Università degli studi di Padova
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Selfear 2.0: un'applicazione mobile con head pose estimation per l'acquisizione di profili acustici individuali

Relatore Ch.mo Prof. Federico Avanzini
Correlatore Dott. Michele Geronazzo

Candidato Filippo Beraldo
Matricola 1111116

Anno Accademico 2015 - 2016

Prefazione

Lo studio della spazializzazione del suono fa parte della frontiera tecnologica di maggior interesse dal punto di vista commerciale e di ricerca nell'ambito del sound computing. Gli sviluppi tecnici per il rendering visivo tridimensionale hanno fatto passi da gigante nel recente passato, tuttavia è evidente la disparità di attenzione verso quest'ultimo rispetto allo studio del rendering tridimensionale del suono. Il futuro vedrà un'ampia pervasività dei sistemi di realtà virtuale e aumentata e per questo gli sviluppi in ambito di *rendering* sonoro si stanno facendo sempre più necessari.

I primi prodotti di tipo commerciale sono già disponibili al mercato di massa e si focalizzano fondamentalmente sul rendering visivo, tuttavia per costruire un sistema di alta resa immersiva e con grande sensazione di *presence* è necessario prestare molta attenzione anche al contributo audio.
Le peculiarità del sistema uditivo, e le particolari relazioni che intercorrono tra i vari componenti dello stesso variano da soggetto a soggetto e rendono marcatamente personalizzata l'esperienza uditiva.

L'abilità umana di percepire ed individuare le sorgenti sonore è dovuta alla capacità di elaborazione di indicatori monoaurali, fenomeni che interessano singolarmente l'orecchio, e di indicatori binaurali ovvero stimoli che sollecitano i due organi uditivi. Lo studio del comportamento del sistema uditivo umano in relazione a stimoli sonori provenienti dall'ambiente circostante e la stima di parametri soggettivi che permettono la percezione spaziale del suono sono attualmente eseguiti con procedure particolarmente costose in termini di tempo e risorse, come ad esempio le misurazioni individuali in camera anecoica.

Il paradigma fondamentale del *HRTF-based rendering* e quello del nostro contesto applicativo è quello di utilizzare dati esistenti per ricavare una rappresentazione quanto più fedele possibile delle *Head-Related Transfer Funcion (HRTF)* dell'ascoltatore perciò non si tratterà di estrarre *HRTF individuali*, compito molto complesso e costoso, bensì di *individualizzare HRTF* rispetto a parametri soggettivi chiave per dell'ascoltatore.

Questo lavoro di tesi ha come obbiettivo primario quello di sviluppare strumenti non invasivi, portabili, economici e di facile utilizzo per la stima del contributo acustico dell'individuo.

In questo lavoro si propone una realizzazione di un'applicazione destinata agli smartphone Android, chiamata Selfear 2.0, con l'obbiettivo di sviluppare un'interfaccia che permetta ad un utilizzatore di realizzare questo tipo di misure autonomamente.
La soluzione proposta integra algoritmi real-time di computer vision per la head pose estimation realizzando l'*online monocular model-based tracking di oggetti 3D rigidi* per ricavare una stima della posa della testa dell'utente a varie angolazioni, sfruttando le informazioni della camera frontale del dispositivo.

Preface

Studying sound spatialization is at the forefront of technological research in sound computing, both for consumer products and academic purposes. Technical developments about visual rendering have made huge progress in the recent past, on the other hand we cannot state that for tridimensional sound rendering. In the future we'll see a pervasive presence of virtual and augmented reality systems in everyday life and for this reason the need for great developments in sound rendering surges.

The first commercial products are already available and they mainly focus on visual redering, and in order to build a highly immersive system with accentuated presence it's critical to take care of the audio component as well.
Peculiarities of the human hearing system and the particular relations between them vary between subjects and make hearing a highly personalized experience.

Human ability to perceive and localize sound sources is mainly due to the human skill of elaborating monoaural cues, phenomenons that occur at the individual ear, and of evaluating binaural cues which involves perception at both ears. Stduying the human hearing behavior when stimulated by sounds occurring in the surrounding environment and the study of tridimensional sound perception are currently carried on with expensive procedures in terms of time and resources, for instance measurements in an anechoic chamber with special hardware.

HRTF-based rendering paradigm, which becomes also ours, is to use existing data in order to attain an estimate of the user's *Head-Related Transfer Functions (HRTFs)* with as much accuracy as possible. Therefore the task is not about obtaining *individual HRTFs*, which is a complex and expensive job, instead it's about *individualizing HRTFs* with respect to the subjective user's key parameters.

This thesis aims at developing portable, economic, easy-to-use and non-invasive tools to estimate the contributions of each individual's hearing system.
This work proposes an application targeting Android smartphones, called Selfear 2.0, which produces an interface to help the user in acquiring such measurements autonomously.
The porposed solution integrates real-time computer vision algorithms with head pose

estimation implementing *online monocular model-based tracking di oggetti 3D rigidi* in order to obtain an estimate of the head orinetation of the user at various angles, exploiting information coming from the front camera of the device.

Sommario

Eseguire il rendering spaziale del suono necessita di una accurata ed individuale parametrizzazione del contributo acustico dell'ascoltatore ed ha innumerevoli applicazioni in ambito di realtà aumentata e realtà virtuale, assistenza alla guida di macchinari e di riabilitazione, per citarne alcuni.
La capacità da parte delle macchine di stabilire il focus di attenzione degli umani è importante per costruire nuovi metodi di interazione uomo-macchina. La head pose estimation è un problema di computer vision che permette ad una macchina (ad es. un robot) di stabilire la posizione della testa di un ipotetico utente che si sta interfacciando con essa.
Il presente lavoro cerca di integrare la head pose estimation in un'applicazione Android al fine di creare un'interfaccia interattiva, *self-adjusting* e di facile utilizzo per affiancare gli utilizzatori nella rilevazione del proprio profilo acustico. Le problematiche affrontate nell'adottare questo approccio sono molteplici: è necessario sviluppare e/o integrare algoritmi *real-time*, robusti ed accurati nelle rilevazioni che siano al tempo stesso in grado di essere eseguiti in ambienti con forti vincoli di tipo hardware, inoltre è necessario produrre un'interfaccia di facile utilizzo cosicché anche utenti non particolarmente addestrati siano in grado di portare a termine la procedura di rilevazione.
Nel Cap. 1 vengono descritte le caratteristiche dell'udito umano, la percezione sonora spaziale, quali sono le peculiarità che è necessario modellare per creare display uditivi virtuali 3D e come questi ultimi sono realizzati.
Il Cap. 2 illustra l'idea alla base del progetto Selfear ed in particolare si descrivono le caratteristiche della prima versione dell'applicazione e i risultati osservati dalle misurazioni acustiche eseguite con essa.
Il Cap. 3 descrive il cuore di questo lavoro di tesi, gli aspetti tecnici coinvolti, una panoramica sugli algoritmi necessari per eseguire la head pose estimation, una breve descrizione di quali di essi sono stati scelti e di cui è stato eseguito un porting sulla piattaforma Android usando NDK.
Seguono i Capp. 4 e 5 rispettivamente di validazione della soluzione ideata e delle conclusioni, con considerazioni sullo stato dello sviluppo, sui miglioramenti attuabili e sui punti critici che sono rimasti aperti a chiusura del lavoro di tesi.

Ringraziamenti

La stesura di questa tesi sancisce il momento in cui termino il mio percorso di Studi Accademici, un lungo periodo arricchito da intense esperienze e cambiamenti, anche personali, di non poco conto.
Dedico questo lavoro alla mia famiglia, in particolare a Fiorenzo che con il suo aiuto mi ha reso più consapevole e indipendente, questo è il trofeo tanto agognato.
Agli amici di sempre, ai compagni con i quali ho condiviso questa esperienza.
Grazie.

Indice

Elenco delle figure

Capitolo 1

La spazializzazione del suono

1.1 Il sistema uditivo umano

Le funzioni uditive sono di supporto a qualunque attività quotidiana e spesso sono cruciali per la vita e la sopravvivenza, la capacità di individuare una sorgente sonora nello spazio ha un ruolo importante nell'esperienza sensoriale poichè permette di compensare e bilanciare la percezione in assenza di altri stimoli come, ad esempio, quelli visivi e aptici. La capacità di percepire il suono ricopre un ruolo difficilmente sostituibile da altri stimoli sensoriali e il fatto che gli esseri umani sperimentino costantemente queste sollecitazioni, ed in misura maggiore rispetto a tutte le altre, rendono lo studio del fenomeno interessante e ricco di applicazioni pratiche.
L'elaborazione degli stimoli binaurali, cioè la capacità del cervello di processare l'informazione proveniente dai due organi uditivi fornisce agli esseri umani l'abilità di localizzare con immediatezza ed accuratezza la posizione di una sorgente sonora, in particolare nel piano orizzontale.

Il sistema uditivo è costituito dagli organi uditivi e dalle connessioni tra essi ed il sistema nervoso, dal punto di vista della disposizione fisica possiamo categorizzarli in orecchio esterno, medio e interno, nervo uditivo e percorsi uditivi centrali (si veda Fig. 1.2) [1].
L'orecchio esterno è costituito dalla pinna e dal canale uditivo: la pinna è la struttura cartilaginea dalla forma parabolica e soggettiva per ciascun individuo, e tra le innumerevoli particolarità minori è caratterizzata dalla forma parabolica della *helix*, la depressione corrispondente alla *concha* e del *tragus* ossia la protuberanza al di sopra di essa (vedi Fig. 1.1), diversamente da quanto si ritiene diffusamente, essa non ha il ruolo primario di veicolatore di onde sonore verso il canale uditivo, visto che la sensibilità agli stimoli rimane invariata rispetto ad un eventuale alterazione della sua forma (Bekesy e Rosenblith, 1958), bensì ha una funzione di risuonatore ed è fondamentale nella localizzazione delle sorgenti sonore (vedi Sez. 1.4).
Il canale di collegamento tra la *pinna* e il timpano è chiamato *canale uditivo* ed è costituito da una congiunzione cartilaginea-ossea, attraverso questo le onde sonore arrivano

ad infrangersi sulla superficie del timpano. Esso varia in forma e dimensioni, tuttavia è possibile idealizzare il suo comportamento come quello di un tubo acustico con una estremità occlusa e una aperta.
Andando più in profondità si trovano la cavità timpanica, la quale assieme al timpano ed alle strutture ossicolari costituisce l'orecchio medio, è questo l'organo che esegue la trasduzione degli stimoli meccanici in stimoli nervosi.
L'orecchio interno ospita gli organi cocleari, vestibolari e l'insieme di fluidi necessari per la trasduzione in informazione processabile dal cervello.
Di particolare interesse è inoltre la struttura ossea sulla quale l'orecchio poggia, di cui definisce proprietà fisiche di rigidità e la conformazione anatomica. L'*osso temporale* risente infatti delle vibrazioni sonore che sollecitano l'orecchio tramite la conduzione ossea, un fenomeno uditivo che veicola l'informazione bypassando l'orecchio esterno e medio grazie alle sollecitazioni sull'osso. Si dimostra così che l'attività cocleare è indipendente da come le sollecitazioni vengono generate. [1]

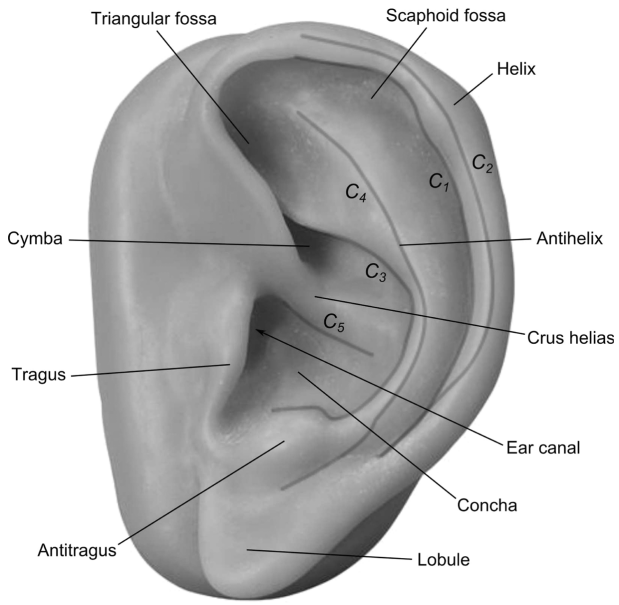


Figura 1.1: L'orecchio umano e le sue peculiarità anatomiche [8]

1.2 La percezione del suono

La *psicoacustica* studia le relazioni che intercorrono tra stimoli sensoriali e sensazioni percettive ed emotive, comprendere questo ci aiuta a progettare codifiche e sistemi che vanno incontro alle esigenze dell'ascoltatore massimizzando la qualità dell'esperienza di ascolto.
Le principali misure in ambito acustico e psicoacustico sono le seguenti: [1] [3]

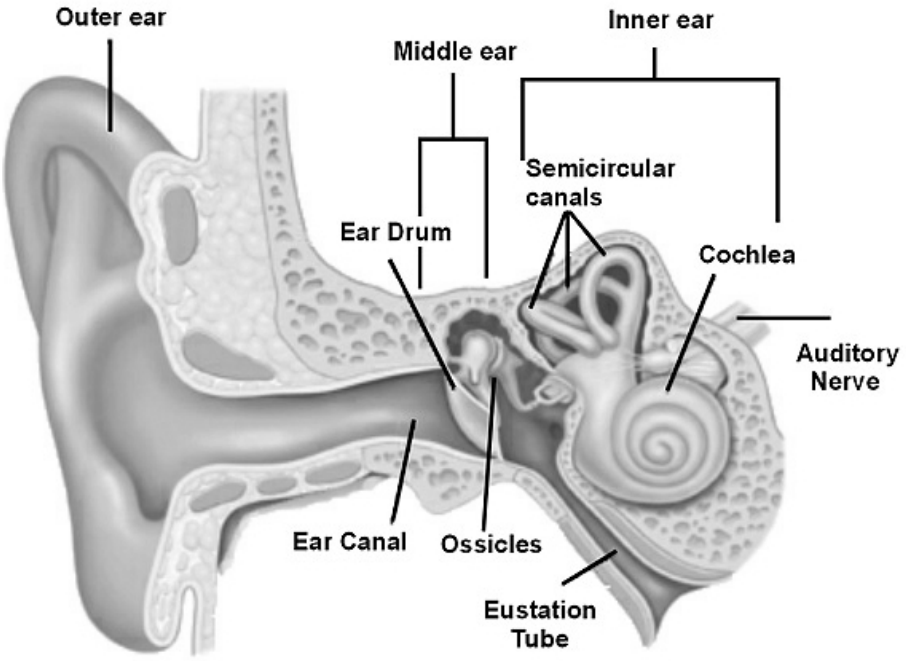


Figura 1.2: Una rappresentazione completa del sistema uditivo[1]

- **Sound pressure** rappresenta la pressione sonora impressa dall'onda che viaggia nel mezzo, definita dall'espressione

$$SPL = 10 \log_{10} (p/p_0)^2 \quad [dB] \tag{1.1}$$

dove $p(t)$ corrisponde alla pressione istantanea e $p_0 = 20\mu Pa$ è la pressione alla soglia in quiete a frequenze di 2 kHz [2]

- **Sound intensity** è la potenza per unità di area, definita con

$$SPL = 10 \log_{10} I/I_0 \quad [W/m^2] \tag{1.2}$$

dove I_0 corrisponde alla potenza di riferimento di un'onda di pressione p_0

- **Loudness** è una misura psicoacustica complementare dell'intensità, e solitamente la si intende come il "volume" di un suono. La sua quantificazione non è messa in rapporto uno-a-uno con la misura oggettiva di intensità, infatti la percezione di volume di un suono varia in base alla frequenza dello stesso. Le curve *equal-loudness contours* illustrano la relazione intensità - volume percepito [3].

- **Pitch** è il corrispondente psicoacustico della frequenza e corrisponde a ciò che abitualmente si definisce come "altezza" di un tono.
La scala percettiva del *pitch* non è lineare né monotonica, non è semplice definirne

una dimensione e spesso si adotta quella delle note musicali e dei loro intervalli. La *mel scale* è una quantificazione non lineare della percezione del pitch ed aiuta a metterlo in relazione con i valori di frequenza delle onde.

- **Hearing area**, la regione dell'udibile 20 Hz - 20 kHz, è la rappresentazione delle aree degli stimoli fisici udibili dall'uomo (vedi Fig. 1.3).
  Il *lower bound*, cioè il limite di sensibilità, è detto soglia in quiete (*threshold in quiet*) ed è importante per determinare qual è il limite percettivo inferiore della loudness alle diverse frequenze (si nota un picco nelle frequenze $2kHz - 20kHz$ corrispondente alla perdita di sensibilità di soggetti sottoposti a forti stress sonori).
  L'*upper bound* rappresenta la soglia del danneggiamento (*threshold of damage*) attorno alla quale si hanno effetti deleteri sull'udito.

- **Frequency masking e temporal masking** sono fenomeni psicoacustici che si verificano quando si sovrappongono più sorgenti sonore in frequenza oppure nel tempo. Spesso le esperienze uditive sono affette da questi fenomeni che alterano le caratteristiche dei suoni puri, studi sulla modella a filtri del sistema uditivo hanno prodotto modelli delle *critical bands*, e modelli di *equivalent rectangular bandwidth (ERB)* [7].

Per quanto concerne la *sound equality* a livello sensoriale, il concetto di *stimuli - sensations* [2] cerca di relazionare la sollecitazione fisica con le dimensioni sensoriali dell'ascoltatore: si cerca di comparare gli step di stimolazione con gli step della conseguente sensazione, cercando di stabilire delle soglie di ampiezza entro cui uno stimolo produce lo stesso effetto. Per rappresentare la relazione tra queste quantità è possibile produrre delle formule o dei grafici di funzione che mettono in relazione l'"intensità dello stimolo" con "l'intensità della sensazione" ottenendo valori di "soglie psicoacustiche". [2]
Ad esempio affermando che "il tono con il pitch più alto è ad un volume maggiore rispetto al tono con pitch più basso" si misurano due entità differenti, il pitch e il volume (o loudness) e possiamo perciò cercare una formulazione che li metta in relazione.
La necessità di queste misure è data dalla volontà di modellare la resa emozionale e sensoriale secondo parametri il più possibile oggettivi e dunque creare strumenti che siano concreti ed efficaci nella resa.

1.3 Suono e *room acoustics*

1.3.1 Propagazione sonora in ambienti chiusi

Le onde sonore sono variazioni di pressione che si propagano in un mezzo, nel caso più comune, l'aria. Nella nostra trattazione del fenomeno, lo consideriamo come un segnale monodimensionale costituito da una composizione di onde di pressione caratterizzate da frequenza, ampiezza e fase eventualmente interessate da fenomeni dovuti all'ambiente e rumori.
Una parte dello studio dell'acustica, detta *room acoustics*, studia le influenze sul suono causate dalla conformazione dell'ambiente in cui esso si diffonde e come questo viene

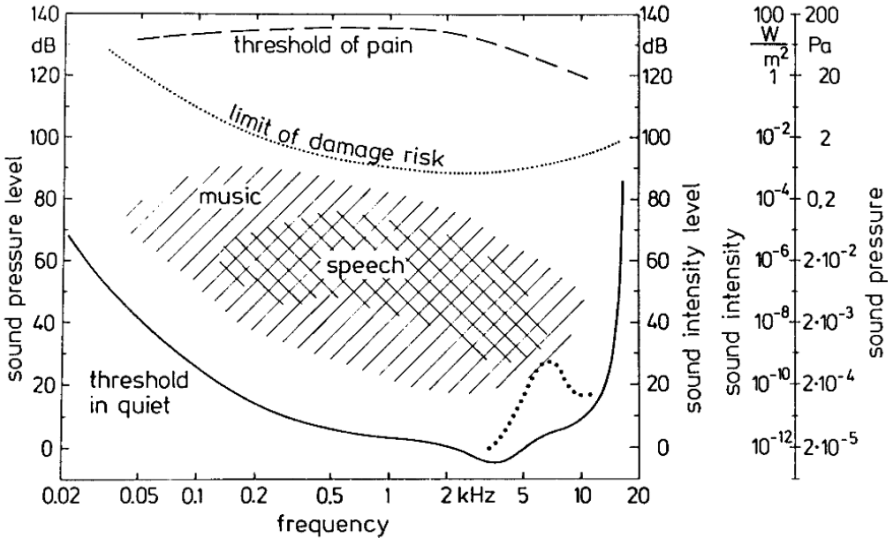


Figura 1.3: La *hearing area*, una rappresentazione grafica dell'area dello spettro udibile, caratterizzata dalla *threshold in quiet*, la soglia in quiete, in basso e la *threshold of damage* la soglia del danneggiamento, in alto [2]

trasformato.
In tutte le situazioni quotidiane siamo immersi in un ambiente sonoro non-anecoico, e qualsiasi suono che giunge alle nostre orecchie ha subito innumerevoli trasformazioni dovute alla conformazione dell'ambiente in cui le onde si sono propagate. Supponiamo di avere un ambiente costituito da una stanza a parallelepipedo, in cui il mezzo trasmissivo elastico è l'aria, i cui muri sono rigidi e all'interno della quale le onde sonore si riflettono. Propagandosi in un ambiente non-anecoico il suono si trasforma e si "colora" arricchendo il proprio contenuto spettrale in base a trasformazioni quali ad esempio: rifrazioni, riflessioni, diffrazioni, effetto Doppler e riverberazione.

Se pensiamo ad un ambiente chiuso come ad un sistema LTI, un risuonatore tridimensionale di grandi dimensioni fisiche, a tali sistemi possiamo applicare la *sintesi modale* cioè caratterizzare il comportamento di un risuonatore usando una sovrapposizione di N oscillatori del secondo ordine, ciascuno dei quali rappresenta una reazione del sistema alle sollecitazioni.
Nel caso in esame la *sintesi modale* ci permette di ricavare la forma delle componenti modali, ma portare a termine questo tipo di sintesi usando equazioni analitiche dell'onda non sempre porta a definire un insieme di risultati concreti perché la complessità di calcolo è proporzionale alla densità modale, la quale esplode rendendo il problema complesso dal punto di vista matematico e computazionale.

Per descrivere i casi più generali in cui la forma geometrica dell'ambiente non è

regolare si studia la *geometrical room acoustics*, il cui fondamento principale è trattare le onde sonore sotto forma di *acoustic rays*, cioè raggi che rappresentano onde a frequenze estremamente elevate. La semplificazione delle onde sferiche in raggi trova giustificazione se il rapporto tra le dimensioni fisiche della stanza e delle mura e la lunghezza d'onda delle onde sonore è ampio, cosa che si verifica facilmente negli ambienti reali. [4]
Un *acoustic ray* è un punto nello spazio rappresentante la parte infinitesimale di un'onda sferica emessa in una determinata posizione nella stanza, tale raggio possiede velocità e direzione ben definite e veicola un ammontare di energia costante, e più in generale vi è grande similitudine con lo studio dei raggi ottici che hanno simili caratteristiche fisiche. L'intensità del raggio acustico decresce con la distanza r dalla sorgente sonora secondo la legge $1/r^2$. Un altro aspetto fondamentale è il modo con cui si assume avvengano le riflessioni: si ipotizza l'assenza di rifrazioni e che le diffrazioni siano trascurabili, inoltre si assume che il raggio riflesso rimanga nel piano definito dal raggio incidente e la normale alla parete e gli angoli di incidenza si conservino uguali in entrata e uscita.

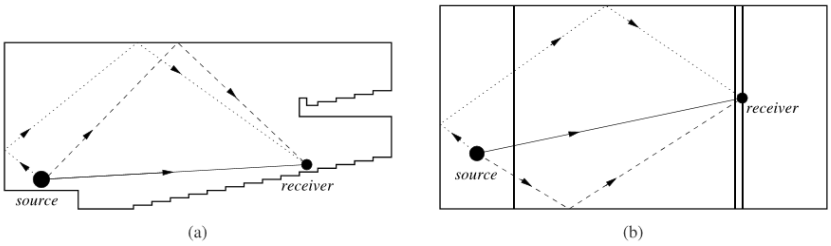
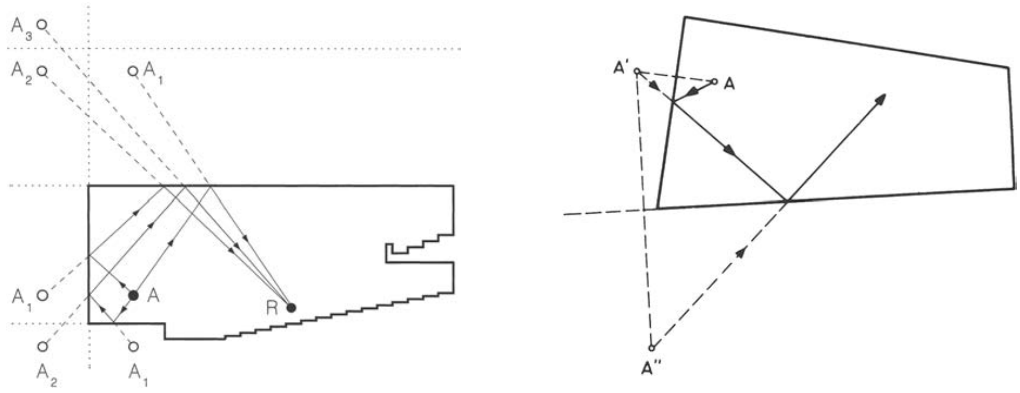


Figura 1.4: Una schematizzazione del concetto di *room acoustics*, una sezione verticale in *(a)* ed una orizzontale in *(b)* in cui si distinguono i percorsi delle riflessioni di primo ordine e quelle di secondo ordine [7]

In figura 1.4 vi è un'analisi semplificata di uno scenario di geometrical room acoustics, tutti i percorsi delle onde dalla sorgente al ricevitore possono essere identificati in base al numero di riflessioni: le onde che arrivano direttamente dalla sorgente all'ascoltatore sono dette *direct sound*, le onde che hanno subito una riflessione prima di giungere all'ascoltatore sono le *first-order reflections*, seguite da un numero ancora maggiore di doppie riflessioni dette *second-order reflections* e così via. Ragionare in questo modo ci permette di ricavare una formulazione della *room impulse respnse (RIR)* cioè la risposta impulsiva particolare dell'ambiente: si ipotizzi di emettere un impulso all'interno di una stanza al tempo $t = 0$, ogni raggio riflesso giungerà al ricevitore dopo un ritardo definito e con energia attenuata a causa della resistenza del mezzo e dall'assorbimento delle pareti. Le riflessioni di ordine inferiore saranno più percepibili, perché sporadiche e ancora ricche di energia mentre le riflessioni di ordini superiori saranno percepibili unitariamente data la loro densificazione nel tempo e inferiore energia.

Per meglio comprendere la distribuzione temporale delle riflessioni si parla del con-



(a) *(b)*

Figura 1.5: Costruzione geometrica di *image sources* in un auditorium di cui è riportata la sezione verticale *(a)* e la sezione orizzontale *(b)* [7]

cetto di *image sources* (vedi figura 1.5), uno strumento che facilita il compito di costruire i percorsi di riflessione delle onde.
Si supponga di avere una sorgente A che emette un suono all'interno della stanza, è possibile considerare l'onda riflessa come se avesse origine A' in un punto esterno alla stanza, con una intensità attenuata di una misura α e una diversa componente spettrale, usando lo stesso principio si modellano le riflessioni di ordine superiore A'' e così via, ed a questo punto la conformazione geometrica della stanza può essere rimossa dalla formulazione in quanto il suo contributo è catturato dalle *image sources* così costruite.
Il suono che si riceve in un punto R della stanza è perciò calcolabile sovrapponendo i contributi di tutte le *image sources*, assumendo che l'emissione del suono sia simultanea, si produrranno un insieme di onde di varia intensità e con ritardi dovuti alle varie distanze che le onde devono percorrere. Se si adotta la semplificazione che stabilisce che l'assorbimento delle pareti sia indipendente dalla frequenza, risulterà che il segnale al ricevitore $s'(t)$ è formato da una serie infinita di copie del suono originale, ciascuna con intensità A_n e ritardo t_n, quindi si definisce

$$s'(t) = \sum_n A_n s(t - t_n) \tag{1.3}$$

al quale corrisponde una risposta impulsiva del tipo

$$g(t) = \sum_n A_n \delta(t - t_n) \tag{1.4}$$

otteniamo dunque una rappresentazione della RIR in termini delle *image sources* che caratterizzano l'ambiente.

1.3.2 Psicoacustica e percezione spaziale

Lo studio della *room acoustics* si focalizza anche sulla resa psicoacustica di una scena virtuale, studiando le cosiddette *spatial impression* e *listener envelopment*, cioè le proprietà di un ambiente di veicolare la tridimensionalità della scena stessa.
Una proprietà dominante è la riverberazione (o riverbero) di una stanza, cioè la misura della presenza del suono dopo che questo ha terminato la propria influenza diretta sulla scena. La riverberazione è caratterizzata dal tempo di riverberazione che corrisponde al tempo necessario perché un suono decada di 60 dB dopo che è cessato, ed il suono percepito come riverbero dall'ascoltatore è costituito dalle diverse riflessioni del suono all'interno della stanza, la RIR rappresenta così accuratamente le proprietà acustiche di un ambiente poiché estrae il profilo energetico di tali componenti, come visto nella Sez. precedente (vedi Fig. 1.6).

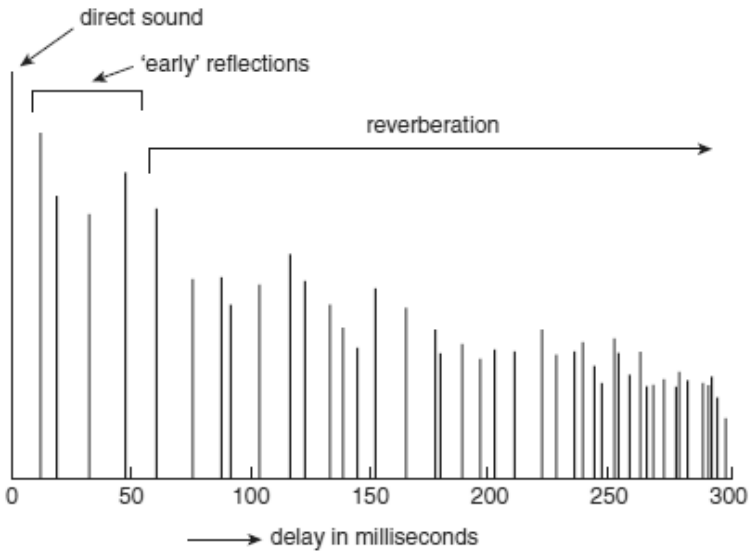


Figura 1.6: Rappresentazione della *room impulse response* come profilo energetico delle riflessioni delle onde [4]

Spesso si identifica una misura complementare alla riverberazione, chiamata riverberanza e definita come la capacità di un ambiente di arricchire spettralmente un suono secco e di dare all'ascoltatore la sensazione che il suono si "infranga" su di esso, ed una rappresentazione pratica di questa proprietà è data dalla cosiddetta *waterfall RIR* (vedi Fig. 1.7) in cui si evince in che modo le riflessioni decadono per ogni range di frequenza nel tempo.

[2] http://blog.bjornroche.com

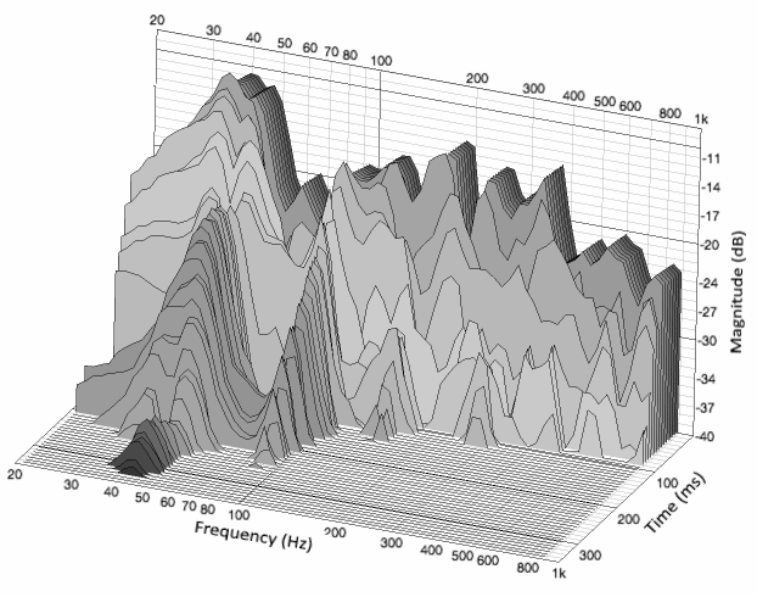


Figura 1.7: Rappresentazione di una *waterfall RIR* in cui si mettono in relazione ampiezza, tempo e frequenza[2]

Morimoto [9] afferma che la *auditory spatial impression* è definita da *auditory source width (ASW)* cioè l'ampiezza di una *sound image* in relazione con la *direct sound image* precedente, e dal *listener envelopment (EV)* inteso come "grado di ricchezza" del suono che circonda l'ascoltatore, entrambi questi indicatori sono in stretta relazione con il primo fronte d'onda distinguibile come la prima componente nel tempo (vedi Fig. 1.6).
Si suddividono i contributi successivi nel tempo in due insiemi [5]:

1. le *early reflections* sono cruciali per veicolare l'intelligibilità del parlato, il volume, la ASW e la *clarity* ovvero alla proprietà di un ambiente di essere "trasparente" rispetto ad un suono, e rendere un brano musicale intelligibile

2. la *late reverberation* è responsabile di "colorare" il suono e renderlo più simile a quanto si percepisce quotidianamente in ambienti chiusi

Interessante è notare come lo studio della psicoacustica volga le proprie attenzioni allo studio tra la percezione sonora e le trasposizione motoria delle sensazioni e viceversa, le cosiddette *motor theory of perception* ed la *embodiment theory*. Queste teorie rappresentano gli studi su come la multimodalità influisce sulla percezione e sul perché è ricca dal punto di vista informativo e stimola il cervello in misura maggiore [7].
Altri studi di psicoacustica mettono in relazioni gli stimoli visivi e quelli sonori per studiare come si influenzano vicendevolmente, fenomeni quali il *McGurk effect* dimostrano quanto queste siano correlate.

