UNIVERSITÀ DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA

LA STANZA LOGO-MOTORIA. UN AMBIENTE MULTIMODALE INTERATTIVO PER L'INSEGNAMENTO A BAMBINI IN SITUAZIONE DI MULTI-DISABILITÀ

Laureando: Leonardo Amico

Relatore: prof. Sergio Canazza

Correlatori: prof. Antonio Rodà, dott.ssa Serena Zanolla

Corso di Laurea Magistrale in Ingegneria Elettronica

Data Laurea 23/10/2012

Anno Accademico 2011/2012

Sommario

Questo lavoro è centrato sullo sviluppo di un sistema interattivo multimodale per l'apprendimento, la Stanza Logo-Motoria. Il progetto si avvale della collaborazione di ricercatori nell'ambito psicologico ed educativo, allo scopo di realizzare uno strumento di supporto ai tradizionali metodi didattici. Con l'ausilio della tecnologia e seguendo l'approccio enattivo, è possibile aiutare le scuole a fornire modalità di insegnamento più efficaci, che tengano conto delle diverse capacità degli allievi anche nei casi di specifiche difficoltà di apprendimento. La Stanza Logo-Motoria analizza in tempo reale i movimenti e i gesti dei bambini all'interno di un ambiente sensorizzato, per poi elaborarli e associarli a una produzione audio-visiva. Una particolare attenzione è rivolta alle gestualità espressive, le quali contengono e trasmettono informazioni emozionali e affettive dei bambini. In questo modo la Stanza Logo-Motoria può essere utilizzata dagli insegnanti sia per la presentazione di materiale didattico attraverso un metodo alternativo, sia per verificare il livello di conoscenza negli alunni, che esprimono meglio le loro capacità per mezzo delle intelligenze visuali, spaziali o corporeo-cinestetiche attivate grazie al sistema.
Sistemi di apprendimento che prevedono l'interazione degli utenti all'interno di spazi fisici sono già stati realizzati nel corso degli anni. Ma gli alti costi e l'elevata complessità di gestione delle tecnologie utilizzate, hanno sempre implicato che il loro utilizzo da parte degli allievi venisse circoscritto a visite occasionali o a brevi periodi di sperimentazione. L'obiettivo che si pone il progetto Stanza Logo-Motoria è fornire uno strumento che possa essere integrato con le ordinarie attività didattiche all'interno della scuola. Per questo motivo si è scelto di impiegare tecnologie di grande diffusione come la *webcam*, di fornire strumenti che permettano agli insegnanti di adattare e produrre autonomamente i contenuti delle attività didattiche da svolgere con il sistema e di implementare un interfaccia utente per il software di gestione che non richieda specifiche competenze informatiche. Secondo queste direttive di ricerca, il lavoro sperimentale di tesi si è occupato della scrittura di un'applicazione C++ che acquisisce i movimenti degli utenti, li elabora e produce materiale audio-visivo secondo delle definite modalità di interazione.
Il testo della tesi si suddivide in quattro sezioni. Nel primo capitolo viene definito il quadro teorico di riferimento della Stanza Logo-Motoria: i concetti di interattività, multimodalità ed enattività nel contesto dell'apprendimento; e vengono descritti i principali progetti di sistemi multimodali interattivi. Il secondo capitolo introduce la Stanza Logo-Motoria e le linee guida che hanno orientato il suo sviluppo. Il capitolo seguente riguarda la ricerca preliminare al lavoro sperimentale di tesi: le esigenze del sistema in campo hardware (la tecnologia di

acquisizione per *motion capture*) e software (l'elaborazione dei dati dei sensori e la gestione delle attività interattive). Questa sezione si occupa di entrambi gli ambiti, pur riservando una maggiore attenzione agli strumenti software che hanno effettivamente costituito la base del successivo lavoro. Nel quarto capitolo viene descritto il software sviluppato in C++, analizzando separatamente la parte relativa al *motion tracking* e quella responsabile della gestione delle attività interattive svolte all'interno della Stanza Logo-Motoria. Infine, nel capitolo conclusivo, si suggeriscono possibili ulteriori sviluppi del software e le prospettive future per la Stanza Logo-Motoria.

Ringraziamenti

Vorrei ringraziare innanzitutto la mia famiglia, i miei genitori Angelo e Amalia, che mi hanno sostenuto affettivamente oltre che economicamente durante tutti gli anni di Università, e i miei fratelli Patrizia e Mauro, insostituibili riferimenti in qualsiasi occasione.
Poi desidero ringraziare sentitamente il mio relatore prof. Sergio Canazza e i correlatori prof. Antonio Rodà e dott.sa Serena Zanolla, che mi hanno supportato durante lo sviluppo e la stesura della tesi. Grazie anche per avermi coinvolto in quella magnifica esperienza che è la competizione per idee d'impresa Startcup. I premi ottenuti sono un grande riconoscimento per il lavoro svolto con il progetto della Stanza Logo-Motoria.
Un ringraziamento va anche a tutti gli amici. Quelli vicini e quelli che, disperso io o dispersi loro in giro per l'Europa, vedo sempre più di rado. Quindi grazie a Fede, Wole, Matteo, Filippo, Marcello, Fabio, Francesco, Enrico, Ale, Giamma. Preziosi compagni di studi, di serate, di viaggi, di film, di concerti, di discorsi... grazie.
Infine grazie anche a tutti i gruppi che hanno suonato ininterrottamente attraverso le mie cuffie durante questi mesi di C++ e Latex, compagni fedeli di scritture, riscritture, problemi e soluzioni. Tangerine Dream, Bardo Pont, Low, Klaus Schulze, Nico, Extra Life, Earth, Barn Owl, Mammifer, Godspeed You! Black Emperor.

Indice

Capitolo 1

Sistemi multimodali interattivi per l'apprendimento in spazi fisici

1.1 Interattività

Il concetto di interattività ha avuto negli anni molte diverse definizioni, a seconda del contesto in cui è stato utilizzato e dalle persone che se ne sono occupate. Nell'ultimo decennio, inoltre, l'affermarsi di nuove tecnologie in ambito informatico ha contribuito a rendere il concetto di interazione un argomento di dominio pubblico, non più ristretto a discussioni in ambito accademico. Si pensi ad esempio al web (la semplice navigazione via *hyperlink* è stata sostituita da esperienze immersive fatte di grafiche complesse, animazioni e interazione nei linguaggi Flash/Action Script, Javascript, html5), alla grande diffusione di nuove interfacce uomo-macchina (le tecnologie touch-screen di smartphone e tablet o i nuovi paradigmi di gioco introdotti dai controller WiiMote o Kinect) o infine alla democratizzazione di strumenti per creare interazione (la piattaforma Arduino, i linguaggi Processing e il toolkit per C++ Openframeworks [1]). Il raffinato livello di comunicazione tra uomo e computer che tali tecnologie rendono possibile è quindi ciò a cui comunemente si pensa quando si parla di interazione. Ma nonostante abbia ormai un suo posto nell'immaginario collettivo e sia argomento di ricerca da oltre 30 anni, una definizione globalmente riconosciuta del termine interazione resta comunque un problema aperto. Una prima trattazione comprensiva del termine viene data da Rafaeli [2] che con esso identifica "l'espressione della misura in cui, in uno scambio comunicativo, le terze e le successive trasmissioni si correlano con i messaggi precedenti che, a loro volta, si riferiscono a messaggi ancora più lontani nel tempo". In questa prima definizione è già possibile individuare un aspetto chiave dell'interazione, che sarà comune anche a tutte le definizioni successive: l'interazione è una proprietà della comunicazione, senza di cui essa non può esistere. Scendendo nel dettaglio della definizione di Rafaeli, si nota inoltre come questa pone particolare accento sull'aspetto ricorsivo dell'inte-

[1] Si veda a questo proposito [1]. Nell'introduzione l'autore identifica il suo target non in informatici o ingegneri, ma in "designer, artisti o *creative thinker* interessati a imparare a programmare allo scopo di creare personalmente applicazioni interattive"

razione (uno scambio è tanto più interattivo quanto più il suo contenuto dipende da scambi anteriori a esso). Caratteristica a cui verrà fatto in seguito riferimento come dipendenza del terzo grado (*third-order dependency*) [3]. Steuer definisce l'interazione come il grado in cui utenti di un dato mezzo di comunicazione possono modificare in tempo reale la forma o il contenuto dell'ambiente in cui esso si trova [4] e in maniera simile Jensen la identifica con la potenziale capacità con cui un mezzo di comunicazione permette all'utente di esercitare un'influenza sul contenuto e sulla forma della comunicazione [5]. Altrove il livello di interattività di un sistema è strettamente correlato al grado in cui riesce a emulare interazioni simili alle comunicazioni interpersonali [6]. Infine Kiousis nel tentativo di integrare le differenti visioni definisce l'interattività come "la misura in cui una tecnologia di comunicazione è in grado di creare un ambiente nel quale i partecipanti possano comunicare (uno-a-uno, uno-a-molti, molti-a-molti) e partecipare a scambi reciproci (dipendenza del terzo grado)" aggiungendo come nel caso di utenti umani essa si riferisca anche "all'abilità di percepire l'esperienza come una simulazione della comunicazione interpersonale" [3].

1.1.1 Interattività e apprendimento

Per quanto riguarda la presente trattazione è necessario analizzare il ruolo che l'interazione può avere in un contesto educativo e i vantaggi che essa può offrire. A tale proposito Moreno e Meyer [7] individuano cinque diverse tipologie di interazione.

1. Dialogo.
2. Controllo.
3. Manipolazione.
4. Ricerca.
5. Navigazione.

Come è stato notato in precedenza l'interazione è una caratteristica degli scambi comunicativi. Ora, se l'oggetto di questa comunicazione è costituito da contenuti didattici è facile riportare le diverse modalità di interazione enunciate da Moreno e Mayer nel contesto dell'apprendimento. Di seguito alcuni esempi. (1) Un confronto/discussione orale in cui lo scambio di informazioni non è unilaterale ma viene data la possibilità agli allievi di fare domande ed esprimere la propria opinione influenzando così il contenuto della lezione. (2) Un'esposizione orale in cui l'allievo ha la possibilità di controllare la velocità e scegliere di interrompere la spiegazione in maniera da fruire dei contenuti didattici secondo il proprio ritmo. (3) Un esperimento scientifico che lascia la possibilità all'allievo di sperimentare differenti parametri e vedere quello che succede. (4) La possibilità di cercare autonomamente informazioni riguardanti una certa materia all'interno di una collezione. (5) La possibilità di personalizzare la fruizione del contenuto didattico attraverso percorsi a scelta multipla, ipertesti, ecc...
L'interattività svolge una riconosciuta azione di supporto all'apprendimento. Secondo Baker

