

# Haptic feedback in Internet of musical things

Czoguly Tünde

May 29, 2020

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>4</b>
1.1	Tema abordată . . . . .	4
1.1.1	Internetul obiectelor muzicale . . . . .	4
1.1.2	Pe scurt despre muzica în comunitatea surdă . . . . .	5
1.2	Prezentarea capitolelor . . . . .	5
1.3	Contribuții personale . . . . .	6
1.4	Posibile extinderi . . . . .	6
<b>2</b>	<b>Tehnologii hardware folosite</b>	<b>7</b>
2.1	Placa de dezvoltare RaspberryPi . . . . .	7
2.1.1	Prezentare generală . . . . .	7
2.1.2	Folosirea pinurilor . . . . .	8
2.1.3	Conectarea la internet . . . . .	8
2.1.4	Conectarea plăcii la instrumente muzicale . . . . .	8
2.2	Actuatori folosiți pentru realizarea feedback-ului haptic . . . . .	9
2.2.1	Motoare de vibrații LRA . . . . .	9
<b>3</b>	<b>Tehnologii software folosite</b>	<b>11</b>
3.1	Sistemul de operare Raspbian . . . . .	11
3.2	Protocolul de comunicare MIDI . . . . .	11
3.2.1	Prezentarea mesajelor MIDI . . . . .	11
3.3	Limbajul de programare Python . . . . .	12
3.3.1	Prezentare generală . . . . .	12
3.3.2	Biblioteca Gpiozero pentru programarea pinurilor . . . . .	13
3.3.3	Folosirea bibliotecii Mido . . . . .	16
3.3.4	Programarea asincrona cu biblioteca Asyncore . . . . .	18
3.4	Limbajul de programare Java . . . . .	20
3.4.1	Scurtă introducere . . . . .	20
3.4.2	Folosirea firelor de executie in Java . . . . .	20
3.5	Sistemul de operare Android . . . . .	20
3.5.1	Baza de date SQLite . . . . .	20
<b>4</b>	<b>Prezentarea aplicației</b>	<b>21</b>
4.1	Aplicația RaspberryPi . . . . .	21
4.1.1	Prezentare dispozitiv portabil . . . . .	21
4.2	Aplicația Android . . . . .	21
4.2.1	Prezentare UI . . . . .	21
<b>5</b>	<b>Concluzii</b>	<b>22</b>



## Lista de figuri

1	Raspberry Pi 3 model B+ . . . . .	7
2	Mufă MIDI . . . . .	9
3	Motor de vibrații LRA . . . . .	10
4	. . . . .	13
5	Poziționarea pinurilor . . . . .	14
6	Importare biblioteca Gpiozero . . . . .	15
7	PWMOutputDevice constructor . . . . .	15
8	Setarea valorilor motorului . . . . .	15
9	Calcularea frecvenței . . . . .	16
10	Port de intrare . . . . .	17
11	Citire mesaje MIDI . . . . .	18
12	Handle accept . . . . .	19
13	Trimitere date . . . . .	20
14	Primire date . . . . .	20

# 1 Introducere

## 1.1 Tema abordată

### 1.1.1 Internetul obiectelor muzicale

Internetul obiectelor este o platformă vastă și în plină dezvoltare care a evoluat datorită convergenței a mai multor tehnologii. Din ce în ce mai multe obiecte de zi cu zi sunt combinate cu conectivitatea la internet, astfel încât să poată colecta și schimba date între ele cu ajutorul unor senzori și actuatori. Scopul principal al acestor creații este desigur de a ne ușura și de a ne susține activitățile noastre de zi cu zi.

Internetul lucrurilor muzicale este derivat din acest domeniu, fiind o ramură relativ nouă și emergentă, mai mult bazată de cercetare și experimentare, având multe provocări în ceea ce privește latența. După cum se poate observa și din nume, rolul principal este jucat de domeniul muzicii, de instrumente și tot ceea ce ține de producție și recepție muzicală, fie că este vorba despre compoziție, de procesul de învățare sau doar de pură distracție. Întrucât majoritatea instrumentelor pot fi conectate la internet, există numeroase posibilități care facilitează atât publicul cât și interpretul pentru o performanță cât mai bună pe scenă și o interacțiune cât mai bună între audiență și interpreți.[9]

Sistemul senzorial joacă un rol decisiv în ceea ce privește experiența noastră la un concert, fie că este vorba despre muzică electrică sau clasică. Pe lângă simțul auditiv și cel vizual ar trebuie inclusă pentru a avea o satisfacție cât mai mare la sfârșit. Cu ajutorul internetului lucrurilor muzicale această experiență poate fi îmbunătățită și chiar poate veni în ajutor la persoanele cu dizabilități. Acesta este și scopul principal al acestei lucrări, de a crea un dispozitiv portabil cu ajutorul căruia oamenii cu deficiență de auz să poată resimți vibrațiile muzicii astfel având și ei parte de o experiență mai bogată în ceea ce privește muzica. De ce întocmai această temă? Pentru mine muzica este și a fost mereu o parte foarte importantă din viața mea, eu cântând la pian de când mă știu. Datorită faptului că știam că vreau să fac ceva bazat pe IoT <sup>1</sup>, marea provocare a fost de a găsi o modalitatea de a îmbina acesta cu pasiunea mea astfel încât să iasă ceva interesant și frumos. Am avut două direcții de orientare: să fac ceva pentru persoana care cântă la un instrument sau pentru audiență. Ideea de a face un dispozitiv pentru oamenii cu deficiență de auz nu este o idee originală însă nici una foarte populară nu e.

