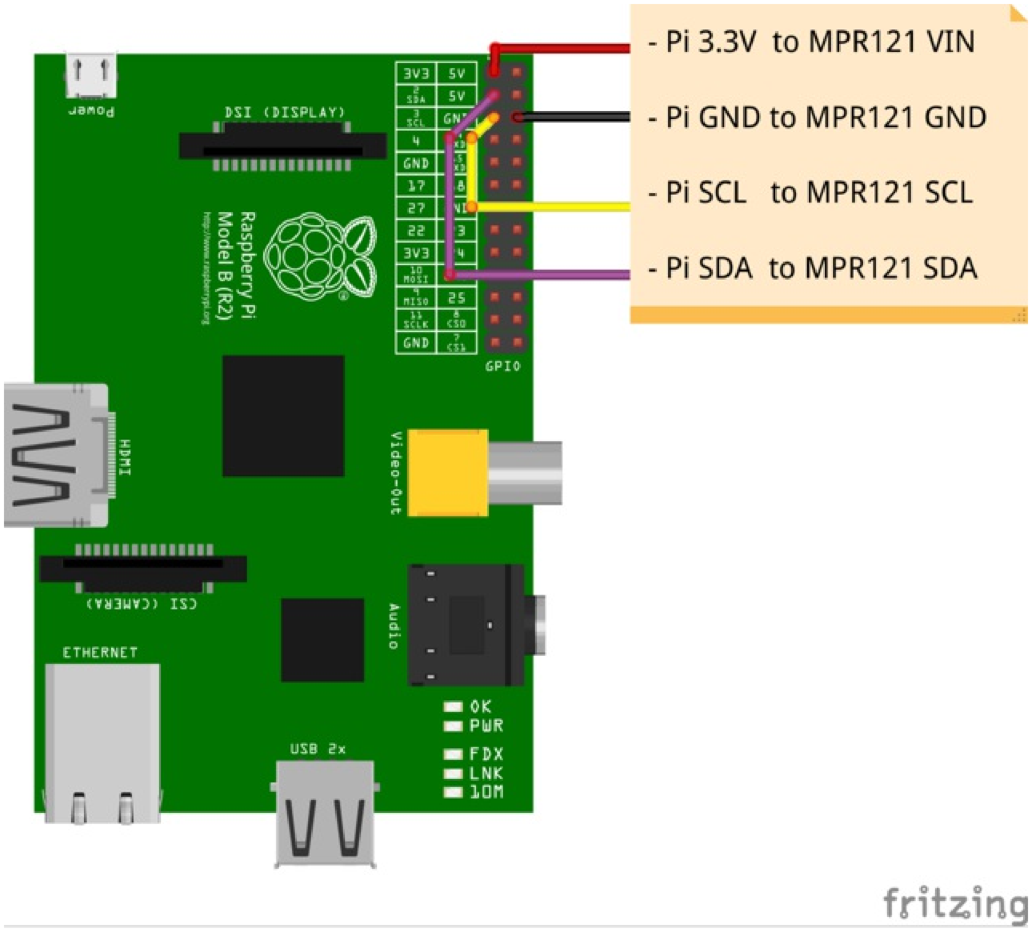
Wiring

Place the MPR121 board into a breadboard and connect its inputs to the electrodes you plan to use. Then follow the wiring below for your platform to connect the MPR121 to an I2C communication channel.

Raspberry Pi

On a Raspberry Pi connect the hardware as follows.

Note: Make sure you've enabled I2C on your Raspberry Pi (https://adafru.it/dEO)!

