

Public Speaking Biofeedback



Ivan Arjona Martínez

Índice

Índice	2
1. Introducción	3
Objetivo	3
Antecedentes	3
Proyecto actual	3
Materiales	5
2. Desarrollo del proyecto	6
GSR	6
HR	12
GSR + HR	15
Conexión inalámbrica	18
3. Conclusiones	20
4. Webgrafía	21

1. Introducción

Objetivo

La finalidad de este proyecto es construir un prototipo para monitorizar la fisiología y el comportamiento de una persona durante la puesta en escena de un discurso ante una audiencia. Se busca poder comprobar empíricamente la mejora ante el miedo escénico.

Antecedentes

La motivación de este proyecto viene dada de un prototipo previo realizado en 2016 en el que una persona puede enfrentarse a un público ficticio mediante una aplicación diseñada en realidad virtual (en adelante VR). En esta aplicación se visualiza un público previamente grabado en vídeo estereoscópico en 180° integrado en un entorno 360° para simular una audiencia atenta al discurso que realizará el emisor desde el escenario principal de un teatro.

Proyecto actual

En el proyecto actual pretendemos prototipar un dispositivo que, vinculado con el proyecto de realidad virtual, nos ayude a monitorizar el estrés y la comunicación no verbal del locutor. Este proyecto estará dividido en dos bloques y cada uno de ellos en varias fases.

Bloque 1

Creación del prototipo de un wearable que detecte el nivel de estrés del participante que se dividirá en las siguientes fases:

Fase 1.a

Montaje de un circuito electrónico que aúne los sensores de ritmo cardíaco (en adelante HR) y respuesta galvánica de la piel (en adelante GSR) también llamada actividad electrodérmica o conductancia de la piel. Los sensores irán conectados directamente a un ESP32 para obtener las mediciones de ambos sensores.

Fase 1.b

Integración a través del protocolo Matter del microcontrolador ESP32 o ESP8266 y una batería recargable conectado por wifi a una Raspberry Pi. Esto dotará al medidor de estrés de autonomía y conectividad inalámbrica.

Fase 1.c

Analizar los datos recibidos (*raw data*) y, mediante un algoritmo, interpretar el nivel de estrés del participante. Graficarlos para poder tener una mayor comprensión de la información. También, en un futuro, poder emparejar dichas gráficas con la imagen que tiene el participante en su visor de VR y,

de esta manera, así poder averiguar cuáles son las situaciones que funcionan como *trigger* ante aumentos bruscos de estrés.

Bloque 2

Creación del prototipo de un dispositivo de reconocimiento de audio e imagen para analizar la comunicación no verbal del participante.

Fase 2.a

Montaje de un circuito electrónico con un micrófono que registre el audio del discurso del participante y analice el tono de su voz. Con esto se pretende interpretar mediante un algoritmo las emociones vividas por el participante durante la experiencia.

Fase 2.b

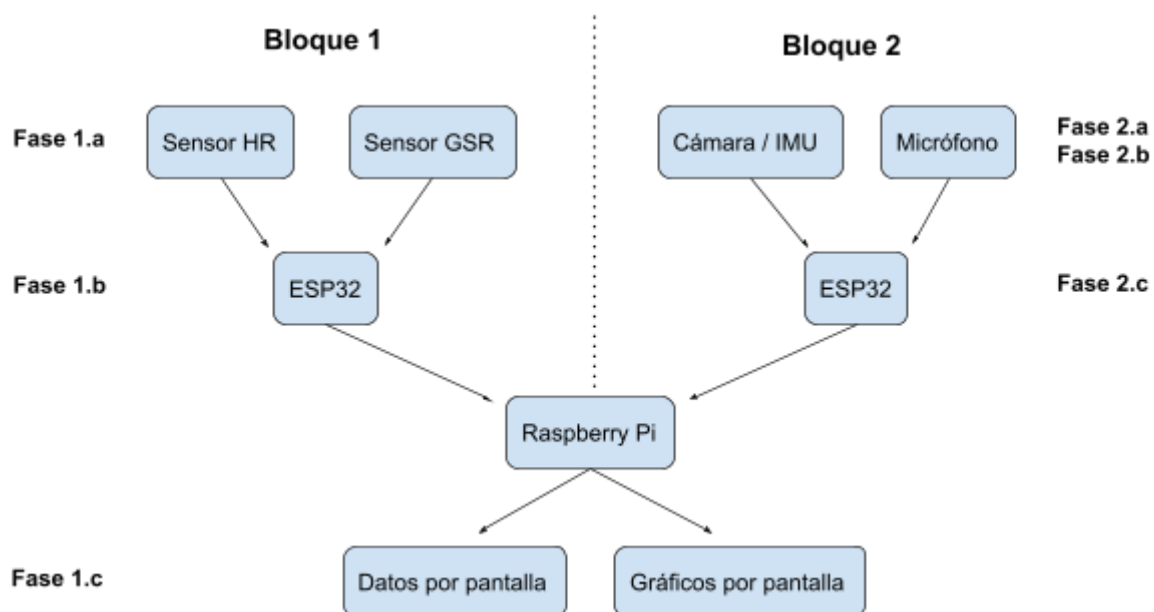
Montaje de un circuito electrónico con una cámara que registre la imagen de los movimientos del participante y analice su gesticulación. Las imágenes no ayudarán a interpretar si está llevando a cabo una buena comunicación no verbal corporal. También es planteable realizarlo con diferentes sensores de unidad de movimiento inercial (IMU) ubicados en las distintas articulaciones del cuerpo.

Fase 2.c

Integración a través del protocolo Matter de un microcontrolador ESP32 o ESP8266 y una batería recargable conectado por wifi a una Raspberry Pi. Esto dotará al asistente de comunicación no verbal de autonomía y conectividad inalámbrica.

Nuestra intención es llegar con solvencia y en pleno funcionamiento a la Fase 1.a e idealmente a la Fase 1.c completando así el Bloque 1.

Dejamos reseñado como posibles exploraciones a futuro el desarrollo del Bloque 2 para la realización del proyecto completo.



Materiales

- Raspberry Pi 3 Model B
- Sensor de ritmo cardíaco (HR): PulseSensor (¿hw827?)
- Sensor de respuesta galvánica de la piel (GSR): Seeedstudio Grove GSR sensor v1.2
- 2 x Microcontrolador ESP32
- 2 x Batería recargable
- Micrófono
- Cámara