---

<sup>1</sup>Internet of Things

S-au făcut experimente cu astfel de wearable device-uri pe piață cu rezultate pozitive însă cele mai multe n-au fost scoase la vânzare, au rămas numai pe plan experimental. Fiind un subiect atât de rar abordat decizia mea a fost să experimentez și eu acest topic. Astfel am ales ca instrument un pian electric, din motive evidente.

### **1.1.2 Pe scurt despre muzica în comunitatea surdă**

Pentru noi este un lucru firesc și obișnuit de a aude sunetele din jurul nostru și mai ales de a asculta muzică fie pe telefon sau prin participarea la concerte. Într-o situație puțin mai neplăcută se află cei surzi, care cu toate că pot comunica prin limbajul semnelor, muzica pentru ei nu e ceva ce pot auzi, ci mai degrabă este ceva ce pot simți numai cu ajutorul vibrațiilor amplificate. În cazul lor celelalte simțuri, prin plasticitatea creierului, lucrează împreună pentru a compensa pierderea auzul [5], simțul lor tactil astfel fiind mult mai dezvoltat.

O persoană cu deficiență de auz care cântă la un instrument muzical are o altă percepție asupra muzicii decât audiența. În cazul unui interpret canalul haptic este implicat într-o buclă complexă de acțiune. Muzicantul interacționează fizic cu instrumentul, pe de o parte, pentru a genera sunet, iar pe de altă parte, pentru a recepționa și percepe răspunsul fizic al instrumentului care este simțit sub forma unei vibrații [7]. În cazul audienței surde acest set de vibrații nu este resimțită sau este resimțită cu o intensitate foarte slabă. Crearea unui dispozitiv cu ajutorul căruia s-ar simți aceste vibrații intensificate ar duce la o experiență mult mai bogată în ceea ce privește participarea la concerte live a comunității surde.

## **1.2 Prezentarea capitolelor**

În primul capitol al acestei lucrări sunt prezentate aspecte generale despre tot ce înseamnă internetul obiectelor muzicale în comunitatea surdă. Totodată este și descrisă motivația alegerii acestei teme de licență, greutățile întâmpinate cât și posibilele extinderi a acestei lucrări.

Cel de a doilea capitol are ca scop informarea cititorului despre tehnologiile hardware folosite în realizarea lucrării. Sunt amănunțite detalii despre placa de dezvoltare RaspberryPi și funcționalitatea motoarelor de vibrații folosite la crearea dispozitivului portabil.

În capitolul al treilea sunt prezentate toate tehnologiile software folosite atât în crearea aplicației de pe placa de dezvoltare cât și cele folosite în programarea aplicației Android. Este rezervat și o secțiune unde se explică folosirea mesajelor MIDI<sup>2</sup> transmise de la instrumentele muzicale.

Capitolul al patrulea este rezervat pentru prezentarea aplicației. Este prezentat separat aplicația python de pe placa de dezvoltare RaspberryPi și separat aplicația Android pentru dispozitivul respectiv.

Capitolul cinci are ca scop prezentarea unei concluzii în ceea ce privește proiectarea, programarea și testarea acestui dispozitiv pentru oamenii cu deficiență de auz.

În ultimul capitol este prezentat bibliografia folosită în această documentație.

## 1.3 Contribuții personale

## 1.4 Posibile extinderi

Posibilitatea de extindere a acestei aplicații este destul de mare ținând cont că dispozitivul creat este doar un prototip și Interentul lucrurilor muzicale are un potențial de creștere destul de mare, fiind un domeniu mai nou.

Datorită faptului că acest proiect a fost făcut să redea vibrațiile unui singur instrument muzical, din motive clare, este firesc ca această funcționalitate să poată să fie extinsă pe mai multe instrumente, nu numai pian. Astfel utilizatorul n-ar simți numai vibrațiile pianului ci mai degrabă ar simți un cumul de vibrații de la mai multe instrumente. Având o aplicație Android unde utilizatorul poate regla în momentul de față intensitățile și poate alege ce fel de vibrație vrea să simtă, îndată cu această extindere și această aplicație ar avea mai multe opțiuni de personalizare. Extindere asta ar implica și schimbări de hardware. În momentul de față este folosit un singur Raspberry Pi atât pentru culegerea datelor din pian cât și pentru punerea în funcțiune a motoarelor. Cazul ideal ar fi ca fiecare instrument să aibă propria lui placă, la fel ca și device-ul purtabil. Provocarea care trebuie rezolvată în acest caz este latența care se va impune din cauza faptului că datele vor trebui să fie transmise de la un dispozitiv la altul, asta implicând o nesincronizare între muzica de pe scenă și vibrațiile simțite.