essa è “un necessario e fondamentale meccanismo per l’acquisizione della conoscenza e lo sviluppo delle capacità fisiche e cognitive” [8], mentre Amthor arriva a quantificare l’apprendimento dichiarando che le persone memorizzano circa il 20% di quanto ascoltano, il 40% di quello che vedono e ascoltano e il 75% di quello che vedono, ascoltano e fanno [9]. La scuola cosiddetta costruttivista della psicologia dello sviluppo, deve il suo nome all’idea che la conoscenza si costruisca in maniera analoga alle strutture fisiche del mondo reale. L’apprendimento deriva dall’azione [10], per cui i bambini costruiscono la propria struttura cognitiva attraverso l’azione e l’attività spontanea [11]. Papert coadiuva tali teorie occupandosi anche del coinvolgimento che gli allievi devono avere nella costruzione e progettazione dell’esperienza conoscitiva [12]. Più recentemente la teoria dell’azione [13] aggiunge un aspetto sociale al costruttivismo specificando come anche al di fuori dei luoghi di apprendimento formali la cognizione e l’azione siano inseparabili, per cui risulta impossibile separare la conoscenza di qualcosa con l’interazione che si ha con essa. Infine anche Moreno e Mayer nel loro modello educativo *CATLM* (Cognitive Affective Theory of Learning with Media) [14] specificano che l’apprendimento avviene in maniera efficiente quando l’allievo “impegna in modo consapevole i processi cognitivi di scelta, organizzazione e integrazione delle nuove informazioni con le conoscenze pregresse”.
Dopo aver individuato quello che si vuole ottenere da un ambiente interattivo è necessario stabilire delle linee guida per la sua progettazione. I principali aspetti da tenere in considerazione sono due: il grado di interazione permesso dal sistema e le conoscenze richieste per interfacciarsi con esso. Si pensi ad esempio alla comunicazione tra esseri umani, dove l’interazione è presente nel suo più alto grado [6]. La sua evoluzione fino alla forma attuale ha richiesto centinaia di anni, dalla formazione del linguaggio al suo perfezionamento. È evidente che a un alto livello di interattività corrispondono maggiori e più elaborate attività, ma d’altra parte ciò richiede una comunicazione più sofisticata e complessa, che per la sua comprensione può richiedere sforzi tali da compromettere l’esperienza dell’utente. Implementare modalità di interazione che siano quanto più semplici e naturali possibile è un aspetto cruciale nella progettazione di sistemi interattivi per l’ambito educativo. Se l’ambiente interattivo deve fornire uno strumento di ausilio all’insegnamento, è fondamentale che esso supporti le attività didattiche senza costituire a sua volta un oggetto di apprendimento. Una soluzione al problema consiste nello sfruttare modalità d’interazione o interfacce già familiari all’utente, il concetto di *anticipable experience* [1]. Un approccio di questo tipo si traduce nello sfruttamento delle cosiddette *affordances* [2] ovvero le caratteristiche degli oggetti o degli ambienti che suggeriscono all’utente quali sono le azioni da compiere per gestirli. Al contrario, riprodurre situazioni conosciute e richiedere che l’utente si comporti in maniera differente da come è abituato confonde e compromette l’esperienza interattiva. Lo studio delle *affordances* per la progettazione di interfacce è diffuso in tutti i campi delle *Human-Computer interfaces*. Nell’ambito dell’apprendimento il bacino di utenza è costituito da bambini e come spiegato in precedenza, un esigenza progettuale è che le competenze ne-

[2]Termine coniato dallo psicologo James J. Gibson in [15] e introdotto nel campo delle *Human-Computer Interfaces* da Donald Norman allo scopo di tracciare delle linee guida per la progettazione di interfacce il più possibile comprensibili e usabili dagli utenti [16]

cessarie per interagire col sistema siano minime. A questo proposito nell'ultimo decennio si sta affermando una nuova disciplina, in cui l'interazione è basata sul gesto e sul movimento. Le interfacce enattive.

1.2 Enattività

L'introduzione nell'ambiente scolastico di tecnologie quali il computer, la lavagna interattiva, il tablet permette di conferire un certo grado di interattività alla presentazione del materiale didattico nel corso delle lezioni, anche se il grado di interattività di queste tecnologie è però limitato al livello di partecipazione degli allievi alle attività. Ma un aspetto sostanziale che tali tecnologie non possono fornire è l'utilizzo del movimento all'interno di un ampio spazio fisico [17]. Di particolare interesse a questo proposito è il concetto di apprendimento enattivo. L'enazione è un termine introdotto dallo psicologo cognitivista Jerome Bruner per indicare uno dei modi possibili di organizzare la conoscenza, insieme alla modalità iconica e simbolica [18]. Le modalità iconica e simbolica si riferiscono rispettivamente ad un tipo di interazione con la realtà che avviene attraverso immagini (come diagrammi o illustrazioni) o attraverso simboli che richiedono di essere tradotti e interpretati (parole, formule matematiche...). L'apprendimento enattivo deriva invece dall'azione, la conoscenza viene costruita mediante l'interazione con il mondo attraverso attività motorie e che coinvolgono l'intera sfera sensoriale. Riconosciuta come la modalità attraverso cui viene costruita la conoscenza nelle prime fasi dello sviluppo cognitivo [19], l'enazione è la maniera più diretta, naturale e intuitiva di apprendimento. Il concetto di *Learning-By-Doing* (imparare facendo) è inoltre utilizzato anche da differenti scuole di pensiero pedagogico. Secondo l'approccio costruttivista l'allievo deve attivamente costruire la propria conoscenza attraverso esperienze basate sul fare [10], per cui i bambini formano autonomamente la propria struttura cognitiva attraverso l'azione e l'attività spontanea [11]. Papert coadiuva tali teorie occupandosi anche del coinvolgimento che gli allievi devono avere nella creazione dell'esperienza conoscitiva [12], riconoscendo così l'importanza dell'esperienza interattiva nel corso dell'apprendimento. Infine in una più recente definizione viene ribadita l'importanza che l'azione e il suo carattere interattivo ricoprono nel processo cognitivo, definito come "l'enazione di un mondo e di una mente sulla base della storia della varietà di azioni che un essere esegue nel mondo" [20].
Inoltre non è da sottovalutare come, nel caso di sistemi di apprendimento in cui l'interazione fisica attraverso gesti e movimenti è privilegiata, l'allievo si trovi più coinvolto, quindi incoraggiato a partecipare alle attività educative. Ricerche nell'ambito di *Human Computer Interface* (*HCI*) si occupano di progettare e sviluppare artefatti che sfruttino il concetto di enazione per esprimere e trasmettere conoscenza. Lo scopo consiste nel fornire delle interfacce che permettano l'interazione con l'utente attraverso le gestualità naturali. A tale proposito è stato costituito nel 2004 dall'Unione Europea un network di centri di ricerca e istituzioni (*ENACTIVE Network of Excellence*) allo scopo di finanziare e coordinare la ricerca sulla conoscenza enattiva e sulle interfacce enattive [21].

1.3 Multimodalità

La comunicazione, e con essa la trasmissione della conoscenza, avviene attraverso due modalità distinte: verbale e non-verbale. La comunicazione verbale fa uso del linguaggio, sia esso scritto o orale, e dipende da precise regole sintattiche e grammaticali, mentre la comunicazione non verbale riguarda gesti ed eventi non linguistici come le espressioni facciali e la postura. Secondo la teoria di Dual-Coding [22] la cognizione è frutto dell'attività di due sistemi distinti, specializzati per i due diversi modi di comunicazione verbale e non-verbale. Se questi sono i modi con cui l'informazione può essere codificata, le modalità percettive identificano invece i canali attraverso cui le informazioni vengono acquisite, i sensi. La comunicazione verbale avviene attraverso le parole e lo scritto. Ci sono poi linguaggi verbali che si basano sui gesti (linguaggio dei sordomuti), sul tatto (Braille per non-vedenti) o su strumenti convenzionali (linguaggio PCS, Picture Comunication Symbols). Il linguaggio non verbale (o analogico) è legato alla corporeità (mimica facciale, sguardo, postura, gestualità, distanza interpersonale, gestione dello spazio e degli oggetti). Tra questi due linguaggi si inserisce quello paraverbale rappresentato dal tono, dal ritmo, dalla vocalizzazione, dalle pause e dalle sottolineature. Nelle sue teorie Gardner [23] identifica nove distinte forme di intelligenza (linguistica, logico matematica, spaziale, corporeo-cinestetica, musicale, interpersonale, intrapersonale, naturalistica, esistenziale), segnalando come i modelli educativi tradizionali prediligano la trasmissione di conoscenza attraverso canali verbali e si limitino principalmente alle intelligenze linguistica e matematica, penalizzando le altre forme di intelligenza. Un ambiente di apprendimento ideale deve invece essere organizzato/progettato in modo che gli allievi possano sviluppare le loro diverse forme di intelligenza e di comunicazione, cercando quindi di tener conto di ognuna di esse nel corso delle attività educative. A tale proposito le abilità spaziali, musicali, cinestetiche non possono essere stimolate attraverso una comunicazione puramente verbale e richiedono che l'ambiente educativo coinvolga diverse modalità sensoriali. L'efficacia di un sistema di questo tipo è confermata dal *multimedia principle* e dal *modality principle of instructional design* secondo i quali l'abilità di comprensione degli studenti può essere migliorata se le spiegazioni verbali vengono associate/sincronizzate con rappresentazioni non verbali della conoscenza. Poiché la struttura cognitiva umana è formata da canali indipendenti a con capacità limitata, la modalità di insegnamento che utilizza un solo canale percettivo può incorrere nel rischio di sovraccaricare la capacità cognitiva degli studenti, rispetto invece alla presentazione multimodale dei contenuti (sincronizzazione di molteplici canali sensoriali) [24].
Un ambiente di apprendimento in cui si faccia uso di presentazioni verbali e non verbali utilizzando differenti modalità percettive è un sistema multimodale.

1.4 Sistemi multimodali interattivi e tecnologia

Si è fin ora discusso di come un sistema multimodale, interattivo, enattivo possa costituire un ambiente dove:

- La trasmissione della conoscenza avvenga per mezzo di presentazioni verbali e non-verbali utilizzando differenti canali percettivi (multimodalità);
- L'apprendimento sia frutto di una costruzione consapevole e attiva della conoscenza (interattività);
- L'attività motoria (manipolazione di oggetti movimenti...) sia la forma di interazione privilegiata (enattività).

L'ambiente interattivo multimodale quindi può fornire significativi vantaggi per l'apprendimento, la ritenzione delle conoscenze acquisite e il coinvolgimento dell'allievo nelle attività didattiche. Nel corso degli ultimi anni il progresso tecnologico ha dato luogo a una proliferazione di installazioni e applicazioni interattive e multimodali. Il mercato dell'intrattenimento, tradizionalmente interessato alla creazione di esperienze immersive per gli utenti, è stato uno dei primi a commercializzare sistemi di questo tipo. Strutture pubbliche quali musei e istituzioni educative stanno iniziando a dotarsi di tecnologie interattive e multimodali, spesso sotto forma di installazione, per attirare e motivare i visitatori, quindi adempiere in maniera efficace al loro compito di trasmissione di conoscenza. A tale proposito nasce il settore cosiddetto dell'*edutainment*: educare in maniera informale attraverso il gioco e l'intrattenimento. Anche in ambito ingegneristico si sta diffondendo un certo interesse per le tecnologie di supporto all'apprendimento, in particolare basate sull'approccio enattivo. Aree di ricerca quali *Computer vision*, *tangible interfaces*, *embeddable systems*, *embodied interaction*, trovano le loro applicazioni anche nella progettazione di sistemi interattivi multimodali.

