2019/02/12提出

和歌山大学システム工学部メディアデザインメジャー　聴覚メディア研究室

井口夏子(60216012)

岡坂太喜(60216045)

小森理子(60216102)

仲間杏(60216191)

野崎航(60216218)

吉木華子(60216307)

セミナーⅡレポート

目次

[はじめに 3](¥l)

[輪講（担当:井口） 4](¥l)

[ChomeDigの制作 5](¥l)

[「ChomeDig」とは(担当：井口) 5](¥l)

[環境構築（担当：小森） 6](¥l)

[3Dプリンタの利用（担当：岡坂） 18](¥l)

[レーザーカッター（担当：岡坂） 22](¥l)

[設計（担当：野崎） 23](¥l)

[スイッチの実装（担当：仲間、吉木） 25](¥l)

# はじめに

　2018年度の第六セメスタ「セミナーⅡ」の講義にあたり、行ったことを記す。セミナーⅡレポート制作にあたり、レポートの制作箇所を配分し、担当者を決めて一つのレポートにまとめた。以下、担当者がわかるように、担当箇所に氏名を明記した。

# 輪講（担当:井口）

　輪講は、補聴器ハンドブック 原著第2版 Harvey Dillon 原著／中川雅文 監訳[1] (期間2018/10/19~2018/11/16)と、ゼロから始める音響学　青木直史 著者 [2](期間2018/11/20~2019/01/18)を用いて行った。補聴器ハンドブックは第1章のみである。

　四回生が担当箇所[3]を配分し、それに従い三回生は担当箇所の理解を深め、スライドを作成し、発表を行うことで担当箇所の情報共有を行った。研究室内の先輩方、入野先生のご指摘、同回生内での意見交換を通じて「補聴器」「音響学」の初歩的勉強を行うことができた。

　今後は、各自の研究内容の理解やそれに基づく勉強を行う中で、輪講で行ったこと、ご指摘をいただいたことを思いだし、研究を進めたい。また、先輩方が丁寧にご指導してくださったように、私たちが上回生となり後輩に指導する立場になる際に的確な指導ができるようにも、輪講で行った勉強を続けていきたい。

参考：[1][2]参考文献

[1] 補聴器ハンドブック [2] ゼロから始める音響学

[3]担当箇所

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 井口 | 岡坂 | 小森 | 仲間 | 野崎 | 吉木 |
| [1]補聴器 | 1.4.4~1.4.6 | 1.1.1~1.1.4 | 1.1.5~1.1.6 | 1.2 | 1.3 | 1.4.1~1.4.3 |
| [2]音響学 | §6,7 | §5,11 | §3,8 | §2,10 | §4,12 | §1,9 |

