UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea triennale in Ingegneria Elettronica

Analisi e implementazione di controlli basati su sensori di movimento e applicati a sistemi audio

Analysis and implementation of controls based on motion sensors and applied to audio systems

Candidato:
D'Amen Elia

Relatore:
Prof. Squartini Stefano

Correlatore:
Ing. Caldari Marco

Anno Accademico 2021-2022

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea triennale in Ingegneria Elettronica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Ringraziamenti

Un ringraziamento sentito agli ingegneri Marco Caldari, Franco Ripa e a Korg Italy per l'ottima esperienza di tirocinio presso la filiale di Korg a Osimo e al Professor Stefano Squartini, che mi ha permesso di realizzarla.
Ringrazio anche i miei familiari per avermi supportato negli studi, in particolare mio fratello Gabriele per avermi dato consigli relativi alla scrittura della tesi.

Ancona, Ottobre 2022

D'Amen Elia

Sommario

Tutti i giorni ci interfacciamo con i dispositivi elettronici utilizzando delle periferiche come tastiere e *touch screen*; l'interfacciamento tra uomo e macchina si è evoluto negli anni con l'obiettivo di rendere l'interazione più naturale, interattiva e confortevole. In questa tesi viene approfondito l'utilizzo di una tecnologia di interazione ancora poco convenzionale che è stata spesso legata all'intrattenimento, la *gesture recognition*, in particolare valutandone l'utilizzo in ambito audio e implementando un sistema di controllo basato sul gesto umano per comunicare con un programma di sintesi sonora installato in un PC.
Nel primo capitolo viene realizzata una descrizione di questa tecnologia partendo dalle sue caratteristiche, e delle sue possibili implementazioni come interfaccia per un'applicazione di elaborazione del suono, con finalità come strumento di performance artistica, di riabilitazione di individui con disabilità motorie e funzionalità legate al mondo della realtà aumentata.
Successivamente, il secondo capitolo introduce una panoramica dello stato dell'arte in questo ambito, citando alcuni dispositivi che hanno avuto successo commerciale con utilizzi analoghi a quelli del progetto proposto da questa tesi, le tendenze di mercato del settore audio e argomenti di ricerca legati a un utilizzo di *machine learning* associato a questo tipo di controllo per creare tecniche di interazione più complesse.
Infine, gli ultimi due capitoli della tesi trattano lo sviluppo del sistema di controllo partendo dalle informazioni acquisite nei capitoli precedenti; nel capitolo tre vengono elencati e descritti i componenti utilizzati: i sensori che catturano il gesto, le schede elettroniche adibite all'acquisizione di dati dal sensore e alla comunicazione con il PC, i *software* utilizzati per la scrittura dell'applicativo e per il processamento delle informazioni atte a effettuare l'elaborazione del suono.
L'ultimo capitolo riguarda la procedura per l'implementazione del sistema, lo schema utilizzato e la descrizione delle due applicazioni risultanti alle quali viene allegato il link di una *repository* su *Github* dove sono stati caricati i *file* del progetto; a ciò segue una valutazione del sistema basato sulla latenza e sull'accuratezza.
Questo progetto universitario è stato realizzato con la collaborazione di Korg Italy Spa.

Indice

Indice vi

Elenco delle figure viii

Elenco delle tabelle ix

1 *Gesture recognition* 1

1.1 Interazione tra uomo e macchina 1
1.2 *Gesture recognition* e applicazioni a un sistema audio 1
1.2.1 *Gesture recognition* come performance musicale 1
1.2.2 *Music therapy* 2
1.2.3 Ascolto immersivo 3
1.3 Tipologie di *Gesture Recognition* 4

2 Stato dell'arte 6

2.1 Dispositivi con interfacce analoghe al progetto 6
2.2 Tendenze delle tecnologie applicate al settore audio 8
2.3 Ulteriori argomenti di ricerca 9

3 Strumenti *hardware* e *software* 10

3.1 Componenti *Hardware* 10
3.1.1 Sensori 10
3.1.2 Schede elettroniche 13
3.1.3 *TFT Touch Shield* 14
3.2 Componenti *Software* 15

4 Sistemi implementati 16

4.1 Schema di base del sistema di controllo 16
4.1.1 Comunicazione tra Sensore e *Board* 16
4.1.2 Comunicazione tra *Board* e PC 17
4.2 Prima applicazione 18
4.3 Seconda applicazione 19
4.3.1 *Firmware* su Arduino 19
4.3.2 Applicazione su Max/Msp 21
4.4 Valutazione della seconda applicazione 22
4.4.1 Latenza 22

4.4.2 Accuratezza del *tracking* e condizioni di utilizzo 26

5 Conclusioni **28**

Bibliografia **30**

Elenco delle figure

1.1 Risultati di un sondaggio riguardante le interfacce in ambito musicale 2

2.1 Korg Kaoss Pad 3+ 6
2.2 Hot Hand USB 7
2.3 Kinect 7
2.4 LeapMotion 8

3.1 Sensore Apds-9960 10
3.2 Sensore Paj7620U2 11
3.3 Lista dei registri accessibili alla lettura in *ProximityMode*. 13
3.4 *Board* Arduino Nano 33 13
3.5 *Board Arduino Due* 14
3.6 *Display TFT Touch Shield* 14

4.1 Schema delle connessioni tra i dispositivi che compongono il sistema 16
4.2 Visualizzazione di una comunicazione tra *master* e *slave* in I2C . . . 16
4.3 Modello dei canali di comunicazione USB. 17
4.4 Funzionamento della prima applicazione 18
4.5 Funzionamento della seconda applicazione 19
4.6 Schema a blocchi del funzionamento della scheda 19
4.7 Sezione del programma adibita alla lettura su Seriale 21
4.8 Sezione del programma che discrimina e normalizza i dati ottenuti . 21
4.9 Conversione del file mp3 in formato *Ambisonics*/Binaurale 22
4.10 Schema della valutazione dei ritardi 22
4.11 Sistema utilizzato per effettuare le misure 23
4.12 Istogramma delle probabilità della durata totale di un ciclo del programma caricato su Arduino. 24
4.13 *Round Trip Time* tra Arduino e il PC calcolato con PuTTy 25
4.14 Latenza audio utilizzando *Asio4all* 26
4.15 Latenza audio utilizzando *FocusriteUSB* e una scheda audio esterna 26
4.16 Range di funzionamento indicato nel *datasheet* del sensore 27

Elenco delle tabelle

3.1 Caratteristiche del funzionamento del sensore influenzate dalla *gaming mode* 12

4.1 Tempi di esecuzione del *firmware* su Arduino. 24
4.2 Tempo di trasferimento tra Arduino e il PC 25

Capitolo 1

Gesture recognition

1.1 Interazione tra uomo e macchina

L'evoluzione dei computer da super calcolatori a dispositivi tascabili ha reso la presenza di questi dispositivi onnipresente in tutti gli aspetti della vita.
Comunemente comunichiamo con essi utilizzando delle *Human Computer interfaces*, come *touchpad*, tastiere e periferiche audio/video, che utilizzano sensori e attuatori per ricevere e inviare informazioni tra il calcolatore e il mondo esterno; a volte, però, questi mezzi non consentono di avere un grado di controllo adeguato all'applicazione.
Da questa necessità è nato un ramo nella ricerca che si occupa della *Human Computer Interaction*, che pone l'enfasi sul design e sullo sviluppo di interfacce capaci di soddisfare i criteri di performance richiesti dai moderni dispositivi elettronici [1]; in un sistema di controllo dinamico questo permette l'utilizzo di controlli remoti, come gesti o mimica facciale (istintivi per l'essere umano come trigger di situazioni complesse) che non richiedono un contatto fisico con il dispositivo.
Tra i sistemi di comunicazione impiegati per questo compito, il riconoscimento dei gesti dell'utente è uno dei più utilizzati in vari campi [2], tra i quali il supporto per persone diversamente abili, applicazioni mediche, *gaming* e *virtual reality*.

1.2 *Gesture recognition* e applicazioni a un sistema audio

Il fulcro di questa tesi è quello di studiare una corrispondenza che associ dei gesti umani, utilizzando un'interfaccia di *gesture recognition*, con dei controlli di un sistema di sintesi sonora.
A seguire, vengono descritte alcune applicazioni per un sistema di questo tipo:

1.2.1 *Gesture recognition* come performance musicale

Storicamente, l'ergonomia e il funzionamento di strumenti e controller elettronici, come le tastiere MIDI, si basa sulle forme degli strumenti acustici, come pianoforti e chitarre, tuttavia gli strumenti elettronici odierni ci permettono di controllare molte variabili del suono, e utilizzare un'interfaccia tradizionale limita il controllo attuabile sui parametri della sintesi sonora [3].
Al tempo stesso, lo strumento acustico conferisce molti più gradi di libertà al musicista,

poichè agisce con un paradigma che associa un gesto a un evento [4], controllandone meccanicamente le caratteristiche; un esempio lampante di ciò si può trovare nella musica adoperante tecniche estese [5], che richiedono al musicista l'esecuzione di un particolari gesti e sollecitazioni meccaniche sullo strumento per ottenere il suono o timbro desiderato.
Acquisisce perciò senso sviluppare un'interfaccia che riconosca gesti complessi per controllare un'applicazione musicale.