1.4 Audio 3D e spazialità del suono

1.4.1 Binauralità: il suono nello spazio

È possibile descrivere l'esperienza uditiva considerando tre suddivisioni di attributi percettivi chiamate *elemental senses*, ciascuna delle quali subisce la valutazione soggettiva dell'ascoltatore e contribuisce alla valutazione della qualità e piacevolezza del suono: [9]

- attributi temporali, quali ad esempio ritmo, durata, riverbero, ecc.
- attributi spaziali come direzione, distanza, *spatial impression* (si veda Sottosez. 1.3.2), ecc.
- attributi qualitativi come volume, timbro, pitch, ecc.

Per quanto concerne l'audio 3D e la spazialità del suono i deve volgere l'attenzione all'area degli attributi spaziali, sono questi gli indicatori che influenzano in maniera decisiva la capacità percettiva della tridimensionalità del suono.
La caratteristica chiave per la percezione spaziale del suono negli esseri umani è la binauralità, cioè la capacità di percepire suoni da entrambi i canali uditivi e di processarli per ottenerne un percetto unitario. Quando si ascoltano suoni monoaurali, ciò accade se ad esempio ci tappiamo accuratamente una cavità uditiva, si accede ad un *aural space*[3] differente da quando ascoltiamo in modalità binaurale.
Le abilità nel localizzare i suoni sono sensibilmente migliori quando un essere umano ascolta il suono usando entrambi i flussi informativi delle proprie orecchie, inoltre aumenta anche la capacità di capire se un suono proviene da una sorgente diffusa oppure concentrata in uno specifico punto nello spazio [6].

L'individuazione della posizione di una sorgente sonora è dovuta principalmente alla *legge del primo fronte d'onda* (o *precedence effect*), il fenomeno che avviene quando si ascoltano suoni in ambienti chiusi non anecoici ed è costituito da *localization dominance* definita dall'onda sonora diretta proveniente dalla sorgente e dalla fusione delle riflessioni della stanza (vedi fig. 1.6) [6].

Lo studio della binauralità della percezione si è rivelato una svolta dal punto di vista della ricerca in molti ambiti relativi all'audio, accelerando la ricerca accademica e lo sviluppo di soluzioni commerciali per i seguenti scopi [6]:

- *Mappatura spaziale delle scene sonore* per produrre stime spaziali delle *aural scene* sia in scenari reali che virtuali
- *Studio delle scene sonore per ricavarne modelli a livello di segnale* per codifica e sintesi di scene uditive e per migliorarne la resa percettiva
- *Studio delle scene sonore per ricavarne modelli a livello simbolico* in modo da mappare gli aspetti più importanti nella resa binaurale di una scena

[3] con il termine *aural space* si definisce la totalità dei suoni percepiti in una scena [6]

- *Valutazione della qualità di una auditory scene* misurando parametri oggettivi e soggettivi per progettare ambienti che siano adatti a performance di parlato, musicali, teatrali ecc.

Nel tragitto dalla sorgente al timpano dell'ascoltatore il suono subisce una serie di trasformazioni che possiamo assumere lineari e perciò modellabili come una funzione di trasferimento. L'obbiettivo è quello di stimare accuratamente le due *Binaural Room Impulse Response (BRIR)* soggettive dell'ascoltatore, ciascuna delle quali è una somma della *Room Impulse Response (RIR)* dell'ambiente (vedi Sez. 1.3) e della *Head-Related Impulse Response (HRIR)* che cattura i contributi dovuti alla conformazione fisica dell'ascoltatore. Nello specifico si cerca di modellare le *HRIR*, denotate da un alto grado di soggettività, attraverso le quali è possibile riprodurre in cuffia il suono come se stesse subendo le stesse alterazioni di una configurazione *open field* reale.
Un fenomeno interessante da questo punto di vista si verifica se si utilizzano funzioni di trasferimento non appartenenti ad un soggetto per realizzare una sintesi sonora a lui destinata, questi non è in grado di localizzare le sorgenti sonore con la stessa accuratezza di quanto si avrebbe con una modellazione tarata sulle sue peculiari caratteristiche fisiche, ad esempio si hanno maggiori difficoltà ad individuare le posizioni verticali delle sorgenti, la loro distanza ed addirittura si incrementa il tasso di errori di localizzazione *front-back* [10].

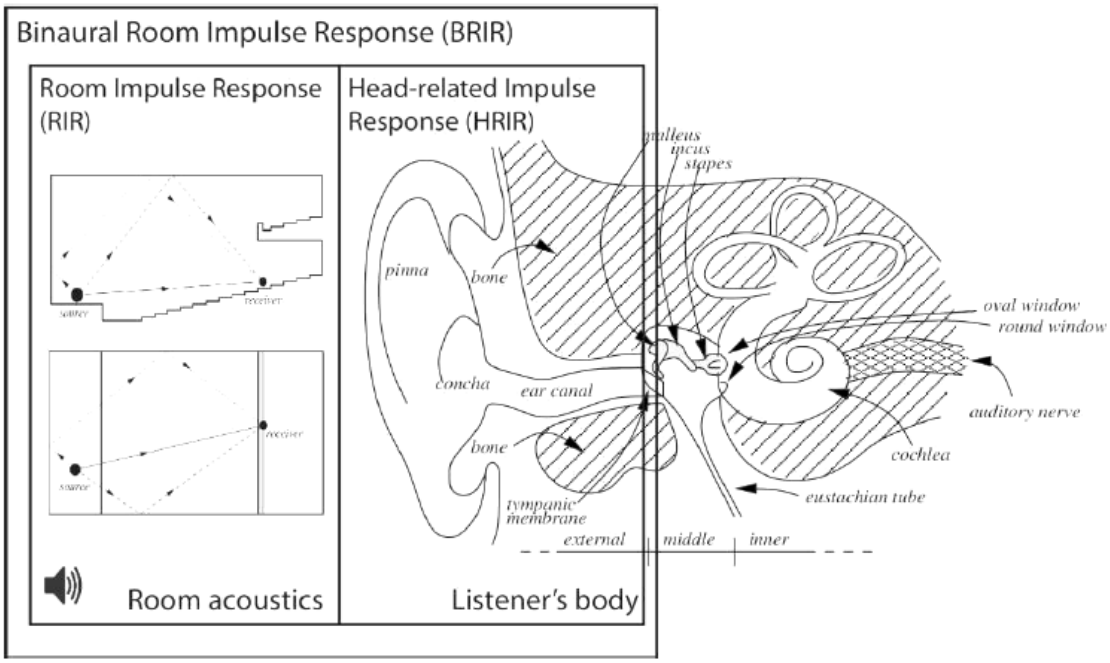


Figura 1.8: Le *Binaural Room Impulse Response (BRIR)* come composizione di contributi ambientali e soggettivi dell'ascoltatore [18]

Vi sono molti approcci differenti per modellare il comportamento binaurale con diversi gradi di semplificazione, per quanto riguarda il nostro lavoro ci concentriamo sullo studio di *modelli di localizzazione* che ci forniscono una stima accurata di come viene percepito il suono in modalità binaurale tramite la valutazione di una serie di *indicatori di localizzazione* e monoaurale sotto forma di filtraggio del suono da parte dell'orecchio esterno.

Il sistema di riferimento tipicamente utilizzato in questo tipo di rilevazioni comprende i tre piani tipicamente utilizzati in biologia: frontale, orizzontale e mediano/sagittale (si veda Fig. 1.9).
Gli angoli tipicamente utilizzati per caratterizzare la posizione di una sorgente audio sono l'angolo di *azimuth* (ovvero l'angolo orizzontale) θ compreso tra le rette giacenti sul piano orizzontale definite dall'asse x e la retta congiungente l'origine degli assi con la proiezione sul piano orizzontale della posizione della sorgente.
L'angolo ϕ di *elevation* (ovvero l'angolo verticale) è compreso tra le rette giacenti sul piano sagittale definite dall'asse z e la retta congiungente l'origine degli assi con la proiezione della posizione della sorgente sonora sul piano sagittale stesso.

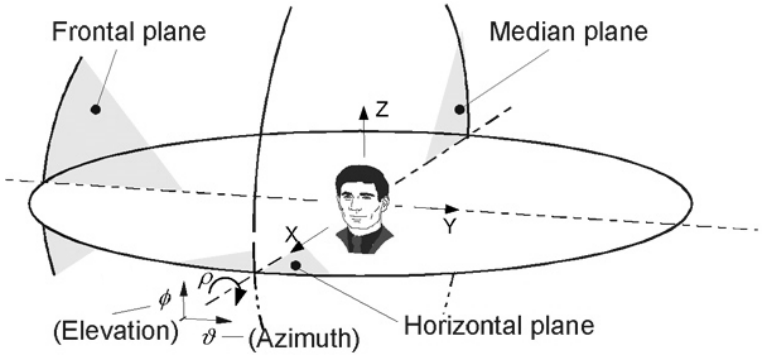


Figura 1.9: Il sistema di riferimento della testa, distinguiamo i diversi assi frontale (o *coronale*), orizzontale (o *trasverso*), mediano (o *sagittale*) e gli angoli salienti θ per l'*azimuth* e ϕ per l'*elevation* [11]

Sebbene si caratterizzino le posizioni spaziali come angoli geometrici misurabili in modalità continua tipicamente i sistemi di coordinate in letteratura considerano solamente posizioni discrete.
L'insieme delle posizioni dalle quali vengono emessi un suono definiscono una griglia spaziale che si può immaginare essere un insieme di punti giacenti su una sfera al cui centro vi è la testa dell'ascoltatore, ciascuno di tali punti è perciò una posizione ad un determinato angolo di azimuth e di elevazione univoci. Il raggio della sfera deve essere superiore ad 1 m per evitare di entrare nella condizione di *near-field* (in cui molte delle semplificazioni fatte perdono di significato), i loudspeakers riproducono a turno un segnale analitico che viene a sua volta registrato tramite microfoni posti nel canale uditivo

dell'ascoltatore, vicino al timpano oppure in alte posizioni dell'organo uditivo.
I sistemi di coordinate più usati sono *interaural polar* (database CIPIC) e *vertical polar* (database LISTEN e molti altri): il primo possiede punti sulla sfera appartenenti all'intersezione tra piani paralleli al piano sagittale e piani giacenti sull'asse interaurale, opportunamente distanziati di un passo prestabilito. Il secondo sistema adotta convenzioni differenti, i punti sono sulla sfera e appartenenti all'intersezione tra piani paralleli al piano orizzontale e piani giacenti sulla retta ortogonale a quella interaurale.

Testa Gli indicatori interaurali sono caratteristici della testa e sono la *interaural time difference (ITD)* e la *interaural level difference (ILD)*, queste quantità influiscono sulla percezione di segnali uditivi ai due orecchi.
La ITD è un ritardo che si misura nella situazione in cui una sorgente sonora si trova lateralmente rispetto al soggetto e perciò vi è il disturbo della *head shadow*, l'onda che si infrange nell'orecchio in posizione ipsilaterale (più vicino) arriva prima rispetto a quella che raggiunge l'orecchio controlaterale (opposto), dato che deve percorrere una distanza superiore per giungere al timpano.

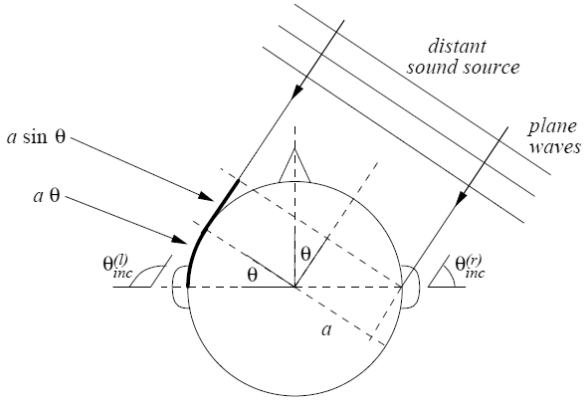


Figura 1.10: Illustrazione della configurazione che produce *ITD* e *ILD*, le onde si infrangono sulla testa dell'ascoltatore e vengono percepite in maniera differente ai due lati [7]

È necessario fare alcune assunzioni, che sono plausibili solamente nel caso di sorgente sonora "distante" dalla testa dell'ascoltatore e considerando una forma perfettamente sferica della testa, perciò nella configurazione in Fig. 1.10 si assume che le onde siano onde piane perciò l'onda diretta all'orecchio controlaterale percorre una distanza Δx rispetto alla distanza percora dall'onda all'orecchio ipsilaterale e dunque la ITD è pari a $\Delta x/c$, se si conosce il diametro della testa vale:

$$ITD \sim \frac{a}{c}(\theta + \sin\theta) \tag{1.5}$$

La formula (1.5) afferma che la *ITD* è pari a zero se la sorgente è perfettamente di fronte all'ascoltatore $\theta = 0$, e si ha *ITD* massima, pari a $a/c(\pi/2 + 1)$, quando l'angolo è $\theta = \pi/2$, inoltre, come si evince dalla formula la *ITD* è indipendente dalla frequenza.
La *ILD* misura la differenza di *sound pressure* ai due timpani ed è una misura fortemente dipendente dalla frequenza,: a basse frequenze l'ombra acustica della testa è pressochè inesistente mentre diviene importante all'aumentare della frequenza dell'onda sonora.
Rimanendo nella configurazione di approssimazione sferica della testa, se la sorgente è "distante" dalla testa (sorgente distante $r > a$), usando le variabili normalizzate $\mu = \omega a/c$ per la frequenza e $\rho = r/a$ per il raggio, si ha:

$$H_{sphere}(\rho, \theta_{inc}, \mu) = -\frac{\rho}{\mu} e^{-i\mu\rho} \sum_{m=0}^{+\infty} (2m+1) P_m(\cos\theta_{inc}) \frac{h_m(\mu\rho)}{h'_m(\mu)} \tag{1.6}$$

dove P_m e h_m sono rispettivamente il polinomio di Legendre e la funzione sferica di Hankel di ordine m-esimo e l'angolo θ_{inc} è l'angolo di incidenza sulla sfera calcolato in termini di θ come $\theta_{inc}^{(l)} = \pi/2 - \theta$ e $\theta_{inc}^{(r)} = \pi/2 + \theta$.
A basse frequenze la formula (1.6) ha modulo unitario, quando μ eccede 1 la dipendenza della formula da θ_{inc} è presente. I valori massimi si hanno quando la sorgente è posta frontalmente all'ascoltatore, con una differenza di circa 6 dB che decresce mano a mano che ci si sposta verso la parte posteriore della testa.

Orecchio esterno Si modella questa componente separando il contributo del padiglione o *pinna* dal contributo del canale uditivo.
La pinna riceve onde da diverse direzioni (vedi Fig. 1.11), e varia la propria risposta in frequenza in funzione della direzione dalla quale proviene il suono. A basse frequenze la pinna è trasparente per quanto riguarda la risposta in frequenza poiché i suoni sono generalmente in fase, mentre al crescere della frequenza il comportamento risuonatore fa sì che alcune frequenze vengano amplificate ed altre attenuate a causa degli effetti di interferenza distruttiva dovuti allo sfasamento delle onde che percorrono distanze differenti per arrivare al canale uditivo.

I risultati evidenziano che il fenomeno di rumore distruttivo nella pinna si riflette nel cosiddetto *pinna notch*, ovvero degli avvallamenti nella risposta in frequenza a determinate frequenze. Sorprendentemente si rileva che la frequenza di notch f_{pn} è in relazione con l'angolazione della pinna rispetto alla posizione verticale, l'angolazione della pinna verso l'esterno e l'altezza della *fossa auricolare* (vedi Fig. 1.1) [7].
Inoltre, i risultati mostrano come i due parametri f_{pn} e *ITD* siano sufficienti a descrivere accuratamente la variazione delle *HRTF* dei soggetti (si veda Sez. 1.4.1), ed emerge che la *ITD*, come si era ipotizzato, è dipendente dalla dimensione della testa e dunque da imputarsi alla *head shadow* (vedi fig. 1.13 (b)).
Ad ogni modo, il *pinna notch* è molto più pronunciato in rilevazioni che variano in *elevazione* rispetto a quelle che variano in *azimuth*, e questo perché variando la posizione sul piano verticale si ha la massima variazione dei percorsi seguiti dalle onde. (vedi Fig. 1.11)

Il canale uditivo agisce come un semplice tubo risuonatore monodimensionale, e agisce come un filtro semplice che varia la propria risposta in frequenza in base alla distanza della sorgente e alla posizione da cui entrano le onde sonore, come già menzionato è modellabile come un tubo risuonatore il cui diametro misura tipicamente 8mm e la lunghezza è di 25mm [7].

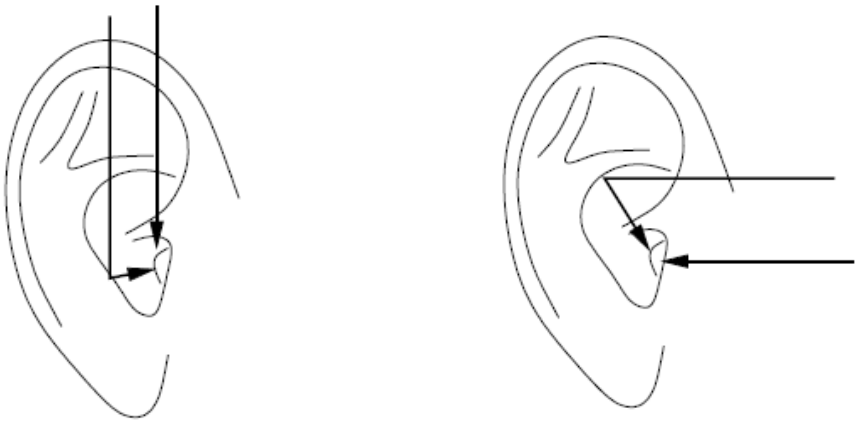
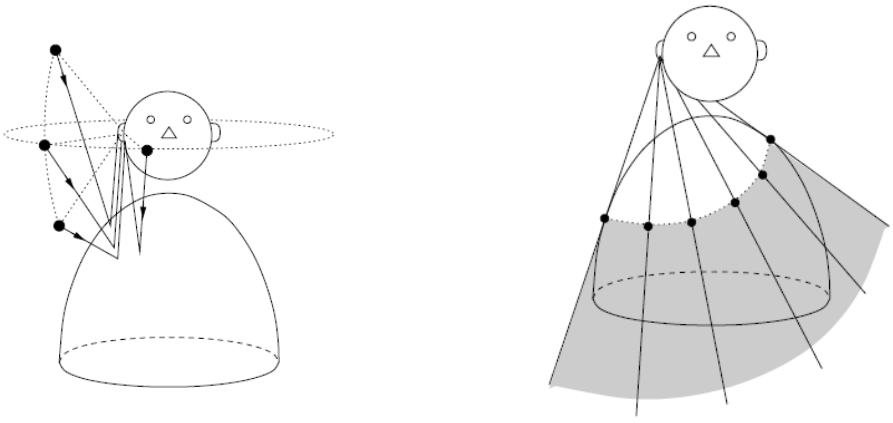


Figura 1.11: Due differenti percorsi delle onde per arrivare all'ingresso del canale uditivo [7]

Torso e spalle Le parti del corpo dell'ascoltatore influiscono in maniera sensibile sulla percezione spaziale del suono a causa di riflessioni e *shadowing*. Questi due fenomeni sono apprezzabili poiché incidono maggiormente alle basse frequenze, in cui i segnali possiedono forte energia e in cui contemporaneamente la risposta dovuta alla pinna è trasparente.
La geometria del torso è complicata ed è necessario approssimarla, si usa tipicamente il cosiddetto *snowman model*. (vedi Fig. 1.12)
A livello qualitativo è lecito affermare che il ritardo introdotto da questi due fenomeni non è apprezzabile se la sorgente si sposta sul piano orizzontale, mentre varia molto di più se si muove sul piano sagittale poiché così facendo il torso ha influenze occlusive sul tragitto delle onde e le spalle hanno influenza più o meno accentuata di tipo riflessivo.

HRIR e HRTF Spesso nella letteratura si introducono le *Head-Related Transfer Functions (HRTF)*, che sono le trasformate delle *HRIR* viste in precedenza. Le *HRTF* catturano tutte le alterazioni appena descritte e sono gli strumenti matematici che si hanno a disposizione per rappresentare le singole influenze dovute all'antropometria in un unico contributo.
Le *HRTF* sono funzioni di tre parametri fondamentali: angolo *azimuth* θ, angolo *elevation* ϕ e raggio r e vengono indicate con $H^{(l),(r)}(r, \theta, \phi, \omega)$ in cui *(l)* e *(r)* diversificano le funzioni per canale sinistro e destro, rispettivamente.



(a) *(b)*

Figura 1.12: Lo *snowman model*, illustra i fenomeni di *reflection (a)* e *shadowing (b)* [7]

Nel caso in cui $r \to +\infty$, che in molti casi date le piccole dimensioni si traduce in $r > 1$ m, si parla di configurazione *far field* e la formulazione è $H^{(l),(r)}(\theta, \phi, \omega)$ e se si opta per l'ipotesi di perfetta simmetria si ha $H(\theta, \phi, \omega)$ in cui $H^{(l)}(\theta, \phi, \omega) = H(-\theta, \phi, \omega)$ e $H^{(r)}(\theta, \phi, \omega) = H(\theta, \phi, \omega)$, in cui si considera soltanto la differenza dell'angolo azimutale.

La definizione formale che si da alle *Head-Related Transfer Function* è il rapporto, dipendente da frequenza e spazio, tra *sound pressure level (SPL)* al timpano $\Phi^{(l),(r)}(\theta, \phi, \omega)$ e la *sound pressure level* in configurazione di *free-field*[4] al centro della testa $\Phi_f(\omega)$ in assenza dell'ascoltatore [7]:

$$H^{(l),(r)}(r, \theta, \phi, \omega) = \frac{\Phi^{(l),(r)}(\theta, \phi, \omega)}{\Phi_f(\omega)} \tag{1.7}$$

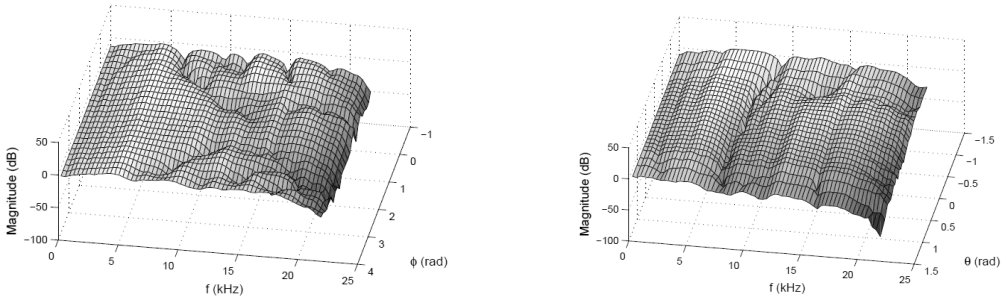
Come è possibile immaginare, avere a disposizione un modello di HRTF è fondamentale per eseguire il rendering sonoro. Molti prodotti commerciali che impiegano le *HRTF* per il rendering si servono di funzioni di trasferimento standard o registrazioni eseguite su manichini di tipo *KEMAR (Knowles Electronics Manikin for Auditory Research)* [13], di cui si dispone per la validazione sperimentale (si veda Cap. 5).

Il database CIPIC[5] contiene rilevazioni sperimentali delle HRIR appartenenti a 45 soggetti a 25 differenti angoli di *azimuth* e 50 differenti gradi di elevazione assieme ad una serie di misurazioni antropometriche sulla posizione degli organi uditivi esterni, dimensioni della testa e misurazioni dettagliate sulla pinna: le misurazioni sono eseguite usando array di *loudspeakers* da cui si emettono suoni che vengono registrati con sonde

[4] La configurazione *free-field* è tipica di un ambiente in cui non avvengono riflessioni delle onde sonore, come ad esempio in camera anecoica, dove le pareti sono pressoché totalmente assorbenti

[5] http://interface.cipic.ucdavis.edu/ e [12]

microfoniche posizionate nel canale uditivo [15]. Altri database di dominio pubblico, tra cui il popolare LISTEN[6], si differenziano nei metodi di setup e nelle scelte di registrazione.



(a) *(b)*

Figura 1.13: Rappresentazioni dell'ampiezza della risposta di HRTF al variare di *(a) elevation* ϕ e *(b) azimuth* θ [7]

1.4.2 Gli indicatori binaurali per la localizzazione dei suoni

Descrivere come avviene la traduzione degli indicatori binaurali in informazione di spazializzazione è un compito molto complicato, considerato l'alto numero di variabili e influenze che è necessario includere nel ragionamento. Per quanto concerne la presente trattazione, si lavorerà in un contesto in cui è forte la semplificazione, come del resto visto nelle sezioni precedenti.

Localizzazione azimutale La *ITD* e la *ILD* sono considerate come principali indicatori per la localizzazione azimutale, a cui spesso si fa riferimento parlando di *Duplex Theory of Localization* [Rayleigh (1907)].
Qualitativamente, se la lunghezza della semionda sonora è superiore al diametro della testa e quindi la frequenza è sufficientemente bassa, la *ITD* è apprezzabile e distinguibile, dall'altro lato aumentando di frequenza si introducono ambiguità poiché avendo una semionda troppo corta non è più possibile distinguere qual'è l'onda principale da quella in ritardo.
Se si osserva la *ILD* la situazione è al rovescio, a basse frequenze la *head shadow* non esiste e la *ILD* non è apprezzabile, mentre alle alte frequenze è più percepibile, è per questo motivo che la teoria afferma la complementarietà di queste due proprietà per la localizzazione azimutale, dato che combinando le due si riesce ad avere un parametro valido per l'intero insieme di frequenze. In aggiunta a quanto descritto, è importante segnalare che il cervello possiede l'abilità di percepire la *ITD* e disambiguarla analizzando gli inviluppi di ampiezza, cioè ricavare il ritardo non dai valori di fase delle onde,

[6] http://recherche.ircam.fr/equipes/salles/listen/

bensì estrae il ritardo ascoltandone i differenti livelli del suono, ad esempio confrontando i ritardi tra due picchi di ampiezza, valutando in questo modo la *Interaural Envelope Difference (IED)*.
In generale la localizzazione frontale può essere ambigua dato che una sorgente ad azimuth θ ed una a $\theta - \pi$ produce identici indicatori perciò si verifica la *front-back confusion* nella percezione. Le motivazioni di queste inesattezze vanno cercate nelle semplificazioni fatte (testa perfettamente sferica, influenze delle asimmetrie facciali non catturate, occlusioni da parte dei capelli, orecchie posizionate sull'asse sferico, ecc.). In ambienti riverberanti la *Duplex Theory* non ha performance ottimali poiché essi risentono del rumore, il quale inficia le proprietà della *Interaural Time Difference (ITD)* illustrate in precedenza.

Localizzazione di elevazione Le semplificazioni adottate nella modellazione precedente producono il fenomeno dei *cones of confusion*, identificati come delle superfici coniche che si estendono all'esterno di un orecchio, su una sfera, producono valori di *ITD* e *ILD* identiche. In realtà, le asimmetrie facciali mitigano questo fenomeno, che tuttavia si può avere in presenza di valori molto simili.

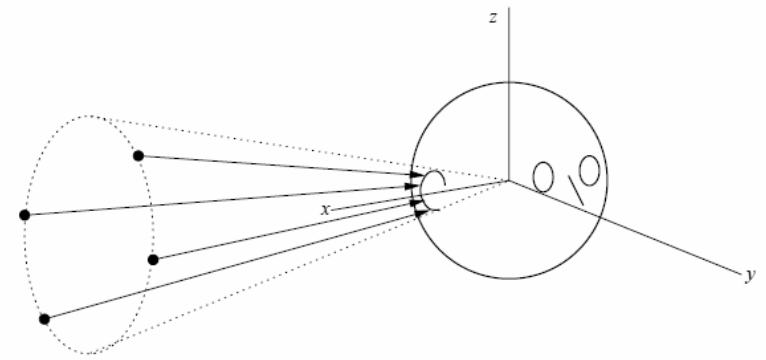


Figura 1.14: Il cono di confusione posto lateralmente alla testa, sul bordo del quale i suoni possiedono indicatori spaziali simili [7]

In generale, la struttura della pinna è di grande aiuto per definire la percezione verticale, la quale si è dimostrato essere un indicatore monoaurale.
Gli indicatori spettrali dati dalla pinna sono molto complessi e si ipotizza che nella percezione di elevazione essa agisca introducendo *notch* e *peak* spettrali, caratteristici di ogni soggetto e per lo specifico angolo di elevazione. Si ritiene inoltre che per l'indicazione di elevazione l'onda debba avere sufficiente energia alle alte frequenze per far sì che la pinna partecipi al processo percettivo.

Lateralizzazione ed esternalizzazione Uno degli scopi del sound rendering è quello di sintetizzare la localizzazione dei suoni manipolando gli indicatori binaurali (o *interaural cues*).

Un particolare fenomeno che si incontra in questo ambito è la *lateralization*, un particolare caso di localization in cui la percezione laterale avviene all'interno della testa, principalmente lungo l'asse interaurale. Quando si sta ascoltando un suono monaurale tramite cuffie stereo, l'ascoltatore percepisce una singola sorgente sonora, e la colloca all'interno della propria testa. Al crescere della *ITD* e della *ILD* si ha l'effetto percettivo di spostamento della sorgente verso uno dei due orecchi, in un fenomeno detto *Inside the Head Localization (IHL)*.
L'azione di mitigazione di queste alterazioni percettive si dice *externalization*, ed è l'obbiettivo che si ha quando si sviluppano sistemi di rendering di audio tridimensionale. Non sono ancora chiare quali siano le cause di questo comportamento, ma si è visto che arricchire il suono con riverberazioni è una strategia favorevole sotto questo punto di vista [7].

Percezione di distanza Tra tutte le misure percettive esposte in questo capitolo, la percezione di distanza è la più complessa poiché è un insieme di differenti parametri di cui non si conosce completamente la correlazione e gli effetti quando questi vengono combinati.
Un parametro rilevante per la stima della distanza è certamente l'intensità sonora, più propriamente nella sua declinazione percettiva *loudness* (si veda Sez. 1.2).
La *loudness* è più efficace in assenza di riverbero, se vi è riverbero infatti la *loudness* non differisce molto per sorgenti sonore vicine e distanti, rendendo difficile la valutazione dato che l'energia delle onde riflesse non decade come quella dell'onda diretta all'ascoltatore. Spesso si fa riferimento al rapporto *reflected to direct energy ratio R/D* in quanto riesce a dare una misura dell'indicatore introdotto con le riflessioni. È importante sottolineare come, in ambienti anecoici, rimanga difficile per l'ascoltatore medio stimare la distanza in assenza quasi totale di riverbero, si osserva quindi che un tale disturbo permette all'ascoltatore di sfruttare anche l'informazione di interferenza del riverbero per localizzare una sorgente.
In relazione a quanto esposto, il grado di familiarità con l'ascolto di un tipo di suono incide nell'abilità di localizzarlo, quindi entra in gioco il meccanismo derivante dall'esperienza di ascolto quotidiana, ed ovviamente la multimodalità uditivo-visiva gioca un ruolo fondamentale nella localizzazione a sua volta.
La condizione di *near-field* merita una considerazione separata rispetto a quella appena fatta in quanto non sono valide molte delle assunzioni di partenza, a partire dalla planarità delle onde.

Indicatori dinamici Molti esseri viventi, compresi gli umani, si servono del movimento della testa per localizzare una sorgente sonora sfruttando il feedback istantaneo in termini di variazioni di *ITD* e *ILD* provocate dal moto. Questo, oltre ad incrementare l'accuratezza, può favorire l'*externalization* nel caso l'ascoltatore possa muovere la testa e non abbia un ascolto in cuffia. Il movimento di rotazione della testa sul piano frontale (il cosiddetto *roll*) è di grande aiuto per localizzare con successo una fonte sonora, gli esseri viventi utilizzano movimenti come questi in continuazione cercando di percepire il

cambiamento del suono in relazione al movimento che si sta eseguendo.
In aggiunta al solo movimento della testa, spesso si sfrutta la valutazione dell'*acoustic* τ ovvero si sfrutta il moto fisico di avvicinamento e spostamento laterale rispetto alla sorgente fissa valutando le differenze degli indicatori, similmente a quanto accade muovendo solamente la testa.

1.4.3 Rendering del suono 3D

Il processo di *rendering sonoro* in tre dimensioni consiste nel sintetizzare un suono a partire da una rappresentazione informativa non spazializzata e da un modello che permetta di trasformare l'informazione in modo da essere fruita sotto forma di ascolto binaurale, eventualmente in cuffia. L'obbiettivo di chi sviluppa display uditivi 3D personalizzati è quello di creare sistemi per "sonificare" una scena, cioè creare l'ambientazione tridimensionale in cui i suoni si propagano.