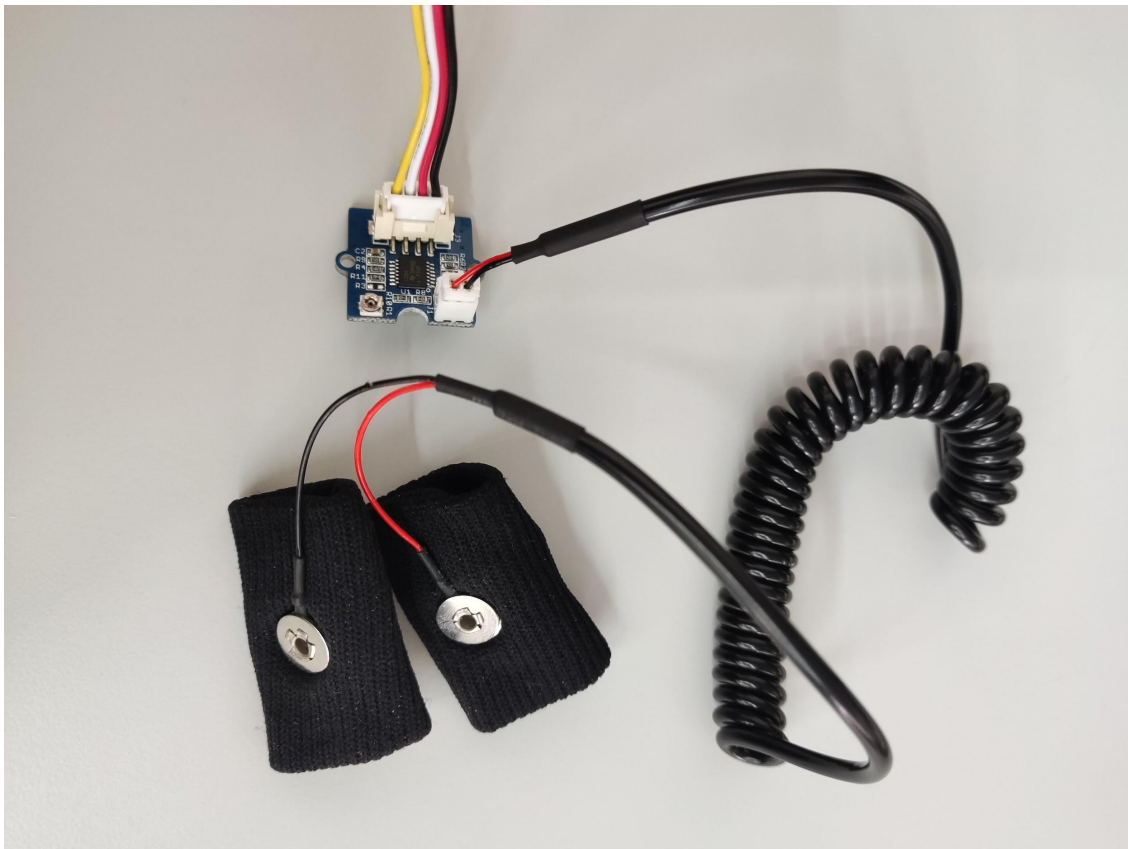
2. Desarrollo del proyecto

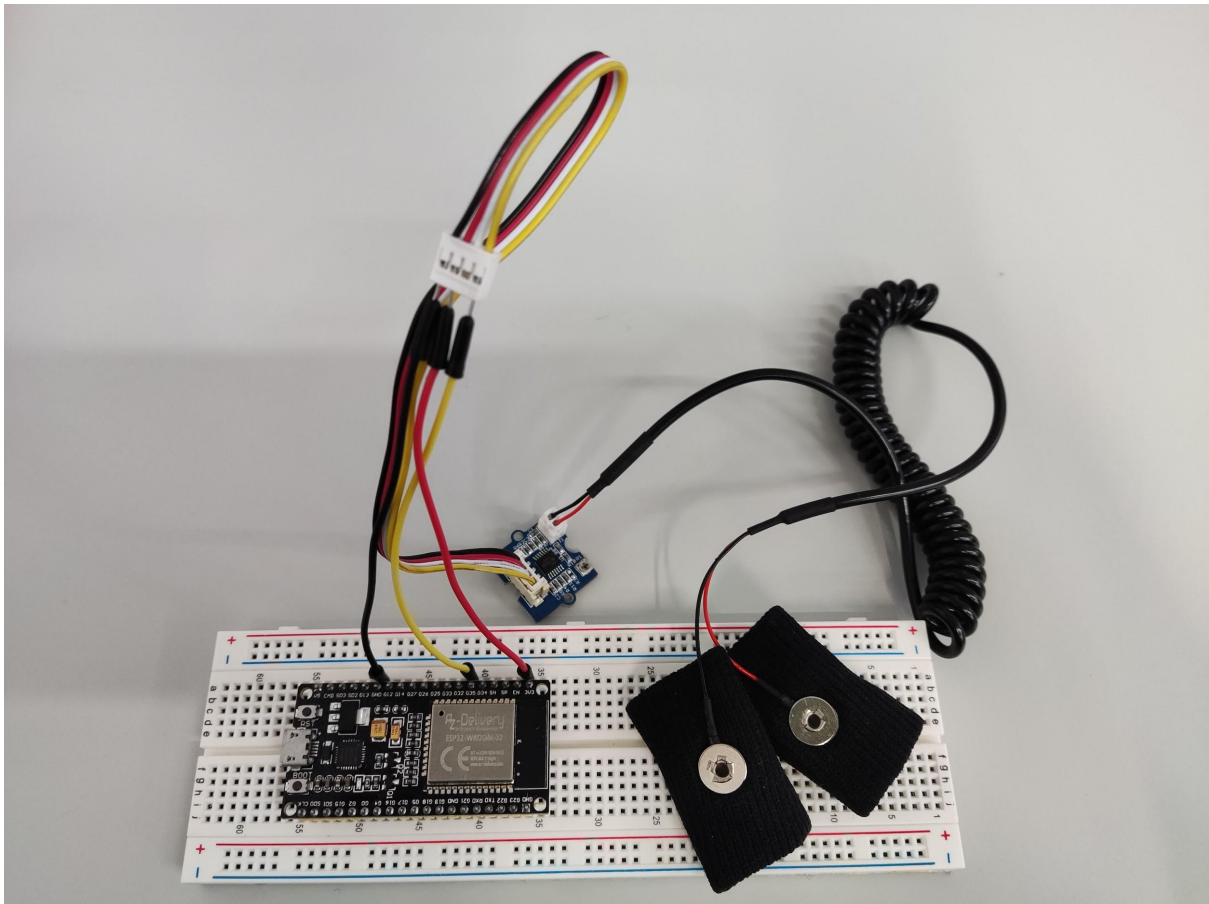
Iniciamos documentándonos para saber qué características tienen cada uno de los sensores de la Fase 1 y qué investigaciones científicas hay al respecto de la medición de la fisiología y la interpretación de los datos para comprender cómo debería funcionar el sistema. Dejo referenciadas las fuentes consultadas en la webgrafía, apartado Papers.

GSR

Conexionado del GSR

Pasamos a construir el primer circuito con el ESP32 y el sensor GSR. En el sensor de GSR podemos encontrar los dos electrodos que debemos vincular al conector J1 de la PCB. El conector J3 lo reservamos para los cables que irán a nuestra placa de desarrollo ESP32.



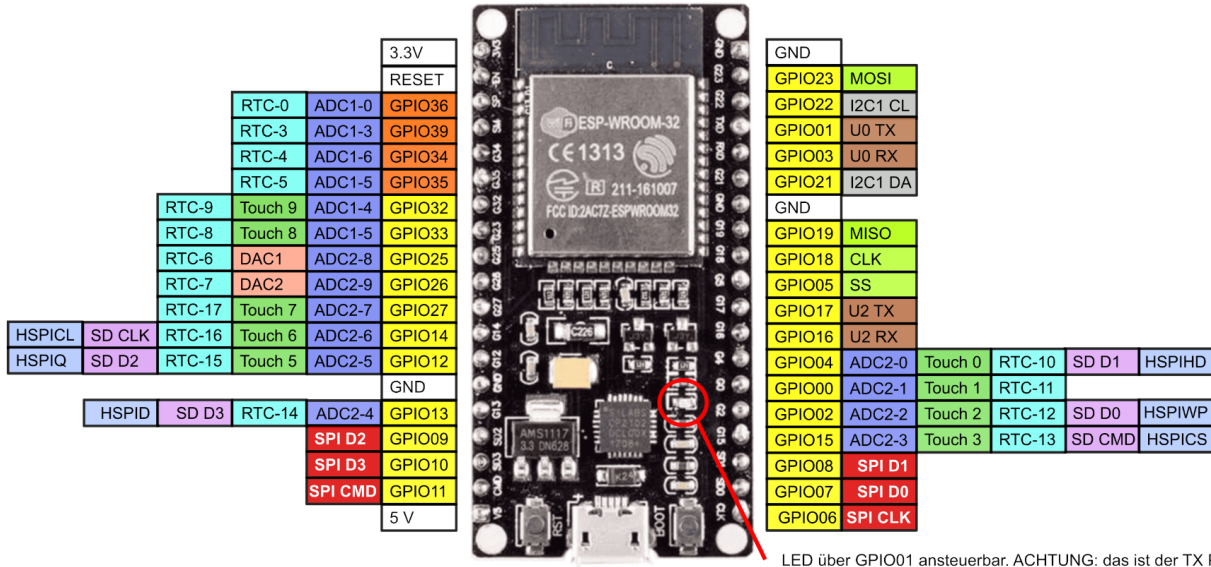


El código de colores que seguiremos es:









- Negro: Ground
- Rojo: Vcc (puede ser alimentado tanto a 3.3V como 5V)
- Blanco: En el caso de este sensor no es necesario conectar este cable.
- Amarillo: Data. Pin analógico del ESP32. En nuestro caso hemos utilizado el 35.

Buscamos también el pinout de la placa de desarrollo para verificar que el pin al que conectamos el sensor es de tipo analógico.





ESP32 NodeMCU Anschlussbelegung



LED über GPIO01 ansteuerbar. ACHTUNG: das ist der TX Pin der seriellen Schnittstelle.

- | | |
|---|---|
|  | Digital Ein- und Ausgänge (alle unterstützen PWM) |
|  | Digital Eingänge |
|  | Analog Eingänge 12-bit, 0 bis 3.3 V |
|  | Analog Ausgänge 8-bit, 0 bis 3.3 V |
|  | Kapazitive Berührungstasten |
|  | IO-Pins vom RTC ultra low power Processor, im Tiefschlafmodus benutzbar |
|  | SD-Karten Schnittstelle |
|  | SPI-Bus für Flash-Speicher, kann nicht genutzt werden |

Die folgenden Pins zeigen die Default-Belegung. Alle diese Signale können auf einen beliebigen Ein- Ausgabe Pin gelegt werden.
Das gilt auch für UART0 und UART1 die in der Default belegung nicht zugreifbar sind

-  I2C Bus (Wire)
-  VSPI Bus
-  Serielle Schnittstelle
-  HSPI Bus

IDE Arduino

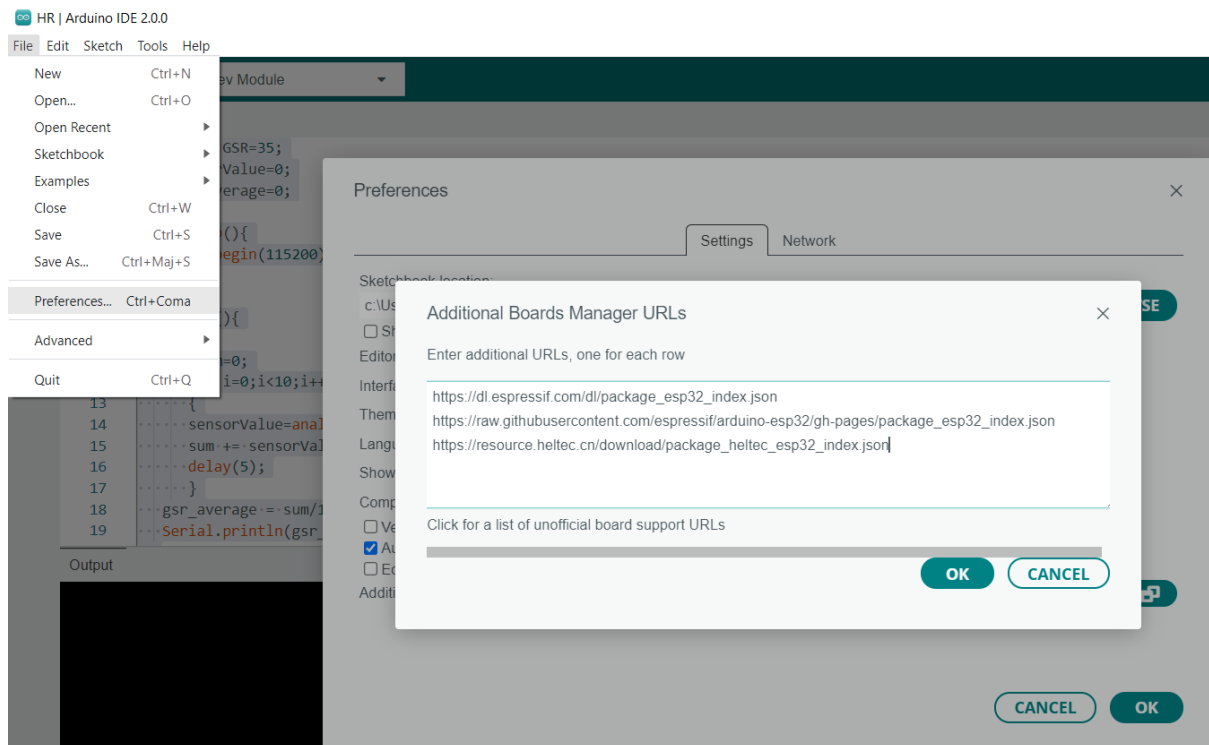
Para escribir y cargar el software al microcontrolador usaremos el IDE (entorno de desarrollo integrado) de Arduino. Para poder utilizarlo con ESP32 y con nuestro sensor debemos instalar algunas librerías.

Por un lado, para añadir la librería de ESP32, en el menú vamos a File/Preferences dentro del cuadro de diálogo, en la pestaña Settings, en la parte inferior añadimos en “Additional boards manager URLs:” los siguientes enlaces que contemplan amplia variedad de fabricantes de placas:

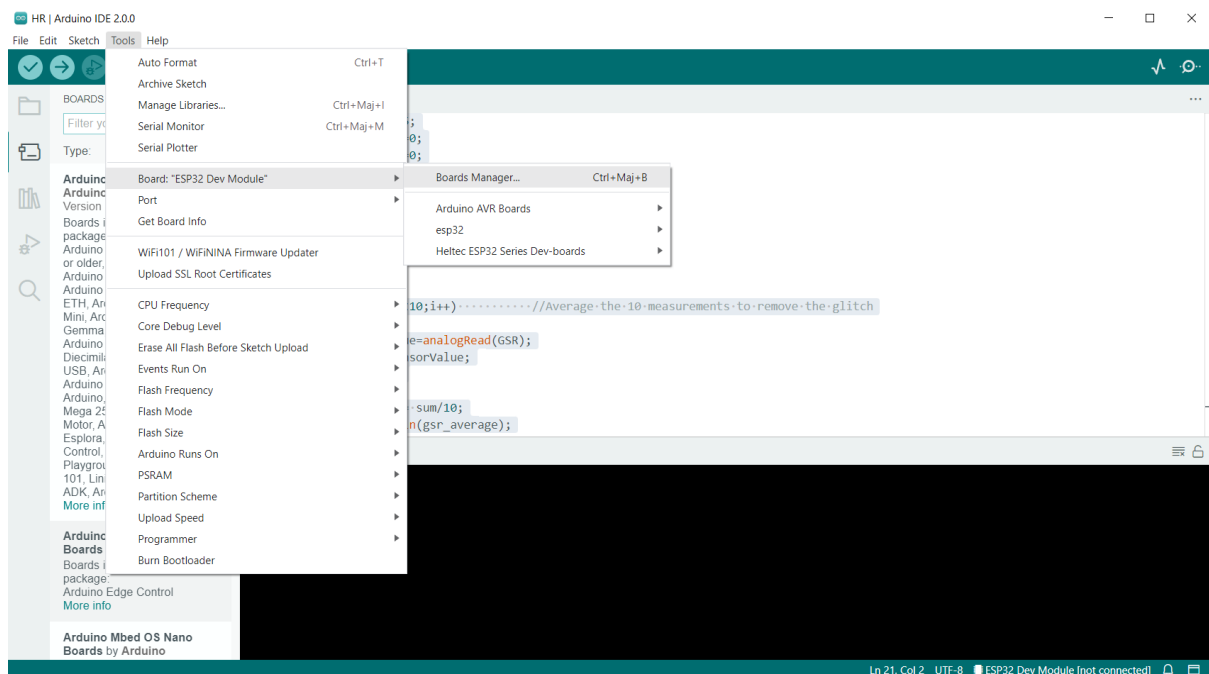
https://dl.espressif.com/dl/package_esp32_index.json