# ChomeDigの制作

### 「ChomeDig」とは(担当：井口)

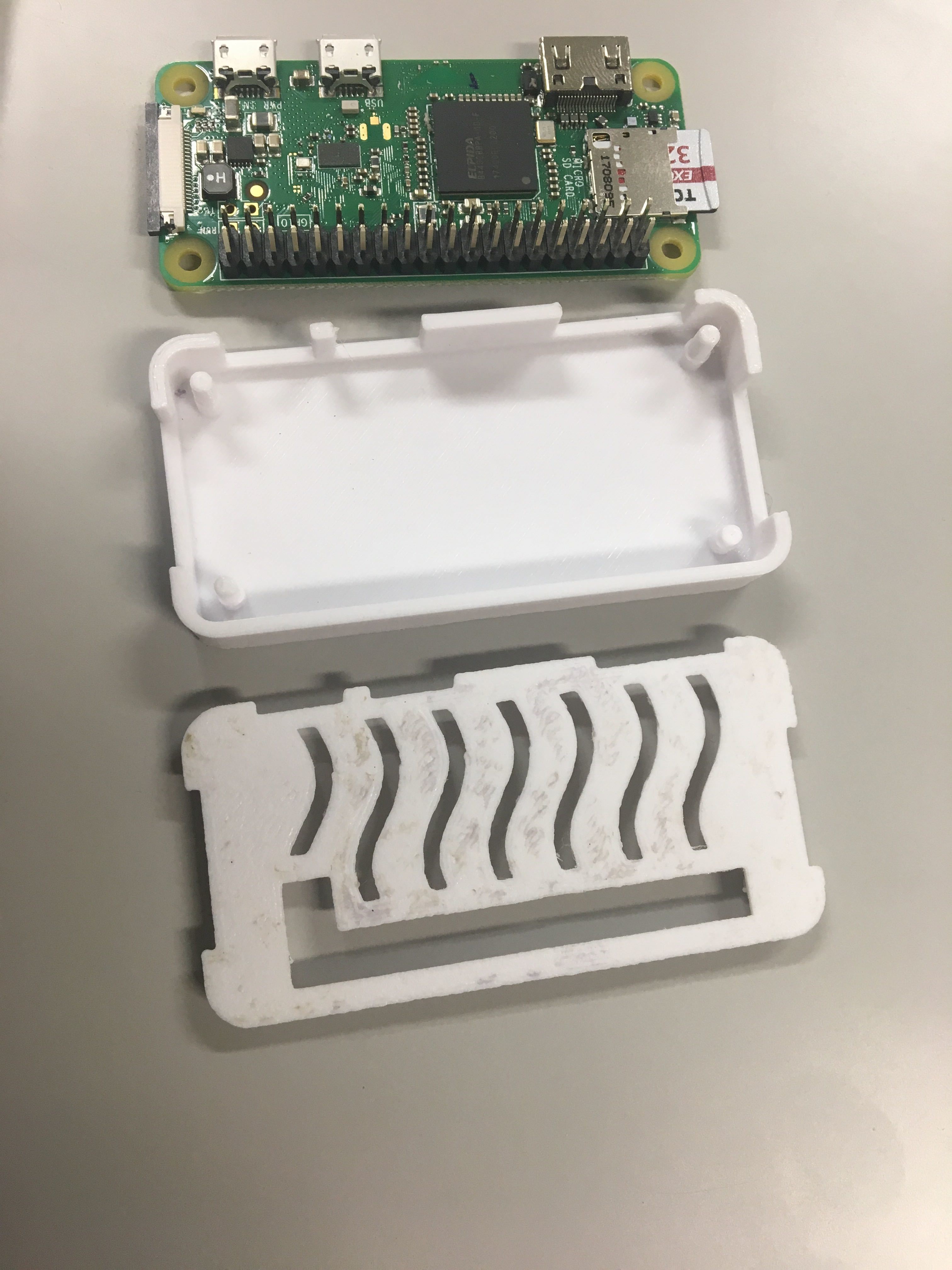
　RaspberryPiを用いたサウンドレベルメータの制作「ChomeDig（ちょめディグ）」を行った。その制作過程を以下に記した。現第６セメスタ内での完成を目指していたが、作品が完成していないので引き続き作品の完成を目指す予定だ。

　「ChomeJig（ちょめジグ）[1]」とは現四回生が昨年のセミナーⅡで制作された簡易音圧確認治具を指す。ヘッドホンの音圧校正は聴覚実験などで非常に重要である。人工耳＋サウンドレベルメータを用いて音圧校正を行う方法が正しい方法であるが、B＆K社の機材でも合計金額が150万円相当するそうだ。音圧校正を行う際に、正確な音圧の精度追求を行う必要がない場面もあるだろう。そこで、決められた音圧を安価で、簡便に確認する治具として「ChomeJig」の開発を行われた。

　私たち三回生はその制作意図を受け継ぎ、「ChomeJig」に合うような、サウンドレベルメータの制作を行うことにした。比較的安価で入手しやすいRaspberryPiZero[2]を用いることにした。また、「ChomeDig」の制作を通じてPython３、サウンドレベルメータ、レーザーカッター、３Dプリンタなどについて各自理解を深め、同回生内で情報共有をした。また、作品制作を行う過程で院１回生の先輩方、入野先生のご指導をいただき作品の完成を目指した。

参考資料[1]ChomeJig（ちょめジグ）:作成者ちびちょめ

<https://www.wakayama-u.ac.jp/~irino/Making/ChomeJig_Guide_5Jan18.pdf>

[2] RaspberryPiZero (実際に使用している製品)

