Haptic feedback in Internet of musical things

Czoguly Tünde

June 5, 2020

Cuprins

Lista de figuri

1 Introducere

1.1 Sursele de motivație pentru realizarea lucrării

1.1.1 Internetul obiectelor muzicale

Internetul obiectelor este o platformă vastă 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 poate colecta și schimba date între ele cu ajutorul unor senzori și actuatori. Scopul principal al acestor creații este de a ne ușura și de a ne susține activitățile noastre zilnice.

Internetul lucrurilor muzicale este derivat din acest domeniu, fiind o ramură relativ nouă și emergentă, mai mult bazată de cercetare și experimentare, având mai 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.[17]

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 este un factor decisiv când vine vorba despre satisfacția pe care o simțim la sfârșitul unui concert. 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 creea un dispozitiv purtabil cu ajutorul căruia oamenii cu deficiență de auz să poată resimți vibrațiile muzicii astfel având și ei parte de o experiență mai bogată in 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 [1], marea provocare a fost de a găsi o modalitatea de a îmbina acesta cu pasiunea mea astfel încăt 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

[1] Internet of Things

nici una foarte populară nu e. S-au făcut experimente cu astfel de wearable device-uri pe piață cu rezultate pozitive, din ce am citit, însă cele mai multe n-au fost scoase la vânzare, au râmas numai pe plan experimental. Fiind un subiect atât de rar abordat decizia mea a fost să experimentez și eu acest topic, alegând ca sursă de muzică un pian electric cu o interfață MIDI.

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 [8], 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 [12]. În cazul audienței surde acest set de vibrații nu este resimțită sau este 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 purtabil.

În capitolul al treailea 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 protocolului de comunicare MIDI[2] și mesajele transmise de la instrumentele muzicale.

Capitolul al patrulea este rezervat pentru prezentarea aplicației. Sunt amănunțite detalii despre procesul de implementare atât a aplicației de pe placa de dezvoltare Rasperry Pi cât și a aplicației mobile.

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

La sfârșit se află materialele bibliografice folosite pentru realizarea documentației.

1.3 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 poate să fie extinsă pe mai multe instrumente, nu numai pian. Astfel utilizatorul n-ar simți numai 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 are și o reprezentare vizuală în culor a muzicii, 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 culegrea datelor din pian cât și pentru punerea în funcțiune a motoarelor. Cazul idealar 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 inpune din cauza faptului că datele vor trebui să fie transmise de la un dispozitiv la altul, asta implicând o posibilă nesincronizare între muzica de pe scenă și vibrațiile simțite.

[2]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 dacă 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 [3]. Un astfel de calculator este și Rasbperry Pi-ul care a fost creat în Marea Britanie cu scopul de a promova predarea informaticii în școlile și țările în curs de dezvoltare [7] 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.

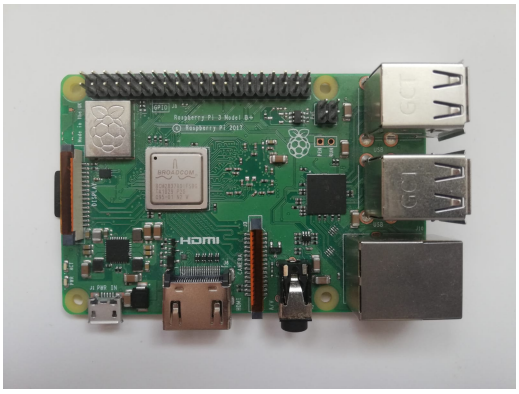

Figura 1: Raspberry Pi 3 model B+

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 componenetele electrice folosind piste conductoare. În

[3]Single-board computer

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

2.1.2 Folosirea pinurilor

Pe fiecare model de placă RaspberryPi se pot observa de-a lungul marginii superioare pinii 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 înseamnând că la ieșire sunt setate la 3V3 și la intrare sunt tolerate la 3V3 [3]. O caracteristică importantă a acestori pini generali este modularea lățimii pulsurilor (PWM[6]) cu ajutorul căruia putem controla dispozitivele analogice cu o ieșire digitală [10] find o metodă alternativă de control armonic.
- aici o să mai fie o completare de spre pwm

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. Există două sisteme de numerotare: sistemul Broadcom (care este folosit cel mai des) și numerotarea fizică a pinurilor.

2.1.3 Conectarea la internet

Conectarea plăcii Raspberry Pi la internet se poate face în două feluri: prin cablu de internet, placa având port de internet, sau prin WIFI. În abele cazuri, dacă vrem să accesăm linia de comandă a plăcii respective, conectarea se face prin prin SSH[7]. Pentru asta trebuie aflată adresa IP. Acest lucru se poate face în mai multe feluri, existând mai multe programe de căutare a adreselor IP. Datorită faptului că această adresă nu e una constantă, ci se schimbă mereu, indicat e folosirea unei adrese statice.

[4]Random access memory
[5]General-purpose input/ouput
[6]Pulse Width Modulation
[7]Secure Shell

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 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 conectoare DIN [8] 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 varinate 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 deajunse 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 [14]. Deci mufa IN se va conecta la portul OUT și mufa OUT la cel de IN.