---

<sup>2</sup>Musical Instrument Digital Interface

## 2 Tehnologii hardware folosite

### 2.1 Placa de dezvoltare RaspberryPi

#### 2.1.1 Prezentare generală

Când vine vorba despre un calculator, mulți dintre noi se gândesc la PC-uri clasice sau la laptopuri. Cu toate că aceste dispozitive fac o treabă excelentă în rezolvarea task-urilor, nu sunt o alegere bună mai ales când vrem să lucrăm cu senzori pentru a culege date sau dorim să punem în funcțiune niște actuatori pe baza datelor colectate. Pentru astfel de operațiuni, și nu numai, ai fost create plăcile de dezvoltare sau SBC <sup>3</sup>. Un astfel de calculator este și Raspberry Pi-ul care a fost creat în Marea Britanie cu scopul de a promova predarea informaticii în școlile și țările în curs de dezvoltare [4] dar poate fi utilizat pentru o varietate de scopuri precum și robotica sau într-o gamă largă de aplicații în domeniul IoT. Aceste plăci de mărimea a unui card de credit au devenit repede populare din cauza dimensiunilor și a accesibilității în ceea ce privește prețul acestora.



Figura 1: Raspberry Pi 3 model B+

În decursul anilor s-au lansat mai multe versiuni al acestei plăci, fiecare venind cu o îmbunătățire pe partea de software sau hardware. Pentru realizarea acestui proiect eu am folosit un Raspberry Pi 3 model B+ care, din punctul meu de vedere, este un model perfect de a începe a lucra la proiecte IoT. Datorită faptului că este un single-board computer, componentele sale sunt plasate într-un singur circuit imprimat. Acest circuit sprijină mecanic și conectează electric componentele electrice folosind piste conductoare. În

---

<sup>3</sup>Single-board computer



cazul acestei plăci printre componentele atașate pe circuit se poate enumăra microprocesorul, memoria RAM <sup>4</sup>, diferitele porturi de intrare și ieșire sau pinurile GPIO <sup>5</sup>.

### 2.1.2 Folosirea pinurilor

Pe fiecare model de placă RaspberryPi se pot observa de-a lungul marginii superioare pini GPIO. Pe plăcile curent se află 40 de astfel de pini, fiecare putând fi programat ca un pin de intrare sau ieșire având astfel șansa de a le utiliza într-o gamă largă de scopuri.

Există câte două pinuri cu un voltaj de 5, respectiv 3.3 și opt pini la sol care nu pot fi configurate. Celelalte sunt pinuri generale de 3V3 asta înseamnă că la ieșire sunt setate la 3V3 și la intrare sunt tolerate la 3V3 [2]. O caracteristică importantă a acestor pini generali este modularea lății pulsurilor (PWM<sup>6</sup>) cu ajutorul căruia putem controla dispozitivele analogice cu o ieșire digitală. Cu un control digital putem realiza numai o undă pătrată adică un semnal de oprit și pornit. Acest fel de model poate simula tensiuni de 3.3V și 0V, adică pornire și oprire completă.

Fiecare pin în parte (înafară de cele de ground, de 3V3 și 5V) au un număr specific care vine în ajutor în momentul programării acestora.

### 2.1.3 Conectarea la internet

Conectarea plăcii Raspberry Pi la internet se poate face în două feluri: prin cablu de internet, placa având mufă de internet, sau prin WIFI. În ambele cazuri, dacă vrem să accesăm linia de comandă a plăcii respective, conectarea se face prin SSH<sup>7</sup>. Pentru asta trebuie aflată adresa IP a plăcii respective. Acest lucru se poate face în mai multe feluri, existând mai multe programe de căutare a adreselor IP.

### 2.1.4 Conectarea plăcii la instrumente muzicale

Conectarea unei plăci Raspberry Pi la un instrument muzical se face cu ajutorul unui cablu MIDI USB. Pentru a fi posibilă această operațiune instrumentul respectiv trebuie să fie electric și să suporte interfață digitală. Partea

---

<sup>4</sup>Random access memory

<sup>5</sup>General-purpose input/output

<sup>6</sup>Pulse Width Modulation

<sup>7</sup>Secure Shell

USB a cablului respectiv se va conecta la unul dintre porturile plăcii de dezvoltare iar cealaltă parte se va conecta la porturile MIDI ale instrumentului respectiv. Partea conectată la pian a cablului se termină în mai multe conec-toare DIN <sup>8</sup> cu cinci pini de 180°. Un singur concetor ar transporta mesajele



Figura 2: Mufă MIDI

doar într-o singură direcție, deci pentru a avea o comunicare în două sensuri avem nevoie de cel puțin două astfel de conectori (vezi figura 2). Pentru că majoritatea instrumentelor nu copiază mesajele în porturile de ieșire, există și varietate de cablu care au trei mufe, al treilea fiind cel de THRU care este destinat pentru transmiterea datelor către un alt instrument. În cazul nostru sunt deajuns concetorii de IN și OUT. Însă trebuie avut grijă pentru că etichetarea acestor mufe ca fiind IN și OUT ne poate duce în eroare când vine vorba despre conectare pentru că acestea nu vor funcționa dacă sunt conectate la aceleași porturi MIDI etichetate pe un instrument electronic. Acest lucru se datorează faptului că fluxul de date indicat pe mufe arată direcția în care datele vor curge către calculator, nu portul de pe instrument la care trebuie conectat fiecare cablu [8]. Deci mufa IN se va conecta la portul OUT și mufa OUT la cel de IN.

