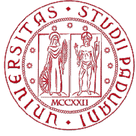

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN
INGEGNERIA DELL'INFORMAZIONE

PROGETTAZIONE DI UN SISTEMA INTERATTIVO INFORMATICO INCLUSIVO PER L'ANALISI VOCALE

Relatore: Canazza Targon Sergio

Correlatore: Fiordelmondo Alessandro

Laureando: Costantin Andrea

ANNO ACCADEMICO 2021-2022
Data di Laurea 22/09/2022

Sommario

In questo lavoro si presentano la progettazione e gli strumenti utilizzati per lo sviluppo di SoundRise, un'applicazione alternativa indirizzata a bambini con disabilità uditive. L'applicazione ha lo scopo di far conoscere le caratteristiche acustiche della voce mediante un feedback visivo.
Il volto di un robot a schermo rappresenterà le caratteristiche del suono: il timbro, l'intensità e l'altezza. L'altezza del suono viene rappresentata dal movimento verticale del volto nello schermo, l'intensità corrisponde alla dimensione del volto e infine il colore del volto del robot corrisponde alla vocale che viene detta, quindi il timbro della voce.
In questa tesi viene trattato solo lo studio dell'altezza della voce, quindi il volto del robottino dovrà solamente alzarsi o abbassarsi in base alla tonalità grave o acuta della voce dell'utente che parla.
Il programma è stato sviluppato tramite Pure Data, un linguaggio di programmazione visuale creato da Miller Puckette, dedicato alla manipolazione e generazione del suono e alla gestione delle immagini tramite la libreria grafica GEM. Inoltre, Pure Data permette di creare applicazioni multi-piattaforma a partire dallo stesso codice.
Verranno presentati gli obiettivi di questo progetto e gli oggetti di Pure Data utilizzati per la realizzazione del prototipo, analizzando poi le diverse sezioni che compongono la patch riguardanti lo studio dell'altezza. Infine, verranno mostrati i risultati ottenuti e i possibili sviluppi futuri.

Ringraziamenti

Vorrei ringraziare prima di tutto i proff. Sergio Canazza e Fiordelmondo Alessandro per avermi dato la possibilità di realizzare questo progetto e per il sostegno che mi hanno fornito in questo tempo. In particolare, grazie al prof. Fiordelmondo che si è reso sempre disponibile e flessibile negli orari per supportarmi nella stesura di questa tesi.

Ringrazio tutta la mia famiglia che mi è sempre stata vicino, soprattutto nei maggiori momenti di difficoltà di questo percorso. In particolare, dico un grandissimo grazie ai miei genitori: senza il vostro supporto non sarei mai riuscito a raggiungere questo importante traguardo.

Un pensiero particolare va poi a tutti i miei amici.

A Mattia, Andrea e Alice, che conosco da sempre e con cui ho condiviso ogni momento importante della mia vita. Finalmente posso aggiungere quest'altro pezzo al puzzle dei ricordi con voi.

A Manuel, Francesco, Nicola, Giacomo, Giulia e Veronica con cui ho condiviso molto e ho avuto bei momenti di spensieratezza utili per distrarmi dalle difficoltà di questo percorso.

Infine Agli amici con cui ho condiviso questo percorso universitario, in particolare Jacopo (che mi ha anche letto e corretto la tesi, sium), Giovanni, Martino e Andrea con cui le lezioni e lo studio sono risultati quasi momenti di "divertimento".

Indice

Sommario I

Ringraziamenti III

1 Introduzione al problema 1

2 Introduzione all'ambiente di lavoro 5
2.1 I linguaggi di programmazione visuali (VPL) 5
2.2 Il linguaggio di sviluppo scelto: Pure Data 6

3 Descrizione del prototipo 11
3.1 Le diverse caratteristiche del suono 11
3.2 Descrizione del codice 14

4 Conclusioni e sviluppi futuri 23

Elenco delle figure

3.1 Moto della particella sulla circonferenza e traslazione del moto . . 12
3.2 Rappresentazione grafica della onda seno e delle caratteristiche del suono 12
3.3 Forma d’onda di un suono puro 13
3.4 Forma d’onda di un suono complesso 14
3.5 Schema a blocchi dell’ADC 15
3.6 Sezione di patch relativa all’avvio della finestra di output 18
3.7 Sezione di patch relativa all’input audio 19
3.8 Sezione di patch relativa all’inizializzazione di GEM 19
3.9 Sezione di patch relativa alla creazione del robot 20
3.10 Sezione di patch relativa allo studio del pitch 21

Capitolo 1

Introduzione al problema

L'avvento della tecnologia portatile, come smartphone e tablet, ha favorito considerevolmente l'utilizzo di strumenti di apprendimento, rendendoli alla portata di chiunque e in qualsiasi momento. Questa portabilità ha sopratutto agevolato lo svolgimento di esercizi che riguardavano anche ambiti extra scolastici. Come si è dimostrato negli anni della pandemia COVID-19, la tecnologia ha portato grandissimi vantaggi nell'ambito dell'apprendimento, permettendo il regolare completamento dei percorsi di studi di tutti gli studenti. Le applicazioni quindi riescono a venire incontro al nostro problema, aiutando con esercizi che mirano al supporto della lacuna dello studente. Queste caratteristiche hanno portato sia insegnanti che terapeuti ad usufruire sempre più di queste applicazioni per migliorare il proprio percorso di insegnamento e supporto. Per esempio, queste tecnologie sono sfruttate nel contesto di supporto per persone con deficit uditivo, in cui software di *speech training*, risultano di grande aiuto per la terapia. Per *speech training* si intendono quei software basati sulla *speech recognition*, ovvero quei programmi che interpretano il linguaggio umano traducendolo in testo o comandi. Questa tecnologia inoltre è sempre in costante sviluppo grazie alle intelligenze artificiali, agli algoritmi di *machine learning* e *deep learning*.
Il lavoro qui presentato, ideato e sviluppato in collaborazione con la dott.ssa Serena Zanolla dell'Università degli Studi di Udine, e dei proff. Federico Avanzini, Antonio Rodà e Sergio Canazza del Dipartimento di Ingegneria dell'informazione dell'Università degli Studi di Padova, riguarda lo sviluppo di una app multipiattaforma a partire dal prototipo realizzato come patch Pure Data. L'applicazione è pensata per il supporto di persone non udenti (o parzialmente) e ha lo scopo di assistere e migliorare il controllo della voce. Questo avviene mediante un riscontro visivo a schermo con dei disegni che interagiscono in base alle carat-

teristiche vocali acquisite.
L'applicazione lavora con quattro caratteristiche del suono:

- altezza (*pitch*): ovvero un attributo di sensazione acustica secondo cui i suoni possono essere ordinati su una scala musicale dal basso verso l'alto;

- intensità: ovvero una descrizione psicoacustica dell'entità di una sensazione acustica secondo cui i suoni possono essere ordinati su una scala dal piano al forte;

- durata: ovvero l'arco di tempo in cui un suono viene percepito o emesso da un corpo in vibrazione;

- timbro: è la caratteristica che ci consente di distinguere il suono di uno strumento da quello di un altro. In questo lavoro viene considerato il timbro delle cinque vocali della lingua italiana, in quanto la caratteristica principale dell'applicazione è dare all'utente la possibilità di sperimentare queste caratteristiche con la propria voce. Di conseguenza l'analisi del timbro non darebbe informazioni coerenti.