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

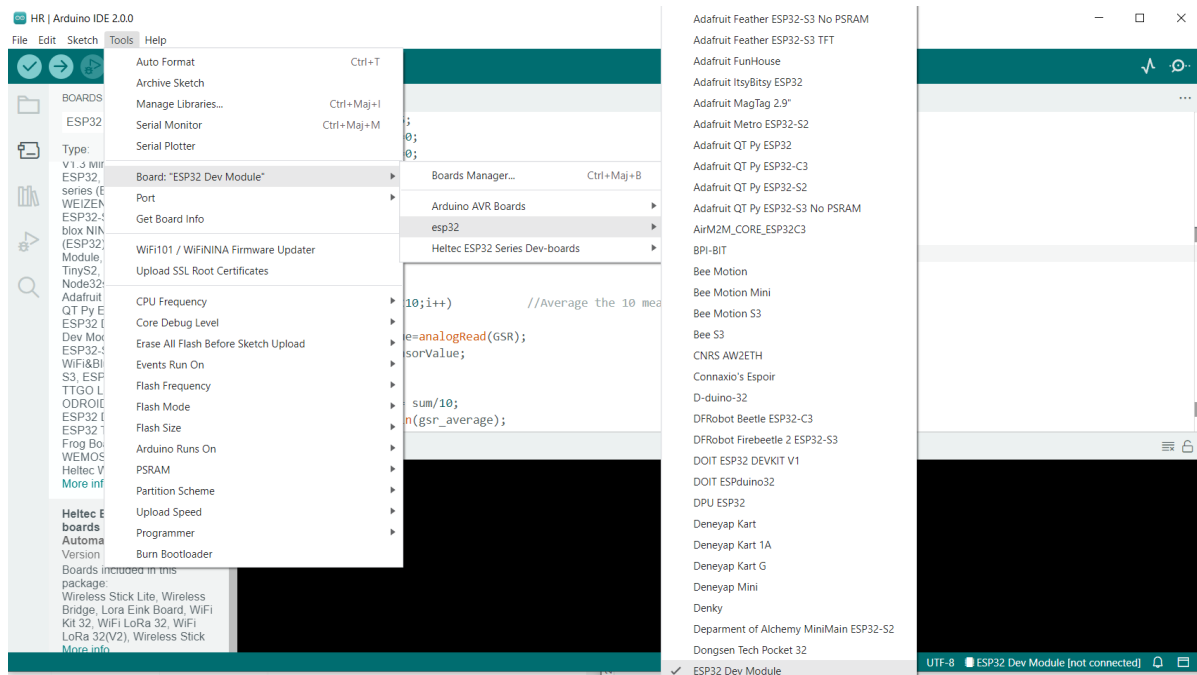
https://resource.heltec.cn/download/package_heltec_esp32_index.json



A continuación vamos a Tools/Board:XXXX/Boards Manager... y en el menú lateral izquierdo que aparece escribimos en el cuadro de búsqueda ESP32 e instalamos tanto “esp32 by Espressif Systems” como “Heltec ESP32 Series Dev-boards by Heltec Automation(TM)”.



Ahora ya podemos seleccionar nuestra placa de desarrollo en el IDE de Arduino yendo a Tools/Board/esp32/ y eligiendo el modelo de ESP que tengamos. En nuestro caso elegiremos ESP Dev Module.



Aún siguiendo todos los pasos, no encontramos con el problema de que no conseguía encontrar el ESP32. Tras leer un poco sobre el asunto, vimos que el componente de microUSB que se monta en algunas placas de desarrollo dan problemas de instalación de forma automática con plug&play. Para solucionarlo tuvimos que averiguar de qué componente exactamente se trataba en buscándolo en Configuración del Sistema y buscando e instalando los Drivers del fabricante del componente: Silicon Labs CP210x USB to UART Bridge.

Código

Para el software primero probamos con códigos y librerías ya existentes en C++ creadas para Arduino y las adaptamos para el ESP32.

```
const int GSR=35;
int sensorValue=0;
int gsr_average=0;

void setup() {
  Serial.begin(115200);
}

void loop() {

  long sum=0;
  for(int i=0;i<10;i++)          //Average the 10 measurements to
  remove the glitch
  {
```

```
    sensorValue=analogRead(GSR) ;  
    sum += sensorValue;  
    delay(5);  
}  
gsr_average = sum/10;  
Serial.println(gsr_average);  
}
```

Creamos e inicializamos la variable GSR a 35 para indicarle el Pin en el que tenemos conectado la señal de datos del sensor.

Hacemos lo mismo con sensorValue dónde recogeremos el valor leído por el sensor y con gsr_average dónde iremos recogiendo el valor medio.

Debemos tener en cuenta según el ejemplo de fabricante que la comunicación con ESP32 será a 115200 en vez de 9600 como suele estar definido para la comunicación serie con Arduino y así se lo especificamos en la línea 5 de nuestro código con la invocación del método `Serial.begin(115200)`; Lo especificaremos de igual manera en el Serial plotter para obtener los datos de salida.

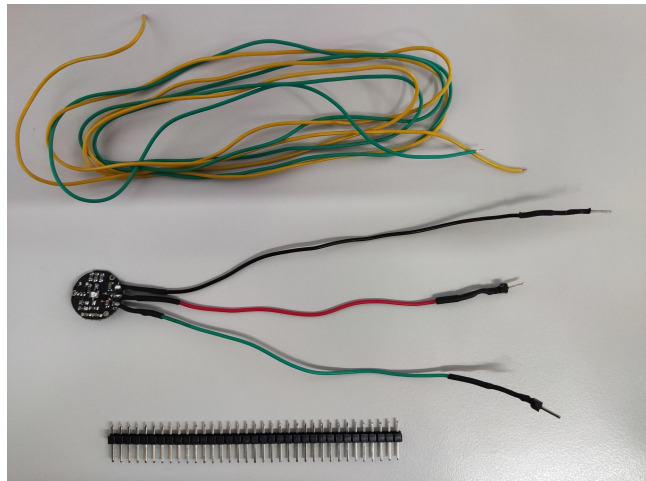
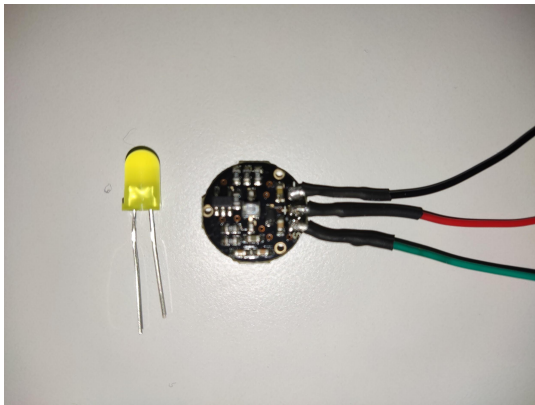
Para obtener unos datos interpretables, debemos calibrar primero el valor que recibimos de los sensores sin estar conectados al sujeto a 512 tal como sugiere el fabricante en su documentación.