Figura 1.1: Risultati di un sondaggio riguardante le interfacce in ambito musicale

A sostegno di ciò sono stati raccolti dei riscontri in un sondaggio riguardante le interfacce *touch* in ambito musicale [6] avente 39 musicisti come partecipanti; 25 di essi hanno riportato un feedback molto positivo per quanto riguarda l'utilizzo di gesti per controllare la sintesi sonora, prediligendo un tipo di controllo continuo, come mostrato in Fig.1.1

Cenni sui controller MIDI

L'approccio convenzionale per far comunicare un input esterno e un suono nei sistemi *real time* è quello di combinare un controllore o chip DSP (*digital signal processing*) con dei convertitori A/D e D/A e il controller seriale MIDI [4].
MIDI è l'acronimo di *Musical Instrument Digital Interface* [3], uno standard introdotto nel 1983 che rende possibile la comunicazione dei dispositivi audio attraverso un *bus* seriale *daisy chain*, un *bus* dove i dispositivi *slave*, ovvero le perifieriche che leggono il segnale di *clock* propagato da un dispositivo *master*, propagano il segnale di *data* da un dispositivo *slave* al successivo, creando così una "catena".
I messaggi MIDI contengono informazioni su parametri di eventi acustici, ed è in questo modo semplice introdurre un computer nella catena di dispositivi.

1.2.2 *Music therapy*

L'utilizzo di questo tipo di interfacce ha applicazione anche nel campo della *Active music therapy* [7], cioè la riabilitazione di individui con disabilità fisiche e motorie tramite l'interazione con strumenti musicali.
Un esempio di ciò è il progetto *Instruments Sans Frontières*, che per questo compito utilizza un'interfaccia di improvvisazione musicale basata sul sistema di tracking di una webcam riscontrando buoni risultati con un incremento nell'attenzione e nella motivazione dei pazienti [7].

1.2.3 Ascolto immersivo

Un campo di interesse nel mondo dell'elaborazione del suono è quello della spazializzazione, ovvero l'insieme dei fenomeni fisici e psicoacustici che ci porta a identificare una sorgente sonora in una determinata direzione; nella scorsa decade si è osservato un incremento nella ricerca di sistemi audio immersivi, ed una rimodellazione di questo campo di studi, accellerando l'adozione, da parte del mercato, di tecnologie legate a questa branca [8].
È risaputo che l'aggiunta di interattività, come un controllo basato su *gesture recognition*, a queste tecnologie contribuisca a incrementare la profondità dell'esperienza immersiva, implementandole in applicazioni di *Virtual Reality*, che, tramite l'utilizzo di dispositivi indossabili con funzionalità di *head tracking*, rendono possibile variare il punto di vista visivo ed uditivo dell'utilizzatore, questo grazie alle nuove tecniche di *rendering* e architettura audio [8].
Il formato che è diventato lo standard nelle applicazioni *Virtual Reality*, anche grazie al supporto di Google e Facebook è *Ambisonics*, che ha storicamente avuto un successo commerciale limitato rispetto ad altri formati audio come *Dolby Surround*. *Ambisonics* codifica il campo di pressione sonora della sorgente tramite coordinate polari utilizzando delle funzioni chiamate Armoniche sferiche

$$p(kr, \theta, \varphi) \tag{1.1}$$

dove k è il numero d'onda, ovvero $1/\lambda$ e r, θ, φ sono rispettivamente raggio, elevazione e azimuth del campo sonoro rispetto all'ascoltatore.
La riproduzione avviene tramite più speaker posizionati sfericamente attorno all'ascoltatore; il numero di speaker richiesti aumenta in base all'ordine delle Armoniche sferiche riproducibili dall'impianto *ambisonics*, ed in base a ciò varia il numero di armoniche sferiche riprodotte e quindi l'accuratezza spaziale del suono.
A causa dell'uso onnipresente delle cuffie, è di importanza critica l'implementazione di conversioni dai formati di audio immersivo a un formato stereo binaurale, che può essere effettuato tramite la convoluzione del segnale audio con le *Head Related Impulse Response*, delle risposte impulsive ottenute applicando una trasformata nel dominio del tempo alle funzioni di trasferimento di un suono che giunge all'orecchio umano.
Il risultato di questa conversione genera dei battimenti nell'audio, ovvero il fenomeno psicoacustico basato sulla riproduzione di due segnali a frequenza vicina in due canali stereo, che verranno interpretati dal cervello umano come un singolo segnale localizzato a un'angolazione variabile sul piano XY dipendente dalla differenza di frequenza tra i due; il formato binaurale produce un risultato piacevole, ma richiede un'operazione di *cross-cancellation* tra i due canali se riprodotti su speaker non a contatto con le orecchie, ed è inflessibile alle manipolazioni in post-produzione.
Queste tecnologie si stanno ritagliando un ruolo nell'industria del film, dell'*home theater* e dell'*automotive* e non è improbabile che ci sia una crescita esplosiva nel-

l'immediato futuro di applicazioni di *Immersive Audio Conferencing*, grazie alla maggiore connettività (es.5G) e dispositivi di comunicazione più versatili, prendendo come esempio il recente supporto per l'audio immersivo nelle *Apple AirPods Pro*, che possono inviare i dati dei movimenti aquisiti con giroscopio e accelerometro al dispositivo Apple tramite Bluetooth e ricevere l'audio binaurale opportunamente spazializzato.

1.3 Tipologie di *Gesture Recognition*

Un gesto può essere classificato in base alla parte del corpo che lo coinvolge, come la mano, la testa, o i muscoli facciali, oppure richiedere il coinvolgimento di tutto il corpo, ed è qualitativamente caratterizzato dalle sue coordinate nello spazio, dal percorso fatto e dal significato simbolico, che può essere specifico di una cultura o un linguaggio [9].
Lo scopo di un controllo tramite gesti viene classificato in comunicativo o manipolativo [10]: nel primo caso è richiesto che l'applicazione utilizzi un vocabolario di gesti che verranno poi tradotti; mentre se è manipolativo posso semplicemente ricondurmi alle variabili spaziali e di movimento che vengono captate dai sensori.
A seguire una descrizione delle più comuni tecnologie utilizzate per il riconoscimento dei gesti:

- **Portable device**
  Questa tipologia comprende tutti i dispositivi di *gesture recognition* che possono essere indossati dall'utilizzatore, detti anche *wearable* e sono stati tra i primi di questo ambito a essere ideati [9].
  Restituiscono una ricostruzione in 3 dimensioni del moto della parte del corpo a contatto con il dispositivo precisa e sensibile a piccoli spostamenti, però, per alcuni utilizzi possono risultare ingombranti.

- **Image based**
  I dispositivi *Image based* utilizzano una fotocamera per acquisire i movimenti e sono più pratici rispetto al caso precedentemente descritto perchè *contactless*, tuttavia devono fare i conti con la visibilità della parte del corpo esaminata e con la difficoltà a catturarne la profondità; tendenziamente un approccio di questo tipo sarà più semplice da implementare, ma restituirà dei dati che corrispondono a un senso generale del movimento, variando in precisione in base al numero di telecamere usate, alla latenza dell'applicazione e al tipo di struttura utilizzato a basso livello per il riconoscimento dell'immagine, chiamata regione [9].

- **Doppler sensing**
  Questa tecnica fa affidamento sull'invio di onde a ultrasuoni a un oggetto e la cattura delle riflessioni ottenute, una pratica comunemente usata in campo

medico ed in analisi strutturali; acquisendo tramite sensori appositi la variazione di frequenza fd indotta dall'effetto *Doppler* di un oggetto in movimento a una velocità v è possibile ricavarne la traiettoria, dato che

$$fd = 2vf/(c - v) \tag{1.2}$$

I vantaggi che porta questa tecnologia sono vari [11]: può funzionare in assenza di luce, e a differenza dell'*image based gesture recognition* l'immagine ottenuta non viene rimpicciolita con l'allontanamento dal sensore, a patto che il segnale abbia un rapporto segnale rumore sufficiente al riconoscimento, inoltre richiede l'utilizzo di meno dati rispetto alle altre metodologie.
Tuttavia, questi vantaggi vengono bilanciati dal maggiore costo dei componenti, in particolare dei sensori *Micro-Doppler* che mi consentono di catturare le riflessioni.

Capitolo 2

Stato dell'arte

2.1 Dispositivi con interfacce analoghe al progetto

Viene riportata in questa sezione una selezione di dispositivi popolari che implementano o hanno implementato delle interfacce di controllo affini a quelle descritte sino ad ora:

- **Korg Kaoss Pad**

Figura 2.1: Korg Kaoss Pad 3+
[1]

La serie di dispositivi *Kaoss pad*, lanciata da Korg nel 1999, raggruppa delle unità di campionamento ed effettistica audio il cui successo è stato decretato dal design intuitivo del controllo tramite un touchpad, le cui aree, divise in tanti piccoli quadranti, sono utilizzabili per effettuare un controllo in tempo reale delle variabili degli effetti sonori in modalità difficilmente replicabili da unità di effetti convenzionali.
L'ultimo prodotto di questa linea, rilasciato nel 2013, è il KP3+, con 150 effetti combinabili e funzionalità di *looping* dei campioni [12].

- **Source Audio Hot Hand**

[1] Per gentile concessione di `https://www.korg.com/it/products/dj/kaoss_pad_kp3_plus/`

Figura 2.2: Hot Hand USB
[2]

Il controller MIDI *Hot Hand USB* dell'azienda *Source Audio* è un dispositivo di *gesture recognition* che utilizza una tecnologia *device based*: è costituito da un anello wireless indossabile con un accelerometro che aquisisce i dati del movimento della mano su tre assi e li comunica a un ricevitore USB, ovvero *Universal Serial Bus*, il quale è connesso a un PC che riceve i valori ottenuti in formato MIDI.
Ha un range di funzionamento molto ampio, fino a 100 ft., un *tracking* preciso ed è utilizzabile con tutti i *software* audio che utilizzano MIDI [13].