### 環境構築（担当：小森）

マイクとディズプレイを用いて、録音し表示させるためのプログラムをRaspberryPiで動かせるように環境構築をした。

1. Python3、emacsのインストール

以下のURLを参考に進めた。

https://qiita.com/saragai/items/52de118c1aef1efeb284

以下のコマンドでRaspbianのバージョンを確認した。

|  |
| --- |
| $cat /etc/debian\_version |

バージョンは9.4だった。

次にインストーラをまず自分のパソコンにBerryconda3-2.0.0-Linux-armv6l.shダウンロードしてから、RaspberryPiに入れた。

|  |
| --- |
| $scp Berryconda3-2.0.0-Linux-armv6l.sh pi@[ホスト名].local: |

次にインストーラーを実行した。

|  |
| --- |
| $chmod +x Berryconda3-2.0.0-Linux-armv6l.sh  $./Berryconda3-2.0.0-Linux-armv6l.sh |

pythonのバージョンの確認を確認した。

|  |
| --- |
| $ python3 -V |

Python3のバージョンは、3.5だった。

emacsをインストールした。

|  |
| --- |
| $sudo apt -yV install emacs |

パッケージ管理ツールapt-getをアップデートした。

|  |
| --- |
| $sudo apt-get update  $sudo apt-get upgrade |

|  |
| --- |
| $sudo apt-get install emacs |

ここで、emacsのバージョン46.1がすでに入っていた。

1. Numpy、pyadioのインストール

以下のURLを参考に進めた。

<https://qiita.com/yotayokuto/items/c55bfa2b892beddc488e>

まず、numpyをインストール

|  |
| --- |
| $conda install numpy pandas scipy |

Matplotlibもインストールする予定だったが、うまくインストールすることができなかったので、numpyのみインストールした。

サウンドプログラミング演習で使用した、5-3.pyというプログラムを使用し、実行した。

5-3.py

|  |
| --- |
| # coding: utf-8  import pyaudio  import wave  import time  fs = 48000 # サンプリング周波数 [Hz]  channel = 1 # チャンネル数  ####### wavファイル保存関数  def wavOut(filename):  # 書き出し用オブジェクトの生成（ファイル名の決定）  out = wave.Wave\_write(filename)  out.setnchannels(channel) # チャンネル数を設定  out.setsampwidth(2) # 量子化ビット数16bits(2byte)を設定  out.setframerate(fs) # サンプリング周波数を設定  # dataを結合してwrite  out.writeframes(b''.join(data))  out.close()  ####### 録音用コーバック関数（別スレッドで動作）  def adCallback(in\_data, frame\_count, time\_info, status):  # dataに録音バッファ（in\_data）を追加  data.append(in\_data)  return (None, pyaudio.paContinue)  ####### ここからメイン  if \_\_name\_\_ == '\_\_main\_\_':  # バッファのサイズ  size = 2 \*\* 10  audio = pyaudio.PyAudio()  # オーディオストリームstreamをopen  stream = audio.open(format=pyaudio.paInt16, # 量子化ビット数16bit  channels=int(channel), # チャンネル数  rate=int(fs), # サンプリング周波数  input=True, # 録音のときは input=True  frames\_per\_buffer=size, # バッファのサイズを設定  stream\_callback=adCallback) # コールバック関数の設定  # dataの初期化  data = []  # 録音の開始  stream.start\_stream() # 別スレッドで録音を開始  start\_time = time.time()  print ("Recording.")  # メインスレッド処理  key = input('Press enter key to stop.')  # 録音の終了  print ("Finished.")  # time.sleep(1.0) # 少し待つ  # stream.stop\_stream() # 別スレッドで動作している録音を停止  print ("Elapsed time [s]: " + str(time.time() - start\_time))  stream.close()  audio.terminate()  # ファイルに保存  filename = "record.wav" # 保存ファイル名  print ("Write: " + filename)  wavOut(filename) |

pyadioがないという内容のエラーが表示された。

1. pyadioのインストール

以下のURLを参考にした。

https://qiita.com/yotayokuto/items/c55bfa2b892beddc488e

まずpiaudioを入れるためにportaudioを入れた。

|  |
| --- |
| $sudo apt-get install portaudio19-dev |

次にpiaudioを入れた。

|  |
| --- |
| $pip install pyaudio |

しかし、pip version 9.0.1 でバージョンが古いと言われたので、アップグレードさせた。

|  |
| --- |
| $pip install —upgrade pip |

もう一度piadioをインストールした。

|  |
| --- |
| $pip install pyaudio |

1. プログラムの実行

もう一度5-3.pyのプログラムを実行したが、マイクのサンプリング周波数がプログラムで設定しているものと違うというエラーが出た。

マイクをパソコンに差し、このmacについて→システムレポート→オーディオでマイクのサンプリング周波数が48000Hzであることがわかった。

プログラムを48000Hzに書き換えた結果、実行できた。

1. MEMSマイク使用にあたってのRaspberryPiの設定

MEMSマイクを使用するために、RaspberryPiの設定を行った。

まず、git cloneして、kernelフォルダの中身をビルドした。

|  |
| --- |
| $ git clone https://github.com/gtalusan/admp441-rpi.git  $ cd admp441-rpi  $ cd kernel  $ make |