Nos resulta muy difícil conseguir el dato de 512 exacto de salida y asignamos uno parecido. Una vez colocado en el sujeto, podemos observar la variabilidad en situaciones distintas de reposo y de agitación (forzando la hiperventilación).

HR

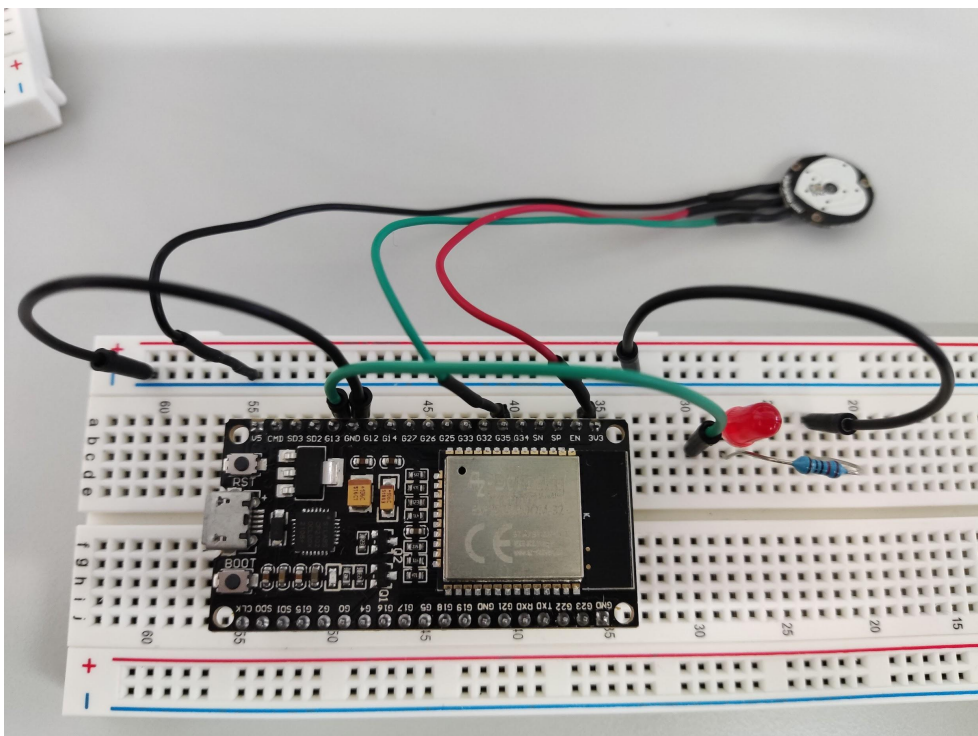
Conexionado del HR

Pasamos a construir el segundo circuito de forma independiente. Nos encontramos con un inconveniente para ello. El sensor viene sin cables para su conexionado lo cuál nos obliga a soldar pines y cables al sensor de minúsculo tamaño. Aprovechamos para protegerlo con fundas termoretráctiles para evitar que se desuelde o se rompa con la manipulación.



Una vez lo tenemos, utilizamos de izquierda a derecha:

- Verde: Marcado con S. Data. Pin analógico del ESP32. En nuestro caso hemos utilizado el 35.
- Rojo: Marcado con el símbolo +. Vcc (puede ser alimentado tanto a 3.3V como 5V)
- Negro: Marcado con el símbolo - . Ground



Código

Para este segundo circuito probamos también con códigos y librerías ya existentes en C++ creadas para Arduino y las adaptamos para el ESP32.

En este caso, tal como ocurría en el GSR también debemos calibrar el sensor. Esto es muy importante ya que uno de los inconvenientes que nos encontramos es que no conseguíamos lecturas comprensibles porque nos habíamos saltado el paso previo de calibración.

Para este sensor, el fabricante nos proporciona, entre otros, un par de códigos para su puesta en marcha. El primero, como decíamos, para calibrar y el segundo para recibir valores. Veamos el primer código:

```
// Variables
int PulseSensorPurplePin = 35; // Pulse Sensor PURPLE WIRE connected
to ANALOG PIN 0
int LED13 = 13; // The on-board Arduion LED
int Signal; // holds the incoming raw data. Signal value can
range from 0-1024
int Threshold = 349; // Determine which Signal to "count as a beat",
and which to ignore.

// The SetUp Function:
void setup() {
    pinMode(LED13,OUTPUT); // pin that will blink to your
heartbeat!
    Serial.begin(9600); // Set's up Serial Communication at
certain speed.
}

// The Main Loop Function
void loop() {
    analogReadResolution(10);
    Signal = analogRead(PulseSensorPurplePin); // Read the
PulseSensor's value.
// Assign this value to
the "Signal" variable.

    Serial.println(Signal); // Send the Signal
value to Serial Plotter.

    if(Signal > Threshold){ // If the signal
```

```
is above "550", then "turn-on" Arduino's on-Board LED.  
    digitalWrite(LED13,HIGH);  
  } else {  
    digitalWrite(LED13,LOW);          // Else, the signal  
must be below "550", so "turn-off" this LED.  
  }  
  delay(10);  
}
```

Algunas cosas que debemos tener en cuenta en este código es que la inicialización de los pines es en el número 35 para el sensor y el 13 para el led. Este led parpadea cada vez que el ESP32 recibe un pulso de nuestro ritmo cardíaco en tiempo real.

De la misma manera que nos pasaba en el otro sensor, para obtener unos datos interpretables, debemos calibrar primero el valor que recibimos de los sensores sin estar conectados al sujeto en este caso no tenemos un valor de referencia así que tenemos que ir variando de 5 en 5 el valor de la variable Threshold hasta conseguir que el led no permanezca constantemente encendido ni apagado y que vaya siguiendo el parpadeo correctamente.

Al principio no logramos que arrojara ningún dato para lo cuál fue necesario pedir nuevos sensores, volver a soldarlos y ver si era el sensor el que fallaba. Al ver que no conseguimos que ninguno de los tres sensores de HR, iniciamos un periodo de investigación y averiguaciones en librerías de Arduino adaptadas para ESP32 y siguiendo el rastro de la pista de la frecuencia de lectura conseguimos ver que el sampling de Arduino es de 0 a 1024 y el de ESP32 es de 0 a 4095. Finalmente encontramos como indicarle al programa el cambio de frecuencia con la primera línea de código de la función void loop():

Signal = analogReadResolution(10)

Donde el valor 8 es para 0-1024 y el valor 10 es para resolución de 0-4095.