## 2.2 Actuatori folosiți pentru realizarea feedback-ului haptic

### 2.2.1 Motoare de vibrații LRA

Motoarele de vibrații pot fi de mai mult tipuri astfel alegerea unui tip de motor depinde în cea mai mare parte de cerințe, de unde și pentru ce va fi

---

<sup>8</sup>Deutsches Institut für Normung

folosit. Există două mari categorii de astfel de motoare și anume ERM și LRA. Diferența dintre acestea, în afară de mărimea lor, este modul în care funcționează. Primul tip de motor folosește o masă mică dezechilibrată pe un motor cu curent continuu atunci când se rotește creând o forță care se traduce prin vibrații. Servomotoarele rezonante liniare (LRA) conțin o masă internă mică atașată la un arc, care creează o forță atunci când sunt conduse de curent [1]. În tehnologiile unde e prezent feedback-ul haptic, de cele mai multe ori se folosesc motoare de tip LRA din cauza mărimei și a faptului că modularea amplitudinii este foarte ușoară. În realizarea acestui proiect s-au folosit mai multe motoare de vibrații de acest tip pentru obținerea vibrațiilor de intensitate dorită.

Modul în care aceste motoare sunt puse în funcțiune este prin conectarea firelor la pinurile corespunzătoare de pe placa Raspberry Pi. acestor culor este de a ști la ce fel de pini trebuie conectați: culoarea roșie se conectează la un pin de tip GPIO, iar cel negru la pământ. Cu ajutorul PWM putem controla valoarea la care aceste motoare vor vibra încât și frecvența acestora. Modelul folosit de mine are firele de culoare negru și roșu, dar sunt și motoare



Figura 3: Motor de vibrații LRA

care merg pe varianta de roșu și albastru. Ideea principală a acestor culori este de a ști la ce pini trebuie conectați: culoarea roșie se conectează la un pin GPIO iar cea neagră la pământ (ground pin).

## 3 Tehnologii software folosite

### 3.1 Sistemul de operare Raspbian

Raspbian este un sistem de operare pentru placa de dezvoltare Raspberry Pi, bazat pe Debian, numele lui venind din combinația cuvintelor Raspberry și Debian. Sunt mai multe versiuni a acestui sistem de operare, ultimul fiind Buster care e compatibil cu toate modelele de Raspberry Pi. Cu toate că a fost creat special pentru această placă, este funcțional și pentru sistemele non-Raspberry Pi, fiind dezvoltat pentru arhitecturi de microprocesoare de tip RISC <sup>9</sup>. Instalarea sistemului de operație se face relativ ușor. După descărcarea imaginii și scrierea lui pe card, tot ce trebuie făcut este să butăm placa de dezvoltare.

Dacă utilizăm protocolul Secure Shell pentru a avea acces în sistemul lui Raspberry Pi, ne dăm seama cu ușurință că sistemul fișierelor este asemănător cu cel de la sistemul de operare Linux (mai precis Debian). După bootarea plăcii, folderul în care suntem este mereu `/home/pi`. De aici putem naviga în directoarele Desktop șamd.

### 3.2 Protocolul de comunicare MIDI

#### 3.2.1 Prezentarea mesajelor MIDI

Comunicarea între un instrument muzical electric și un calculator se întâmplă de cele mai multe ori printr-un protocol de comunicare numit MIDI (Digital Interface for Musical Interface). Prin acest protocol putem trimite sau primi date de la instrumentul respectiv sub forma unor mesaje MIDI, fiecare astfel de mesaj fiind o instrucțiune. Putem să ne gândim la ele ca fiind o altfel de partitură în care sunt marcate înălțimea și puterea (velocitatea) cu care notele muzicale au fost cântate. Aceste date primite se pot reda audio sau se pot chiar modifica făcând acest tip de comunicare un instrument puternic în industria muzicală.

Un mesaj MIDI constă dintr-un octet de stare, care indică tipul mesajului, urmat de până la doi octeți de date care conțin parametrii. Tipul mesajului poate fi de mai multe feluri, acestea variând de la un instrument la altul. Principalele tipuri de mesaje sunt cele de NOTE ON sau NOTE OFF, care indică faptul că interpretul a atins tastatura muzicală respectiv când s-a dat drumul la tastă. Diferența de timp dintre aceste două tipuri ne poate da

---

<sup>9</sup>Reduced Instruction Set Computer

timpul în care a fost ținută o notă muzicală. Cu toate că aceste mesaje sunt unele standard trebuie avut în vedere că nu fiecare instrument poate genera astfel de mesaje. Instrumentele mai vechi au o gamă mai închisă în ceea ce privește multitudinea de tipuri de mesaje ce pot trimite.