[8] Deutsches Institut für Normung

2.2 Actuatori folosiți pentru realizarea feedback-ului haptic

2.2.1 Motoare de vibrați 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 folosit. Există două mari categorii de astfel de motoare și anume ERM și LRA. Diferența dintre acestea, înafară 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 cazua mărimii 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 motore sunt puse în funcțiune este prin conectarea firelor la pinurile corespunzătoare de pe placa Raspberry Pi. 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

Figura 3: Motor de vibrații LRA

roșu, dar sunt și motoare care merg pe varianta de roșu și albastru. Scopul principal 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 Rasberry Pi, bazat pe Debian, numele lui venind din combinația cuvintelor Raspberry și Debian. Sunt mai multe versiuni a acestui sistem de perare, ultimul fiind Buster care e compatibil cu toate modelele de Rasperry Pi. Cu toate că a fost creat special pentru acestă placă, este funcțional și pentru sistemele non-Raspberry Pi, fiind dezvoltat pentru arhitecturi de microprocesoare de tip RISC [9]. Instalarea sistemlui de operație se face relaitv ușor. După descărcarea imaginii și scrierea lui pe card, tot ce trebuie făcut este boot-area plăcii de dezvoltare Raspberry Pi. O parte a codului din acest proiect se execută pe acest sistem de operare.

Cea mai ușoară cale de a vedea interfața sistemului de operare este prin conectarea plăcii Rasperry Pi la un televizor cu ajutorul unui cablu HDMI[10] ș folosirea unei tastaturi și a unui mouse pentru controlarea acestuia. Cealaltă cale, care necesită mai puțin efort, este accesarea terminalului cu ajutorul protocolului Secure Shell. Programul pe care l-am folosit pentru acest acces se numește Putty. Aceasta accesează terminalul cu ajutorul adresei IP a plăcii Raspberry Pi.

- se va insera o poza cu cum arata interfata (mi-am amintit la sfarsit)

3.2 Protocolul de comunicare MIDI

3.2.1 Prezentarea mesajelor MIDI

Comunicarea între un instrument muzical electric și un calculatori 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 facând acest tip de comunicare un instrument puternic în industria muzicală.

[9] Reduced Instruction Set Computer
[10] High-Definition Multimedia Interface

Un mesaj MIDI constă dintr-un octet de stare, care indică tipul mesajului, urmat de până la doi octeți de date care conțin parametrii [4]. 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 o tastă a pianului respectiv când s-a dat drumul la tasta respectivă. Diferența de timp dintre aceste două tipuri ne poate da 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 in ceea ce privește multitudinea tipurilor de mesaje ce pot trimite.

Parametrii unui mesaj MIDI, după cum s-a discutat și în paragraful anterior, pot conține nota muzicală și velocitatea cu care aceasta a fost cântată. Reprezentarea acestei note muzicale se face printr-un număr de la 0 pana 127 [5], 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 [5], 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, științifice sau de divertisment în companiile cum ar fi Google, Facebook, Instagram sau Spotify [13]. Este foarte mult folosit și în proiectele ce țin de Internetul lucrurilor, având destule biblioteci ce pot fi folosite pen-

tru 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 citat și ca unul dintre cele mai mari puncet forte ale sale, oferind multe tool-uri pentru difeite 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 indenatat incorect 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: Exemplu indentare

seminfică 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 fii schimbată, existând ma 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.

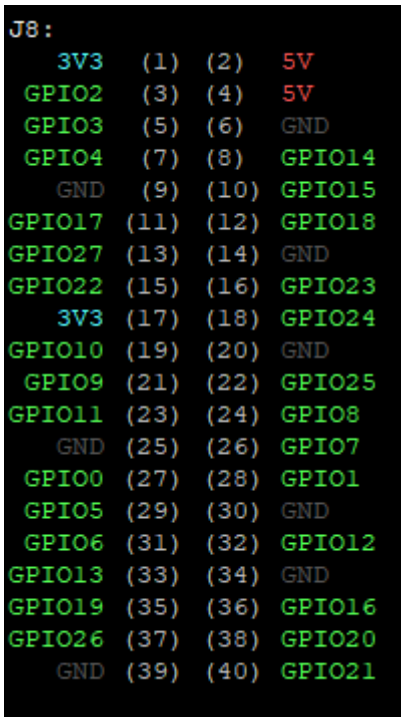

Figura 5: Poziționarea pinurilor

În comparație cu alte biblioteci, în Gpiozero se pote 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. Asftel 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 modularea lățimii pulsului. Importarea bibliotecii se face simplu, conform figurei 6. În momentul instanțierii clasei, în constructor putem da de la unu pana la

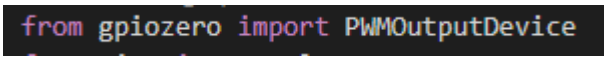

Figura 6: Importare biblioteca Gpiozero

cinci parametrii, numai primul fiind obligatori, 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 parametrii 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

Datoriă 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 [11]. 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 deajuns 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 modificator opțional și numărul de octave). Folosirea acestei clase se poate vedea în figura 9.