1.5 Descrizione di alcuni progetti esistenti

A partire dagli anni sessanta sono stati realizzati ambienti multimodali in cui gli utenti interagiscono in uno spazio aumentato tecnologicamente attraverso gestualità naturali. Inizialmente pochi di essi si sono rivolti al campo educativo ma in seguito il progresso tecnologico (aumento delle prestazioni dei computer, maggiore accessibilità di strumenti informatici e riduzione del costo dei dispositivi) e un maggiore interesse all'argomento da parte delle istituzioni (scuole, centri didattici, musei, istituzioni governative...) ha fatto sì che più centri di ricerca iniziassero ad occuparsi delle possibili applicazioni di questi sistemi nell'ambito dell'apprendimento. Di seguito sono riportati alcuni esempi significativi.
Videoplace [25] di Myron Krueger è il primo progetto di realtà virtuale in cui l'utente interagisce con l'ambiente circostante senza l'ausilio di guanti o dispositivi ottici. Le principali applicazioni del sistema consistono in installazioni interattive ("ambienti reattivi", come sono stati battezzati dall'autore), in cui gli utenti sono rappresentati sotto forma di silhouette e attraverso le loro azioni coordinano e controllano immagini e oggetti grafici all'interno di una proiezione. In alcune occasioni il sistema è stato anche utilizzato in contesti di apprendimento informale, come esplorazioni scientifiche virtuali per bambini, o di riabilitazione.
SOUND=SPACE [26] è stato il primo ambiente di apprendimento interattivo a essere utilizzato con i bambini in maniera continuativa. Sviluppato da Rolf Gehlhaar dal 1985 al 1989

Figura 1.1: Alcuni esempi delle simulazioni del sistema Videoplace.

questo ambiente interattivo impiega tecnologie di eco-localizzazione per acquisire dati relativi alla posizione e allo spostamento di persone all'interno di uno spazio. Tali informazioni sono poi utilizzate da un computer per generare direttamente suoni e/o segnali di controllo midi con cui controllare sintetizzatori o *samplers*, con diversi modi possibili di strutturare lo spazio e di "suonare" gli strumenti (topografie di interazione musicale). Il sistema na-

Figura 1.2: Alcuni utenti interagiscono con l'ambiente SOUND=SPACE.

sce come uno strumento musicale "ambientale" che oltre alle applicazioni artistiche che ne possono derivare (installazioni, concerti, spettacoli di danza...) rende possibile creare musica in una maniera giocosa e senza necessità di acquisire le abilità richieste da strumenti tradizionali quali la chitarra o il pianoforte. In questa prospettiva SOUND=SPACE è stato installato permanentemente in una comunità artistica di Edinburgh, dove è stato utilizzato in una serie di *workshop* rivolti prevalentemente a bambini con difficoltà comportamentali e d'apprendimento e bambini con varie disabilità cognitive o motorie. Obiettivo dei workshop era lo sviluppo della creatività e dello spirito di collaborazione. Grazie alle modalità di interazione permesse dal sistema, che non richiedono specifiche conoscenze informatiche, e la possibilità di essere impostato per rispondere in tempo reale ai movimenti minimi degli utenti, esso può costituire la prima esperienza espressiva musicale per molti partecipanti. Lo stesso gruppo di ricerca ha in seguito prodotto una versione meno costosa del sistema che utilizza una telecamera, in cui il flusso video viene utilizzato invece dell'eco-localizzazione per l'analisi del movimento.
L'installazione interattiva Wheel of Life [27] ricrea un mondo virtuale in cui l'utente è chiamato a esplorare diverse regioni, ognuna ispirata ai quattro elementi naturali: acqua, terra, aria e fuoco. Muovendosi attraverso lo spazio e interagendo con un mondo che risponde alle sue azioni attraverso luci, suoni e immagini, l'esploratore deve cercare di decifrare le regole che governano ogni area. Il progetto è stato sviluppato nel corso di un *workshop* in cui gli studenti sono stati suddivisi in vari gruppi, ognuno dei quali preposto a uno specifico compito nel corso della ideazione e realizzazione dell'installazione. In particolare ogni

mondo doveva essere dotato di caratteristiche proprie, tipiche dell'elemento che rappresentava: scenografia, immagini, suoni e modalità di interazione sono quindi stati progettati di conseguenza. Un aspetto interessante di Wheel of Life è costituito dall'introduzione in un ambiente interattivo multimodale di una figura esterna con il compito di guidare l'esplorazione dei visitatore. La guida non accede al mondo virtuale, ma aiuta gli utenti controllando da un computer alcuni agenti multimediali all'interno della simulazione. La progettazione di questa particolare modalità di comunicazione tra guida e esploratore ha richiesto particolari sforzi, immaginando ogni volta nuove modalità per stabilire un "contatto" tra le due parti in maniera coerente con ognuno dei diversi ambienti. Un altro importante progetto

Figura 1.3: All'interno dell'ambiente di simulazione Wheel of Life.

è l'ambiente di story-telling interattivo per bambini KidsRoom [28]. Questo sistema ricrea una speciale camera da letto dotata di tecnologia per realtà aumentata. Due delle pareti sono del tutto simili a pareti reali, mentre le altre due sono costituite la larghi schermi su cui vengono retro-proiettate delle immagini. La stanza è anche dotata di un illuminazione controllata e da quattro altoparlanti, mentre quattro videocamere e un microfono vengono utilizzati per l'acquisizione degli input da parte degli utenti. Sei computer sono poi utilizzati per l'elaborazione dei dati e per controllare l'ambiente. Gli obiettivi con cui il sistema è stato progettato sono i seguenti: (1) la comunicazione tra gli utenti e il sistema avviene attraverso azioni e interazioni nello spazio reale e non in quello virtuale; (2) permettere agli utenti di collaborare tra loro interagendo con oggetti sia fisici che virtuali; (3) utilizzare algoritmi di *computer vision* per acquisire ed elaborare i dati relativi alle azioni degli utenti, evitando che questi debbano indossare speciali apparecchiature o sensori che rendano intrusiva la presenza della tecnologia; (4) applicare una struttura narrativa alle simulazioni per focalizzare

Figura 1.4: Una guida interagisce con l'ambiente di simulazione Wheel of Life.

le attività percettive degli utenti e lasciare che siano essi a controllare lo svolgimento della storia; (5) creare un ambiente il più possibile immersivo e interattivo.
L'obiettivo del sistema è guidare e stimolare le naturali capacità di immaginazione dei bambini, fornendo un ambiente dove la loro abilità di inventare storie e situazioni possa essere agevolata e incentivata per mezzo di feedback uditivi e visivi. Gli scopi del progetto MEDIATE (Multisensory Environment Design for an Interface between Autistic and Typical Expressiveness) [29] consistono nella progettazione, produzione, costruzione e convalidazione di un ambiente intelligente, immersivo, multisensoriale, interattivo adatto a tutte le tipologie di disturbi dello spettro autistico. Il team responsabile della progettazione e realizzazione dell'ambizioso sistema è spiccatamente multidisciplinare e coinvolge ingegneri, psicologi, informatici e tecnici di Olanda, Spagna e Inghilterra. Il sistema realizzato dal progetto MEDIATE è dotato di sensori e attuatori capaci di permettere interazioni di tipo visuale, uditivo e vibro-tattile. Il sistema si pone come obiettivo la costituzione di uno spazio dove gli utenti possano divertirsi, esprimersi e sperimentare un senso di controllo, fattori che nei casi di autismo possono considerarsi terapeutici.

Tutti i progetti illustrati fin ora adottano tecnologie complesse e costose, sono realizzati quasi sempre in un unico esemplare e difficilmente vengono spostati e installati in posti diversi da quello iniziale. Un'eccezione a questa norma è costituita dal progetto SMALLab (Situated Multimedia Arts Learning Laboratory) [30], recentemente integrato insieme ad altri strumenti di apprendimento sperimentali nel programma didattico di una scuola pubblica di New York [31]. Si tratta però di un programma speciale, reso possibile grazie al sostegno

Figura 1.5: Bambini interagiscono con una storia all'interno dell'ambiente interattivo multimodale KidsRoom.

Figura 1.6: Dall'esterno dell'ambiente interattivo multimodale KidsRoom sono possibili vedere le apparecchiature che controllano l'ambiente.

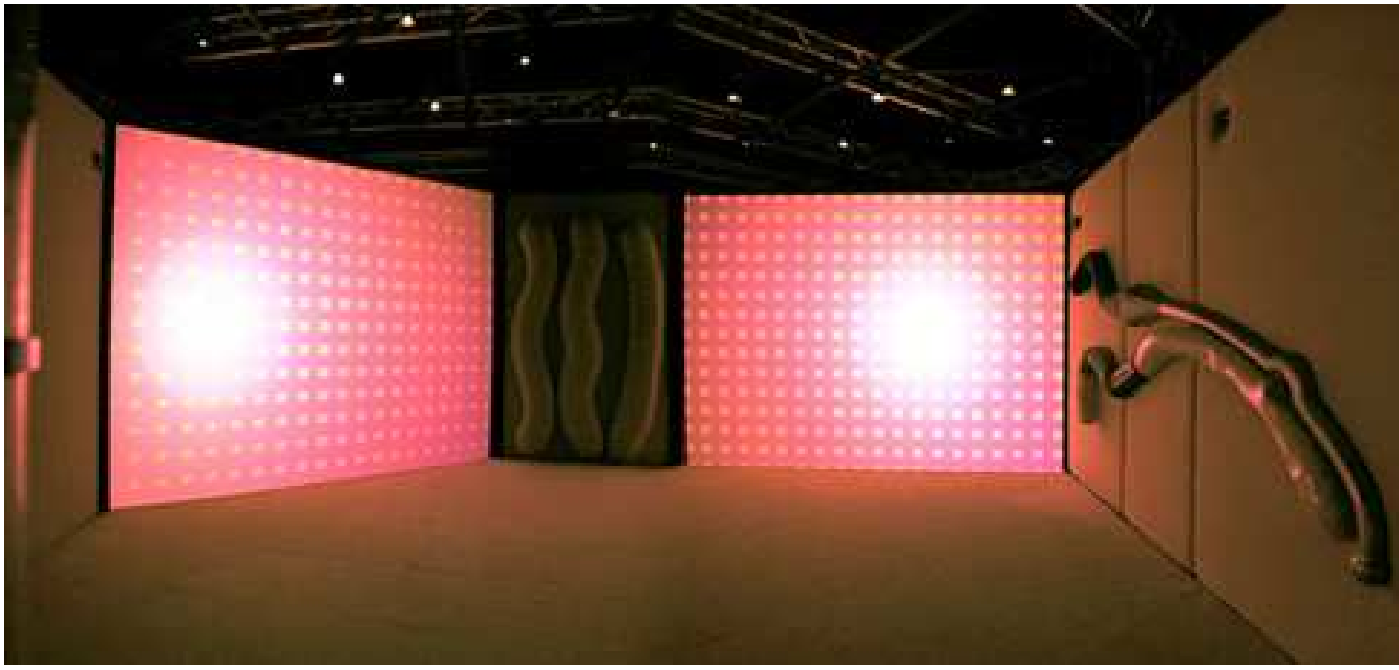


Figura 1.7: L'ambiente interattivo MEDIATE. Sono visibili gli schermi di proiezione e le superfici vibro-tattili.

Figura 1.8: Un utente interagisce con una proiezione all'interno dell'ambiente MEDIATE.

finanziario di diverse entità (The John D. and Catherine T. MacArthur Foundation, The Bill and Melinda Gates Foundation, Margulf Foundation) senza le quali non si sarebbe potuto realizzare.

Figura 1.9: L'ambiente interattivo multimodale SMALLab in una delle prime versioni presso l'Arizona State University.