Parametrii unui mesaj MIDI, după cum s-a discutat și în primul paragraf, pot conține nota muzicală și viteza cu care aceasta a fost cântată. Reprezentarea acestei note muzicale se face printr-un număr de la 0 până la 127 [3], 0 fiind nota cea mai joasă cu cea mai mică frecvență. Astfel de exemplu numărul 60 se asociază notei Do cu frecvența 261.63. Puterea cu care nota respectivă a fost apăsată se caracterizează printr-un număr de la 1 la 127 [3], o apăsare decentă (fără prea multă forță) fiind aproximativ 64. Datorită faptului că pianul folosit în acest proiect are o reprezentare imprecisă a acestui număr, în aplicație ne vom folosi numai numărul notei MIDI din care vom calcula frecvența, folosind o formulă, cu care trebuie să vibreze motoarele când o clapă e apăsată. Când vine vorba despre programarea acestor mesaje, modul în care sunt reprezentate depinde de librăria folosită în limbajul de programare respectiv. Despre parsarea acestora vom detalia în subcapitolele ce urmează.

### 3.3 Limbajul de programare Python

#### 3.3.1 Prezentare generală

Python este un limbaj de programare devenit foarte popular datorită faptului că este proiectat să fie ușor de învățat și de înțeles. Este un limbaj de scriptare ceea ce înseamnă că codul scris nu este compilat ci mai degrabă interpretat. Ca și celelalte limbaje de nivel înalt și Python suportă mai multe paradigme de programare cum ar fi programare procedurală, orientată pe obiect sau funcțională sprijinind astfel dezvoltarea unei game largi de aplicații. Poate fi folosit atât pentru realizarea proiectelor de dimensiuni mai mici cât și pentru cele mai mari, fiind cel mai des utilizat în crearea aplicațiilor web (este folosit de companiile Google și Yahoo!), științifice sau de divertisment, cum ar fi jocuri. Este foarte mult folosit și în proiectele ce țin de Internetul lucrurilor, având destule biblioteci ce pot fi folosite pentru programarea senzorilor și actuatorilor.

Faptul că limbajul de programare Python oferă o gamă largă de biblioteci și extinderi, constituie un avantaj în programarea aplicațiilor complexe. Biblioteca standard a limbajului este deseori citată și ca unul dintre cele mai mari puncte forte ale sale, oferind multe tool-uri pentru diferite sarcini.

Sintaxa și semantica limbajului de programare sunt concepute astfel încât codul să fie cât mai ușor de scris și de înțeles de către programator. Spre deosebire de alte limbaje de programare, Python nu folosește acolade pentru delimitarea blocurilor, aceste delimitări făcându-se cu ajutorul indentării. Un cod indenat corect va arunca eroare de tipul `IndentationError`. Se poate vedea în figura 4 o bucată de cod indentat corect. O scăderii a indentării

```
def _setValues(self, value, switch_type):
    print("In the set values method")
    print(str(value) + str(switch_type))
    if value == MESSAGES[Message.ON]:
        DEFAULT_VALUES[switch_type] = True
    elif value == MESSAGES[Message.OFF]:
        DEFAULT_VALUES[switch_type] = False
```

Figura 4:

semnifică sfârșitul unui bloc curent în timp ce creșterea indică începerea unei noi afirmații. Un statement (afirmație) este o unitate sintactică a limbajelor de programare imperative care exprimă unele acțiuni care trebuie efectuate.

În Python, ca și în celelalte limbaje de programare, putem face diferența dintre o metodă și o funcție. Ambele trebuie să înceapă cu cuvântul cheie `def` urmat de denumirea funcției/metodei. Spre deosebire de alte limbaje de programare, în cazul metodelor în Python trebuie specificat explicit cuvântul `self` în parametrii metodei. Acest cuvânt indică instanța clasei de care aparține metoda respectivă.

### 3.3.2 Biblioteca Gpiozero pentru programarea pinurilor

Gpiozero este o bibliotecă care oferă o interfață simplă pentru programarea GPIO-urilor de pe placa de dezvoltare Raspberry Pi. Spre deosebire de alte biblioteci, aceasta oferă o curbă de învățare mai lină pentru începători și posibilitatea dezvoltării proiectelor complicate cu mai multă ușurință și mai puține linii de cod. La început Gpiozero a fost doar un layout peste biblioteca `RPi.GPIO`, ulterior fiind adăugate diferite suporturi. În prezent `RPi.GPIO` este folosit ca bibliotecă implicită pentru Gpiozero, această configurație putând fi schimbată, existând mai multe astfel de biblioteci, cum ar fi de exemplu `pigpio`. Instalarea acestei biblioteci este necesară numai în cazul în care se folosește pe alt sistem de operare decât Raspbian, pe

acesta fiind deja inclus. Odată cu instalarea ei putem rula comanda pinout din linia de comandă pentru a vedea vizual detalii despre pinii disponibili de pe Raspberry Pi (vezi figura 5). Acest lucru devine în ajutor în momentul în care, la configurarea pinurilor, trebuie să precizăm numărul pinului ales astfel fiind mult mai ușor de urmat poziționarea fiecărui pin în parte.

```

J8:
  3V3  (1) (2)  5V
 GPIO2 (3) (4)  5V
 GPIO3 (5) (6) GND
 GPIO4 (7) (8) GPIO14
  GND  (9) (10) GPIO15
 GPIO17 (11) (12) GPIO18
 GPIO27 (13) (14) GND
 GPIO22 (15) (16) GPIO23
  3V3  (17) (18) GPIO24
 GPIO10 (19) (20) GND
  GPIO9 (21) (22) GPIO25
 GPIO11 (23) (24) GPIO8
  GND  (25) (26) GPIO7
  GPIO0 (27) (28) GPIO1
  GPIO5 (29) (30) GND
  GPIO6 (31) (32) GPIO12
 GPIO13 (33) (34) GND
 GPIO19 (35) (36) GPIO16
 GPIO26 (37) (38) GPIO20
  GND  (39) (40) GPIO21