In questa tesi verrà trattato solamente lo studio dell'altezza del suono (pitch): come feedback visivo si vedrà il volto di un robottino che si muove verticalmente, alzandosi con il crescere del pitch.
Moltissimi altri progetti simili sono stati sviluppati nel corso degli anni. Un esempio a livello Europeo è OLP (Ortho-logo-pedia)[1], un sistema informatico integrato che completa la normale terapia per aiutare le persone con turbe dell'elocuzione. OLP offre un feedback visuale che aiuta a migliorare l'articolazione del parlato. Inoltre, un sistema automatico di riconoscimento vocale,F valuta i miglioramenti ottenuti e fornisce un'interfaccia con la tecnologia assertiva e/o la sintesi vocale. Con OLP, la terapia può essere eseguita fuori sede, tramite internet e in questo modo, personalizza il tipo di feedback e l'elemento di riconoscimento a seconda di uno specifico gruppo oppure di una sola persona. L'utilizzo del software OLP è utile ai logopedisti che si servono del computer per la rieducazione della parola in vari modi. Contiene mappe fonetiche che visualizzano i suoni vocali bersaglio. I suoni prodotti dai pazienti sono registrati su una mappa fonetica in tempo reale, che comporta anche i suoni bersaglio per una comparazione diretta. In questo modo viene indicata la buona articolazione, per un graduale spostamento verso la fonazione corretta. Le mappe possono essere personalizzate per rispondere ai bisogni dei singoli clienti. Un altro modo in cui il software

OLP risulta utile è che riconosce automaticamente le parole pronunciate e le valuta una per una a fronte di una parola bersaglio stabilita dal logopedista. Infine OLP aiuta a controllare la fonazione visualizzando l'intensità sonora e gli intervalli della voce del paziente.I logopedisti possono conservare una registrazione del cliente e creare per ciascuno di essi un programma di esercizi. Quindi possono scegliere e disegnare gli esercizi pratici più appropriati sia per il cliente che per il terapista.
Altro progetto sviluppato in questo ambito è il "Padua Rehabilitation Tool" (PRT)[2] un software composto da 35 esercizi suddivisi per le varie funzioni cognitive maggiormente coinvolte: attenzione, memoria, linguaggio, ragionamento logico, riconoscimento, orientamento e controllo motorio. Il software è stato sviluppato con un'interfaccia e stimoli appositamente semplificati in modo da consentire un utilizzo più immediato; gli oggetti presentati sono ben visibili e le istruzioni sempre molto semplici e dirette. Grazie alla varietà degli esercizi è possibile sottoporre ad ogni paziente training diversi in base alle proprie difficoltà. Il software è quindi utilizzabile con una varietà di pazienti a prescindere dalla patologia neurologica acquisita: è possibile creare programmi di intervento specificatamente focalizzati sulle necessità di ogni paziente.
Ogni esercizio prevede la medesima modalità di risposta ovvero quello di indicare la risposta corretta tra quelle proposte con un semplice tocco. Viene fornito un chiaro e immediato feedback in modalità sia visiva che uditiva. La difficoltà dei vari compiti è strutturata secondo livelli di crescente complessità, in modo che il paziente abbia la possibilità di iniziare con compiti semplici sperimentando un senso di adeguatezza e supportando la motivazione.
Infine, ulteriore progetto di *speech training* è *Speech Illumina Mentor* (SIM)[3] che si propone come un software indirizzato ai bambini in età scolare con deficit uditivo e si avvale di due componenti: un modulo di 'addestramento che presenta al bambino esempi di produzioni e un modulo di pratica che presenta in forma di gioco un sistema di riconoscimento vocale che analizza le produzioni del bambino. Il modulo di addestramento presenta una animazione del movimento articolatorio esterno e interno, nel secondo caso mostra il movimento dei componenti del tratto orofaringeo normalmente nascosti, come lingua, palato duro, palato molle e faringe. La visualizzazione di queste animazioni avvengono in modo simile all'utilizzo di un riproduttore video dove tutti i parametri della riproduzione sono accessibili (play, stop, forward, rewind, zoom e volume). Durante una sessione di pratica il sistema analizza i tentativi del bambino di produrre la sillaba richiesta e propone una risposta visuale per indicare il grado di successo o fallimento. Si utilizza uno stile giocoso e personaggi animati per stabilire una buona

comunicazione e motivare l'utente. SIM può essere visto come una serie di moduli didattici specifici per l'apprendimento dei diversi fonemi con diversi gradi di difficoltà, tutti utilizzanti la struttura vista precedentemente (un componente di addestramento e uno di pratica).

Capitolo 2

Introduzione all'ambiente di lavoro

2.1 I linguaggi di programmazione visuali (VPL)

Per la realizzazione di questa tesi, è stato ritenuto opportuno procedere nella realizzazione di un prototipo del programma sfruttando un linguaggio di programmazione visuale (detto anche VPL - *Visual Programmig Language*). I *Visual Programming Language*[1] sono dei linguaggi ad alto livello, quindi linguaggi orientati al programmatore, a differenza di quelli a basso livello il cui scopo è quello di essere facilmente eseguibili da un processore. I VPL permettono agli sviluppatori di creare applicazioni descrivendo i processi mediante l'uso di oggetti grafici e vengono ideati partendo dall'idea del flowchart, sviluppata nel 1927. Nonostante però i veri primi utilizzi della VPL si hanno intorno agli anni 1990 in ambito di creazione di videogiochi, strumenti per il multimedia e database, si ha già lo sviluppo di un non-GUI Ada IDE a circa metà del 1980 con l'azienda Rational Software (acquisita successivamente dal IBM nel 2003), che ha inoltre definito il processo di sviluppo software. Lo sviluppo poi ha portato allo Unified Modelling Language (UML) che è stato creato per realizzare un linguaggio di modellazione visivo comune, ricco sia nella semantica che nella sintassi, per l'architettura, la progettazione e l'implementazione di sistemi software complessi sia dal punto di vista strutturale che comportamentale. Lo UML infatti ha il potenziale di sviluppare ogni parte di un sistema senza scrivere alcuna linea di codice e ha definito un linguaggio standardizzato e comprensibile per descrivere sistemi object-oriented, portando però anche alcuni aspetti negativi dal punto di vista dello sviluppo software: la troppa pianificazione (alcuni progetti possono anche richiedere molto