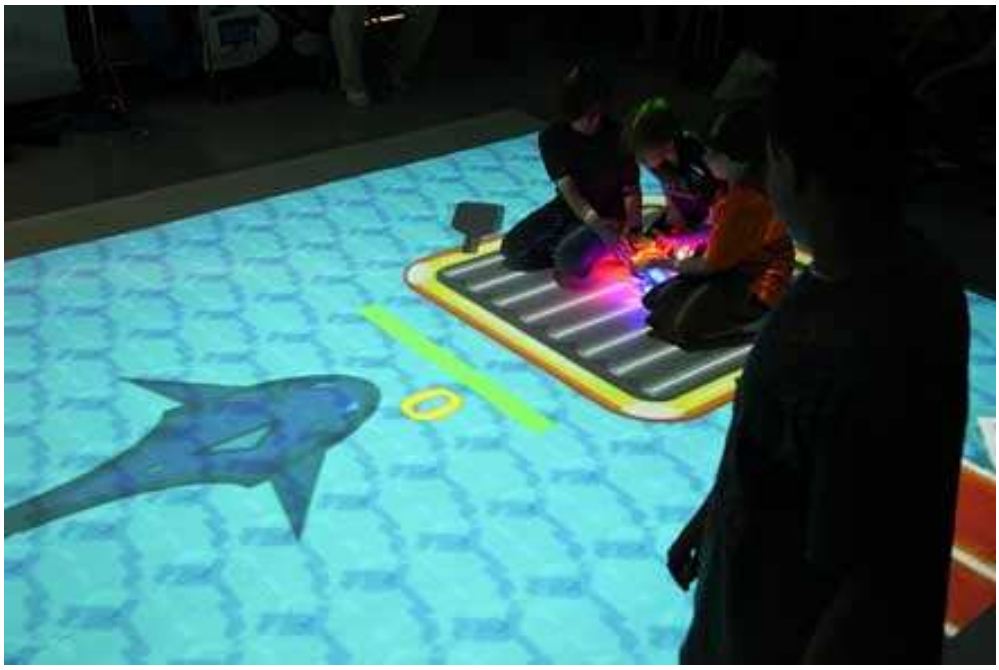


Figura 1.10: L'ambiente interattivo multimodale SMALLab nella scuola di New York dove è stata installata nel 2010.

Capitolo 2

La Stanza Logo-Motoria

Tutti i sistemi multimodali interattivi presentati nel capitolo precedente, sono accomunati da un costo elevato e l'utilizzo di tecnologie complesse, che richiedono per la loro gestione personale informatico e tecnico specializzato. Spesso costituiti da apparecchiature ingombranti e difficili da installare, il loro utilizzo è stato sempre ridotto dentro dentro confini limitati di tempo (nel corso di una visita o un progetto speciale) e di spazio (presso il laboratorio/istituzione dove l'ambiente era installato).
La certezza che l'adozione di tali tecnologie sarebbe davvero efficace se potesse contare su una maggiore diffusione e un utilizzo continuativo nelle scuole, affiancato alle ordinarie attività didattiche, ha portato alla progettazione e sviluppo del sistema Stanza Logo-Motoria. Tali premesse sono alla base delle seguenti linee guida adottate per la sua progettazione.

1. Basso costo della tecnologia utilizzata (computer, sensori, interfacce...). Necessario affinché il sistema possa essere adottabile in scuole di ogni ordine e grado, a prescindere da differenze economiche.

2. Usabilità del sistema. Con l'obiettivo di ottenere la maggiore diffusione possibile del sistema Stanza Logo-Motoria è indispensabile che gli insegnanti siano in grado di gestire/utilizzare il sistema senza difficoltà, pur non possedendo competenze avanzate di informatica.

3. Customizzabilità del sistema. Necessario affinché gli insegnanti siano in grado di adattare e produrre autonomamente i contenuti delle attività didattiche da svolgere nella Stanza Logo-Motoria.

4. Robustezza del sistema. In maniera da poter funzionare in un ampio spettro di condizioni, indipendentemente dall'ambiente in cui viene installato. Questo requisito è necessario per ridurre la necessità da parte della scuola di richiedere l'ntervento di tecnici per la manutenzione del sistema e dover per questo investire ulteriori risorse economiche.

Sulla base di queste linee guida il Centro di Sonologia Computazionale del Dip. di Ingegneria di Padova e Serena Zanolla, insegnante di sostegno e attualmente dottoranda di ricerca

dell'Università di Udine, hanno realizzato una versione della Stanza Logo-Motoria, che attualmente è installata presso la Scuola Primaria "E. Frinta" di Gorizia (Istituto Comprensivo Gorizia 1 [32]). Questa versione utilizza una *webcam* come dispositivo di acquisizione e l'ambiente di programmazione grafico EyesWeb XMI per l'elaborazione dei dati e per la gestione delle attività interattive. La webcam posizionata al centro del soffitto riprende il movimento degli utenti su una superficie quadrata 2.2. Degli algoritmi di motion-capture elaborano le immagini e restituiscono il baricentro delle persone che si muovono nello spazio.

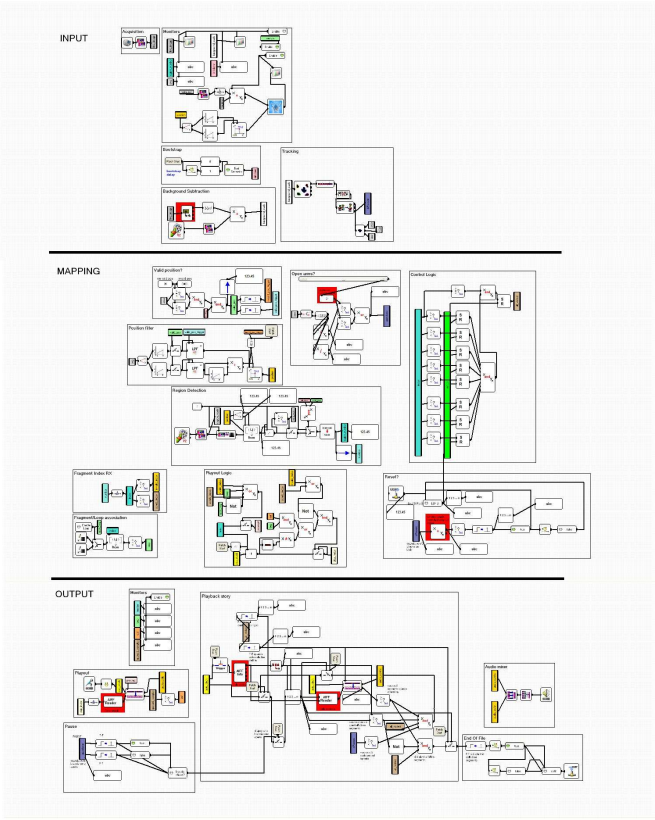


Figura 2.1: Patch di EyesWeb XMI del software prototipale della Stanza Logo-Motoria.

2.1 *Resonant Memory*

Un esempio di come può essere utilizzata la Stanza Logo-Motoria è l'applicazione *Resonant Memory*, derivante da una prima *patch* EyesWeb realizzata, nel 2008, con il supporto dell'InfoMus Lab dell'Università di Genova. Questa prevede che la superficie venga suddivisa virtualmente in nove regioni, otto periferiche e una centrale, a ognuna delle quali è associato un suono. L'applicazione implementa nella Stanza Logo-Motoria un'attività interattiva costituita da due fasi. La prima cosiddetta di "esplorazione" in cui vengono percorse le regioni periferiche, durante la quale all'allievo è richiesto di memorizzare il suono e la posizione che

Figura 2.2: La Stanza Logo-Motoria installata in un'aula della scuola "E. Frinta" di Gorizia.

gli corrisponde mentre la regione centrale rimane inattiva. Nella seconda fase all'utente è richiesto di recarsi nella regione centrale, attivata dopo aver percorso tutte le altre zone, e a questo punto inizia la riproduzione di una storia sotto forma di narrazione audio. La storia farà riferimento ad oggetti (luoghi, animali, concetti...), a ognuno dei quali corrispondono i suoni delle regioni periferiche. L'allievo è poi incoraggiato a ritrovare nelle varie regioni i suoni ascoltati durante l'esplorazione e a introdurli "fisicamente" nella narrazione, realizzandone così in tempo reale la colonna sonora. Il materiale didattico fruito attraverso questo tipo di attività interattiva è più facilmente memorizzato ed elaborato dagli allievi, che oltre alle usuali modalità percettive attivano anche le regioni della mente preposte al movimento. Sperimentando quindi un esperienza educativa in linea con le ricerche sull'apprendimento enattivo e l'*embodied learning* [33].

2.2 Una metodologia didattica

I positivi risultati ottenuti con la sperimentazione della Stanza Logo-Motoria in modalità *Resonant Memory* [17] hanno suggerito al gruppo di ricerca di estendere il sistema ad altre applicazioni. Ciò è già avvenuto con *Fiaba Magica* [34], un applicazione in cui gli allievi sono chiamati a interagire con i personaggi di una storia riprodotta attraverso dei contenuti audiovisuali. Con l'applicazione *Fiaba Magica* i movimenti degli utenti controllano la navigazione della storia (spostandosi nella superficie attiva, suddivisa in tre regioni) e l'animazione dei personaggi (muovendo le braccia). In futuro è prevista un'ulteriore estensione delle attività collegate con l'utilizzo della Stanza Logo-Motoria, attraverso lo sviluppo di nuove applicazioni. L'intenzione del progetto di ricerca è che la Stanza Logo-Motoria possa costituire un *framework* gerarchicamente strutturato in tre livelli.

Piattaforma. Ovvero i moduli hardware (videocamera o altri dispositivi di acquisizione, unità di elaborazione, diffusori, proiettore...) e software (programmi finalizzati all'in-

terpretazione dei dati in input derivanti dai dispositivi di acquisizione, alla rielaborazione e alla trasmissione delle informazioni sul movimento, alla gestione dei dispositivi di riproduzione audio e video e alla costituzione di un host per le applicazioni) che costituiscono l'infrastruttura necessaria al funzionamento del sistema.

Applicazioni. Pacchetti software, compatibili con la piattaforma, che implementano le differenti attività interattive che possono essere svolte con la Stanza Logo-Motoria.

Contenuti. Unità di apprendimento multimediali suddivise per ambito disciplinare e pensate per ogni diversa applicazione della Stanza Logo-Motoria.

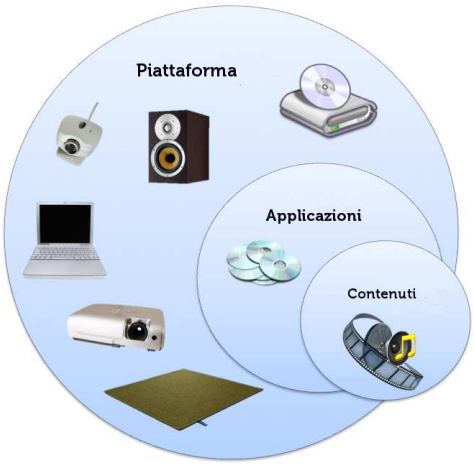


Figura 2.3: I tre livelli del sistema Stanza Logo-Motoria.

