



# ViDS: Virtual Drum Set



Raspberry Pi per prototips IoT.  
CIFO La Violeta.

Manuel Cañete Carrillo  
Novembre 2024



## Índex

Introducció .....	3
Descripció del producte.....	3
Objectius del projecte. ....	3
Fases de desenvolupament. ....	3
Materials necessaris. ....	4
- Acceleròmetre + giroscopi: Mòdul GY-521 .....	4
Fase 2. Proves bàsiques del sistema.....	4
Comunicacions I2C. ....	4
Mòdul GY521.....	6
Sortida d'àudio. ....	9
Requisits del Mínim Producte Viable .....	11
Descripció: .....	11
Maquinari: .....	11
Programari:.....	11
Característiques principals: .....	11
Esquemàtic .....	11
Desenvolupament del programari .....	12
ViDS V0.1 .....	12
ViDS V0.2 .....	13
ViDS V0.3 .....	13
ViDS V0.4 .....	17
ViDS V0.7 .....	19



## Introducció

El Tomàs (Tom per amics i familiars) es un apassionat de la música. Ha tocat diferents instruments, des de els mes clàssics como la guitarra espanyola o el piano passant pels sons mes rockers de la guitarra elèctrica i el baix. Però sens dubte el seu favorit es la bateria. De fet, fa uns mesos ha format amb el amics una banda de música per gaudir tocant versions dels clàssics del blues, rock y pop. I com no, el Tom toca la bateria. Tres cops per setmana assagen en un petit local cedit per hores pel ajuntament del poble. I això vol dir carregar amb la bateria per anar al local d'assaig, muntar-la per assajar i tornar a desmuntar-la per tornar a casa. Una feinada. El Tom es mira amb certa enveja els companys que toquen la guitarra, el baix o el cantant, que tenen molts menys problemes logístics.

Una freda i plujosa tarda de diumenge que no convidava a sortir de casa, el Tom feia zàping a la tele sense decidir-se per cap programa. Per un moment es va aturar al “30 minuts” de TV3, on parlaven de la prometedora tecnologia d’Internet de les Coses, les seves possibilitats i la revolució que de ben segur portarà a terme. I de entre un munt de sensors, actuadors, xarxes sense fils, programaris i demés, el Tom s’hi va fixar en unes plaques que eren com potents ordinadors, de petites dimensions i preus continguts. “Amb aquests miniordinadors, la comunitat *maker* ha fet de tot, des de prototips de vehicles autònoms, fins a impressores 3D”, comentava el presentador del ‘30 minuts’. I al Tom se li va encendre un llum: potser no es podria fer una bateria virtual per assajar? Segurament no seria perfecta i tindria les seves limitacions però estalviarà temps i reduirà les possibilitats de que la bateria rebí un cop durant els múltiples trasllats de casa al local d’assajos i del local a casa. I dit i fet, aquella mateixa tarda es va posar a buscar mes informació i a contactar amb grups de persones amb coneixements i experiència en el món de l’IoT. En qüestió de dies ja tenia un esborrany del projecte, amb materials, objectius i fases de implementació.

## Descripció del producte

Sistema de bateria virtual que identifiqui els moviments de les mans del músic i reproduueixi el so de l’element corresponent de la bateria (plats, caixa, tambors...). Com a millora i en funció del tems i materials disponibles, es poden afegir també els moviments dels peus.

## Objectius del projecte.

1. Posada en marxa del diferents sensors i actuadors connectats a una Raspberry Pi (veure apartat de materials), creant llibreries adients en llenguatge Python.
2. Desenvolupar el programari principal en Python, identificant quin paradigma es el mes adient per un òptim funcionament del sistema (multi processos, multi fil, interrupcions, esdeveniments)
3. Reciclat de components com pantalles gràfiques i altaveus provinents d’aparells espatllats per incorporar-los al projecte.

## Fases de desenvolupament.

Amb l’objectiu de disposar d’un pla de treball que permeti avaluar correctament l’avanç del projecte, s’han definit 4 fases, cadascuna amb les seves feines i fites a assolir.

- **Fase 1: Materials.** Identificació del sensors i actuadors necessaris, fulls de característiques, esquemes bàsics de connexions, alimentació necessària, establiment del requisits globals del sistema, cerca de materials complementaris, etc.
- **Fase 2. Proves bàsiques del elements del sistema.** Posada en marxa de cadascun del sensors i actuadors seleccionats, validació i substitució per altres si no compleixen amb els requisits necessaris. Esquemàtic global de la solució, llibreries Python de cada sensor/actuador.



- **Fase 3. Construcció del prototip de Producte Mínim Viable** . El resultat d'aquesta fase ha de ser un prototip mínim funcional, que permeti tant veure el seu funcionament, com identificar els punts de millora a implementar en següents fases.
- **Fase 4. Implementació de millores**. Sobre el prototip de la fase anterior s'implementaran tant les millores detectades com les noves característiques identificades, en un procés cíclic de prova-error-millora.

Dintre dels objectius del curs es preveu completar com a mínim la fase 3 i dependent del temps disponible i les dificultats trobades, realitzar alguna part de la fase 4.

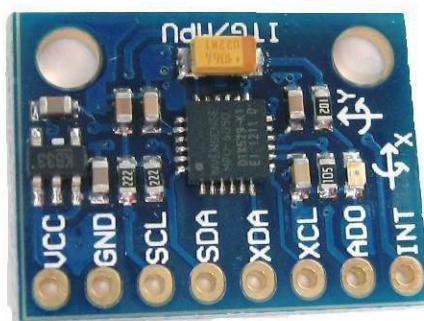
## Materials necessaris.

Inicialment s'han identificat els següents materials necessaris que es validaran durant la primera fase del projecte.

- 1 x Raspberry Pi 3 Model B+
- 2 x Acceleròmetre
- 2 x Giroscopi
- 2 x bronzidor passiu (o altaveus)
- 1 x Display alfanumèric
- 2 x DAC (mínim 8 bits)
- 1 x *BreadBoard* i/o placa prototips
- LEDs, resistències, condensadors, cables de connexió...

En particular, el element mes important del sistema es el mòdul GY-521:

- **Acceleròmetre + giroscopi: Mòdul GY-521**
  - Alimentació: 5V (no cal alimentació externa)
  - Processador: MPU6050
  - Comunicació I2C
  - Llibreries Python



## Fase 2. Proves bàsiques del sistema

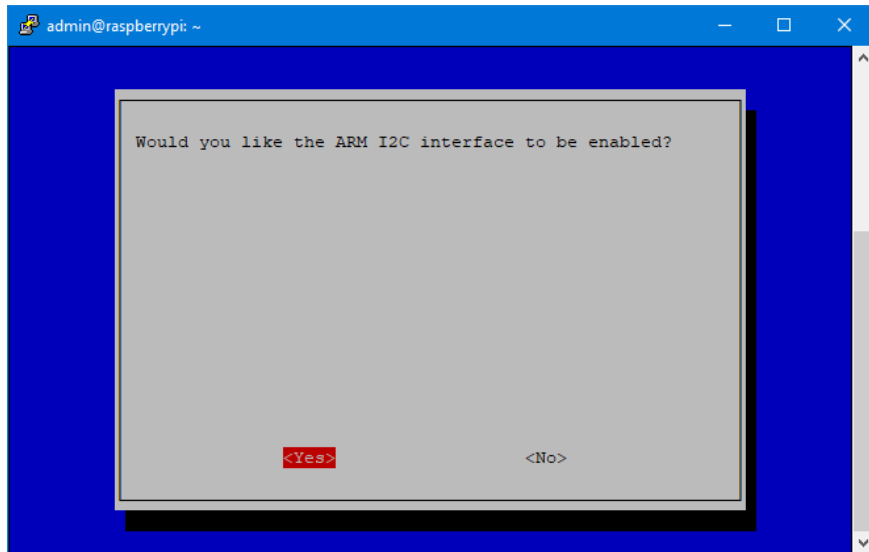
### Comunicacions I2C.

Com veurem mes endavant, el mòdul fa servir el protocol I2C per rebre/enviar informació. Per això necessitem posar en marxa aquest sistema de comunicacions de que disposa la Raspberry Pi i utilitzarem la llibreria 'smbus'.