```

Figura 5: Poziționarea pinurilor

În comparație cu alte biblioteci, în Gpiozero se poate observa o abordare orientată pe obiecte, acesta folosind clase în loc de funcții. Pe lângă niște clase specifice, Gpiozero oferă două clase mari de bază, InputDevice și OutputDevice, celelalte fiind derivate din acestea, fiecare având metode și proprietăți specifice care sunt adaptate dispozitivului controlat. Clasa OutputDevices și tot ce derivă din ea este folosită pentru controlarea dispozitivelor de ieșire cum ar fi de exemplu LED-urile, pe când cea de InputDevice este pentru dispozitivele care sunt folosite pentru a furniza date și semnale de control. Astfel alegerea clasei de lucru este mult mai vizibilă și mult mai ușoară.

Pentru controlarea motoarelor de vibrații în proiectul de față s-au folosit clasa PWMOutputDevice derivat din OutputDevice care permit și modu-

larea lățimii pulsului. Importarea bibliotecii se face simplu, conform figurei 6. În momentul instanțierii clasei, în constructor putem da de la unu pana la

```
from gpiozero import PWMOutputDevice
```

Figura 6: Importare biblioteca Gpiozero

cinci parametrii, numai primul fiind obligatoriu, celelalte având deja valorile setate implicit. Parametrul obligatoriu este numărul pinului de pe Raspberry Pi cu care lucrăm. Numerotarea pinurilor în cazul acestei biblioteci se face cu ajutorul sistemului Broadcom. Ceilalți parametri ai constructorului sunt pentru a seta valoarea de început (duty cycle), frecvența, dacă să fie activ sau nu și nu în ultimul rând putem schimba fabricii de pini, asta fiind o caracteristică avansată pe care majoritatea utilizatorilor o pot ignora. Un exemplu de instanțiere a acestei clase se poate vedea în figura 7, unde parametrul gpio indică numărul pinului setat. .

```
# Constructor
def __init__(self, gpio):
    self._vibrationMotor = PWMOutputDevice(gpio)
    print("Hi from the motor constructor. GPIO {} set.".format(str(gpio)))
```

Figura 7: PWMOutputDevice constructor

Datorită faptului că clasa permite modularea lățimii pulsurilor, se pot seta intensități între valorile 1 și 0, 1 fiind intensitatea maximă cu care motorul poate să vibreze și 0 fiind starea în care se oprește din vibrat. Totodată se poate seta și frecvența, aceasta implicit fiind 100Hz, numărul acesta indicând rata de apariție a unui eveniment repetitiv [6]. Exemplu de setare a acestor valori poate fi observat în figura 8. Clasa dispune mai multe metode și proprietăți cu ajutorul cărora putem porni sau opri dispozitivul. În cazul nostru este de ajuns să setăm proprietățile value și frequency pentru a da drumul sau a opri vibrațiile.

```
self._vibrationMotor.value = value
self._vibrationMotor.frequency = frequency
```

Figura 8: Setarea valorilor motorului



Biblioteca Gpiozero pune la dispoziție și o clasă `Tone` destinată utilizatorilor care folosesc `TonalBuzzer` pentru a putea reprezenta cu ușurință notele muzicale. Cu toate că în proiectul de față nu este folosit un astfel de dispozitiv, clasa asta ne poate deveni în ajutor la calcularea frecvențelor pornind de la numărul notei MIDI transmis de pian. Datorită faptului că clasa poate fi construită într-o varietate de moduri, asta incluzând și prin numere MIDI, se poate cu ușurință obține frecvența notei respective cu ajutorul atributului `frequency`. Codul din spatele clasei calculează frecvența cu formula

$$f = 2^{(d-69)/12} * 440Hz$$

unde parametrul `d` reprezintă numărul notei muzicale MIDI. Pe lângă nota MIDI, clasa se poate construi pornind și de la frecvență sau de la specificarea notei prin sistemul alfabetic (acesta consistând dintr-o literă majusculă de la A până la G urmat de un modificador opțional și numărul de octave). Folosirea acestei clase se poate vedea în figura 9.

```
def convert_midi_number_to_frequency(midiNumber):
    try:
        tone = Tone(midi=midiNumber)
        return tone.frequency
    except:
        print("Operation problem (convert midi number to frequency)")
```

Figura 9: Calcularea frecvenței

### 3.3.3 Folosirea bibliotecii Mido

Parsarea și reprezentarea datelor primite de la instrumentul muzical este un pas important în ceea ce privește rularea acestui proiect. Din fericire limbajul de programare Python ne oferă mai multe posibilități prin care putem să ajungem la același rezultat, depinde doar de noi cu ce vrem să lucrăm și care modalitate se potrivește task-ului respectiv. Pentru citirea, parsarea și modificare mesajelor/fișierelor MIDI există mai multe tipuri de biblioteci, fiecare făcând în mare parte același lucru, cu puține diferențe. Biblioteca cu care s-a lucrat în acest proiect se numește `Mido` și datorită documentației bine pusă la punct este relativ ușor de urmărit ce metode/funcționalități are. Pe parcursul dezvoltării proiectului am încercat să lucrez cu mai multe astfel de tipuri de biblioteci, însă cele mai multe s-au dovedit a fi mai greu de folosit, ducând lipsă de funcționalități sau având numai metodele minime necesare.

