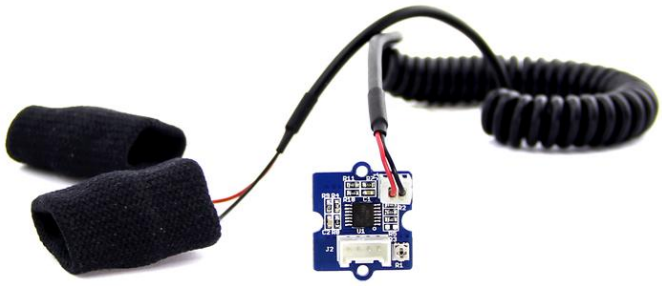
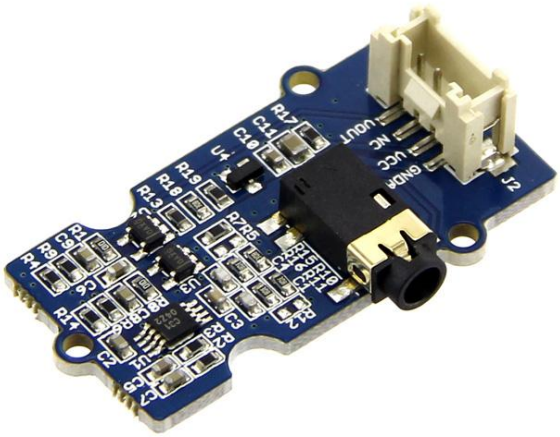
1.- Hardware

1.1- connect Galvanic skin response GSR sensor to WIO terminal



https://wiki.seeedstudio.com/Grove-GSR_Sensor/

1.2.- connect EMG sensor to WIO terminal via the Grove male jumper conversion Analog sensor port

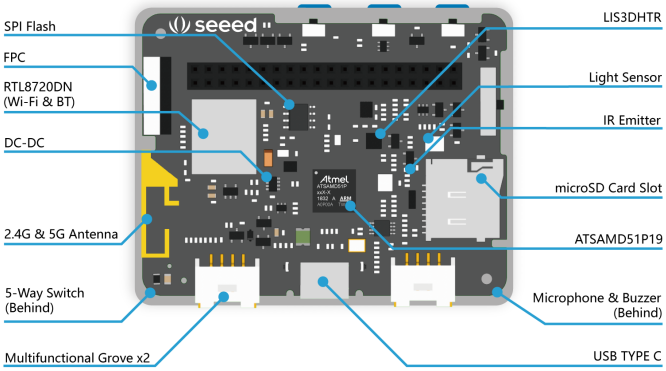


https://wiki.seeedstudio.com/Grove-EMG_Detector/

https://wiki.seeedstudio.com/Wio-Terminal-Getting-Started/

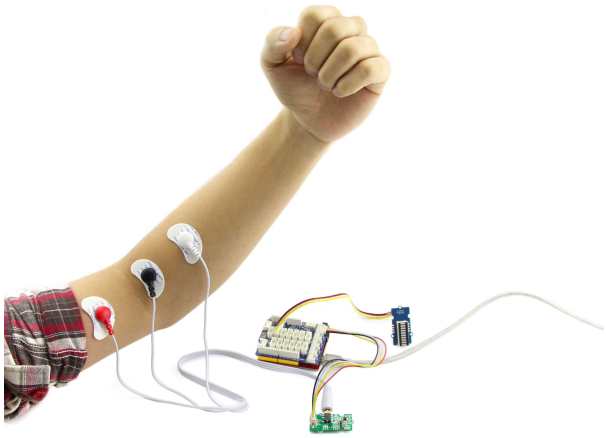
1.3 Wio Terminal :

```
//                          Grove - GSR sensor
// A0  -------------------------- Grove Connector
//                          Grove - EMG Detector
// A2  -------------------------- Grove Connector
```



In standby mode, the output voltage is 1.5V. When a detected muscle is active, the output signal rises, and the maximum voltage is 3.3V. You can use this sensor in a 3.3V or 5V system.

Finally, tack the three electrodes to your muscle, and keep a distance between each electrode.