[1]https://www.outsystems.com/glossary/what-is-visual-programming

tempo per strutturare il flowchart del programma senza scrivere alcuna linea di codice) e la scarsa implementazione. Si è pensato poi che l'Exexutable UML potesse fornire una soluzione, solo che con i tentativi svolti si è subito virato ad altri linguaggi di dynamic scripting come PHP, Ruby o Rails. Un esempio interessante di executable UML è stato sviluppato da Rational Software: Rational Rose infatti è una suite di tools che permette di creare software tramite UML e poi generare l'eseguibile in un linguaggio come Java o C++.
La principale differenza tra la programmazione visuale e quella tradizionale dipende soprattutto dai tool messi a disposizione dal linguaggio di programmazione grafico scelto. Per esempio, se dovessimo creare un elenco puntato con un linguaggio grafico, il programmatore deve semplicemente disegnare il flow dell'app. Il flowchart descrive schermate, interazioni dell'utente e i dati ad ogni stage. Successivamente, il tool converte questo flowchart in un software. Gli sviluppatori sanno che i tradizionali linguaggi di programmazione sono incentrati principalmente nell'implementazione: si devono creare precisi step affinché il computer riesca a creare l'esperienza che vuole essere data all'utente. Sicuramente, i linguaggi ad alto livello e i moderni framework, forniscono delle scorciatoie che permettono di accelerare e semplificare alcuni passaggi che a basso livello non possono essere omessi. Alcuni linguaggi di programmazione grafica sfruttano altrimenti l'unione di parte grafica e di testo: per esempio, per la creazione di una schermata di una applicazione si possono posizionare tutti gli elementi grafici che ci risultano necessari tramite VPL e successivamente associare a ciascun elemento una callback in cui si scrive l'azione che deve svolgere.

2.2 Il linguaggio di sviluppo scelto: Pure Data

Per la realizzazione del prototipo dell'applicazione, è stato scelto di utilizzare Pure Data[4], un linguaggio di programmazione visuale open source ideato da Puckette Miller che può essere eseguito praticamente su qualsiasi dispositivo: PC, Raspberry PI e smartphone. Pure Data (abbreviato con PD) permette a musicisti, artisti visuali, performer, ricercatori e sviluppatori di creare software graficamente e senza scrivere alcuna linea di codice. Il suo utilizzo trova maggiore applicazione nella generazione e processo di suoni, MIDI ma anche video e grafica 2D/3D. Inoltre può interoperare in maniera remota con altri strumenti tecnologici (per esempio tastiere MIDI, schede con microcontrollore etc.) e per questo si pone anche come una potente interfaccia per la realizzazione di sistemi complessi. Gli algoritmi sono rappresentati attraverso box visuali, chiamati

oggetti, disposti all'interno di canvas. I flussi di dati attraverso gli oggetti avvengono grazie a particolari connettori visuali chiamate corde. Ogni oggetto può svolgere attività specifiche, da semplici operazioni matematiche a complesse manipolazioni di audio e video come riverbero, trasformazione FFT e decoding del video. Pure Data nasce come tentativo di upgrade per la risoluzione di alcuni problemi che presentava quello che era ,fino alla nascita di PD, il principale ambiente di sviluppo e progettazione di software dedicati ad applicazioni musicali e multimediali in tempo reale, ovvero Max/MSP.
Max/MSP[2] venne progettato e sviluppato all'Ircam di Parigi sempre da Miller Puckette, tra il 1980 e il 1990, dopo una serie di esperimenti nella progettazione informatica. Acquisito poi dalla Opcode nel 1991 per diventare un prodotto commerciale, nel 2000 viene definitivamente assimilato da Cycling '74, venendo ulteriormente ampliato con MSP per il trattamento audio e Jitter per il trattamento di immagini e video. Max è pensato per la manipolazione di dati MIDI, comunica quindi con tutte le apparecchiature che supportano tale controllo. Questo da la possibilità di creare patch/applicazioni/programmi per gestire eventi MIDI in tempo reale. I programmi vengono scritti utilizzando oggetti grafici e non testuali, questo rende la programmazione più intuitiva e riduce la difficoltà di apprendimento semplicemente collegando oggetti tra loro. Max è basato sul linguaggio di programmazione in C: questo permette di espandere le possibilità dell'ambiente scrivendo nuovi oggetti da includere nelle librerie. Supporta inoltre sia Java che Javascript.
Nel corso degli anni di sviluppo di Pure Data, ovviamente ci sono state delle evoluzioni sia a livello hardware che software e tra i miglioramenti più richiesti dagli utenti c'era la possibilità di combinare immagini e suoni mediante l'uso del computer. Quando Max era il programma principalmente utilizzato, si era arrivati alla conclusione che fosse molto meglio descrivere il processo rispetto ai dati come veniva fatto nei classici linguaggi di programmazione e Max presentava anche problemi di discordanza di gestione dati. Quindi Pure Data viene creato per venire incontro a queste richieste, cioè avere tutte le principali caratteristiche di Max (incluso il processo di segnali alla Max/FTS) ma con l'intento di supportare la definizione e l'editing di strutture di dati composte in una maniera più sofisticata rispetto a quella di Max, come per esempio: liste di messaggi Max-style, collezioni di breakpoints di sviluppo o tabelle di array. Le collezioni di dati e gli oggetti Max-style sono rappresentati in una finestra "canvas" unificata e le collezioni di dati possono essere utilizzate come parte della patch. Pd possiede una compatibilità con Max/FTS e sfrutta l'ambiente GEM per la gestione di immagini

[2]https://www.musicainformatica.it/argomenti/maxmsp-2.phpv

e grafica 3-D nella stessa patch.
Prima della nascita di Pure Data, nel 1987, lo sviluppo nella live computer music era dettato dalla serie di processori 4x, sviluppati all'Ircam dal fisico italiano Giuseppe Di Giugno nel lato hardware, mentre nel contesto di ambienti di sviluppo interattivi per la musica c'era Macintosh 4.3. Max uscì come tool per il controllo del 4x tramite MIDI da Macintosh. In quei tempi, si tentava di creare un ambiente di sviluppo che lavorasse in maniera real time che permettesse la diretta manipolazione del suono, e non solo affidarsi al controllo automatico delle ghiere del MIDI. Con i processori 4x e successivi, venne sviluppato l'IRCAM Signal Processing Workstation (ISPW), un processore di segnali digitale (DSP) sviluppato da IRCAM e Ariel Corporation negli ultimi anni del 1980. L'ISPW fu progettato e costruito per avere un livello superiore delle performace, mediante hardware multiprocessore al fine che fossero adatti per la sintesi audio real-time e per il processo di segnali. A livello hardware i livelli necessari erano stati raggiunti, ma allo stesso modo, era necessario compiere dei grossi cambiamenti e avanzamenti anche a livello software. Quindi al posto di dividere un pezzo di codice in eseguibile real-time controllato da GUI ad alto livello, era possibile avere l'editing e le funzioni di manipolazione allo stesso indirizzo della memoria dei calcoli real-time. Max era stato progettato per integrare video, immagini, grafici e audio processing in un unico ambiente di sviluppo real-time e Pure Data ha reso ciò possibile, portando come altra innovazione lo spostamento del lavoro dal dominio del tempo a quello della frequenza. L'interesse per l'analisi real time dell' audio risultava interessante per gesture recognition al fine di controllare vari aspetti dell'audio real-time o del visual-processing. In Max tutto questo era molto più difficile a causa della scarsa presenza di strutture dati e strumenti di manipolazione.
La principale ragione del design di Pd è quella di semplificare il raccoglimento di grandi quantità di dati, alcuni dei quali possono essere ottenuti tramite analisi dei segnali audio (o anche segnali audio stessi) che possono essere considerati come uno spartito per il computer che può essere "suonato" da Pd. Pure Data asomiglia a Max in superficie ma rispetto a quest'ultimo, c'è una migliore gestione delle strutture dati visto che possono avere una gerarchia e sono definibili dall'utente. Nei sistemi Max, poiché la computazione real-time avviene in un multiprocessore, la porzione di "patch editing" del sistema avviene in una macchina differente (chiamata host) rispetto all'ambiente real-time. Questo sistema però costringe ad avere una copia delle strutture dati in entrambe le parti. Una soluzione al problema era quella di spostare le strutture dati in una ulteriore sezione condivisa per contenere una copia di ogni struttura, in modo che potessero essere usate da