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

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 bibiotecii 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 [6].

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 corespânduse cu un parametru a notei MIDI respective. Un exemplu de astfel de parsare

se poate vedea în figura 11. Vectorul găsit conține 3 parametrii printre care

```
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

ultimele două indică valoarea notei MIDI respectiv puterea cu care acesta a fost cântată/apăsat. Biblioteca mido suportă majoritatea tipurilor de mesaje descrise în prezentarea mesajelor MIDI.

3.3.4 Programarea asincrona cu biblioteca Asyncore

Una dintre metodele cele mai rapide și simple de a trimite mesaje într-o rețetea este comunicare prin sockets. În Python socket-urile sunt obiecte care oferă o modalitate de schimb de informații între două procese între-o manieră directă și independentă de platformă. Aceștia pot fi implementate pe mai multe tipuri de canale diferite cum ar fi TCP[11] sau UDP[12]. Varianta cea mai de încredere este utilizarea protocolului TCP, care este și cel utilizat în mod implicit, datorită faptului că pachtele 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 datoriă 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 serveril vor trimite date între ei.

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 la care va asculat conexiunile de la client la adresa respectivă. Îndată ce un client socket, cu aceeași adresă și port s-a concetat la server

[11]Transmission Control Protocol
[12]User Datagram Protocol

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 atăt de multe metode ce trebuie utilizate. Când vine vorba despre trimiterea datelor 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 socketurilor 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 trimite 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 clase `asyncore.dispatcher`. Fiecare astfel de canal de rețea creat se adaugă la o hartă globală care eset folosită de funția `loop()`. Apelarea acestei metode `loop()` activează aceste canale, care rulează până când ultimul canal a fost închis [2].

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 frameworkul 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, atlfel să returneze False. Dacă există date de trimis, framewok-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 represenation
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 vream 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 clase, 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 cel mai mult cu cel de C sau C++. Orientarea pe obiect este o caracteristică foarte importantă în ceea ce privește structura acestui limbaj datorită faptului că tot codul scris trebuie să fie într-o clasă. Astfel, în structura unui proiect scris în Java, fiecare fișier reprezintă o clasă separată. Fiecare element de date este un obiect, excepție făcând numai tipurile primitive de date. În comparație cu limbajul de programare C++, Java nu suportă moștenirile multiple și suprascrierea operatorilor.

Pentru gestionarea memoriei într-un ciclu de viață a unui obiect Java folosește un colector automat de gunoi[13] care are ca responsabilitate recuperarea memoriei odată ce un obiect nu mai este folosit. Acest lucru se face cu ajutorul referințelor la obiectul respectiv.

Limbajul de programare Java este folosit în dezvoltarea a mai multor tipuri de aplicații cum ar fi de exemplu aplicațiile de desktop, mobile, procesare a datelor mari[14] sau de sisteme încorporate. Scopul acestui limaj în proiectul de lincență de față este crearea unei aplicații Android pentru personalizarea experienței utilizatorilor și redarea într-un context vizual sunetele notelor muzicale folosind culorile și datele colectate de la pian.

3.4.2 Folosirea firelor de executie in Java

Firele de execuție joacă un rol important în momentul în care dorim să creăm o aplicație mobilă cu ajutorul limbajului de programare Java. Un fir de execuție poate fi definită ca fiind ce mai mică unitate de procesare ce poate fi programată spre execuție de către un sistem de operare. Ele permit programelor să facă mai multe lucruri simultan.Acest lucru permite unui utilizator să facă ceva în timp ce altceva se întâmplă pe fundal [9], firele de execuție execută porțiuni de cod în paralel în interiorul aceluiași proces. În timp ce comunicarea între procese se poate face numai cu mecanisme de comunicare interprocesare, comunicarea între firele de execuție se întâmplă mult mai ușor și mai rapid, ele având un spațiu de memorie comun și un sistem de semaforizare.

Java este un limbaj de programare care are printre principiile sale și suportarea programării pe mai multe fire de execuție. Sistemul Java de rulare depinde de thread-uri datorită a mai multor lucruri. Firele de execuție reduc ineficiența prin prevenirea pierderii ciclirilor procesorului [15].

[13]Garbage collector
[14]Big Data

Noțiunea de multithreading înseamnă rularea unei aplicații pe mai multe fire de execuție. În cazul limbajului de programare Java acest sistem este bazat pe clasa `Thread`, pe metodele acestuia și pe interfața `Runnable`. Astfel pentru a crea un fir de execuție nou avem două soluții: putem extinde clasa `Thread` și să suprascriem metoda `run()` sau putem să implementăm interfața `Runnable`, ambele având același efect. În interiorul metodei `run()` se va scrie codul care se va executa în dată ce firul de execuție e pornit. Astfel de exemplu se poate vedea în figura 15. Implementarea acestei metode și