2.- Software

2.1.- Arduino IDE 1.8.19

2.1.1.- Download the Arduino IDE to your computer from here:
https://www.arduino.cc/en/software

2.1.2.- Launch the Arduino application and connect the WIO Terminal to USB:

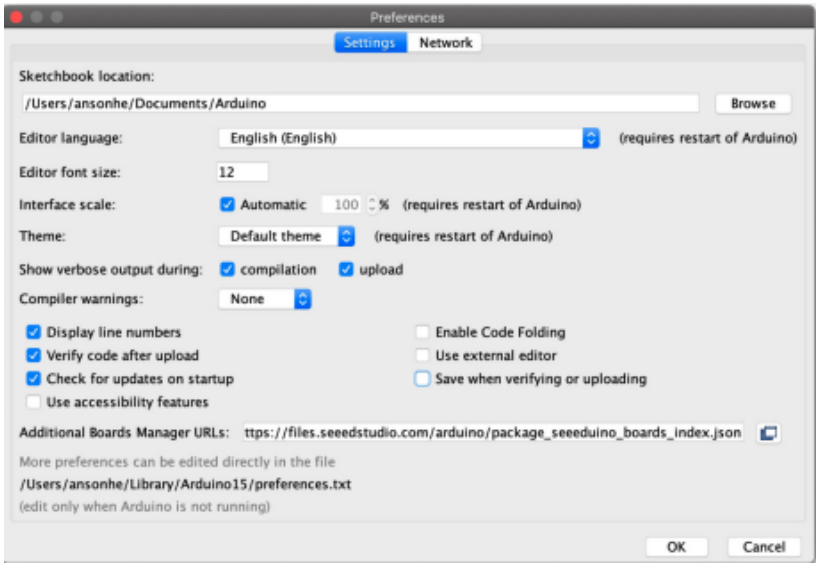
Double-click the Arduino IDE application you have previously downloaded.

2.2.- Add WIO Terminal Board Library

Go to Arduino File | Preferences

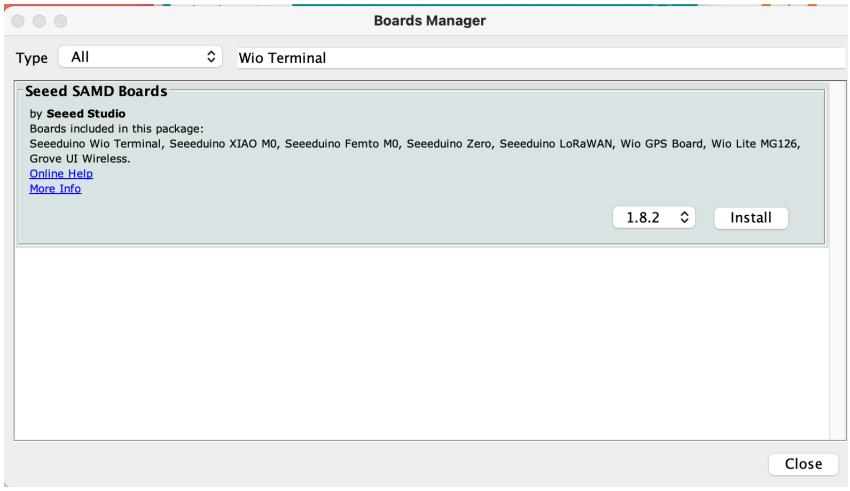
Add Additional Boards Manager URL:

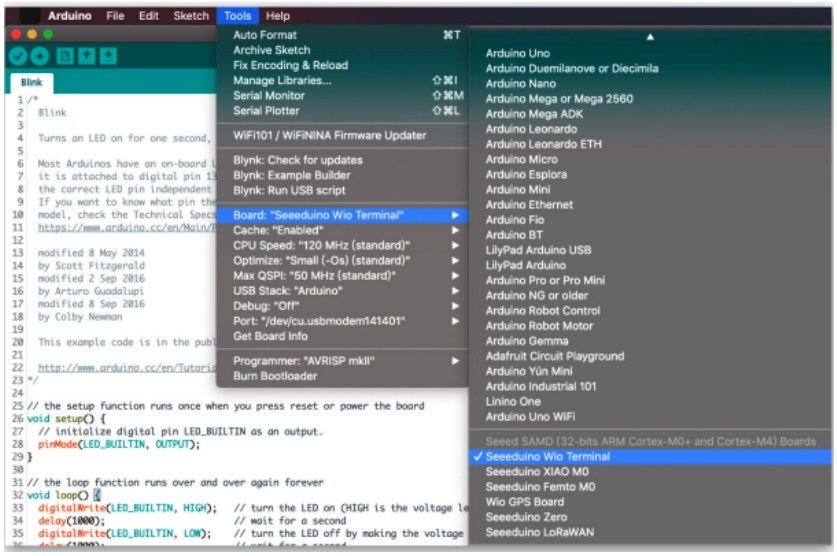
https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json



2.3.- Set up the Seeed SAMD Arduino Core

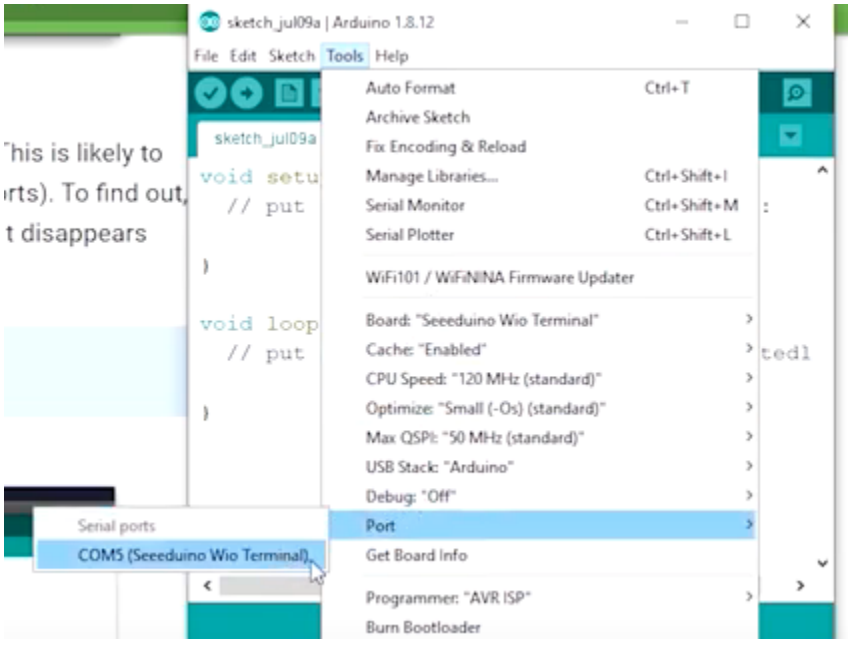
Open the Arduino IDE, click Tools ➡ Board ➡ Boards Manager, and search for Wio Terminal in the search box. Then, install Seeed SAMD Boards





2.4.- Select your board and port:

Select the Tools > Board menu entry that corresponds to your Arduino. Select the Wio Terminal



Note

For Mac Users, it will be something like /dev/cu.usbmodem141401

2.5.- Upload the program to WIO Terminal

Now, simply click the Upload button in the environment. Wait a few seconds and if the upload is successful, the message "Done uploading." will appear in the status bar.