En la Raspberry Pi cal activar el dispositiu I2C mitjançant les següents instruccions:



```
$ sudo raspi-config
3 - Interface options
I5 - I2C
<Yes>
```



En les proves inicials hem vist que el sistema donava alguns errors de comunicacions en el protocol I2C. Molt probablement es degut a la utilització del extensor del GPIO de la RPi, la *breadboard* i els cables de connexió de la *breadboard* cap al mòdul, que afegeixen capacitats paràsites que poden fer fallar el protocol. En la primera fase de proves baixem la velocitat del bus I2C a 1 kbps (pot funcionar fins a 400 kbps). Per baixar la velocitat cal modificar el fitxer '/boot/firmware/config.txt', afegir la línia

```
dtoverlay=i2c_arm_baudrate=1000
```

i reiniciar la raspberry:

```
$ sudo nano /boot/firmware/config.txt
$ sudo reboot
```



```
admin@raspberrypi: ~  
GNU nano 7.2 /boot/firmware/config.txt  
# For more options and information see  
# http://rptl.io/configtxt  
# Some settings may impact device functionality. See link above for details  
  
# Uncomment some or all of these to enable the optional hardware interfaces  
dtparam=i2c_arm=on  
#dtparam=i2s=on  
#dtparam=spi=on  
  
# Ajustamos velocidad i2c a 1 kbits para pruebas  
dtparam=i2c_arm_baudrate=1000  
  
# Enable audio (loads snd_bcm2835)  
dtparam=audio=on  
  
# Additional overlays and parameters are documented  
# /boot/firmware/overlays/README  
  
# Automatically load overlays for detected cameras  
camera_auto_detect=1  
  
^G Help      ^O Write Out ^W Where Is  ^R Cut       ^T Execute   ^C Location  
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Un cop reiniciada la RPi i connectar el mòdul GY-521, comprovarem que el I2C funciona i detecta el mòdul amb el comando 'i2cdetect -y 1' on el 1 indica que hem connectar al bus I2C número 1 de la RPi:

```
$ sudo i2cdetect -y 1
```

```
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00:                -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- 68 -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Podem veure que detecta un dispositiu connectat en l'adreça 68H, que correspon al mòdul GY-521.

### **Mòdul GY521.**

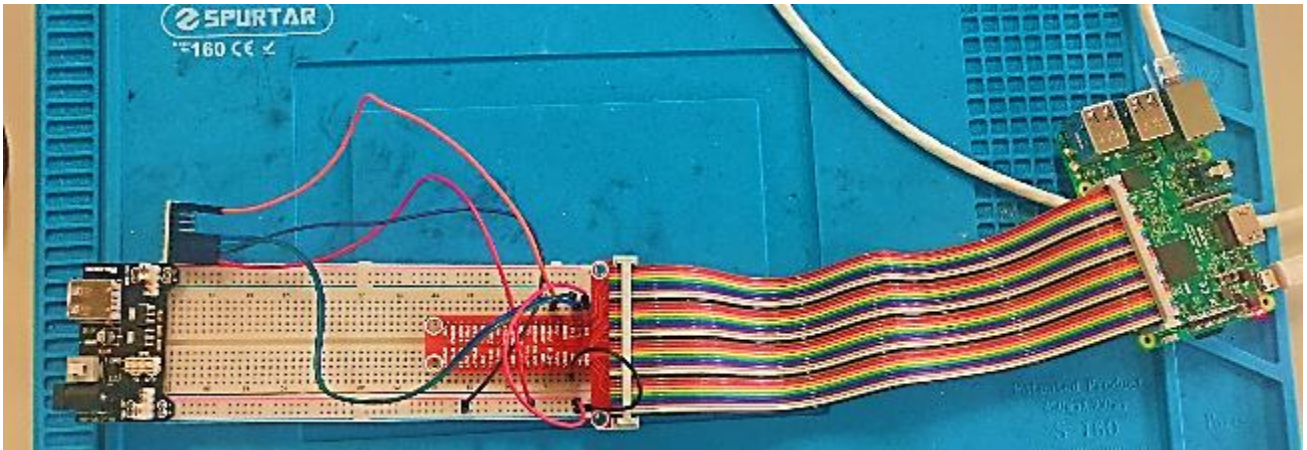
Aquest mòdul conté un acceleròmetre de tres eixos i un giroscopi de tres eixos que farem servir pel nostre projecte. El mòdul té més prestacions que en fase inicial no es consideren útils pel projecte. La comunicació amb el mòdul es fa mitjançant un protocol I2C des de la Raspberry Pi.

L'alimentació del mòdul vindrà directament de la Raspberry Pi, ja que els baixos requeriments de potencia fan innecessari la utilització d'una font d'alimentació externa.

El mòdul disposa de un pin de selecció de adreça (AD0) que correspon al LSB d'aquesta adreça I2C. Per defecte esta a un valor 0 gràcies a una resistència de *pull-down* connectada a GND i per tant la adreça I2C del dispositiu serà 68H. Per tal de utilitzar dos mòduls en paral·lel sobre el mateix bus I2C, caldrà que un dels mòduls tingui aquest pin connectat a VCC per tal de que la seva adreça canviï a 69H.

Per tal provar tant les comunicacions I2C como les funcionalitats del mòdul muntem un primer prototip sobre un placa *breadboard* connectada a la Raspberry Pi amb un extensor de bus.





Per posar el mòdul en marxa, consultem ChatGPT per un programa mínim que llegeixi els valors i el presenti en la terminal en mode text. Inicialment no funciona (probablement perquè fa servir interrupcions que no estan ben configurades) però amb unes modificacions del codi passem a una versió que fa consultes periòdiques dels valors del acceleròmetre i del giroscopi. Amb això comprovem que el mòdul funciona correctament i canvien els valors quan movem el dispositiu.

Adicionalment fem una cerca a Internet per veure altres projectes basats en aquest mòdul i descobrir les seves funcionalitats. Trobem el repositori '[https://github.com/ZHomeSlice/Simple\\_MPU6050](https://github.com/ZHomeSlice/Simple_MPU6050)'. També revisem i provem aquestes funcions en Python per assegurar el correcte funcionament i comprensió del sistema inicial.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Acelerómetro: -15452 -2984 -1000
Giroscopio:    -267 -185 113