- **Microsoft Kinect**

Figura 2.3: Kinect
[3]

Microsoft kinect è una periferica di *motion sensing* inizialmente sviluppata per funzionalità di *gaming*, poi utilizzata anche per altre applicazioni.
Il suo funzionamento si basa sull'utilizzo di una videocamera, un sensore a infrarossi per acquisire informazioni sulla profondità dell'immagine e una disposizione di quattro microfoni che consente anche di implementare funzionalità di *voice recognition* [14].
SimpleKinect è un'applicazione di interfacciamento sviluppata da Jon Bellona per essere utilizzata assieme a questo dispositivo, consente di tradurre la posizione degli arti in messaggi OSC, capaci di controllare applicazioni audio [7]

[2] Per gentile concessione di `https://www.sourceaudio.net/hot-hand.html`
[3] Per gentile concessione di `https://www.pcmag.com/news/adapter-lets-you-use-xbox-one-kinect-sensor-with-windows`

e per far ciò implementa algoritimi di auto-calibrazione e di *skeleton tracking* [15].

- **Leap Motion controller**

Figura 2.4: LeapMotion
[4]

Il controller *Leap Motion* è una periferica USB rilasciata nel 2013 che utilizza telecamere a infrarossi per effettuare il riconoscimento delle locazioni delle mani e delle dita, rendendo possibile all'utente un'interazione gestuale con il computer, ad esempio afferrando e sfogliando pagine, con utilizzi anche nel campo della manipolazione in interfacce grafiche 3D [7].
Utilizzando l'API (*Application Programming Interface*) fornita con il dispositivo, che fornisce strumenti per sviluppare applicazioni utilizzando il *tracking* della mano, è possibile creare interfacce di *gesture recognition* che forniscono messaggi MIDI, utilizzabili in campo musicale, come Geco, o l'oggetto *aka.leapmotion* su Max/Msp [7].
Tra i progetti citabili che hanno fatto utilizzo di *Leap Motion* vi è [A]2 [7], uno spettacolo musicale dal vivo facente utilizzo dell'applicazione di sintesi Max 3.2 per modulare degli effetti sonori controllandone le caratteristiche sui tre assi del range di visione dei sensori.

2.2 Tendenze delle tecnologie applicate al settore audio

Facendo una panoramica sul mercato globale dei dispositivi audio ci si aspetta una crescita dell'indice CAGR, del 5,4% dal 2022 al 2026 [16]; ciò è dovuto alla ripresa post COVID-19 e interesserà soprattutto il comparto *wireless* dei dispositivi audio, come cuffie e *speaker* con funzioni di connessione BLUETOOTH, che stanno ormai divenendo lo standard per la fruizione di prodotti musicali.
CAGR è una sigla che significa *Compound Annual Growth Rate*, il ritorno degli investimenti in un periodo di un anno, usato per indicare la crescita di un settore, mentre BLUETOOTH è un protocollo di comunicazione utilizzato anche nel campo della sensoristica dei sistemi di controllo.
È stato inoltre annunciato a inizio 2020 un nuovo standard BLUETOOTH *LE AUDIO*, ideato appositamente per l'utilizzo sopracitato che garantisce una migliore

[4] Per gentile concessione di `https://www.ultraleap.com/product/leap-motion-controller/`

qualità di ascolto e un canale a basso consumo (*LE - low energy*); per ottenere ciò è stato implementato un nuovo CODEC, ossia un sistema di codifica delle informazioni digitali più efficiente e un meccanismo di connessione che consente di inviare i dati a più dispositivi associati (broadcast) su canali isocroni.
Inoltre la connessione viene mantenuta stabile tramite un algoritmo di auto-calibrazione, una funzionalità che può avere il suo utilizzo anche in altre applicazioni legate all'interfacciamento dei sensori.

2.3 Ulteriori argomenti di ricerca

Machine learning è un termine che si riferisce a una disciplina scientifica che studia e crea algoritmi capaci di dedurre output appropriati a partire da un modello statistico generato da un insieme di dati di *training*[17].
L'*interactive machine learning*, in particolare, è una tecnica che consente alle persone di creare rapidamente un modello di *machine learning*, dove l'utente registra degli input di *training* con un output desiderato per ciascuno di essi, dopodichè, aggiungendo nuovi input, il sistema di *interactive machine learning* consente di migliorare e valutare il modello sperimentandolo in tempo reale o cambiando degli input di *training*, utilizzando degli algoritimi che richiedono poco tempo di calcolo e pochi esempi.
In questo contesto, un utilizzo di *Machine Learning* interattivo consente di creare una mappatura complessa tra i gesti dell'utilizzatore con i parametri dell'applicazione controllata semplicemente eseguendo i movimenti desiderati; grazie a un riscontro veloce, cio' rende piu' rapido e intuitivo il design dell'interfaccia [18].
Tra le applicazioni che utilizzano questa tecnologia per controllare un'applicazione audio *gesture-based* si possono citare Wekinator [17], sviluppato per un controllo musicale da Rebecca Fiebrink, poi estesa anche ad altri campi artistici e *ml.lib*, una library di tool specifica per le applicazioni di sintesi Max/Msp e *Pure Data* 3.2 [17], tuttavia, questa tecnica non è ancora largamente utilizzata nel campo, e necessita di un'opportuna formazione [6].

Capitolo 3

Strumenti *hardware* e *software*

Al fine delle funzionalita' di interfaccia che sono oggetto di studio in questo lavoro, sono stati utilizzati i seguenti dispositivi *hardware* e delle applicazioni installate su di un computer.

3.1 Componenti *Hardware*

3.1.1 Sensori

- **Apds-9960**

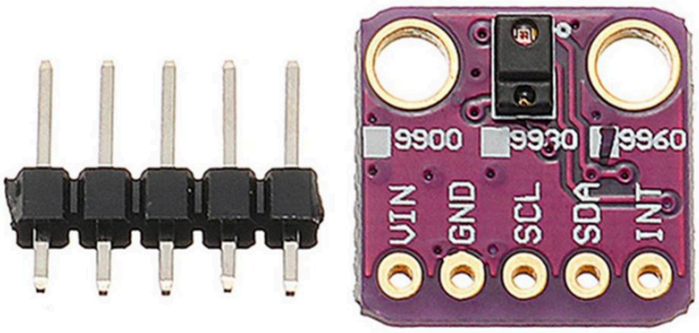


Figura 3.1: Sensore Apds-9960
[5]

Apds-9960 è un sensore ottico che incorpora un led a infrarossi con delle funzionalità di *gesture detection*, *proximity detection*, ed è capace di riconoscere colori e intensità della luce ambientale [19].
Il funzionamento del sensore si basa su quattro fotodiodi direzionali capaci di catturare la luce a infrarossi riflessa da oggetti nel raggio di azione, in modo da acquisirne i parametri del moto (velocità, direzione e distanza) e convertirli in informazione digitale; l'interfaccia di comunicazione seriale del dispositivo supporta un protocollo I2C.
Possiede due modalità di utilizzo:

[5] Per gentile concessione di `https://www.joom.com/it/products/61baf450a33dc4016fc91a73`

- *Gesture detection*
  Le feature incluse nella *gesture detection* sono l'attivazione automatica del sensore basata sulla prossimità dell'oggetto, il bilanciamento della luce ambientale e una comunicazione dei movimenti tramite una linea di *interrupt*, che possono essere relative a 4 direzioni di movimento o combinazioni più complesse di esse.
- *Proximity detection*
  La *proximity detection* permette di misurare la distanza tra l'oggetto ed il sensore, solitamente problematica per i dispositivi con una tecnologia *image based* [9] grazie all'intensità dell'energia riflessa catturata dal sensore. Questa funzionalità è ulteriormente migliorata dal sistema di riconoscimento di luce ambientale, inoltre permette di utilizzare la prossimità della mano come sorgente di *interrupt*.
  I fattori che influenzano i risultati della *proximity detection* sono tre: l'intensita' degli impulsi del led a infrarossi, la ricezione del sensore, e fattori ambientali, inclusa la riflettivita' della superficie.

- **Paj7620u2**

Figura 3.2: Sensore Paj7620U2
[6]

Il sensore Paj7620u2 presenta funzionalità simili al dispositivo precedente; tuttavia, rispetto ai 4 fotodiodi presenta una distribuzione di 60x60 fotodiodi e ha a diposizione più modalità di utilizzo:

- *Gesture Mode*
  In Gesture Mode può riconoscere 9 gesti della mano: *up, down, left, right, forward, backward, clockwise, anticlockwise, shake.*

[6] Per gentile concessione di `http://wiki.sunfounder.cc/index.php?title=Gesture_Recognition_Sensor_Module(GY-_PAJ7620U)`

- *Cursor Mode*
  La modalità *cursor* [20] è una funzionalità di *finger tracking* dove il centro dell'oggetto più vicino al sensore è riportato utilizzando le coordinate x,y; questa modalità può essere utilizzata per applicazioni che richiedono una ricostruzione più fedele del moto della mano.
- *Image Mode*
  La modalità *image* consente di ottenere l'immagine dal sensore con una risoluzione 30x30 inviando però i dati attraverso un *bus* SPI, un protocollo di comunicazione sincrono, e prevede un *rate* da un minimo di 22 MHz nel caso di invio in modalità *frame subtraction*, con una riduzione del rumore di fondo e un *rate* in media di 44 MHz inviando *raw data*;
  per entrambe le modalità la frequenza massima è 48 MHz [21].
- *Gaming Mode*
  La modalità *gaming* è un'opzione utilizzabile assieme alle altre modalità di funzionamento che consente di aumentare la responsività del sensore e aumentare la frequenza di aggiornamento dei gesti, rendendo il controllo più fluido.