2.6.- Installing the SD Card library for Wio Terminal

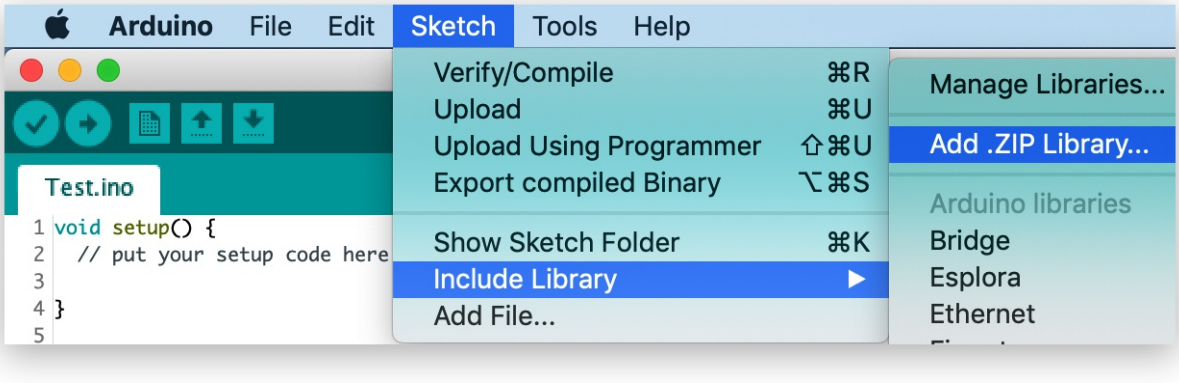
This repo introduces how to install the File System library used on Wio Terminal. It provides the basic functionality of File operating with the SD card, allowing to Read/Write in or from the SD card using the SPI interface.

Installing the File System Library

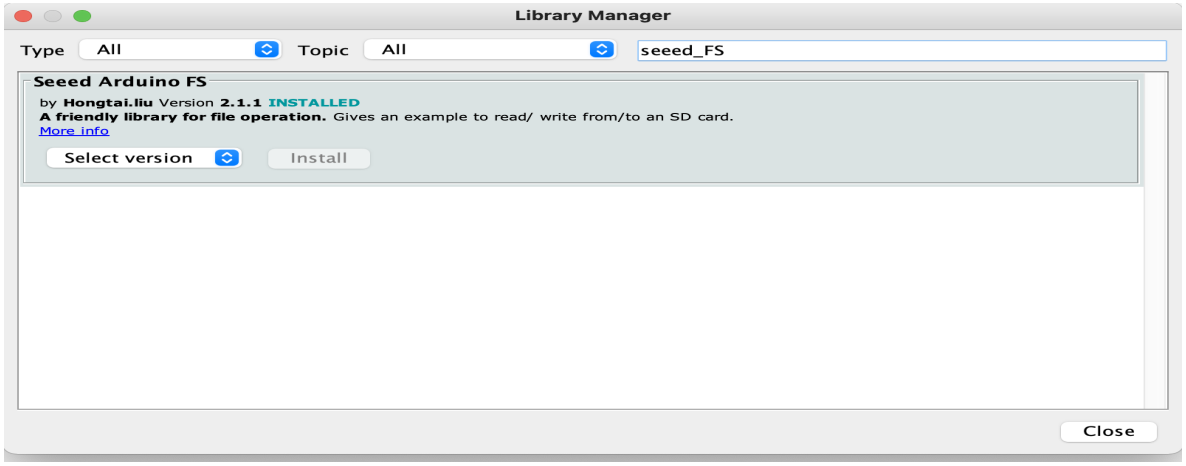
Download the entire repo Seeed_Arduino_FS:

https://github.com/Seeed-Studio/Seeed_Arduino_FS

Open the Arduino IDE, click sketch -> Include Library -> Add .ZIP Library, and choose the Seeed_Arduino_FS file that you have just downloaded.