Acelerómetro: -15508 -3048 -972
Giroscopio:    -251 -157 130

Acelerómetro: -15560 -3156 -1052
Giroscopio:    -268 -186 134

Acelerómetro: -15560 -3052 -1264
Giroscopio:    -231 -175 123

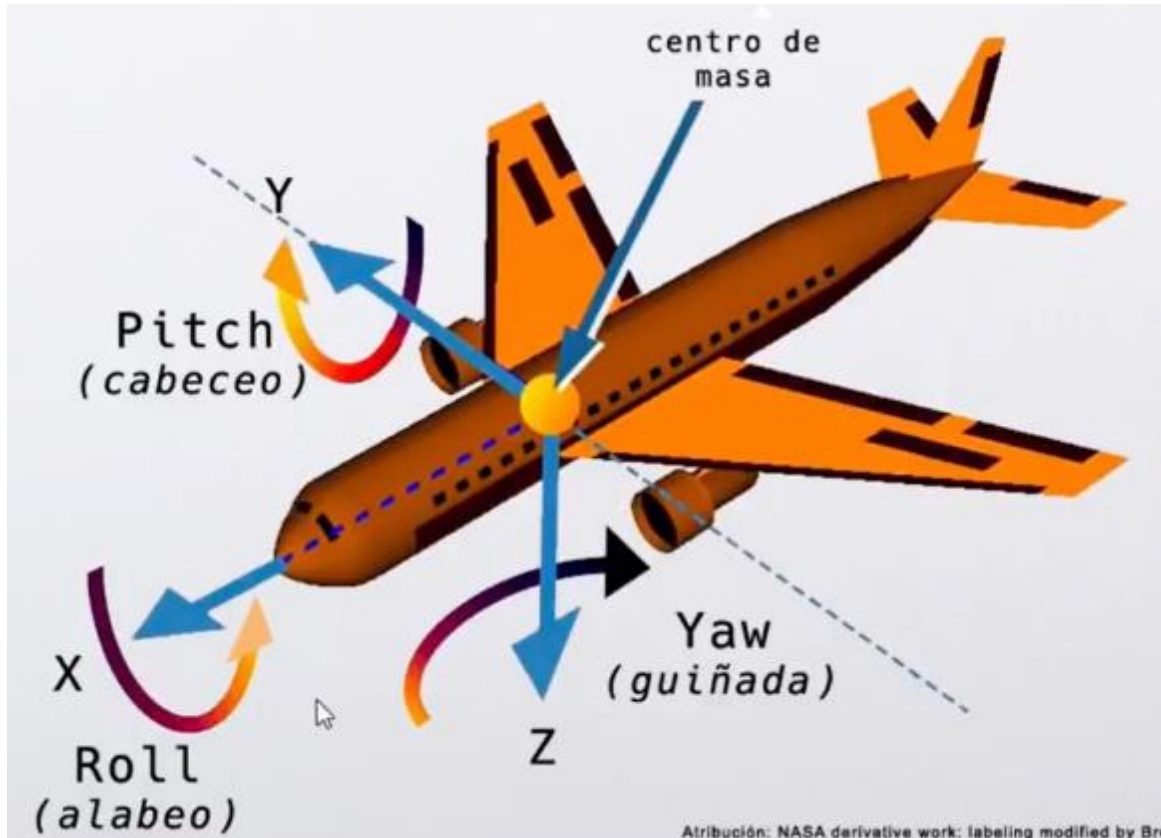
SSH: 192.168.5.2  0 0 0 0
```

A continuació provem algunes funcionalitats del mòdul per tindre-les en compte i veure de cara a futur si poden ser interessants pel projecte:

- Sensor intern de temperatura. En el nostre cas mesuraria la temperatura de les mans que és on se situen els sensors.
- Filtres passa-baix del acceleròmetre i del giroscopi. Aquest filtres eliminen soroll o petits moviments del dispositiu, deixant passar només els moviments mes lents. Aquesta funcionalitat es considera molt interessant i es comprovarà mes endavant amb el prototip acabat.
- Processador intern del xip MPU6050. Segons el *datasheet* del mòdul, aquest processador es pot programar per detectar diferents esdeveniments com ara caigudes i girs bruscos o quan es superen certs llindars de valors en els sensors. Quan es dona un esdeveniment programat, el mòdul genera una interrupció que pot recollir la Raspberry i en aquest moment llegir els valors dels sensors. Aquesta funcionalitat es pot interessant ja que descarrega el processador de la Raspberry dels càlculs i redueix l'ús del bus de comunicacions I2C. No obstant això en el full de característiques disponible no surt informació de com programar aquestes funcionalitat i de moment no fem servir.

Finalment repassem el vídeo 65 del curs d'Arduino que parla de les funcionalitats bàsiques d'aquest mòdul per tal d'entendre al màxim el seu funcionament i el vocabulari associat. D'aquesta manera veiem com es parla de *Roll*, *Pitch* i *Yaw* per referir-se als moviments que detecta el giroscopi al voltant del eix X, Y i Z del acceleròmetre respectivament. El vídeo il·lustra amb la figura d'un avió com es defineixen els eixos i els moviments al voltant dels mateixos.



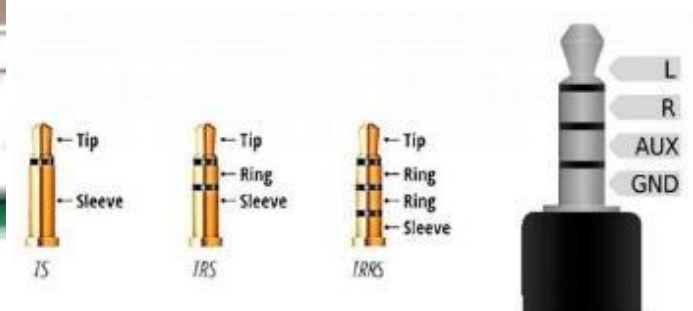


### Sortida d'àudio.

Inicialment es va considerar que es necessitaria un DAC (*Digital Analog Converter*) per emetre els diferents sons de la bateria. Amb una recerca més acurada i consultant *ChatGPT* veiem que la Raspberry Pi disposa de un sortida de Àudio/Video analògic que, tot i que no donarà la mateixa qualitat que un DAC, pot ser útil pel prototip i no suposa un afegit al projecte.

Aquesta sortida d'àudio/vídeo disposa d'un connector femella de 3,5 mm situat entre el connector RJ45 (corresponent a la xarxa Ethernet) i el connector per una càmera externa. Permet la connexió d'un mascle tipus TRS (*Tip, Ring, Sleeve*) de uns auriculars sense micròfon o un altaveu, amb dos canals de so (estèreo).

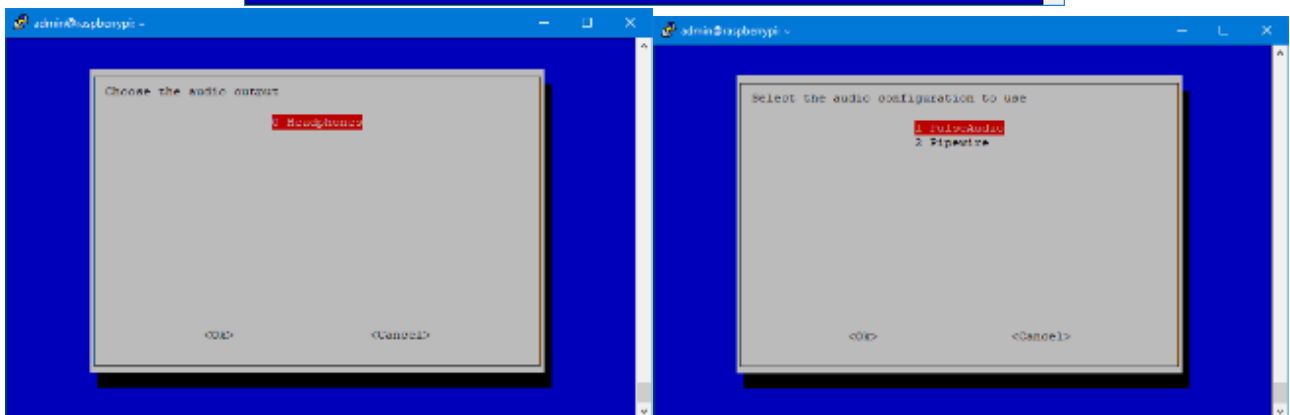
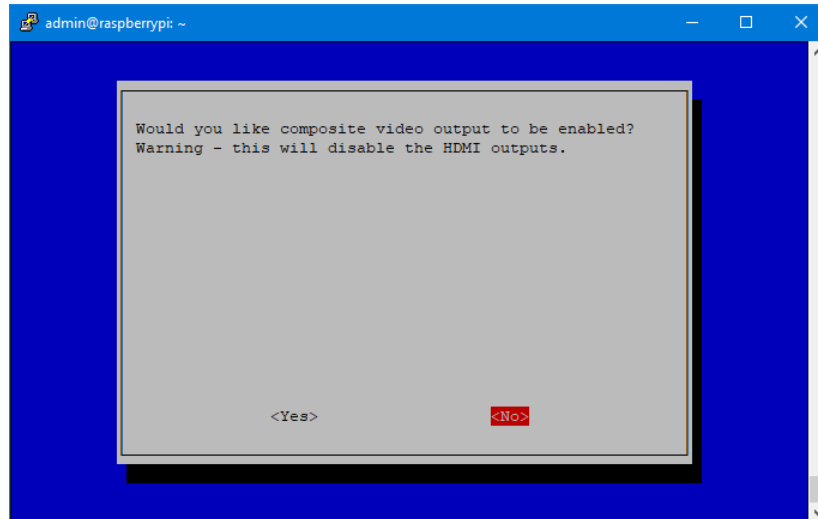
Per tal de que es puguin reproduir sons en aquesta sortida queda en mode '*Pulseaudio*'. Així mateix la sortida de vídeo no la farem servir i la deixarem desactivada en la configuració.



En la configuració de la Raspberry ens hem d'assegurar de desactivar la sortida de vídeo compost (*composite video*) i activar la sortida d'àudio '*headphone*' en mode '*PulseAudio*':