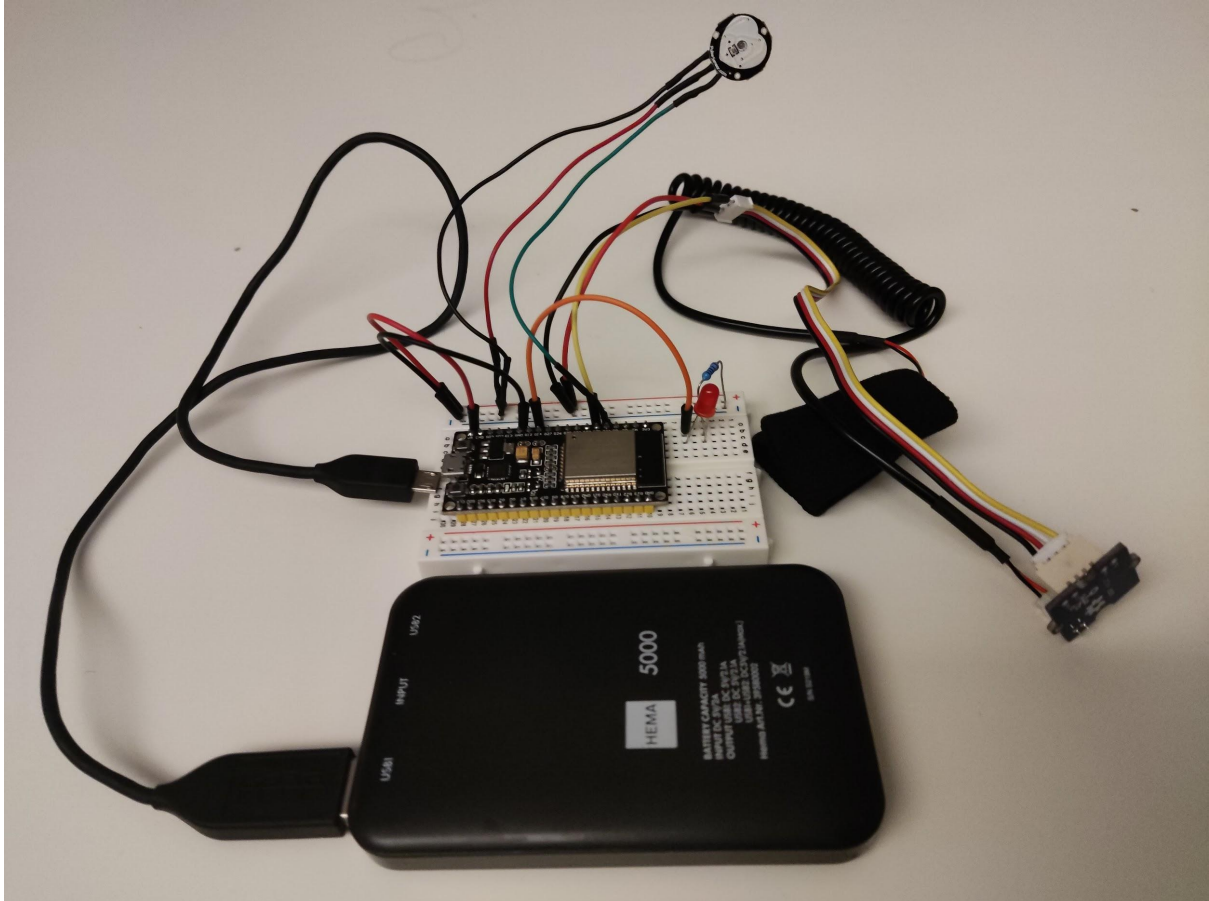
Una vez configurada la resolución correcta, los valores empiezan a cobrar sentido y la representación de los latidos a través del led también. También comprobamos su variabilidad en situaciones de reposo y agitación (hiperventilación)

Gracias a este inconveniente, añadiendo la misma instrucción en el código anterior de GSR conseguimos que el calibrado y variabilidad de los datos del primer sensor también fueran mucho más estables y cobraran sentido.

Los datos de calibración para la variable Threshold para un sensor fue 349 y para el segundo sensor de 514. Nótese la diferencia e importancia de la calibración inicial según el sensor y la persona que está siendo monitorizada.

GSR + HR

Conexionado de ambos sensores GSR + HR



Como vemos en la imagen superior, hemos simplificado el circuito uniéndolo todo en una protoboard más pequeña y manejable. Hemos unificado Grounds y Vcc's (5v) y hemos reubicado los pines de I/O de la siguiente forma:

- Led: Pin 14
- HR: Pin 34
- GSR: Pin 35

También hemos buscado una alternativa al transformador de corriente de pared o puerto USB de PC por una batería (power bank) de 5000 mAh con salida de 5V para alimentar el ESP32 con un tamaño similar al de la protoboard. Para poder dotarla de independencia una vez se consiga la comunicación inalámbrica del ESP32 con la Raspberry Pi.

Código

```
// Variables
int LED = 13;    // The on-board Arduion LED
int HR = 34;     // Pulse Sensor PURPLE WIRE connected to ANALOG
PIN 0
```



```

const int GSR = 35;

int SignalHR=0;           // holds the incoming raw data. Signal
value can range from 0-1024
int ThresholdHR = 721;    // Determine which Signal to "count
as a beat", and which to ignore.
int SignalGSR=0;
int AverageGSR=0;
int Beats=0;
int Counter=0;
int BPM=0;
int Conductancia=0;

// The SetUp Function:
void setup()
{
    pinMode(LED,OUTPUT);    // pin that will blink to your
heartbeat!
    Serial.begin(9600);    // Set's up Serial Communication at
certain speed.
}

// The Main Loop Function
void loop()
{
    analogReadResolution(10); // Change the resolution of th analog
read from 0-1024 to 0-4095
    SignalHR = analogRead(HR); // Read the PulseSensor's value.
    // Assign this value to the "Signal" variable.

    long sum;
    for (int i=0;i<10;i++)
    {
        //analogReadResolution(10); // Change the resolution of th analog
read from 0-4095 to 0-1024
        SignalGSR = analogRead(GSR); // Read the PulseSensor's value.
        sum += SignalGSR;
        delay (5);
    }
    AverageGSR = sum/10;

    Serial.println(SignalHR);           // Send the Signal

```



```

value to Serial Plotter.
    Serial.println(AverageGSR);           // Send the Signal
value to Serial Plotter.

    if(SignalHR > ThresholdHR)
    {                                     // If the signal is above "550", then
"turn-on" Arduino's on-Board LED.
        digitalWrite(LED,HIGH);
        Beats = Beats+1;
    } else {
        digitalWrite(LED,LOW);           // Else, the sigal must
be below "550", so "turn-off" this LED.
    }

    Counter=Counter+1;

    if (Counter == 18)
    {
        BPM = Beats*10;
        Serial.println("Tu ritmo cardiaco es de: ");
        Serial.println(BPM);
        //Conductancia=1024-AverageGSR;
        //Serial.println("Tu resistividad es de: ");
// Send the Signal value to Serial Plotter.
        //Serial.println(AverageGSR);           // Send the
Signal value to Serial Plotter.
        Beats=0;
        Counter=0;
    }

    delay(328);
}

```

En el código que tenemos encima, además de implementar los dos sensores en un mismo programa, hemos añadido algunas variables como Beats, BPM, Counter, para poder, no solo obtener los datos del sensor de ritmo cardíaco sino también las pulsaciones por minuto registrando en el bucle el tiempo que pasa según la suma de milisegundos de los delays y calculando la cantidad de beats registrados en 6 segundos que, multiplicados por 10 nos darían los beat por minuto.

Conexión inalámbrica

En este punto hemos avanzado en dos sistemas de forma paralela con tal de intentar conectar el ESP32. Por un lado mediante el protocolo de comunicación MQTT en el cuál se ha instalado el broker en la raspberry pi para que haga gestione y reciba la información que le pase el ESP32 de los dos sensores que hasta ahora hemos estado leyendo por el Serial plotter de Arduino IDE. Queda pendiente asignar las identificaciones y añadir al programa que haga la publicación. En este punto recomendamos visitar el proyecto del compañero Carles Barrachina para entrar en más detalle en el funcionamiento de MQTT y poder finalizar la comunicación.