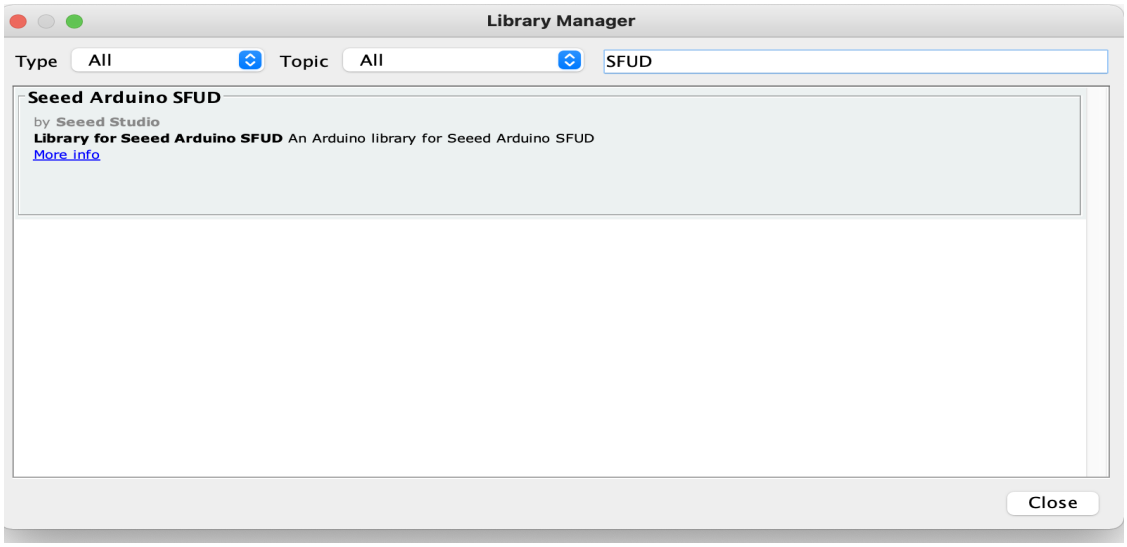
Check on sketch | Library Manager | Search for seeed_FS



2.7.- Install the Dependent SFUD Libraries

https://github.com/Seeed-Studio/Seeed_Arduino_SFUD

Check on sketch | Library Manager | Search for SFUD



2.8.- Install Seeed_Arduino_Linechart:

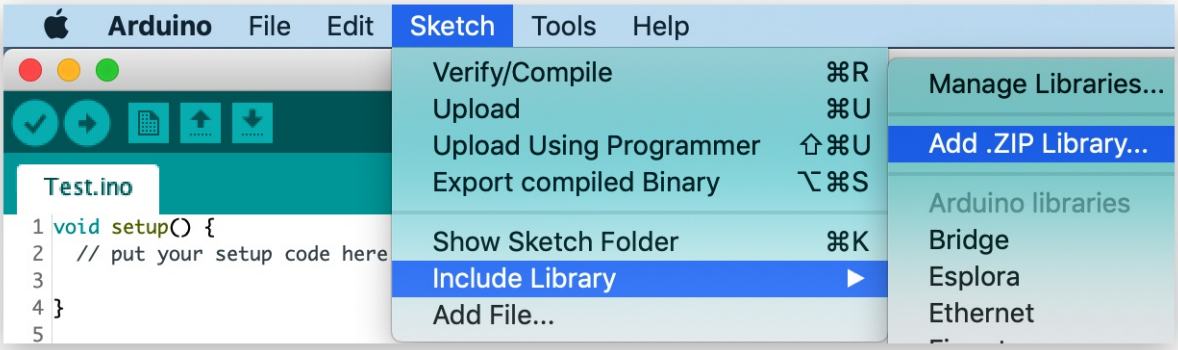
https://github.com/Seeed-Studio/Seeed_Arduino_Linechart

2.9.-Installing the TFT LCD Library Separately¶

Visit the Seeed_Arduino_LCD repositories and download the entire repo to your local drive.

https://github.com/Seeed-Studio/Seeed_Arduino_LCD

Now, the TFT LCD library can be installed to the Arduino IDE. Open the Arduino IDE, and click sketch -> Include Library -> Add .ZIP Library, and choose the Seeed_Arduino_LCD file that you've just downloaded.



3.- Install the project

3.1.- To display the converted BMP files on the TFT LCD screen, move them to the SD card.

3.2.- Then, copy the RawImage.h file to the sketches on the Arduino IDE.

https://files.seeedstudio.com/wiki/Wio-Terminal/res/RawImage.h

3.3.- Include the required libraries.

```
#include <SPI.h>
#include <Seeed_FS.h>
#include "TFT_eSPI.h"
#include "seeed_line_chart.h"
#include "SD/Seeed_SD.h"
#include "RawImage.h"
```

3.4.- Define the TFT screen and the sprite settings.

```
TFT_eSPI tft;

// Define the sprite settings:
#define max_size 50 // maximum size of data
doubles gsr_data, emg_data;
TFT_eSprite spr = TFT_eSprite(&tft);
```