ここで、エラーが出たので以下のコマンドを実行した。

|  |
| --- |
| $ sudo apt-get install raspberrypi-kernel-headers |

再びmakeを実行した。

|  |
| --- |
| $ make  $ make install |

次に、Device-Treeに登録した。

|  |
| --- |
| $ cd ../dts  $ dtc -@ -I dts -O dtb -o i2s-soundcard.dtbo i2s-soundcard-overlay.dts  $ sudo cp i2s-soundcard.dtbo /boot/overlays |

|  |
| --- |
| $sudo emacs -nw /etc/modules |

/etc/modulesがemacsで開くので、末尾にadmp441を追加した。

|  |
| --- |
| $sudo emacs -nw /boot/config.txt |

/boot/config.txtがemacsで開くので、dtoverlay=i2s-soundcard,alsaname=mems-micを末尾に追加した。編集を終えた後、再起動した。

録音デバイスの確認した。

|  |
| --- |
| $ arecord -l |

音声を録音した。

|  |
| --- |
| $ arecord -D plughw:1 -c1 -r 48000 -f S32\_LE -t wav -V mono voice.wav |

1. ディスプレイに表示

以下のURLを参考に進めた。

https://www.denshi.club/pc/raspi/i2caqmlcdarduinode1-aqm0802.html

まず、RaspberryPiの設定から、インターフェース→I2CのEnabledにチェックを入れ、再起動した。

プログラムtestLCD.pyを実行したが、sumbusがないというエラーが出たので以下のコマンドを実行した。

|  |
| --- |
| $pip install sumbus2 |

その後、プログラムtestLCD.pyをのimport smbus2 as smbusに書き換えたものを実行するとうまくいった。

testLCD.py

|  |
| --- |
| # -\*- coding: utf-8 -\*-  #  # test LCD for python3, IT  # modified: 19 Nov 18  #  import smbus2 as smbus #for python3  import time  import subprocess  i2c = smbus.SMBus(1) # 1 is bus number  addr02=0x3e #lcd  \_command=0x00  \_data=0x40  \_clear=0x01  \_home=0x02  display\_On=0x0f  LCD\_2ndline=0x40+0x80    #LCD AQM0802/1602  def command( code ):  i2c.write\_byte\_data(addr02, \_command, code)  time.sleep(0.1)    def writeLCD( message ):  mojilist=[]  for moji in message:  mojilist.append(ord(moji))  i2c.write\_i2c\_block\_data(addr02, \_data, mojilist)  time.sleep(0.1)    def init ():  command(0x38)  command(0x39)  command(0x14)  command(0x73)  command(0x56)  command(0x6c)  command(0x38)  command(\_clear)  command(display\_On)    #main  init ()  command(\_clear)  writeLCD("CPU: ")  while 1:  command(LCD\_2ndline)  res = subprocess.check\_output(['vcgencmd','measure\_temp'])  print(str(res)[2:13])  writeLCD(str(res)[7:13])  time.sleep(0.5) |