Capitolo 3

Tecnologie

3.1 Esigenze progettuali

La versione prototipale della Stanza Logo-Motoria ha indicato due possibili sviluppi del sistema, che inizialmente sono stati entrambi presi in considerazione come lavoro di tesi. Il primo riguarda la parte hardware del sistema, ovvero la tecnologia relativa all'acquisizione delle informazioni sul movimento degli utenti. Il secondo sviluppo è invece relativo al software, ovvero la parte di elaborazione dei dati relativi al movimento e di gestione dell'applicazione. Si è scelto inizialmente di utilizzare una *webcam* per la sua accessibilità economica e reperibilità sul mercato, e per il fatto che, a differenza di altre tecnologie per l'acquisizione di movimento, è immediatamente interfacciabile con il computer, e quindi particolarmente adatta per la realizzazione rapida di un prototipo funzionante. Ma volendo tracciare i movimenti dell'intero corpo degli utenti all'interno di uno spazio, la videocamera dev'essere posizionata al di sopra della superficie dove avviene l'interazione e questo pone dei vincoli sull'altezza della stanza in cui il sistema può essere installato. È stata quindi svolta una ricerca sulle possibili tecnologie di acquisizione di movimento nella letteratura scientifica e ne sono stati evidenziati vantaggi e svantaggi considerando le necessità della Stanza Logo-Motoria.
Il secondo possibile sviluppo del sistema è stato suggerito dalle sperimentazioni della Stanza Logo-Motoria effettuate presso la scuola primaria "E. Frinta" di Gorizia. L'ambiente di programmazione EyesWeb XMI con cui è stato realizzato il software prototipale richiede una serie di competenze informatiche che non sempre è possibile ritrovare nel personale delle scuole d'infanzia e primarie, a cui il sistema è principalmente rivolto. A ciò si aggiunge la mancanza di interfaccia grafica (vedi figura 2.1), il software instabile, il caricamento di contenuti didattici complicato. Nel corso della sperimentazione alle insegnanti è stato infatti affiancato un dottorando di ricerca, che oltre a creare contenuti didattici per la Stanza Logo-Motoria, assisteva la classe durante le sessioni con il sistema. Ma nella prospettiva che le scuole possano utilizzare la Stanza Logo-Motoria autonomamente nel corso delle delle attività didattiche ordinarie, è necessario che il programma sia fatto in modo da essere usabile anche da un utente generico.

Alla fine il lavoro di tesi sperimentale si è rivolto alla progettazione e sviluppo di un software che ricoprisse il ruolo svolto dall'ambiente di prototipazione EyesWeb XMI, fornendo al contempo una interfaccia grafica per permettere agli insegnanti di gestire agevolmente le applicazioni e i contenuti della Stanza Logo-Motoria. In ogni caso ad uso di futuri sviluppi del progetto sono riportate anche le ricerche riguardante i sistemi di *motion-capture* alternativi al video. Mentre nei paragrafi seguenti di questo capitolo vengono introdotte le tecnologie e le tecniche adoperate per lo sviluppo del nuovo software per la Stanza Logo-Motoria.

3.2 Tecnologie non-visuali per *motion-capture*

Per ogni tecnologia di *motion-capture* presa in esame sono stati evidenziati pro e contro rispetto alle esigenze del progetto Stanza Logo-Motoria. Sono stati presi in esame in particolare due aspetti

Necessità tecniche del sistema Stanza Logo-Motoria. Ovvero le prestazioni previste dalla tecnologia utilizzata come sistema di localizzazione/*tracking* di persone in una superficie quadrata di circa 5 metri per lato. Principale fonte di riferimento sono precedenti ricerche che hanno fatto uso di tale tecnologia e informazioni ricavate dai siti web di case produttrici di sensori.

Costi di realizzazione del sistema. Tenendo in considerazione che il progetto è pensato per essere utilizzato come supporto all'apprendimento in scuole primarie e considerando la limitatezza dei fondi di cui queste dispongono, è stato attribuito un valore positivo a soluzioni tecnologiche di basso costo.

3.2.1 Tecnologia *Acoustic Tabletop*

Questa tecnologia consiste nel localizzare l'origine di un evento sonoro prodotto dall'interazione tra un oggetto ed una superficie inerte utilizzando un certo numero di microfoni o accelerometri posizionati ai bordi dell'area attiva. Tipicamente è utilizzato per realizzare superfici interattive senza l'utilizzo di telecamere in cui l'interazione si limita principalmente ad azioni del tipo mouse-click realizzati attraverso colpetti con le dita (*tap*) sulla superficie. Nessuna ricerca che utilizzasse tale tecnologia come sistema di localizzazione di persone è stata trovata, sarebbe quindi da valutare se le vibrazioni prodotte dai passi sono rilevabili o meno da strumenti come microfoni o accelerometri e in quali superfici si hanno risultati migliori.
Soluzioni implementative [35].

TDOA (Time Delay of Arrival). La posizione viene localizzata misurando i differenti tempi di propagazione dell'onda sonora dalla sorgente ai ricevitori.

LTM (Location Template Matching). Considerando che l'interazione sonora con la superficie produce una eccitazione diversa per ogni suo punto, si registrano i suoni generati

dall'interazione in una mappa discreta di punti della superficie. Dopo questa fase di *learning*, gli eventi vengono localizzati confrontandoli statisticamente con la base di dati acquisita.

Il vantaggio ricavato dall'adozione di questa tecnologia è che i piezoelettrici, microfoni a elettrete e accelerometri utilizzati nell'analisi comprensiva di tale tecnologia di Pham [35] sono dispositivi facilmente reperibili e a basso costo. D'altra parte questa tecnologia richiede che venga prodotto un suono con, come origine, il punto da localizzare. L'energia acustica prodotta dai passi dell'utente potrebbe non essere sufficientemente grande da essere rilevata dai sensori. L'utilizzo di superfici con specifiche caratteristiche acustiche potrebbe essere una soluzione a tale problema, ma a quel punto sarebbe da valutare la possibilità di disporre di una ampia superficie di tale materiale e il costo che questo comporterebbe.

3.2.2 Tecnologia a ultrasuoni (SONAR)

Un emettitore produce un suono dalle frequenze superiori al range udibile. La presenza di un corpo lungo il percorso di tale suono causa un onda riflessa che poi viene recepita da un ricevitore. Dall'informazione sul suono ricevuto è possibile localizzare la posizione di tale corpo. Diverse possibilità implementative sono valutabili, relative al numero di emettitori, ricevitori e alla geometria del sistema sensoristico. Tale scelta è funzionale alle due principali tipologie di analisi adottabili per l'acquisizione di dati da questa tecnologia.

TDOA (Time Delay of Arrival). Lo stesso principio che si ha nel caso della tecnologia per *Acoustic Tabletop*. In questo caso vengono prodotti segnali impulsi (*ping*) che, riflessi da un corpo, vengono recepiti da un ricevitore, solitamente posizionato vicino all'emettitore. Il tempo che intercorre tra l'istante in cui il suono è stato prodotto e quello in cui è stato ricevuto fornisce informazione della distanza tra il corpo ed il sensore [36]

Analisi effetto doppler. In questo caso il segnale trasmesso è un onda continua, non un impulso, che viene ancora riflessa da un corpo e recepita da un ricevitore. L'informazione sul movimento del corpo è ottenuta dalla variazione di frequenza tra il segnale prodotto e quello ricevuto, causata dall'effetto Doppler [37], [38].

Con questa tecnologia il *tracking* non è condizionato dalle proprietà acustiche della superficie e dell'intensità di suono prodotta dai passi degli utenti. Anche se superiore alla tecnologia *acoustic tabletop*, per un utilizzo limitato alla localizzazione di persone in uno spazio ridotto, come nel sistema Stanza Logo-Motoria, la tecnologia sonar può essere utilizzabile mantendosi con bassi costi. D'altra parte ci sono anche dei punti a sfavore.

- Sensibilità del sistema a vento e temperatura.

- Emettitori SONAR standard per applicazioni di basso costo raggiungono una massima distanza di circa sei metri. Dispositivi disponibili nel mercato [1] costituiti da un emettitore e un ricevitore posizionati nella stessa scheda hanno inoltre un angolo di rilevazione massimo di 50ř. Sia nel caso di un sistema emettitore-ricevitore specifico che nell'utilizzo dei sensori di prossimità a ultrasuoni disponibili nel mercato è necessario l'utilizzo di più emettitori per coprire tutta l'area attiva del sistema Stanza Logo-Motoria.

- La modalità con onda continua ed effetto Doppler risulta molto più efficace per *tracking* del movimento, ma in una configurazione in cui sono presenti più emettitori, che per problemi di interferenza tra diverse onde devono essere attivati singolarmente, l'informazione continua sul movimento non è più ottenibile rendendo così tale tecnica non applicabile al meglio.

3.2.3 Tecnologie a infrarossi. *Range finders*

Sensori di questo tipo funzionano inviando un impulso di luce infrarossa e misurando l'intensità della luce riflessa ricevuta dal sensore attraverso un foto-transistor [2]. Anche questa tecnologia è facilmente implementabile e sono anche disponibili nel mercato sensori di prossimità a basso costo che la utilizzano. Per quanto riguarda invece i problemi legati all'adozione di questi dispositivi, sono i seguenti.

- I sensori ad infrarossi sono strettamente monodirezionali, quindi anche in questo caso sono necessari array di sensori per le esigenze del sistema Stanza Logo-Motoria ed in particolare si avrebbe bisogno di molti più sensori di quelli a ultrasuoni.

- Comportamento non lineare nel rilevare la distanza dall'oggetto da localizzare.

- Il funzionamento di questa tecnologia dipende molto dalla riflettività dell'oggetto rilevato.

- Range massimo di 3 metri, quindi anche nel caso di disporre array di sensori in tutti i lati della superficie del sistema Stanza Logo-Motoria potrebbero esserci problemi nel localizzare la posizione di persone nel centro della superficie.

[1] Disponibili sul mercato dispositivi a basso costo (intorno ai 20-60$) composti da recettore e trasmettitore che funzionano con la tecnica dell'impulso e calcolo del ritardo dell'onda riflessa (*Robot Electronics SRF*, *Maxbotix*, *Parallax PING*). In particolare i dispositivi prodotti da *Maxbotix* già prevede l'utilizzo di più sensori simultaneamente attraverso un pin in ingresso di lettura ed un impulso prodotto alla fine di ogni lettura che può essere inviato al sensore successivo.

[2] Il più popolare è quello prodotto da Sharp www.acroname.com/robotics/info/articles/sharp/sharp.html

3.2.4 *Motion-capture* mediante campo elettrico

Questa tecnologia che sfrutta lo stesso principio di funzionamento del Theremin [3]. Ovvero la proprietà del corpo umano di caricare capacitivamente un componente di un circuito elettrico (antenna, superficie metallica...) in maniera diversa a seconda della sua posizione. Le soluzioni Implementative trovate sono le seguenti [40], [41].

Shunt. Il corpo intercetta l'onda EM e la scarica a terra, diminuendo così la corrente indotta dal campo EM nel ricevitore.

Transmit. Riguarda la modalità di contatto del corpo con il trasmettitore, tanto più esso si avvicina al ricevitore tanto più aumenta la corrente indotta in quest'ultimo.

Loading mode. Questa è la modalità utilizzata per il Theremin. Si dispone di un solo elettrodo che funge da trasmettitore e ciò che viene misurato è la corrente spinta da questo al corpo.

Questa tecnologia è particolarmente robusta. Non è influenzata da fonti di disturbo esterne come invece nel caso della luce solare per i sensori a infrarosso e rumore meccanico e variazioni di temperatura per sensori a ultrasuoni. Inoltre non prevede l'utilizzo di tecnologie avanzate. Quindi nonostante non siano disponibili nel mercato sistemi di rilevazione che sfruttano tale tecnologia, la sua implementazione non richiede costi elevati. D'altra parte il problema di ottenere informazioni sull'oggetto che perturba il campo elettrico dalle cariche misurate non è lineare. Rendendo quindi di non facile implementazione tale sistema di rilevazione.