3.5.- Initialize the File class and define the CSV file name.

```
File myFile;
const char* data_file = "mouse_fatigue_data_set.csv";
```

3.6.- Define the sensor voltage (signal) pins:

```
#define GSR A0
#define EMG A2
```

3.7.- Define the data holders:

```
int gsr_value, emg_value;
uint32_t background_color = tft.color565(31,32,32);
uint32_t text_color = tft.color565(174,255,205);
```

3.8.- setup:

```
void setup()
{

   Serial.begin(115200);
```

3.8.1.- Configurable Buttons:

```
 pinMode(WIO_KEY_A, INPUT_PULLUP);
 pinMode(WIO_KEY_B, INPUT_PULLUP);
 pinMode(WIO_KEY_C, INPUT_PULLUP);
```

3.8.2.- Check the connection status between Wio Terminal and the SD card:

```
 if(!SD.begin(SDCARD_SS_PIN, SDCARD_SPI)) while (1);
```

3.8.3.- Initiate the TFT screen:

```
 tft.begin();
 tft.setRotation(3);
 tft.fillScreen(background_color);
 tft.setTextColor(text_color);
 tft.setTextSize(2);
 // Create the sprite.
 spr.createSprite(TFT_HEIGHT / 2, TFT_WIDTH);
```

3.9.3.- Define and display the required 16-bit images saved on the SD card:

```
 drawImage<uint16_t>("data_collect.bmp", TFT_HEIGHT, 0);
 drawImage<uint16_t>("carpal_tunnel.bmp", TFT_HEIGHT/2, 0);
 drawImage<uint16_t>("mouse.bmp", TFT_HEIGHT/2, TFT_WIDTH-90);
}
```

3.10.- In the get_GSR_data function, calculate the average of the last ten GSR sensor measurements to remove the glitch.

Also, add a calibration value if necessary to rectify the generated results.

```
void get_GSR_data(int calibration)
{
  long sum = 0;
  // Calculate the average of the last ten GSR sensor measurements to remove the glitch.
  for(int i=0;i<10;i++){
    sum += analogRead(GSR);
    delay(5);
  }
  gsr_value = (sum / 10) - calibration;
  Serial.print("GSR Value => "); Serial.println(gsr_value);
}
```

3.11.- In the get_EMG_data function, evaluate the summation of the last 32 EMG sensor measurements.

Then, shift the summation by five with the right shift operator (>>) to obtain the correct EMG value.

```
void get_EMG_data(){
  long sum = 0;
  // Evaluate the summation of the last 32 EMG sensor measurements.
  for(int i=0;i<32;i++){
    sum += analogRead(EMG);
  }
  // Shift the summation by five with the right shift operator (>>) to obtain the EMG value.
  emg_value = sum >> 5;
  Serial.print("EMG Value => "); Serial.println(emg_value); Serial.println();
  delay(10);
}
```

3.12.- In the display_line_chart function, define the line graph title (header) settings and draw it on the sprite.

Then, define the line chart's customizable appearance settings and draw the line chart on the sprite.

```
void display_line_chart(int header_y, const char* header_title, int chart_width, int chart_height, doubles data, uint32_t graph_color, uint32_t line_color){
```

```
  // Define the line graph title settings:
  auto header =  text(0, header_y)
                .value(header_title)
                .align(center)
                .valign(vcenter)
                .width(chart_width)
                .color(tft.color565(243,208,296))
                .thickness(2);
  // Define the header height and draw the line graph title.
  header.height(header.font_height(&tft) * 2);
  header.draw();
  // Define the line chart settings:
  auto content = line_chart(0, header.height() + header_y);
  content
  .height(chart_height) // the actual height of the line chart
  .width(chart_width) // the actual width of the line chart
  .based_on(0.0) // the starting point of the y-axis must be float
  .show_circle(false) // drawing a circle at each point, default is on
  .value(data) // passing the given data array to the line graph
  .color(line_color) // setting the line color
  .x_role_color(graph_color) // setting the line graph color
  .x_tick_color(graph_color)
  .y_role_color(graph_color)
  .y_tick_color(graph_color)
  .draw();
}
```