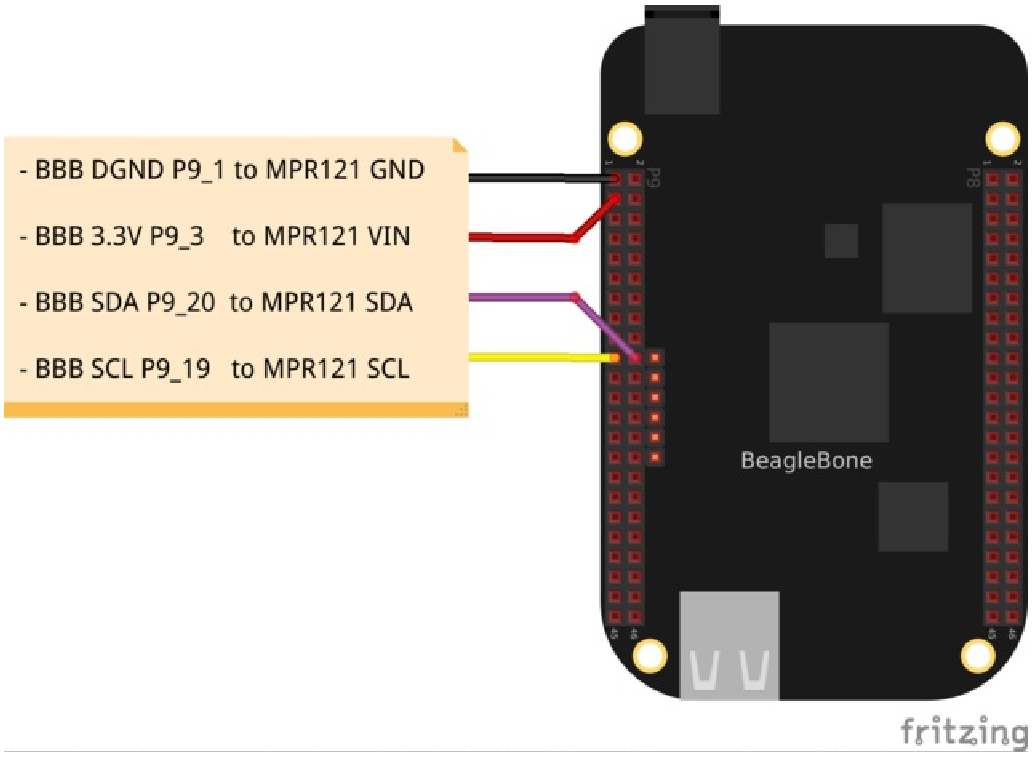


- Connect **Raspberry Pi 3.3V** to **MPR121 VIN**.

- Connect **Raspberry Pi GND** to **MPR121 GND**.
- Connect **Raspberry Pi SCL** to **MPR121 SCL**.
- Connect **Raspberry Pi SDA** to **MPR121 SDA**.

BeagleBone Black

On a BeagleBone Black connect the hardware as follows. If you aren't familir with how pins are numbered on the BeagleBone Black be sure to read this guide first (https://adafru.it/dCI).


- BBB DGND P9_1 to MPR121 GND
- BBB 3.3V P9_3 to MPR121 VIN
- BBB SDA P9_20 to MPR121 SDA
- BBB SCL P9_19 to MPR121 SCL
P9
P8
BeagleBone
fritzing

- Connect **BeagleBone Black 3.3V pin P9_3** to **MPR121 VIN**.
- Connect **BeagleBone Black DGND pin P9_1** to **MPR121 GND**.
- Connect **BeagleBone Black I2C2 SCL pin P9_19** to **MPR121 SCL**.
- Connect **BeagleBone Black I2C2 SDA pin P9_20** to **MPR121 SDA**.

Once the hardware is connected to your development board, continue on to learn how to install the software for this project.

Software

Note: The latest Raspbian Jessie and Stetch releases changed the I2C driver to not support a repeated start mode used by the MPR121. See this thread for more details and a workaround (mentioned below): https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=184377

As noted above the latest Raspbian Jessie and Stretch releases have an updated I2C driver that isn't compatible with how this library enables repeated I2C start bits. From the thread about this issue (https://adafru.it/eWs) there's a workaround to enable the older I2C driver. Edit the **/boot/config.txt** on your SD card and add:

```
dtoverlay=i2c-bcm2708
```

Then restart your Pi and you should be able to use the library below. If this still fails you might need to fall back to an earlier version of Raspbian which comes with the older driver, see the historical Raspbian images here (https://adafru.it/Cbj) and pick one from 2016 like 2016-11-29.