3.2.5 Superficie munita di Sensori di forza

Tale tecnologia prevede l'utilizzo di una superficie munita di sensori. Differenti maniere di implementare tale sistema sono state realizzate [42], [43], ma in entrambi i casi si tratta di strutture costose e/o di difficile realizzazione oltre che difficilmente trasportabili. Tale soluzione è quindi difficilmente adottabile per il sistema Stanza Logo-Motoria.

3.2.6 Conclusioni

Le esigenze del sistema Stanza Logo-Motoria, principalmente limitate alla localizzazione di persone nell'area attiva ed eventualmente estese al *tracking* del movimento, ci permettono di escludere l'utilizzo una superficie munita di sensori di forza. I costi di quest'ultima sono

[3] Strumento musicale inventato nel 1919 dal fisico sovietico Lev Sergeevic Termen, noto in occidente col nome di Léon Theremin. Questo strumento è composto fondamentalmente da due antenne poste sopra e a lato di un contenitore nel quale è alloggiata tutta l'elettronica. Il controllo avviene allontanando e avvicinando le mani alle antenne: mediante quella superiore (posizionata verticalmente) si controlla l'altezza del suono, quella laterale (posta orizzontalmente) permette di regolarne l'intensità. Il suono può variare tra quello di un violino e quello vocale. Lo strumento è considerato molto difficile da suonare proprio perché lo si utilizza senza toccarlo [39]

giustificati dalle sue prestazioni che però sono al di la delle necessità del sistema Stanza Logo-Motoria. Per quanto riguarda la tecnologia degli *Acoustic Tabletop*, data l'assenza di documentazione riguardo al suo utilizzo in architetture simili a quella della Stanza Logo-Motoria, la sua fattibilità andrebbe valutata più in dettaglio. Le tre restanti tecnologie: Sonar, Sensori IR e rilevazione mediante campo elettrico, sono tutte valide, anche se con alcuni limiti riguardo la distanza massima di rilevazione (che potrebbe dare dei problemi nel caso dei sensori a infrarosso). Le prime due permetterebbero di ottenere risultati più immediati data la reperibilità nel mercato di sensoristica a basso costo che implementa tale tecnologia, mentre la rilevazione mediante campo elettrico richiederebbe uno studio dedicato.

3.3 Openframeworks

3.3.1 Introduzione

Openframeworks è un toolkit C++ per *creative coding*. Ovvero pensato per dare la possibilità di creare le proprie applicazioni di audio, grafica e interattività utilizzando un linguaggio potente come C++ ma senza doversi occupare di integrare singolarmente le librerie necessarie per questi tipi di applicazione. Inoltre Openframeworks è scritto per essere il più possibile indipendente dalla piattaforma in cui viene utilizzato. Utilizzando solo le librerie del toolkit *core* (senza quindi aggiungere parti esterne, che possono esser dipendenti da sistema operativo) il codice può essere compilato indipendentemente nelle piattaforme Windows, OS X, Linux, iOS e Android.
Uno dei creatori del progetto, Zach Lieberman utilizzava per i suoi lavori una libreria per C++ distribuita privatamente, sviluppata al MIT Media Lab's Aesthetics and Computation Group. Lieberman avrebbe voluto condividerla con i suoi studenti alla Parson School of Design dove insegnava, ma questa non era molto mantenuta e soprattutto non aveva licenze aperte. Quindi per avere la possibilità di permettere agli studenti di sperimentare con gli strumenti software da lui stesso usati, iniziò a costruire una libreria ispirandosi a quella del MTI, ma con licenze *opensource*. Theo Watson, uno dei suoi ex studenti si unì al progetto ed insieme decisero di renderla pubblica.
Oltre alla libreria ACU, lo sviluppo di Openframeworks è stato ispirato dal software e dalla comunità di Processing [4]. Dopo la creazione di un sito web e grazie all'aiuto di artisti e informatici nella comunità di New Media Art, il progetto iniziò a farsi conoscere su scala mondiale e si venne a creare una forte comunità online (forum, wiki, tutorials...) e offline (workshop, festival, eventi...). Ad oggi il progetto Openframeworks è mantenuto da Zach Lieberman, Theodore Watson e Arturo Castro (*the core*) insieme a una attiva comunità di utenti [45], [46].
Il software è distribuito secondo la licenza MIT, ovvero il suo codice è liberamente utilizzabi-

[4]Processing è un linguaggio e ambiente di sviluppo *opensource* basato in Java che consente di sviluppare applicazioni per immagini, animazioni e interattività. Nato inizialmente allo scopo di costituire una piattaforma per l'apprendimento di programmazione in un contesto grafico, si è successivamente evoluto in uno strumento completo anche per applicazioni commerciali [44]

le e modificabile, ma non è necessario ridistribuire lavori derivati con la stessa licenza, come è ad esempio richiesto dalla licenza GPL.

3.3.2 Struttura software

Nel corso della stesura della presente tesi è stata rilasciata la versione 07.1, con alcune differenze rispetto alla precedente in alcuni percorsi di file. Il lavoro sperimentale è però iniziato con la versione 07 ed è quindi a questo che si farà riferimento di seguito. Il toolkit Openframeworks è composto da un pacchetto software differente a seconda della piattaforma utilizzata. La versione per Windows ha una struttura come quella mostrata nella figura 3.1 Il contenuto di ogni cartella è il seguente:

Figura 3.1: Struttura delle cartelle di Openframeworks in Windows

addons

Questa cartella contiene funzioni di Openframeworks aggiunte dagli utenti, ma che non fanno parte delle funzionalità base del toolkit (API per librerie esterne, funzioni d'uso comune in qualche ambito...). È necessario includere specificatamente ognuno di questi *addon* nel proprio progetto se si intende utilizzarli.

apps

I propri programmi dovrebbero essere localizzati in sottocartelle di questa. Naturalmente è possibile scegliere una disposizione differente a seconda delle proprie preferenze, ma dato che i percorsi relativi delle varie librerie e codice fanno riferimento a una struttura di questo tipo, è consigliabile attenervisi.

libs

Qui sono contenuti i file delle librerie che compongono le funzionalità base di Openframeworks. Inoltre nella cartella *openframeworks* sono presenti tutti gli *header* che utilizzano tali librerie e definiscono le funzionalità base, come descritto più in dettaglio di seguito.

openframeworks

La cartella *openframeworks* all'interno di *libs* contiene undici sottocartelle, ognuna per una specifica attività del toolkit.
La cartella *apps* contiene gli *header* per la classe *ofBaseApps*, che è la classe di base per ogni programma Openframeworks. Questa permette di creare una finestra, utilizzare funzionalità base per disegnare su schermo e interagire con mouse e tastiera. La cartella *communication* contiene strumenti per comunicare con il *mondo reale* via porte seriali e la cartella *events* gestisce la ricezione e trasmissione di eventi in un'applicazione Openframeworks.
La cartella *graphics* contiene cinque classi: *ofBitmapFont* per lavorare con caratteri sotto forma di bitmap, *ofGraphics* per grafica 2D, *ofImage* per immagini in bitmap, *ofTexture* per le texture di OpenGL e *ofTrueTypeFont* per lavorare con caratteri di testo.
La cartella *sound* contiene due classi per lavorare con i suoni, *ofSoundPlayer* per creare e riprodurre file audio e *ofSoundStream* per avere la possibilità di interagire in più basso livello coi suoni.
La cartella *video* contiene codice per acquisire il flusso video da una camera connessa al computer e per riprodurre video acquisito o memorizzato nel computer.
Infine la cartella *utils* contiene codice di utilità generica. Funzioni matematiche, per acquisire e misurare il tempo o strutture dati utili per tipici programmi di Openframeworks.
Per quanto riguarda le librerie utilizzate, condizione fondamentale è che esse siano compatibili con la licenza di Openframeworks. Ovvero liberamente distribuibili ma senza la condizione di *share alike* ovvero l'obbligo di ridistribuire liberamente il software che le utilizza. Ad oggi le principali librerie utilizzate dal core di Openframeworks (ovvero quelle per cui non c'è bisogno di includere per le diverse piattaforme sono le seguenti.

- OpenGL, GLEW, GLUT, libtess2 e cairo per la grafica.
- rtAudio, PortAudio o FMOD e Kiss FFT per l'acquisizione, la riproduzione e l'analisi audio.
- FreeType per i caratteri.

- FreeImage per il salvataggio e il caricamento di immagini.
- Quicktime e videoInput per la riproduzione e l'acquisizione video.
- Poco per varie utilità.

3.3.3 Struttura di un programma di Openframeworks

Un tipico programma di Openframeworks è composto dai file *main.cpp*, *testApp.h*, *testApp.cpp*. Il primo è responsabile di *lanciare* l'applicazione , quindi creare una finestra su schermo ed eseguire le funzioni contenute nella classe *testApp* che eredita da *ofBaseApp*.
ofBaseApp è quindi la struttura centrale di un applicazione Openframeworks. I suoi metodi sono tutti relativi alla gestione di eventi. Scrivere un'applicazione consiste principalmente nell'implementare le funzioni definite virtualmente in *ofBaseApp()*, le quali verranno chiamate in occasione di determinati eventi (del programma o esterni) e aspettare che questi vengano chiamati nell'esecuzione dell'applicazione. Di seguito si mostra la struttura del file *testApp.h* di cui verrà poi descritto ogni comando.

```
#include "ofMain.h"

class testApp : public ofBaseApp{

public:
void setup();
void update();
void draw();
void exit();

void keyPressed  (int key);
void keyReleased(int key);
void mouseMoved(int x, int y );
void mouseDragged(int x, int y, int button);
void mousePressed(int x, int y, int button);
void mouseReleased(int x, int y, int button);
void windowResized(int w, int h);
void dragEvent(ofDragInfo dragInfo);
void gotMessage(ofMessage msg);
.
.
.


};
```

ofMain.h

ofMain è il file in cui sono inclusi tutti gli *header* delle librerie di base di Openframeworks.

setup()

Questa funzione viene chiamata per prima e per una sola volta quando il programma viene eseguito. Qui si hanno inizializzazione di variabili, allocamenti in memoria...

update()

Questa funzione è chiamata per ogni ciclo di funzionamento del programma. Aggiornamenti di dati, strutture condizionali e in generale tutto ciò che non riguarda la parte grafica del programma va inserito in questa funzione.

draw()

In ogni ciclo dell'applicazione, dopo *update()* è eseguita questa funzione. Questa funzione è ottimizzata in maniera da eseguire al meglio funzioni relative alla grafica.

exit()

Questa funzione è chiamata quando si esce dall'applicazione. Funzioni di salvataggio finali e relative al disallocamento di dati in memoria vanno posizionate qui.

keyPressed(), keyReleased()...

Oltre alle quattro precedenti ci sono altre funzioni definite in *ofBaseApp* che vengono eseguite in occasione di specifici eventi. Queste riguardano eventi relativi a periferiche (la tastiera, il mouse, dispositivi di input audio...) e ad altri tipi di eventi (*drag-drop* di file, eventi relativi alla finestra di esecuzione...)