Parametro	Minimo	Media	Massimo	Condizioni
Gesture Speed Response(°/s)	60		600	Normal Mode
	60		1200	Gaming Mode
Gesture Update Rate (Hz)		120		Normal Mode
		240		Gaming Mode

Tabella 3.1: Caratteristiche del funzionamento del sensore influenzate dalla *gaming mode*

Approfondimento sui valori ottenibili da Paj760u2

Il sensore Paj760U2 possiede molteplici registri in memoria che contengono il dato relativo a parametri di funzionamento del sensore.
Andando a leggere dal registro *AvgY* è possibile acquisire una stima della vicinanza dell'oggetto basata sull'intensità della luce IR riflessa, e su *State* è possibile leggere lo stato del moto della mano: se in avvicinamento o in allontanamento.
Questi registri fanno parte di un'ulteriore modalità di funzionamento, *Proximity*, che non è implementata nella libreria di Arduino del sensore, ma è menzionata nel *datasheet*; questo può essere dovuto al fatto che, testando il sensore in altre condizioni di utilizzo è possibile utilizzare delle funzioni che restituiscono informazioni sulla vicinanza di un oggetto: confrontando il funzionamento dalla *proximity mode* con la funzione *getObjectBrightness* già implementata nella libreria in modalità *cursor* e *gesture* non sono state notate differenze sostanziali, pertanto *getObjectBrightness*

può essere utilizzata al posto di questa ulteriore modalità.

5.9 Proximity Mode Controls

Bank	Address	Register Name	Default Value	R/W
0	0x69	R_Pox_UB[7:0]	0xC8	R/W
0	0x6A	R_Pox_LB[7:0]	0x40	R/W
0	0x6B	S_State	-	R
0	0x6C	S_AvgY[8:1]	-	R

Figura 3.3: Lista dei registri accessibili alla lettura in *ProximityMode*.
[7]

3.1.2 Schede elettroniche

- **Arduino Nano 33 BLE**

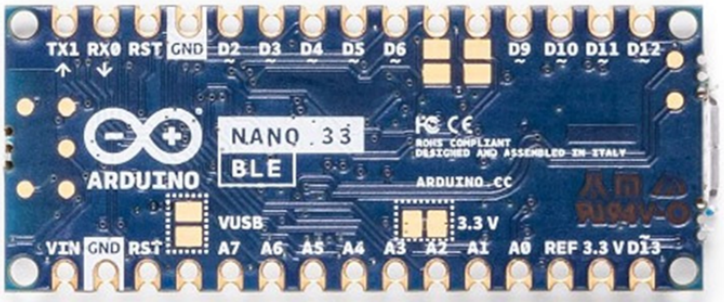


Figura 3.4: *Board* Arduino Nano 33
[8]

La scheda Arduino Nano 33 è una *board* programmabile compatibile con l'ambiente di sviluppo integrato Arduino IDE; è costruita sul microcontrollore nRF52840[22], sistema operativo ARM® Mbed™ OS e un processore Cortex-M4F a 64 MHz che la rende adatta ad applicazioni a basso consumo; può inoltre comunicare utilizzando il protocollo BLUETOOTH *Low Energy* 5.0.
Ha una Memoria *flash* da 1 MB, possiede un'unità di misurazione che comprende un accelerometro, giroscopio e magnetometro per captare orientamento, movimento e vibrazioni a cui possa essere soggetta la scheda.
Per comunicare utilizza anche un sistema di comunicazione seriale SPI (32 MHz), un connettore micro USB connesso a un convertitore Seriale-USB, 14 pin di input/output digitale e 13 pin Led.

- **Arduino Due**

[7] Per gentile concessione di `https://github.com/acrandal/RevEng_PAJ7620/wiki`
[8] Per gentile concessione di
`https://store.arduino.cc/collections/nano-family/products/arduino-nano-33-ble`

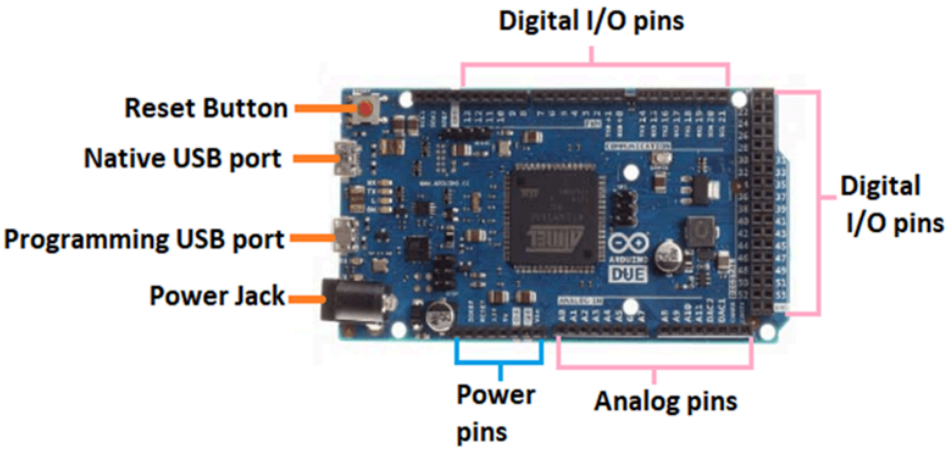


Figura 3.5: *Board Arduino Due*
[9]

La *board* Arduino Due possiede un processore Cortex-M3 e possiede funzionalità adatte ad applicazioni più performanti.
Ha una memoria *flash* da 512 KB, una frequenza di *clock* di 84 MHz [23]; la comunicazione avviene tramite due porte micro Usb, di cui una (*Native* USB) connessa direttamente al microcontrollore SAM3X MCU, mentre l'altra (*Programming* USB), con un *throughtput* minore, è connessa a un convertitore da protocollo USB a Seriale;
Possiede anche pin di input/output analogici, PWM e 54 *pin* di input/output digitale, con supporto alla comunicazione SPI.

3.1.3 *TFT Touch Shield*

Figura 3.6: *Display TFT Touch Shield*
[10]

Come periferica da utilizzare per avere un riscontro visivo del funzionamento corretto del sistema è stato scelto *TFT Touch Shield*, uno schermo *touch screen* resistivo [24].

[9] Per gentile concessione di `https://www.javatpoint.com/arduino-due`
[10] Per gentile concessione di `https://wiki.seeedstudio.com/2.8inch_TFT_Touch_Shield_v2.0/`

Compatibile con le schede Arduino, di 2,8 pollici in larghezza, 320*240 pixel di risoluzione e connettibile con la scheda attraverso un *bus* SPI.

3.2 Componenti *Software*

- **Arduino IDE**
  IDE sta per *Integrated Development Enviroment*, è un *software open source* usato per scrivere, compilare un codice e generare un file eseguibile da caricare nei moduli Arduino [25], ed è spesso consigliato ai programmatori neofiti.
  Tramite l'inclusione di librerie scritte in C/C++, create in modo specifico per le periferiche connesse alla scheda, è possibile semplificare la stesura di un codice con funzionalità I/O complesse utilizzando funzioni ad alto livello di astrazione; per esempio si può attivare un timer con un periodo prefissato con un solo comando.
  Utilizzando la finestra di monitor seriale all'interno dell'applicazione è possibile visualizzare i dati ricevuti e inviati dalla scheda con un *baud rate* variabile, utile per verificare il funzionamento corretto del programma, assieme al *debugger* incluso.

- **Max**
  Per sviluppare strumenti virtuali, il *software* più utilizzato è Max, un ambiente di programmazione visuale per sintesi di segnale audio, video, applicazioni MIDI e grafiche.
  La sezione Max del programma si occupa di operazioni discrete e di controllo di dati MIDI, la sezione Msp si occupa del DSP (*Digital Signal Processing*, ovvero l'elaborazione dei segnali) e della riproduzione dell'audio, e una sezione *Jitter* di *rendering* grafico e manipolazione video.
  Ha un funzionamento basato su blocchi di elaborazione di dati e di variabili interne, connessi tra loro tramite *patchcord*, che permettono l'invio e la ricezione di dati in input e output; posso avere due tipologie di segnale in uscita o in ingresso dai blocchi: i segnali audio o di controllo; i primi possono comunicare con l'interfaccia audio tramite convertitori DAC o ADC, mentre i secondi servono a modificare i parametri interni dei blocchi di elaborazione.

Capitolo 4

Sistemi implementati

Sulla base delle conoscenze ottenute e degli strumenti a disposizione sono state implementate due applicazioni.

4.1 Schema di base del sistema di controllo

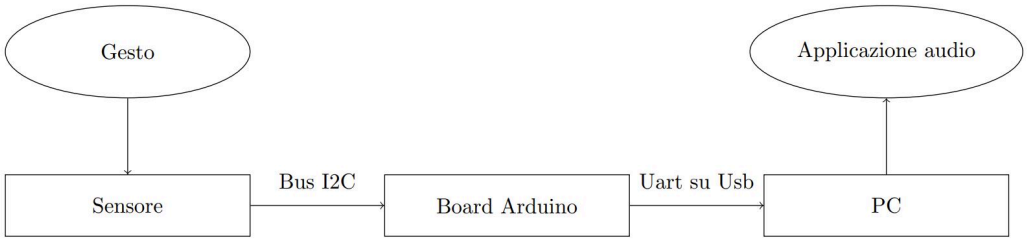


Figura 4.1: Schema delle connessioni tra i dispositivi che compongono il sistema

Il sistema di controllo si basa sulla lettura di un movimento della mano da parte del sensore di *Gesture recognition* per poter controllare un'applicazione di *processing* audio.
Il sensore è connesso a una scheda Arduino che si occupa di leggere i dati ottenuti e inviarli opportunamente al PC dove è installata l'applicazione da controllare.