```
class MessageThread implements Runnable{

    @Override
    public void run() {
        while(true){
            try {
                byte[] message = new byte[bufferSize];
                in.readFully(message);
                String decoded = new String(message, charsetName: "UTF-8");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Figura 15: Crearea firelor de execuție

crearea unei instanțe a acestei clase nu este deajuns pentru a porni un nou fir de execuție în aplicație. Un fir de execuție are mai multe stări în dealungul vieții lui. Când instanțiem clasa respectivă, îl punem în starea `New`, aceasta indicând că firul de execuție e într-o stare nouă, însă nu se rulează. Rularea unui thread-ului se întâmplă în momentul în care apelăm metoda `start()` pe firul de execuție respectiv, cum se pate vedea și în figura 16. Până atunci acesta va rămâne în starea nouă.

```
new Thread(new MessageThread()).start();
```

Figura 16: Rularea unui fir de execuție

Un fir de execuție poate fi și într-o stare de așteptare ceea ce înseamcă că așteaptă un alt thread să efectueze o sarcină. Acest thread poate să

treacă înapoi în starea de rulare numai în cazul în care se semnalează acest lucru. Firul de execuție poate fi într-o fază terminată dacă se întâmplă una dintre următoarele două situații: când rularea programului s-a terminat în totalitate sau a apărut o excepție care nu a fost tratată.

3.5 Crearea unei aplicații Android cu Android Studio

3.5.1 Sistemul de operare Android

Android este un sistem de operare care a fost conceput pentru dispozitivele mobile și care inițial a fost dezvoltat de compania Google. Este bazat pe sistemul de operare Linux însă mai multe drivere și biblioteci au fost modificate sau dezvoltate pentru a permite Android-uli să funcționeze cât mai eficient pe dispozitivele mobile. Aplicațiile Android sunt scrise în limbajul de programare Java și rulate pe o mașină virtuală. Se pot dezvolta aplicații și în limbajul C acestea fiind compilate în cod mașină ARM. Dezvoltarea acestui sistem de operare se întâmplă într-un pas destul de rapid, având un release major nou în fiecare lună [16]. Pentru stocarea datelor Android folosește software-ul SQLite

3.5.2 Android Studio

Pentru crearea aplicației Android s-a folosit Android Studio datorită faptului că oferă multe funcții care îmbunătățesc productivitatea atunci când vine vorba despre construirea aplicațiilor Android. Android Studio ne oferă un mediu unificat în care putem dezvolta aplicații pentru toate dispozitivele Android având integrat un emulator rapid cu ajutorul căruia putem testa aplicația fără a conecta un dispozitiv Android la calculator. Emulatorul ne oferă aproape toate capabilitățile unui dispozitiv Android real. Desigur avem posibilitatea de a testa aplicație și pe un telefon real, făcând numai câteva setări din telefon.

Structura unui proiect în Android Studio conține mai multe directoare cu codul sursă atât pentru interfață cât și pentru clasele cu ajutorul cărora aplicația va rula încât și clasele pentru testarea aplicației. La prima vedere fișierele cele mai importate sunt `AndroidManifest.xml` în care putem găsi informații esențiale despre aplicația noastră, fișierul java `MainActivity` care cel mai bine se poate compara cu funcția `main()` din C++, fiind locul de unde se încarcă componenta UI a aplicației. Dacă avem o aplicație cu ecrane multiple, pentru fiecare ecran o să avem o activitate diferită, activitatea principală fiind fereastra cu care pornește aplicația. În cazul acestei aplicații vom avea

doar o singură fereastră, ciclul acestuia fiind controlată din `MainActivity`. Există și un folder numit res, care conține toate resursele care nu sunt cod cum ar fi șiruri UI și imagini bitmap. Aici putem vedea vizual cum va arăta interfața aplicației fiind și locul în care putem construi UI-ul printr-o simplă tragele a componentelor din paletă.

Paleta din Android Studio conține destule widget-uri pentru a dezvolta o interfață complexă și frumoasă pentru aplicația noastră. Fiecare astfel de element are propriile atribute și poate fi personalizat după placul nostru. În proiectul de față s-au folosit mai multe astfel de componente, fiecare având un rol aparte în funcționarea aplicației.

Pentru reprezentarea a două stări (pornit și oprit) s-a folosit un `Switch`, care este un comutator cu ajutorul căruia afișa/ascunde ceva sau porni/opri executarea unei bucăți de cod. Starea comutatorului poate fi setat încă de la pornirea aplicației cu ajutorul metodei `setChecked()` care primește ca parametru un bulean indicând starea în care se va afla.

Butonul eset una dintre elementele de cele mai bază când vine vorba despre proiectarea unei interfețe mobile. Prin apăsarea unei componente `Button` putem da drumul unei acțiuni cu ajutorul unei metode ascultătoare. Metoda `setOnClickListener()` ascultă la atingerea butonului de către utilizator și execută codul dinăuntru când se întâmplă această acțiune. Un exemplu de astfel de cod se poate vedea în figura 17.

```
resetButton.setOnClickListener((v) → {
        resetValues();
        if(clientThread != null){
            clientThread.sendMessage("reset all");
        }
});
```