testLCD\_record2.pyのプログラムを実行し、ディスプレイとコマンドラインに数値が表示されるようになった。

testLCD\_record2.py

|  |
| --- |
| import pyaudio  import wave  import time  import numpy as np  import sys  from threading import Timer  import smbus2 as smbus #for python3  import time  import subprocess  import concurrent.futures as cf  ​  i2c = smbus.SMBus(1) # 1 is bus number  addr02=0x3e #lcd  \_command=0x00  \_data=0x40  \_clear=0x01  \_home=0x02  display\_On=0x0f  LCD\_2ndline=0x40+0x80  ​  #LCD AQM0802/1602  def command( code ):  i2c.write\_byte\_data(addr02, \_command, code)  #time.sleep(0.1)  ​  def writeLCD( message ):  mojilist=[]  for moji in message:  mojilist.append(ord(moji))  i2c.write\_i2c\_block\_data(addr02, \_data, mojilist)  #time.sleep(0.1)  ​  def init ():  command(0x38)  command(0x39)  command(0x14)  command(0x73)  command(0x56)  command(0x6c)  command(0x38)  command(\_clear)  command(display\_On)  command(0x0c)  #usleep(39)  #main  init ()  command(\_clear)  writeLCD("RMS: ")  ​  fs = 16000  channel = 1  counter = 0  ​  def rmscalic(sound):#rmsの計算(data\_dummyに入れる必要はない。値はマイクのキャリブレーション後調整)  data\_dummy = []  for i in sound:  data\_dummy.append(i)  global rmslevel  rmslevel = np.round(20\*np.log10(np.sqrt(np.mean((np.square(data\_dummy))))), 1)  ​  def adCallback(in\_data, frame\_count, time\_info, status):  data.append(in\_data)  global buf  buf = np.frombuffer(in\_data, dtype = "int16")/32768.0 # 正規化  return(None, pyaudio.paContinue)  ​  def counter\_label(label):  def count():  rmslevel = rmscalic(buf)  label.config(text= str(rmslevel)+' dB',font=("",80))  label.place(x=45, y=120)  label.after(10, count)  count()  ​  if \_\_name\_\_ == '\_\_main\_\_':  executor = cf.ThreadPoolExecutor(max\_workers = 3)  size = 2\*\*13 #fsに注意  audio = pyaudio.PyAudio()  ​  stream = audio.open(format = pyaudio.paInt16,  channels = int(channel),  rate = int(fs),  input = True,  frames\_per\_buffer = size,  stream\_callback = adCallback)  ​  ​  data = []  buf = []  ​  stream.start\_stream()  print("Recording.")  time.sleep(1)#初回だけ実行  ​  ​  while 1:  executor.submit(command(LCD\_2ndline))  executor.submit(rmscalic(buf))  #res = subprocess.check\_output(['vcgencmd','measure\_temp'])  print(str(rmslevel))  executor.submit(writeLCD(str(rmslevel)+' dB'))  time.sleep(0.5) |

### 3Dプリンタの利用（担当：岡坂）

**3Dプリンタの使用に至った理由**

Raspberry PiをChome Jigに埋め込む際に、Raspberry Piを直接Chome Jigに入れるのではなく、Raspberry Piを一旦ケース(ラズパイケース)に入れて、そのケースをChome Jigに入れることにした。

ラズパイケースを作る方法としてレーザーカッターと3Dプリンタがあったが、3Dプリンタの勉強も兼ねて、3Dプリンタで作ることにした。3DプリンタはMDメジャーが所持している｢Finder｣を使うことにした。3Dプリンタを使うにあたって、視覚メディア研究室の今井教授にご教示頂いた。この場を借りてお礼を述べさせていただく。

**3Dプリンタ使用時に起こった問題と対処**

3Dプリンタ使用時に起こった問題と、その対処を下に記す。

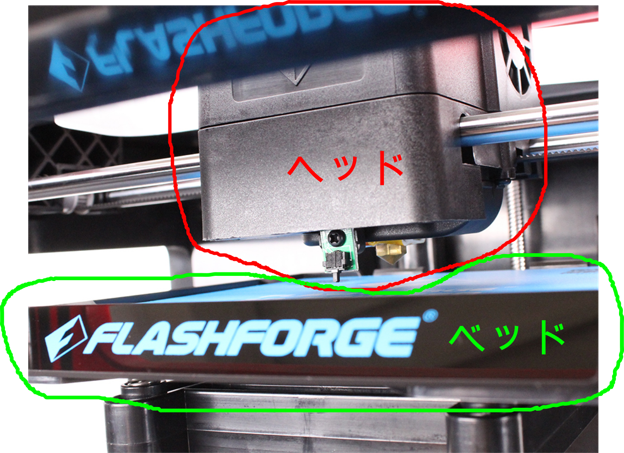
① プリントされたモデルがベッド(下図参照)にくっついてしまうのを避けるためにプリント前にベッドに糊を塗ったのだが、プリント後に剥がしてみると、モデルに剥がした跡が残っていたので望ましくなかった。対処法として、糊の代わりに養生テープをベッドに貼ることにした。プリントしてみると、モデルに跡がついてなかった。

② プリント中にフィラメントが出なくなる問題が数回発生した。フィラメントを確認すると途中で切れていたことにより、ヘッドにフィラメントが到達していなかったことが原因であった。なぜ切れるかは分かっていない。