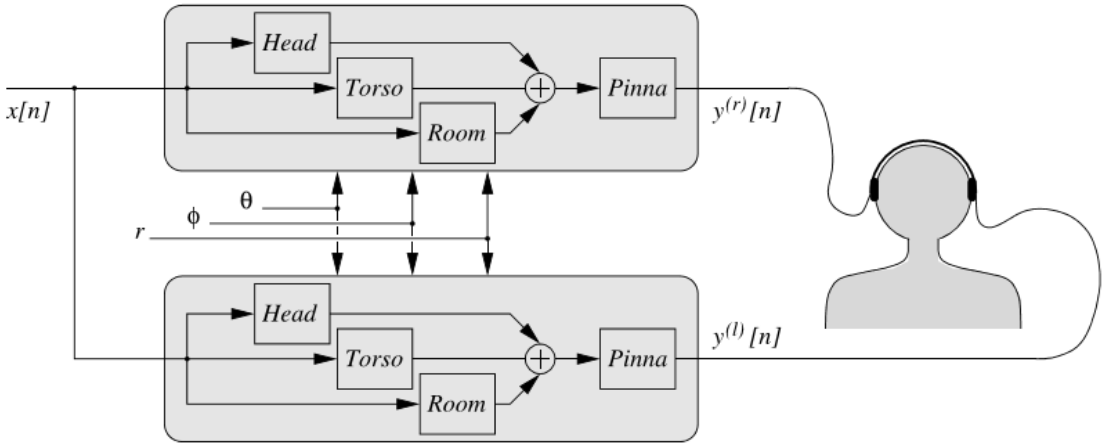


Figura 1.15: Schema a blocchi per un sistema di rendering tridimensionale del suono [7]

Da molti anni si studia come affrontare questo problema ed è stato stabilito che avere un modello accurato di HRTF è sufficiente per avere una resa tale da rendere le prestazioni in termini di localizzazione dell'ascoltatore comparabili a quelle che avrebbe in configurazione di *free-field* e perciò in condizioni reali [14].

Per quanto riguarda i nostri scopi è necessario ricavare una rappresentazione quanto più fedele possibile delle *HRTF* destra e sinistra per poter portare a termine, con una buona resa spaziale, un'elaborazione di individualizzazione.
Nelle sezioni precedenti si è osservato quanto complesso sia descrivere ed analizzare il processo di ascolto spazializzato per gli esseri umani, e considerando anche l'alto grado di soggettività dei parametri in gioco, ci si rende conto di quanto è complesso e costoso

ricavare una stima accurata delle *HRTF* utili ai nostri scopi.

Da ultimo non dobbiamo dimenticare l'aspetto hardware del processo, visto che non esistono soluzioni di tipo commerciali sufficienti mature, spesso si devono utilizzare cuffie che non agevolano il lavoro di rendering binaurale, poiché spesso l'ascoltatore medio non gradisce utilizzare cuffie la cui risposta in frequenza è piatta, le cuffie commercialmente disponibili arricchiscono il suono adottando scelte di progettazione che modificano la risposta in frequenza intrinseca per migliorare la resa *general purpose*. Questa condizione degrada in maniera importante il risultato di un trasformazione binaurale, arrivando anche ad inficiare la localizzazione uditiva che è cruciale in questo tipo di contenuti, si cerca perciò di contro-equalizzare questi effetti per raggiungere la *trasparentizzazione* delle cuffie [16].

Si può affermare quindi che l'obiettivo ultimo del *binaural rendering* è l'*adequacy*, cioè fornire esperienze sensoriali che non necessariamente sono accurate in termini assoluti ma sono accettabili ed adeguate nella misura in cui l'ascoltatore sente un suono e lo giudica come reale o verosimile [17].

1.4.4 *HRTF-based rendering* per display uditivi 3D personalizzati

Per sviluppare sistemi adatti al rendering tridimensionale è necessario fissare alcuni requisiti a partire da come il contenuto dovrà essere fruito, per i nostri scopi si assume che i contenuti debbano essere veicolati in cuffia, evitando così i problemi di posizionamento dell'ascoltatore rispetto a degli speakers ed eliminando completamente il problema del *crosstalk* ovvero il passaggio di informazione destinata ad un canale al canale opposto, inoltre si elimina il problema del riverbero dell'ambiente in cui riprodurre i suoni.
L'ascolto in cuffia presenta alcuni limiti (ciascuno dei quali è superabile), ad esempio l'eccessiva sensazione di vicinanza della sorgente, l'impossibilità di monitorare il movimento dell'ascoltatore (per cui è indispensabile eseguire *head tracking*), e la risposta in frequenza non piatta (impiegando cuffie che compensano opportunamente).

Il paradigma fondamentale del *HRTF-based rendering* e quello del nostro contesto applicativo è quello di utilizzare dati esistenti per ricavare una rappresentazione quanto più fedele possibile delle *HRTF* dell'ascoltatore perciò non si tratterà di estrarre *HRTF individuali*, compito molto complesso e costoso, bensì di *individualizzare HRTF* rispetto all'ascoltatore.

HRTF misurate L'idea generale in questo processo di rendering (vedi Fig. 1.15) è quella di usare l'informazione di *HRIR*/*HRTF* preregistrate. Data la coppia (θ, ϕ) e un segnale anecoico è possibile introdurre dei ritardi in modo da simulare l'azione della *ITD* oppure convolvere tale segnale con le funzioni di trasferimento destra e sinistra al fine di "localizzare" il segnale nel punto spaziale desiderato. L'obbiettivo generale è quello di avere delle *HRTF* ideali derivanti da un soggetto o in alternativa utilizzare misurazioni

di *HRTF generiche* che siano rappresentative di gran parte degli ascoltatori ottenute eseguendo la media delle misurazioni antropometriche su soggetti multipli, tuttavia acquisendo generalità si perdono le caratteristiche spettrali particolari. In alternativa è possibile affidarsi a misure eseguite su manichini (es. KEMAR) che riproducono le proprietà antropometriche generali del torso e della pinna, utilizzare queste misure è un trade-off che comporta una minore accuratezza che si riflette in una ridotta abilità di localizzazione dell'ascoltatore a cui sono destinati i contenuti, con il vantaggio di minimizzare il costo di ricerca di una individualizzazione più accurata.

Successivamente le *HRTF* misurate devono essere sottoposte a *post-processing*, come ad esempio equalizzazione e filtraggio, per lenire gli effetti di nonlinearità introdotte dai loudspeakers e nel processo di registrazione, per compensare limitazioni dovute agli strumenti hardware usati e limitare l'azione del canale uditivo del soggetto.
Una successiva fase prevede di eliminare la ridondanza nei dati delle funzioni di trasferimento individuando una *Common Transfer Function (CTF)*, la quale possiede tutte le informazioni comuni alle *HRTF* registrate. Ricavare la *CTF* permette di estrarre le *Directional Transfer Function (DTF)* sottraendola dalla *HRTF* di partenza e quindi vale

$$H^{(l),(r)}(r, \theta, \phi, \omega) = C(\omega) D^{(l),(r)}(\theta, \phi, \omega) \tag{1.8}$$

dove $C(\omega)$ è la trasformata comune *CTF* e $D^{(l),(r)}(\theta, \phi, \omega)$ è la la trasformata direzionale *DTF*.
La terza fase consiste nel produrre delle ricostruzioni di *minimum-phase filters* di ciascuna *HRTF*: l'ampiezza della risposta è la stessa mentre i poli e gli zeri sono tutti contenuti nella circonferenza unitaria, e sebbene rappresentino una semplificazione rispetto ai filtri originali, mantengono inalterate le caratteristiche salienti ed introducono il vantaggio di un minor costo computazionale [7].

Rilevazioni antropometriche Per realizzare l'individualizzazione delle *HRTF* si può fare affidamento a misure dirette sull'ascoltatore per estrarne caratteristiche antropometriche utili a selezionare un insieme di funzioni di trasferimento non-individuali da un database esistente (si veda Sez. 1.4.1).
È possibile sfruttare la *computer vision* per eseguire una misura antropometrica della pinna a partire da immagini 2D e a partire da un *modello di riflessione della pinna* e cercare un a corrispondenza delle caratteristiche per il modello.
Risultati sperimentali confrontano i risultati ottenuti su soggetti trattati con misurazioni antropometriche generali e gli stessi trattati utilizzando una stima del modello di riflessione, si dimostra che un tale approccio implica una diminuzione sensibile degli errori nella localizzazione verticale ed una riduzione della front-back confusion. [20]

HRTF sintetizzate L'uso di *HRTF* registrate richiede un carico computazionale elevato poiché sono necessari parametri per l'individualizzazione, inoltre si vogliono impiegare coordinate spaziali arbitrarie e sono generalmente matematicamente complesse

da utilizzare. Per questo motivo l'uso di *HRTF* sintetizzate è da preferirsi, dato che si possono approssimare i comportamenti più percettivamente rilevanti delle *HRTF* originali mantenendo il livello di computazione relativamente basso, un vantaggio che si nota maggiormente se è necessario eseguire rendering di ambienti complessi in termini di struttura riverberante e aventi sorgenti sonore multiple. Compiere questo tipo di analisi ci permette, come effetto complementare, di capire meglio quali sono le caratteristiche delle *HRTF* preminenti in ambito percettivo.
Vi sono due differenti strategie per produrre *HRTF* sintetizzate: i *pole-zero models* producono delle rappresentazioni sotto forma di filtri, mentre le *series expansions* cercano di fornire un'espressione di *HTRF* come somma di funzioni più basilari.

Interpolazione Questo processo cerca di mitigare la limitazione fondamentale derivante dalle registrazioni di *HRTF* eseguite su un insieme finito di posizioni (θ, ϕ). Nel caso sia necessario servirsi di una funzione di trasferimento per una coppia di angoli non presente nell'insieme delle rilevazioni disponibili non si hanno buoni risultati applicando ad esempio una semplice politica di ricerca del *nearest neighbor*, la quale introdurrebbe artefatti nel risultato finale. L'interpolazione permette di ricavare una stima delle funzioni di trasferimento per ogni posizione eseguendo una media pesata dei parametri che caratterizzano la quadrupla di posizioni note più vicine.

Modelli strutturali misti (MSM) Il paradigma a modelli strutturali misti affronta il problema del rendering da una prospettiva differente rispetto a quelle viste in precedenza.
La strategia alla base è quella di modellare separatamente gli effetti più importanti che permettono la localizzazione del suono in elementi filtranti, in questo modo si adotta una modellazione a blocchi in cui i filtri interagiscono tra loro, tuttavia la separazione tra le varie componenti non è netta e perciò si deve considerare un grado di approssimazione soprattutto nelle mutue influenze delle componenti.
È possibile costruire ogni singolo blocco in base a pure sintetizzazioni o in base a misurazioni reali e dunque operando così si ha un modello più flessibile, la modellazione si serve di due componenti principali: una *HRTF parziale (pHRTF)* che cattura caratteristiche misurate isolando le varie parti del corpo o ricavandole dalle misure esistenti tramite elaborazioni di segnale digitale, ed una *pHRTF sintetica* che cattura contributi di specifiche parti del corpo eseguendone la sintesi ed è generata artificialmente e con tecniche di simulazione. Il modella strutturale misto combina componenti di tipo individuale, componenti selezionate da misurazioni esistenti e componenti sintetizzate per produrre una approssimazione finale.

Il concetto chiave del modello *MSM* è la categorizzazione degli indicatori spaziali in base alla componente strutturale che li produce, in particolare [19]:

- I contributi di localizzazione azimutale e di distanza a tutte le frequenze sono associati alla testa

- Gli indicatori di elevazione ad alte frequenze sono associati alle pinne auricolari
- Gli indicatori di elevazione alle basse frequenze sono associati all'azione del torso e delle spalle

Il modello per rappresentare l'azione della testa include gli indicatori di azimuth e distanza già visti in precedenza: la ITD e la ILD. A questa analisi affianca uno studio più accurato dell'azione nella condizione di *near-field* oltre a quella già discussa di *far-field*. In secondo luogo, viene impiegato principalmente il modello semplificato a "testa sferica" in cui è possibile variare il parametro di diametro della sfera per meglio rappresentare le caratteristiche del soggetto in esame.

Il modello che rappresenta l'azione del torso e delle spalle introduce contributi generalmente meno marcati rispetto agli altri presenti nel sistema, tuttavia contribuiscono tramite perturbazioni ed altri indicatori deboli alla localizzazione della sorgente sonora. Alle basse frequenze il torso agisce da disturbo mascherando l'azione delle onde provenienti dal basso, analogamente le spalle introducono riflessioni per le onde provenienti dall'alto.

Il modello per i contributi della pinna contiene l'analisi dei *peak* e *notch* in frequenza, introducendo l'ulteriore analisi modale dell'azione della conca auricolare che introduce diffrazioni e risonanze. Inoltre viene introdotta un'analisi complementare della conformazione antropometrica dell'orecchio.

Ciascuna delle componenti appena descritte è costituita da un insieme di filtri e funzioni di trasferimento, e ciascuna contribuisce all'interno del modello strutturale misto, integrando così metodi di selezione e metodi di sintesi. Un aspetto importante che emerge di questo approccio è che modellando i singoli blocchi si introducono una serie di combinazioni con le quali è possibile determinare il grado di carico computazionale (e perciò di accuratezza della ricostruzione) in base ai requisiti richiesti per la particolare applicazione, ad esempio si possono ammettere carichi computazionali maggiori nell'elaborazione per videogiochi o carichi più leggeri su piattaforme meno performanti.
Le caratteristiche di una modellazione così eseguita la rendono una strategia promettente per l'applicazione a questo lavoro.

Capitolo 2

Il progetto Selfear

2.1 Il contesto tecnologico

Eseguire rilevazioni personali di *HRTF*, come già largamente anticipato, presenta alcune difficoltà e richiede un dispendio di risorse e tempo non indifferente. Per avere delle rilevazioni personali è necessario avere strumentazione costosa come ad esempio una camera anecoica ed è necessario eseguire rilevazioni difficili da portare a termine correttamente. Realizzare un sistema che permetta di eseguire una stima accurata delle *HRTF* in ambienti non controllati da parte dell'utente stesso è un compito arduo perché pone sfide sia sul piano tecnico (ad es. compensare la rumorosità dell'ambiente ed estrarre le caratteristiche salienti) sia sul piano dell'usabilità dell'interfaccia utente (ad e. minimizzare il tempo di "apprendimento" della procedura corretta).
Le tecniche illustrate in Sez. 1.4.4 sono lo stato dell'arte per quanto riguarda la stima mirata alla personalizzazione delle esperienze audio immersive, ciascuna però presenta limiti di maturità e/o di applicabilità a causa dell'alto livello di supervisione necessario per portare a termine le rilevazioni.

L'*augmented reality* è una commistione tra elementi reali ed elementi completamente sintetizzati dotati di una posizione nel mondo reale e visibili tramite opportuni dispositivi. Secondo la definizione di Azuma (1997) si parla di realtà aumentata quando si verificano tre condizioni:

1. vi è combinazione tra oggetti reali e virtuali
2. il sistema è reattivo e interattivo in tempo reale
3. il sistema è renderizzato nelle tre dimensioni

L'*audio augmented reality (AAR)* si diversifica rispetto alla *visual augmented reality* in alcuni tratti fondamentali: è omnidirezionale, veicola informazioni transitorie e non persistenti come le immagini ed è più difficile distogliere l'attenzione da un flusso informativo come si potrebbe fare ad esempio chiudendo gli occhi. L'omnidirezionalità in molti casi è un vantaggio consistente poiché permette di travalicare le limitazioni di campo visivo

e permette quindi di ideare interazioni anche indirette con l'ambiente virtuale.

Il mercato dei dispositivi adatti alla realtà aumentata è all'apice della curva di *hype* e sta per verificarsi la diffusione di massa di tecnologie di questo tipo, e sebbene dal lato di elaborazione e rendering video la tecnologia sia matura, non si può affermare lo stesso per l'ambito di rendering audio. È necessario perciò sviluppare sistemi che rimangano al passo con gli sviluppi tecnologici delle piattaforme, visto e considerato che le applicazioni *audio augmented reality* non solo sono complementari ai *visual augmented environments* ma possiedono addirittura maggiori applicazioni pratiche, basti pensare alle applicazioni in cui l'utente deve mantenere un controllo visivo della realtà mentre può essere assistito dall'audio in sottofondo nel portare a termine un task.
Nella definizione ideale di *mobile Audio Augmented Reality (mAAR)* l'ascoltatore deve poter disporre di cuffie trasparenti rispetto ai suoni esterni, ed in questo la trasparentizzazione delle cuffie trova la sua collocazione (si veda Sez. 1.4.3) analogamente all'utilizzo di un *see-through display* per la parte visiva, per far si che avvenga la fase di "sovrapposizione" delle realtà. Questa viene realizzata realizzando un'auralizzazione real-time dinamica e parametrica della scena, che è realizzabile soltanto conoscendo le *BRIR* specifiche dell'utente e per la scena per essere efficace. Il motore che realizza la $mAAR$ è il dispositivo mobile che l'utente porta con sé, e deve essere in grado di applicare i filtraggi e le trasformazioni DSP per compensare le non idealità introdotte dagli headset.

I mezzi tecnologici con cui si fruisce di esperienze di realtà virtuale e aumentata possono essere personal computer, console e smartphone. Questi ultimi sono protagonisti di grossi progressi tecnologici e di diffusione e vedono l'aumento delle soluzioni destinate ad essere fruite su tali piattaforme. I maggiori produttori di hardware vendono attualmente soluzioni per la realtà aumentata su smartphone poiché questi strumenti sono portabili ed adeguati ad essere montati su particolari visori, creando così un mercato di dispositivi che si rifanno ai *virtual reality goggles* al costo irrisorio di sviluppare gli alloggiamenti per smartphone.

È in questo contesto che questo progetto intende agire per creare un sistema che sfrutti il lavoro precedente in ambito di selezione *HRTF*, tramite un nuovo sistema di rilevazione che pone al centro dell'azione l'utente finale e servendosi di database di misurazioni si vuole realizzare una personalizzazione dell'esperienza uditiva che sia ottimizzata e quanto più accurata possibile.

È originata così l'idea di produrre un'applicazione mobile per il sistema operativo Android che permetta di eseguire le rilevazioni, il nome "Selfear" racchiude in sé lo spirito a cui punta il progetto, quello di creare un "selfie dell'orecchio" dell'ascoltatore guidandolo nel processo di registrazione delle proprie *Pinna-Related Transfer Functions (PRTF)* all'interno di ambienti non anecoici tramite una procedura *self-adjusting*.

2.2 Descrizione

Il progetto Selfear affronta il problema di ridurre al minimo il costo economico e di risorse nella rilevazione di *HRTF* individuali, focalizzandosi sull'estrazione di *Pinna-related Transfer Functions (PRTF)* destinate all'uso su sistemi *mAAR*.
Realizzare una soluzione di questo tipo pone due ostacoli principali: in primo luogo la registrazione eseguita in ambiente non anecoico introduce pesanti effetti di *frequency coloration* e *phase shifts* nelle onde registrate ed è necessario compensare questi effetti eseguendo filtraggi, in seconda battuta si deve tenere conto degli strumenti hardware coinvolti, i quali non sono di qualità paragonabile alla strumentazione di uno studio.

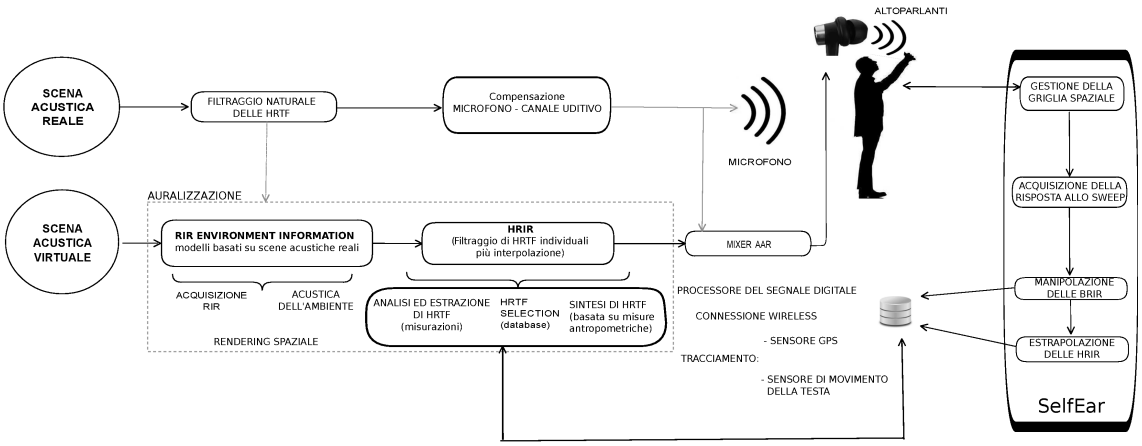


Figura 2.1: Schematizzazione del funzionamento di Selfear, all'interno di un sistema di *mobile augmented audio reality (mAAR)*

Il funzionamento dell'applicazione è illustrato in figura 2.1. Al fine di ricavare il contributo testa-busto-spalle si prevede che il dispositivo costruisca una griglia virtuale varie angolazioni rispetto alla testa dell'ascoltatore, a ciascuna delle quali viene riprodotto un impulso tramite gli altoparlanti del dispositivo. Il segnale impulsivo, trasformato dalla riverberazione della stanza e dalle caratteristiche dell'orecchio dell'ascoltatore viene registrato in modalità stereo (canale destro e sinistro) tramite una scheda di acquisizione e delle cuffie stereo in-ear che possiedono dei microfoni in grado di registrare suoni nella direzione esterna al condotto uditivo. Si ottiene così una stima delle *raw BRIR* ovvero funzioni di trasferimento *BRTF* grezze in quanto risentono delle non idealità tipiche del contesto. Una fase di post-processing esegue delle elaborazioni di segnale digitale che realizzano le compensazioni per le non idealità dei microfoni, degli altoparlanti e che mitigano le alterazioni introdotte dalla necessità di indossare le cuffie in-ear.
Le rilevazioni devono essere fatte in modo tale da ricalcare la struttura delle misurazioni eseguite nel database di riferimento nel quale si vuole eseguire il lookup alla ricerca di funzioni di trasferimento simili a quelle dell'utente.

2.3 Funzionamento

La procedura *self adjusting* dell'applicazione prevede che l'ascoltatore debba tenere lo smartphone in mano, a braccio completamente esteso e debba posizionarlo in punti ben definiti rispetto alla postura naturale della testa, avendo cura di non muoverla. Il segnale impulsivo impiegato che viene riprodotto dal dispositivo è uno *sweep* audio, cioè un breve segnale sonoro che contiene un suono che varia la propria frequenza, spaziando nello spettro dell'udibile ovvero 20 Hz - 20 kHz. Questo permette di avere una rappresentazione dell'intero spettro umanamente udibile e perciò è possibile ricavare il massimo dell'informazione disponibile dalla registrazione.

La qualità della riproduzione è influenzata dalla qualità degli altoparlanti e soprattutto dalla posizione degli stessi sul dispositivo, è importante conoscere questa caratteristica per poter mitigare le alterazioni causate dalle diverse posizioni di emissione adottando strategie opportune, quali ad esempio indicare all'utente di tenere il dispositivo con l'altoparlante rivolto verso di sé.

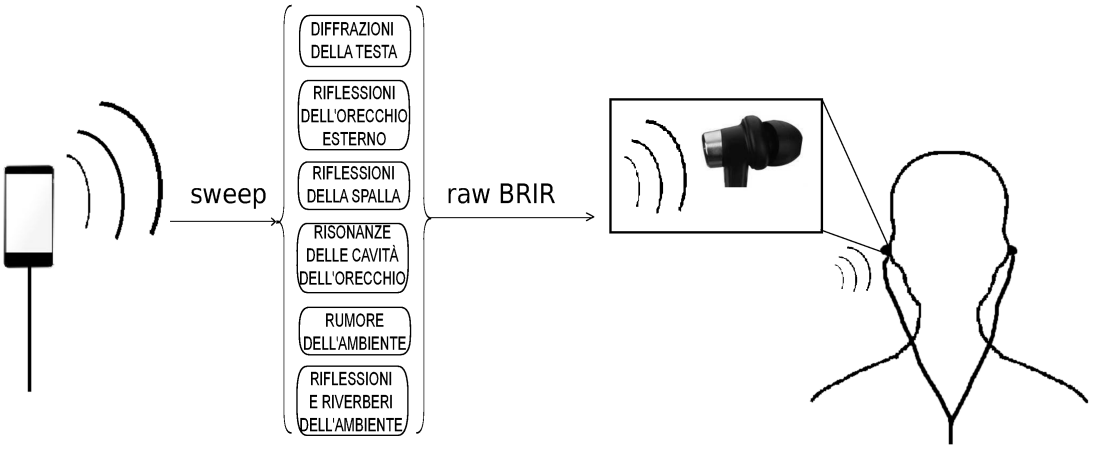


Figura 2.2: Il percorso del segnale dall'istante di riproduzione all'istante di registrazione stereo

La versione 1.0 di Selfear stabilisce una serie di angoli di elevazione prestabiliti, nello specifico con un passo di 5,625° in elevazione ϕ in un intervallo [-40°,+40°] e mantenendo un azimuth θ costante a 0° quindi una scala di posizioni giacenti sul piano mid-sagittale, come stabilito nel protocollo di misurazione del database di riferimento CIPIC (si veda Sez. 1.4.1).
È altresì possibile impostare un insieme di angoli obbiettivo differenti, nel caso si decida di effettuare rilevazioni secondo un diverso *framework*.

Il dispositivo non è in grado di acquisire autonomamente le registrazioni dei microfoni stereo che l'utilizzatore indossa durante il processo di acquisizione, tali auricolari stereo

sono connessi ad una opportuna interfaccia di acquisizione stereo collegata ad un PC, il quale possiede un software appositamente progettato per l'acquisizione e la successiva elaborazione dei segnali.
Il dialogo tra smartphone e PC avviene tramite l'utilizzo del protocollo OSC[1] per l'invio dei comandi dall'interfaccia utente verso il sistema di registrazione. Questo è possibile solamente dopo aver accoppiato i due sistemi, in questo caso connettendo entrambi i dispositivi alla stessa rete e specificando nell'applicazione l'indirizzo IP del PC a cui indirizzare i comandi.

Le fasi di utilizzo dell'applicazione sono:

1. *Raggiungimento del target spaziale* Il controllo del posizionamento del dispositivo nella griglia spaziale fa affidamento al sensore di accelerazione per ricavare l'angolo di inclinazione assunto dal dispositivo stesso rispetto alla posizione verticale.
   Un segnale *beep* è di ausilio all'utente per aiutarlo a posizionare il dispositivo nell'angolo obbiettivo stabilito, la riduzione tra gli intervalli del suono segnala il grado di avvicinamento all'obbiettivo, agevolando la procedura di puntamento, considerando che in molti casi l'utente non può direttamente osservare lo schermo dello smartphone.

2. *Controllo della posizione* Una volta raggiunta una quota angolare vicina all'obbiettivo (ad es. in un range di 3°), scatta una procedura che verifica la stabilità della posizione per alcuni secondi per prepararsi ad emettere il suono. Nel caso si esca dal campo di stabilità la procedura fallisce ed è necessario ripetere il puntamento. È possibile inoltre impostare un grado di sensibilità per lo scostamento dall'angolo ideale, facilitando in questo modo l'utilizzo a discapito di una possibile misurazione meno accurata.

3. *Riproduzione dello sweep* Se la procedura di verifica di stabilità ha successo, viene riprodotto lo *sweep* dagli altoparlanti del dispositivo e contestualmente viene mandato un comando di registrazione al PC associato.

4. *Registrazione delle HRIR* Al termine dello sweep, un comando OSC indirizzato al PC associato invia il segnale di successo, e vengono salvati su file i dati della rilevazione. Se per un qualsiasi motivo la procedura fallisce, viene inviato un comando di registrazione non valida.
   Si prosegue ripetendo la procedura per tutti gli angoli dell'insieme degli obbiettivi stabiliti.

5. *Fine della sessione* Al termine di tutte le acquisizioni la procedura termina e si possiedono tutti i dati per proseguire il processo.

La natura sperimentale dell'applicazione la pone di fonte ad una eterogeneità di problemi realizzativi, nei confronti dei quali sono state adattate soluzioni transitorie e

[1] http://opensoundcontrol.org/

approssimazioni.
L'interfaccia utente deve essere chiara e presentare in modo ottimale l'informazione di rilievo ovvero l'angolazione obbiettivo e lo scostamento da essa, inoltre è necessario adattare il funzionamento in base a come si suppone il dispositivo debba essere tenuto nella mano: il fatto che diversi dispositivi possiedano altoparlanti sul fondo o sul retro impone di effettuare le rilevazioni mantenendo l'altoparlante rivolto verso l'utente precludendo la possibilità di osservare lo schermo, questa è un'eventualità che si verifica anche per angolazioni agli estremi del range impostato ed ha reso indispensabile l'utilizzo di un feedback sonoro.
Dal punto di vista della robustezza delle rilevazioni, facendo affidamento solamente sulla stima dell'inclinazione del dispositivo è necessario che l'utente posizioni correttamente il telefono nella propria mano evitando di inclinarlo dalla sua posizione ideale mentre lo maneggia. Questo aspetto incide sulla capacità dell'utilizzatore di effettuare misure efficaci in tempi brevi poiché probabilmente è necessario un breve addestramento per riuscire a portare a termine la procedura correttamente.

2.4 Risultati raggiunti

L'obbiettivo di produrre un articolo scientifico sul progetto ha spinto gli autori ad eseguire misurazioni in ambiente controllato e confrontando le rilevazioni fatte dall'applicazione con i risultati di letteratura. [23]
Sono stati opportunamente misurati i contributi di disturbo in *diffuse field* delle componenti usate in camera silente, quali ad esempio l'altoparlante che riproduceva lo *sweep*, per essere poi considerati in fase di analisi delle registrazioni.

Dalle rilevazioni sperimentali si evince che il *framework* adottato conserva le caratteristiche osservate in letteratura e perciò si dimostra la possibilità di ottenere una stima corretta delle *PRTF* e l'estrazione dei caratteristici *notch* e *peak* della risposta in frequenza (vedi Fig. 2.3).
Dati questi parametri è possibile stimare la *central frequency*, il *gain* e la *banda* di tali risposte e tali informazioni sono sfruttabili per essere applicate in modelli di sintesi delle *PRTF* e/o strategie di selezione delle *HRTF* secondo modelli strutturali misti. È possibile inoltre usare direttamente le misurazioni per effettuare un rendering.
È opportuno sottolineare quanto sia importante l'analisi delle non-idealità (ad es. riverbero, noise cancellation e movimenti della testa) e l'impiego di pre e post processing per compensare adeguatamente le registrazioni.
Una naturale evoluzione del sistema è quella di implementare un algoritmo per la head pose estimation sfruttando la camera del dispositivo per tracciare la posizione della testa, in secondo luogo è desiderabile realizzare anche l'acquisizione delle *BRIR* eseguendo in step separati misurazione delle *RIR* e delle *HRIR* al fine di realizzare un rendering accurato della scena audio 3D.

Complessivamente, considerando la qualità e la variabilità dei risultati e considerando il minimo impatto in termini di utilizzo di risorse, questo sistema si è rivelato essere

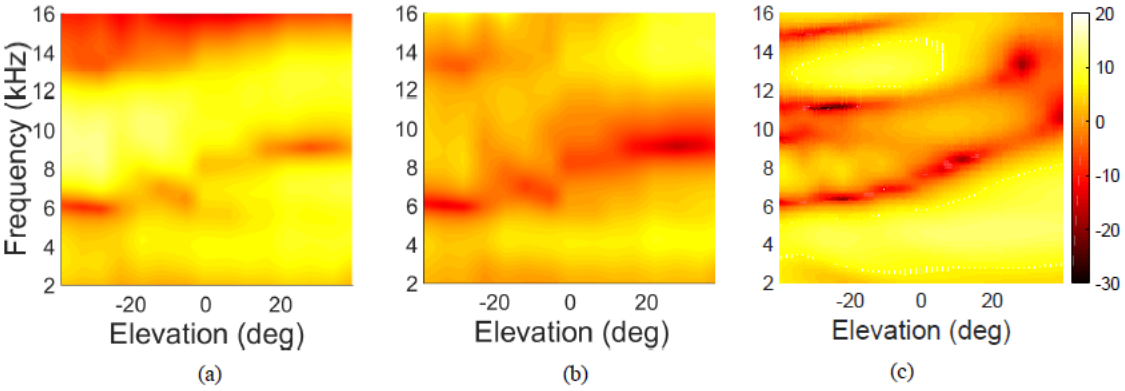


Figura 2.3: PRTF nel piano mediano, misurate utilizzando Selfear 1.0, in *(a)* l'acquisizione originale, in *(b)* le misurazione con compensazione *diffuse-field*, in *(c)* misurazioni eseguite sul manichino KEMAR, con compensazione *free-field*

una buona strategia low-cost per ottenere modelli delle *PRTF*.

Le validazioni sono state eseguite in ambiente controllato, sono necessarie procedure di pre-processing e post-processing per compensare una eventuale rilevazione da eseguirsi in un ambiente qualsiasi.
Le rilevazioni risentono anche del movimento involontario della testa dell'ascoltatore rispetto alla posizione ideale, è necessario usare procedure di computer vision per migliorare questo aspetto.
Per poter realizzare un rendering completo in real-time è necessario implementare una procedura per ricavare le *RIR* e le *HRIR* in modo da avere una rappresentazione completa delle *BRIR*.

Capitolo 3

Selfear 2.0

Selfear 2.0 rappresenta il passo successivo del lavoro già svolto al Centro di Sonologia Computazionale (CSC) dell'Università degli Studi di Padova in Selfear 1.0 [22] per lo sviluppo di un'applicazione Android per l'acquisizione low-cost di profili acustici individuali attraverso l'individualizzazione delle *HRTF*.

L'evoluzione del lavoro svolto precedentemente richiede l'introduzione di algoritmi di visione artificiale per realizzare una rilevazione spaziale anche su posizioni non appartenenti al piano mid-sagittale per ricavare un modello più accurato e soprattutto per poter cogliere nella sua completezza il fenomeno dell'ascolto spaziale del suono.
Per sviluppare un algoritmo adeguato ai nostri scopi sono state esplorate le tecniche di rilevazione facciale ed algoritmi per estrarne le caratteristiche, a partire dalle quali si è ricavato una mappatura su un modello tridimensionale in modo tale da poter calcolare con accuratezza la direzione della testa rispetto alla camera. Questo insieme eterogeneo di problemi da risolvere ha comportato un grande dispendio di tempo per capire le migliori strategie adottabili per accuratezza ed efficienza, considerando anche le limitate capacità computazionali di cui si dispone.