```
$ sudo raspi-config
  2 Display Options
    D4 Composite
      <No>
  6 Advanced Options
    A7 Audio Config
      1 PulseAudio
  1 System Options
    S2 Audio
      0 Headphones
```



Un cop modificades aquestes opcions, cal reiniciar la Raspberry Pi per tal que els canvis tinguin efecte:

```
$ sudo reboot
```

Un cop la Raspberry Pi ha reiniciat, ajustem el volum de la sortida d'àudio amb la funció *amixer*:

```
$ amixer set Master 65535
```

El valor de 65535 correspon al 100% volum. El deixem amb aquest valor ja que aquesta sortida no té gaire potencia i per tant aquest valor es adient per uns auricular o un petit altaveu.



## Requisits del Mínim Producte Viable

Un cop posats es marxa els dispositius necessaris pel projecte i amb un coneixement mes detallat dels mateixos, passem a detallar els requisits del MVP (*Minimum Viable Product*).

### Descripció:

Dispositiu que captura els moviments de les mans fent servir acceleròmetres i giroscopis, identifica els moviments corresponents a tocar una bateria amb diferents elements i fa sonar els sons corresponents.

### Maquinari:

- **1 x Raspberry Pi 3 model B+.**
- **2 x mòdul GY-521.** Inicialment cada mòdul es connectarà a un bus I2C separat per evitar inicialment problemes de bloqueig de recursos.
- **1 x altaveu estèreo.** Es connectarà a la sortida de A/V de la Raspberry Pi, tot i que només es farà servir la part d'àudio (per precaució es desactiva la sortida de video).
- **Leds.** S'utilitzaran com indicadors visuals en paral·lel amb els altaveus.
- **2 x cables de connexió.** Es munten dos cables que permetin connectar la Raspberry Pi amb els mòduls GY-521 i permeti un moviment còmode dels braços. Aquest element es considera crític ja que ha de permetre el moviment còmode de l'usuari i a la vegada servir com bus de comunicacions I2C amb la Raspberry Pi.

### Programari:

- **Llenguatge de programació:** Python V3.11.2
- **Llibreries inicialment necessàries:**
  - o RPi.GPIO: llibreria estàndard de la Raspberry Pi per control de les entrades i sortides.
  - o Pygame: llibreria àmpliament utilitzada per dissenyar jocs amb Python. Es faran servir les funcions que permeten enviar un so per la sortida A/V analògica de la RPi.

`$ sudo apt install python3-pygame`

- o Smbus: llibreria estàndard per manejar el bus de comunicacions I2C:

`$ sudo apt install python3-smbus`

- o Leds. Llibreria per manejar leds de nova creació

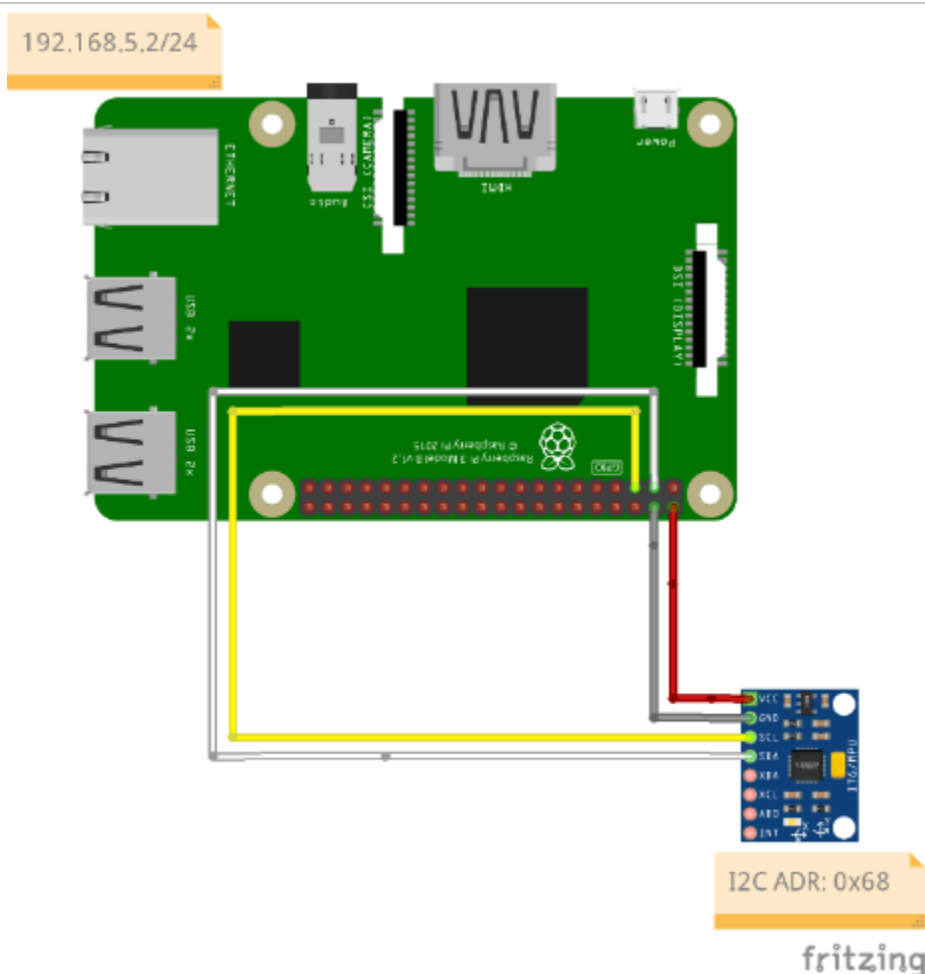
- o MDU6050. Llibreria desenvolupada per Martijn ([martijn@mrtijn.nl](mailto:martijn@mrtijn.nl)) i disponible per instal·lar (<https://pypi.org/project/mpu6050-raspberrypi/>) sota llicència MIT (<https://github.com/mrtijn/mpu6050>). (`sudo apt install python3-mpu6050`)

### Característiques principals:

- El control de cada dispositiu GY-521 es independent per busos I2C separats. Això evita inicialment els problemes de conflictes per accés a un mateix recurs compartit.
- El paradigma de lectura del GY-521 es per consulta periòdica (*pooling*) cada 100 ms. Això permet detectar fins a 10 moviments per segon, que es considera suficient en una primera versió. Cal tindre en compte que en musics professionals s'han mesurat fins a 300 beats per minut en una bateria, que equival a 5 cops per segon.

### Esquemàtic

Dibuixem el esquema d'interconnexions amb el programari Fritzing.

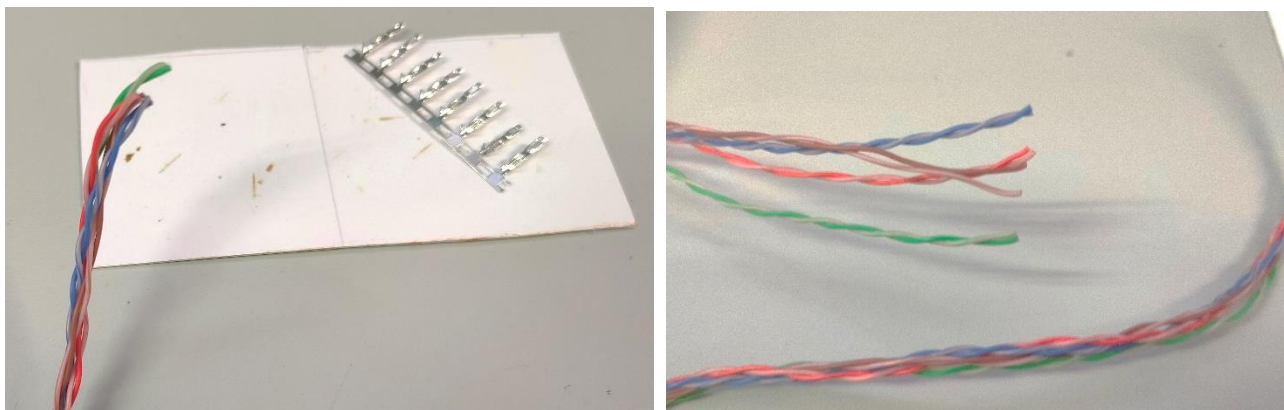


## Desenvolupament del programari

Un cop muntat el prototip comencem a construir el programari de control, combinant els diferents sensors (acceleròmetre, giroscopi) i actuadors (sortida de so). Ens ajudem de *ChatGPT* per tal de resoldre dubtes i generar codi més ràpidament. En cada pas anem guardant versions successives del codi per si cal tornar enrere i les codifiquem amb versió V0.x per ressaltar que es un versió de desenvolupament. A continuació es recullen les part testejades, validades i descartades en aquest procés iteratiu:

### ViDS V0.1

Sistema bàsic utilitzant el paradigma de consulta periòdica (*pooling*) del sensor i activació de un so. Amb aquesta primera versió ja veiem que les comunicacions I2C seran un punt crític per la longitud dels cables. Cal tenir en compte que el bus I2C està pensat per treballar a nivell de placa PCB i no tant de cables, degut a la seva arquitectura de portes lògiques amb tecnologia de col·lector obert (*open-collector*). De fet, després de construir un cables I2C fent servir cable ethernet, tornem a les connexions amb cables Dupont, ja que aquests funcionem pitjor.



Finalment tenim una primera versió de programari amb una visió global del sistema que volem dissenyar.

### ViDS V0.2

En aquesta segona evolució volem millorar la estructura del codi fent servir classes de Python, el que permetrà una millor lectura i evolució del programari. En aquest punt trobem amb l'ajuda de ChatGPT una llibreria Python que inclou una classe bàsica per fer servir el processador MPU6050 que inclou el mòdul GY-571 (<https://github.com/m-rtijn/mpu6050>). La llicència MIT d'aquesta llibreria ens permet utilitzar-la sense limitacions pel nostre projecte. Revisem la llibreria, la documentació i els exemples, i procedim a instal·lar-la amb el mètode habitual:

```
$ sudo apt install python3-mpu6050
```

però ens dona l'error "no se encuentra candidato". Finalment es copia directament el fitxer 'mpu6050.py' en la carpeta de paquets de Python3 i es comprova que la llibreria queda correctament instal·lada i funcional.

Amb aquesta llibreria ja aconseguim detecció de si el músic està tocant el plat o la caixa mitjançant la lectura del acceleròmetre del eix X. Detectem que el sistema és bastant sensible a la posició inicial del sensor que s'ha de trobar en posició horitzontal inicialment, per tal de que el processador obtingui una posició de referència.

copiant directament el fitxer 'mpu6050.py' en la carpeta de paquets de Python3.

- Queremos probar la libreria disponible en '<https://github.com/m-rtijn/mpu6050>', que ya probamos en el código de commissioning del MPU6050. Queremos hacer la misma lógica pero con la clase la definida en este repositorio.
- Intentamos instalar la libreria (según las instrucciones del repositorio) y no se consigue:
  - \$ sudo apt install python3-mpu6050 (no se encuentra candidato)
  - \$ pip install mpu6050-raspberrypi
- Seguimos adelante copiando utilizando el código directamente (Licencia MIT).
- Se consigue discriminar entre plató y tambor, mediante la lectura de la aceleración en el eje X.
- El sistema es muy sensible a la calibración inicial. El sensor debe estar plano inicialmente (????)
- Se cierra esta versión el 27/11/2024 @ 14:05h

### ViDS V0.3

En aquesta nova evolució volem comprovar algunes de les funcionalitats avançades que hem llegit al *datasheet* com són el filtre i el rang de funcionament. Tornem a fer una recerca a Internet per buscar tota la informació disponible i en especial la referent als registres i funcionalitats avançades. Finalment ho trobem i



podem començar a fer-las seguir. En concret apliquem un filtre de 20 Hz al sensor d'acceleració i canviem el rang a  $\pm 4g$  (4 vegades l'acceleració terrestre). Veiem que el sistema funciona una mica millor tot i que aquesta millora no es espectacular. Ho deixem per corroborar mes endavant.

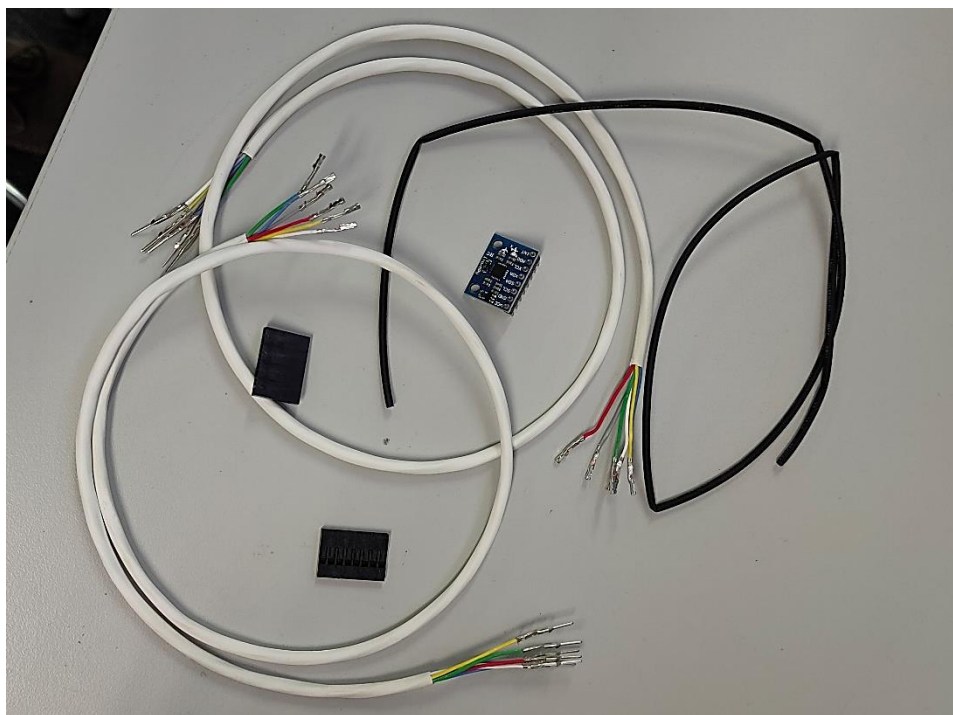
Per altre banda, en aquesta versió volem incrementar la velocitat del bus I2C per tal que no sigui un coll d'ampolla per la detecció de moviments i tampoc doni errors de comunicacions. Partim de una velocitat molt baixa (1 kbps) i amb cables Dupont de aproximadament 15 cm.

Amb l'oscil·loscopi visualitzem els senyals de rellotge (SCL) i dades (SDA) del bus I2C i veiem que sense cap tipus d'espera ni temporització en el bucle principal, tenim 0,312 ms entre paquets de lectures (sobre el senyal SCL) i el paquet complet de rellotge té una duració de 0,240 ms.

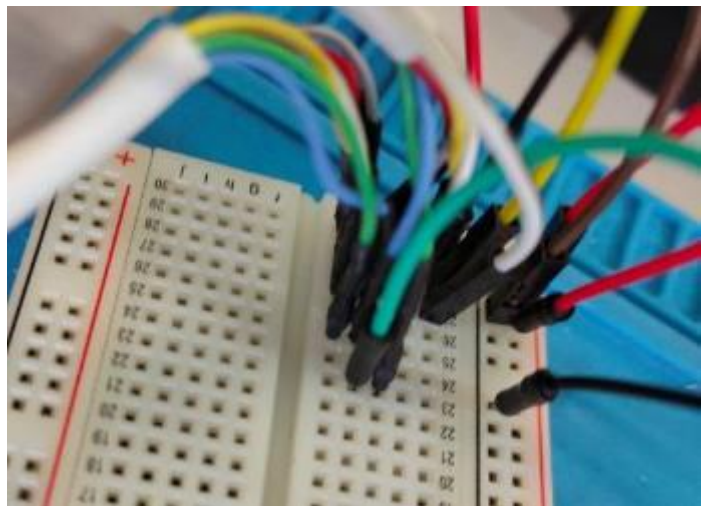
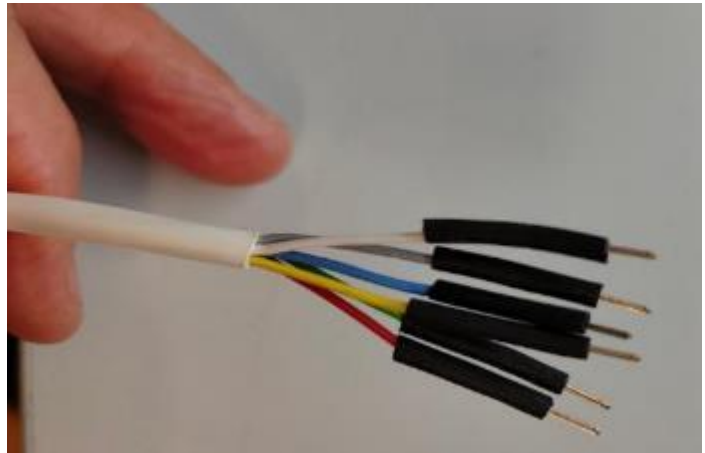
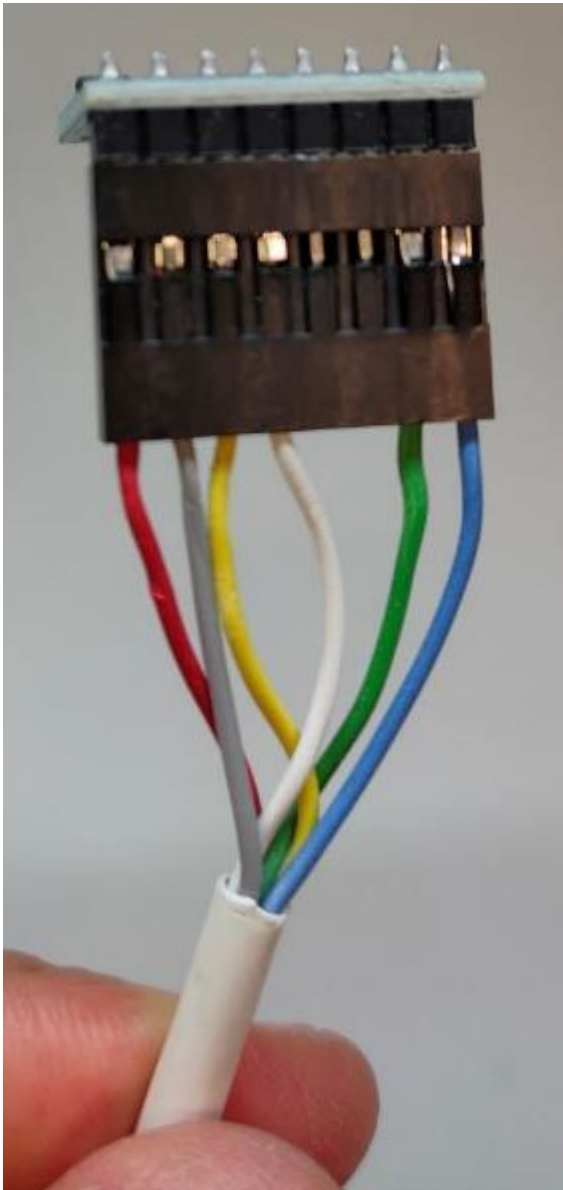
A continuació posem una temporització de 100 ms en el bucle principal (`time.sleep(0.1)`) i verifiquem que el funcionament es correcte mesurant que el temps entre paquets de SCL es de 102 ms, que correspon al temps d'espera utilitzat mes el temps de processament de les instruccions del bucle principal. També verifiquem que el sistema necessita un total de set paquets I2C per llegir totes del dades, el que suposa un temps total de 1,980 ms.

Finalment verifiquem que els flancs de pujada i baixada dels senyals I2C (SDA i SCL) no presenten distorsions importants i compleixen amb els requeriments del sistema I2C. La degradació d'aquest flancs (sobre tot el de pujada) es degut a les capacitats paràsites causades pels cables Dupont, a la utilització del extensor de bus de la Raspberry Pi i a la pròpia placa *breadboard* de prototipats.

En aquest punt, allarguem els cables Dupont fins a 20 cm i repetim les mesures que ens permeten veure que les característiques dels senyals gairebé no han canviat i per tant podem continuar amb aquesta longitud de cables, que permet una millor experiència de l'usuari.









- 
- Queremos probar la configuración del sensor en lo referente a rango y filtro. Cambiando el acelerómetro a rango 4G y con un filtro de 20 Hz, aparentemente funciona un poco mejor, lo que tendremos que confirmar en los siguientes pasos.
  - A continuación vamos a verificar el correcto funcionamiento de las comunicaciones I2C, empezando con los cables Dupont de una longitud aproximada de 15 cm y la velocidad del I2C en 1000 bps como se comentó en apartados anteriores.
  - Hemos tomado captures en el osciloscopio, ejecutando el programa sin ningún `time.sleep()` en el bucle principal. Esto nos da un tiempo de 0.312 ms entre paquetes de lecturas (leído en el clock SCL). También miramos que el paquete completo de SCL tiene una duración de 0,240 ms y el tiempo entre paquetes de SCL es aproximadamente de 48 us.
  - A continuación ponemos un `time.sleep(0.1)` en el bucle principal para verificar el correcto funcionamiento. Comprobamos que entre paquetes de SCL hay un tiempo de 0,102s, entendiendo que el excedente de 0,1s es el tiempo de procesado, ya que no se tiene en cuenta ese tiempo de procesado. Ahora podemos comprobar también que el pack de SCL dura un total de 1,980 ms y se puede ver un total de 7 paquetes de SCL.
  - Por último comprobamos las pendientes de subida/bajada de SCL y SDA, para verificar que las capacidades parásitas no afectan el funcionamiento del I2C.
- 
- A continuación alargamos los cables unos 20 cm con cable Dupont. La imagen en el osciloscopio apenas cambia y las subidas de SCL y SDA son prácticamente iguales, lo que permite asegurar que la influencia de estos nuevos cables es pequeña. Repetimos las medidas anteriores para comprobar que todo sigue funcionando. El tiempo de los 7 bloques de SCL es de 1,980ms (capturado+foto). El tiempo entre bloque y bloque de SCL es de 0,102s (bmp\_6\_015). Por último ponemos un `time.sleep(0.03)` y verificamos que el tiempo entre packs de SCL es de 32ms. El excedente de 30ms es el tiempo de procesado de la RPi.

Ara es el moment de incrementar la velocitat de les comunicacions I2C per veure els límits i deixar-la en un correcte balanç entre velocitat i seguretat de les comunicacions. L'increment de velocitat ens permetrà més endavant afegir més processat i més funcions al codi sense que el bus I2C sigui una limitació. Recordem que per canviar la velocitat cal modificar el fitxer 'config.txt' de la Raspberry Pi:

```
$ sudo nano /boot/firmware/config.txt
```

Apugem la velocitat a 50 kbps i reiniciem la Raspberry Pi per fer efectius els canvis. Repetim les verificacions explicades en els apartats anteriors i arribem a la conclusió de que aquesta velocitat I2C es també segura i no presenta gaires errors de comunicació. El nou temps entre inicis dels blocs de SCL es 0,640 ms. Amb una pausa de 30 ms en el bucle principal (`time.sleep(0.03)`), que correspon a 33,3 consultes per segon, el temps entre inicis de les consultes de 39,4 ms. La diferència amb el temps programat es de 9,4 ms, que correspon al temps de processat. També comprovem el funcionament amb 50 consultes per segon, es a dir, una pausa de 20 ms (`time.sleep(0.02)`).

Per últim, provem a una velocitat a 100 kbps i repetim tot el protocol de verificacions. Veiem que s'incrementen els error de bus I2C tot i que no es dramàtic i molt probablement es podran corregir amb uns cables de millor qualitat.

Una conclusió important treu en aquesta iteració que haurem de fer servir algun tipus de paradigma (*time-slicing*, *multiprocesses*, *multithreads*) per tal de garantir un temps constant entre mostres del sensor, per tal que eliminar la influència del temps de processat del bucle principal tot contribuint a una millor detecció dels moviments i una sensació més propera a la realitat del funcionament de la bateria.



- **Canvi en la velocitat del I2C.** Canbiamos la velocidad del I2C y la ponemos a 50kbps cambiando el fichero config.txt: —

- ~~\$ sudo nano /boot/firmware/config.txt~~

~~Reboot para que tome los cambios. La detecció del I2C es correcta (i2cdetect -y 1). El tiempo entre inicios de bloques de clock es de 0,640ms (6\_017). El tiempo entre inicios de poolings es de 39,4 ms, con time.sleep(0,03). La diferencia debe ser el tiempo de procesado.~~

~~Per ultimo, canvien a time.sleep(0.02) i comprovem que entre el final dels blocs de SCL y el principi del següent hi ha 20ms, es a dir, es correcte. Ara el block de SCL es de 9,800 ms (6\_021).~~

- **Canvi a 100kbps.** Como última prueba cambiamos a velocidad 100 kbps. Comprobamos que ahora el paquete de SCL dura 5,6 ms (6\_022). Hay 25,60 ms entre inicios de paquetes. Hay 20 ms entre final e inicio de paquetes SCL, que coincide con el sleep(0.02) del código. También vemos que aumentan los errores en el bus i2c (OSError) però no en gran cantidad.

## ViDS V0.4

Amb aquesta nova iteració volem millorar el calibratge del sistema i millorar la detecció del moviment en vertical (eix Z), així com la inclinació. Tot això per determinar quin element de la bateria ha de sonar. En aquest punt afegim un segon mòdul GY-521 amb l'objectiu de tenir com a mínim quatre sons diferents, dos per la ma esquerra i dos per la ma dreta. El segon mòdul es connecta al mateix bus I2C però afegint un fil entre el pin AD0 i +5V amb el que canvien l'adreça I2C a 69H. Aquest sistema permet estalviar pins de la Raspberry Pi que mes endavant es podrien utilitzar per funcions addicionals.