Instalarea bibliotecii pe sistemul de operare Raspbian se face cu executarea comenzii `pip install mido`, iar pentru utilizarea porturilor trebuie instalată și biblioteca `rtmidi` cu ajutorul comenzii `pip install python-rtmidi`. `Rtmidi` este backend-ul folosit în mod implicit care se poate schimba cu altele, cum ar fi `PortMidi` sau `Pygame`. Este recomandat să se folosească `rtmidi` din datorită faptului că este mult mai ușor de instalat și are toate caracteristicile celorlalte biblioteci.

Pentru a putea primi date de la pian, trebuie deschis un port de intrare specificând numele portului respectiv (a se vedea figura 10). Pentru a putea

```
def __init__(self, port):  
    self._midiIn = mido.open_input(port)
```

Figura 10: Port de intrare

găsi numele portului există metoda `mido.get_input_names()` a bibliotecii, care afișează toate proturile disponibile. Sunt incluse și alte tipuri de porturi în bibliotecă înafară de cele de intrare și de ieșire, precum multi portul care ne permite să citim și să scriem mesaje pe mai multe porturi, `SocketPort`-ul sau `IOPort`-ul care este o combinație între cel de intrare și ieșire. După nevoi se pot implementa și altfel de tipuri.

Mesajele MIDI vor veni pe acest port deschis iar în funcție de arhitectura instrumentului muzical acestea pot veni fără întreruperi, asta însemnând că programul primește date chiar și atunci când nicio o tastă nu a fost apărată pe pian, sau pot veni doar doar când a fost cântată o notă muzicală. În funcție de asta poate fi implementată metoda de parsare a acestor mesaje. În cazul nostru pianul trimite date încontinuu asta însemnând că vor trebui selectate mesajele de care suntem interesate. Iterarea peste mesaje se întâmplă cu ajutorul unui `for` până când portul se va închide. Pentru o parsare mai ușoară, mesajele de intrare vor fi mai întâi convertate la octeți MIDI ceea ce înseamnă că mai departe se va lucra cu un vector de valori, fiecare valoare corespundându-se cu un parametru a notei MIDI respective. Un exemplu de astfel de parsare se poate vedea în figura 11. Vectorul găsit conține 3 parametri printre care ultimele două indică valoarea notei MIDI respectiv puterea cu care acesta a fost cântat/apăsat. Biblioteca `mido` suportă majoritatea tipurilor de mesaje descrise în prezentarea mesajelor MIDI.

```

for message in self._midiIn:
    if bridge.ev.is_set():
        self._processEvent()
    bytes_array = message.bytes()
    if(self._is_pressed_note(bytes_array)):
        note, velocity = self._get_note_and_velocity(bytes_array)
        ansi_note = Helper.number_to_note(note)

```

Figura 11: Citire mesaje MIDI

### 3.3.4 Programarea asincrona cu biblioteca Asyncore

Una dintre metodele cele mai rapide și simple de a trimite mesaje într-o rețea este comunicare prin sockets. În Python socket-urile sunt obiecte care oferă o modalitate de schimb de informații între două procese într-o manieră directă și independentă de platformă. Aceștia pot fi implementate pe mai multe tipuri de canale diferite cum ar fi TCP<sup>10</sup> sau UDP<sup>11</sup>. Varianta cea mai de încredere este utilizarea protocolului TCP, care este și cel utilizat în mod implicit, datorită faptului că pachetele aruncate în rețea sunt detectate și retransmise de expeditor și datele sunt citite în ordinea trimiterilor lor. E important să ținem cont de acest lucru datorită faptului că nu există nicio garanție ca datele trimise să ajungă la destinație. Cele mai comune aplicații cu socket-uri sunt cele de tipuri client-server, unde clientul trimite date către server iar serverul le procesează. Trimiterea acestor mesaje poate fi și una bi-direcțională cum va fi și în cazul nostru. Atât clientul cât și serverul vor trimite date între ele.

În limbajul de programare Python aceste socket-uri pot fi implementate cu ajutorul modulului socket, care oferă o interfață de programare a aplicației convenabilă și consistentă. Modul în care aceste socket-uri funcționează este în felul următor: serverul crează un socket, legându-se la o adresă și un port, care va asculta conexiunile de la client la adresa respectivă. Îndată ce un client socket, cu aceeași adresă și port s-a conectat la server se poate începe trimiterea de date. La sfârșit, după ce s-a încheiat tot procesul, clientul poate închide conexiunea cu serverul printr-o simplă metodă `socket.close()`, după care se închide și partea de server.