4.1.1 Comunicazione tra Sensore e *Board*

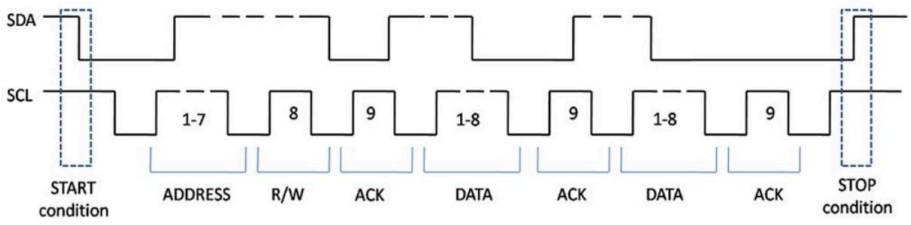


Figura 4.2: Visualizzazione di una comunicazione tra *master* e *slave* in I2C

L'interfacciamento tra il sensore e la board avviene attraverso un *bus* I2C, un protocollo di comunicazione seriale richiedente quattro linee: *serial data* (SDA),

serial clock (SCK), *ground* (GND) e alimentazione (VDD) [26].
I2C è comunemente usato tra circuiti integrati e prevede che uno dei dispositivi ai capi della comunicazione agisca da *master*, che, come già citato precedentemente 1.2.1, per poter sincronizzare la comunicazione tra i dispositivi fornisce un segnale di temporizzazione ai dispositivi *slave*, il cui *rate* può raggiungere 400khz se impostato in *Fast Mode*.
I messaggi della comunicazione vengono delimitati da specifici *flag bit*, uno all'inizio (*start condition*) e uno alla fine (*stop condition*), che corrispondo rispettivamente a un livello forzato basso o alto sulla linea SDA; mentre il livello del segnale di *clock* viene mantenuto alto, caratteristica non condivisa dai *bit* informativi, che prevedono la linea SCK bassa nelle variazioni di SDA, inoltre, i dispositivi *slave* vengono riconosciuti tramite un sistema di indirizzamento a 7 bit che vengono inviati nel primo *byte* della trasmissione a cui seguirà una conferma, il segnale di *acknowledgement* ACK.
In questo caso il dispositivo *master* sarà la scheda elettronica, mentre lo *slave* sarà il sensore.

4.1.2 Comunicazione tra *Board* e PC

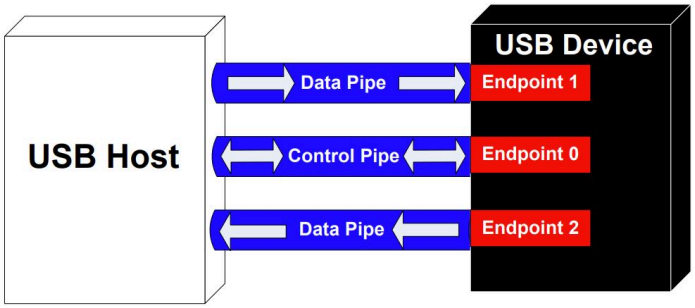


Figura 4.3: Modello dei canali di comunicazione USB.

La comunicazione tra la scheda Arduino e il PC avviene tramite un'interfaccia USB, facile alla connessione e all'uso; USB è uno standard dell'industria utilizzato per connettere periferiche a un computer, ideato per essere condiviso tra i vari dispositivi. Un sistema di comunicazione USB consiste in un dispositivo *host* (il PC) e più periferiche connesse in una topologia a stella [27]; lo scopo dell'*host* è quello di utilizzare il *software* implementato per adempire a vari compiti:

- Riconoscere e gestire la connessione e la disconnessione dei dispositivi.
- Gestire il flusso di data tra l'*host* e i dispositivi connessi.
- Provvedere all'alimentazione dei dispositivi.
- Monitorare l'attività sul bus.

A ogni *endpoint* connesso verrà dato un indirizzo di 7 bit dall'*host*, utilizzato nella comunicazione, che viene effettuata tramite *Pipes* (condotti), dei percorsi di connessione tra l'*host* e un *endpoint* indirizzabile. Posso utilizzare pipes bidirezionali di controllo per inviare comandi e configurazioni o *pipes* unidirezionali di data.
La velocità di comunicazione su USB varia dal tipo di dispositivo connesso, è può essere classificata con *Low speed*, *full speed*, *hi speed* e *Super speed*; la scheda Arduino Due è capace di utilizzare una connessione USB *High speed*, fino a 480 Mbit/s, anche se vi è poca documentazione riguardo alla porta *Native* USB, mentre la scheda Arduino Nano 33 utilizza una connessione *full speed*, a 12 Mbit/s [22].

4.2 Prima applicazione

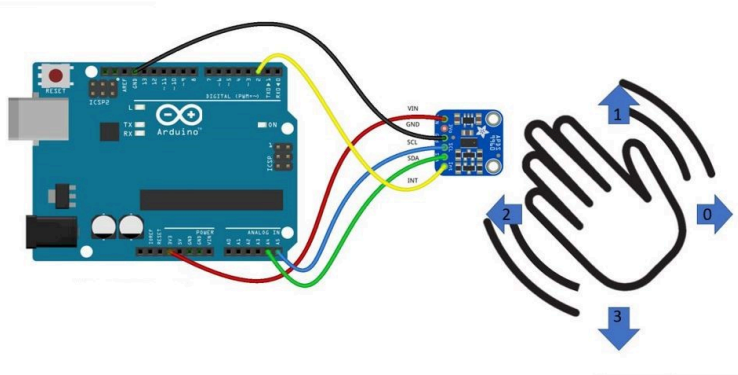


Figura 4.4: Funzionamento della prima applicazione

Per questa prima applicazione sono stati utilizzati il sensore Apds-9960 e la scheda Arduino Nano 33.
Sul *firmware* della scheda è stato scritto un semplice programma per testare il funzionamento di questa disposizione, il cui scopo è quello di poter selezionare un valore tramite la direzione del gesto della mano e inviarlo al PC; il dato viene poi letto all'interno dell'applicazione *Pure Data*, una versione *open source* di Max/Msp, tramite il blocco *comport*, nel quale viene specificato l'identificativo della connessione e il *rate* di lettura.
L'applicazione audio controllata consiste in un sintetizzatore polifonico, generato su *Pure Data* e connesso a un riverbero; dal blocco *comport* viene letto il simbolo che seleziona uno di quattro diversi *preset* del riverbero; è così possibile selezionare in tempo reale la quantità di effetto applicata al suono con il gesto della mano.

4.3 Seconda applicazione

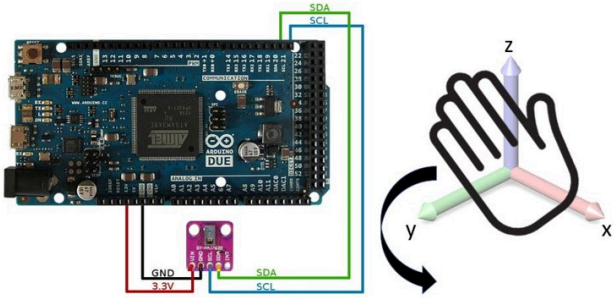


Figura 4.5: Funzionamento della seconda applicazione

In questo secondo sistema di controllo sono stati utilizzati il sensore Paj7620U2 e la board Arduino Due.
Lo scopo di questa applicazione è quello di acquisire le coordinate spaziali della mano e utilizzarle per controllare una funzionalità di Max adibita alla conversione di un *file* audio mp3 in formato *ambisonics*; si ha quindi un ascolto immersivo interattivo che varia la percezione della direzione della sorgente sonora in base allo spostamento della mano.
Su `https://github.com/EDAmen/Max-gesture-control` è possibile accedere al codice della scheda Arduino e al *file* di Max, in aggiunta ad approfondimenti sull'applicazione.

4.3.1 *Firmware* su Arduino

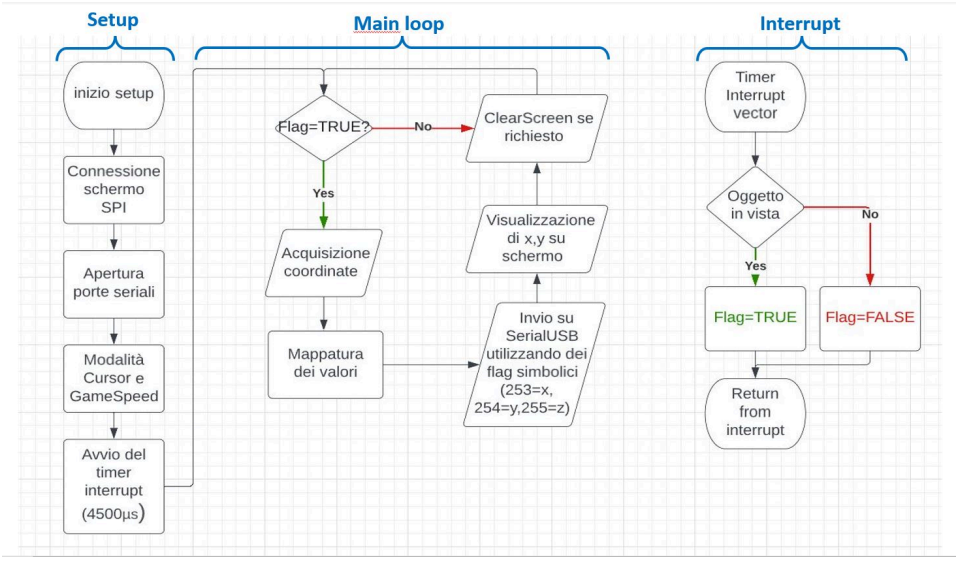


Figura 4.6: Schema a blocchi del funzionamento della scheda

Il programma implementato su Arduino è suddivisibile in tre sezioni:

- **Setup**
  La prima parte di codice prevede l'inizializzazione delle librerie richieste dal programma:
  RevEngPAJ7620.h permette di utilizzare delle funzioni per accedere ai registri del sensore e cambiare le modalità di funzionamento; *XPT2046.h* e *LCD.h* consentono di utilizzare un *driver* per il *touchscreen* e delle funzioni basilari per utilizzarlo; ho anche delle librerie per utilizzare la comunicazione SPI e i *timer*.
  Nella sezione di *Setup* viene inizializzata la connessione SPI e viene acceso lo schermo, viene aperta poi la porta *serialUSB* a un *baud rate* di 115200, che corrisponde alla porta USB nativa.
  La logica attende dunque la conferma dell'avvio del sensore e viene settata la modalità *cursor* in *gaming speed*, che mi fornirà le informazioni spaziali del centro della mano, e viene infine avviato un *timer* che ogni 4,5 ms genera un *interrupt*.