entrambe le sezioni. Pd viene a risolvere anche questo problema visto che lavora in ambienti a singolo processore, è possibile mettere l'editor nell'ambiente di sviluppo real-time.
Una patch di Pure Data appare come un insieme di elementi canvas molto simili alle patch di Max, ma possiedono un generale assortimento di elementi. Questi possono includere:

- patchable object come in Max;
- una definizione generale degli array rimpiazzare gli oggetti tab1 e table di Max;
- liste eterogenee e dinamiche di oggetti che sono descritti con coordinate spaziali bidimensionali

La scelta di sviluppare il prototipo è ricaduta su Pure Data per diversi motivi: in primo luogo è uno degli ambienti di sviluppo migliori per la gestione di segnali audio, è open source ed è ricco di librerie per maneggiarli. Secondariamente, è un ambiente di sviluppo multi-piattaforma, quindi permette la distribuzione del prodotto finale a qualsiasi bacino di utenza. Anche dal punto di vista della parte grafica, grazie alla libreria GEM, la creazione della schermata in cui si ha il riscontro visivo delle caratteristiche del suono è stata di facile implementazione. Infine, Pure Data risulta uno strumento facile e intuitivo per svolgere questa prima fase di prototipazione dell'applicazione.

Capitolo 3

Descrizione del prototipo

3.1 Le diverse caratteristiche del suono

La natura fisica del suono corrisponde ad un moto di tipo ondulatorio. Sono onde meccaniche che trasportano energia lontano dalla sorgente sonora che è un oggetto in vibrazione. Ciò che viaggia quindi non è del materiale, ma un segnale, una vibrazione continua di qualche parametro legato all'ambiente in cui avviene la propagazione. Esistono due tipi di onde, longitudinali e trasversali. Le longitudinali sono quelle che l'asse lungo il quale avviene la vibrazione è lo stesso di quello della propagazione (ad esempio, il segnale causato da una pressione esercitata a una estremità della molla), le trasversali invece sono quelle che l'asse lungo il quale avviene la vibrazione è perpendicolare a quello della propagazione (come per esempio il movimento di un estremo di una fune).
Il segnale sonoro è un'onda di tipo longitudinale, è sempre generato da una vibrazione e ogni vibrazione completa è detta *ciclo*. Ogni segnale sonoro comprende molti di questi cicli, quindi può essere descritto in ambito fisico tramite i moti oscillatori, ovvero quando una particella oscilla (o vibra) intorno ad una posizione di equilibrio. Per descrivere l'evoluzione nel tempo di una oscillazione completa conviene descrivere il moto su un cerchio come indicato in figura 3.1: si vede come la particella che compie un intero giro attorno alla circonferenza significa che ha completato una oscillazione. Se oltre a rappresentare la posizione della particella, vogliamo rappresentare la successione delle variazioni della distanza dall'origine con la variazione dell'angolo a cui la particella si trova, si arriva ad ottenere la funzione seno: infatti il fenomeno ondulatorio connesso a tale vibrazione è detto *onda seno* (fig. 3.2). Se come grandezza venisse rappresentato il tempo anziché i gradi, otterremo l'andamento della particella che vibra

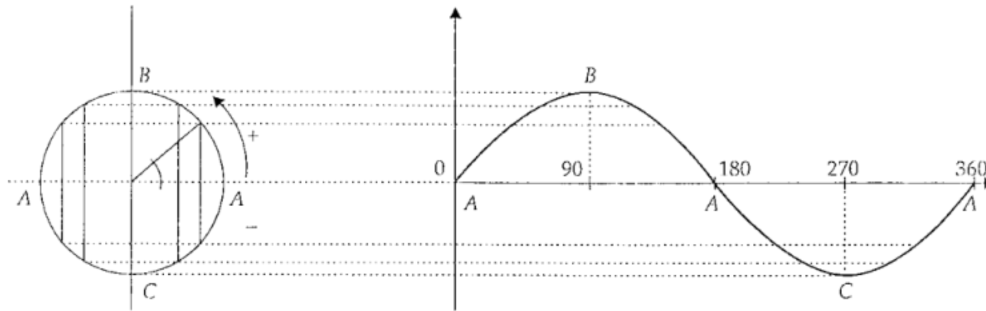

Figura 3.1: *Moto della particella sulla circonferenza e traslazione del moto*

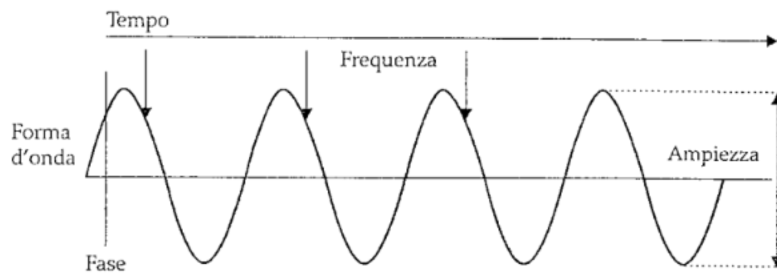

Figura 3.2: *Rappresentazione grafica della onda seno e delle caratteristiche del suono*

nel tempo. Questo cambiamento può risultare utile perché porta ad avere ulteriori informazioni aggiuntive ai fini dello studio del suono. Prima caratteristica che può essere individuata è *l'ampiezza* dell'oscillazione: maggiore è la distanza percorsa dalla particella allontanandosi dalla posizione di equilibrio, maggiore è l'intensità sonora. Suoni deboli (come un bisbiglio) quindi compiranno oscillazioni molto vicine al punto di equilibrio, suoni forti (come un esplosione) farà ampie oscillazioni rispetto al punto di equilibrio.
Altro parametro che può essere studiato invece è la *frequenza*, ovvero il numero di giri che la particella compie nella circonferenza per unità di tempo. Il suo inverso porta ad avere il *periodo* dell'oscillazione, ovvero il tempo che impiega la particella a compiere un giro. La frequenza è un fattore determinante per l'altezza del suono, infatti maggiore è la frequenza, più acuto è il suono.
Ulteriore modo per caratterizzare la velocità di oscillazione di un segnale è la nozione di *lunghezza d'onda*: è la distanza tra due punti identici in cicli adiacenti di un segnale. Nel caso di onde sonore, è la distanza tra due particelle d'aria che si trovano nella stessa fase in cicli adiacenti.