3.1 Aspetti tecnici

3.1.1 OpenCV

La libreria OpenCV[1] è un prodotto open source[2] per lo sviluppo di software di computer vision, scritta in linguaggio C e C++ e disponibile per tutte le maggiori piattaforme commerciali.
L'obbiettivo del progetto è permettere lo sviluppo di sistemi che sfruttano la computer vision complessi in maniera semplice, possiede molte applicazioni data la grande disponibilità di risorse e funzioni. A completare il pacchetto vi sono diverse funzioni *general purpose* per il *machine learning*, dato che queste discipline sono in stretta relazione tra

[1] http://opencv.org
[2] http://opensorce.org

loro.
La libreria è distribuita con licenza BSD[3] la quale permette lo sviluppo di software commerciale senza richiedere compensi od altre forme di royalty, incoraggiando la proliferazione di software commerciale e non, grazie al supporto di grandi aziende di tecnologia e software e della comunità open source.

La disciplina della computer vision si pone l'ambizioso obbiettivo di estrarre informazioni di tipo qualitativo, quantitativo e semantico a partire da rappresentazioni statiche o dinamiche 2D e 3D di immagini e corpi. È chiaro il motivo per cui OpenCV ospita moduli per machine learning, altrimenti sarebbe impossibile inferire informazioni semantiche a partire dalle immagini, e questa capacità permette lo sviluppo di sistemi intelligenti in ambito industriale e in generale per la robotica autonoma.

3.1.2 Algoritmi per il riconoscimento facciale

Rilevare le facce umane a partire dalle immagini è da sempre un argomento di interesse per la computer vision, con applicazioni in ambito di sicurezza e sorveglianza, ma anche e soprattutto per sviluppare metodi per la *Human Computer Interaction (HCI)* cioè modi con cui l'utente può interagire con un sistema differenti dai tradizionali (mouse, tastiera, ecc.). È possibile infatti ricavare informazioni biometriche e sulla fisiologia dei movimenti dell'utente per costruire nuovi canali di interfacciamento con le macchine.
Grandi sforzi sono stati messi in atto in questo campo, in particolare negli ultimi anni, e la sempre crescente capacità computazionale a disposizione ha incoraggiato lo sviluppo di software commerciali adatti ad essere eseguiti su dispositivi quali smartphone e tablet. Considerando il fatto che tali dispositivi acquisiscono un peso importante nelle attività quotidiane e dispongono di sensori che i normali PC non possiedono si può immaginare che questo campo abbia grandi potenzialità dal punto di vista commerciale, anche per sistemi di *mAAR*.

Face detection

Nell'ambito della *face detection* si possono identificare due tecniche principali, la *face detection* cioè la localizzazione delle regioni di interesse con le facce e la *facial feature detection* cioè la rilevazione delle caratteristiche facciali quali la posizione della bocca, del naso, degli occhi, ecc.
Il problema di *face detection* [25] è formalizzato in questo modo: data un'immagine arbitraria, l'obbiettivo della face detection è quello di determinare la presenza di facce umane nell'immagine ed eventualmente individuare i punti e le regioni dove esse sono presenti.
Gli ostacoli principali quando si affronta questo problema sono:

- *Posa*: la posa della testa del soggetto è estremamente variabile

[3] https://it.wikipedia.org/wiki/Berkeley_Software_Distribution

- *Componenti strutturali*: barba, capelli, occhiali assumono forme e colori molto differenti
- *Espressioni facciali*: introducono variazioni di fisionomia della faccia, cambiandone la forma
- *Occlusioni*: movimenti ed oggetti nascondono parti del viso
- *Orientazione dell'immagine*: le facce pur conservando le caratteristiche possono essere difficili da riconoscere se l'immagine è ruotata
- *Condizioni ambientali*: illuminazione, colore dell'illuminazione e parametri fisici della lente della camera cambiano l'aspetto di una faccia in un'immagine

Il problema della *face detection* può essere espresso in prima istanza come un problema di classificazione di regioni di un'immagine in cui si classificano regioni "facciali" e regioni "non-facciali" ed è uno dei primi esempi in cui a causa della larga *within-class variability* è stato necessario sviluppare algoritmi che estraggono informazioni da immagini reali e non da rappresentazioni astratte. Un secondo aspetto degno di nota è stata la necessità di creare algoritmi in grado di gestire un grande *feature space* in cui applicare dei criteri di decisione estremamente non-lineari in maniera efficiente, data la grande mole di informazione da processare.

La rilevazione facciale a partire da singole immagini è l'esempio più semplice di *face detection* ed è una tecnica che è stata oggetto di larga ricerca nel corso degli anni. Possiamo identificare quattro tipologie di risoluzione in questa categoria di problemi [24]:

Metodi knowledge-based Sono procedure che cercano una codifica in regole di ciò che la conoscenza umana usa per determinare le caratteristiche che costituiscono una faccia. La difficoltà in questo caso è rappresentata dal definire un insieme di regole che catturino accuratamente la conoscenza delle strutture facciali in presenza di una larga variabilità, perciò è necessario definire regole né troppo stringenti né troppo generali per poter riuscire a rilevare con successo una faccia.
Alcune strategie adottano una politica *top-down* per ricavare informazione a diversi gradi di dettaglio in maniera progressiva per ridurre il carico computazionale e migliorare la precisione.

Metodi feature-based Al contrario della precedente, questa strategia realizza un'analisi *bottom-up* realizzata sulla base di caratteristiche che si mantengono invariate alla variabilità di posa e illuminazione ambientale.
In alcuni casi si incrociano tecniche di rilevazione degli *edge* nell'immagine con euristiche utili a rimuovere il rumore introdotto dalle occlusioni e ombre per poi mettere in relazione le aree rilevate alla ricerca di *invariant features*, in molti casi servendosi di un pattern di riferimento (ad es. costruiti su modelli probabilistici o su grafi di distanze

mutue) su cui eseguire confronti delle aree candidate ad essere una faccia. Altre strategie cercano di ricavare informazione dal colore dell'immagine confrontandolo con particolari *texture* o cercando le *skin-region*. Non è raro trovare strategie che combinano le due precedenti per eseguire un'ipotesi iniziale della posizione della faccia e successivamente eseguire confronti più dettagliati.

Template-matching Dispone di un template di una faccia, tipicamente in posa frontale in cui viene misurata la correlazione tra il riferimento ed i punti di interesse quali ad esempio il contorno facciale, bocca, occhi ecc.
Un approccio possibile è quello con template predefiniti oppure deformabili per i punti di interesse costituiti da semplici linee da confrontare con gli *edge* dell'immagine, in una strategia adottata anche più in generale di *focus of attention and subtemplates*. È altresì possibile cercare correlazioni tra feature space rappresentati da regioni dell'immagine e che sono posti in relazione di adiacenza, con il vantaggio di avere risultati più robusti alle variazioni di illuminazione.

Metodi appearance-based Sono in generale i metodi più complessi in quanto sfruttano *machine learning* e analisi statistica a partire da un insieme di esempi su cui eseguire *training*. Osservando il problema da un punto di vista probabilistico, a partire da un'immagine oppure un *feature vector* visto come una variabile aleatoria $\mathbf{x}$, l'obbiettivo è costruire una distribuzione di probabilità condizionata $P(\mathbf{x}|face)$ e $P(\mathbf{x}|nonface)$ ma a causa della grande dimensionalità spesso si cercano alcune euristiche per stimare le probabilità. Un'altra strategia è quella di costruire *discriminant functions* per eseguire classificazioni, introducendo così il *machine learning* nel processo. In questo ambito i più importanti approcci sono: *Eigenfaces*, *Metodi distribution-based*, *Neural Networks*, *Suport Vector Machines*, *Sparse Network of Winnows*, *Naive Bayes Classifiers*, *Hidden Markov Models*, *Information-Theoretical Approach* e *Inductive Learning*. [24]

Face recognition

L'obbiettivo della *face recognition* è l'identificazione di una faccia in un'immagine cercando un riscontro su un database di facce. Questo problema è in genere più complesso del precedente per la difficoltà nell'identificare l'insieme di caratteristiche univoche della faccia di un individuo, con i requisiti di essere non intrusivo e di non richiedere particolari attività da parte del soggetto (categoria della *passive biometric recognition*).
Le possibili applicazioni di questo approccio spaziano dalla *face authentication* cioè verificare l'identità di individui, *face tracking* in cui si cerca di stimare la posizione della faccia in tempo reale e la *facial expression recognition* per identificare emozioni a partire da caratteristiche facciali.
Il problema si formalizza precisamente come *Pose Invariant Face Recognition (PIFR)* che definisce il problema di ricavare informazioni salienti di una faccia umana in condizioni di posa variabile essendo questo lo scenario che rende realmente utile questa categoria di algoritmi. In questo caso le difficoltà maggiori da affrontare sono: [25]

- *Self-occlusion* È il fenomeno per cui una testa in posizione ruotata occlude parte di sé stessa alla visione

- *Perdita di informazione* Nella rotazione di una posa non vengono mantenute linearmente le relazioni tra le caratteristiche facciali

- *Carenza di informazione* Possedere immagini a bassa risoluzione o facce particolarmente in lontananza nelle immagini è una carenza di informazioni poiché molte delle caratteristiche facciali salienti sono definite da piccole regioni di pixel

La *face recognition* pur nella sua complessità sta vedendo negli ultimi anni un grande sviluppo e di anno in anno vengono proposte soluzioni sempre più sofisticate e performanti che riescono a risolvere con successo questa tipologia di problemi. Le principali strategie sono [25]:

Pose-robust feature extraction Si pone l'obbiettivo di ricavare rappresentazioni facciali che siano robuste al cambiamento di posa e comunque discriminative per quanto riguarda il riconoscimento facciale.
Le *engineered features* sono tecniche che cercano di ricostruire la relazione la corrispondenza semantica nell'estrazione di *feature* facciali tramite la rilevazione di *landmark facciali* od in alternativa tramite l'estrazione e il confronto di *feature vector* dell'immagine.
Le *learning-based features* sono in genere più complesse delle precedenti, che risultano essere efficaci sono in casi di moderate variazioni della posa. Vengono impiegate tecniche di machine learning per superare i problemi di *self-occlusion* e di alta variabilità della posa, e per modellare la non-linearità vengono impiegate le *deep neural network* in varie forme e di varia profondità, oppure vengono eseguiti confronti multipli con template.

Multi-view subspace learning Si serve di un insieme di immagini dello stesso soggetto in varie pose cercando di ricavare le relazioni che si mantengono nei cambiamenti di posa. Si possono trovare implementazioni che costruiscono modelli lineari oppure non-lineari, con lo svantaggio principale della grande mole di dati di partenza necessari per il training rendendo questa strategia non adatta a soluzioni reali.

Sintesi facciale con modelli 2D Cerca di risolvere il *matching* tra facce con pose differenti tentando di trasformarle affinché assumano una stessa posa.
Le diverse strategie si occupano di trasformare le immagini in base a strutture *mesh* costruite a partire dai landmark, dividendo l'immagine in regioni più piccole risolvendo sotto-problemi di ottimizzazione oppure trasformando i singoli pixel secondo formulazioni globali. In questo modo si mantengono gran parte delle relazioni semantiche in gioco richiedendo un training molto limitato, tuttavia non si hanno risultati robusti a grandi variazioni di posa.
Altri approcci di *linear* e *non-linear regression* sono più robusti in questo senso ma

richiedono training più intensivi e dataset di grandi dimensioni, i risultati possono comunque soffrire di degradazione di qualità dell'immagine e perciò perdita di informazioni rilevanti per il riconoscimento facciale.

Sintesi facciale con modelli 3D È un task molto complesso che ha l'obbiettivo di ottenere una rappresentazione tridimensionale di un soggetto a partire da un insieme di immagini 2D, la variazione di illuminazione e la qualità del materiale di partenza influiscono sul risultato finale che in molti casi può perdere le particolarità necessarie per il riconoscimento.

3.1.3 Algoritmi per la head pose estimation

Head pose estimation è il termine che definisce il processo di inferire la direzione ed orientazione di una testa umana a partire da immagini, negli umani è una capacità innata mentre per le macchine eseguire questo tipo di valutazione è un task estremamente complesso. Rispetto a face detection e face recognition esistono relativamente pochi metodi rigorosamente validati per risolvere questo tipo di problemi, pur essendo una tecnica che fa affidamento su diversi aspetti di entrambe.
Il range di variazione di orientazione della testa di un umano è mediamente rappresentata da una estensione e flessione sagittale in $[-60.4°, 69.6°]$, una rotazione del collo dalla posizione frontale in $[-40.9°, 36.3°]$ e una rotazione assiale orizzontale in $[-79.8°, 75.3°]$ [26].
Con queste premesse, la head pose estimation è spesso valutata su uno spazio di 3 gradi di libertà *three degrees of freedom (3DOF)* che sono identificati come *yaw*, *pitch* e *roll* in cui la testa viene trattata come un oggetto rigido.

Derivare l'orientamento della testa è di aiuto per ricavare la cosiddetta *gaze direction* ovvero la direzione e il punto di attenzione (focus) degli occhi di una persona essendo questa una combinazione della direzione degli occhi e della direzione della testa. Questo è uno degli strumenti con cui una macchina riesce ad interagire con gli umani, si evince perciò quanto sia di interesse compiere studi in questo ambito.
Molteplici soluzioni sono state proposte, impiegando diverse tecnologie come ad esempio immagini stereo, immagini di profondità oppure singole immagini su cui identificare landmark facciali. Le più importanti sono [26]:

Metodi appearance-template Confronta l'immagine in esame con un insieme di immagini di riferimento di cui è conosciuta la posizione. I vantaggi derivano dall'assenza di un training pregresso e dalla semplicità nella creazione di un insieme di immagini di riferimento, così come dalla robustezza dell'analisi di immagini a bassa risoluzione. Gli svantaggi sono da identificare nella scarsa flessibilità del modello, il quale riesce solamente a rilevare un insieme discreto di pose, in secondo luogo rilevare l'esatta posizione della testa nell'immagine può influenzare l'accuratezza e non ultimo, l'effetto delle particolari

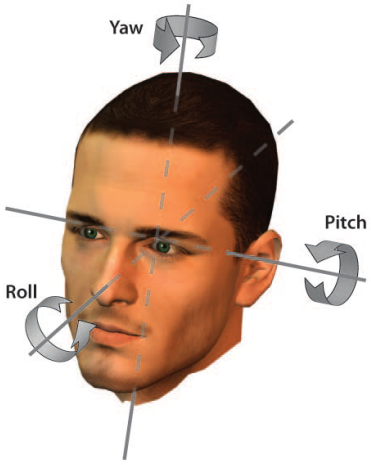


Figura 3.1: Rappresentazione grafica dei termini di *yaw (angolo azimutale), pitch (angolo di elevazione), roll (angolo di rotazione)* per il problema di *three degrees of freedom (3DOF)* tipicamente definiti in problemi di *head pose estimation* [26]

fisionomie può influenzare in maniera decisiva l'accuratezza, rendendo necessario un filtraggio per rimuovere le particolarità e mantenere intatti i tratti distintivi per la posa.

Metodi detector array Utilizzano in parallelo una serie di classificatori discreti oppure continui, tipicamente costruiti con *support vector machines (SVM)* ottenute tramite *supervised learning* con il vantaggio dato dal fatto che integrano la capacità di localizzare le facce nell'immagine. Gli svantaggi derivano dalla necessità di eseguire *training* estesi su un grande insieme di immagini su cui viene fatta una annotazione manuale della posa, in molti casi inoltre i classificatori non sono continui e perciò si richiede un training di molteplici classificatori binari.

Metodi con non-linear regression Tipicamente sono realizzati con strumenti avanzati quali le reti neurali che sono in grado di mappare trasformazioni non lineari, il vantaggio principale in questo caso è la possibilità di sfruttare la *backpropagation* che aggiorna dinamicamente la rete neurale, inoltre è possibile separare l'analisi di ciascun grado di libertà con una rete separata permettendo in generale di raggiungere risultati molto accurati, al netto di una localizzazione accurata della testa che rimane la limitazione di questa scelta.

Metodi manifold embedding Sfrutta una struttura di tipo varietà geometrica su cui mappare le posizioni, solo dopo aver eseguito una opportuna *dimensionality reduction* ad esempio con *Principal Component Analysis (PCA)*. Gli svantaggi di questo approccio sono dati dall'influenza dei tratti individuali e dall'illuminazione sul training, e dalla difficoltà di ottenere un dataset adatto a questo task.

Metodi a modelli flessibili A differenza dei precedenti, questo approccio introduce il concetto di modelli flessibili come strutture non-rigide che si conformano sulla faccia dell'individuo. Nel caso discusso in precedenza del *appearance template* si cerca di registrare la faccia in esame con un insieme di facce di riferimento per ricavare la posa che maggiormente corrispondente, ma le peculiarità facciali influiscono sull'accuratezza dell'algoritmo. I modelli deformabili costruiti come un grafo i cui nodi sono landmark facciali superano questa limitazione variando la propria forma in base alla faccia in esame. È necessario eseguire un training con immagini opportunamente annotate ed eventualmente creando un insieme di descrittori (*bunch* di descrittori) a ciascun nodo.
Un *Elastic Bunch Graph* così costruito si deforma iterando le deformazioni della struttura deformabile sula faccia in esame fino ad arrivare alla minima distanza di ciascun nodo del grafo rispetto alla posizione del landmark facciale. Così facendo è possibile eseguire un confronto tra il modello deformabile ed un insieme discreto di pose al fine di cercare la massima similarità, aumentando la robustezza ma allo stesso tempo preservando lo svantaggio di dover produrre un grande insieme di pose discrete di riferimento.
Un approccio differente chiamato *Active Appearance Model (AAM)* a partire da un insieme di immagini di training opportunamente annotate estrae un insieme di *feature* che identificano le differenti pose, tramite *dimensionality reduction* si ottiene un modello detto *Active Shape Model (ASM)* che rappresenta le componenti primarie del cambiamento di posa. Il *fitting* della ASM viene eseguito iterativamente comparando l'immagine osservata e quella sintetizzata cercando di minimizzare la differenza di posizionamento dei landmark di riferimento, infine cercando un mappatura tra le posizioni trovate e una stima della posa. Questo metodo è molto robusto alle variazioni di posa in virtù della propria capacità di variare in base alla forma in esame, con lo svantaggio di richiedere che entrambi i punti facciali corrispondenti agli occhi siano visibili per poter eseguire la stima.

Metodi geometrici Sfruttano la forma della testa e la precisa localizzazione di landmark facciali per stimare la posa, assumendo ad esempio una misura media tra le distanze delle varie componenti e ricavando una stima dell'angolazione del naso oppure ancora per l'angolo azimutale ricavare la differenza dimensionale tra i punti degli angoli oculari, per il *roll* ricavando la posizione degli occhi rispetto all'orizzonte e per l'angolo di elevazione misurando la distanza tra punta del naso e retta interoculare secondo misure antropometriche prestabilite.
Il vantaggio di queste strategie è dovuto alla loro semplicità, a partire da alcuni landmark facciali è possibile ricavare una buona stima, tuttavia non sempre rilevazioni accurate dei landmark sono possibili, ad esempio in condizioni di scarsa qualità dell'immagine o in condizioni *far-field*.

Metodi di tracking Fanno affidamento sul tracciamento del movimento della testa nel tempo, al fine di incrementare la convergenza degli algoritmi. Per stimare la posa è possibile inoltre eseguire valutazioni su tutte le possibili rotazioni che la testa tracciata può compiere e selezionare il modello di rotazione più simile al movimento osservato.

Metodi ibridi Combinano una o più delle strategie osservate in precedenza, un metodo molto usato è la combinazione di un sistema di head pose estimation statico affiancato da un tracker che mantiene traccia della posa. Nel caso quest'ultimo perda il riferimento facciale, esso viene reinizializzato dal rilevatore statico permettendo così di avere la precisione di un tracciamento dinamico riducendo il rischio che esso degeneri la rilevazione.

È facile immaginare quanto questi algoritmi siano importanti per ridurre la "distanza" tra macchine ed umani, è perciò cruciale sviluppare sistemi in grado di riprodurre questa *skill* naturale degli umani per incoraggiare l'interazione tra questi due mondi.
Gli sviluppi visti negli ultimi anni sono incoraggianti, con un miglioramento sensibile nelle applicazioni *real-world* in cui hanno maggiore peso le variazioni di ambiente a discapito di quelle nelle immagini, con l'abbandono di approcci *appearance-based* e l'adozione di sistemi *non-linear embedded manifold*, e lo spostamento dell'attenzione nel fornire stime accurate di head pose anche tramite il calcolo dei 3DOF.
È desiderabile tuttavia arrivare ad implementazioni robuste e facilmente utilizzabili al fine di accomodare un numero ampio di applicazioni, perciò è importante sottolineare gli aspetti più importanti in questi sistemi, che trovano applicazione anche nel caso specifico di questa tesi [26]:

- *Accuratezza* È una richiesta indispensabile per sviluppare una qualsiasi applicazione, con un range di discostamento accettabile di 5°

- *Monocularità* Un algoritmo destinato a questi scopi deve essere in grado di fornire risultati accurati senza specifici hardware, ad esempio su semplici camere

- *Autonomia* Non si può avere un sistema in cui debba essere inserito un insieme di informazioni derivate manualmente a priori

- *Simultaneità* Il sistema deve essere in grado di riconoscere più persone nella stessa scena

- *Invarianza all'identità ed all'illuminazione* Il sistema di riconoscimento non può prevedere il funzionamento solo su soggetti selezionati bensì su un qualsiasi esempio e deve essere robusto a cambiamenti ambientali quali l'illuminazione

- *Robustezza in risoluzione* L'algoritmo deve essere in grado di fornire risultati accurati sia in condizioni *near-field* sia in condizioni *far-field* e funzionare con un ampio range di risoluzioni

- *Ampia capacità di rilevazione* È desiderabile avere un sistema in grado di rilevare un ampio range di pose

- *Real-time* Il sistema deve essere adatto ad un utilizzo *real-time*, data la natura specifica di alcune applicazioni questo è un requisito fondamentale

3.1.4 I database di riferimento

Il proliferare di un così vasto insieme di differenti tecniche per la face detection e la face recognition ha reso indispensabile costruire un insieme di strumenti per il *benchmarking* al fine di valutare consistentemente l'accuratezza di un algoritmo proposto.
Il requisito fondamentale comune ai database è la presenza di un alto numero di immagini di test disponibili, così come la disponibilità di dataset preventivamente annotati, ad esempio l'annotazione della posizione facciale e della posizione di landmark seguita secondo un protocollo prestabilito.
Nel corso degli anni si sono susseguiti una serie innumerevoli di questi database, ciascuno con una particolarità dovuta alle condizioni di registrazione delle immagini, alla tipologia dei soggetti in esame, al metodo di annotazione delle immagini, al numero di immagini disponibili, alla variabilità della posa dei soggetti e non ultimo dalle sorgenti di informazione quali ad esempio filmati reali di telecamere di sorveglianza, foto di riconoscimento degli organi di polizia e immagini ottenute in sistemi utilizzati realmente. Non stupisce che esistano database ottenuti da sistemi di sorveglianza reali, vista l'importanza degli sviluppi pratici di questi algoritmi e gli sforzi economici messi in campo dalle istituzioni e dai governi per costruire sistemi di sicurezza avanzati.[4] I database che si sono distinti maggiormente sono:

FERET È storicamente il primo database proposto per specificamente per questi scopi, ed è stato finanziato dal programma *FERET (Face Recognition Technology)* del *Department of Defense (DoD)* degli USA, con l'obbiettivo di incoraggiare la ricerca e la standardizzazione delle valutazioni. A questo programma va il merito di aver costruito il primo database indipendente dai partecipanti al programma, ed attualmente possiede 1800 immagini di 200 differenti individui su 9 differenti posizioni in $[-65°, +65°]$ per l'angolo azimutale, intervallate di 16°e senza alcuna altra variazione di posa e fattori ambientali.
Un secondo merito da attribuire a questo programma è quello di aver uniformato il sistema di valutazione per poter avere un confronto consistente tra diverse soluzioni, permettendo tramite valutazioni indipendenti di identificare punti di forza e debolezza degli approcci proposti di fatto indirizzando appropriatamente la ricerca della comunità scientifica.

CMU PIE Fu rilasciato da Carnegie Mellon University poco dopo il FERET database e per questo ricopre ancora un ruolo importante come strumento di test. Il database contiene 41,368 immagini di 68 soggetti in 13 differenti pose, 43 condizioni di illuminazione differenti e 4 espressioni facciali. Le pose acquisite includono variazioni solamente in angolo azimutale e combinazione di variazioni di elevazione e azimuth.

[4] http://www.face-rec.org/databases/

Multi PIE È un database recente ed estende il database PIE includendo un maggior numero di soggetti (portandoli a 337) ed ampliando l'insieme di espressioni facciali registrate, così come un miglioramento in accuratezza delle registrazioni.

LFW Il database *Labeled Faces in the Wild* si differenzia dai precedenti poichè contiene un insieme di immagini scattate in ambienti non controllati ("in the wild" esplica questo aspetto) e perciò adeguate alla valutazione di algoritmi destinati all'utilizzo pratico in ambienti sconosciuti per sfondo, condizioni di illuminazione, posa ed espressione del soggetto ed eventuali occlusioni facciali, un insieme di condizioni definibile come *unconstrained face detection/recognition.* L'obbiettivo principale di questo dataset è quello di aiutare nello sviluppo di algoritmi di *pair matching recognition* tra facce sconosciute e perciò possiede un insieme limitato e non prestabilito di pose.

IJB-A è il più recente dataset messo a disposizione dall'istituzione statunitense *Intelligence Advanced Research Projects Activity (IARPA)* e contiene 5712 immagini facciali e 2,805 video acquisiti dal web e quindi adeguati al riconoscimento facciale non controllato. Un aspetto particolarmente interessante è la presenza di annotazioni manuali di posizioni delle facce e landmark facciali. Un aspetto che è importante sottolineare è la necessità evidenziata di colmare il gap prestazionale tra algoritmi che operano in condizioni di posa controllata e/o limitata e algoritmi che puntano al riconoscimento facciale in pose arbitrarie per la *unconstrained face detection/recognition.*

AFLW È stato creato per agevolare il riconoscimento facciale e l'estrazione di feature da immagini, il database è sufficientemente grande e dettagliato per eseguire training non controllato, costituito da un insieme di immagini tratte dal web, su ciascuna delle quali è stata eseguita una annotazione manuale delle posizioni delle facce e di 21 punti facciali, rendendolo specialmente appetibile per il training di algoritmi per la *shape prediction.*
Il database è costituito da un insieme eterogeneo di immagini che ritraggono soggetti differenti per età, etnia, espressione facciale ecc. nelle loro attività quotidiane in condizioni di illuminazione qualsiasi e in pose che spaziano ampiamente per quanto riguarda l'angolo azimutale e in misura più limitata seppur comunque soddisfacente negli altri due gradi di libertà.
I database descritti in precedenza, seppur ricchi in termini numerici non forniscono o forniscono in misura limitata le annotazioni delle posizioni delle facce così come dei landmark facciali, in molti casi le limitazioni sono date dalla limitata variabilità della posa, dalle condizioni di registrazione non *real-world* e dal limitato numero di immagini annotate.
Le caratteristiche che distinguono questo database da tutti gli altri sono date dalla ricchezza di informazione ivi contenuta: [27]

- Vi sono 25,993 facce contenute in 21,997 immagini, per la quasi totalità a colori, di molteplici dimensioni. Il 56% dei visi ritratti sono maschili, il rimanente 44% sono visi femminili

- Il numero di singole annotazioni in tutto il database è di 389,473 in una struttura prestabilita di 21 punti di interesse, annotati in base alla visibilità degli stessi
- I soggetti sono ritratti in un range estremamente vario di pose, arrivando al 66% di pose non frontali
- Le annotazioni sono arricchite da rettangoli facciali ed ellissi, secondo la notazione del protocollo FDDB
- Un insieme di strumenti software completa il database, ad esempio per importare dati da altri database esistenti

Questo database rappresenta attualmente la soluzione più completa per il training e il testing di algoritmi per la face detection, face recognition e head pose estimation.
Altri database simili a questo e degni di menzione sono LFPW[5] ed HELEN[6] ed i database *300-W*[7].

Al fine di valutare i progressi nell'evoluzione degli algoritmi da parte della comunità scientifica sono state istituite svariate *challenge* in cui i partecipanti danno prova del proprio lavoro svolto e lo presentano al resto della comunità. Una serie di questi eventi è tenuta anche da organizzazioni governative, come già visto in precedenza molto attive in questo ambito, le principali sono: *300 Faces in-the-wild Challenge*[8], *Face in Video Evaluation (FIVE)* [9], *Face Recognition Vendor Test (FRVT)*[10] e *Face Recognition Grand Challenge (FRGC)*[11].

[5] http://neerajkumar.org/databases/lfpw/
[6] http://www.ifp.illinois.edu/ vuongle2/helen/
[7] http://ibug.doc.ic.ac.uk/resources/300-W/
[8] http://ibug.doc.ic.ac.uk/resources/300-W/
[9] https://www.nist.gov/programs-projects/face-video-evaluation-five
[10] https://www.nist.gov/programs-projects/face-recognition-vendor-test-frvt
[11] https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc

Capitolo 4

Realizzazione

Sviluppare un algoritmo adeguato ad essere eseguito su piattaforme mobili richiede un accurato processo di selezione dei componenti costruttivi del sistema che si intende realizzare. In virtù della grande proliferazione di differenti soluzioni per la head pose estimation e recognition, gran parte del lavoro preliminare è stato fatto in questo senso al fine di valutare vantaggi e svantaggi nel preferire una soluzione rispetto ad un'altra.

Le ragioni che hanno guidato la scelta della soluzione adottata derivano dalle limitazioni che caratterizzano i dispositivi mobili, quali ad esempio l'assenza di CPU multiple e/o l'incapacità di parallelizzare sulle stesse, la limitazione di memoria volatile utilizzabile, la limitazione di storage e le varie limitazioni sulle dimensioni delle pipeline di flusso dei dati (ad es. flussi di immagini). La ragione principale che ha influenzato la realizzazione è l'assenza di hardware adeguato per compiere questo tipo di task, in molti casi infatti gli algoritmi proposti fanno affidamento su camere stereo e/o camere di profondità per acquisire l'immagine mentre in questo caso l'input deriva solamente dalla tradizionale camera del dispositivo.

4.1 Ambiente di sviluppo

Il sistema operativo Android è costruito su tre layer fondamentali: un kernel linux-based per il dialogo con l'hardware, un hardware abstraction layer scritto in C e C++ che realizza tutte le funzioni di middleware, librerie e API e un application framework che ospita le librerie Java necessarie per realizzare applicazioni nel linguaggio nativo di Android. Il sistema possiede due metodi per la compilazione e interpretazione di codice Java, la tradizionale Dalvik Virtual Machine e una più recente e performante Android Runtime (ART).

La libreria di C per Android *Bionic* supporta le più popolari architetture hardware `ARM`, `ARM64`, `x86`, `x86-64` e `MIPS` ma per ragioni di performance rappresenta un sottoin-

sieme delle funzionalità offerte dalla libreria standard completa *libc/glibc*[1] sviluppata e mantenuta da GNU.
Attualmente il sistema possiede il supporto a C++ con un set limitato di funzionalità[2].

Integrazione tra Java e C++

Il sistema operativo Andorid estende le proprie funzionalità al di fuori del dominio Java attraverso il *Native Development Kit (NDK)*[3], in questo modo il sistema gode della flessibilità di utilizzare più linguaggi di programmazione per le proprie applicazioni.

L'integrazione è realizzata dalla *Java Native Interface (JNI)*[4] che fa parte dell'ecosistema tradizionale *Java Development Kit (JDK)* e permette il dialogo tra codice Java che viene eseguito all'interno di una *Virtual Machine (VM)* e codice C, C++ e assembly. Tramite i *java native methods* è possibile perciò eseguire codice da librerie *platform-specific*, da librerie di codice preesistente o piccoli frammenti di esse senza dover riscrivere codebase potenzialmente ampie.
La natura di interfaccia è ideale poichè isola dalle possibile problematiche di incompatibilità introdotte dalle varie soluzioni *vendor-specific* per le VM.

I concetti fondamentali per lo sviluppo di applicazioni che si servono di codice nativo sono i seguenti, per i dettagli è consigliato consultare la letteratura di riferimento.

Invocation API JavaVM e JNIenv sono due strutture necessarie per il funzionamento dell'integrazione, la prima corrisponde alla virtual machine (in Android è possibile averne solamente una per ciascun processo) mentre la seconda contiene le mappature in *function table* e fornisce la totalità delle funzioni della JNI, per questo ogni invocazione ad un metodo nativo necessita di un riferimento ad essa.