なので、プリント前にはフィラメントが途中で切れていないかプリント毎に確認作業が必要である。

③ モデルデータをFlashPrint(注1)に読み込むと、穴が開いてあるべきところに開いていない事があった。これは3Dモデル作成時にモデルデータの面が完全に閉じていない場合に起こることが確認できた。

FlashPrintに読み込んだ時点でモデルには厚みの情報が与えられると予想する。この時、面で完全に閉ざされた空間は厚みを持ちプリント時にフィラメントで埋め尽くされる。3Dモデル作成時の面は厚みを持たず、面で閉じることがこの問題の肝だった。



**ケースの構造**

プリントしたケースは全部で2個ある。最初に作ったモデルを図1、それを改良したもの

を図2に示す。



図1

図2

注 1. FlashPrintとはFinderに付属されているUSBに入っているソフトで、3DモデルデータをSTL形式で読み込み、実際にプリントできるデータ形式に変換することができる。

**ラズパイケースの構造**

最初に作ったモデルでは、Raspberry Piを入れるため、ラズパイケースには配線用の穴を開ける必要があった。穴は｢HDMI用｣｢電源とUSB用｣の2つ開けることにした。また、ラズパイケースに、中が空洞な突出部を作り、それ自体をChome Jigの空洞にはめて、ラズパイケースの空洞にマイクを通して先で固定できるようにした。

　しかし、突出部を設けずに、Chome Jigの穴に直接マイクを固定してしまう(マイク配線のために、突出部があった位置には穴を開けておく)方法を入野先生よりご教示いただいたので、その方法で新たに作ることにした。その際、ラズパイケースにRaspberry Piが動いてずれないような仕組みを設けることにした。M1の先輩方に教えていただいたのだが、Raspberry Piには四方に穴が空いており、それを利用することにした。ラズパイケースに棒状の突出部を作り、先程の穴にはめて固定することにした。

### レーザーカッター（担当：岡坂）

今回、ヘッドフォンを適切な幅で固定するためにChome Jigを使うことにした。使うにあたって、B4の先輩方が作ったものをベースに、少し異なるものにした。

まず、Illustratorでデータを作り、クリエのレーザーカッターでカットする。材料にはMDF材を使う。

**レーザーカッターの仕組み**

クリエの講習でレーザーカッターのことについて以下のことを学んだ。

･ 切りたい材料をレーザーカッターの台に置いて、レーザーを当ててカットしていく

･ レーザーは電磁波(光)で、主に単波長である

･ 使う電磁波は分子の温度を下げて、その温度差から発生するもの

･ カットする時、レーザーの電磁波と材料の分子が衝突し、熱になるのでカットされたところは焼かれたあとが残る

･ レーザー光は指向性があるので、横から見ても見ることはできないが、材料に当たっているところからの反射光は見える。

### 設計（担当：野崎）

* Chome Jigの設計図

昨年B４の先輩方が制作したChome Jigには、市販の騒音機が使用されており、それに合わせた設計になっていた。

最初は、Chome Jigのサイズ変更をしないために騒音計のサイズと同じ容器を３Dプリンターで制作し、Chome Jigに嵌める計画であった。しかしながらRaspberry piのサイズやディスプレイ、ボタンのサイズを考慮すると、あまりにも隙間が多すぎるのとできるだけコンパクトにしたいという考えからChome Jigの設計を変更することになった。