```
$ sudo i2cdetect -y 1
```

```
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- 68 69 -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- --
```



```
admin@raspberrypi ~  
login as: admin  
admin@192.168.5.2's password:  
Linux raspberrypi 6.6.51+rpt-rpi-v8 #1 SMP PREEMPT Debian 1:6.6.51-1+rpt3 (2024-10-08) aarch64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Mon Dec  2 13:30:53 2024  
admin@raspberrypi:~$ sudo i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- 68 69 -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  
admin@raspberrypi:~$
```

Per últim, gràcies a la ajuda del Juan Francisco, disposem de uns cables de millor qualitat i longitud per connectar els mòduls a la Raspberry Pi.

Repetim els protocols de comprovacions i mesures del bus I2C verificant que tenim correctament connectats els dos mòduls GY-521 i que tots dos funcionen correctament.

El nou esquemàtic del sistema es el següent:



Amb la versió V0.4 deixem congelat el disseny del maquinari del prototip i treballem amb el programari, sobre tot en els següents punts:

- Millorar el calibratge del sistema complet
- Millorar la detecció dels moviments, evitant sons quan no hi moviment i moviments sense cap so.
- Reproducció de sons simultanis, que es com habitualment funciona una bateria. Cal indicar que fins ara només es reproduïa un so.

Per tal d'avançar més ràpidament utilitzarem ChatGPT amb indicacions clares i precises del que ha de fer i el que no ha de fer, gràcies a tota la experiència i coneixement adquirit en aquestes primeres fases de desenvolupament. No es comenten les versions entre la V0.4 i la V0.7 ja que corresponen a proves amb errors o modificacions sense gran rellevància.

Entre altres funcionalitats, li demanem un programari separat de calibratge dels sensors, gravació de fitxers tipus CSV ('ViDS\_config.csv') per tal d'analitzar-los, càlcul automàtic de llinars aplicant tècniques estadístiques, detecció de moviments i acceleracions en els diferents eixos, etc. Finalment el calibratge queda guardada en un fitxer de configuració que llegirà posteriorment el programa principal. Realment comprovem que amb les indicacions clares i concises el codi Python generat per ChatGPT funciona aplicant només petits canvis o fins i tot indicant-li els errors o les modificacions que volem.



En el programa principal li indiquem que volem un sistema de detecció de moviments amb histèresi i amb control de estats per tal de evitar la repetició de sons amb petits moviments de les mans i garantint que abans de sonar un nou element la ma ha passat per la seva posició inicial.

Després de varies iteracions, proves i correccions i millores ens quedem amb les versions següents:

- Calibratge del sistema: ViDS\_Calib\_V0.2.py
- Bateria virtual: ViDs\_V0.7.py

Que comprovem que tenen un funcionament prou correcte.

~~Queremos mejorar la calibración inicial y una mejor detección del movimiento en Z y en inclinación para determinar que elemento debe sonar.~~

~~— Queremos añadir el segundo acelerometro, y disponer de 4 sonidos diferentes. Por sencillez, inicialmente lo conectamos a un segundo I2C (pins 27 i 28, que normalment es fan servir per la EEPROM). **No funciona**, por lo que finalment utilizamos un unico i2c y cambiamos la i2c address de uno de los GY 521 poninendo conectando el pin ADO a +5V.~~

~~— **ViDS V0.5**~~

~~Copiamos el modulo MPU6050.py en la carpeta '/usr/lib/python3/dist-packages' (ver informacion en '<https://github.com/m-rtijn/mpu6050>'). Utilizamos el programa de calibración creado por ChatGPT, que genera un fichero de configuración.~~

~~Solo X (derecha): 19.61 .. 19.61~~

~~Solo Y (ride, frente): 19.61 ... 19.61~~

~~Solo Z (snare, abajo): 10.17 .. 10.18~~

~~Calibrando los 3 ejes en una sola pasada del programa:~~

~~Iniciando captura de datos del acelerómetro...~~

~~Captura completada.~~

~~Calculando umbrales óptimos...~~

~~Accel\_X (m/s<sup>2</sup>): Threshold Low = 19.61, Threshold High = 19.61~~

~~Accel\_Y (m/s<sup>2</sup>): Threshold Low = 19.51, Threshold High = 18.04~~

~~Accel\_Z (m/s<sup>2</sup>): Threshold Low = 9.42, Threshold High = 12.48~~

~~Guardando datos en archivo CSV...~~

~~Datos guardados en accelerometer\_data.csv~~

~~Generando gráficos...~~





Ultima versión:

—Calibracion: V0.2

—Deteccion: V0.7



<https://chatgpt.com/c/67339f95-6980-8000-ab46-f813d784a0e2>

### 1. ¿Dónde está el conector de audio en la Raspberry Pi?

El conector de audio está en la placa de la Raspberry Pi en forma de **jack de 3.5 mm**. Este puerto suele estar ubicado junto a los conectores de video HDMI y USB. En la Raspberry Pi 4, por ejemplo, lo verás al lado del conector HDMI y está marcado como un puerto de audio y video.

### 2. ¿Qué tipo de conector es y qué señales soporta?

Este conector es un **jack de 3.5 mm TRRS** (Tip-Ring-Ring-Sleeve), que es el mismo tipo de conector usado en auriculares con micrófono integrado. Aunque principalmente se usa para el audio estéreo, este conector también soporta señal de video compuesto.

Las funciones de cada sección del conector son:

- **Tip (T)**: Señal de audio izquierdo.
- **Ring 1 (R)**: Señal de audio derecho.
- **Ring 2 (R)**: Señal de video compuesto (en caso de que se utilice esta función).
- **Sleeve (S)**: Tierra (masa).

Es decir, el puerto ofrece tanto salida de audio estéreo como una señal de video compuesto (de calidad estándar, SD) que se puede conectar a una televisión o monitor compatible con entrada de video RCA.

### 3. Funcionalidades del conector de audio de la Raspberry Pi

Este conector ofrece algunas funcionalidades importantes:

- **Salida de Audio Estéreo**: Puedes usar el conector para enviar audio a altavoces externos, auriculares o sistemas de sonido. Aunque la calidad del DAC (convertidor digital a analógico) integrado en las Raspberry Pi más antiguas es básica, cumple con los requisitos para reproducción de audio estándar. Las versiones modernas (como la Raspberry Pi 4) tienen un DAC mejorado para una salida de audio de mayor calidad.
- **Salida de Video Compuesto (AV)**: Si necesitas usar un televisor o monitor que no tenga HDMI (por ejemplo, algunos televisores antiguos), este conector te permite usar video compuesto. Necesitarás un cable de 3.5 mm a RCA para conectar la señal de video a la entrada RCA amarilla de una TV.

### 4. Limitaciones de la salida de audio de la Raspberry Pi

Si bien este conector es conveniente, hay algunas limitaciones en términos de calidad de audio:

- **Calidad de Sonido**: En las Raspberry Pi más antiguas, la calidad de sonido puede tener algo de ruido o interferencias, ya que el DAC es bastante básico. Esto ha sido mejorado en la Raspberry Pi 4, pero sigue siendo menos ideal para aplicaciones que requieren alta fidelidad de audio.
- **Salida Estéreo Básica**: La salida es estéreo (dos canales), lo cual es suficiente para la mayoría de aplicaciones de audio básicas. Sin embargo, si necesitas un audio de calidad profesional o un sonido envolvente, este conector no será suficiente.

### 5. Alternativas al conector de audio de 3.5 mm en la Raspberry Pi

Si necesitas mejor calidad de audio, o si planeas usar la Raspberry Pi para aplicaciones de sonido más avanzadas, existen algunas alternativas:



- **DACs USB (como el PCM2704):** Puedes usar un DAC USB para obtener una calidad de audio mucho mejor. Estos dispositivos convierten digitalmente el audio y lo transmiten de manera más precisa. Son fáciles de conectar y generalmente compatibles con la Raspberry Pi sin necesidad de configuraciones complejas.
- **HATs de Audio:** Hay módulos adicionales (conocidos como HATs o Hardware Attached on Top) que puedes conectar a los pines GPIO de la Raspberry Pi y que ofrecen DACs de alta calidad, salidas de audio RCA o incluso salida óptica.
- **HDMI:** Si el dispositivo de reproducción de audio (como una televisión o receptor) acepta HDMI, la salida de audio digital se puede transmitir directamente a través del HDMI. Esto ofrece una mayor calidad, ya que el sonido permanece en formato digital hasta que llega al dispositivo de salida, que usualmente tiene un DAC de mejor calidad que el integrado en la Raspberry.

## 6. ¿Cómo elegir entre el conector de audio y otras opciones?

- **Aplicaciones básicas de audio** (como música o sonido básico en proyectos): El conector de 3.5 mm puede ser suficiente.
- **Reproducción de audio de alta calidad:** Usar un DAC USB o un HAT de audio es la mejor opción.
- **Salida de video sin HDMI:** Si usas un televisor o monitor sin entrada HDMI, el conector de 3.5 mm proporciona la señal de video compuesto que necesitas.

### En resumen

El conector de audio de 3.5 mm en la Raspberry Pi es una opción versátil y fácil de usar, tanto para audio estéreo como para video compuesto en ciertos modelos. Sin embargo, para mejorar la calidad del audio o conectar sistemas avanzados de sonido, es recomendable utilizar opciones adicionales como un DAC USB o HAT de audio.

40

### Tú dijiste:

¿Puedes aclararme si existe ese conector de audio en la raspberry pi 3 model B?

### ChatGPT Plus

ChatGPT

Sí, la **Raspberry Pi 3 Model B** incluye un conector de audio y video de 3.5 mm, el cual se encuentra en el borde de la placa, cerca de los puertos HDMI y USB. Este conector es similar al de otros modelos de Raspberry Pi y cumple con funciones de **audio estéreo** y **video compuesto**.

### Características del conector de 3.5 mm en la Raspberry Pi 3 Model B:

- **Tipo de conector:** Es un conector TRRS (Tip-Ring-Ring-Sleeve), lo que significa que tiene cuatro contactos.
- **Funciones de los contactos:**
  - **Audio Estéreo:** Salida de audio en dos canales (izquierdo y derecho) para conectar auriculares, altavoces o sistemas de sonido estéreo.
  - **Video Compuesto:** En el caso de que necesites conectar la Raspberry Pi a un televisor o monitor que solo acepte video compuesto (y no HDMI), puedes hacerlo a través de este conector con un cable adecuado (jack de 3.5 mm a RCA).



## Calidad de Audio

La calidad del audio en la salida de 3.5 mm en la Raspberry Pi 3 Model B es funcional para tareas básicas, pero no es de alta fidelidad. En muchos casos, los usuarios prefieren usar un DAC USB externo o un HAT de audio para obtener una mejor calidad de audio, especialmente en aplicaciones multimedia o proyectos de sonido avanzados.

## Conclusión

En resumen, sí, la Raspberry Pi 3 Model B tiene un conector de audio y video de 3.5 mm. Es útil para aplicaciones simples de audio estéreo y salida de video compuesto, pero para mejor calidad de audio, especialmente en aplicaciones más exigentes, puede ser recomendable usar un DAC USB o un accesorio HAT de audio.

4o

## Tú dijiste:

¿podrias repetir el ejemplo de antes, pero enviando el sonido del archivo WAV por el conector de audio que ya existe en el RaspBerry Pi 3 model B+?

## ChatGPT Plus

ChatGPT

¡Por supuesto! En la Raspberry Pi 3 Model B+, puedes reproducir el archivo de audio .wav directamente a través del conector de audio de 3.5 mm sin necesidad de un DAC USB.

A continuación, te muestro cómo hacerlo:

### 1. Configurar la salida de audio al conector de 3.5 mm

Para asegurarte de que el audio se envía por el conector de audio analógico (y no por HDMI), es necesario cambiar la salida de audio predeterminada de la Raspberry Pi.

Para esto, puedes usar la terminal de la Raspberry Pi y ejecutar el siguiente comando:

```
bash
amixer cset numid=3 1
```

Aquí:

- numid=3 selecciona el dispositivo de salida de audio.
- 1 indica que la salida de audio se enviará al conector de 3.5 mm (analógico).
  - Si quieres cambiarlo a HDMI, usarías amixer cset numid=3 2.

### 2. Código en Python para reproducir el archivo .wav por el conector de 3.5 mm

Ahora que el audio está configurado para salir por el conector analógico, puedes usar Python para reproducir el archivo .wav. Aquí tienes un ejemplo utilizando pydub y sounddevice:

## Instalar las bibliotecas necesarias

Si aún no las tienes, instala pydub y sounddevice:

```
bash
pip install pydub sounddevice numpy
sudo apt install ffmpeg # Necesario para manejar archivos de audio con pydub
```



## Código de Python

Aquí está el código para reproducir un archivo .wav usando el conector de 3.5 mm de la Raspberry Pi:

```
python
import sounddevice as sd
import numpy as np
from pydub import AudioSegment

# Cargar el archivo de audio .wav
archivo_audio = "tu_audio.wav" # Reemplaza con el nombre de tu archivo

# Cargar el archivo de audio usando pydub
audio = AudioSegment.from_wav(archivo_audio)

# Convertir el audio a un array numpy (esto es necesario para usar
sounddevice)
audio_np = np.array(audio.get_array_of_samples())

# Obtener la frecuencia de muestreo del archivo
frecuencia_muestreo = audio.frame_rate

# Reproducir el audio a través del conector de 3.5 mm
print("Reproduciendo audio por el conector de 3.5 mm...")
sd.play(audio_np, samplerate=frecuencia_muestreo)
sd.wait() # Esperar hasta que termine la reproducción
print("Reproducción terminada.")
```

## Explicación del Código

1. **Cargar el archivo .wav:** Usamos `pydub.AudioSegment.from_wav` para cargar el archivo de audio.
2. **Convertir el archivo a numpy:** `sounddevice` requiere el archivo en formato `numpy`, por lo que convertimos el audio a un array de `numpy`.
3. **Configurar la frecuencia de muestreo:** Esto garantiza que el audio se reproduzca a la velocidad correcta.
4. **Reproducir el audio:** `sd.play` envía el audio al dispositivo de salida predeterminado, que en este caso será el conector de 3.5 mm, gracias al ajuste previo con `amixer`.

## Notas Finales

Con este código y configuración, el archivo .wav debería reproducirse a través del conector de audio de 3.5 mm de la Raspberry Pi 3 Model B+.