Infine, l'ultimo parametro che caratterizza un segnale è la *forma d'onda*: non è necessario che il segnale abbia per forza un andamento strettamente sinusoidale perché in natura i segnali hanno un andamento molto complesso e la forma d'onda rappresenta una parte importante della caratterizzazione del segnale.
Vediamo ora nel dettaglio quella che sarà la caratteristica che viene analizzata in questa patch di Pure Data ovvero l'altezza (detta anche *pitch*). La frequenza del suono è la principale responsabile dell'altezza del suono che è il parametro legato alla sensazione di acutezza/gravità del suono. Se volessimo categorizzare i suoni in base alla frequenza, si potrebbero suddividere in due categorie: i suoni *puri* e *suoni complessi*.

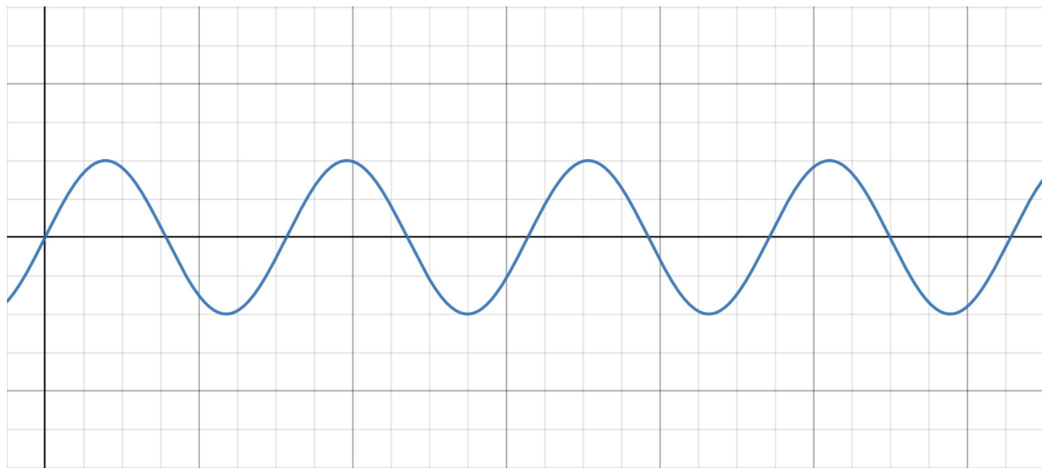

Figura 3.3: *Forma d'onda di un suono puro*

Il suono puro (che è detto anche *tono*, in fig.3.3) è costituito da una sola frequenza ed è quindi descritto da una onda sinusoidale semplice. L'andamento del segnale è molto arrotondato e il periodo è composto da una singola compressione e una singola rarefazione ben definite. Un'onda complessa invece (fig. 3.4) è una somma di onde a frequenze diverse che creano una nuova onda dall'andamento articolato. In questo caso, in un singolo periodo possono essere presenti più alternanze di compressioni e rarefazioni intermedie. Normalmente, i suoni in natura sono di tipo complesso e lo specifico andamento deriva dal metodo di produzione del suono da parte della sorgente, quasi fatta eccezione per il diapason che produce un suono sinusoidale quasi puro (attualmente l'unico modo per riprodurre dei veri e propri suoni puri è solo in laboratorio mediante l'uso di oscillatori). Normalmente i suoni hanno frequenze tra i 20 Hz e 20k Hz, oltre a questi limiti invece vengono definiti *ultrasuoni* (se al di sopra dei 20k Hz) e *infrasuoni* (al di

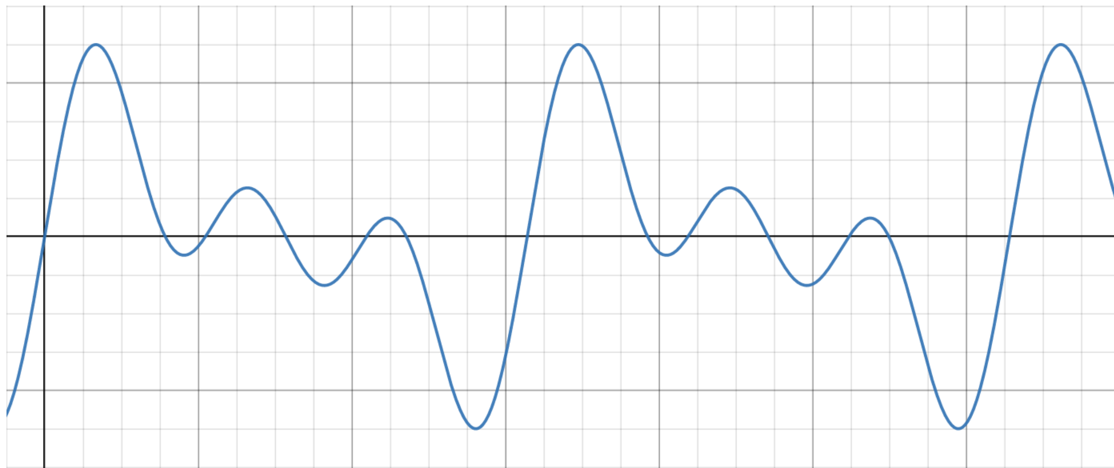

Figura 3.4: *Forma d'onda di un suono complesso*

sotto dei 20 Hz). Un suono complesso contiene diverse frequenze e perché si possa individuare una frequenza speciale occorre che il segnale sia periodico. Ciò in natura non avviene, ma per i suoni musicali (ovvero quei suoni prodotti mediante l'uso di strumenti musicali) si ottengono delle fasi di periodicità significative e per essi ha senso parlare della sensazione di altezza.