3.4 OpenCV

3.4.1 Introduzione

OpenCV è una libreria *opensource* di *computer vision* scritta in C e C++ per le piattaforme Windows, Linux e Mac OS X. OpenCV è progettata con una forte attenzione per applicazioni in tempo reale, quindi con particolare riguardo all'efficienza computazionale. Uno degli obiettivi della libreria è permettere agli utenti di costruire applicazioni avanzate di *computer vision* in maniera rapida ed efficiente. OpenCV comprende oltre 500 funzioni di molte aree della *computer vision*: ispezione di prodotti industriali, immagini mediche, sicurezza, interfacce utente, calibrazione di camere, robotica e visione stereo. La licenza *opensource* di

OpenCV non esclude il suo utilizzo in progetti commerciali, e di fatto, molte tra le principali compagnie informatiche (IBM, Microsoft , Intel, SONY, Siemens, Google...) e maggiori centri di ricerca (Stanford, MIT, CMU, Cambridge, INRIA...) fanno uso della libreria. Esiste inoltre una grande comunità di utenti distribuita in tutto il mondo, che ad oggi include oltre 20.000 membri[5].
Intel allo scopo di testare applicazioni intensive per la cpu iniziò diversi progetti. Tra questi *real time ray tracing* e *3d display walls*. Visitando centri di ricerca universitari uno degli autori vide che al MIT Media Lab nel corso degli anni avevano sviluppato un'infrastruttura software per *computer vision*, in modo che non ci fosse sempre bisogno di riscrivere da zero funzioni di uno comune. OpenCV nasce con gli stessi obiettivi, ma con la volontà di rendere una simile libreria disponibile a tutti, utilizzando licenze *opensource*. Allo sviluppo del software hanno contribuito anche i gruppi di Intel Performance Library Team e il laboratorio di ricerca di Intel in Russia che avevano le competenze per far si che il programma fosse il più possibile ottimizzato in termini di performance computazionale. Inizialmente Intel non appoggiò esplicitamente il lavoro, anche se questo non fu comunque osteggiato. La ragione per cui una compagnia di hardware scelse poi di supportare un così ambizioso progetto è nel fatto che secondo la direzione dell'azienda questo avrebbe contribuito alla diffusione di applicazioni per *computer vison* e conseguentemente a una maggiore richiesta di processori con elevate capacità.
Gli obiettivi che il team si propose con la creazione di OpenCV sono.

- Avanzare la ricerca sulla *computer vision* fornendo non solo codice aperto ma anche ottimizzato per le funzioni più comuni.

- Diffondere la conoscenza sulla *computer vision* fornendo una infrastruttura software da cui costruire le applicazioni, in modo che il codice possa essere più facilmente leggibile e scambiabile.

- Innovare le applicazioni commerciali basate sulla *computer vision* fornendo una libreria ottimizzata computazionalmente e liberamente disponibile, ma senza una licenza che obbliga lavori derivati a essere distribuiti allo stesso modo.

3.4.2 OpenCV e Openframeworks

Openframeworks fornisce un *addon* per utilizzare la libreria OpenCV nei suoi programmi. Includendo i relativi *file* al progetto è quindi possibile accedere alle funzioni di OpenCV, per alcune delle quali è anche previsto un codice di *wrapping* scritto nello stile di Openframeworks. Inoltre viene definita la classe *ofxCvImage*, che ha un interfaccia simile a quella della classe di base per la gestione di immagini in Openframeworks *ofImage*, e allo stesso tempo può venire utilizzata senza bisogno di conversioni di formato con le funzioni di OpenCV.

[5]Il forum della comunità si trova all'indirizzo groups.yahoo.com/group/OpenCV

3.5 Analisi video e *motion-capture*

Per quanto riguarda la parte del programma relativa all'elaborazione dei dati video, l'esigenza primaria della Stanza Logo-Motoria è quella di individuare le persone (in particolare una, nell'applicazione *Resonant Memory*) e tracciarne i movimenti. La tecnica di *motion-capture* adoperata prevede tre fasi.

***Background learning/subtraction*.** Acquisizione di informazioni sullo sfondo e successivo isolamento del oggetto di interesse dalla scena.

Ottimizzazione. Eliminazione di rumore e ottimizzazione della figura da tracciare per mezzo di filtri e trasformazioni morfologiche. Questa fase deve estrarre dalla scena la silhouette quanto più possibile accurata delle persone nella stanza.

***Contour finder/blob tracking*.** Identificazione e tracciamento dei movimenti figura mediante il suo baricentro.

3.5.1 *Background learning/subtraction*

Il problema della estrazione di un soggetto da una scena statica è tipico delle principali applicazioni della *computer vision*. Le tecniche più utilizzate prevedono la costruzione di un modello dello sfondo e un successivo confronto tra tale modello e i successivi *frame* del video acquisiti, in maniera di identificare e isolare oggetti che prima non erano presenti nella scena.
Le difficoltà relative alla costruzione del *background model* dipendono soprattutto dal fatto che lo sfondo non è mai costituito da una immagine statica ma varia con il tempo, in grado maggiore o minore a seconda delle situazioni. Si possono distinguere i seguenti tipi di variazioni dello sfondo.

- Cambi di geometrie. Nella scena di sfondo vengono introdotti/eliminati alcuni oggetti.
- Cambi di movimento. Sono presenti nella scena alcuni oggetti che si muovono col tempo, in maniera più o meno periodica.
- Variazioni di luminosità. Graduali o improvvisi, dovuti ad illuminazione elettrica o naturale.

Per quanto riguarda i cambi di geometrie o oggetti in movimento, la Stanza Logo-Motoria prevede che gli utenti abbiano la più possibile libertà di movimento all'interno della superficie del sistema. Questo esclude la possibilità che nella scena ci siano oggetti estranei e che questi possano venire spostati nel corso di una sessione didattica con la Stanza Logo-Motoria. Rispetto ai cambi di luminosità è necessario fare una precisazione. Il sistema della Stanza Logo-Motoria è attivo soltanto nel corso del suo utilizzo. Se quindi il modello di sfondo viene costruito all'inizio di ogni attivazione, questo sarà per la maggior parte dei casi sufficiente a rappresentare la situazione della scena di sfondo nel corso delle due ore successive, tempo

tipico di utilizzo del sistema. Disturbi al corretto funzionamento del sistema possono invece essere introdotti da cambi improvvisi di luminosità (ad esempio l'accensione o lo spegnimento di una luce all'interno dell'ambiente). Ma dato che il sistema funziona sempre sotto la supervisione di un istruttore, si riserva a lui il compito di riavviare il sistema, in maniera che un nuovo modello di sfondo venga costruito. Tali considerazioni evidenziano come il sistema non richieda particolari accorgimenti riguardo alle variazioni graduali e improvvise di luminosità, come invece nel caso dei sistemi di videosorveglianza, che rimanendo attivi in un arco di tempo molto lungo devono funzionare sotto condizioni di luce molto diverse.
Viste le condizioni favorevoli si è optato per l'adozione di tecniche convenzionali, fondate sull'ipotesi che nelle immagini i pixel adiacenti siano tra loro indipendenti. In alcuni casi tale ipotesi non è applicabile, ed esistono a questo proposito dei metodi che per ogni pixel conservano informazioni sul suo valore e sul valore dei pixel circostanti. Tali tecniche richiedono almeno il doppio di memoria e risorse computazionali [47], quindi dati i risultati soddisfacenti ottenuti con tecniche più tradizionali, si è preferito limitarsi a queste ultime.

Averaging background method

Il metodo utilizzato è basato sulla modellizzazione statistica dello sfondo per ogni canale di colore e sulla rilevazione con *threshold* adattativo [47], [48]. Dopo aver costruito un modello statistico dello sfondo, i nuovi *frame* vengono confrontati con esso e si ha la produzione di quella che si chiama una *confiance image*. Ovvero un immagine in scala di grigi, dove il valore di ogni pixel rappresenta la probabilità che esso faccia parte di una regione dell'immagine non appartenente allo sfondo.
Prima di discutere nel dettaglio le procedure con cui il modello dello sfondo viene calcolato, risulta utile chiarire alcune caratteristiche fisiche riguardo al colore e la sua rappresentazione del caso di camere di tecnologia *Capacity Coupled Device* o *Active Pixel Sensor* [49]. Ci sono tre caratteristiche della rilevazione dell'immagine che debbono essere tenute in considerazione.

Variazione di colore. Anche in una ripresa video statica, il valore di un dato pixel non è costante, ma varia continuamente a causa di fluttuazioni dell'illuminazione e del rumore della camera

Sbilanciamento di banda. Tipicamente le videocamere hanno differenti sensibilità per colori differenti, che si traduce in una diversa deviazione standard nel corso di un certo intervallo temporale. È necessario tenerne conto in modo da poter confrontare correttamente i valori di pixel per ogni canale.

Saturazione. I sensori delle camere permettono un range dinamico limitato, che restringe la variazione di colore in un cubo RGB in cui i colori primari rosso, verde e blu formano gli assi ortogonali. In immagini rappresentate con 24-bit, un colore viene rappresentato da una punto nello spazio tridimensionale all'interno del cubo limitato dai vertici $[0, 0, 0]$ e $[255, 255, 255]$. Colori all'infuori di esso non possono essere rappresentati e di conseguenza i valori dei pixel vengono saturati alla superficie cubica. Ciò comporta

una varianza quasi nulla per i pixel saturati, ma che non corrisponde alla situazione reale. In considerazione di questo è necessario porre un valore minimo di deviazione standard per i pixel.

Alla luce di queste considerazioni la costruzione del modello di sfondo viene fatta calcolando per ogni canale la media e la deviazione standard di ogni pixel nel corso di un certo numero di *frame,* acquisiti da riprese statiche sullo sfondo. Si ha quindi l'espressione

$$\mu_t = \alpha x_t + (1-\alpha)\mu_{t-1}$$

dove μ_t è la media calcolata fino al frame t, α è il *coefficiente di apprendimento* del modello e x_t il valore del pixel al frame t. Questa calcola una media pesata di tipo esponenziale di tutti i precedenti valori del pixel. Sottraendo i *frame* video successivamente all'acquisizione dello sfondo permetterà di identificare quali pixel hanno cambiato valore.
È poi definita la deviazione standard, che per ogni pixel è calcolata come

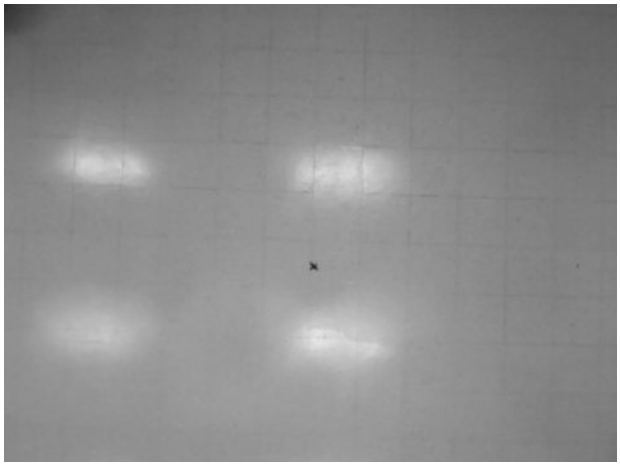
$$\sigma_t^2 = \alpha(x_t - \mu_t)^2 + (1-\alpha)\sigma_{t-1}^2.$$