Figura 17: Listener

O componentă mai interesantă este cel de `Seekbar` care este o extensie a barei de progres și cu ajutorul căruia putem seta prin glisare o valoare a progresului. Această valoare poate fi un număr întreg iar valoarea minimă și maximă poate fi setată din cod cu ajutorul atributelor `min` și `max`. Putem prelua valoarea modificării cu metoda `onProgressChanged()` (figura 18).

```
seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        Toast.makeText(getApplicationContext(), text: "seekbar progress: " + progress, Toast.LENGTH_SHORT).show();
        if(clientThread != null){
            clientThread.sendMessage("progress " + progress);
        }
    }
```

Figura 18: SeekBar

Fiecare dintre aceste componente primește un identificator unic cu ajutorul cărora le putem indentifica în cod. Pentru setarea atributului text în Android Sudio se folosesc resursele de șir[15] care se găsesc în fișierul `strings.xml`. Punerea șirurilor de caractere în astfe de resurse previne folosirea dublicatelor, iar în cazul în care se schimbă texul unei componente, schimbarea poate fi făcută dor dintr-un singur loc, textul fiind actualizat în toată aplicația.

3.5.3 Baza de date SQLite

De completat cand va fi gata.
- scurta descriere despre sqlite

[15]string resources

4 Prezentarea aplicației

4.1 Prezentarea scurtă a aplicației finale

Cu ajutorul tehnologiilor software și hardware scopul acestei lucrări a fost de a crea un dispozitiv care va veni în ajutor în comunitatea surdă când vine vorba despre ascultarea muzicii la un instrument anume. Datorită faptului că celelalte simțuri a acestor persoane sunt mult mai dezvoltate am mers pe idea de a crea o aplicația care să redea nu numai vibrațiile muzicii ci și să facă și o reprezentare cât mai bună posibilă a acestor note în culori. Astfel utilizatorii vor putea avea și o experiență vizuală plăcută pe lângă cea tactilă. Pentru a face posibilă acest lucru mai întâi s-a dezvoltat o aplicație pe placa de dezvoltare Raspberry Pi, care are ca scop atât colectarea și trimiterea informațiilor de la pian către aplicația Android cât și redarea vibrațiilor cu ajutorul motoarelor LRA. Aplicația Android a venit ca o completare a acestui proiect din nevoia personalizării experienței a utilizatorilor și pentru redarea efectului vizual din datele primite de la placa de dezvoltare Raspberry Pi. În ambele părți s-au folosit tehnologii prezentate în capitolele anterioare iar în cele ce urmează se vor detalia atât metodele de implementare alese cât și cum s-a ajuns la varinata finală în ceea ce privește structura hardware a lucrării.

4.2 Implementarea aplicației pe placa de dezvoltare RaspberryPi

4.2.1 Metode de asamblare a componentelor hardware

Pentru realizarea dispozitivului purtabil s-a folosit nouă bucăți de motoare de tip LRA conectate da placa de dezvolatre Raspberry Pi. Datorită faptului că pentru acest număr de motoare trebuie 9 pini de ground, iar placa are numai 8 la bucăți, s-a folost un breadboard pe care se poate conecta un singur pin de ground. Conectarea, după cum se vede și din figura ... s-a făcut cu ajutorul unor cabluri de tip tată-tată și mamă-tată, motoarele de vibrații fiind și ele lipite de aceste fire pentru un contact mai bun.

Numărul total de motoare de vibrații LRA folosite influențează experiența utilizatoruluil din motive destul de evidente. Mai multe motoare înseamnă vibrații intensificate și de mai multe feluri. Datorită faptului că am folosit un instrument muzical cu o gamă largă de note era evident că pentru a avea un rezultat cât mai bun este nevoie de mai multe motoare. Nu s-au folosit un număr mai mare de motoare decât nouă din cazua posibilităților de echipa-

mente hardware. Modul în care aceste motoare sunt distributie și vibrează este în felul următor: folosind un pian electric cu 61 de clape am împărțit aceste clape în trei categorii de note muzicale. Primele două octave (adică notele joase) reprezintă prima categorie. A doua categorie reprezintă notele din mijloc, aceasta fiind formată numai dintr-o ocatvă. Ultima categorie este reprezentată de cele două octave rămase adică notele cele mai înalte. Fiecare astfel de categorie "primește" trei motoare. Când se va apăsa o clapă una dintre cele trei categorii de motoare vor începe să vibreze la o anumită valoare și frecvență calculată. Metoda prin care au fost calculate aceste categorii se poate observa în figura 19 unde se verifică numărul MIDI a notei muzicale iar în funcție de asta se va returna un bulean.

```
# we consider the bass range to stop at middle C
@staticmethod
def is_in_bass_range(midiNumber):
    return True if midiNumber < constants.MIDDLE_C else False

# treble range  == the range that we play with our right hands
@staticmethod
def is_in_treble_range(midiNumber):
    return True if midiNumber >= constants.MIDDLE_C and midiNumber <= 70 else False
```

Figura 19: Împărțirea motoarelor în categorii

Breadbourd-ul folosit pentru prototipare este unul cu 830 de puncte, având destul loc pentru conectarea motoarelor. Pentru conecatrea la ground s-a folosit un singur pin de pe placa de dezvoltare Raspberry Pi, acesta fiind conectat la rândul corespunzător de pe breabdoard. Cei nouă pini folosiți pentru alimentarea cu tensiune sunt: 14, 15 și 18 pentru prima categorie, 17, 27, 22 pentru a doua respectiv 10, 9 și 11 pentru ultima categorie de motoare, fiecare dintre aceștia putând fi programat pentru modularea lățimii pulsului.