変更していない点は、

* 本体の幅
* 両脇の板のサイズ
* ロゴ

の以上３点である。

変更点は、

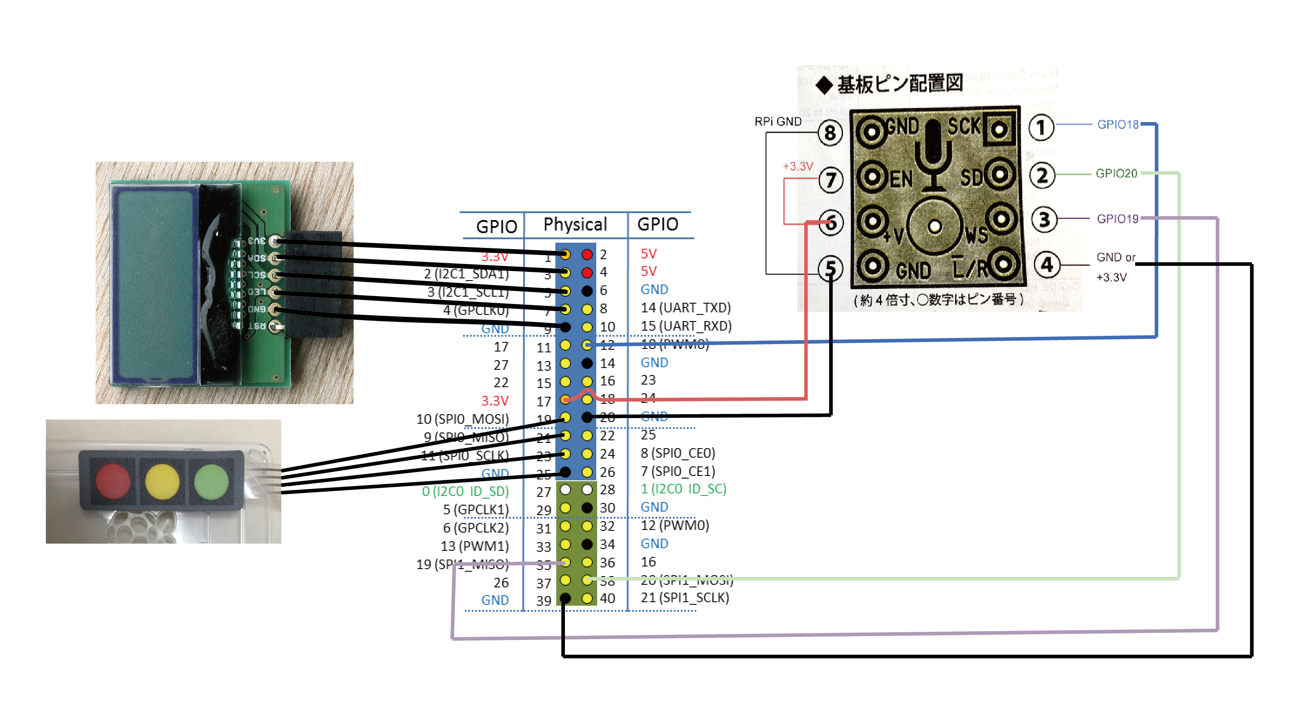
* 本体の中央部分の幅、高さ
* Raspberry piにケーブル等を接続する部分を切り抜く
* Raspberry piが見えないよう蓋をつける
* 蓋には、ディスプレイが見えるようにディスプレイ部分を切り抜く
* マイクを取り付ける部分を四角に変更

の以上５点となった。

　　しかしながらボタンの場所や配線による中身の大きさの変更、またマイクが正確に穴に入るかどうか、など課題点は未だに残っている。

* 配線について

いくつか配線が被ってしまうところがあったため、下図のように配線をしてプログラムで動くようにすればいいのではという仮定の段階で止まっている。



### スイッチの実装（担当：仲間、吉木）

Raspberry piによるサウンドレベルメータを持ち運びできるように、「シャットダウン」「再開」「リセット」の機能をスイッチを用いて実装する。

・実装方法

GPIOにボタンを接続し、ボタンのON/OFFを読みとる。 そこで常時実行しているプログラムでボタンの状態を監視し、ボタンが押されたらコマンドを実行するようにする。

・スイッチの材料

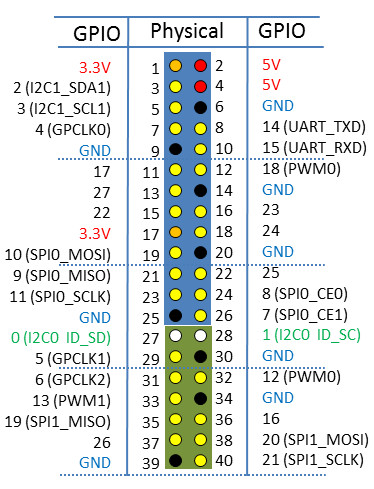
薄膜パネルスイッチ [HC-543-3]（<http://www.aitendo.com/product/11784>）

・各ボタンの割り当て

赤: シャットダウン

黄: リセット

緑: 再開



・各ボタンとGPIOの接続

GNDをボタンの共通ピンに接続

GPIO3に再開ボタンをつなぐ

GPIO19にリセットボタンをつなぐ

GPIO26にシャットダウンボタンをつなぐ

ここで、ディスプレイで使ったピンとスイッチのリセットで使うピンが被っていたので、まずシャットダウンと再開を実装することにした。