Compilazione, loading e linking La dichiarazione di un metodo invocato tramite JNI deve essere specificata tramite la keyword `native` seguita dalla firma del metodo, ad esempio:

```
public native int add(int x, int y);
```

Il caricamento di una libreria deve essere eseguito in Java in uno *static class initializer*, supponendo di avere un libreria `lib.so` si deve eseguire il caricamento specificandone il nome senza decorazioni, sarà il sistema ad identificare la libreria adeguata all'architet-

[1] https://www.gnu.org/software/libc/index.html
[2] https://developer.android.com/ndk/guides/cpp-support.html
[3] https://developer.android.com/ndk/index.html
[4] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html

tura, un esempio è:

Listing 4.1: Firma del metodo nativo in Java e caricamento della libreria

```
1 package com.my.pkg;
2
3 class Cls {
4         public native int add(int x, int y);
5
6     static {
7         System.loadLibrary("lib");
8     }
9 }
```

La libreria dinamica associata dovrà contenere l'implementazione del metodo, la cui firma deve rispettare alcuni vincoli imposti dalla JNI al fine di permettere il *resolving* del metodo stesso, in particolare deve possedere il prefisso `jint JNIEXPORT JNICALL Java_` seguito dal qualificatore completo della classe, seguito a sua volta dal nome del metodo, ad esempio:

Listing 4.2: Firma del metodo nativo nel codice C

```
1 jint JNIEXPORT JNICALL Java_com_my_pkg_add(
2         JNIEnv *env,     /* interface pointer */
3         jobject obj,     /* "this" pointer */
4         jint x,          /* argument #1 */
5         jint y)          /* argument #2 */
6 {
7     // code
8 }
```

È opportuno fornire un metodo di inizializzazione che verifichi il corretto caricamento della libreria, questo metodo viene invocato contestualmente a `System.loadLibrary` e ritorna dei valori di default:

Listing 4.3: Metodo invocato da System.loadLibrary

```
1 jint JNI_OnLoad(JavaVM* vm, void* reserved)
2 {
3     JNIEnv* env;
4     if(vm->GetEnv(reinterpret_cast<void**>(&env), JNI_VERSION_1_6) !=
          JNI_OK) {
5         return -1;
6     }
7
8     return JNI_VERSION_1_6;
9 }
```

Tipi primitivi e *reference* I tipi primitivi sono copiati direttamente nella parte nativa di codice (nella quale possiedono le controparti `jint`, `jdouble`, `jboolean` ecc.) mentre gli oggetti generici sono passati per *reference*, la virtual machine deve però tener traccia di tutte le istanze di oggetti inizializzati o riferiti dal codice nativo per evitare

che subiscano la *garbage collection*, a sua volta il codice nativo deve poter godere di strumenti con cui segnalare l'abbandono dei riferimenti alla virtual machine in modo che essa possa liberare spazio in memoria.
Le reference si suddividono in locali e globali e la differenza si riflette sulla lunghezza del ciclo di vita degli oggetti, nel caso essi siano riferiti localmente essi sono liberati una volta che il metodo in cui risiedono termina, nel secondo caso godono di un ciclo di vita più lungo.
La scelta di quali tipologie di reference utilizzare influenza in maniera determinante l'efficienza del codice, e in contesti critici rappresenta uno dei fattori di rischio di esaurimento della memoria virtuale destinata al processo.

Identificatori Nel codice nativo i riferimenti alle classi, ai metodi e ai campi degli stessi risiedono in strutture chiamate rispettivamente `jclass`, `jmethodID` e `jfieldID`. Questi elementi sono veri e propri puntatori che devono essere inizializzati opportunamente in quanto eseguirne il lookup ripetuto ha ripercussioni sull'efficienza, per questo motivo è opportuno invocare un metodo accessorio di inizializzazione successiva al caricamento della libreria utlizzando le funzioni `FindClass`, `GetMethodID` e `GetFieldID` e trasformando le reference ottenute in reference globali che possono essere utilizzate in un successivo momento:

Listing 4.4: Inizializzazione

```
1  static jclass Cls;
2  static jmethodID ClsAdd;
3
4  jint JNIEXPORT JNICALL Java_com_my_pkg_jniInit(
5          JNIEnv* env,
6          jobject thiz)
7  {
8          // Get local reference
9          jclass cls_local = env->FindClass("com/my/package/Cls");
10         if(cls_local == NULL) {
11                 return JNI_ERR;
12         }
13
14         // Cast into global reference
15         Cls = reinterpret_cast<jclass>(env->NewGlobalRef(cls_local));
16
17         // Get reference to a method
18         ClsAdd = env->GetMethodID(Cls,"add","(II)I");
19         if(ClsAdd == NULL) {
20                 return JNI_ERR;
21         }
22
23         // ... delete local refs
24         env->DeleteLocalRef(cls_local);
25
26         return JNI_OK;
27 }
```

I parametri necessari per invocare GetMethodID sono la reference alla classe (precedentemente ricavata), il nome del metodo e una codifica che rappresenta la firma del metodo, in particolare tra parentesi il tipo dei parametri in input seguiti dal tipo del dato in output. Queste proprietà devono essere quelle specificate precedentemente nel metodo `native`.
Analogamente al precedente, è opportuno liberare la memoria tramite un metodo di de-inizializzazione da invocare quando l'applicazione sta per arrestarsi:

Listing 4.5: De-inizializzazione

```
1 jint JNIEXPORT JNICALL Java_com_my_pkg_jniDeInit(
2         JNIEnv* env,
3         jobject thiz) {
4         env->DeleteGlobalRef(cls);
5
6         return JNI_OK;
7 }
```

JNI permette di sfruttare appieno la potenza del linguaggio C/C++ ospitandolo all'interno di applicazioni Android e, sebbene possa sembrare complicato ad un primo impatto, facilita di molto il lavoro di chi intende utilizzare algoritmi complessi e dispendiosi dal punto di vista di risorse i quali spesso non possiedono implementazioni in Java a causa della scarsa affinità del linguaggio con contesti *resource-constrained*.
Sviluppare con NDK è perciò l'unica strada percorribile per portare a termine il progetto, e le limitazioni del supporto alla libreria standard C++ ha inciso in maniera decisiva sulla scelta degli algoritmi da sfruttare in questo caso.

4.2 Algoritmi

La parte iniziale di questo lavoro di tesi è stata impiegata per la consultazione della letteratura al fine di individuare gli strumenti necessari per raggiungere l'obbiettivo. Una grande mole di tempo è stata necessaria per valutare l'adeguatezza delle implementazioni disponibili per quanto riguardava la face detection e la face alignment, vista anche l'ampia scelta a disposizione. I princípi che hanno guidato le scelte sono stati dettati dalla piattaforma a cui era destinato il software e dalle sue limitazioni hardware e software.

La fase di face detection dell'algoritmo è affidata ad un algoritmo che realizza il training di una support vector machine (SVM) per il riconoscimento della posizione della faccia nell'immagine.
La stima della *face shape* ovvero la stima dei landmark facciali (spesso definita anche *face alignment*) è eseguita a partire dalla rilevazione del rettangolo facciale impiegando un sistema che stima la posa eseguendo più iterazioni sull'immagine, nelle sezioni che seguono vi sono i dettagli preminenti.

4.2.1 Face detection

Boosted Cascade of Simple Features

Estrarre informazioni da immagini a colori RGB è sempre stato un compito estremamente dispendioso dal punto di vista computazionale a causa della grande mole di informazione che ciascuna immagine porta. È necessario ricavare una rappresentazione alternativa di *feature* delle immagini per rendere il processo di rilevamento più semplice, le *Haar-like features* sono alla base di questo algoritmo (vedi Fig. 4.1) [28].
L'algoritmo estrae informazione dall'immagine tramite un rettangolo che viene fatto scorrere su di essa, per ciascuna sezione dell'immagine individuata dal rettangolo vengono calcolate le Haar-like features, che corrispondono a proprietà dei rettangoli adiacenti in cui vengono calcolate le differenze di intensità dei pixel e confrontate con template conosciuti, ad esempio la regione degli occhi è generalmente più scura della regione della guancia sottostante ecc. In base a queste proprietà vengono ricavati dei *weak classifier* ovvero dei classificatori che forniscono un piccolo contributo informativo a livello decisionale ma che combinati in cascata in gran numero, ciascuno rappresentante una feature differente, producono un classificatore sufficientemente accurato. L'algoritmo si compone di 4 passi fondamentali:

1. *Selezione delle Haar-like feature* Vi sono 4 forme fondamentali per i rettangoli con cui calcolare le aree di intensità, ciascuna forma è convenientemente costruita per riuscire a catturare le caratteristiche facciali (vedi Fig. 4.1)

2. *Creazione di una integral image* Una *integral image* permette di calcolare in tempo costante il valore complessivo di intensità dei pixel contenuti nei rettangoli in esame, osservando che il contributo di ciascun rettangolo è calcolabile come contributo dei rettangoli adiacenti calcolati in precedenza

3. *Training AdaBoost* Il training non può essere eseguito assumendo come input le features calcolate nelle varie regioni dell'immagine, visto che le *Haar-like feature* sono in numero molto grande, perciò viene adattato un training di tipo AdaBoost permette di selezionare solamente un sottoinsieme di feature salienti

4. *Cascading di classificatori* Organizzare i classificatori è importante per incrementare le performance dell'algoritmo scartando ad esempio le *non-promising regions* e focalizzandosi sulle aree più interessanti dell'immagine

Un punto importante di questo approccio è la creazione di un *weak learning algorithm* il quale seleziona il ristretto insieme di feature che sono rappresentative delle caratteristiche degli oggetti da rilevare, fondamentalmente la difficoltà è nell'individuare l'insieme di queste. Per ciascuna feature il *weak learner* determina la soglia ottima di classificazione che minimizza gli errori di classificazione.

La cascata di classificatori è configurata come un albero decisionale degenere in cui la stragrande maggioranza delle *window* sull'immagine vengono scartate in funzione di

una opportuna soglia in ciascun nodo dell'albero, minimizzando il *false negative rate*. Il comportamento a cascata è caratteristico di questa struttura in quanto l'algoritmo si focalizza solo sulle finestre che non sono state scartate nei nodi antecedenti dell'albero, costituendo la cosiddetta *attentional cascade*.

Il training di un classificatore di questo tipo impone di trovare un complesso trade-off tra il numero di nodi del classificatore, il numero di feature presenti ad ogni nodo e la sogliatura nel tentativo di minimizzare il numero di feature valutate. In termini pratici si produce un framework efficiente che ad ogni nodo della cascata riduce il *false positive rate* e conseguentemente diminuisce il *detection rate*, in funzione di questo viene impostato inizialmente un parametro obbiettivo per la minima accettabile riduzione dei *false positive* ed uno per la massima accettabile perdita di *detection*. In ciascun nodo vengono aggiunte feature finchè non si incontrano questi due obbiettivi e nella cascata complessivamente si aggiungono nodi fino a giungere all'obbiettivo complessivo stabilito. Utilizzando questo metodo si arriva alla costruzione di un modello con 38 differenti nodi che valutano complessivamente un insieme di oltre 6000 feature con prestazioni ottimali in termini di tempo di esecuzione ed accuratezza.

I risultati raggiunti con questo metodo hanno superato di gran lunga lo stato dell'arte ed hanno posto questo lavoro come riferimento per tutti i successivi tentativi di miglioramento di questa categoria di algoritmi, interessante è inoltre notare che questa metodologia permette di eseguire un training per una qualsiasi applicazione di detection su immagini rendendo necessaria solamente l'estrazione di un insieme di feature adatte all'oggetto che si intende rilevare.
Viola e Jones furono i primi a fornire un'implementazione di un algoritmo di object detection con prestazioni real-time [28], questo fu storicamente il primo esempio che dimostrò l'applicabilità di algoritmi di questo tipo. La suite OpenCV fornisce un face detector che riproduce il lavoro svolto da questi due autori.

(a) *(b)*

Figura 4.1: Due esempi di *Haar-like features* usate nel Viola-Jones detector [28]

Histogram of Oriented Gradients (HOG)

Histograms of Oriented Gradients for Human Detection [29] è un caposaldo degli algoritmi in object detection tramite Support Vector Machines (SVM), costituisce un punto di riferimento in termini di prestazioni sui database di riferimento.
In matematica il concetto di gradiente è una generalizzazione del concetto di derivata

di funzioni di più variabili, corrisponde ad un vettore le cui componenti sono le derivate parziali della funzione, e similmente alle derivate, il gradiente misura la pendenza della tangente ad una curva, si orienta nel verso di maggior incremento di pendenza e la sua ampiezza è proporzionale alla pendenza. Nel campo dell'image processing il gradiente è calcolato su una funzione di due variabili (la funzione di intensità dei pixel).
L'assunzione di fondo è che la forma locale da individuare è ben caratterizzata dalla distribuzione locale delle intensità e dalla direzione dei gradienti che definiscono gli *edge* dell'immagine. In pratica questa descrizione viene ricavata estraendo istogrammi 1-dimensionali da regioni (o "celle") dell'immagine rappresentanti i gradienti ovvero le direzioni degli *edge* sui pixel appartenenti alla cella.
L'insieme di questi istogrammi è una rappresentazione dell'immagine che viene normalizzata localmente rispetto all'intensità delle celle per essere più robusta a variazioni di illuminazione, i descrittori a blocchi normalizzati si dicono descrittori *Histogram of Oriented Gradients (HOG)*. Combinando i vari blocchi tra loro (ad es. sovrapponendo i blocchi localmente) è possibile ottenere feature vector adatti al training di una Support Vector Machine per la rilevazione di facce.

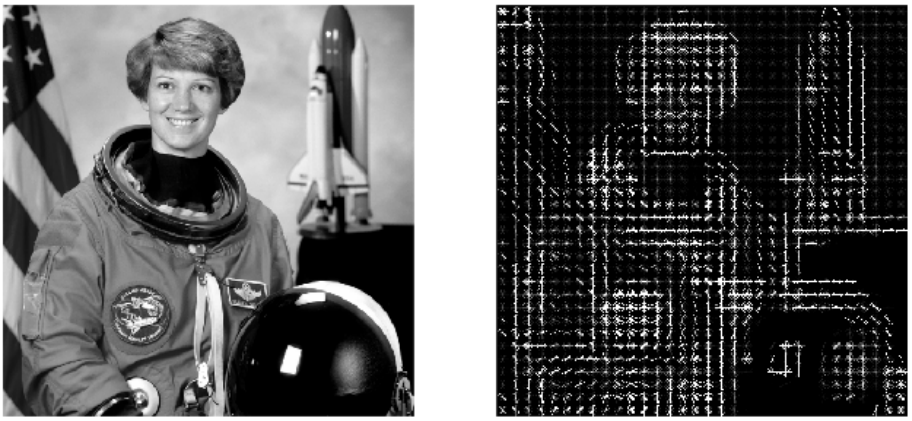


Figura 4.2: Un'immagine in cui sono stati ricavati i descrittori *Histogram of Oriented Gradients (HOG)*[5]

La rappresentazione con descrittori HOG ha numerosi vantaggi poiché cattura la struttura degli *edge* e dei gradienti che sono caratteristiche forti del pattern facciale, robuste alle variazioni di rotazione e illuminazione.
Le scelte compiute sui parametri dell'algoritmo sono state guidate da prove eseguite sperimentalmente su un insieme di immagini di persone in piedi ritratte in varie pose e con vari livelli di occlusione. Riassumendo brevemente i passaggi principali sono:

1. *Normalizzazione colore* Non viene eseguita alcuna normalizzazione nel dominio del colore dell'immagine, poiché si è dimostrato non contribuisce a migliorare i risultati

[5] http://scikit-image.org/docs/dev/auto_examples/plot_hog.html

2. *Calcolo dei gradienti* Il calcolo dei gradienti influenza pesantemente l'efficienza computazionale, utilizzando un semplice filtro di tipo gaussiano unidimensionale porta risultati ottimali

3. *Raggruppamento spaziale* Ciascun pixel dell'immagine contribuisce con un peso del gradiente centrato su di esso nell'istogramma delle orientazioni e i pesi sono raggruppati secondo le varie celle. Le celle possono avere forma quadrata o circolare e gli istogrammi di orientazione sono suddivisi secondo angolazioni $[0°, 180°]$ (orientazione senza segno) oppure $[0°, 360°]$ (orientazione con segno)

4. *Normalizzazione e blocchi di descrittori* Le intensità dei gradienti sono soggette a forti variazioni dovute al contrasto di illuminazione e dal contrasto con lo sfondo, perciò la normalizzazione aiuta a mitigare queste alterazioni. Solitamente si organizzano le celle adiacenti in blocchi in cui viene eseguita una normalizzazione complessiva, sovrapporre i blocchi aiuta inoltre a migliorare le performance ampliando l'influenza di ciascun pixel su più blocchi. I blocchi possono essere quadrati, rettangolari o circolari e possono essere organizzati su griglie rettangolari e quadrate oppure su griglie polari, ciascuna delle quali riesce a catturare aspetti differenti dell'immagine

I contributi introdotti da questo algoritmo sono stati di grande importanza e superano tutte le soluzioni precedenti in termini di accuratezza, il calcolo dei gradienti con grande definizione, la suddivisione fine in orientamenti e la normalizzazione del contrasto a livello locale hanno contribuito a rendere questo uno dei lavori più importanti nel recente passato.

Object Detection with Discriminatively Trained Part-Based Models

Questo lavoro [30] si basa sul concetto di *pictorial structures* per rappresentare oggetti in immagini costituiti da parti combinate in una configurazione deformabile, ciascuna parte rappresenta una particolare proprietà dell'oggetto mentre la deformabilità della struttura deriva da connessioni di tipo "a molla" tra coppie di parti.
Usare questo approccio è particolarmente indicato per identificare oggetti di struttura semi-rigida che possiedono potenzialmente ampie variazioni delle singole parti.
Si consideri il problema di eseguire il training di un *part-based model* avendo a disposizione una collezione di immagini in cui sono stati annotati gli oggetti di interesse tramite quadrati, detti *bounding box*. A partire da un algoritmo performante quale è HOG, si è cercato di arricchirlo usando un filtro (corrispondente ad un insieme di pesi) di tipo *star-structured part-based model* (detto anche filtro *root*) e un insieme di *part filters* e modelli di deformazione. Si pensi al *detector* come un classificatore che riceve in input un'immagine, una posizione all'interno dell'immagine ed una scala, il modello è rappresentato da un semplice filtro perciò è possibile calcolare uno *score* come $\beta * \Phi(x)$ dove β è il filtro, x è l'immagine e Φ è un feature vector, lo score è calcolato su un'immagine eseguendo il *dot product* tra il filtro e una finestra della *feature pyramid* calcolata sul-

l'immagine stessa al fine di catturare le caratteristiche a scale diverse.

Gli *object model* impiegati sono filtri che eseguono la pesatura di caratteristiche in regioni di una *feature pyramid* in cui le *pixel-level feature maps* sono costruite misurando $\theta(x, y)$ ovvero l'orientazione ed $r(x, y)$ come intensità del gradiente in ciascun pixel (x, y) dell'immagine, nel caso sia a colori si considera il canale con più ampia intensità come riferimento. I gradienti sono calcolati esattamente come in HOG utilizzando filtri a differenze finite del tipo $[-1, 0, +1]$ e la loro trasposta.
Per il modello a parti deformabili per un viso, una stima del *root filter* serve per individuare la posizione della faccia, mentre una serie di filtri a più fine risoluzione identificano le varie parti quali il naso, gli occhi ecc.
Un aspetto importante di questo approccio è la riduzione dimensionale delle HOG feature eseguita in virtù dell'osservazione che molte di queste condividono caratteristiche comuni riducendo così il costo per ricavarne l'insieme e perciò incrementando sensibilmente le prestazioni.

Questo approccio ha introdotto nuovi metodi per eseguire il training discriminativo per classificatori che usano *latent information* e sfrutta ampiamente le tecniche per il matching di modelli deformabili per immagini. Le innovazioni introdotte hanno permesso di sviluppare un algoritmo che ha impostato nuovi standard di accuratezza per questo tipo di task.

Max Margin Object Detection (MMOD)

La *Max-Margin Object Detection (MMOD)* [31] è una strategia che si integra con gli algoritmi di object detection appena discussi, risultando in un incremento dell'accuratezza.
Il meccanismo di training è tipicamente eseguito estraendo esempi positivi ed esempi negativi udano *image windows* sulle immagini di training, un classificatore binario viene allenato su questo insieme di posizioni note, in seguito il classificatore viene testato su posizioni prive di target e tutte le eventuali *false alarm windows* vengono incluse nell'insieme di training, e con questo insieme di informazioni più complete si riesegue il training del classificatore ed eventualmente si itera il procedimento. Procedere in questo modo non utilizza in modo efficiente tutte le informazioni a disposizione, la limitazione è dovuta la fatto che viene scelto solamente un sottoinsieme di *image windows* disponibili e quelle che si sovrappongono parzialmente sono fonte di *false alarms* e non è possibile assegnarle come esempi nell'insieme di training poiché in molti casi rappresentano una "semi-rilevazione valida". L'accuratezza del classificatore è dunque sub-ottimale e tramite il metodo introdotto in questo lavoro scientifico si migliora tale accuratezza permettendo l'utilizzo di tutte le *sub-windows* disponibili.

Questa strategia ottimizza l'estrazione delle *sub-windows* incrementando l'accuratezza del rilevatore poiché considera informazioni che altrimenti verrebbero ignorate. La scoperta più eclatante fatta usando questo approccio è che usando un semplice filtro per

HOG si ottengono prestazioni superiori rispetto all'utilizzo di sistemi più complessi come ad esempio i mixture models introdotti in [29].

4.2.2 Face recognition e face alignment

One Millisecond Face Alignment with an Ensemble of Regression Trees

L'algoritmo introdotto nel paper di Kazemi e Sullivan [32] introduce una soluzione che riesce a raggiungere prestazioni di 1ms per realizzare l'alignment dei landmark facciali, ottenute integrando stategie di *prior face alignment* nel sistema costistuito da una cascata di funzioni di regressione.
Il lavoro si basa sulla grande mole di soluzioni implementate precedentemente, in particolare nei paper di P. Dollàr *et al.* [33] e Cao *et al.* [34] per quanto riguarda le tematiche di *cascaded shape regression* ed *explicit shape regression* rispettivamente.
Il funzionamento si basa sull'utilizzo di una cascata di regressori che rappresentano una serie di iterazioni in cui gli *shape parameters* sono stimati eseguendo una trasformazione dal sistema di coordinate globale ad un sistema di coordinate normalizzato in base alla stima corrente della forma iterando fino a giungere ad una convergenza. Secondariamente, si risolve il problema degli ottimi locali in fase di testing introducendo un vincolo ulteriore che impone che la *shape* cercata debba giacere in un sottospazio lineare definito dalle *shape* di training riducendo in questo modo lo spazio di ricerca ed incrementando le prestazioni.

Il contributo di ciascun regressore nella cascata stima un update della shape corrente eseguendo predizioni sui valori di intensità dei pixel le cui posizioni dipendono dalla stima corrente. In questo modo ad ogni iterazione si aggiorna il modello fino a convergere, a patto che sia stata fornita una *shape* di partenza anch'essa giacente nel sottospazio lineare in cui si converge, una stima adeguatamente precisa si può ottenere da una semplice *mean shape* dei dati di training opportunamente scalata e centrata utilizzando un face detector.

Ciascun albero di regressione opera in questo modo: ogni *split node* dell'albero esegue una decisione in base alla soglia sulla differenza di intensità di due pixel le cui posizioni sono determinate rispetto alla *mean shape* di partenza, a questi pixel corrispondono altrettanti pixel le cui posizioni devono essere determinate in funzione della *current shape* mediante una trasformazione (per alcuni aspetti approssimativa) dei punti.
Gli *split node* sono scelti tra un insieme di candidati generati casualmente, scegliendo con strategia greedy il nodo che minimizza l'errore quadratico complessivo. Il numero di candidati è quadratico rispetto al numero di pixel nell'immagine e perciò è difficile ottenere un insieme di candidati di partenza adeguato senza cercarne un numero elevato, per questo motivo si introduce una misura che incentiva la scelta di split con pixel vicini tra loro.

Le innovazioni introdotte con questo approccio sono di importante rilevanza applicativa poiché permettono l'esecuzione di questo algoritmo su CPU singole con prestazioni

notevoli in termini di velocità ed accuratezza, in secondo luogo sono state introdotte componenti per agevolare il training anche in presenza di dati incompleti.

4.2.3 3D tracking di oggetti rigidi

Un aspetto fondamentale dell'implementazione dell'algoritmo in Selfear 2.0 è rappresentato dalla capacità del software di eseguire un tracciamento (o riconoscimento) della struttura di un oggetto ritratto in un'immagine e la capacità di ricavare l'orientamento di tale oggetto sfruttando la conoscenza pregressa delle proprietà fisiche (e tridimensionali) dell'oggetto stesso. Questo è possibile affiancando ad algoritmi di computer vision alcune tecniche molto utilizzate in contesti industriali, con applicazioni in servoassistenza visiva di robot industriali, sistemi di realtà aumentata e sistemi di volo.

Nel florilegio di scelte presenti in letteratura, il ramo rilevante in questo progetto è senza dubbio quello dell'*online monocular model-based tracking di oggetti 3D rigidi* [35] che cerca di ricavare il posizionamento di un oggetto 3D in *near real-time* rispetto ad una camera singola che lo ritrae. Questa scelta deriva direttamente dal sistema che intendiamo sviluppare, privo di particolare hardware fotografico e con necessità di elaborazione in tempo reale.
Per la rilevazione dell'oggetto di interesse si possono utilizzare sistemi con marker o sistemi che rilevano le caratteristiche naturali dell'oggetto, e questa seconda categoria è quella rilevante ai fini del progetto.

Perspective projection model

Consideriamo il modello di camera cosiddetto *pinhole camera model*, la formulazione matematica che esplica la traduzione dallo spazio immagine tridimensionale all'*image plane* è data dalla seguente descrizione: sia $\mathbf{M} = [X, Y, Z]^T$ un punto espresso in coordinate euclidee nello spazio, ad esso corrisponde un punto nel piano immagine $\mathbf{m} = [u, v]^T$.
Il legame geometrico è espresso dall'equazione

$$s\tilde{\mathbf{m}} = P\tilde{\mathbf{M}} \tag{4.1}$$

in cui s è un fattore di scala, $\tilde{\mathbf{m}} = [u, v, 1]^T$ e $\tilde{\mathbf{M}} = [X, Y, Z, 1]^T$ sono le coordinate omogenee dei punti $\mathbf{m}$ e $\mathbf{M}$ rispettivamente.
Definiamo $\mathbf{P}$ matrice di proiezione, essa descrive la trasformazione del punto ed è composta in questo modo

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{4.2}$$

dove $\mathbf{K}$ è la matrice di calibrazione (detta anche matrice intrinseca) e $[\mathbf{R}|\mathbf{t}]$ è una matrice $3x4$ (detta anche matrice estrinseca) che rappresenta il cambiamento di coordinate dal *world coordinate system* al *camera coordinate system*.

La matrice intrinseca $\mathbf{K}$ contiene i parametri caratteristici del dispositivo quali la distanza focale, le coordinate del *principal point* ovvero l'intersezione dell'asse ottico con il piano immagine (e spesso approssimabile con il centro dell'immagine) e un parametro

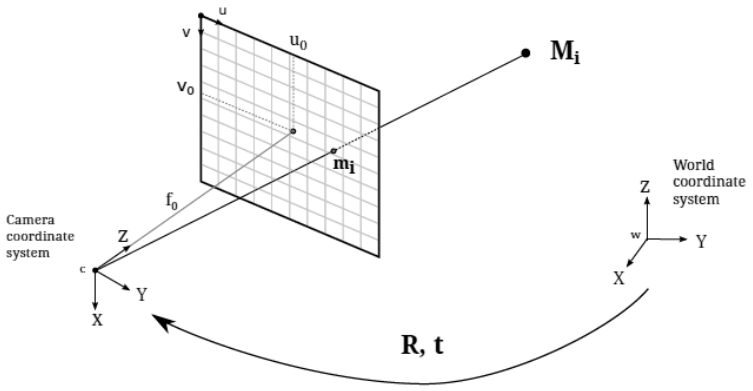


Figura 4.3: Il *prospective projection model* per la *pinhole camera*, la trasformazione delle coordinate è determinata da **R** e **t** ed è dipendente dai parametri intrinseci della camera **K**

di *skew*. Nel tracking spesso è necessario conoscere questi parametri e nel caso in cui si conoscano una camera si dice calibrata, altrimenti esistono strumenti software appositi (da utilizzare in separata sede) per ricavare questi parametri servendosi di oggetti di geometria conosciuta. La matrice estrinseca $[\mathbf{R}|\mathbf{t}]$ è composta da **R** che è una matrice $3x3$ rappresentante una rotazione della camera mentre **t** ne descrive una traslazione.

Parametrizzazione della posa

È importante è stabilire il metodo con cui si sceglie di stimare la posa, in questo contesto gli *euler angles* sono i più adatti, perché ci permettono di rappresentare la matrice di rotazione **R** come rotazioni sui tre assi X, Y, Z tramite il prodotto di tre matrici ciascuna rappresentante una rotazione angolare.

Stima della matrice estrinseca

La stima della matrice estrinseca è il fulcro della funzionalità del progetto ovvero dato un insieme di n punti 3D $\mathbf{M_i}$ e i corrispondenti punti $\mathbf{m_i}$ del piano immagine si cerca una matrice di proiezione **P** che risolva per ogni i la corrispondenza $\mathbf{P}\tilde{\mathbf{M}}_\mathbf{i} = \tilde{\mathbf{m}}_\mathbf{i}$ a meno di un fattore di scala, questo problema è risolto impiegando algoritmi di risoluzione del problema *PnP*.

Perspective-n-point (PnP) problem

Tra le svariate tecniche per eseguire il calcolo della matrice di proiezione la formulazione del *Perspective-n-Point (PnP) problem* è di particolare interesse. La lettera "n" nella formulazione del nome indica che il problema ha l'obbiettivo di ricavare le informazioni cercate a partire dalla corrispondenza tra n punti 3D e altrettanti corrispondenti punti

2D. Questo è un problema geometrico estremamente complesso da risolvere e complicato da descrivere, è importante notare tuttavia che esistono una serie di algoritmi che riescono a trovare soluzioni al problema in tempo lineare. [36] [37]

La suite OpenCV possiede svariate implementazioni, in particolare la soluzione iterativa, la soluzione *P3P* [36] e la soluzione *EPnP* [37], richiedendo in input un insieme dei punti 2D localizzati nell'immagine ed un insieme di punti 3D derivanti da un modello analizzato a priori inserito manualmente nel codice. L'algoritmo richiede altresì una matrice di caratteristiche intrinseche della camera e fornisce in uscita le matrici di rotazione $\mathbf{R}$ e di traslazione $\mathbf{t}$. Queste matrici contengono l'informazione sufficiente per ricavare una stima dei tre angoli di rotazione rispetto agli assi X, Y, Z che corrispondono perciò alla direzione dell'oggetto osservato rispetto alla camera.

Dlib-ml: A Machine Learning Toolkit

Al termine di svariate prove la scelta è ricaduta su una libreria di software per il machine-learning implementata in C++ [39] la quale si distingue per essere open-source (con licenza Boost Software License[6]), *cross-platform* e sviluppata seguendo il paradigma del *contract-programming / component-based engineering* rendendola così una collezione di componenti indipendenti, accuratamente descritti e documentati e perciò facilmente integrabili in applicazioni C++ esistenti sia in ambito di ricerca sia in ambito commerciale. La libreria ospita la suo interno un ampia scelta di algoritmi che sono costantemente aggiornati con gli avanzamenti in ambito di ricerca nei campi di algebra lineare, machine learning, reti bayesiane e neurali e ottimizzazione, inoltre in molti casi non solo si hanno implementazioni aggiornate ma sono disponibili algoritmi *state-of-the-art*.
La categoria di machine learning che è stata impiegata in questo progetto è sviluppata in modo tale da fornire flessibilità per integrare questi algoritmi con software preesistente ed in particolare essa ospita implementazioni degli algoritmi Histogram of Oriented Gradients (HOG) [29] con Max Margin Object Detection (MMOD) [31] per la face detection e One Millisecond Face Alignment with an Ensemble of Regression Trees [32].
La suite implementa inoltre gli strumenti per eseguire il training di nuovi modelli per la detection e l'alignment per sfruttare a pieno le possibilità offerte da tali algoritmi.

Algoritmo gazr per la head pose estimation

L'algoritmo di cui Selfear 2.0 si serve per eseguire la head pose estimation è stato studiato dall'istituto elvetico École Polytechnique Fédérale de Lausanne (EPFL) ed in particolare dal Computer-Human Interaction in Learning and Instruction Laboratory (CHILI). Si evince da quest'ultimo nome il fatto che lo studio della head pose estimation, come peraltro già ampiamente affermato, ha un ruolo fondamentale per costruire strumenti di comunicazione uomo-macchina ed in particolare questo algoritmo è utilizzato per studiare il fenomeno della *with-me-ness* ovvero una misura del grado di attenzione e di coinvolgimento degli umani nello svolgere task assieme a dei robot, tramite la head pose

[6] http://dlib.net/license.html

estimation si ricava la percentuale di tempo in cui il robot è riuscito a stimare correttamente il coinvolgimento dell'utente.

Il processo di estrazione della posa della testa è il seguente:

1. *Acquisizione del frame* proveniente dalla camera
2. *Face detection* tramite l'implementazione di HOG con MMOD [29] [31] per ricavare la posizione del viso nell'immagine
3. *Face alignment* sfruttando l'implementazione ERT [32] per ricavare le posizioni dei landmark 2D nell'immagine
4. *Risoluzione PnP* eseguita con una delle implementazioni attualmente disponibili in OpenCV per ricavare la matrice estrinseca di parametri
5. *Trasformazione* delle informazioni calcolate in angoli di rotazione rispetto agli assi tridimensionali

4.3 Applicazione

La semplicità della soluzione studiata ed il fatto che fosse sviluppata in C++ ha permesso un più agevole porting verso la piattaforma Android con NDK, con un calo fisiologico ma comunque accettabile delle prestazioni.
La realizzazione dell'applicazione Selfear 2.0 consiste primariamente nel porting dell'algoritmo *gazr* (si veda Sottosez. 4.2) e il codice deriva in parte da realizzazioni precedenti destinate al porting della libreria dlib[7] e frammenti della libreria Google Samples[8].
I principali problemi affrontati in questa parte del lavoro sono stati:

- Analizzare la soluzione esistente, che costituiva il punto di partenza su cui costruire l'applicazione, per identificare le componenti che sarebbero tornate utili alla realizzazione e scartare le funzionalità non necessarie
- Eseguire correttamente il meccanismo di build con NDK, considerata la relativa immaturità degli strumenti di sviluppo
- Modificare la componente Java per dialogare correttamente con la parte nativa, che a sua volta è stata modificata per gestire il passaggio dei dati delle rilevazioni
- Adeguare l'interfaccia iniziale al fine di renderla utilizzabile da un utente in configurazione di *self-adjustment* (cambiare input della pipeline della camera, modificare le trasformazioni dell'immagine)
- Integrare la capacità di eseguire più algoritmi differenti tra quelli disponibili in OpenCV

[7] https://github.com/tzutalin/dlib-android
[8] https://github.com/googlesamples

- Ideare un metodo per salvare i dati delle rilevazioni e i dati delle prestazioni per poterli sfruttare in fase di validazione
- Migliorare l'algoritmo studiando possibili semplificazioni adeguate allo specifico caso (ridurre lo spazio di ricerca escludendo scale di immagini troppo piccole, utilizzare immagini in bianco e nero per eseguire l'algoritmo HOG)
- Migliorare l'algoritmo cercando di completare un training per la face detection e la face alignment a partire da immagini ottenute da un dataset annotato, estraendone le informazioni sviluppando programmi in C++ per ricavare i dati da database SQL, creare un dataset bilanciato di pose, verificare manualmente la bontà del dataset rimuovendone il rumore

Nelle sezioni seguenti sono elencate le principali componenti dell'applicazione e vengono brevemente illustrate le motivazioni alla base delle scelte di implementazione.