Conexión con MQTT

- Instalar broker MQTT en la raspberry:
https://www.industrialshields.com/es_ES/blog/raspberry-pi-para-la-industria-26/post/como-instalar-mosquitto-el-mqtt-broker-en-raspberry-plc-338
Para averiguar la versión de raspberry escribimos:
lsb_release -a
sudo wget <http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key>
cd /etc/apt/sources.list.d/
sudo wget <http://repo.mosquitto.org/debian/mosquitto-bullseye.list>
sudo apt update
sudo apt install mosquitto mosquitto-clients
Probamos:
Escribimos en cliente: mosquitto_sub -d -h localhost -p 1883 -t "prueba"
Escribimos en broker: mosquitto_pub -d -h localhost -p 1883 -t "prueba" -m "Hello"
- Importar librería de python para MQTT: import paho.mqtt
- Abrir conexión como Publicador (pendiente de implementar)
- Abrir conexión como Suscriptor (pendiente de implementar)
- Publicar (pendiente de implementar)

Conexión con Matter

Por otro lado, hemos querido innovar y adentrarnos en el nuevo protocolo de comunicación llamado Matter (durante su creación, llamado CHIP) del cuál se ha publicado la primera versión open source de su SDK a día 4 de octubre de 2022, es decir, dos semanas antes del inicio de este proyecto. En este nuevo protocolo están implicadas las empresas más importantes del sector del IoT como Google, Amazon, Apple, Samsung, Huawei, Ikea, entre otras. La idea detrás de esto es crear una alianza de conectividad estándar para eliminar las barreras y facilitar la interoperabilidad de cualquier dispositivo entre todos los fabricantes. Entre las empresas implicadas en su desarrollo se encuentra Espressif, el fabricante que está detrás de los famosos MCU ESP32 y ESP8266 y dado que en este proyecto nos hemos centrado en el uso del ESP32 nos ha motivado este nuevo protocolo para intentar adaptarlo al futuro del IoT. Empezamos:

Para ello necesitamos cambiar de sistema operativo para seguir las indicaciones de la documentación oficial e ir sobre seguro.

Tenemos que instalar en una tarjeta SD nueva Ubuntu server 22.04 con Raspberry imager
Una vez instalado configuramos el archivo de red:

```
sudo nano /etc/netplan/50-cloud-init.yaml
```

En este punto nos encontramos con el problema de que, aún siguiendo la documentación de Ubuntu o ejemplos ya publicados por internet, la configuración no funcionaba. Para atajar el problema, primero probamos de separar la configuración del router y hacerla manual:

```
sudo ip route add default via 10.199.160.254
```

Configuramos también las DNS en otro archivo:

```
sudo nano /etc/resolv.conf  
nameserver 8.8.8.8  
nameserver 8.8.4.4
```

Finalmente, encontramos una nueva solución para configurar el router, incluimos la siguientes líneas en el archivo 50-cloud-init.yaml

o bien:

```
gateway4: 10.199.160.254 (en el que nos lanza el warning “deprecated”)
```

o:

```
routes:
```

```
- to: 0.0.0.0/0
```

```
via:10.199.160.254
```

Mencionar que en el apartado “- to:” intentamos poner “default” y “0/0” tal como mencionan en muchos ejemplos y en la documentación pero no conseguimos hacerlo funcionar.

Configuración final:

```
network:  
  wifis:  
    wlan0:  
      dhcp4: no  
      addresses: [10.199.160.155/24]  
      nameservers:  
        addresses: [8.8.8.8]  
      access-points:  
        "IOT2022":  
          password: "arduinouno"  
      routes:  
        - to: 0.0.0.0/0  
          via: 10.199.160.254
```

Al actualizar los repositorios nos encontramos con otro problema:

Hacemos sudo apt update del sistema y da error. Encontramos una solución que funciona. Escribimos:

```
echo “nameserver 8.8.8.8” | sudo tee /etc/resolv.conf > /dev/null
```

Seguimos con las instrucciones de github con la instalación de prerequisites en Linux y, después, un poco más abajo, la instalación de prerequisites en Raspberry.

<https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/BUILDING.md>

Instalamos los prerequisites para el ESP-IDF

```
sudo apt-get install git wget flex bison gperf python3 python3-pip  
python3-setuptools cmake ninja-build ccache libffi-dev libssl-dev dfu-util libusb-1.0-0
```

Descargamos e instalamos el Espressif IoT Development Framework (ESP-IDF)

https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/esp32/setup_idf_chip.md

Preparamos el entorno de CHIP

Al lanzar el comando **sources scripts/activate.sh** no encuentra el archivo. Buscamos tanto con el comando **find** como el comando **locate** pero no encontramos la carpeta. Echamos la vista atrás para revisar y nos damos cuenta de que al instalar **pi-bluetooth avahi-utils**, no lo hizo correctamente y empezamos a instalar todas las dependencias de nuevo.

Al descargarse e instalar todas las librerías de Matter algunos archivos fallan y, en los últimos vemos que nos quedamos sin espacio en la SD card durante la instalación de Matter y debemos hacer una imagen de la tarjeta para migrarlo a otra SD card de mayor capacidad. La que habíamos estado usando hasta ahora era de 16Gb. Recomendamos empezar directamente con una de 32Gb.

Realizamos una imagen de la tarjeta con el programa imageUSB.

Queda pendiente continuar desde este punto.

3. Conclusiones

Después de nuestro periplo por el proyecto del medidor de estrés podemos considerar que hemos conseguido nuestro objetivo que era llegar con solvencia a la lectura de los dos sensores y recibir los datos de forma alámbrica por puerto USB.

Tal como planteamos al principio nos gustaría haber llegado a poder transmitirlo de forma inalámbrica y creemos que, para ser Matter una tecnología tan nueva, hemos conseguido avanzar bastante teniendo en cuenta que el punto en el que está este protocolo es el de disponer una documentación limitada, escasa y con pocos ejemplos destinada a los fabricantes industriales de dispositivos domóticos.

Pensamiento crítico:

De forma retrospectiva sobre el proyecto podemos decir que los sensores que registran datos fisiológicos son bastante sensibles físicamente, no son fáciles de calibrar y la interpretación de los datos, más allá de la que hemos podido hacer de forma amateur, deberíamos seguir revisando el estado del arte de los papers sobre la materia para utilizar algoritmos ya creados y obtener datos filtrados y más fiables.

Por otro lado, Matter, a parte de que comparte un concepto parecido de comunicación con MQTT por su sistema de publicación-suscripción, se nota que está amparado por las Big Tech y que es un gigante con buenas bases lo que hace que su interpretación, instalación y uso sea más complejo y pesado que MQTT, mucho más liviano y asequible.

Finalmente, esperamos que este proyecto sirva de inspiración para futuros proyectos y que, como me he encontrado yo mismo, por muchos problemas que nos aparezcan por el camino nos animemos a solventarlos y perderle el miedo a las cosas que puedan parecer difíciles.

También agradecer a Joan Masdemont, formador del curso de “Creación de prototipos de IoT con Raspberry Pi” los conocimientos y soporte ofrecidos y dar rienda suelta a la creatividad de cualquier proyecto que se le ha presentado sin coartar la libertad de los alumnos. Y, por supuesto, también dar las gracias a los compañeros que han dado apoyo en los momentos que ha sido necesario para poder seguir avanzando y al centro de formación CIFO La Violeta de Barcelona por ofrecer esta formación subvencionada y poner a nuestra disposición el aula y todos los materiales necesarios sin los cuales no sería posible el aprendizaje.