3.2 Descrizione del codice

Per riuscire a realizzare un programma che potesse analizzare l'altezza del segnale audio presente all'ingresso del microfono, è stato pensato di suddividere il processo di realizzazione in diverse sezioni, ognuna con un compito ben preciso:

a) *avvio* - in cui c'è l'avvio dell'applicazione e la creazione della finestra dove verrà visualizzato l'output

b) *input* - sezione dedicata alla ricezione del suono presente nell'input

c) *processing* - in cui viene eseguita l'analisi matematica del pitch

d) *output* - dove avviene l'avvio dell'ambiente di sviluppo grafico GEM e la realizzazione del volto del robot usato per il riscontro visivo

Prima di analizzare in modo specifico le sezioni precedentemente descritte, vediamo quali sono gli oggetti fondamentali per la realizzazione di questa patch

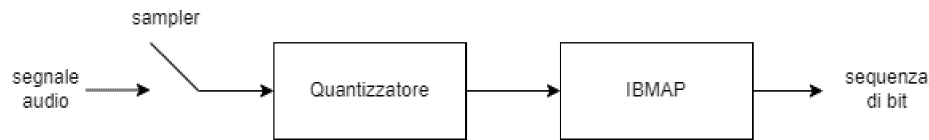

Figura 3.5: *Schema a blocchi dell'ADC*

poichè alcuni di questi sono utilizzati in più parti di codice.
Il primo oggetto importante è l'`adc~` (fig. 3.5) poichè è lo strumento che permette alla patch di elaborare il segnale audio di ingresso del microfono. L'acronimo ADC infatti sta per *Analog to Digital Converter*: il suo compito è quello di convertire il segnale che viene recepito dal microfono (quindi un segnale analogico), in una sequenza di bit (quindi un segnale digitale) in modo da permetterne al computer l'elaborazione. L'ADC in realtà è composto da diverse componenti:

- il *campionatore* (o *sampler*) trasforma il segnale continuo in un segnale discreto dato dall'insieme di campioni presi dal *sampler*;
- il *quantizzatore* approssima ogni valore campionato precedentemente in un valore quantizzato preso da un alfabeto finito di valori;
- l'*inverse bit map* (IBMAP) che rappresenta ogni elemento dell'alfabeto finito in una sequenza di bit.

Altro passaggio necessario per il funzionamento della patch è l'attivazione del DSP di Pure Data. Il DSP, ovvero *Digital Signal Processing*, può essere inteso come l'atto di trasformare una sequenza di numeri che rappresentano un certo segnale nel dominio del tempo discreto in un'altra serie di numeri, agendo su di essi per mezzo di un algoritmo. Le strutture risultanti possono rappresentare una manipolazione esplicita del segnale di ingresso o qualsiasi altra struttura di informazioni sulla natura del segnale di ingresso. Se consideriamo il DSP in tempo reale che genera segnali udibili, il segnale di uscita e quello di ingresso devono essere associati a una frequenza di campionamento. Lo scenario più semplice si verifica quando i segnali di ingresso e di uscita hanno la stessa frequenza di campionamento. In quest'ultimo, un campione di uscita è sempre associato ad uno di ingresso. Il tempo tra l'arrivo di un campione e la disponibilità di uno di uscita è il ritardo tra i segnali di input e di output. Per scopi in tempo reale, il ritardo tra i campioni di ingresso e di uscita deve essere sufficientemente piccolo in modo che l'uscita avvenga allo stesso "tempo percettivo" dell'ingresso. In genere, ciò significa un ritardo di 50 ms o meno. Per l'attivazione del DSP in questa patch,

verrà utilizzato l'oggetto `messagge` collegato ad un `toggle`: quando verrà premuto il `toggle`, verrà inviato un valore logico alto che andrà ad abilitare iol DSP, permettendo al microfono di "catturare" i segnali audio.
Vediamo ora invece l'oggetto `signmud~` (ovvero una nuova versione dell'oggetto `fiddle~`) che permette l'analisi del pitch di un segnale audio. Infatti `signmud~` analizza un suono presente nel suo ingresso in componenti sinusoidali che possono essere riportati individualmente o combinati per avere una stima del pitch. In output da questo oggetto vengono presentati diversi parametri che sono messi poi a disposizione dell'utente e sono:

- *pitch* - il valore del pitch continuo
- *notes* - il valore del pitch ad inizio della nota
- *env* - l'ampiezza della sinusoide
- *peaks* - i picchi delle sinusoidi in ordine di ampiezza
- *tracks* - i picchi delle sinusoidi organizzati per tracce

Possono essere impostati alcuni parametri (come è stato fatto nella patch) e quelli che sono stati usati sono:

- *npts* - il numero di punti in ogni finestra di analisi (di default 1024)
- *hop* - il numero di punti tra ogni ogni analisi (di default 512)
- *npeaks* - numero di picchi sinusoidali (di dafault 20)
- *maxfreq* - massima frequenza sinusoidale in Hz

Successivamente, introduciamo GEM[5], ovvero la libreria inclusa in Pure Data che ha permesso la realizzazione della parte grafica utilizzata come riscontro visivo per l'utente. GEM (*Graphics Enviromet for Multimedia*) è una libreria per Pure Data ideata da Mark Danks che permette processi grafici basati su openGL (*open Graphic Library*), una API dedicata al rendering di vettori bi- e tri- dimensionali. Originariamente GEM nasce come ambiente di sviluppo grafico per Max ma quando iniziò il progetto di Pure Data nella metà degli anni '90, GEM viene trasferito in questo nuovo ambiente di sviluppo. Successivamente nel 2001, Danks si ritirò dal progetto di sviluppo di GEM. Da quel momento questa libreria è mantenuta aggiornata dall' Institut für Elektronische Musik und Akustik (IEM) di

Graz. Questa libreria fornisce diversi elementi geometrici di base(come rettangoli, circonferenze o anche elementi 3D come sferi e cubi) che possono essere usati in uno spazio 3D. Mentre questi oggetti sono molto semplici da utilizzare, non possono essere modificati arbitrariamente (per esempio non è possibile spostare solamente un vertice di un cubo). Prima il design del pixel processing di GEM risultava sub-ottimale, dal momento in cui le immagini colorate erano processate nel formato RGBA, che consuma molto in termini di memoria e potenza di CPU. Attualmente invece ha un formato packed pixel, che incrementa notevolmente il processo dell'immagine.
Nelle precedenti versioni di GEM, il render grafico era compilato in un set di grafiche statiche mentre il rendering è attivato. Questo permette un uso efficiente della lista di oggetti da mettere a schermo. Tuttavia, ci sono alcuni svantaggi nell'usare questa strategia. Per esempio, lavorare su una patch mentre sta renderizzando non è possibile, semplicemente la modifica non viene riconosciuta dal sistema: dopo aver disconnesso un oggetto che era presente in una patch, la modifica non si vede istantaneamente ma è necessario riavviare il motore di render per fare in modo che il render grafico sia ricompilato. Altro problema che si presenta è che la lista degli elementi deve essere ricompilata da openGLsystem tutte le volte che un elemento della lista subisce una modifica (per esempio una rotazione). Questo accelera notevolmente il render di scene statiche, mentre decrementa la velocità di scene dinamiche, in particolar modo se sono presenti oggetti 3D in continuo movimento.
Da questa libreria, sono due gli oggetti più interessanti da prendere in considerazione, ovvero `gemwin~` e `gemhead~`. L'oggetto `gemwin~` permette all'utente di interagire con la finestra di output di GEM. Passa diversi messaggi al window manager, controllando gli attributi della finestra e controlla il render-cycle. Multiple istanze di questo oggetto possono interagire con finestre separate. I parametri da passare in input a questo blocco sono diversi e possiamo suddividerli in tre categorie: input da mandare pre-creazione, input a riguardo della creazione della finestra e input da mandare post-creazione. `gemhead~`, che è il respondabile dell'avvio della rendering chain e connette gli oggetti di GEM al window manager. L'inizio di qualsiasi gemList inizia con `gemhead~` e senza di esso gli oggetti di GEM non ricevono il comando di rendering. Qualsiasi oggetto di GEM che viene connesso dopo `gemhead~` riceve un comando di render per frame e tutte le render-chain sono renderizzate una dopo l'altra. L'utente può controllare precisamente l'ordine di rendering dando a `gemhead~` un valore di priorità. Di default è impostato a 50, e più piccolo è il valore, maggiore sarà la priorità data da `gemhead~` nel dare il comando di rendering. Il minimo valore che può es-