La *shared library* per la libreria dlib

Android NDK fornisce gli strumenti per eseguire il build[9] di librerie destinate ad essere integrate nelle applicazioni, per questo progetto è stata utilizzato lo shell script `ndk-build` per la creazione della shared library in formato `.so` destinata a Sefear 2.0. Tale libreria include quasi totalmente il codice dell'algoritmo gazr, il quale a sua volta include frammenti della libreria bullet3[10].

Componenti di utilità Le componenti di supporto all'esecuzione servono a compiere la traduzione dei frame tra le diverse codifiche di colore (ed in questo il fatto di essere scritte in C++ favorisce l'efficienza del pacchetto software), permettono la gestione dei file di input (ad es. il caricamento del face shape model che si specifica via Java), la gestione delle varie tipologie di informazione e la loro trasformazione e utilità per eseguire i calcoli matriciali (ad es. estrazione degli euler angles di rotazione).

Componente jni_head_pose_det.cpp Questa parte del codice ospita le definizioni dei metodi necessari a JNI (vedi Sottosez. 4.1), esegue l'inizializzazione e de-inizializzazione, e nella funzione `jniBitmapExtractFaceGazes()` traduce l'informazione da matrice di rotazione ad angolo di rotazione, creando gli oggetti Java che contengono tale informazione.

Componente head_pose_estimation.cpp Questa componente esegue l'inizializzazione acquisendo i parametri intrinseci e caricando il file per il modello facciale.
La funzione principale è tuttavia `HeadPoseEstimation::pose()` che dati i punti rilevati sull'immagine esegue il fitting usando uno dei tre metodi per PnP disponibili, per poi eseguire il disegno dell'overlay della rilevazione sull'immagine originale comprensiva di assi direzionali.

[9] https://developer.android.com/ndk/guides/build.html
[10] https://github.com/bulletphysics/bullet3

Il *submodule* dlib

Il build system standard per Android è Gradle[11], un sistema che rende semplice l'integrazione di *codebase* preesistenti, facilita la creazione di *build variants* e semplifica la configurazione e la personalizzazione del *processo di build* di un software, in questo caso di un'applicazione Android.
Il plugin Android gestisce le risorse in progetti multipli con il meccanismo delle dependency, è quindi possibile ospitare all'interno dello stesso progetto: Il progetto principale, con un proprio file `.gradle`, ha la facoltà di integrare funzionalità di altri progetti (o *submodule*) ciascuno dei quali subisce una build in base al proprio file `.gradle`. Si posono integrare componenti esclusivamente Java come *java projects* (che vengono introdotti sotto forma di file `.jar`) oppure è possibile integrare progetti sotto forma di *library projects* (che vengono introdotti come package `.aar`) i quali possono ospitare anche codice compilato, come ad esempio le librerie shared di C in formato `.so`.

Il *submodule* per la shared library dlib ospita la libreria compilata per le due diverse architetture armeabi-v7a e arm64-v8a quindi è possibile eseguire l'applicazione su gran parte dei dispositivi Android in commercio, compresi quelli di ultima generazione dotati di architettura a 64 bit. La componente Java di questo *submodule* costituisce il cuore del dialogo tra la libreria compilata e il codice dell'app, e le sue componenti principali sono descritte nei paragrafi seguenti.

Classe HeadPoseGaze La classe HeadPoseGaze.java è un oggetto Java "classico" che rappresenta il dato di ciascuna rilevazione, con i campi *yaw*, *pitch* e *roll* porta con sé l'informazione derivante dall'elaborazione di uno specifico frame. Istanze di oggetti di questa classe vengono creati direttamente dal codice C++, il quale popola un array dinamico di tipo ArrayList per ogni frame elaborato, ciascun elemento conterrà l'elaborazione dei dati corrispondenti ad una faccia presente nel frame in analisi.

Classe HeadPoseDetector La classe HeadPoseDetector.java si occupa del caricamento della libreria tramite System.loadLibrary(), dell'inizializzazione (con jniInit()) e deinizializzazione (con jniDeinit()) dell'oggetto C++ che esegue la rilevazione e dialoga con esso fornendo il frame da processare e ricavando i risultati della rilevazione.
L'inizializzazione include la scelta del modello di *face shape* da utilizzare con il detector, il metodo di soluzione del problema PnP, e i parametri intrinseci della camera che possono essere caricati tramite una coppia di appositi file di configurazione in formato `.xml` permettendo l'utilizzo del sistema anche su altri dispositivi che possiedono parametri intrinseci differenti.

Il *submodule* app

La componente del progetto interamente realizzata in Java si occupa di costruire l'interfaccia, inizializzare le componenti hardware, gestire il flusso di informazioni e presentare

[11] https://tools.android.com/

i risultasti dell'elaborazione in real-time.

Activity e classi di utilità L'activity MainActivity.java dell'applicazione è quella che si apre all'avvio dell'app, propone all'utente di selezionare quale metodo di risoluzione intende usare per la rilevazione, e se si vuole salvare i risultati della rilevazione in un opportuno file `.xml`. Il ruolo principale di questa componente è tuttavia il controllo delle necessarie autorizzazioni richieste dal sistema per poter accedere alle risorse hardware, così come il setup delle cartelle di utilità e verifica della possibilità di salvataggio dei file con le rilevazioni.

Acquisizione dei frame Il widget "launch" dell'applicazione invoca l'esecuzione dell'activity CamerActivity.java la quale verifica le autorizzazioni run-time presenti in Android 6.0 e successivi e ospita al proprio interno il componente CameraConnectionFragment.java il quale esegue il meccanismo fondamentale di acquisizione del frame.
Il metodo openCamera() accede alla risorsa hardware disponibile ne acquisisce il controllo tramite un semaforo Java, contestualmente a ciò invoca i metodi setUpCameraOutputs() e configureTransforms(). Il primo analizza la lista di dispositivi disponibili e seleziona la fotocamera frontale, ed esegue dei metodi accessori che determinano la massima risoluzione e l'aspect ratio da adottare per i frame, successivamente controlla se sono disponibili parametri di calibrazione intrinseci della camera e qualora non fossero presenti esegue un caricamento da file `.xml`, in alternativa l'algoritmo utilizza dei valori di default. Il secondo metodo menzionato applica una trasformazione all'immagine in preview affinché essa mostri all'utente un'immagine specchiata e non un'immagine fedele alla realtà (ossia l'utente che ruota la propria testa verso la propria destra viene rappresentato nella preview come se stesse compiendo una rotazione verso la propria sinistra). Successivamente a questo, il metodo createCameraPreviewSession() istanzia un'apposita SurfaceView che ospita i frame non elaborati provenienti dal *live feed* della fotocamera.

Elaborazione Il Fragment CameraConnectionFragment è responsabile del passaggio di informazioni al codice C++ in esecuzione, per fare ciò si serve di un apposito *listener* chiamato OnGetImageListener il quale viene inizializzato una volta pronta la pipeline di immagini. Questa classe implementa l'interfaccia OnImageAvailableListener, la componente che agisce non appena un frame proveniente dalla camera è disponibile.
Il listener viene associato al flusso di immagini della camera, una volta chiamato il suo metodo initialize() esso inizializza la il detector ed il suo metodo onImageAvailable() viene invocato ogni qualvolta un frame è disponibile. Per ovvie ragioni l'elaborazione non è in grado di considerare l'enorme mole di singole immagini che giungono al listener, il quale scarta tutti gli input una volta che ne possiede uno da elaborare.

Per ciascun frame è necessario eseguire una traduzione online dello spazio di colori dell'immagine, in Android infatti è predefinita la codifica `YUV420`[12] quando si utilizza

[12]https://developer.android.com/reference/android/graphics/ImageFormat.html

un ImageReader mentre il formato necessario per eseguire le elaborazioni richiede una codifica `ARGB8888`, una volta compiuta questa conversione il frame viene processato con il metodo HeadPoseDetector.bitmapDetection() il quale è responsabile anche di disegnare gli artefatti facciali in corrispondenza dei punti rilevati. Quest'ultimo metodo ritorna contestualmente anche l'ArrayList di oggetti HeadPoseGaze contenenti i dati con le rilevazioni, che vengono mostrati all'utente.

Interfaccia grafica Nell'attuale configurazione l'utente osserva sul display un flusso continuo proveniente dalla fotocamera e un flusso secondario, oggetto dell'elaborazione, in una finestra separata in cui osserva anche il disegno della rilevazione. Viene mostrato in output anche l'insieme di parametri intrinseci caricato, così come i vari angoli ricavati in ogni elaborazione del frame.
La finestra secondaria è fluttuante e permette di essere spostata per meglio osservare il proprio posizionamento, la dimensione della stessa è determinata in funzione dello schermo affinché essa assuma sempre proporzioni adeguate a lasciare spazio alle immagini del flusso continuo rimanendo comunque visibile da una certa distanza.

Capitolo 5

Validazione dei risultati

La fase di validazione ha il compito di stabilire la precisione delle misurazioni effettuate dal dispositivo, confrontandole con un valore di verità sufficientemente preciso ed affidabile.
La configurazione hardware della scena di validazione prevede:

1. Il sistema di *motion capture* Phasespace Impulse, il quale si serve di una serie di marker attivi di cui traccia il movimento ed esegue la registrazione delle posizioni nello spazio tridimensionale

2. Il dispositivo utilizzato per la validazione dei risultati è Huawei P9 Lite (VNS-L31) le cui specifiche salienti sono:

   OS Android OS, v6.0 (Marshmallow)

   Chipset HiSilicon Kirin 650

   CPU Octa-core (4x2.0 GHz Cortex-A53 & 4x1.7 GHz Cortex-A53)

   GPU Mali-T830MP2

   Camera frontale 8MP

3. Il manichino per rilevazioni acustiche KEMAR [13]

5.1 Setup di validazione

Il manichino KEMAR è stato dotato di 5 LED posizionati sulla testa e giacenti sul piano frontale mentre il dispositivo è stato dotato di 4 LED posizionati sul lato posteriore. Il manichino è stato posizionato ad una distanza di circa 60cm (corrispondente ad una misura verosimile della lunghezza di un braccio) dal dispositivo, il quale è stato alloggiato su un treppiede di tipo fotografico, avendo cura di allineare i due componenti opportunamente in altezza ed allineando la fotocamera del dispositivo affinché puntasse al meglio verso il viso del manichino. Mantenendo fissa la posizione del dispositivo e ruotando sul proprio asse verticale il manichino KEMAR è stata simulata la rotazione di posizione,

in modo da mantenere inalterate il più possibile le proprietà della scena di ripresa.

Il sistema di motion capture stima la posizione dei singoli marker, tramite le quali un software in C++ appositamente sviluppato ricava una modellazione geometrica di una superficie giacente sui vertici definiti dai marker e ne estrae un vettore normale. Ricavando le coordinate polari del vettore ottenuto è stato possibile confrontare le due angolazioni di elevazione e azimuth del dispositivo e del manichino e tali informazioni sono state confrontate con le registrazioni prodotte contestualmente dall'applicazione.

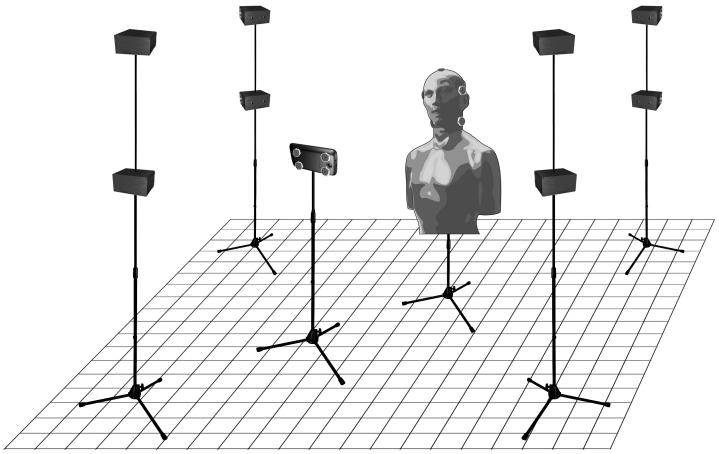


Figura 5.1: Rappresentazione della scena di validazione

5.2 Risultati dei test

Non essendo possibile stabilire un insieme di angoli fissi di rotazione, sono state eseguite delle misurazioni facendo ruotare il manichino in posizioni arbitrarie cercando di distribuire le misurazioni opportunamente sull'asse azimutale.
Per ogni angolazione scelta sono stati scelti 3 frame estratti casualmente dalla registrazione del motion capture, per ciascuno sono state calcolate le angolazioni del manichino e del dispositivo e si è ricavata una media delle angolazioni rilevate e tale media è stata impostata come ground truth della rilevazione.
Al fine di valutare la precisione di ogni singolo algoritmo disponibile, sono state eseguite rilevazioni per ciascuna soluzione.
Le rilevazioni ottenute con l'applicazione (e da questa registrate in un file `.xml`) sono state acquisite tramite fogli di calcolo, si è ricavata una stima della media e della deviazione standard di tali dati, che sono poi stati confrontati con il valore di ground truth corrispondente. Analogamente, dalle stesse rilevazioni sono state ricavate le misure di performance media in termini di tempo di esecuzione.

Tabella 5.1: Errori assoluti di rilevazione (deg)

Ground Truth		ITERATIVE		P3P		EPNP	
Yaw	Pitch	Yaw	Pitch	Yaw	Pitch	Yaw	Pitch
-47.06	-3.11	2.93	7.42	2.85	2.31	1.79	8.02
-37.06	-3.15	8.59	5.66	15.34	1.10	8.64	5.39
-18.38	-6.63	1.61	8.27	6.48	1.42	0.05	10.17
-8.34	-6.27	6.39	7.25	7.70	2.31	10.63	9.30
-2.52	-0.80	3.20	3.84	2.92	4.07	7.10	12.50
0.12	-5.81	2.08	6.79	0.77	1.85	2.16	8.84
-7.24	-6.38	2.93	7.42	2.85	2.31	1.79	8.02
19.64	-6.14	7.92	7.66	9.62	3.22	3.55	7.64
23.56	-6.45	6.18	9.85	9.05	2.01	3.33	8.69
34.76	-1.98	10.06	4.68	16.39	1.53	10.17	3.86
37.39	-0.66	11.12	3.21	17.13	2.39	12.29	3.20
RMS Err (deg)		**6.56**	**6.82**	**9.99**	**2.36**	**6.92**	**8.20**
Avg Ex Time (s)		**0.475**		**0.474**		**0.474**	

Come possiamo notare i risultati ottenuti presentano aspetti positivi e aspetti negativi, a livello qualitativo possiamo affermare che impiegare il metodo *P3P* porta a risultati meno rumorosi e più robusti. Le altre due soluzioni osservando i singoli valori, in particolare *EPnP*, si nota una più marcata instabilità tra frame contigui nelle rilevazioni, a parità di scena (si vedano le Figg. 5.2 e 5.3). La ragione di tale instabilità va ricercata nell'assenza di tracking nell'utilizzo dell'algoritmo ovvero il mancato sfruttamento dell'informazione pregressa della rilevazione, "slegando" ciascun frame dai precedenti e dai successivi.

Per quanto concerne il tempo di esecuzione, esso si mantiene per tutte le soluzioni al di sotto dei 500ms portando l'applicazione a superare i 2 FPS (*frames per second*) permettendo un agevole utilizzo da parte dell'utilizzatore.

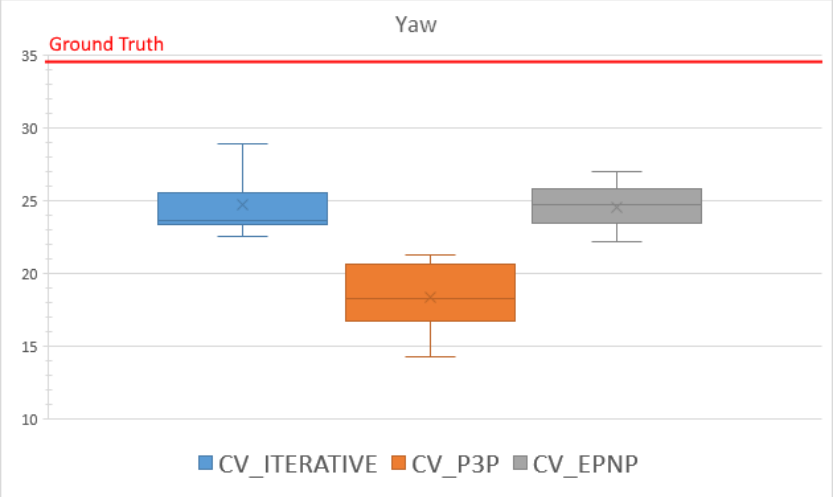


Figura 5.2: Distribuzione dei valori appartenenti ad una rilevazione di yaw

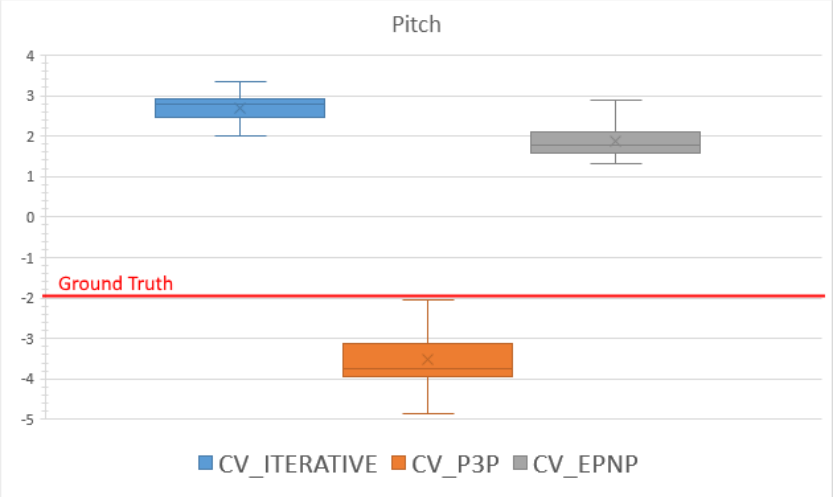


Figura 5.3: Distribuzione dei valori appartenenti ad una rilevazione di pitch

Capitolo 6

Conclusioni

6.1 Miglioramenti all'implementazione

La versione prodotta in questo lavoro e utilizzata per effettuare la validazione presenta alcune limitazioni di utilizzo, al netto delle limitazioni prestazionali.
Sviluppare questo progetto ha posto questioni non banali e sfide eterogenee e complesse, ed in alcuni casi le soluzioni ai problemi incontrati rimangono ancora una questione aperta.

6.1.1 Training

Già durante lo sviluppo di una soluzione è emerso il problema derivante dal ridotto range di pose rilevabili dall'algoritmo per la face detection di profili facciali e la conseguente shape prediction. La ragione alla base di questa ridotta funzionalità va ricercata nella natura dell'applicazione originale, che non richiede capacità di detection particolarmente accentuate e perciò ha previsto un training su un insieme relativamente ristretto di pose.
Nella parte finale dello sviluppo del progetto è stata posta molta attenzione a questo aspetto, poiché ampliare le capacità di rilevazione dell'algoritmo è un requisito desiderabile per l'applicazione, che sarebbe così in grado di acquisire informazioni acustiche da un insieme maggiore di posizioni e perciò si riuscirebbe a produrre una rilevazione più completa ed accurata.
La suite `dlib` fornisce tutti gli strumenti per realizzare un training per la detection di oggetti di tipo arbitrario così come per eseguire il training della shape prediction. In quest'ottica sono stati analizzati i diversi database contenenti informazioni quali bounding box e annotazioni dei landmark facciali, la scelta è ricaduta sul database AFLW il quale presenta un insieme completo per quanto riguarda le variazioni di illuminazione, posa e soggetti e sufficientemente ampio (si veda la Sottosez. 3.1.4).
Il dataset del database AFLW presenta le proprie informazioni sotto forma di un database SQL, perciò sono stati sviluppati software a supporto dell'estrazione dei dati dal database al fine di produrre un dataset processabile dagli algoritmi ed il cui formato è documentato nella libreria. Una cospicua quota di tempo è stata necessaria per prepa-

rare i dati, in qualche caso rumorosi, e per costruire dataset che fossero sufficientemente rappresentativi di ogni posa che si intendeva rilevare e privi di qualsivoglia imprecisione, come specificato dall'autore.
In secondo luogo è stato necessario ottenere accesso alle risorse di calcolo di ateneo dato che il training di support vector machines per questa tipologia di algoritmi richiede un ampio insieme di immagini e conseguentemente un'ampia disponibilità di memoria e di risorse di elaborazione.
È stato portato a termine con successo un primo training per la rilevazione delle aree facciali a tutte le pose comprese in [-90°,+90°] e perciò è stata completata la fase di face detection, tuttavia nonostante gli sforzi profusi per il secondo training non è stato possibile ottenere una shape prediction sufficientemente accurata.

Eseguire il training è stato un compito arduo da svolgere, dovendo dapprima scegliere ed ottenere un dataset adeguato, sviluppare software C++ a supporto per l'estrazione di informazioni, ricavare un insieme rappresentativo del dataset, rimuoverne il rumore verificando manualmente l'accuratezza e, probabilmente l'ostacolo più difficile, stimare un insieme di parametri dell'algoritmo ricavabili solo sperimentalmente ripetendo numerosi tentativi. Tutto questo, assieme alla mancanza di esperienza, ha giocato a sfavore di un esito che si è rivelato solo in parte positivo.
È stata dimostrata tuttavia la grande flessibilità che questa componente software introduce, permettendo di adattare una soluzione adeguata al problema che si intende risolvere con relativamente poco sforzo.

Un interrogativo rimane aperto in questo ambito, essendo questo un requisito importante si deve indagare ulteriormente e cercare di produrre una shape prediction precisa, dato che eseguendo questa prima parte del lavoro si è dimostrata la bontà di questo approccio.

6.1.2 Prestazioni

In termini di tempo di esecuzione le prestazioni sono pienamente soddisfacenti, riuscendo a superare i 2 frame per second rendendo l'applicazione utilizzabile poiché a tale velocità il meccanismo di feedback del sistema è accettabile, ovvero il sistema riporta in output i cambiamenti di posa a velocità sufficiente affinché l'utilizzatore possa avere il controllo del proprio movimento.
Il tempo di esecuzione del processing di un frame è stabilmente al di sotto dei 500ms, in cui una grande quota è dovuta all'esecuzione dell'algoritmo di face detection mentre il tempo di esecuzione della rilevazione dei landmark si attesta, secondo alcune prove sperimentali e dalla documentazione del paper di riferimento, intorno ai 10ms su un dispositivo mobile. Per questo è necessario concentrare gli sforzi nel miglioramento della prima parte, vincolando maggiormente i requisiti di generalità dell'algoritmo HOG [29], ad esempio considerando l'approccio a multi scala nella rilevazione si può inserire un vincolo di minima dimensione di un viso rilevabile rimuovendo in questo modo inutili

computazioni.

L'accuratezza, come evidenziato nel Cap. 5 presenta alcuni aspetti negativi, e soprattutto per quanto riguarda le pose più estreme il margine di errore è sensibile. La ragione di questo comportamento risiede nella natura dell'algoritmo stesso e del training che è stato eseguito, è del tutto probabile che l'assenza di training su pose estreme renda particolarmente difficile la rilevazione dei landmark facciali in tali configurazioni, anche a causa della probabile scarsa precisione della rilevazione della bounding box per la faccia che viene eseguita precedentemente.
Inoltre, da verifiche puramente sperimentali (seppur senza validazione) è emerso che il degrado della precisione della stima derivi anche dalla scala dell'oggetto all'interno dell'immagine da processare, in termini pratici la distanza del soggetto dal dispositivo (pari a circa 60cm) fa sì che il viso ritratto sia in scala ridotta e questo potrebbe influenzare l'accuratezza. Una contromisura potrebbe essere quella di eseguire una scalatura della regione dell'immagine corrispondente alla faccia rilevata prima di eseguire la rilevazione dei landmark facciali.

In generale, migliorare l'accuratezza è possibile impiegando un training più completo, come già citato, e cercando di introdurre una qualche forma di tracking della forma facciale, considerato che la variazione di posa non è "brusca" ma è invece frutto di un movimento "fluido" dell'utente. A supportare questa ipotesi vi è anche il documento di riferimento [32] in cui si afferma che utilizzare il tracking della posa potrebbe incrementare la precisione.

6.2 Usabilità dell'applicazione

Un aspetto che richiede uno studio approfondito è la parte di interazione dell'utilizzatore con l'applicazione. Il lavoro già svolto per la versione precedente rappresenta un ottimo punto di partenza per progettare un'interfaccia di semplice utilizzo senza alcuna particolare procedura di addestramento.
Per costruire una "griglia sferica" virtuale e per far si che l'utente sia in grado di "esplorarla" è necessario produrre un meccanismo di feedback sonoro e aptico affinché esso si possa spostare agevolmente da una posizione all'altra. Ideare un tale meccanismo è necessario poiché l'utilizzatore nella maggior parte dei casi non può mantenere un controllo visivo sul dispositivo essendo costretto a mantenere la testa in una posizione frontale, in questi casi si possono utilizzare dei segnali di *beep* ravvicinati in base alla distanza dall'obbiettivo, affiancandoli ad un analogo sistema che sfrutta la vibrazione del dispositivo: i *beep* potrebbero segnalare la distanza rispetto all'angolo obbiettivo di elevazione, mentre le vibrazioni segnalerebbero la distanza dall'angolo obbiettivo sul piano orizzontale. Come già fatto nella versione precedente si deve includere la parte di riproduzione del suono *sweep* in frequenza ed includere la procedura di acquisizione del segnale utilizzando il PC a supporto.

Una volta apportate queste migliorie è necessario eseguire una validazione sul campo che cerchi di stabilire se un'interfaccia così costruita sia semplice da utilizzare e faciliti effettivamente la procedura di puntamento.

6.3 Considerazioni finali

La crescita che ho avuto a livello personale e di competenze tecniche affrontando questa sfida non è facilmente descrivibile con un insieme di aggettivi, una certezza granitica è che la realizzazione di questo progetto mi ha posto davanti ad un completo scenario di sviluppo software di livello ingegneristico, cominciando dall'analisi dei requisiti, la formulazione di specifiche funzionali, la valutazione delle proprietà prestazionali richieste e la conseguente impostazione di un processo di sviluppo, la scelta degli strumenti di supporto, la scelta della piattaforma, la scelta delle componenti principali di tipo software, l'esecuzione dello sviluppo, la valutazione della soluzione introdotta ed i ragionamenti sui possibili miglioramenti corredati da un'analisi dettagliata dei passaggi fondamentali a livello algoritmico.

Non è mancato nulla dal punto di vista dell'impiego delle competenze sviluppate nel mio percorso accademico quali ad esempio lo sviluppo in Java, l'utilizzo delle competente di database con SQL, ed un aspetto importante ed estremamente formativo è l'insieme di ulteriori competenze che ho sviluppato nei mesi di sviluppo quali ad esempio i princípi di machine learning, lo studio approfondito dello sviluppo in C++, lo sviluppo Android con NDK, lo studio delle soluzioni per la computer vision applicate alla human computer interaction, ecc. Inoltre, ho potuto accompagnare tutto ciò con un approfondimento dettagliato dello studio del sistema uditivo umano, come esso funziona, con particolare attenzione al rendering di suono tridimensionale.
Non si sono fatte desiderare nemmeno le difficoltà, in qualche modo accentuate dalla relativa "gioventù" di soluzioni di computer vision in ambito mobile, unite alla mia ridotta familiarità nell'affrontare questa tipologia di problemi. In qualche caso le problematiche sono state anche insite strutturalmente nel processo di sviluppo, ad esempio ho impiegato una cospicua quota di tempo per realizzare training fallimentari su macchine del centro di calcolo che alla fine non si sono rivelate adeguate a compiere il task. Inoltre come già citato in precedenza, mi sono confrontato per la prima volta con l'esecuzione di training di tipo sperimentale, la mia inesperienza ha giocato certamente un ruolo fondamentale nella quantità di tempo che ho dovuto investire per eseguire l'insieme di esperimenti ripetuti necessari per arrivare ad una soluzione sufficientemente precisa.

Rimangono numerose questioni aperte su come si potrebbe migliorare questa applicazione in primo luogo completando l'interfaccia utente ed incrementando l'accuratezza delle rilevazioni. Solo investendo altro tempo sarà possibile capire se è opportuno migliorare ulteriormente l'implementazione attuale, tuttavia il contributo realmente di valore è che eseguire un porting di un algoritmo di questo tipo è possibile, perciò non è preclusa la possibilità di utilizzare altri algoritmi più performanti e complessi, considerando che

anche durante lo stesso sviluppo del progetto sono emersi approcci molto interessanti per questo tipo di applicazioni.

La ricerca in Sound e Music Computing rimane estremamente interessante per quanto riguarda la produzione di sistemi per il rendering tridimensionale del suono e vedrà sicuramente un grande sviluppo negli anni a venire, la tecnologia sta compiendo passi da gigante e la diffusione di massa di dispositivi ad altissima resa di realtà virtuale pervaderanno la nostra quotidianità, e questo è solo l'inizio.

Appendici

Appendice A

Codice

A.1 Configurazione della build

I seguenti file contengono le informazioni fornite al sistema di build `ndk-build` di Android per generare le librerie.

Listing A.1: Application.mk

```
1  NDK_TOOLCHAIN_VERSION := clang
2  APP_ABI := armeabi-v7a arm64-v8a
3  # armeabi-v7a x86 arm64-v8a x86_64
4  APP_CPPFLAGS := -std=c++11 -frtti -fexceptions
5  APP_PLATFORM := android-8
6  APP_STL := gnustl_static
```

Listing A.2: Android.mk

```
1  LOCAL_PATH := $(call my-dir)
2
3  # Define root directories of miniglog, opencv and dlib (you should have
       downloaded them and ectracted somewhere in your machine)
4  MINIGLOG_DIR := C:\Users\Filippo\VS2015_Workspace\miniglog
5  OPENCV_ANDROID_SDK := C:\Users\Filippo\VS2015_Workspace\OpenCV-android-sdk
6  DLIB_DIR := C:\Users\Filippo\Desktop\dlib-android\jni\dlib
7
8  #dlib static library
9  include $(CLEAR_VARS)
10 LOCAL_MODULE := dlib
11 LOCAL_C_INCLUDES := $(DLIB_DIR)
12 LOCAL_EXPORT_C_INCLUDES := $(DLIB_DIR)
13
14 LOCAL_SRC_FILES += \
15      $(DLIB_DIR)/dlib/threads/threads_kernel_shared.cpp \
16      $(DLIB_DIR)/dlib/entropy_decoder/entropy_decoder_kernel_2.cpp \
17      $(DLIB_DIR)/dlib/base64/base64_kernel_1.cpp \
18      $(DLIB_DIR)/dlib/threads/threads_kernel_1.cpp \
19      $(DLIB_DIR)/dlib/threads/threads_kernel_2.cpp
20
```

```
21 include $(BUILD_STATIC_LIBRARY)
22
23 #miniglog static library
24 include $(CLEAR_VARS)
25 LOCAL_MODULE := miniglog
26 LOCAL_EXPORT_C_INCLUDES := $(MINIGLOG_DIR)
27 LOCAL_C_INCLUDES := $(MINIGLOG_DIR)
28 LOCAL_SRC_FILES := $(MINIGLOG_DIR)/glog/logging.cc
29
30 include $(BUILD_STATIC_LIBRARY)
31
32 #user defined jni shared library
33 include $(CLEAR_VARS)
34 OpenCV_INSTALL_MODULES := on
35 OPENCV_CAMERA_MODULES := off
36 OPENCV_LIB_TYPE := STATIC
37 include $(OPENCV_ANDROID_SDK)/sdk/native/jni/OpenCV.mk
38
39 LOCAL_MODULE := head_pose_det
40
41 LOCAL_C_INCLUDES += \
42             $(OPENCV_ANDROID_SDK)/sdk/native/jni/include
43
44 LOCAL_SRC_FILES += \
45     jni_head_pose_det.cpp \
46     imageutils_jni.cpp \
47     common/rgb2yuv.cpp \
48     common/yuv2rgb.cpp \
49     common/bitmap2mat2bitmap.cpp
50
51 LOCAL_LDLIBS += -lm -llog -ldl -lz -ljnigraphics -latomic
52
53 # import static libraries
54 LOCAL_STATIC_LIBRARIES += dlib
55 LOCAL_STATIC_LIBRARIES += miniglog
56
57 ifeq ($(TARGET_ARCH_ABI), armeabi-v7a)
58     LOCAL_ARM_MODE := arm
59         LOCAL_ARM_NEON := true
60     #LOCAL_CFLAGS= -march=armv7-a -mfloat-abi=softfp -mfpu=neon
61     #LOCAL_LDFLAGS= -march=armv7-a -Wl,--fix-cortex-a8
62 endif
63
64 include $(BUILD_SHARED_LIBRARY)
```

A.2 Componente JNI

Il seguente listato di codice contiene il programma JNI per la comunicazione tra la parte Java ed il software nativo.