4. Webgrafía

Fuentes de diversa índole consultadas para la realización de este proyecto:

Papers científicos relacionados:

<https://www.mendeley.com/catalogue/cdabe911-770b-3a8d-b8d3-043f478c63b7/>
<https://www.mendeley.com/catalogue/052705ad-241e-337a-8554-6f63279a44bf/>
<https://www.mendeley.com/catalogue/b9e1afe2-de34-3478-852f-86b296ded74e/>
<https://www.mendeley.com/catalogue/b5f65bc7-e54e-30e9-9b01-c8521fb27795/>
<https://www.mendeley.com/catalogue/9582062c-1d5d-328c-b297-620172588393/>
<https://www.mendeley.com/catalogue/3fbaee0b-2fb3-3cab-bc2f-08060c0a8230/>
<https://www.mendeley.com/catalogue/cdabaed6-c28f-3af8-a14c-ac6798ac4b23/>
<https://www.mendeley.com/catalogue/f8ba2b27-6a47-337e-be21-624657d5829b/>
<https://www.mendeley.com/catalogue/9582062c-1d5d-328c-b297-620172588393/>
<https://www.mendeley.com/catalogue/a5ccedb2-63b2-3cee-ab2c-cce3406bf4ed/>
<https://www.mendeley.com/catalogue/36ffa08c-a4af-34d5-8934-e15d9d3efb2f/>
<https://www.mendeley.com/catalogue/c050fe5d-e84b-3af9-a148-d4c869619211/>
<https://www.mendeley.com/catalogue/25c34d88-723a-3cd3-bad3-7cb69d968bbe/>
<https://www.mendeley.com/catalogue/86ffc5cd-8b7e-3547-b992-8bab159c3319/>
<https://www.mendeley.com/catalogue/350eb7e6-2a4d-3c6b-8fd8-fcf4b9629190/>
<https://www.mendeley.com/catalogue/07c56fa6-8f63-3810-92b0-ad00f354f82d/>
<https://www.mendeley.com/catalogue/04d568d3-b50c-3881-b90e-82f71e225368/>
<https://www.mendeley.com/catalogue/98ab7d30-cbe5-3ac9-95ae-f23e3fbfda8e/>
<https://www.mendeley.com/catalogue/80f3d76e-0f36-3baa-86d1-c009da8d6420/>
<https://www.mendeley.com/catalogue/61e3fca8-8cac-323a-aa23-e8b303a5ba31/>
<https://www.mendeley.com/catalogue/b5f65bc7-e54e-30e9-9b01-c8521fb27795/>
https://www.researchgate.net/publication/352579454_Low-Cost_Heart_Rate_Sensor_and_Mental_Stress_Detection_Using_Machine_Learning

Información general GSR:

<https://www.brainsigns.com/es/science/s2/technologies/gsr>

Sensores:

Pulse Sensor - SEN-11574:

<https://pulsesensor.com/pages/open-hardware>
https://cdn.shopify.com/s/files/1/0100/6632/files/Pulse_Sensor_Data_Sheet.pdf?14358792549038671331
<https://processing.org/reference/libraries/io/index.html>

GSR sensor:

https://wiki.seeedstudio.com/Grove-GSR_Sensor/
<https://www.seeedstudio.com/Grove-GSR-sensor-p-1614.html>
https://files.seeedstudio.com/wiki/Grove-GSR_Sensor/res/Grove-GSR_Sensor_WiKi.pdf
https://github.com/SeeedDocument/Seeed-WiKi/blob/master/docs/Grove-GSR_Sensor.md

IMU:

<https://vr-experience.es/slimevr-el-full-body-tracking-asequible-esta-cerca>

EEG:

<https://www.youtube.com/watch?v=h3Blku7MqSo>
<https://hackaday.com/2012/12/20/modifying-an-eeg-headset-for-lucid-dreaming/>
<https://lsdbase.org/2012/12/11/First-Dream-with-Modified-MindWave/>
<https://hackaday.com/2015/07/15/muse-eeg-headset-teardown/>

Librerías Sensor HR

https://github.com/bhawiyuga/PulseSensor_Playground_esp32/blob/master/src/PulseSensor_Playground.cpp

Integración de ESP32 en Arduino IDE

<https://aliakbarfani.wordpress.com/part-1-integrating-arduino-ide-with-esp32/>
<https://aliakbarfani.wordpress.com/part-2-setting-up-pulse-sensor-with-esp32-and-arduino-ide/>

Problema de la definición del PIN:

<https://stackoverflow.com/questions/59317822/esp32-with-pulsesensorplayground>

Pulse sensor con placa bluetooth HC-05:

<https://www.instructables.com/Pulse-Sensor-With-Bluetooth-and-Arduino/>

Aumentar/disminuir señal sensor PulseSensor:

<https://soloelectronicos.com/tag/sensor-cardiaco-arduino/>

ESP32 web server:

<https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/>

Host error

<https://askubuntu.com/questions/59458/error-message-sudo-unable-to-resolve-host-name>

Microcontrolador ESP32: (comprendiendo NodeMCU)

<https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/>

Entender ADC (Conversor Analógico Digital)

<https://tecnotizate.es/basicos-esp32-lectura-de-valor-analogico/>

HR to ESP32:

<https://microcontrollerslab.com/pulse-sensor-esp32-tutorial/>

<https://soloelectronicos.com/tag/sensor-cardiaco-arduino/>

HotGlue: <https://vimeo.com/58657081>

GSR to Arduino(ESP32):

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

Batería:

<https://programarfacil.com/esp8266/programar-esp32-ide-arduino/>

De ESP32 a Raspberry por wifi MQTT

<https://www.youtube.com/watch?v=Q2HL8rwZ20A>

De ESP32 a Raspberry por RX/TX

<https://blog.330ohms.com/2020/07/07/como-conectar-arduino-y-raspberry-pi-por-comunicacion-serial/>

Stress Detector Arduino

<https://www.instructables.com/Arduino-Stress-Detector/>

Pulse + GSR

<https://www.hackster.io/factoryeight/emotion-towards-a-better-future-a01489>

Interpretar GSR

<https://www.neuromarketingschool.com/respuesta-galvanica/>

<https://www.brainsigns.com/es/science/s2/technologies/gsr>

Entender como cargar programa ESP32

<https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/>

Problema Analog Read Resolution

<https://crisalctime.com/esp32-entradas-adc/>

<https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution/>

Instalar Python en nuestro PC

<https://www.python.org/downloads/release/python-3110/>

Micropython para ESP32

<http://docs.micropython.org/en/latest/esp8266/tutorial/intro.html#deploying-the-firmware>

Configuración de red en ubuntu

<https://netplan.io/examples/>

Blogs Espressif ESP32: Matter

<https://blog.espressif.com/matter-38ccf1d60bcd>

<https://blog.espressif.com/announcing-matter-previously-chip-on-esp32-84164316c0e3#getting-started>

https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.0.2/matter/BUILDING.html

Matter

<https://github.com/project-chip/connectedhomeip>

<https://github.com/project-chip/connectedhomeip/blob/master/docs/README.md>

<https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/BUILDING.md>

<https://github.com/project-chip/connectedhomeip/tree/master/examples/all-clusters-app/esp32>

https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/esp32/setup_idf_chip.md

Detector de movimiento con YOLO:

<https://medium.com/@enriqueav/detecci%C3%B3n-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246>

Recordar seleccionar la opción: Add Python to environment variables
Teensy: Placa de desarrollo