- secțiunea aceasta va mai fi completată cu poze cu dispozitivul asamblat (firele sunt lipite, trebuie doar să le conectez într-un mod corespunzător la breadboard)

4.2.2 Metoda de calculare folosită pentru motoarele de vibrații

Calcularea intensității și a frecvenței cu care vibrează un motor se face pe baza notei MIDI primite de la pian. Nota muzicală cântată ajunge sub forma unui număr în programul nostru asta însemnând că cu cât e mai mare numărul respectiv cu atât nota muzicală cântată este mai înaltă. Pentru că

există formulă numai pentru calcularea frecvenței, pentru calcularea valorii am mers mai insinctiv.

Știm că un motor de vibrații în cazul nostru poate lua valori între 0 și 1, asta însemnând că cu valoarea 0.5 motorul ar avea o vibrație decentă, fără prea multă putere. Luând această valoarea ca baza, am folosit regula de trei simpli pentru a calcula valoarea notei cântate. Nota care va vibra cu valoarea 0.5 am ales să fie A4 (la) fiind referința de ton care sunt reglate celelalte instrumente muzicale, acesta având și o frecvență întreagă (440 Hz) și fiind și clapa care este aproape de mijlocul pianului. O parte a metodei în care se calculează aceste valori se poate vedea în figura 20. Știind că nota

```
if midiNote == constants.MIDDLE_A:
    print("Middle A value {}".format(str(constants.MIDDLE_A_VALUE)))
    return constants.MIDDLE_A_VALUE
else:
    vibration_value = Helper._rule_of_3(midiNote)
    print("Calculated vibration value: {}".format(str(vibration_value)))
    if vibration_value > 1:
        return 1
    else:
        return vibration_value
```

Figura 20: Calcularea valorii unei vibrații

la este baza după care se calculează celelate note, apăsarea acestuia nu va chema metoda ce returnează rezultatele de la regula de 3 simpli. În cazul în care valoarea calculată este mai mare decât 1, se va returna valoarea 1 datorită faptului că biblioteca Mido va arunca excepție în caz contrar. Pentru valori mai mici de 0 nu o să avem caz pentru că este imposibil să primim un număr negativ. Metoda care calculează regula de trei simpli este prezentată în figura 21, unde `MIDDLE_A` reprezintă numărul MIDI a notei muzicale A4 iar (69) iar `MIDDLE_A_VALUE` este valoarea 0.5. Cu această metodă vom avea o distribuire în ceea ce privește aceste valori. Notele joase vor fi simțite cu o intensitate mai mare în timp ce notele mai înalte se vor simți mai puțin (în acest caz valoarea nu scade prea mult, va fi în jurul valorii 0.2, ceea ce încă se poate simți).

Formula cu care se calculează frecvența notelor muzicale a fost deja descris în subcapitolul 3.3. Utilizarea acestor valori (înafară setarea vibrației) se va discuta în subcapitolul ce urmează, acesta fiind folosit și în aplicația Android.

```
@staticmethod
def _rule_of_3(note):
    return (constants.MIDDLE_A * constants.MIDDLE_A_VALUE)/note
```

Figura 21: Regula de 3 simpli

4.2.3 Implementarea firelor de execuție pentru evitarea blocajului

- descriere de ce este nevioe in acest caz sa folosim fire de executie

- descrierea modului de implementare a acestor fire de executie

4.2.4 Comunicarea dintre firele de execuție

Datorită faptului că această aplicație se rulează pe mai multe fire de execuție este important să avem o comunicare bine sincronizată dintre aceastea în momentul când dorim să schimbăm mesaje între ei. Utilizarea socket-urilor impune folosirea unui fire de execuție separat ceea ce înseamnă că în momentul trimiterii sau a primii unor date, trebuie să avem acces la o memorie comună la care au acces ambele fire de execuție pentru a putea face schimb de informații. În cazul acestei aplicații trimiterea acestor datelor se face de pe ambele părți: atât aplicația Android cât și cea de pe placa de dezvoltare trebuie să trimită și să primească informații, ce vor fi parsate în următoarele etape.

Gestionarea memoriei comune în aplicația de pe placa de dezvoltare se întâmplă cu ajutorul unei cozi. În momentul în care se primesc date, acestea sun decodate și puse în coadă. Celalalt fir de execuție, care trebuie să parseze aceste mesaje, preia informațiile din ea. Coada nu rămâne niciodată plină, mereu se scoate toată informația din ea de către celălalt fir de execuție. Pentru a putea implementa aceste acțiuni avem nevoie să știm momentul în care putem prelua aceste informații din coadă, din moment ce acestea nu vin încontinuu.