- **Interrupt**
  Nella gestione dell'*interrupt* viene controllata la presenza della mano nelle vicinanze del sensore e in base a ciò viene sovrascritto un *bit* di *flag*.
  La scelta di utilizzare un *interrupt* ogni 4,5ms per questo tipo di operazione è dovuta al funzionamento del sensore: se Arduino accede a questa funzione prima che il sensore venga aggiornato, quindi con una frequenza superiore a 240 Hz, le latenze del sensore peggiorano drasticamente; il periodo di 4,5 ms è stato scelto empiricamente in quanto il valore che forniva un risultato migliore.
  É preferibile leggere questo dato con degli *interrupt* scanditi nel tempo al posto di interrogare il sensore in *polling* poichè potrei avere un valore variabile di latenza.

- **Main Loop**
  Nel main loop viene testato il valore logico del *flag*, e se è *true*, ovvero se ho un oggetto in vista azzero il *flag* e utilizzo le funzioni *getCursor* e *getObjectBrightness* per acquisire le coordinate spaziali; utilizzo le funzioni *constrain* e *map* per mappare i valori acquisiti come interi da un *byte* tra 0 e 252 e li invio sulla porta seriale intervallati dai valori 253, 254 e 255 che verranno interpretati su Max come x,y e z.
  I valori x e y vengono visualizzati sullo schermo e infine, al di fuori della sezione di codice eseguita se ho un oggetto in vista, viene ripulita l'immagine a schermo in caso di pressione sul *touch screen*.

4.3.2 Applicazione su Max/Msp

Su Max è stata realizzata una disposizione di oggetti, che, tramite la funzionalità *Serial*, legge i valori inviati da Arduino e li aggiorna ogni 2 ms, il periodo minimo supportato dall'applicazione per il segnale di controllo.

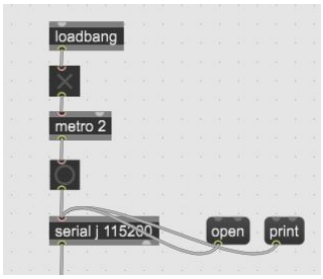


Figura 4.7: Sezione del programma adibita alla lettura su Seriale

I dati verranno passati a una sezione del programma, *Map* che si occupa di catalogare i dati ottenuti grazie ai *flag* simbolici precedentemente descritti e normalizzarli in modo da poter essere letti correttamente dall'oggetto *Ambipoint*, che si occupa della conversione delle coordinate in informazioni utilizzabili per la conversione in *Ambisonics*.

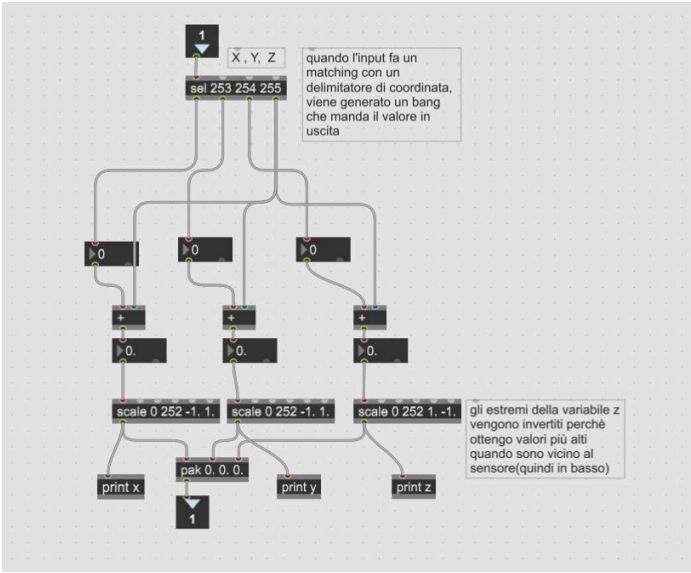


Figura 4.8: Sezione del programma che discrimina e normalizza i dati ottenuti

Come applicazione di conversione è stata scelta *Ambiencode*, un *plugin*, ovver un programma utilizzabile all'interno di un software *Host*, in questo caso Max; é presente anche l'opzione di convertire l'uscita audio in un formato stereo binaurale 1.2.3 per l'ascolto in cuffia tramite il *plugin Ambix Binaural*.

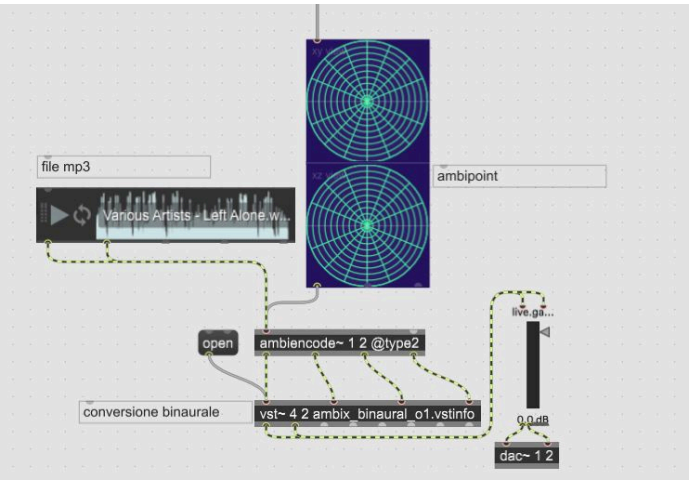


Figura 4.9: Conversione del file mp3 in formato *Ambisonics*/Binaurale

4.4 Valutazione della seconda applicazione

Sono state valutate le prestazioni del secondo sistema per annotarne le caratteristiche e le prestazioni.

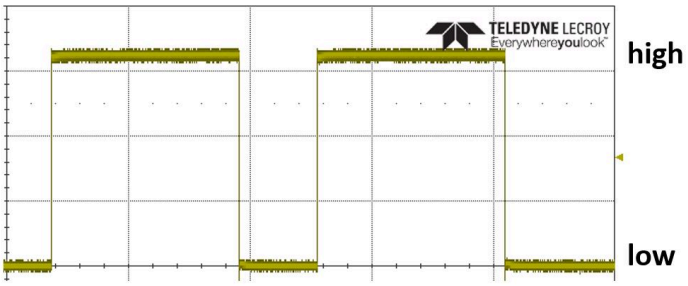
4.4.1 Latenza

La valutazione della latenza è una questione fondamentale che interessa i sistemi digitali per valutare l'adeguatezza alle condizioni imposte dall'applicazione; in questo paragrafo viene studiato il ritardo tra l'interazione dell'utente e la reazione corrispondente, un evento sonoro.
Il sistema per ottenere un riferimento temporale degli eventi, in modo da calcolare le durate delle varie sezioni dell'applicazione consiste nell'utilizzo di un pin digitale in modalità output della scheda Arduino in connessione con l'oscilloscopio digitale *Teledyne Lecroy*.

Figura 4.10: Schema della valutazione dei ritardi

Le sezioni che vengono calcolate nei paragrafi seguenti riguardano i tempi di lettura, esecuzione e invio dei dati e la latenza audio dell'applicazione.

- Tempi di esecuzione del ciclo del programma e invio dei dati




```
digitalWrite(35, HIGH); // punto 1
ifsensor=0;
int cursor_x = sensor.getCursorX();
int cursor_y = sensor.getCursorY();
int cursor_z = sensor.getObjectBrightness();

digitalWrite(35, LOW); // punto 2
```

Figura 4.11: Sistema utilizzato per effettuare le misure

Per ottenere le durate delle varie sezioni del programma è stato inserita una riga di codice che porta il *pin* digitale esterno a un livello logico alto prima della sezione da analizzare e lo riabbassa una volta terminata, in modo da poter visualizzare la variazione di tensione nell'oscilloscopio.
La media del tempo richiesto al *firmware* di Arduino di eseguire un ciclo completo del programma si attesta intorno a 870,7µs, tuttavia, nonostante la deviazione standard sia contenuta, il *jitter* può, in casi rari far arrivare il ritardo a 6,2 ms; ciò è imputabile al ritardo generato dall'invio dei dati su porta seriale.
è bene anche menzionare il ritardo aggiuntivo che si otterrebbe implementando un cambio di modalità all'interno del programma, stabile intorno a 3,09 ms.