Crearea unui server simplu este ușor de realizat în Python, nefiind atât de multe metode ce trebuie utilizate. Când vine vorba despre trimiterea datelor

<sup>10</sup>Transmission Control Protocol

<sup>11</sup>User Datagram Protocol

către un client la un anumit timp (când sunt disponibile) este mult mai ușor și avantajos de folosit modulul Asyncore, care ne oferă, pe lângă o implementare reactivă a unui socket, posibilitatea de a avea un program asincron ceea ce poate fi un avantaj mare mai ales când avem mai multe socket-uri deschise. Asyncorel ne oferă o infrastructură de bază pentru scrierea socket-urilor de client și server. Astfel programatorul are posibilitatea de a scrie cod care se apelează numai atunci când trebuie să se întâmple ceva (să se trimită sau să se primească anumite date) în loc de a apela metode și de a crea obiecte de tip socket. Ideea de bază în spatele acestui modul este de a crea unul sau mai multe instanțe de clasă `asyncore.dispatcher`. Fiecare astfel de canal de rețea creat se adaugă la o hartă globală care este folosită de funcția `loop()`. Apelarea acestei metode `loop()` activează aceste canale, care rulează până când ultimul canal a fost închis.

În acest proiect s-au folosit două clase de dispatcher, unul pentru ascultarea socket-ului și unul pentru canalul de client, de unde se vor apela metodele de trimitere și de primire a mesajelor. Implementarea acestor clase se face prin suprascrierea a câtorva metoda care vor manipula acțiunile necesare. Se va apela metoda `handle_accept()` (vezi figura 12) când se face o cerere de conectare de la socket-ul client. Aici se va crea o instanță nouă de clasă dispatcher pentru se vor manevra mesajele. Constructorul acestei clase de MessageHandler are ca parametrii un obiect socket care reprezintă conexiunea, adresa clientului conectat și o instanță a clasei de server din care se pornește conexiunea.

```
def handle_accept(self):
    (conn_sock, client_addr) = self.accept()
    self.handlerClass(conn_sock, client_addr, self)
```

Figura 12: Handle accept

Pentru trimiterea datelor din serverul de socket către client framework-ul `asyncore` are implementat o metodă `writable()` care verifică dacă este de trimis un mesaj, acesta implicit returnând `True`. Suprascrierea acestei metode implică impunerea a unor condiții prin care metoda să verifice dacă chiar este ceva de trimis, altfel să returneze `False`. Dacă există date de trimis, framework-ul va apela automat metoda de `handle_write()` în care este chemată metoda din modulul socket, `self.send()` (figura 13).

```

def writable(self):
    if len(bridge.message_to_client_q) != 0:
        return True
    return False

# send frequencies to client for color representation
def handle_write(self):
    to_send = bridge.message_to_client_q.get()
    sent = self.send(to_send.encode())

```

Figura 13: Trimitere date

Primirea datelor de la client (fie el implementat în Python sau în orice alt limbaj de programare) se face asemănător. Metoda `writable()` determină dacă serverul poate să primească date, iar dacă da, acestea sunt primite și decodate în metoda `handle_read()` (figura 14). În cazul nostru vrem să primim mereu date de la client așa metoda respectivă va returna mereu True.

```

def handle_read(self):
    data = self.recv(constants.BUFFER_SIZE)
    decoded_data = data.decode()
    bridge.enqueue_message(decoded_data)

```

Figura 14: Primire date

## 3.4 Limbajul de programare Java

### 3.4.1 Scurtă introducere

Limbajul de programare Java este un limbaj cu scop general, bazat pe clasă, orintat pe obiect și proiectat să aibă cât mai puține dependențe de implementare fiind cel mai popular limbaj de programare în 2019. Sintaxa limbajului seamănă cu cel de la C sau C++

### 3.4.2 Folosirea firelor de executie in Java

## 3.5 Sistemul de operare Android

### 3.5.1 Baza de date SQLite

## 4 Prezentarea aplicației

### 4.1 Aplicația RaspberryPi

#### 4.1.1 Prezentare dispozitiv portabil

### 4.2 Aplicația Android

#### 4.2.1 Prezentare UI

## 5 Concluzii

## 6 Bibliografie si webografie

- [1] An Introduction To Vibration Motors. <https://www.precisionmicrodrives.com/vibration-motors/>.
- [2] GPIO. <https://www.raspberrypi.org/documentation/usage/gpio/>.
- [3] MIDI Tutorial. <https://www.cs.cmu.edu/~music/cmsip/readings/MIDI\%20tutorial\%20for\%20programmers.html>.
- [4] James Cusick, William Miller, Nicholas Laurita, and Tasha Pott. Design, construction, and use of a single board computer beowulf cluster: Application of the small-footprint, low-cost, insignal 5420 octa board. *Arxiv*, 12 2014.
- [5] Rachel Elaine. How Deaf People Experience Music, 2017.
- [6] Michael Lombardi. *Frequency Measurement*, page 24 p. 02 1999.
- [7] Stefano Papetti and Charalampos Saitis. *Musical Haptics: Introduction*, pages 1–7. 05 2018.
- [8] Alexis Rohlin. How to Use MIDI Cable In & Out Plugs. <https://smallbusiness.chron.com/use-midi-cable-out-plugs-49492.html>.
- [9] Luca Turchet, Carlo Fischione, and Mathieu Barthet. Towards the internet of musical things. 07 2017.