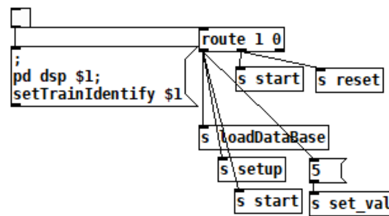

Figura 3.6: *Sezione di patch relativa all'avvio della finestra di output*

sere impostato è pari a 1 ma possono essere anche assegnati valori negativi che ricevono il comando di rendering dopo che sono stati fatti tutti quelli con priorità positiva. In input possono essere messi dei message-boxe convalore 0/1 per abilitare/disabilitare il rendering o un bang per forzare il rendering quando viene premuto. Se non viene messo nulla, `gemhead~` comincia le operazioni dopo la creazione della finestra tramite `gemwin~`. In output invece, viene messa una gemList che è messa a disposizione dell'utente per porci gli oggetti necessari da essere visualizzati a schermo.
Due oggetti che risultano molto utili per migliorare la leggibilità della patch sono `send~` e `receive~`: questi due oggetti lavorano in coppia e consentono una connessione single-to-many non locale per l'invio di un segnale presente nel buffer di input dal blocco `send~` a tutti i blocchi `receive~` che hanno lo stesso nome (più `receive~` infatti possono essere messi in relazione allo stesso send). Entrambi lavorano solamente con blocchi di lunghezza finita del buffer. Molto simile il funzionamento della coppia di oggetti `throw~` e `catch~`: `throw~` infatti permette la somma di segnali per poi metterla in condivisione nel buffer di connessioni non locali degli oggetti `catch~` che hanno lo stesso nome con cui è stato iniziato il `throw~`.

Vediamo ora quindi nello specifico le diverse sezioni realizzate per arrivare all'obiettivo finale.
Come elencato in precedenza, la sezione `avvio` riguarda l'avvio della finestra di output: in figura 3.6, quando viene premuto il `toggle`, viene inviato un valore logico alto a tutti gli oggetti `messagge` collegati e quindi avverranno due cose: viene attivato il DSP, permettendo quindi a Pure Data di campionare i suoni tramite il microfono collegato e contemporaneamente viene inviato un valore logico alto nell'inlet di `route`. In questo oggetto avviene la correlazione tra il primo elemento del messaggio inviato con ognuno degli elementi di `route`. Se il match

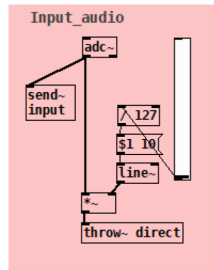

Figura 3.7: *Sezione di patch relativa all'input audio*

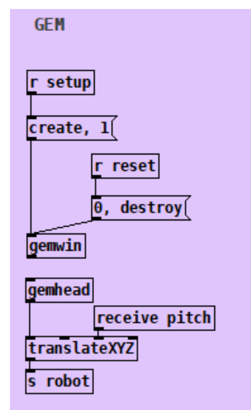

Figura 3.8: *Sezione di patch relativa all'inizializzazione di GEM*

avviene, allora il resto del messaggio viene presentato in output. In questo caso, se il valore di ingresso risulta un valore logico alto, viene creata la finestra in cui è visualizzato l'output di cui verrà spiegato meglio il funzionamento in seguito. Se si presenta un valore logico basso, avviene la chiusura della finestra di output.

Successivamente abbiamo la sezione *input* con riferimento alla figura 3.7, il cui scopo semplicemente è quello di convertire il segnale audio in ingresso tramite l'ADC per poi renderlo disponibile in altre sezioni della patch visto che è collegato all'oggetto `send∼`.

La sezione *output* prende in considerazione due parti: quella di figura 3.8 rappresenta la parte di patch che serve alla generazione della finestra di output e dell'avvio di GEM. Con l'oggetto `messagge` nominato create, viene creata la finestra di output in cui verranno visualizzati i risultati. Con il `messagge` destroy invece possiamo chiuderla. Creata la finestra, `gemhead` è il responsabile di "disegnare" le figure richieste. La lista creata da`gemhead`, passa successivamente

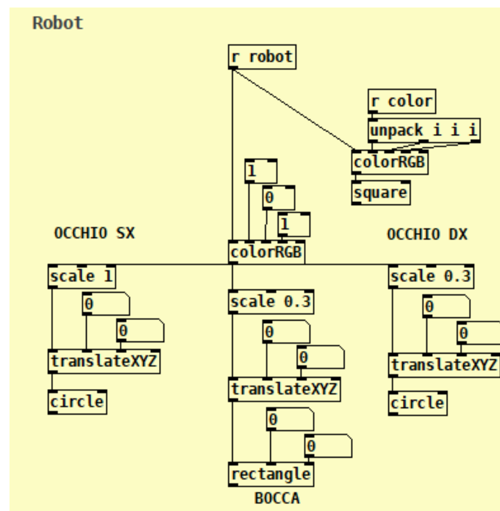

Figura 3.9: *Sezione di patch relativa alla creazione del robot*

per `translateXYZ`, il quale fa in modo che gli oggetti presenti nella schermata, si spostino in verticale in base al valore che il pitch possiede. Infatti, il terzo inlet di `translateXYZ` è responsabile dello spostamento verticale dell'immagine e ci è stato collegato un `receive` del pitch. Infine l'output di `translateXYZ` viene spedito alla sezione riguardante la creazione del volto del robot che verrà di seguito spiegata.

Successivamente, nella figura 3.9, viene rappresentata la parte di patch relativa alla creazione del volto del robot, usato per dare il riscontro visivo. Si è pensato di ricreare il volto di un robot, poiché l'applicazione principalmente era rivolta ad un pubblico di bambini e per la sua creazione sono state utilizzate delle figure elementari: due cerchi per gli occhi, un rettangolo per la bocca e un quadrato per la forma del viso.
In questa sezione della patch all'inizio, è stato messo un `receive~` in cui si riceve la gemList passata dal codice di figura 3.8 la cui uscita è stata poi collegata all'input di quelli che sono i diversi elementi che compongono il volto del robot. La gemList passata tramite il `receive~`, prima di arrivare agli input delle figure, viene fatta passare per alcuni blocchi per ogni figura utilizzata, ovvero `colorRGB`, `scale` e `translateXYZ`.
L'oggetto `colorRGB` accetta in ingresso una gemList e setta il colore di tutte le forme collegate ad esso; `scale` effettua semplicemente una scala della figura a cui è stato collegato; il `translateXYZ` è stato utilizzato per dare alle varie figure la