Listing A.3: jni_head_pose_det.cpp

```
1  #include <android/bitmap.h>
2  #include <common/bitmap2mat2bitmap.h>
3  #include <jni.h>
4  #include <glog/logging.h>
5  #include "head_pose_estimation.cpp"
6  #include "LinearMath/Matrix3x3.h"
7
8  using namespace std;
9  using namespace cv;
10
11 namespace {
12   std::shared_ptr<HeadPoseEstimation> gHeadPoseEstimationPtr;
13 }
14
15 #ifdef __cplusplus
16 extern "C" {
17 #endif
18
19 static jclass HeadPoseGaze;
20   static jmethodID HeadPoseGazeConstructor;
21
22 static jclass ArrayList;
23   static jmethodID ArrayListAdd;
24
25 jint JNI_OnLoad(JavaVM* vm, void* reserved) {
26   JNIEnv* env = NULL;
27
28   if (vm->GetEnv((void**)&env, JNI_VERSION_1_6) != JNI_OK) {
29     return JNI_ERR;
30   }
31
32   return JNI_VERSION_1_6;
33 }
34
35 // Macro to define correctly native method names
36 #define DLIB_JNI_METHOD(METHOD_NAME) \
37   Java_edu_unipd_dei_dlib_HeadPoseDetector_##METHOD_NAME
38
39 jint JNIEXPORT JNICALL DLIB_JNI_METHOD(jniBitmapExtractFaceGazes)(JNIEnv*
      env, jobject thiz, jobject bitmap, jobject gazesList) {
40
41   if (gHeadPoseEstimationPtr) {
42     cv::Mat rgbaMat;
43     cv::Mat bgrMat;
44     jnicommon::ConvertBitmapToRGBAMat(env, bitmap, rgbaMat, true, false,
          false);
45     cv::cvtColor(rgbaMat, bgrMat, cv::COLOR_RGBA2BGR);
46
47     jint size = gHeadPoseEstimationPtr->detect(bgrMat);
48     LOG(INFO) << "Number of faces detected: " << size;
49
50     auto poses = gHeadPoseEstimationPtr->poses();
51
```

```
52      int i = 0;
53      jobject gaze_found = NULL;
54      LOG(INFO) << "{";
55      for(auto pose : poses) {
56          pose = pose.inv();
57
58          double raw_yaw, raw_pitch, raw_roll;
59          tf::Matrix3x3 mrot(
60                  pose(0,0), pose(0,1), pose(0,2),
61                  pose(1,0), pose(1,1), pose(1,2),
62                  pose(2,0), pose(2,1), pose(2,2));
63          mrot.getRPY(raw_roll, raw_pitch, raw_yaw);
64
65          raw_roll = raw_roll - M_PI/2;
66          raw_yaw = raw_yaw + M_PI/2;
67
68          double yaw, pitch, roll;
69
70          roll = raw_pitch;
71          yaw = raw_yaw;
72          pitch = -raw_roll;
73
74          i++;
75          // Call add method on an object created from another method call
76          gaze_found = env->NewObject(HeadPoseGaze, HeadPoseGazeConstructor,
                gHeadPoseEstimationPtr->todeg(yaw), gHeadPoseEstimationPtr->
                todeg(pitch), gHeadPoseEstimationPtr->todeg(roll));
77          env->CallBooleanMethod(gazesList, ArrayListAdd, gaze_found);
78      }
79
80      // Produce the bitmap to display
81      cv::Mat rgbaResultMat;
82      cv::cvtColor(gHeadPoseEstimationPtr -> resultMat, rgbaResultMat, cv::
            COLOR_BGR2RGBA);
83      jnicommon::ConvertRGBAMatToBitmap(env, bitmap, rgbaResultMat, true);
84
85      return JNI_OK;
86    } else return JNI_ERR;
87  }
88
89  jint JNIEXPORT JNICALL DLIB_JNI_METHOD(jniInit)(JNIEnv* env, jobject thiz,
        jstring landmarkPath, jint mode, jfloat fx, jfloat fy, jfloat cx,
        jfloat cy, jfloat k1, jfloat k2, jfloat p1, jfloat p2, jfloat k3) {
90    // Initialize a new estimator if it's not already there
91    if (!gHeadPoseEstimationPtr) {
92      const char* landmarkmodel_path = env->GetStringUTFChars(landmarkPath,
            0);
93      gHeadPoseEstimationPtr = std::make_shared<HeadPoseEstimation>(
            landmarkmodel_path, mode, fx, fy, cx, cy, k1, k2, p1, p2, k3);
94      env->ReleaseStringUTFChars(landmarkPath, landmarkmodel_path);
95    }
96
97    // Initialize references to classes and methods
```

```
98      jclass HeadPoseGaze_local = env->FindClass("edu/unipd/dei/dlib/
            HeadPoseGaze");
99      if(HeadPoseGaze_local == NULL) {
100         return JNI_ERR;
101     }
102
103     jclass ArrayList_local = env->FindClass("java/util/ArrayList");
104     if(ArrayList_local == NULL) {
105         return JNI_ERR;
106     }
107
108     // Obtain global refs...
109     HeadPoseGaze = reinterpret_cast<jclass>(env->NewGlobalRef(
            HeadPoseGaze_local));
110     HeadPoseGazeConstructor = env->GetMethodID(HeadPoseGaze, "<init>", "(DDD)
            V");
111     if(HeadPoseGazeConstructor == NULL) {
112         return JNI_ERR;
113     }
114
115     ArrayList = reinterpret_cast<jclass>(env->NewGlobalRef(ArrayList_local));
116     ArrayListAdd = env->GetMethodID(ArrayList, "add", "(Ljava/lang/Object;)Z"
            );
117     if(ArrayListAdd == NULL) {
118         return JNI_ERR;
119     }
120
121     // ...delete local refs
122     env->DeleteLocalRef(HeadPoseGaze_local);
123     env->DeleteLocalRef(ArrayList_local);
124
125     return JNI_OK;
126 }
127
128 jint JNIEXPORT JNICALL DLIB_JNI_METHOD(jniDeInit)(JNIEnv* env, jobject thiz
        ) {
129     gHeadPoseEstimationPtr.reset();
130     env->DeleteGlobalRef(HeadPoseGaze);
131     env->DeleteGlobalRef(ArrayList);
132
133     return JNI_OK;
134 }
135
136 #ifdef __cplusplus
137 }
138 #endif
```

A.3 Head pose estimation

I seguenti listati di codice contengono la parte software in C++ che esegue la rilevazione delle facce, ne rileva i landmark, e ricava la soluzione al problema PnP in base ai valori

di inzializzazione.

Listing A.4: head_pose_estimation.hpp

```
1  #ifndef _HEAD_POSE_ESTIMATION
2  #define _HEAD_POSE_ESTIMATION
3
4  #include <opencv2/core/core.hpp>
5  #include <dlib/opencv.h>
6  #include <dlib/image_processing.h>
7  #include <dlib/image_processing/frontal_face_detector.h>
8
9  #include <vector>
10 #include <array>
11 #include <string>
12
13 // 3D rigid structure points
14 const static cv::Point3f P3D_SELLION(0., 0.,0.);
15 const static cv::Point3f P3D_RIGHT_EYE(-20., -65.5,-5.);
16 const static cv::Point3f P3D_LEFT_EYE(-20., 65.5,-5.);
17 const static cv::Point3f P3D_RIGHT_EAR(-100., -77.5,-6.);
18 const static cv::Point3f P3D_LEFT_EAR(-100., 77.5,-6.);
19 const static cv::Point3f P3D_NOSE(21.0, 0., -48.0);
20 const static cv::Point3f P3D_STOMMION(10.0, 0., -75.0);
21 const static cv::Point3f P3D_MENTON(0., 0.,-133.0);
22
23 static const int MAX_FEATURES_TO_TRACK=100;
24
25 // Interesting facial features with their landmark index
26 enum FACIAL_FEATURE {
27     NOSE=30,
28     RIGHT_EYE=36,
29     LEFT_EYE=45,
30     RIGHT_SIDE=0,
31     LEFT_SIDE=16,
32     EYEBROW_RIGHT=21,
33     EYEBROW_LEFT=22,
34     MOUTH_UP=51,
35     MOUTH_DOWN=57,
36     MOUTH_RIGHT=48,
37     MOUTH_LEFT=54,
38     SELLION=27,
39     MOUTH_CENTER_TOP=62,
40     MOUTH_CENTER_BOTTOM=66,
41     MENTON=8
42 };
43
44 const static int MODE_ITERATIVE = 0;
45 const static int MODE_P3P = 1;
46 const static int MODE_EPNP = 2;
47
48 typedef cv::Matx44d head_pose;
49
50 class HeadPoseEstimation {
```

```
51
52 public:
53     HeadPoseEstimation(const std::string& face_detection_model = "
           shape_predictor_68_face_landmarks.dat",
54         int mode=MODE_ITERATIVE,
55         float fx = 0,
56         float fy = 0,
57         float cx = 0,
58         float cy = 0,
59         float k1 = 0,
60         float k2 = 0,
61         float p1 = 0,
62         float p2 = 0,
63         float k3 = 0);
64
65     int detect(cv::Mat& image);
66
67     head_pose pose(size_t face_idx) const;
68
69     std::vector<head_pose> poses() const;
70
71     virtual inline double todeg(double rad) {  return rad * 180 / M_PI; }
72
73     cv::Matx33f cameraMatrix;
74     cv::Mat1f distCoeffs;
75
76     mutable cv::Mat resultMat;
77
78     int mode;
79
80 private:
81     dlib::cv_image<dlib::bgr_pixel> current_image;
82
83     dlib::frontal_face_detector detector;
84     dlib::shape_predictor pose_model;
85
86     std::vector<dlib::rectangle> faces;
87
88     std::vector<dlib::full_object_detection> shapes;
89
90     /** Return the point corresponding to the dictionary marker.
91     */
92     cv::Point2f coordsOf(size_t face_idx, FACIAL_FEATURE feature) const;
93
94     /** Returns true if the lines intersect (and set r to the intersection
95      *  coordinates), false otherwise.
96      */
97     bool intersection(cv::Point2f o1, cv::Point2f p1,
98                       cv::Point2f o2, cv::Point2f p2,
99                       cv::Point2f &r) const;
100 };
101
102 #endif // _HEAD_POSE_ESTIMATION
```

Listing A.5: head_pose_estimation.cpp

```
1  #include "head_pose_estimation.hpp"
2  #include <opencv2/calib3d/calib3d.hpp>
3
4  using namespace dlib;
5  using namespace std;
6  using namespace cv;
7
8  inline Point2f toCv(const dlib::point& p) {
9      return Point2f(p.x(), p.y());
10 }
11
12 HeadPoseEstimation::HeadPoseEstimation(const string& face_detection_model,
       int mod,
13     float fx, float fy, float cx, float cy,
14     float k1, float k2, float p1, float p2, float k3) {
15     // Load face detection and pose estimation models.
16     detector = get_frontal_face_detector();
17     deserialize(face_detection_model) >> pose_model;
18     mode = mod; // Set correct mode
19
20     // Set cameraMatrix
21     cv::Mat m = cv::Mat::zeros(3,3,CV_32F);
22     cameraMatrix = m;
23     cameraMatrix(0,0) = fx; // focalLengthX
24     cameraMatrix(1,1) = fy; // focalLengthY
25     cameraMatrix(0,2) = cx; // opticalCenterX
26     cameraMatrix(1,2) = cy; // opticalCenterY
27     cameraMatrix(2,2) = 1;
28
29     distCoeffs = (Mat1d(1, 5) << k1, k2, p1, p2, k3);
30 }
31
32 int HeadPoseEstimation::detect(cv::Mat& image) {
33     // If optical center contains default(=invalid values), use an estimate
           of them
34     if(cameraMatrix(0,0) == 0) {
35         cv::Mat m = cv::Mat::zeros(3,3,CV_32F);
36         cameraMatrix = m;
37         cameraMatrix(0,0) = 455.; // focalLength
38         cameraMatrix(1,1) = 455.; // focalLength
39         cameraMatrix(0,2) = image.cols / 2; // opticalCenterX
40         cameraMatrix(1,2) = image.rows / 2; // opticalCenterY
41         cameraMatrix(2,2) = 1;
42
43         distCoeffs = (Mat1d(1, 5) << 0, 0, 0, 0, 0);
44     }
45
46     // Check that the image is valid
47     if (image.empty()) return 0;
48
49     // Set as current image
50     current_image = dlib::cv_image<dlib::bgr_pixel>(image);
```

```
51      // Perform detection
52      faces = detector(current_image);
53      // Put the results into a collection, and update how many found
54      shapes.clear();
55      int count = 0;
56      for (auto face : faces){
57          shapes.push_back(pose_model(current_image, face));
58          count++;
59      }
60
61      // Get a clone to draw on
62      resultMat = image.clone();
63
64      // Draw lines for landmarks
65      auto color = Scalar(0,255,0);
66      for (unsigned long i = 0; i < shapes.size(); ++i)
67      {
68          const full_object_detection& d = shapes[i];
69
70          for (unsigned long i = 1; i <= 16; ++i)
71              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
72
73          for (unsigned long i = 28; i <= 30; ++i)
74              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
75
76          for (unsigned long i = 18; i <= 21; ++i)
77              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
78          for (unsigned long i = 23; i <= 26; ++i)
79              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
80          for (unsigned long i = 31; i <= 35; ++i)
81              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
82          line(resultMat, toCv(d.part(30)), toCv(d.part(35)), color, 2, CV_AA
                );
83
84          for (unsigned long i = 37; i <= 41; ++i)
85              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
86          line(resultMat, toCv(d.part(36)), toCv(d.part(41)), color, 2, CV_AA
                );
87
88          for (unsigned long i = 43; i <= 47; ++i)
89              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
90          line(resultMat, toCv(d.part(42)), toCv(d.part(47)), color, 2, CV_AA
                );
91
92          for (unsigned long i = 49; i <= 59; ++i)
93              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
```

```
                CV_AA);
94          line(resultMat, toCv(d.part(48)), toCv(d.part(59)), color, 2, CV_AA
                );
95
96          for (unsigned long i = 61; i <= 67; ++i)
97              line(resultMat, toCv(d.part(i)), toCv(d.part(i-1)), color, 2,
                    CV_AA);
98          line(resultMat, toCv(d.part(60)), toCv(d.part(67)), color, 2, CV_AA
                );
99      }
100
101     return count;
102 }
103
104 head_pose HeadPoseEstimation::pose(size_t face_idx) const {
105
106     std::vector<Point3f> head_points;
107     std::vector<Point2f> detected_points;
108
109     // Initializing the head pose 1m away, roughly facing the robot
110     // This initialization is important as it prevents solvePnP to find the
111     // mirror solution (head *behind* the camera)
112     Mat tvec = (Mat_<double>(3,1) << 0., 0., 1000.);
113     Mat rvec = (Mat_<double>(3,1) << 1.2, 1.2, -1.2);
114
115     if(mode == MODE_ITERATIVE) {
116         // List of 3D points
117         head_points.push_back(P3D_SELLION);
118         head_points.push_back(P3D_RIGHT_EYE);
119         head_points.push_back(P3D_LEFT_EYE);
120         head_points.push_back(P3D_RIGHT_EAR);
121         head_points.push_back(P3D_LEFT_EAR);
122         head_points.push_back(P3D_MENTON);
123         head_points.push_back(P3D_NOSE);
124         head_points.push_back(P3D_STOMMION);
125
126         // List of 2D points
127         detected_points.push_back(coordsOf(face_idx, SELLION));
128         detected_points.push_back(coordsOf(face_idx, RIGHT_EYE));
129         detected_points.push_back(coordsOf(face_idx, LEFT_EYE));
130         detected_points.push_back(coordsOf(face_idx, RIGHT_SIDE));
131         detected_points.push_back(coordsOf(face_idx, LEFT_SIDE));
132         detected_points.push_back(coordsOf(face_idx, MENTON));
133         detected_points.push_back(coordsOf(face_idx, NOSE));
134
135         // Stommion is the mean point between upper and lower lip, I must
                calculate it since there's not such landmark
136         auto stomion = (coordsOf(face_idx, MOUTH_CENTER_TOP) + coordsOf(
                face_idx, MOUTH_CENTER_BOTTOM)) * 0.5;
137         detected_points.push_back(stomion);
138
139         // Find the 3D pose of our head
140         solvePnP(head_points, detected_points,
```

```
141             cameraMatrix, distCoeffs,
142             rvec, tvec, true, SOLVEPNP_ITERATIVE);
143     } else if(mode == MODE_P3P) {
144         // List of 3D points
145         head_points.push_back(P3D_NOSE);
146         head_points.push_back(P3D_RIGHT_EAR);
147         head_points.push_back(P3D_LEFT_EAR);
148         head_points.push_back(P3D_MENTON);
149 
150         // List of 2D points
151         detected_points.push_back(coordsOf(face_idx, NOSE));
152         detected_points.push_back(coordsOf(face_idx, RIGHT_SIDE));
153         detected_points.push_back(coordsOf(face_idx, LEFT_SIDE));
154         detected_points.push_back(coordsOf(face_idx, MENTON));
155 
156         // Find the 3D pose of our head
157         solvePnP(head_points, detected_points,
158             cameraMatrix, distCoeffs,
159             rvec, tvec, true, SOLVEPNP_P3P);
160     } else if(mode == MODE_EPNP) {
161         // List of 3D points
162         head_points.push_back(P3D_SELLION);
163         head_points.push_back(P3D_RIGHT_EYE);
164         head_points.push_back(P3D_LEFT_EYE);
165         head_points.push_back(P3D_RIGHT_EAR);
166         head_points.push_back(P3D_LEFT_EAR);
167         head_points.push_back(P3D_MENTON);
168         head_points.push_back(P3D_NOSE);
169         head_points.push_back(P3D_STOMMION);
170 
171         // List of 2D points
172         detected_points.push_back(coordsOf(face_idx, SELLION));
173         detected_points.push_back(coordsOf(face_idx, RIGHT_EYE));
174         detected_points.push_back(coordsOf(face_idx, LEFT_EYE));
175         detected_points.push_back(coordsOf(face_idx, RIGHT_SIDE));
176         detected_points.push_back(coordsOf(face_idx, LEFT_SIDE));
177         detected_points.push_back(coordsOf(face_idx, MENTON));
178         detected_points.push_back(coordsOf(face_idx, NOSE));
179 
180         // Stommion is the mean point between upper and lower lip, I must
                calculate it since there's not such landmark
181         auto stomion = (coordsOf(face_idx, MOUTH_CENTER_TOP) + coordsOf(
                face_idx, MOUTH_CENTER_BOTTOM)) * 0.5;
182         detected_points.push_back(stomion);
183 
184         // Find the 3D pose of our head
185         solvePnP(head_points, detected_points,
186             cameraMatrix, distCoeffs,
187             rvec, tvec, true, SOLVEPNP_EPNP);
188     }
189 
190     Matx33d rotation;
191     Rodrigues(rvec, rotation);
```

```
192
193     head_pose pose = {
194         rotation(0,0),    rotation(0,1),    rotation(0,2),    tvec.at<
                double>(0)/1000,
195         rotation(1,0),    rotation(1,1),    rotation(1,2),    tvec.at<
                double>(1)/1000,
196         rotation(2,0),    rotation(2,1),    rotation(2,2),    tvec.at<
                double>(2)/1000,
197                   0,                0,                0,
                                                        1
198     };
199
200     // Istantiate head_points, reproject them with rvec and tvec, and draw
            them onto the resultMat
201     std::vector<Point2f> reprojected_points;
202     projectPoints(head_points, rvec, tvec, cameraMatrix, noArray(),
            reprojected_points);
203
204     for (auto point : reprojected_points) {
205         circle(resultMat, point,2, Scalar(0,255,255), 2);
206     }
207
208     // Istantiate axes, reproject them with rvec and tvec, and draw them
            onto the resultMat
209     std::vector<Point3f> axes;
210     axes.push_back(Point3f(0,0,0));
211     axes.push_back(Point3f(50,0,0));
212     axes.push_back(Point3f(0,50,0));
213     axes.push_back(Point3f(0,0,50));
214
215     std::vector<Point2f> projected_axes;
216     projectPoints(axes, rvec, tvec, cameraMatrix, noArray(), projected_axes
            );
217
218     line(resultMat, projected_axes[0], projected_axes[3], Scalar(255,0,0)
            ,2,CV_AA);
219     line(resultMat, projected_axes[0], projected_axes[2], Scalar(0,255,0)
            ,2,CV_AA);
220     line(resultMat, projected_axes[0], projected_axes[1], Scalar(0,0,255)
            ,2,CV_AA);
221
222     return pose;
223 }
224
225 std::vector<head_pose> HeadPoseEstimation::poses() const {
226     std::vector<head_pose> res;
227     for (auto i = 0; i < faces.size(); i++){
228         res.push_back(pose(i));
229     }
230     return res;
231 }
232
233 /** Return the point corresponding to the dictionary marker.
```

```
234 */
235 Point2f HeadPoseEstimation::coordsOf(size_t face_idx, FACIAL_FEATURE
        feature) const {
236     return toCv(shapes[face_idx].part(feature));
237 }
```

A.4 Elementi Java

Si riportano alcuni passaggi fondamentali della componente Java dell'applicazione Android.

Listing A.6: HeadPoseGaze.java

```
1  package edu.unipd.dei.dlib;
2
3  public class HeadPoseGaze {
4      private double[] ypr;
5
6      public HeadPoseGaze() {
7          ypr = new double[3];
8      }
9
10     public HeadPoseGaze(double y, double p, double r) {
11         ypr = new double[3];
12         ypr[0] = y;
13         ypr[1] = p;
14         ypr[2] = r;
15     }
16
17     // Convenience method to produce a new object (always calling it when
           adding a new item fro JNI!)
18     public static HeadPoseGaze newInstance(double y, double p, double r) {
19         return new HeadPoseGaze(y, p, r);
20     }
21
22     public void setGaze(double y, double p, double r) {
23         ypr[0] = y;
24         ypr[1] = p;
25         ypr[2] = r;
26     }
27
28     public double[] getGaze() {
29         return ypr;
30     }
31
32     // According to my new frame of refence, I must add 180 degrees to
           pitch and yaw in order to geto (0,0,0) for a front looking fcae.
33     public double getYaw() {
34         return ypr[0] - 180;
35     }
36
37     public double getPitch() {
```

```
38          return ypr[1] - 180;
39      }
40
41      public double getRoll() {
42          return ypr[2];
43      }
44
45      public String toString() {
46          return "(" + getYaw() + ", " + getPitch() + ", " + getRoll() + ")";
47      }
48  }
```

Listing A.7: HeadPoseDetector.java

```
1   package edu.unipd.dei.dlib;
2
3   import android.content.Context;
4   import android.graphics.Bitmap;
5   import android.support.annotation.NonNull;
6   import android.util.Log;
7
8   import java.util.ArrayList;
9
10  import hugo.weaving.DebugLog;
11
12  public class HeadPoseDetector {
13      private static final String TAG = "HeadPoseDetector";
14      protected static boolean initialized = false;
15      protected static boolean initializing = false;
16
17      static {
18          try {
19              System.loadLibrary("head_pose_det"); // Load native library
20          } catch (UnsatisfiedLinkError e) {
21              Log.d(TAG, "### Native library not found! ###");
22          }
23      }
24
25      public static void init(String model_path, int alg_mode, boolean
            use_custom, float[] intrinsics, float[] distortions) {
26          if (!initializing) {
27              Log.d(TAG, "*** Requested a fresh initialization with jniInit
                    () ***");
28              initializing = true;
29              if (jniInit(model_path, alg_mode, intrinsics[0], intrinsics[1],
                     intrinsics[2], intrinsics[3], distortions[0], distortions
                    [1], distortions[2], distortions[3], distortions[4]) == 0)
                    { // If all went ok, state as initialized true
30                  initializing = false;
31                  initialized = true;
32              } else {
33                  initializing = false;
34                  initialized = false;
35                  Log.d(TAG, "*** jniInit() ERROR ***");
```

```
36              }
37          } else {
38              Log.d(TAG, "␣***␣Requested␣initialization␣with␣jniInit()␣while␣
                    already␣initializing␣***");
39          }
40      }
41
42      public static void deInit() {
43          Log.d(TAG, "␣***␣Requested␣deinitialization␣with␣jniDeInit()␣***");
44          if (jniDeInit() == 0) {
45              Log.d(TAG, "␣***␣jniDeInit()␣OK␣***");
46              initialized = false;
47          } else {
48              Log.d(TAG, "␣***␣jniDeInit()␣ERROR␣***");
49          }
50      }
51
52      @NonNull
53      @DebugLog
54      public ArrayList<HeadPoseGaze> bitmapDetection(@NonNull Bitmap bitmap)
            {
55          if (!initialized) {
56              Log.e(TAG, "␣***␣HeadPoseDetector␣is␣not␣initialized␣,␣use␣
                    jniInit()␣***");
57              return new ArrayList<>();
58          }
59
60          // A set of results, say a set of gazes, populated by the jni call
61          ArrayList<HeadPoseGaze> gazes_set = new ArrayList<>();
62          jniBitmapExtractFaceGazes(bitmap, gazes_set);
63
64          return gazes_set;
65      }
66
67      private static native int jniInit(String landmarkModelPath, int
            alg_mode, boolean use_custom, float fx, float fy, float cx, float
            cy, float k1, float k2, float p1, float p2, float k3);
68
69      private static native int jniDeInit();
70
71      private native int jniBitmapExtractFaceGazes(Bitmap bitmap, ArrayList<
            HeadPoseGaze> set);
72  }
```