Misura	Mean	Min	Max	St.dev
Tempo totale	870,7µs	796,3µs	6,2ms	159,7µs
Acquisizione coordinate	541,4µs	541,3µs	541,5µs	48ns
Elaborazione coordinate	4,09µs	4,08µs	4,1µs	7ns
Disegno su display	114,8µs	103,3µs	116,2µs	1,5µs
Scrittura su SerialUSB	248,1µs	121,1µs	4ms	191,6µs
Passaggio di modalità(non usato)	3,1ms	3,1ms	3,1ms	542ns

Tabella 4.1: Tempi di esecuzione del *firmware* su Arduino.

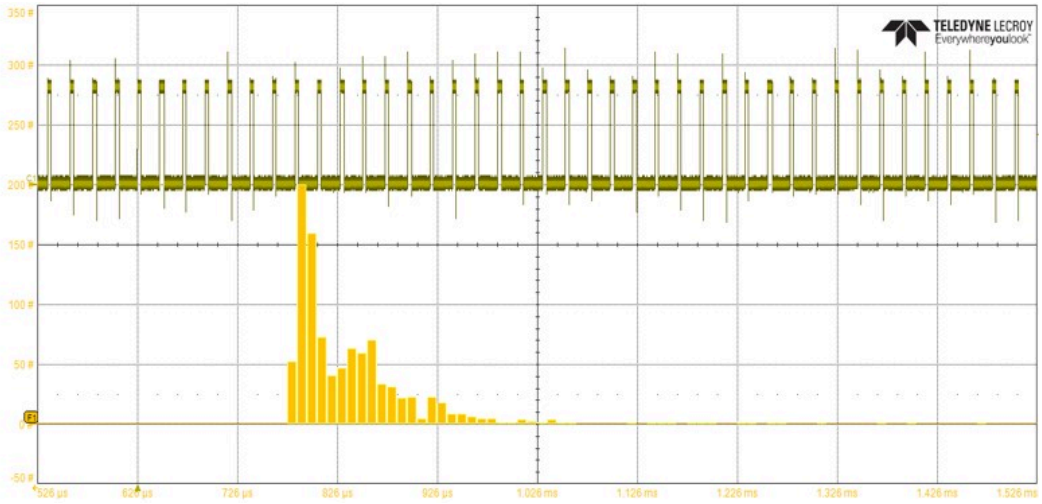


Figura 4.12: Istogramma delle probabilità della durata totale di un ciclo del programma caricato su Arduino.

Il grafico è stato calcolato a partire da 1000 campioni; gli impulsi visibili nella sezione superiore dell'immagine corrispondono al voltaggio del *pin* misurato

- Tempo di trasferimento tra Arduino e il PC
  Questa latenza è stata calcolata utilizzando il *software* PuTTY, un *software open source* con funzionalità di emulatore di terminale.
  Tramite l'utilizzo di uno *sketch* Arduino sono stati inviati 6 *byte* su *SerialUSB* terminati da un valore prefissato, al quale PuTTY ha risposto inviando un messaggio di conferma.
  Osservando il valore del pin esterno come nel test precedente, è stato possibile calcolare il *Round Trip Time* tra Arduino e l'applicazione, comprendente la connessione USB e il bus interno del PC; si può stimare la durata dell'invio dei dati dimezzando questi valori.

  Come si può osservare anche il trasferimento dalla board all'applicazione ha una misura variabile, con una deviazione standard superiore alla misura della scrittura su seriale, e picchi vicini ai 2 ms.

Misura	Mean	Min	Max	St.dev
Tempo totale	782,5µs	194,6µs	3,83ms	450,5µs

Tabella 4.2: Tempo di trasferimento tra Arduino e il PC

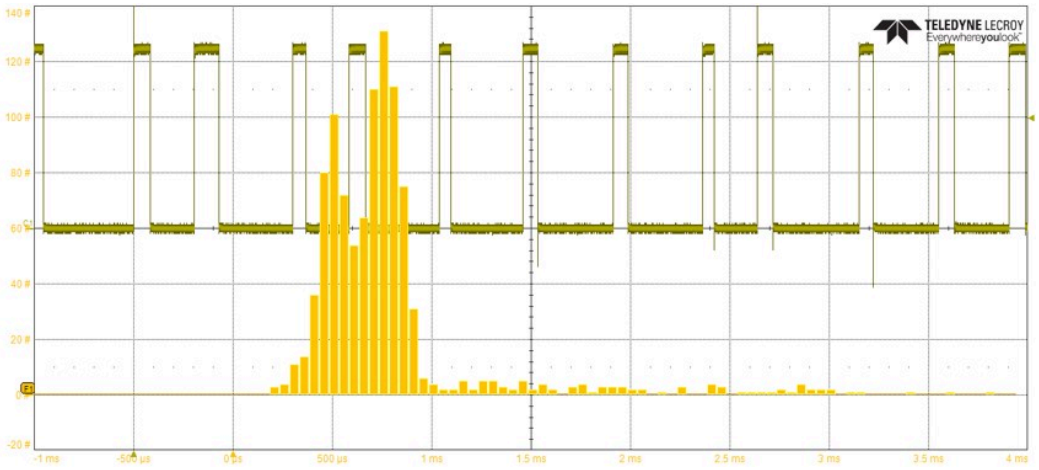


Figura 4.13: *Round Trip Time* tra Arduino e il PC calcolato con PuTTy

- Latenza dell'audio su Max/Msp
  Per ultima è stata calcolata la latenza che intercorre tra l'invio dei dati da Arduino e l'uscita audio; per fare ciò sono stati inviati da Arduino dei messaggi volti a variare l'intensità di un segnale audio su Max, ed è stato connesso l'oscilloscopio con l'uscita *jack* dell'interfaccia audio.
  Per comunicare efficacemente con un dispositivo audio, come una scheda audio o un altoparlante, il sistema operativo del computer deve utilizzare una tipologia di applicazioni chiamate *driver* audio [28]; le latenze del suono sono state testate utilizzando due *driver* audio connessi a due uscite, il primo è *Asio4all*, che connette Max all'uscita cuffie del pc, mentre il secondo è *Focusrite USB Asio*, che connette l'applicazione alla scheda audio esterna *Scarlett Solo 3rd gen.*
  Entrambi i driver sono stati impostati con un *I/O vector size* di 64, ovvero il numero di campioni del segnale audio inviati all'interfaccia [29], ridurre la dimensione del *I/O vector size* ha effetti sulla latenza, ma non sulla qualità dell'audio [29], a spese, però della quantità di calcoli richiesta alla *cpu*.
  Un'altra opzione rilevante ai fini della latenza è la funzionalità di Max/Msp *Scheduler Overdrive*, che consente di gestire gli eventi all'interno del programma a una priorità maggiore.

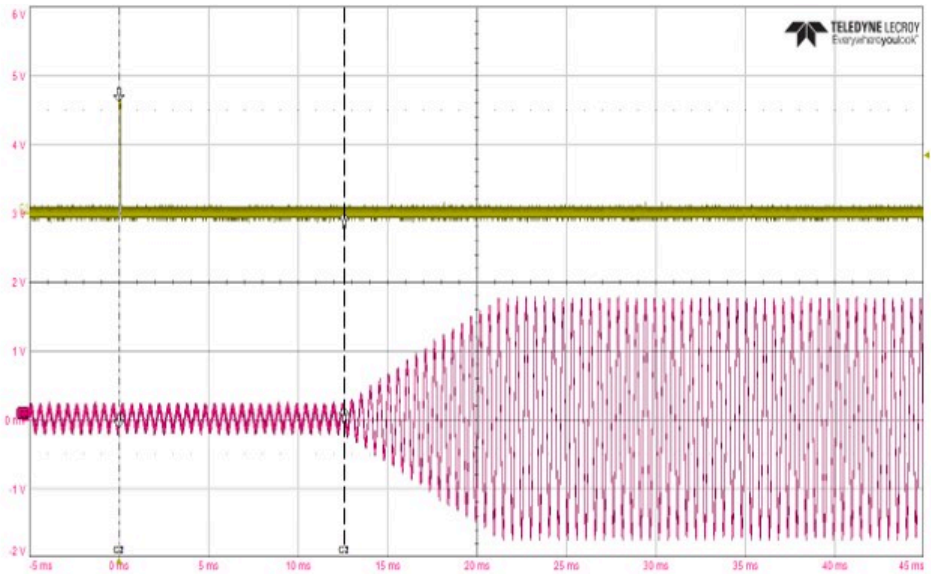


Figura 4.14: Latenza audio utilizzando *Asio4all*

12,6ms

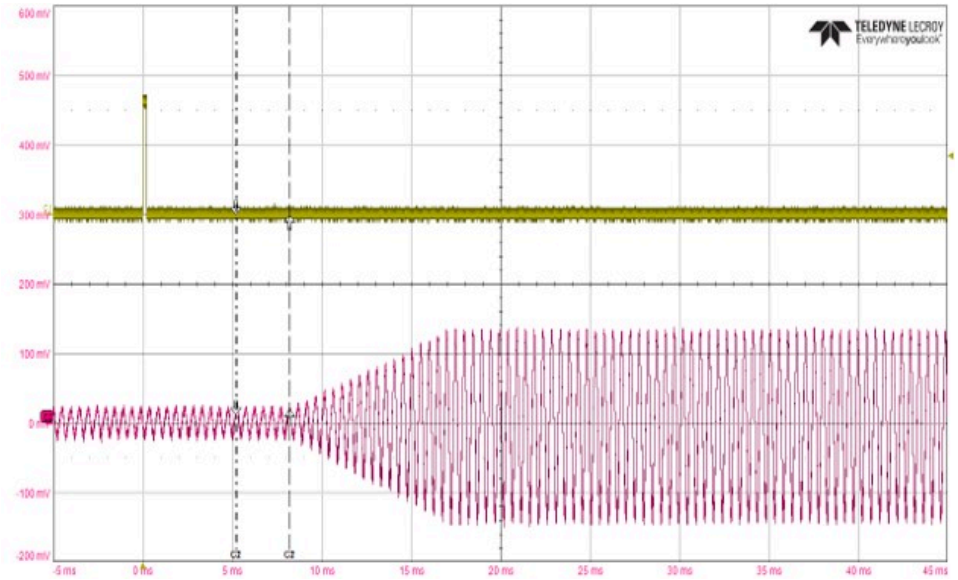


Figura 4.15: Latenza audio utilizzando *FocusriteUSB* e una scheda audio esterna

8,2ms

4.4.2 Accuratezza del *tracking* e condizioni di utilizzo

Da un punto di vista qualitativo, la precisione del sensore di riconoscere la mano varia con la distanza dalla centro del *range*; in particolare si può notare una curvatura del cursore sullo schermo connesso alla board nelle zone periferiche; ciò è dovuto al fatto che i dati ottenuti in *cursor mode* restituiscono il centro geometrico