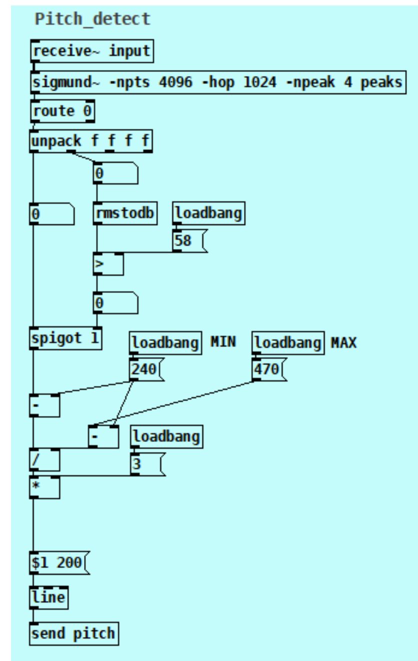

Figura 3.10: *Sezione di patch relativa allo studio del pitch*

posizione di partenza corretta.
Quindi questa parte di patch unisce le diverse figure al fine di creare il volto del robot. Quando nella gemList c'è la modifica del valore della posizione dell'immagine dovuta alla variazione del pitch, la nuova posizione viene inviata a tutti gli elementi che compongono il volto che si sposteranno verso l'alto quando l'utente starà parlando. Maggiore è il valore del pitch, più in alto si sposterà il volto fino a raggiungere il valore massimo della finestra di output. Quando poi si smette di parlare, il programma riporterà in posizione iniziale tutte le figure.

Infine, in figura 3.10, vediamo la parte di *processing*, responsabile dell'analisi del pitch. Ciò che è presente nell'output dell'oggetto `sigmund~`, si spartisce tramite l'oggetto `route`: in questo modo si può accedere alle diverse componenti analizzate. Di queste ce ne interessano solo due: il pitch e il volume. Il volume è stato utilizzato per avere un ulteriore controllo sul segnale audio: se il volume non supera una determinata soglia (in questo caso di 58 dB), il valore del pitch viene bloccato dall'oggetto `spigot`. In questo modo si evitano fastidiosi sfarfallii dell'immagine proposta come output a causa di possibili disturbi presenti nel microfono. Se invece viene superata questa soglia, allora il pitch viene scalato prima di tutto in modo da risultare un valore compreso tra 0 e 1 con la seguente

formula:

$$\frac{pitch - min}{max - min} \tag{3.1}$$

dove con *min* e *max* si indicano rispettivamente il valore minimo e massimo del pitch (che possono essere settati tramite i due `messagge box` nominati con MIN e MAX in figura). Poi viene moltiplicato il valore ottenuto dalla precedente espressione per 3. Il valore del pitch è stato trattato in questo modo perché nella finestra di output, i valori che possono essere dati all'oggetto `translateXYZ` affinché l'immagine del robot restasse all'interno della visuale devono andare tra -3 e 3. Successivamente, il valore scalato del pitch ottenuto, viene fatto passare tramite un `line` (un oggetto che crea una rampa) in modo da avere valori del pitch che cambiano in maniera lineare e non repentina. Questo porta ad una maggiore fluidità nel movimento del volto del robot nella finestra di output. In conclusione, il valore del pitch modificato, viene spedito tramite l'oggetto `send` all'inlet relativo allo spostamento sull'asse y dell'immagine del robot.

Capitolo 4

Conclusioni e sviluppi futuri

In questo lavoro si è arrivati ad avere un prototipo esclusivo per lo studio del pitch. Pure Data si è dimostrato un ottimo ambiente di sviluppo per il prototipo dell'applicazione, poiché ha reso lo sviluppo intuitivo e senza mettere ostacoli per operazioni che in altri linguaggi richiedono una maggiore dedizione, come per esempio la cattura dell'audio da microfono. Essendo questa patch solo una parte dell'intero progetto, il primo avanzamento che può essere fatto è quello di realizzare un prototipo simile anche per le altre caratteristiche del suono. In questo modo, conclusa la prototipazione, si sarà agevolati nello sviluppo dell'applicazione. Ulteriore avanzamento che potrebbe essere fatto è quello di sviluppare un algoritmo basato sul *machine learning*: questo risulterebbe molto utile poiché il microfono che viene utilizzato come input può sempre presentare dei disturbi e quindi l'analisi della voce può risultare incorretta. Con il *machine learning* invece, si potrebbe creare un algoritmo in grado di "apprendere" e differenziare l'errore sul segnale vocale, in modo da eliminarlo per avere un risultato più accurato. Per l'applicazione sarebbe fondamentale anche lo sviluppo per dispositivi mobili e multi-piattaforma, in modo da raggiungere un altro obbiettivo del progetto, cioè di essere usufruibile da tutti in qualsiasi istante. Per questo scopo si potrebbe usare Pure Data (come detto in precedenza è supportato dai dispositivi mobili) ma tuttavia presenta delle limitazioni che altri linguaggi ad alto livello (come Python) non hanno, visto che sono utilizzati da un bacino di utenti molto più grande e presentano librerie maggiormente sviluppate. Per migliorare ulteriormente l'applicazione, potrebbe essere integrata con un sistema cloud, sul quale potrebbero essere salvati i risultati degli esercizi svolti degli utenti. In questo modo chi sta cercando la migliore terapia per l'utente, può vedere gli avanzamenti e in caso, modificare il gruppo di esercizi da svolgere in base ai risultati ottenuti.

Bibliografia

[1] Öster, Anne-Marie and House, David and Protopapas, Athanassios and Hatzis, Athanassios, (2002) *Proceedings of the XV Swedish Phonetics Conference (Fonetik 2002)* Presentation of a new EU project for speech therapy: OLP (Ortho-Logo-Paedia) https://www.researchgate.net/profile/Miller-Puckette/publication/230554908_Pure_Data/links/577c1cca08aec3b743366f5c/Pure-Data.pdf

[2] Cardullo, Stefano and Valeria, Pes Maria and Ilaria, Tognon and Ambra, Pesenti and Gamberini, Luciano and Mapelli, Daniela, (2015) *International Conference on Games and Learning Alliance* Padua rehabilitation tool: A pilot study on patients with dementia https://www.sciencedirect.com/science/article/abs/pii/S0967586819305600

[3] Soleymani, Ali JA and McCutcheon, Martin J and Southwood, (1997) *Proceedings of the 1997 16 Southern Biomedical Engineering Conference* Design of speech illumina mentor (SIM) for teaching speech to the hearing impaired https://ieeexplore.ieee.org/abstract/document/583205

[4] Puckette, Miller S and others, (1997) *ICMC* Pure data https://www.researchgate.net/profile/Miller-Puckette/publication/230554908_Pure_Data/links/577c1cca08aec3b743366f5c/Pure-Data.pdf

[5] Zmölnig, Johannes M, (2004) *ICMC*, Gem for pd-recent progress, https://puredata.info/downloads/gem/documentation/manual/pub/zmoelnig2004gem.pdf