Evenimentele din biblioteca `threading` ne vin în ajutor în momentul în care dorim să primim o notificare că s-a întâmplat un lucru, fiind cel mai simplu mod de a comunica unui al fir de execuție că a avur loc a acțiune. Acestea funcționează ca niște booleane care pot fi setate în momentul în care s-a întâmplat o acțiune. În cazul nostru de fiecare date când se pune sau se ia un element în coadă, acest eveniment este setat pe `True` respectiv pe

`False` (a se vedea figura 22). Verificarea stării acestui eveniment se face prin apelarea metodei `is_set()`. Setarea evenimentului se verficiă în momemntul

```
def enqueue_message(message):
    message_q.put(message)
    ev.set()


def dequeue_message():
    message = message_q.get()
    message_q.task_done()
    ev.clear()
    return message
```

Figura 22: Setare eveniment

în care citim datele de la pian. Datorită faptului că mesajele ce vin de pe aplicația Android influențează modul în care motoarele vor vibra, trebuie să preluăm și să parsăm aceste mesaje înaintea să setâm valoarea și frecvența motoarelor. În momentul în care acest eveniment a fost setat pe `True`, se apelează o metoda din figura 23 care va procesa aceste date iar în funție de rezultate se vor modifica niște valori dintr-un dicționar. Mesajele primite vor fi mereu compuse din două cuvinte, deci parsarea lor este una ușoară. După ce mesajul a fost scos din coadă, evenimentul va fi setat pe `False` până când un nou mesaj va fi introdus.

```
def _processEvent(self):
    message = bridge.dequeue_message()
    note_type, value = Helper.parse_dequeued_message(message)
    print("Message received in the readmidi class {}".format(message))
    if note_type == MESSAGES[Message.SWITCH]:
       self._setValues(value, Message.SWITCH)
    elif note_type == MESSAGES[Message.BASS]:
        self._setValues(value, Message.BASS)
    elif note_type == MESSAGES[Message.TREBLE]:
        self._setValues(value, Message.TREBLE)
    elif note_type == MESSAGES[Message.HIGH]:
        self._setValues(value, Message.HIGH)
```

Figura 23: Parsare mesaje

4.3 Integrarea unei aplicații mobile în proiect

4.3.1 Nevoia de folosire a unui aplicații Android

Aplicația de placa de dezvoltare Raspberry Pi, după cum s-a discutat și în capitolele anterioare, oferă posibilitatea redării muzicii cântate la pian în formă de vibrații. Utilizatorul acestui dispozitiv, fără o aplicație conectată, nu are nici un fel de control asupra acestor lucruri, nu poate să intervină să oprescă sau să personalizeze vibrațiile simțite. Crearea unei aplicații Android pentru acest dispozitiv nu are scopul doar de a controla aceste vibrații dar și de a oferi un feedback vizual în ceea ce privește aceste note muzicale.

În subcapitolele ce urmează vor fi prezentați atât funcționalitățile incluse în aplicația Android cât și algoritmii aleși pentru implementarea acestora.

4.3.2 Prezentarea UI-ului aplicației mobile

Interfața aplicației mobile a fost făcut în așa fel încă să fie cât mai ușor de utilizat.

4.3.3 Metoda de calcul pentru modificarea intensităților motoarelor de vibrații

Modificarea intensităților cu care motoarele conectate vibrează este o funcționalitatea a cărui implementare necesită recalcularea valorilor cu ajutorul niștor formule. Am încercat să aleg metoda cea ma corectă de a calcula aceste schimbări astfel încât rezultatul final să fie unul cât mai corect și mai vizibil. Pe partea de aplicație Android am implementat numai partea de interfață a acestei funcționalități, urmând ca în aplicația de pe placa de dezvoltare Raspberry Pi să se facă calculelel necesare.

În aplicația Android schimbarea valorilor a fost implementată cu ajutorul unei bare de progres. Aceste bare, a căror progrese pot fi setat manual de către utilizator, pot lua valori întregi într-o anumită limită ce poate fi specificată în cod. În cazul nostru aceste valori sunt cuprinse între 0 și 100, valoarea implicită fiind 50 datorită faptului că intensitatea poate să și scadă, nu numai să crească. Astfel dacă un utilizator trage bara de progres până la valoarea 0, intensitatea va scădea iar la 100 va crește cu până la 0.3 de valori. În momentul în care aplicația a sesizat o schimbare de valori în bara de progres, se trimite valoarea curentâ prin socket către aplicația de pe placa de dezvoltare. Mesajul se trimite sub forma unor două cuvinte (a se vedea

figura 24), toate mesajele fiind trimise așa pentru o parsare mai ușoară. Astfel în aplicația scrisă în python, prin despărțirea acestor două cuvinte, putem să ne dăm seama ușor operațiile ce trebuie făcute după primirea mesajelor.

```
if(clientThread != null){
    clientThread.sendMessage("progress " + progress);
}
```