Dependencies

Before you install the software you'll first need to install a few dependencies. Make sure your Raspberry Pi or BeagleBone Black is connected to the internet with a wired or wireless connection. Then connect to a terminal on the device using SSH and execute the commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip git
```

When prompted to install the dependencies answer yes, and after a few moments the necessary libraries and software should be installed. You can also ignore any messages about dependencies which are already installed.

Installation

Follow the steps below to install the software for the Python MPR121 library (https://adafru.it/eij). At a terminal execute the following commands to download the library into a subdirectory of your user's home directory.

```
cd ~
git clone https://github.com/adafruit/Adafruit_Python_MPR121.git
```

Next execute the following commands to install the library:

```
cd Adafruit_Python_MPR121
sudo python setup.py install
```

The setup script will download a few dependencies and install the library. If there are errors, carefully check all the dependencies above were installed and try again.

That's it, the MPR121 library is now installed! Continue on to learn how to run an example program and learn how to use the library.

Usage

Example Code

Once the library is installed you can run an example program to test the capacitive touch inputs. Navigate to the **examples** folder of the library and run the **simpletest.py** example by executing:

```
cd examples
sudo python simpletest.py
```

Note that the example is run as a root user with the sudo command, this is required to access the hardware on the Pi.

Once the example is running you should see text like the following printed on the screen:

```
Adafruit MPR121 Capacitive Touch Sensor Test
Press Ctrl-C to quit.
```

Try pressing some of the capacitive inputs you've connected (or even just touching the solder joints of the inputs!). You should see a message each time an input is pressed and released, for example:

```
11 touched!
10 touched!
10 released!
11 released!
10 touched!
11 touched!
10 released!
11 released!
```

When you're finished, press **Ctrl-C** to quit the program.

Library Usage

To understand how the library works, let's examine the **simpletest.py** example code in detail. Open the file in a text editor and follow along as I explain each important part below.

```
import sys
import time

import Adafruit_MPR121.MPR121 as MPR121


print 'Adafruit MPR121 Capacitive Touch Sensor Test'
```

First you'll see the example import modules which it will use and print a message on the screen. Note the **Adafruit_MPR121.MPR121** module which is imported as the shorter MPR121 name. This module is the MPR121 library that was installed earlier and this line will make it available to our python program.

```
# Create MPR121 instance.
cap = MPR121.MPR121()

# Initialize communication with MPR121 using default I2C bus of device, and
# default I2C address (0x5A).  On BeagleBone Black will default to I2C bus 0.
if not cap.begin():
    print 'Error initializing MPR121.  Check your wiring!'
    sys.exit(1)

# Alternatively, specify a custom I2C address such as 0x5B (ADDR tied to 3.3V),
# 0x5C (ADDR tied to SDA), or 0x5D (ADDR tied to SCL).
#cap.begin(address=0x5B)

# Also you can specify an optional I2C bus with the bus keyword parameter.
#cap.begin(bus=1)
```

Next an instance of the MPR121 class is created and initialized. You can see the first line creates the object, and the next lines call the **begin()** function to initialize the device. It's very important to make sure you call **begin()** before you call any other functions on the MPR121 object.

Also notice in the comments that you can customize the I2C address or bus by sending optional parameters to the **begin()** function. By default the function will try to pick the right I2C bus for your device but these parameters allow you to specify explicit values.