dell'immagine, quindi quando la visibilità della mano non è ottimale, si ottiene un risultato inesatto, di fatto escludendo i bordi del *Gesture detecting range* indicato nel *datasheet* del sensore.

4.4 Gesture Functional Specifications

Parameters	Symbol	Min.	Typ.	Max.	Unit	Condition
Gesture Detecting Range	d_{OP}	5	-	15	cm	Calculated from PAJ7620U2 sensor center
Gesture Detecting View Angle	θ_{OP}	-	60	-	degree	Calculated diagonally

Figura 4.16: Range di funzionamento indicato nel *datasheet* del sensore [1]

Utilizzando dei parametri di mappatura su Max che escludano le zone problematiche, ad esempio utilizzando il blocco *scale* con i soli valori centrali tra quelli acquisiti dal sensore, è possibile attenuare il problema.
Anche l'uso di un oggetto appuntito e riflettente come cursore contribuisce a migliorare la qualità del *tracking*, come spiegato in 3.1.1, ad esempio un pennarello bianco, in modo che venga ridotto il contributo di imprecisione dato dalla visualizzazione della mano.

[11] Per gentile concessione di `https://github.com/acrandal/RevEng_PAJ7620/wiki`

Capitolo 5

Conclusioni

Nei primi due capitoli, partendo dallo studio dell'interazione tra uomo e macchina e descrivendo alcuni dispositivi già esistenti, sono stati delineati i possibili utilizzi della *gesture recognition* applicata al controllo audio e cosa ci si può aspettare dal futuro di questa tecnologia; da queste informazioni si è passati all'implementazione di un sistema di controllo, utilizzando le schede Arduino Nano 33 e Arduino Due, assieme ai sensori Paj7620u2 e Apds-9960, per sviluppare un'applicazione con lo scopo di acquisire le caratteristiche dei gesti della mano, in modo da controllare il programma di elaborazione audio Max; infine, utilizzando un insieme di elementi per riuscire a ottenere una stima delle prestazioni, si è giunti a dei valori indicativi dell'accuratezza e della latenza del progetto.
In *Problems and Prospects for Intimate Musical Control of Computers* [30] viene fissato uno standard di qualità sulla latenza di risposta di uno strumento musicale digitale di minimo 10 ms, dato dalla percezione psicoacustica di un evento sonoro, anche se questo valore può variare molto in base al contesto; infatti, per quanto riguarda la percezione di eventi percussivi, i limiti sono anche più stringenti, mentre un tipo di controllo continuo richiede minore sensibilità, arrivando a essere accettabile tra i 20-30 ms [31].
Anche il *jitter* del ritardo, ovvero la variazione temporale rispetto al valore medio, è un fattore da tenere in considerazione: mentre l'utilizzatore può adattare i propri movimenti a una latenza fissa, non può invece farlo con una variazione non predicibile. Un asincronismo di 6 ms è percettibile all'orecchio, ma per i casi più stringenti è consigliato un *jitter* limitato a 1ms [31]; il controllo in questa tesi è, di conseguenza, non performante come applicazione *real time*, a causa dei contributi occasionali di *jitter* dati dalla scrittura dei dati, anche se l'utilizzo di una scheda audio esterna consente di rimanere in media sotto i 10 ms di latenza.
È bene inoltre specificare che per calcolare il ritardo dell'audio non è stata aggiunta la latenza di conversione in Ambisonics, e che riguarda quindi la generazione dell'audio con i blocchi già esistenti in Max.
I limiti di questo sistema sono da ricercarsi anche nelle durate della comunicazione su USB tra la scheda e il PC e nelle prestazioni dei sensori utilizzati, che, pur essendo funzionanti, non si sono dimostrati particolarmente adatti per questo tipo di interazione, come invece sarebbe stato un sensore con un'area di rilevazione più estesa

o con un dispositivo *device based*; questo lavoro si pone come studio di parametri e metriche di interesse per questo tipo di applicazione e pone le basi sperimentali per iterazioni future.

Bibliografia

[1] Siddharth S. Rautaray and Anupam Agrawal. Design of gesture recognition system for dynamic user interface. In *2012 IEEE International Conference on Technology Enhanced Education (ICTEE)*, pages 1–6, 2012.

[2] Yuxiang Li and Jingya Xu. Gesture recognition related technology and development challenges. In *2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI)*, pages 568–571, 2022.

[3] J.A. Paradiso. Electronic music: new ways to play. *IEEE Spectrum*, 34(12):18–30, 1997.

[4] David Wessel and Matthew Wright. Problems and prospects for intimate musical control of computers. *Computer Music Journal*, 26(3):11–22, 2002.

[5] Douglas Hill. *Extended Techniques for the Horn: A Practical Handbook for Students*. Alfred Music Publishing, 1996.

[6] Diemo Schwarz, Wanyu Liu, and Frédéric Bevilacqua. A Survey on the Use of 2D Touch Interfaces for Musical Expression. In *New Interfaces for Musical Expression (NIME)*, Birmingham, United Kingdom, July 2020. Final version as available for download on the NIME conference site, adding 2 more responses from the audience.

[7] Lamtharn Hantrakul and Konrad Kaczmarek. Implementations of the leap motion device in sound synthesis and interactive live performance. *ACM International Conference Proceeding Series*, 06 2014.

[8] Xuejing Sun. Immersive audio, capture, transport, and rendering: a review. *APSIPA Transactions on Signal and Information Processing*, 10:e13, 2021.

[9] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3):311–324, 2007.

[10] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677–695, 1997.

[11] Bhiksha Raj, Kaustubh Kalgaonkar, Chris Harrison, and Paul Dietz. Ultrasonic doppler sensing in hci. *IEEE Pervasive Computing*, 11(2):24–29, 2012.

[12] www.korg.com/it/KP3+.

[13] www.sourceaudio.net/hot-hand-usb.html.

[14] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, 2012.

[15] https://github.com/jpbellona/simpleKinect.

[16] https://www.thebusinessresearchcompany.com/report/audio-equipment-global-market-report.

[17] Jamie Bullock and Ali Momeni. Ml.lib: Robust, cross-platform, open-source machine learning for max and pure data. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, NIME 2015, page 265–270, Baton Rouge, Louisiana, USA, 2015. The School of Music and the Center for Computation and Technology (CCT), Louisiana State University.

[18] Clarice Hilton, Nicola Plant, Carlos González Díaz, Phoenix Perry, Ruth Gibson, Bruno Martelli, Michael Zbyszynski, Rebecca Fiebrink, and Marco Gillies. Interactml: Making machine learning accessible for creative practitioners working with movement interaction in immersive media. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Apds-9960 digital proximity, ambient light, rgb and gesture sensor data sheet, 2015.

[20] https://github.com/acrandal/RevEng_PAJ7620.

[21] https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html.

[22] https://docs.arduino.cc/static/1f8788ca47f23eaf21c4fcf681ae5ce1/ABX00030-datasheet.pdf.

[23] https://store.arduino.cc/products/arduino-due-without-headers.

[24] https://www.waveshare.com/wiki/2.8inch_TFT_Touch_Shield#Features.

[25] https://www.theengineeringprojects.com/2018/10/introduction-to-arduino-ide.html.

[26] Prasanna Bagdalkar and Layak Ali. Interfacing of light sensor with fpga using i2c bus. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 843–846, 2020.

[27] Robert Murphy. Usb 101: An introduction to universal serial bus 2.0. https://www.infineon.com/dgdl/Infineon-AN57294_USB_101_An_Introduction_to_Universal_Serial_Bus_2.0-ApplicationNotes-v09_00-EN.pdf?fileId=8ac78c8c7cdc391c017d072d8e8e5256, 2017.

[28] https://www.dell.com/support/home/it-it/drivers/driversdetails?driverid=15j6v#:~:text=Il%20driver%20audio%20%C3%A8%20un,esempio%20schede%20audio%20e%20altoparlanti.

[29] https://docs.cycling74.com/max8/tutorials/04_mspaudioio.

[30] David Wessel and Matthew Wright. Problems and prospects for intimate musical control of computers. *Computer Music Journal*, 26(3):11–22, 2002.

[31] Andrew McPherson, Robert Jack, and Giulio Moro. Action-sound latency: Are our tools fast enough? In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 20–25, Brisbane, Australia, July 2020. NIME.