Figura 24: Trimiterea valorii barei de progres

Calculul pentru schimbarea valorii a fost gândită în felul următor: dacă utilizatorul schimbă bara de progress cu 10 valori (de la 50 la 60 sau de la 50 la 40), intensitatea motoarelor se va ridica sau va scădea cu același procent. Așa pentru o schimbare de 50 de valori (de la 50 la 100 sau de la 50 la 0) va produce în ambele cazuri o recalculare a valorilor de vibrații de 0.3. În aplicația python, cu ajutorul unui dicționar este păstrat valoarea implicită a acestui porgres (50), astfel modificările de valori se vor întâmpla doar în cazul în care acest progres este diferit de valoarea implicită (a se vedea figura 25). În această situație este calculat o valoarea adițională care va fi adăugat la valoarea calculată în mod normal.

```
if DEFAULT_VALUES[Message.PROGRESS] != 50:
    changed_amount = Helper._calculate_additional_progress()
    if midiNote == constants.MIDDLE_A:
        return constants.MIDDLE_A_VALUE + changed_amount
    else:
        vibration_value = Helper._rule_of_3(midiNote)
        vibration_value += changed_amount
        print("Calculated vibration value: {}".format(str(vibration_value)))
        if vibration_value > 1:
            return 1
        else:
            return vibration_value
```

Figura 25: Calculul valorii cu pogres schimbat

Metoda care calculează valarea adițională care va fi adăugată are la bază o regulă de trei simpli. Se verifică dacă valoarea progresului trimis de pe aplicația Android este mai mic sau mai mare decât 50 pentru a ști dacă noua valoarea returnată trebuie sau nu să fie un număr negativ. Pentru că știm că valoarea maximă cu care se poate schimba bara de progres este de

50 și în cazul acesta intensitatea se schimbă cu 0.3, este ușor de calculat cu cât se schimbă intensitatea motoarelor de vibrații dacă valoarea acestui bare este diferit de 50. Acest calcul se poate observa în figura 26, unde `MAX_MIN_PROGRESS_VALUE` este valoarea cu cares se poate schimba intensitatea motoarelor de vibrații iar `DEFAULT_VALUE_FOR_SLIDER` este valoarea implicită pe care o are bara și totodată valoarea maximă cu care se poate schimbă ce implicită.

```
@staticmethod
def _calculate_additional_progress():
    if DEFAULT_VALUES[Message.PROGRESS] < 50:
        changed_amount = 50 - DEFAULT_VALUES[Message.PROGRESS]
        return -((changed_amount * constants.MAX_MIN_PROGRESS_VALUE)/constants.DEFAULT_VALUE_FOR_SLIDER)
    elif DEFAULT_VALUES[Message.PROGRESS] > 50:
        changed_amount = DEFAULT_VALUES[Message.PROGRESS] - 50
        return ((changed_amount * constants.MAX_MIN_PROGRESS_VALUE)/constants.DEFAULT_VALUE_FOR_SLIDER)
```

Figura 26: Progress adițional

4.3.4 Realizarea comunicarii cu aplicația RaspberryPi

- conectarea cu sockets

-descriere metode de decodare a datelor ajunse pe aplicatia android

4.3.5 Integrarea vizualizării culorilor cu ajutorul frecvențelor

-descriere metode alese pentru afisarea culorii in Android studio

- explicarea necesitatii frecventelor pentru a realiza acest lucru

- exemplificare cu cod cum s-au calculat reprezentarile hexa

-functionalitatea aceasta este inca in curs de implementare

4.3.6 Prezentarea bazei de date folosite în aplicația Android

5 Concluzii

6 Bibliogragie si webografie

[1] An Introduction To Vibration Motors. https://www.precisionmicrodrives.com/vibration-motors/.

[2] Asynchronous socket handler. https://docs.python.org/2/library/asyncore.html.

[3] GPIO. https://www.raspberrypi.org/documentation/usage/gpio/.

[4] Midi message format. https://www.songstuff.com/recording/article/midi_message_format/.

[5] MIDI Tutorial. https://www.cs.cmu.edu/~music/cmsip/readings/MIDI\%20tutorial\%20for\%20programmers.html.

[6] Mido - midi objects for python. https://mido.readthedocs.io/en/latest/.

[7] 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.

[8] Rachel Elaine. How Deaf People Experience Music, 2017.

[9] Hana Esmaeel. Threads and multithreaded in java. 11 2019.

[10] Janet Heath. Pwm: Pulse width modulation: What is it and how it works?, 04 2017. https://www.analogictips.com/pulse-width-modulation-pwm/.

[11] Michael Lombardi. *Frequency Measurement*, page 24 p. 02 1999.

[12] Stefano Papetti and Charalampos Saitis. *Musical Haptics: Introduction*, pages 1–7. 05 2018.

[13] Jason Reynold. 8 World-Class Software Companies That Use Python. https://realpython.com/world-class-companies-using-python/.

[14] Alexis Rohlin. How to Use MIDI Cable In & Out Plugs. https://smallbusiness.chron.com/use-midi-cable-out-plugs-49492.html.

[15] Samarpit Tuli. Java Thread Tutorial: Creating Threads and Multithreading in Java, 2018. https://dzone.com/articles/java-thread-tutorial-creating-threads-and-multithr.

[16] Ahamed Shibly. Android operating system: Architecture, security challenges and solutions, 03 2016.

[17] Luca Turchet, Carlo Fischione, and Mathieu Barthet. Towards the internet of musical things. 07 2017.