```
# Main loop to print a message every time a pin is touched.
print 'Press Ctrl-C to quit.'
last_touched = cap.touched()
while True:
    current_touched = cap.touched()
    # Check each pin's last and current state to see if it was pressed or released.
    for i in range(12):
        # Each pin is represented by a bit in the touched value.  A value of 1
        # means the pin is being touched, and 0 means it is not being touched.
        pin_bit = 1 << i
        # First check if transitioned from not touched to touched.
        if current_touched & pin_bit and not last_touched & pin_bit:
            print '{0} touched!'.format(i)
        # Next check if transitioned from touched to not touched.
        if not current_touched & pin_bit and last_touched & pin_bit:
            print '{0} released!'.format(i)
    # Update last state and wait a short period before repeating.
    last_touched = current_touched
    time.sleep(0.1)
```

Now the program enters its main loop where it reads each input and checks if it's changed.

The most important thing to see is the **touched()** function. This function returns a 12-bit value where each bit represents one of the 12 inputs on the MPR121 board. Bit 0 represents input 0, bit 1 represents input 1, etc. all the way up to bit 11 and input 11. If a bit is 1 then the input is being touched, and if it's 0 then the input is not being touched.

This loop will compare the previous value for each bit to its current value, so if the value changes a message can be printed. Detecting if an input is touched is as simple as calling the touched() function and checking the value of a bit!

```
    # Alternatively, if you only care about checking one or a few pins you can
    # call the is_touched method with a pin number to directly check that pin.
    # This will be a little slower than the above code for checking a lot of pins.
    #if cap.is_touched(0):
    #    print 'Pin 0 is being touched!'
```

Further below you can see commented code which describes how to use the **is_touched()** function. This function is a little simpler than the **touched()** function as it only checks if one input is being touched and returns **True** (input is being touched) or **False** (input is not being touched). If you only need to check one or a few inputs this function might be an easier option.

```
    # If you're curious or want to see debug info for each pin, uncomment the
    # following lines:
    #print '\t\t\t\t\t\t\t\t\t\t\t\t\t 0x{0:0X}'.format(cap.touched())
    #filtered = [cap.filtered_data(i) for i in range(12)]
    #print 'Filt:', '\t'.join(map(str, filtered))
    #base = [cap.baseline_data(i) for i in range(12)]
    #print 'Base:', '\t'.join(map(str, base))
```

Finally the last commented section shows how you can see interesting debug info for the device. The **filtered_data()** and **baseline_data()** functions each can be called to look up the filtered data and baseline data register values for an input. Read the MPR121 datasheet (https://adafru.it/dKG) to better understand how the chip uses baseline and filtered data to detect touches.

That's all there is to using the MPR121 library! If you find issues or would like to contribute to the library, feel free to do so at it's home on GitHub (https://adafru.it/eij).

Raspberry Pi Virtual Keyboard

One great use for the MPR121 is as a capacitive touch keyboard, where pressing a touch input causes a key to be pressed on a Raspberry Pi. This is kind of like a Makey Makey (http://adafru.it/1068) but built right into the Pi using just the MPR121 and some special software. You could for example configure the MPR121 to act as a giant gamepad that controls games on the Raspberry Pi!

Wiring

To use the MPR121 as a virtual keyboard you'll first want to make sure you've followed the earlier pages in this guide to connect the MPR121 to the Raspberry Pi and install the software.

Next you'll need to add one additional wire to the MPR121 setup. This is to connect the MPR121 IRQ line to an input on the Raspberry Pi so the software can quickly detect when an input has been touched without having to continually poll the MPR121 (which takes a lot of CPU!).

By default the virtual keyboard example assumes input **#26** (only exposed on the Pi Model A+ or B+!) is connected to the **IRQ** line. If you're using an earlier Pi, connect the IRQ line to any free digital input on the Pi, such as input **#18**.

Dependencies

Now open a terminal on the Raspberry Pi using SSH and execute the following commands to install a few dependencies required by the virtual keyboard script:

```
sudo apt-get update
sudo apt-get install libudev-dev
sudo pip install python-uinput
```

Configuration

After the dependencies are installed navigate to the MPR121 library **examples** folder again. Open the **keyboard.py** script in a text editor such as nano by executing:

```
nano keyboard.py
```

Now scroll down to the key configuration near the top of the file:

```
# Define mapping of capacitive touch pin presses to keyboard button presses.
KEY_MAPPING = {
                0: uinput.KEY_UP,    # Each line here should define a dict entry
                1: uinput.KEY_DOWN,  # that maps the capacitive touch input number
                2: uinput.KEY_LEFT,  # to an appropriate key press.
                3: uinput.KEY_RIGHT, #
                4: uinput.KEY_B,     # For reference the list of possible uinput.KEY_*
                5: uinput.KEY_A,     # values you can specify is defined in linux/input.h:
                6: uinput.KEY_ENTER, # http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/include
                7: uinput.KEY_SPACE, #
              }                      # Make sure a cap touch input is defined only
                                     # once or else the program will fail to run!

# Input pin connected to the capacitive touch sensor's IRQ output.
# For the capacitive touch HAT this should be pin 26!
IRQ_PIN = 26
```

The **KEY_MAPPING** variable is a dictionary that maps an input number on the MPR121 to a keyboard button that will be sent when the input is pressed.

For example the code above configures input 0 to the UP key, input 1 to the DOWN key, input 2 to the LEFT key, etc.

Adjust the inputs and key codes depending on your needs. Most of the key codes are self explanatory (i.e. the key code for the letter Q is **uinput.KEY_Q**), but if you are unsure of an input you can find the name of a keycode in the Linux input header here (https://adafru.it/eik). Take the key name and add **uinput.** to the front of it to get the key code that should be in the configuration above.

If you need to add more inputs you can add them as new lines after input 7 above. **Be careful to make sure each new line ends in a comma so the python dictionary is defined correctly.**

Also if you had to connect the IRQ line to a different input than **#26**, be sure to set the **IRQ_PIN** value to the input number you are using.

After you've configured your key mapping save the file by pressing **Ctrl-O** and **Enter**, then quit by pressing **Ctrl-X**.

Usage

Now run the program by executing:

```
sudo python keyboard.py
```

After a moment you should see a message displayed that tells you to press **Ctrl-C** to quit the program. If you press inputs to the MPR121 they should send the keys you've configured to the Raspberry Pi!

Note that you won't see any output from the program when keys are pressed.

Quit the program by pressing **Ctrl-C**.

Launch In Background

Running the program by itself is great, but you probably want to run the program in the background while a game or other application runs and takes input from the MPR121 key presses. To do this you can launch the program into the background by executing at the terminal:

```
sudo python keyboard.py &
```

You should see a response such as:

```
[1] 2251
```

This tells you the program is launched in the background and is currently running under the process ID 2251. Try to remember the process ID as it will help you shut down the program later (but don't worry, I'll show you how to shut down the program even if you forget the ID).

Now run a game or other program that relies on keyboard input. Try pressing inputs on the MPR121 and you should see them register as keyboard presses!

Stop Background Process

To stop the background process you'll need to tell Linux to kill the python keyboard.py process that was launched in the background earlier. If you remember the process ID number you can skip below to the kill command. However if you forgot the process ID number you can find it by executing a command like this to search all running processes for the keyboard.py script:

```
ps aux | grep keyboard.py
```

You should see a list of processes such as:

```
root      2251  0.5  0.3   5136  1488 pts/0    S    09:13   0:00 sudo python keyboard.py
root      2252 13.2  1.4  18700  5524 pts/0    Sl   09:13   0:00 python keyboard.py
pi        2294  0.0  0.2   4096   804 pts/0    S+   09:13   0:00 grep --color=auto keyboard.py
```

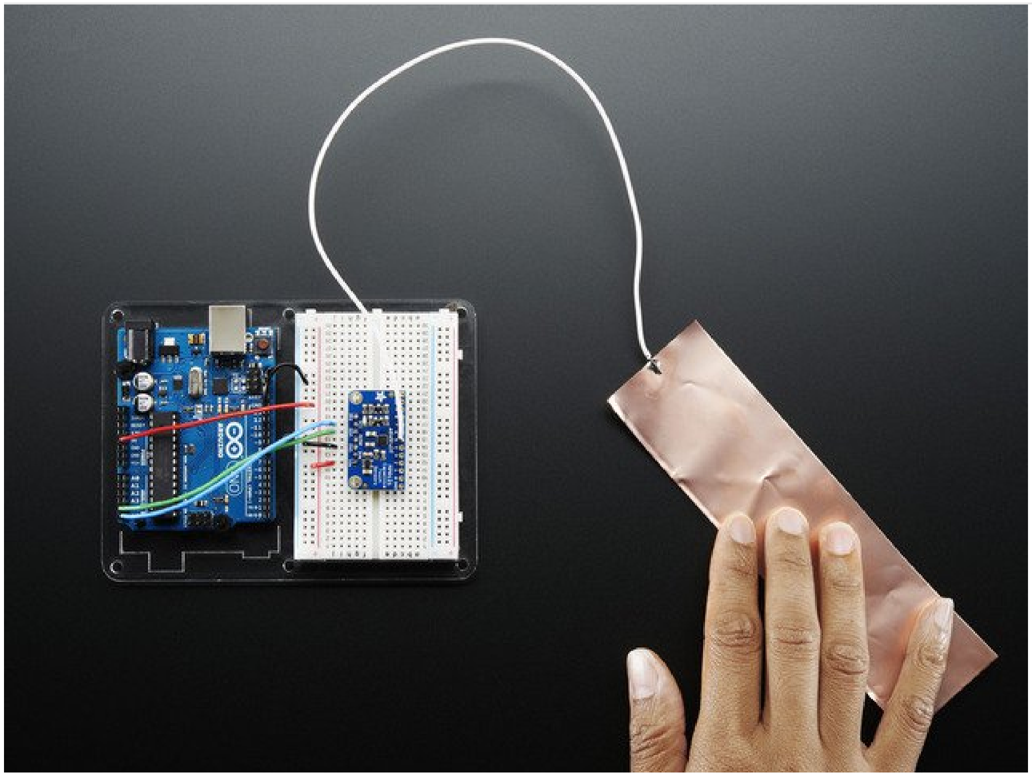
The first line with 'sudo python keyboard.py' is the background process that was launched earlier. You can kill this process by running:

```
sudo kill 2251
```

If you run the ps command above again you should now see the python keyboard.py processes have terminated.

That's all there is to using the MPR121 virtual keyboard on a Raspberry Pi. Have fun using the capacitive touch buttons to control your own games and programs!

Electrodes



Once you have the MPR121 breakout working you'll want to construct electrodes. These are large conductive piece of copper, foil, paint, etc that will act as the "thing you touch"

Remember that electrodes must be electrically conductive! We suggest copper foil tape, conductive fabrics, ITO, pyralux flex PCB, etc. We have tons of great conductive materials in our Materials category. Some can be soldered to, others can be clipped to with alligator chips. (https://adafru.it/dKI)

Remember, it doesn't have to be metal to be electrically conductive. Other things that work are tap or salt water, many kinda of food, even fruit!

We suggest soldering a wire to the electrode pad on the breakout and then soldering or clipping it to whatever you want your electrode to be.

The wires and electrodes themselves have a certain amount of 'inherent capacitcance'!

This means that whenever you attach an alligator clip, or a large piece of copper, or whatever your electrode is, the capacitive sense chip will detect it and may think you're touching it. What you have to do is *recalibrate* the sensor. The easiest way to do that is to restart the python sketch since calibration is done when the chip is initialized. So, basically...

connect all your wires, electrodes, fruit, etc...*then* start up the capacitive touch program!