Listing A.8: CameraConnectionFragment.java

```
1  package edu.unipd.dei.selfear2;
2
3  import android.Manifest;
4  import android.annotation.SuppressLint;
5  import android.app.Activity;
6  import android.app.AlertDialog;
7  import android.app.Dialog;
8  import android.app.DialogFragment;
```

```
 9 import android.app.Fragment;
10 import android.content.Context;
11 import android.content.DialogInterface;
12 import android.content.pm.PackageManager;
13 import android.graphics.ImageFormat;
14 import android.graphics.Matrix;
15 import android.graphics.RectF;
16 import android.graphics.SurfaceTexture;
17 import android.hardware.camera2.CameraAccessException;
18 import android.hardware.camera2.CameraCaptureSession;
19 import android.hardware.camera2.CameraCharacteristics;
20 import android.hardware.camera2.CameraDevice;
21 import android.hardware.camera2.CameraManager;
22 import android.hardware.camera2.CaptureRequest;
23 import android.hardware.camera2.CaptureResult;
24 import android.hardware.camera2.TotalCaptureResult;
25 import android.hardware.camera2.params.StreamConfigurationMap;
26 import android.media.ImageReader;
27 import android.os.Build;
28 import android.os.Bundle;
29 import android.os.Handler;
30 import android.os.HandlerThread;
31 import android.support.v4.app.ActivityCompat;
32 import android.util.Log;
33 import android.util.Size;
34 import android.util.SparseIntArray;
35 import android.view.LayoutInflater;
36 import android.view.Surface;
37 import android.view.TextureView;
38 import android.view.View;
39 import android.view.ViewGroup;
40 import android.widget.Button;
41 import android.widget.TextView;
42 import android.widget.Toast;
43
44 import java.util.ArrayList;
45 import java.util.Arrays;
46 import java.util.Collections;
47 import java.util.Comparator;
48 import java.util.List;
49 import java.util.concurrent.Semaphore;
50 import java.util.concurrent.TimeUnit;
51
52 import edu.unipd.dei.selfear2.utils.XMLReader;
53 import edu.unipd.dei.selfear2.view.AutoFitTextureView;
54 import hugo.weaving.DebugLog;
55 public class CameraConnectionFragment extends Fragment {
56     /**
57      * The camera preview size will be chosen to be the smallest frame by
            pixel size capable of
58      * containing a DESIRED_SIZE x DESIRED_SIZE square.
59      */
60     private static final int MINIMUM_PREVIEW_SIZE = 320;
```

```
61      private static final String TAG = "CameraConnFragment";
62      /**
63       * Conversion from screen rotation to JPEG orientation.
64       */
65      private static final SparseIntArray ORIENTATIONS = new SparseIntArray()
            ;
66      private static final String FRAGMENT_DIALOG = "dialog";
67      static CameraCharacteristics mCameraCharacteristics;
68
69      static float[] mCameraIntrinsics = new float[5];
70      static float[] mCameraDistortions = new float[5];
71
72      /**
73       * A {@link Semaphore} to prevent the app from exiting before closing
             the camera.
74       */
75      private final Semaphore cameraOpenCloseLock = new Semaphore(1);
76      private final OnGetImageListener mOnGetPreviewListener = new
            OnGetImageListener();
77      private final CameraCaptureSession.CaptureCallback captureCallback =
78              new CameraCaptureSession.CaptureCallback() {
79                  @Override
80                  public void onCaptureProgressed(
81                          final CameraCaptureSession session,
82                          final CaptureRequest request,
83                          final CaptureResult partialResult) {
84                  }
85
86                  @Override
87                  public void onCaptureCompleted(
88                          final CameraCaptureSession session,
89                          final CaptureRequest request,
90                          final TotalCaptureResult result) {
91                  }
92              };
93      private TextView mPerformanceView;
94      private TextView mResultsView;
95      private TextView mInfoView;
96
97      private boolean buttonsClickable = false;
98      protected Button stopButton;
99      protected Button discardButton;
100     private String cameraId;
101     private AutoFitTextureView textureView;
102     private CameraCaptureSession captureSession;
103     private CameraDevice cameraDevice;
104     private Size previewSize;
105     private HandlerThread backgroundThread;
106     private Handler backgroundHandler;
107     private HandlerThread inferenceThread;
108     private Handler inferenceHandler;
109     private ImageReader previewReader;
110     private CaptureRequest.Builder previewRequestBuilder;
```

```
111     private CaptureRequest previewRequest;
112     private final CameraDevice.StateCallback stateCallback =
113             new CameraDevice.StateCallback() {
114                 @Override
115                 public void onOpened(final CameraDevice cd) {
116                     // This method is called when the camera is opened.  We
                            start camera preview here.
117                     cameraOpenCloseLock.release();
118                     cameraDevice = cd;
119                     createCameraPreviewSession();
120                 }
121
122                 @Override
123                 public void onDisconnected(final CameraDevice cd) {
124                     cameraOpenCloseLock.release();
125                     cd.close();
126                     cameraDevice = null;
127
128                     if (mOnGetPreviewListener != null) {
129                         mOnGetPreviewListener.deInitialize();
130                     }
131                 }
132
133                 @Override
134                 public void onError(final CameraDevice cd, final int error)
                        {
135                     cameraOpenCloseLock.release();
136                     cd.close();
137                     cameraDevice = null;
138                     final Activity activity = getActivity();
139                     if (null != activity) {
140                         activity.finish();
141                     }
142
143                     if (mOnGetPreviewListener != null) {
144                         mOnGetPreviewListener.deInitialize();
145                     }
146                 }
147             };
148     private final TextureView.SurfaceTextureListener surfaceTextureListener
            =
149             new TextureView.SurfaceTextureListener() {
150                 @Override
151                 public void onSurfaceTextureAvailable(
152                         final SurfaceTexture texture, final int width,
                                final int height) {
153                     openCamera(width, height);
154                 }
155
156                 @Override
157                 public void onSurfaceTextureSizeChanged(
158                         final SurfaceTexture texture, final int width,
                                final int height) {
```

```
159                     configureTransform(width, height);
160                 }
161
162                 @Override
163                 public boolean onSurfaceTextureDestroyed(final
                        SurfaceTexture texture) {
164                     return true;
165                 }
166
167                 @Override
168                 public void onSurfaceTextureUpdated(final SurfaceTexture
                        texture) {
169                 }
170             };
171     @SuppressLint("LongLogTag")
172     @DebugLog
173     private static Size chooseOptimalSize(
174             final Size[] choices, final int width, final int height, final
                Size aspectRatio) {
175         // Collect the supported resolutions that are at least as big as
                the preview Surface
176         final List<Size> bigEnough = new ArrayList<Size>();
177         for (final Size option : choices) {
178             if (option.getHeight() >= MINIMUM_PREVIEW_SIZE && option.
                    getWidth() >= MINIMUM_PREVIEW_SIZE) {
179                 Log.i(TAG, "Adding␣size:␣" + option.getWidth() + "x" +
                        option.getHeight());
180                 bigEnough.add(option);
181             } else {
182                 Log.i(TAG, "Not␣adding␣size:␣" + option.getWidth() + "x" +
                        option.getHeight());
183             }
184         }
185
186         // Pick the smallest of those, assuming we found any
187         if (bigEnough.size() > 0) {
188             final Size chosenSize = Collections.min(bigEnough, new
                    CompareSizesByArea());
189             Log.i(TAG, "Chosen␣size:␣" + chosenSize.getWidth() + "x" +
                    chosenSize.getHeight());
190             return chosenSize;
191         } else {
192             Log.e(TAG, "Couldn't␣find␣any␣suitable␣preview␣size");
193             return choices[0];
194         }
195     }
196
197     public static CameraConnectionFragment newInstance() {
198         return new CameraConnectionFragment();
199     }
200
201     private void showToast(final String text) {
202         final Activity activity = getActivity();
```

```
203             if (activity != null) {
204                 activity.runOnUiThread(
205                         new Runnable() {
206                             @Override
207                             public void run() {
208                                 Toast.makeText(activity, text, Toast.
                                        LENGTH_SHORT).show();
209                             }
210                         });
211             }
212         }
213
214         @Override
215         public View onCreateView(final LayoutInflater inflater, final ViewGroup
                container, final Bundle savedInstanceState) {
216             return inflater.inflate(R.layout.camera_connection_fragment,
                    container, false);
217         }
218         @Override
219         public void onViewCreated(final View view, final Bundle
                savedInstanceState) {
220             stopButton = (Button) view.findViewById(R.id.stop_capt);
221             discardButton = (Button) view.findViewById(R.id.discard_capt);
222             textureView = (AutoFitTextureView) view.findViewById(R.id.texture);
223             mPerformanceView = (TextView) view.findViewById(R.id.performance_tv
                    );
224             mResultsView = (TextView) view.findViewById(R.id.results_tv);
225             mInfoView = (TextView) view.findViewById(R.id.info_tv);
226
227             stopButton.setOnClickListener(new View.OnClickListener() {
228                 @Override
229                 public void onClick(View view) {
230                     if(buttonsClickable) getActivity().onBackPressed();
231                 }
232             });
233
234             discardButton.setOnClickListener(new View.OnClickListener() {
235                 @Override
236                 public void onClick(View view) {
237                     MainActivity.saveFile = false;
238                     if(buttonsClickable) getActivity().onBackPressed();
239                 }
240             });
241         }
242         @Override
243         public void onActivityCreated(final Bundle savedInstanceState) {
244             super.onActivityCreated(savedInstanceState);
245         }
246         @Override
247         public void onResume() {
248             super.onResume();
249             startBackgroundThread();
250
```

```
251         // When the screen is turned off and turned back on, the
                SurfaceTexture is already
252         // available, and "onSurfaceTextureAvailable" will not be called.
                In that case, we can open
253         // a camera and start preview from here (otherwise, we wait until
                the surface is ready in
254         // the SurfaceTextureListener).
255         if (textureView.isAvailable()) {
256             openCamera(textureView.getWidth(), textureView.getHeight());
257         } else {
258             textureView.setSurfaceTextureListener(surfaceTextureListener);
259         }
260     }
261     @Override
262     public void onPause() {
263         closeCamera();
264         stopBackgroundThread();
265         super.onPause();
266     }
267
268     @DebugLog
269     @SuppressLint("LongLogTag")
270     private void setUpCameraOutputs(final int width, final int height) {
271         final Activity activity = getActivity();
272         final CameraManager manager = (CameraManager) activity.
                getSystemService(Context.CAMERA_SERVICE);
273         try {
274             for (final String cameraId : manager.getCameraIdList()) { //
                    Cycle through all available cameras
275                 final CameraCharacteristics characteristics = manager.
                        getCameraCharacteristics(cameraId); // Inspect this
                        camera characteristics
276                 final Integer facing = characteristics.get(
                        CameraCharacteristics.LENS_FACING); // Get which facing
                         it is
277                 if (facing != null && facing == CameraCharacteristics.
                        LENS_FACING_FRONT) { // If it faces front, we've found
                         it
278                     mCameraCharacteristics = characteristics; // Get the
                            characteristics of the camera and set them as an
                            attribute
279
280                     // See if it complies with what we need, otherwise skip
                             to another camera
281                     final StreamConfigurationMap map =
282                             mCameraCharacteristics.get(
                                    CameraCharacteristics.
                                    SCALER_STREAM_CONFIGURATION_MAP);
283                     if (map == null) {
284                         continue;
285                     } // SKIP
286
287                     // For still image captures, we use the largest
```

```
                    available size.
288                 final Size largest = Collections.max(
289                         Arrays.asList(map.getOutputSizes(ImageFormat.
                                YUV_420_888)),
290                         new CompareSizesByArea());
291
292                 // Danger, W.R.! Attempting to use too large a preview
                        size could  exceed the camera
293                 // bus' bandwidth limitation, resulting in gorgeous
                        previews but the storage of
294                 // garbage capture data.
295                 previewSize =
296                         chooseOptimalSize(map.getOutputSizes(
                                SurfaceTexture.class), width, height,
                                largest);
297
298                 // Set aspect ratio for the textureView in order to
                        comply with landscape mode
299                 textureView.setAspectRatio(previewSize.getWidth(),
                        previewSize.getHeight());
300
301                 // Set the camera as the selected
302                 CameraConnectionFragment.this.cameraId = cameraId;
303
304                 if(Build.VERSION.SDK_INT >= 23) {
305                     if(mCameraCharacteristics.get(CameraCharacteristics
                            .LENS_INTRINSIC_CALIBRATION) != null) {
306                         mCameraIntrinsics =  mCameraCharacteristics.get
                                (CameraCharacteristics.
                                LENS_INTRINSIC_CALIBRATION);
307                         mInfoView.setText("Camera intrinsics available
                                !\n[f_x, f_y, c_x, c_y, s]\n" +
                                mCameraIntrinsics);
308                     }
309                     else {
310                         mCameraIntrinsics = XMLReader.
                                loadIntrinsicParams(getActivity());
311                         mInfoView.setText("No camera intrinsics
                                prebuilt values available for this device!
                                Using:\n" +
312                                 "[f_x] = " + mCameraIntrinsics[0] + "
                                    [f_y] = " + mCameraIntrinsics[1] + "
                                    [c_x] = " + mCameraIntrinsics[2] +
                                     "[c_y] = " + mCameraIntrinsics
                                    [2]);
313                     }
314
315                     if(mCameraCharacteristics.get(CameraCharacteristics
                            .LENS_RADIAL_DISTORTION) != null) {
316                         float[] values = mCameraCharacteristics.get(
                                CameraCharacteristics.
                                LENS_RADIAL_DISTORTION);
317                         mCameraDistortions[0] = values[1]; // k1
```

```
318                         mCameraDistortions[1] = values[2]; // k2
319                         mCameraDistortions[2] = values[4]; // p1
320                         mCameraDistortions[3] = values[5]; // p2
321                         mCameraDistortions[4] = values[3]; // k3
322                         mInfoView.append("Camera distortions available
                                !\n[k_1, k_2, p_1, p_2, k_3]\n" +
                                mCameraDistortions);
323                     }
324                     else {
325                         mCameraDistortions = XMLReader.
                                loadDistortionParams(getActivity());
326                         mInfoView.append("\nNo camera distortions
                                prebuilt values available for this device!
                                Using:\n" +
327                                 "[k_1] = " + mCameraDistortions[0] + "
                                    [k_2] = " + mCameraDistortions[1] +
                                     " [p_1] = " + mCameraDistortions
                                    [2] + " [p_2] = " +
                                    mCameraDistortions[3] + " k_3] = "
                                    + mCameraDistortions[4]);
328                     }
329                 } else {
330                     mCameraIntrinsics = XMLReader.loadIntrinsicParams(
                            getActivity());
331                     mInfoView.setText("No camera intrinsics prebuilt
                            values available for this device (API is <23)!
                            Using:\n" +
332                             "[f_x] = " + mCameraIntrinsics[0] + " [f_y]
                                = " + mCameraIntrinsics[1] + " [c_x] =
                                " + mCameraIntrinsics[2] + " [c_y] = "
                                 + mCameraIntrinsics[2]);
333                     mCameraDistortions = XMLReader.loadDistortionParams
                            (getActivity());
334                     mInfoView.append("\nNo camera intrinsics prebuilt
                            values available for this device (API is <23)!
                            Using:\n" +
335                             "[k_1] = " + mCameraDistortions[0] + " [k_2
                                ] = " + mCameraDistortions[1] + " [p_1]
                                = " + mCameraDistortions[2] + " [p_2]
                                = " + mCameraDistortions[3] + " k_3] =
                                " + mCameraDistortions[4]);
336                 }
337
338                 return;
339             }
340         }
341     } catch (final CameraAccessException e) {
342         Log.e(TAG, "Exception!", e);
343     } catch (final NullPointerException e) {
344         // Currently an NPE is thrown when the Camera2API is used but
                not supported on the device this code runs.
345         ErrorDialog.newInstance(getString(R.string.camera_error))
346                 .show(getChildFragmentManager(), FRAGMENT_DIALOG);
```

```
347             }
348         }
349 
350         @SuppressLint("LongLogTag")
351         @DebugLog
352         private void openCamera(final int width, final int height) {
353             setUpCameraOutputs(width, height);
354             configureTransform(width, height);
355             final Activity activity = getActivity();
356             final CameraManager manager = (CameraManager) activity.
                    getSystemService(Context.CAMERA_SERVICE);
357             try {
358                 if (!cameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS
                        )) {
359                     throw new RuntimeException("Time␣out␣waiting␣to␣lock␣camera
                            ␣opening.");
360                 }
361                 if (ActivityCompat.checkSelfPermission(this.getActivity(),
                        Manifest.permission.CAMERA) != PackageManager.
                        PERMISSION_GRANTED) {
362                     Log.w(TAG, "checkSelfPermission␣CAMERA");
363                 }
364                 manager.openCamera(cameraId, stateCallback, backgroundHandler);
365                 Log.d(TAG, "open␣Camera");
366             } catch (final CameraAccessException e) {
367                 Log.e(TAG, "Exception!", e);
368             } catch (final InterruptedException e) {
369                 throw new RuntimeException("Interrupted␣while␣trying␣to␣lock␣
                        camera␣opening.", e);
370             }
371         }
372 
373         @DebugLog
374         private void closeCamera() {
375             try {
376                 cameraOpenCloseLock.acquire();
377                 if (null != captureSession) {
378                     captureSession.close();
379                     captureSession = null;
380                 }
381                 if (null != cameraDevice) {
382                     cameraDevice.close();
383                     cameraDevice = null;
384                 }
385                 if (null != previewReader) {
386                     previewReader.close();
387                     previewReader = null;
388                 }
389                 if (null != mOnGetPreviewListener) {
390                     mOnGetPreviewListener.deInitialize();
391                 }
392             } catch (final InterruptedException e) {
393                 throw new RuntimeException("Interrupted␣while␣trying␣to␣lock␣
```

```
                camera closing.", e);
394         } finally {
395             cameraOpenCloseLock.release();
396         }
397     }
398
399     @DebugLog
400     private void startBackgroundThread() {
401         backgroundThread = new HandlerThread("ImageListener");
402         backgroundThread.start();
403         backgroundHandler = new Handler(backgroundThread.getLooper());
404
405         inferenceThread = new HandlerThread("InferenceThread");
406         inferenceThread.start();
407         inferenceHandler = new Handler(inferenceThread.getLooper());
408     }
409
410     @SuppressLint("LongLogTag")
411     @DebugLog
412     private void stopBackgroundThread() {
413         backgroundThread.quitSafely();
414         inferenceThread.quitSafely();
415         try {
416             backgroundThread.join();
417             backgroundThread = null;
418             backgroundHandler = null;
419
420             inferenceThread.join();
421             inferenceThread = null;
422             inferenceThread = null;
423         } catch (final InterruptedException e) {
424             Log.e(TAG, "error", e);
425         }
426     }
427
428     @SuppressLint("LongLogTag")
429     @DebugLog
430     private void createCameraPreviewSession() {
431         try {
432             final SurfaceTexture texture = textureView.getSurfaceTexture();
433             assert texture != null;
434
435             // We configure the size of default buffer to be the size of
                    camera preview we want.
436             texture.setDefaultBufferSize(previewSize.getWidth(),
                    previewSize.getHeight());
437
438             // This is the output Surface we need to start preview.
439             final Surface surface = new Surface(texture);
440
441             // We set up a CaptureRequest.Builder with the output Surface.
442             previewRequestBuilder = cameraDevice.createCaptureRequest(
                    CameraDevice.TEMPLATE_PREVIEW);
```

```
443             previewRequestBuilder.addTarget(surface);
444
445             Log.i(TAG, "Opening camera preview: " + previewSize.getWidth()
                    + "x" + previewSize.getHeight());
446
447             // Create the reader for the preview frames.
448             previewReader =
449                     ImageReader.newInstance(
450                             previewSize.getWidth(), previewSize.getHeight()
                                , ImageFormat.YUV_420_888, 2);
451
452             previewReader.setOnImageAvailableListener(mOnGetPreviewListener
                    , backgroundHandler);
453             previewRequestBuilder.addTarget(previewReader.getSurface());
454
455             // Here, we create a CameraCaptureSession for camera preview.
456             cameraDevice.createCaptureSession(
457                     Arrays.asList(surface, previewReader.getSurface()),
458                     new CameraCaptureSession.StateCallback() {
459
460                         @Override
461                         public void onConfigured(final CameraCaptureSession
                                cameraCaptureSession) {
462                             // The camera is already closed
463                             if (null == cameraDevice) {
464                                 return;
465                             }
466
467                             // When the session is ready, we start
                                displaying the preview.
468                             captureSession = cameraCaptureSession;
469                             try {
470                                 // Auto focus should be continuous for
                                        camera preview.
471                                 previewRequestBuilder.set(
472                                         CaptureRequest.CONTROL_AF_MODE,
473                                         CaptureRequest.
                                            CONTROL_AF_MODE_CONTINUOUS_PICTURE
                                            );
474                                 // Flash is automatically enabled when
                                        necessary.
475                                 previewRequestBuilder.set(
476                                         CaptureRequest.CONTROL_AE_MODE,
                                            CaptureRequest.
                                            CONTROL_AE_MODE_ON_AUTO_FLASH);
477
478                                 // Finally, we start displaying the camera
                                        preview.
479                                 previewRequest = previewRequestBuilder.
                                        build();
480                                 captureSession.setRepeatingRequest(
481                                         previewRequest, captureCallback,
                                            backgroundHandler);
```

```
482
483                             } catch (final CameraAccessException e) {
484                                 Log.e(TAG, "Exception!", e);
485                             }
486                         }
487
488                         @Override
489                         public void onConfigureFailed(final
                                CameraCaptureSession cameraCaptureSession) {
490                             showToast("onConfigureFailed()");
491                         }
492                     },
493                     null);
494         } catch (final CameraAccessException e) {
495             Log.e(TAG, "Exception!", e);
496         }
497
498         mOnGetPreviewListener.initialize(getActivity(), mCameraIntrinsics,
                mCameraDistortions, mPerformanceView, mResultsView,
                inferenceHandler);
499         buttonsClickable = true;
500     }
501
502     @DebugLog
503     private void configureTransform(final int viewWidth, final int
            viewHeight) {
504         final Activity activity = getActivity();
505         if (textureView == null || previewSize == null || activity == null)
                {
506             return;
507         }
508
509         final int rotation = activity.getWindowManager().getDefaultDisplay
                ().getRotation();
510         final Matrix matrix = new Matrix();
511         final RectF viewRect = new RectF(0, 0, viewWidth, viewHeight);
512         final RectF bufferRect = new RectF(0, 0, previewSize.getHeight(),
                previewSize.getWidth());
513         final float centerX = viewRect.centerX();
514         final float centerY = viewRect.centerY();
515         if(rotation == Surface.ROTATION_90) {
516             //Log.d(TAG, "Rotation is Surface.ROTATION_90");
517             bufferRect.offset(centerX - bufferRect.centerX(), centerY -
                    bufferRect.centerY());
518             matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.
                    FILL);
519             final float scale =
520                     Math.max(
521                             (float) viewHeight / previewSize.getHeight(),
522                             (float) viewWidth / previewSize.getWidth());
523             matrix.postScale(scale, scale, centerX, centerY);
524             matrix.postRotate(-90, centerX, centerY);
525         }
```

```
526             textureView.setTransform(matrix);
527         }
528
529         static class CompareSizesByArea implements Comparator<Size> {
530             @Override
531             public int compare(final Size lhs, final Size rhs) {
532                 // We cast here to ensure the multiplications won't overflow
533                 return Long.signum(
534                         (long) lhs.getWidth() * lhs.getHeight() - (long) rhs.
                            getWidth() * rhs.getHeight());
535             }
536         }
537
538         public static class ErrorDialog extends DialogFragment {
539             private static final String ARG_MESSAGE = "message";
540
541             public static ErrorDialog newInstance(final String message) {
542                 final ErrorDialog dialog = new ErrorDialog();
543                 final Bundle args = new Bundle();
544                 args.putString(ARG_MESSAGE, message);
545                 dialog.setArguments(args);
546                 return dialog;
547             }
548
549             @Override
550             public Dialog onCreateDialog(final Bundle savedInstanceState) {
551                 final Activity activity = getActivity();
552                 return new AlertDialog.Builder(activity)
553                         .setMessage(getArguments().getString(ARG_MESSAGE))
554                         .setPositiveButton(
555                                 android.R.string.ok,
556                                 new DialogInterface.OnClickListener() {
557                                     @Override
558                                     public void onClick(final DialogInterface
                                        dialogInterface, final int i) {
559                                         activity.finish();
560                                     }
561                                 })
562                         .create();
563             }
564         }
565 }
```

Listing A.9: OnGetImageListener.java

```
1  package edu.unipd.dei.selfear2;
2
3  import android.app.Activity;
4  import android.content.Context;
5  import android.graphics.Bitmap;
6  import android.graphics.Bitmap.Config;
7  import android.graphics.Canvas;
8  import android.graphics.Matrix;
9  import android.graphics.Point;
```

```
10 import android.media.Image;
11 import android.media.Image.Plane;
12 import android.media.ImageReader;
13 import android.media.ImageReader.OnImageAvailableListener;
14 import android.os.Handler;
15 import android.os.Trace;
16 import android.util.Log;
17 import android.view.Display;
18 import android.view.WindowManager;
19 import android.widget.TextView;
20
21 import junit.framework.Assert;
22
23 import org.w3c.dom.Document;
24
25 import java.io.File;
26 import java.math.RoundingMode;
27 import java.text.DecimalFormat;
28 import java.text.SimpleDateFormat;
29 import java.util.ArrayList;
30 import java.util.Date;
31
32 import edu.unipd.dei.dlib.HeadPoseDetector;
33 import edu.unipd.dei.dlib.HeadPoseGaze;
34 import edu.unipd.dei.selfear2.utils.FileUtils;
35 import edu.unipd.dei.selfear2.utils.ImageUtils;
36 import edu.unipd.dei.selfear2.utils.XMLWriter;
37 import edu.unipd.dei.selfear2.view.FloatingCameraWindow;
38
39 /**
40  * Class that takes in preview frames and converts the image to Bitmaps to
        process with dlib lib.
41  */
42 public class OnGetImageListener implements OnImageAvailableListener {
43     private static final int NUM_CLASSES = 1001;
44     private static final int INPUT_SIZE = 240;
45     private static final int IMAGE_MEAN = 117;
46     private static final String TAG = "OnGetImageListener";
47
48     private int mScreenRotation = 0;
49
50     private int mPreviewWdith = 0;
51     private int mPreviewHeight = 0;
52     private byte[][] mYUVBytes;
53     private int[] mRGBBytes = null;
54     private Bitmap mRGBframeBitmap = null;
55     private Bitmap mRGBrotatedBitmap = null;
56     //private Bitmap mCroppedBitmap = null;
57
58     private boolean mIsComputing = false;
59     private Handler mInferenceHandler;
60
61     private Context mContext;
```

```
62      private HeadPoseDetector mHeadPoseDetector;
63      private TextView mPerformanceView;
64      private TextView mResultsView;
65      private FloatingCameraWindow mWindow;
66
67      private DecimalFormat df;
68      private Document detectionDocument;
69
70      private double overallTime = 0;
71      private int valid_cycles = 0;
72
73      public void initialize( final Context context, final float[] intrinsics
            , final float[] distortions, final TextView mPerformanceView, final
             TextView mResultsView, final Handler handler) {
74          this.mContext = context;
75          this.mPerformanceView = mPerformanceView;
76          this.mResultsView = mResultsView;
77          this.mInferenceHandler = handler;
78          mHeadPoseDetector = new HeadPoseDetector();
79          mWindow = new FloatingCameraWindow(mContext);
80
81          // Ensure the model file is properly deserialized into destination
82          File model = new File(FileUtils.getPreference(mContext, FileUtils.
                DATA_DIR_PREFS_NAME), FileUtils.PREDICTOR_FILE_NAME);
83          if (!model.exists()) {
84              Log.d(TAG, "Copying␣landmark␣model␣to␣" + model.getAbsolutePath
                    ());
85              FileUtils.copyFileFromRawToOthers(mContext, R.raw.
                    shape_predictor_68_face_landmarks, model.getAbsolutePath())
                    ;
86          }
87
88          // Initialize the headpose detector with its parameters
89          mHeadPoseDetector.init(model.getAbsolutePath(), MainActivity.mode,
                MainActivity.useCustom, intrinsics, distortions);
90
91          // Initialize the formatter for the strings to be shown
92          df = new DecimalFormat("##.##");
93          df.setRoundingMode(RoundingMode.DOWN);
94
95          if(MainActivity.saveFile) detectionDocument = XMLWriter.newDocument
                (MainActivity.mode);
96      }
97
98      public void deInitialize() {
99          synchronized (OnGetImageListener.this) {
100             if(MainActivity.saveFile) {// Update performance info and save
                    the file
101                 XMLWriter.addTimePerformance(detectionDocument, overallTime
                        / valid_cycles); // Add performance field
102                 SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmss
                        ");
103                 XMLWriter.saveDocumentToFile(mContext, detectionDocument, "
```

```
                    detection_" + sdf.format(new Date(System.
                    currentTimeMillis())) + ".xml");
104             }
105             if (mHeadPoseDetector != null) {
106                 mHeadPoseDetector.deInit();
107             }
108
109             if (mWindow != null) {
110                 mWindow.release();
111             }
112         }
113     }
114
115     private void drawResizedBitmap(final Bitmap src, final Bitmap dst) {
116         Display display = ((WindowManager) mContext.getSystemService(
                Context.WINDOW_SERVICE)).getDefaultDisplay();
117         Point point = new Point();
118         display.getSize(point);
119         int screen_width = point.x;
120         int screen_height = point.y;
121
122         if (screen_width < screen_height) { // Screen is in portrait
123             mScreenRotation = 0;
124         } else { // Screen is in landscape
125             mScreenRotation = 90;
126         }
127
128         Assert.assertEquals(dst.getWidth(), dst.getHeight()); // Make sure
                the destination bitmap is square
129         final float minDim = Math.min(src.getWidth(), src.getHeight());
130
131         final Matrix matrix = new Matrix();
132
133         // We only want the center square out of the original rectangle.
134         final float translateX = -Math.max(0, (src.getWidth() - minDim) /
                2);
135         final float translateY = -Math.max(0, (src.getHeight() - minDim) /
                2);
136         matrix.preTranslate(translateX, translateY);
137
138         // Set the scale to accomodate the least between height and width
                of the source
139         final float scaleFactor = dst.getHeight() / minDim;
140         matrix.postScale(scaleFactor, scaleFactor);
141
142         // Rotate around the center if necessary.
143         if (mScreenRotation != 0) {
144             matrix.postTranslate(-dst.getWidth() / 2.0f, -dst.getHeight() /
                    2.0f);
145             matrix.postRotate(mScreenRotation);
146             matrix.postTranslate(dst.getWidth() / 2.0f, dst.getHeight() /
                    2.0f);
147         }
```

```
148
149             final Canvas canvas = new Canvas(dst);
150             canvas.drawBitmap(src, matrix, null);
151         }
152
153         private void drawUnmirroredRotatedBitmap(final Bitmap src, final Bitmap
                dst, final int rotation) {
154             final Matrix matrix = new Matrix();
155             //matrix.postTranslate(-dst.getWidth() / 2.0f, -dst.getHeight() /
                    2.0f);
156             matrix.postRotate(rotation);
157             matrix.setScale(-1, 1);
158             matrix.postTranslate(dst.getWidth(), 0);
159
160             final Canvas canvas = new Canvas(dst);
161             canvas.drawBitmap(src, matrix, null);
162         }
163
164         @Override
165         public void onImageAvailable(final ImageReader reader) {
166             Image image = null;
167             try {
168                 image = reader.acquireLatestImage();
169
170                 if (image == null) {
171                     return;
172                 }
173
174                 // No mutex needed as this method is not reentrant.
175                 if (mIsComputing) {
176                     image.close();
177                     return;
178                 }
179                 mIsComputing = true;
180
181                 Trace.beginSection("imageAvailable");
182
183                 final Plane[] planes = image.getPlanes();
184
185                 // Initialize the storage bitmaps once when the resolution is
                        known.
186                 if (mPreviewWdith != image.getWidth() || mPreviewHeight !=
                        image.getHeight()) {
187                     mPreviewWdith = image.getWidth();
188                     mPreviewHeight = image.getHeight();
189
190                     Log.d(TAG, String.format("Initializing␣at␣size␣%dx%d",
                            mPreviewWdith, mPreviewHeight));
191                     mRGBBytes = new int[mPreviewWdith * mPreviewHeight];
192                     mRGBframeBitmap = Bitmap.createBitmap(mPreviewWdith,
                            mPreviewHeight, Config.ARGB_8888);
193                     mRGBrotatedBitmap = Bitmap.createBitmap(mPreviewWdith,
                            mPreviewHeight, Config.ARGB_8888);
```

```
194                 //mCroppedBitmap = Bitmap.createBitmap(INPUT_SIZE,
                        INPUT_SIZE, Config.ARGB_8888);
195
196                 mYUVBytes = new byte[planes.length][];
197                 for (int i = 0; i < planes.length; ++i) {
198                     mYUVBytes[i] = new byte[planes[i].getBuffer().capacity
                            ()];
199                 }
200             }
201
202             for (int i = 0; i < planes.length; ++i) {
203                 planes[i].getBuffer().get(mYUVBytes[i]);
204             }
205
206             final int yRowStride = planes[0].getRowStride();
207             final int uvRowStride = planes[1].getRowStride();
208             final int uvPixelStride = planes[1].getPixelStride();
209             ImageUtils.convertYUV420ToARGB8888( mYUVBytes[0], mYUVBytes[1],
                     mYUVBytes[2], mRGBBytes, mPreviewWdith, mPreviewHeight,
                    yRowStride, uvRowStride, uvPixelStride, false);
210
211             image.close();
212         } catch (final Exception e) {
213             if (image != null) {
214                 image.close();
215             }
216             Log.e(TAG, "Exception!", e);
217             Trace.endSection();
218             return;
219         }
220
221         mRGBframeBitmap.setPixels(mRGBBytes, 0, mPreviewWdith, 0, 0,
                mPreviewWdith, mPreviewHeight);
222         drawUnmirroredRotatedBitmap(mRGBframeBitmap, mRGBrotatedBitmap, 0);
223         //drawResizedBitmap(mRGBframeBitmap, mCroppedBitmap);
224
225         mInferenceHandler.post(
226                 new Runnable() {
227                     @Override
228                     public void run() {
229                         final long startTime = System.currentTimeMillis();
230                         ArrayList<HeadPoseGaze> results;
231                         synchronized (OnGetImageListener.this) {
232                             results = mHeadPoseDetector.bitmapDetection(
                                    mRGBrotatedBitmap);
233                         }
234                         final long endTime = System.currentTimeMillis();
235
236                         ((Activity) mContext).runOnUiThread(new Runnable()
                                {
237                             @Override
238                             public void run() {
239                                 mPerformanceView.setText("Time cost\n" +
```

```
                                        String.valueOf((endTime - startTime) /
                                        1000f) + " sec");
240                                 }
241                             });
242
243                             // Update the score textview with info on result
244                             if(!results.isEmpty()) {
245                                 // Update performance timing
246                                 overallTime += ((endTime - startTime) / 1000f);
247                                 valid_cycles++;
248
249                                 final HeadPoseGaze r = results.get(0);
250                                 ((Activity) mContext).runOnUiThread(new
                                        Runnable() {
251                                     @Override
252                                     public void run() {
253                                         mResultsView.setText("Gaze angles\nYaw:
                                            " + df.format(r.getYaw()) +
254                                                 "\nPitch: " + df.format(r.
                                                    getPitch()) +
255                                                 "\nRoll: " + df.format(r.
                                                    getRoll()));
256                                     }
257                                 });
258                                 if(MainActivity.saveFile) XMLWriter.addResult(
                                        detectionDocument, System.currentTimeMillis
                                        (), r.getYaw(), r.getPitch(), r.getRoll());
259                             }
260
261                             mWindow.setRGBBitmap(mRGBrotatedBitmap);
262                             mIsComputing = false;
263                         }
264                     });
265
266             Trace.endSection();
267         }
268     }
```

Bibliografia

[1] Gelfand, S. A. (2015), *Hearing: An Introduction to Psychological and Physiological Acoustics*, 5th Edition, Informa Healthcare

[2] Fastl, H., Zwickler, E. (2007), *Psychoacoustics - Facts and Models*, 3rd Edition, Springer

[3] Bosi, M., Goldberg, Richard E. (2003), *Introduction to Digital Audio Coding and Standards*, 2003rd Edition, Springer Science+Business Media, LLC

[4] Kuttruff, H. (2009), *Room Acoustics*, 5th Edition, Spon Press

[5] Vörlander, M. (2008), *Auralization - Fundamentals of Acoustics, Modelling, Simulation, Algorithms and Acoustic Virtual Reality*, 1st Edition, Springer

[6] Blauert, J. (2012), *The technology of Binaural Listening*, 1st Edition, Springer

[7] Avanzini, F., De Poli, G. (2009), *Algorithms for sound and music computing*

[8] Spagnol, S., Geronazzo, M., Avanzini, F. (2013), *On the relation between pinna reflection patterns and head-related transfer function features* in IEEE transactions on audio, speech, and language processing 21.3 (2013): 508-519

[9] Morimoto, M. (2002), *The relation between spatial impression and the precedence effect* in Proceedings of Int. Conf. on Auditory Display (ICAD 2002), pp. 297-306

[10] Møller, H., Sorensen, M. F., Jensen, C. B., Hammershoi, A. D. (1996), *Binaural technique: Do we need individual recordings?* in Journal of the Audio Engineering Society 44.6 (1996): 451-469

[11] Lentz, T., Assenmachery, I., Vörländer, M., Kuhleny, T. (2006), *Precise Near-to-Head Acoustics with Binaural Synthesis* in Journal of Virtual Reality and Broadcasting, Volume 3 (2006), no. 2

[12] Algazi, V. R., Duda R. O., Thompson, D. M., Avendano, C. (2001), *The CIPIC HRTF Database* in IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2001

[13] Gardner, B., Martin, K. (1995), *HRTF Measurements of a KEMAR DummyHead Microphone*, in Journal Acoustic Society of America, Vol. 97, no. 6, Pp.3907–3908

[14] Middlebrooks, J. C., Green, D. M. (1991), *Sound Localization by Human Listeners* in Annual Review of Psychology, Vol. 42: 135-159

[15] Geronazzo, M., Granza, F., Spagnol, S., Avanzini, F. (2013), *A standardized repository of Head-Related and Headphone Impulse Response data*, Convention Paper 8902, AES 134th Convention, Rome, Italy

[16] Boren, B., Geronazzo, M., Brinkmann, F., Choueiri, E. (2015), *Coloration metrics for headphone equalization* in The 21th International Conference on Auditory Display (ICAD), Graz, Austria

[17] Martens, W. L. (2003), *Perceptual evaluation of filters controlling source direction: Customized and generalized HRTFs for binaural synthesis*, Multimedia Systems Laboratory, University of Aizu, Aizu-Wakamatsu, 965-8580 Japan

[18] Geronazzo, M. (2015), *L'acustica dell'orecchio esterno: un approccio a modelli strutturali misti per display uditivi virtuali* in Rivista Italiana di Acustica Vol. 39 (2015), N. 1, pp. 32-48

[19] Geronazzo, M., Spagnol, S., Avanzini, F. (2013), *Mixed structural modeling of head-related transfer functions for customized binaural audio delivery* in Digital Signal Processing (DSP), 2013 18th International Conference on. IEEE

[20] Geronazzo, M., Spagnol, S., Bedin, A., Avanzini, F. (2014), *Enhanching vertical localization with image-guided selection of non-individual head-related transfer function* in 2014 IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)

[21] Albrecht, R. (2016), *Methods and applications of mobile audio augmented reality*, Doctoral Dissertation in DOCTORAL DISSERTATIONS 122/2016, Aalto University publication series

[22] Fantin, J. (2016), *The Selfear project: Gestione della griglia spaziale per l'acquisizione low cost di HRIR individuali*, Tesi di Laurea Triennale in Ingegneria Informatica, Università degli Studi di Padova

[23] Geronazzo, M., Fantin, J., Sorato, G., Baldovino, G., Avanzini, F. (2016), *The SelfEar Project: a Mobile Application for Low-cost Pinna-Related Transfer Function Acquisition* in 13th Sound and Music Computing Conference (SMC 2016), Hamburg, Germany

[24] Yang, M., Kriegman, D. J., Ahura, N. (2016), *Detecting Faces in Images: A Survey* in IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 24, No. 1

[25] Ding, C., Tao, D. (2016), *A Comprehensive Survey on Pose-Invariant Face Recognition* in ACM Transactions on Intelligent Systems and Technology (TIST), 7(3), 37

[26] Murphy-Chutorian, E., Trivedi, M. M. (2009), *Head Pose Estimation in Computer Vision: A Survey* in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), VOL. 31, NO. 4, April 2009

[27] Kostinger, M., Wohlhart P., Roth, P.M., Bischof H. (2011), *Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization* in First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies, 2011

[28] Viola, P., Jones, M. (2001), *Rapid Object Detection Using a Boosted Cascade of Simple Features* in Computer Vision and Pattern Recognition, 2001 (CVPR 2001) Proceedings of the 2001 IEEE Computer Society Conference on. Vol. 1. IEEE, 2001

[29] Dalal, N., Triggs, B. (2005), *Histograms of Oriented Gradients for Human Detection* in International Conference on Computer Vision and Pattern Recognition (CVPR 2005), June 2005

[30] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., Ramanan, D. (2010), *Object Detection with Discriminatively Trained Part-Based Models* in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), VOL. 32, NO. 9, September 2010

[31] King, D. E. (2015), *Max-margin object detection*, arXiv preprint arXiv:1502.00046

[32] Kazemi, V., Sullivan J. (2014), *One Millisecond Face Alignment with an Ensemble of Regression Trees* in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014), June 2014

[33] Dollár, P., Welinder, P., Perona, P. (2010), *Cascaded pose regression* in IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2010), pp. 1078-1085

[34] Cao, X., Wei, Y., Wen, F., Sun, J. (2014), *Face alignment by explicit shape regression* in International Journal of Computer Vision, 107(2), 177-190

[35] Lepetit, V., and Fua, P. (2005), *Monocular model-based 3D tracking of rigid objects*, in Foundations and Trends in Computer Graphics and Vision, Vol. 1, No 1 (2005) 1–89, Now Publishers Inc, 2005

[36] Gao, X., Hou, X., Tang, J., Cheng, H. (2003), *Complete solution classification for the perspective-three-point problem* in IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 25.8 (2003): 930-943

[37] Lepetit, V., Moreno-Noguer, F., Fua, P. (2009), *Epnp: An accurate O(n) solution to the pnp problem*, International Journal of computer vision 81.2 (2009): 155-166

[38] Lemaignan, S., Garcia, F. Jacq, A., Dillenbourg, P. (2016), *From real-time attention assessment to with-me-ness in human-robot interaction* in The Eleventh ACM/IEEE International Conference on Human Robot Interation. IEEE Press, 2016

[39] King, D. E. (2009), *Dlib-ml: A Machine Learning Toolkit* in Journal of Machine Learning Research vol. 10, pp. 1755-1758