Le immagini derivanti saranno utilizzate per la normalizzazione della *confiance image*. Nelle figure 3.2 e 3.3 sono riportate le immagini della media e della deviazione standard di due diverse riprese statiche [6] per i canali rosso e blu, calcolate utilizzando 50 *frame*, in cui si possono osservare differenti distribuzioni di colore nelle due scene. Dopo aver acquisito il modello dello sfondo, i *frame* video successivi vengono sottratti alla immagine della media, canale per canale. A questo punto per ogni immagine differenza si realizza una normalizzazione su ogni singolo pixel, utilizzando due *threshold* $m\sigma$ e $M\sigma$ ricavati dalle immagini di deviazione standard (a cui è stato fissato un valore minimo, per i motivi spiegati in precedenza). Se il valore della differenza è inferiore a $m\sigma$, la *confiance* C^c (ovvero la probabilità che il pixel analizzato corrisponda ad una regione non facente parte dello sfondo) è posta a 0%; se è superiore a $M\sigma$ la *confiance* è posta a 100%; se è in un valore intermedio il valore di *confiance* viene scalato linearmente secondo l'espressione

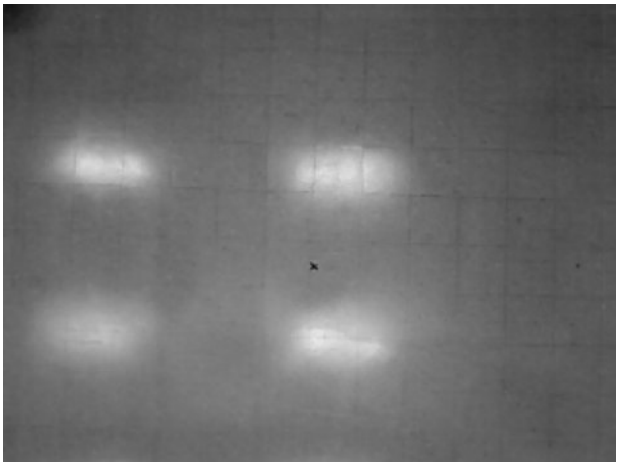
$$C^c = \frac{D - m\sigma}{M\sigma - m\sigma} X 100$$

dove D è il valore di differenza. Poiché poi una variazione in ogni canale di colore può indicare la presenza di una regione non appartenente allo sfondo, la *confiance image* finale avrà come valore di ogni pixel il massimo del pixel corrispondente nelle tre immagini dei canali di colore. Per quanto riguarda i valori di $m\sigma$ e $M\sigma$ essi vengono ricavati dal prodotto tra l'immagine di deviazione standard σ e due numeri scalari m e M. Le *confiance image* e le immagini dei *frame* da cui sono state calcolate, per le due sequenze video precedenti, sono riportate nella figura 3.4, dove 3.4a e 3.4c sono quelle relative alla Stanza Logo-Motoria.

[6] La sequenza da cui sono state ricavate le immagini 3.2a, 3.2b e 3.3a, 3.3b è stata acquisita da una istallazione della Stanza Logo-Motoria, mentre quella le immagini 3.2c, 3.2d e 3.3c e 3.3d sono state ricavate a partire da uno dei video di test messi a disposizione dall'ISE Lab [50]



(a) Canale rosso

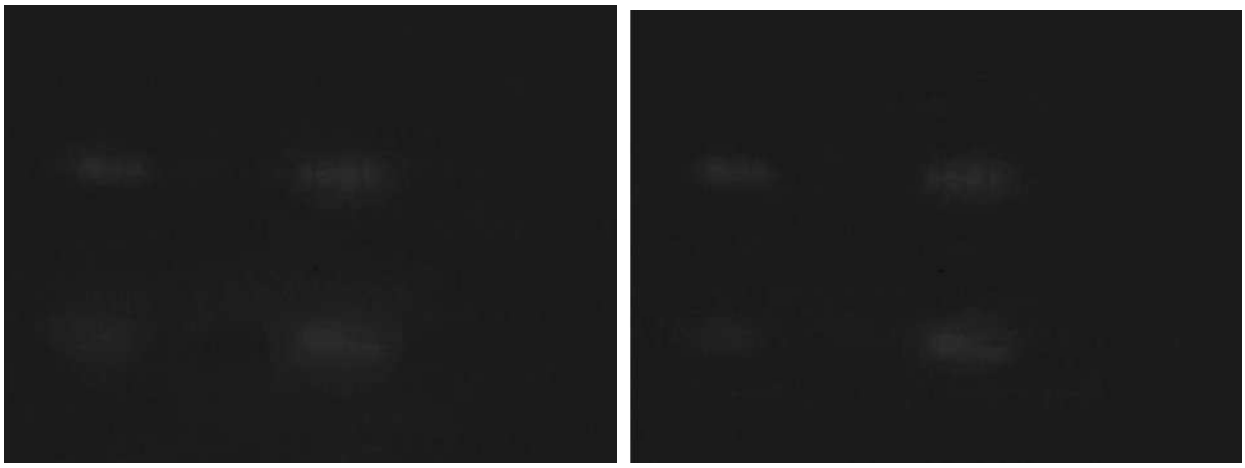


(b) Canale blu

(c) Canale rosso

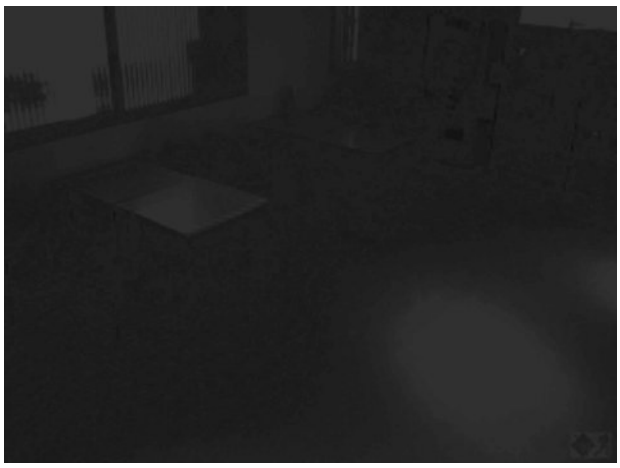
(d) Canale blu

Figura 3.2: Medie calcolate da una ripresa statica di 50 *frames*



(a) Canale rosso

(b) Canale blu

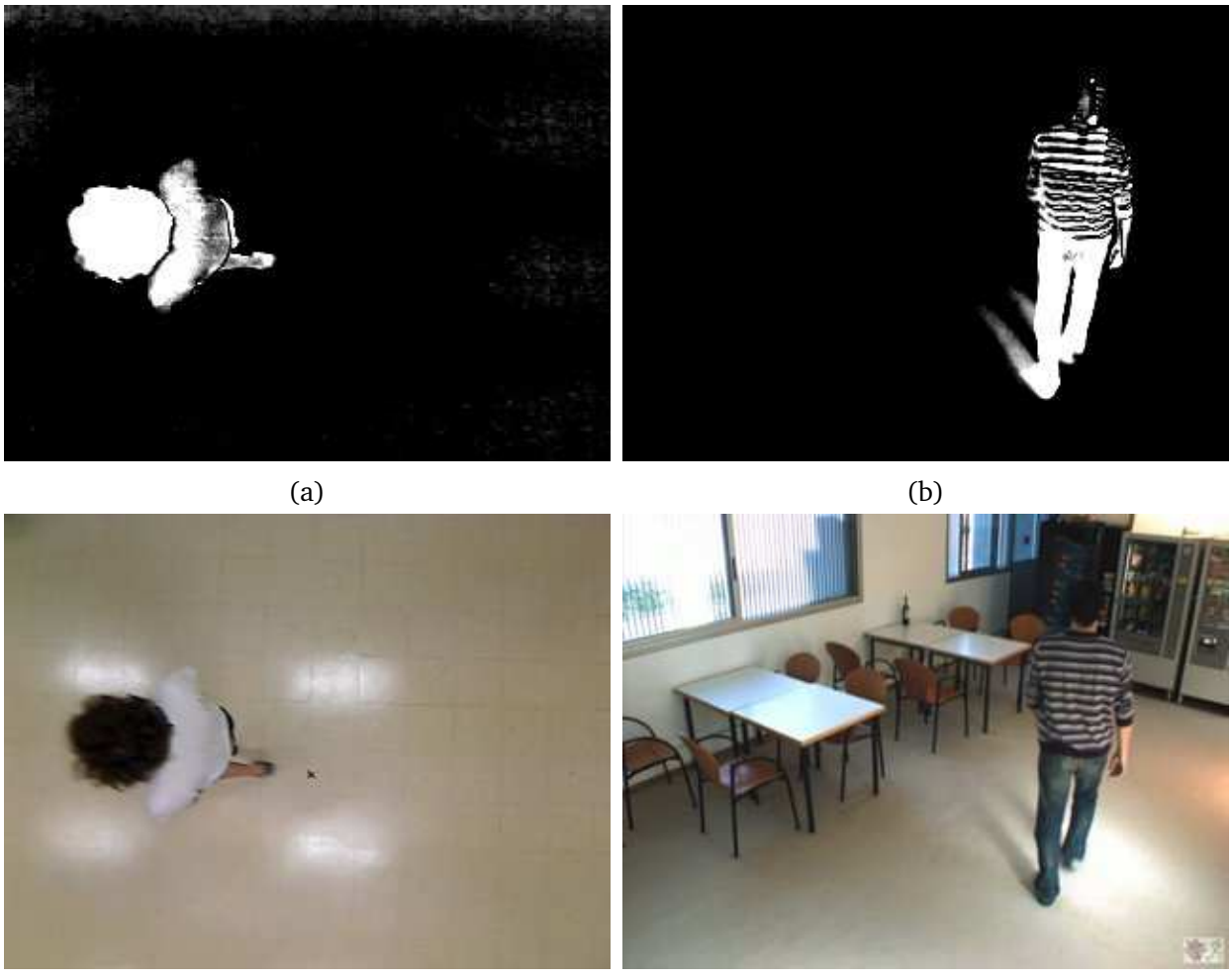


(c) Canale rosso

(d) Canale blu

Figura 3.3: Immagini della deviazione standard calcolate da una ripresa statica di 50 *frame*. Sono state equalizzate dello stesso fattore per rendere più visibili le loro caratteristiche

Per le immagini mostrate sono stati utilizzati i valori di soglia $m = 7$ e $M = 10$, scelti sperimentalmente per ottenere un compromesso tra una buona efficienza nel individuare il soggetto nell'immagine e robustezza in differenti condizioni di luce. Come è possibile notare nelle *confiance image* 3.4a, 3.4b, la sagoma individuata non è uniforme, ma ha zone scure o nere nei casi in cui delle regioni della figura in primo piano non si discostano abbastanza dallo sfondo. Questo accade per pixel con bassa saturazione, in cui quindi l'informazione sul colore non è discriminante per distinguere sfondo e primo piano, e in zone critiche, in cui l'immagine di deviazione standard dello sfondo ha valori elevati. A questo proposito il valore di soglia m è stato scelto appositamente ridotto, tale soluzione ha l'inconveniente di impedire che alcune componenti rumorose dell'immagine vengano eliminate (come nel caso della figura 3.4a, dovute alle fluttuazioni del video). Nel paragrafo seguente si illustreranno alcune soluzioni a questo problema.



(a) (b)

(c) (d)

Figura 3.4: *Confiance image* per due diverse sequenze video

3.5.2 Estrazione *silhouette* della figura

Dopo aver ottenuto la *confiance image* è necessario applicare alcune trasformazioni allo scopo di ridurre componenti rumorose ed ottenere un immagine binaria dove la *silhouette* dell'oggetto in primo piano è quanto più definita possibile, in modo che gli algoritmi di *contour finder* si possano applicare correttamente. Verranno illustrate le differenti tecniche sperimentate e la combinazione di esse che si è rivelata ottimale.

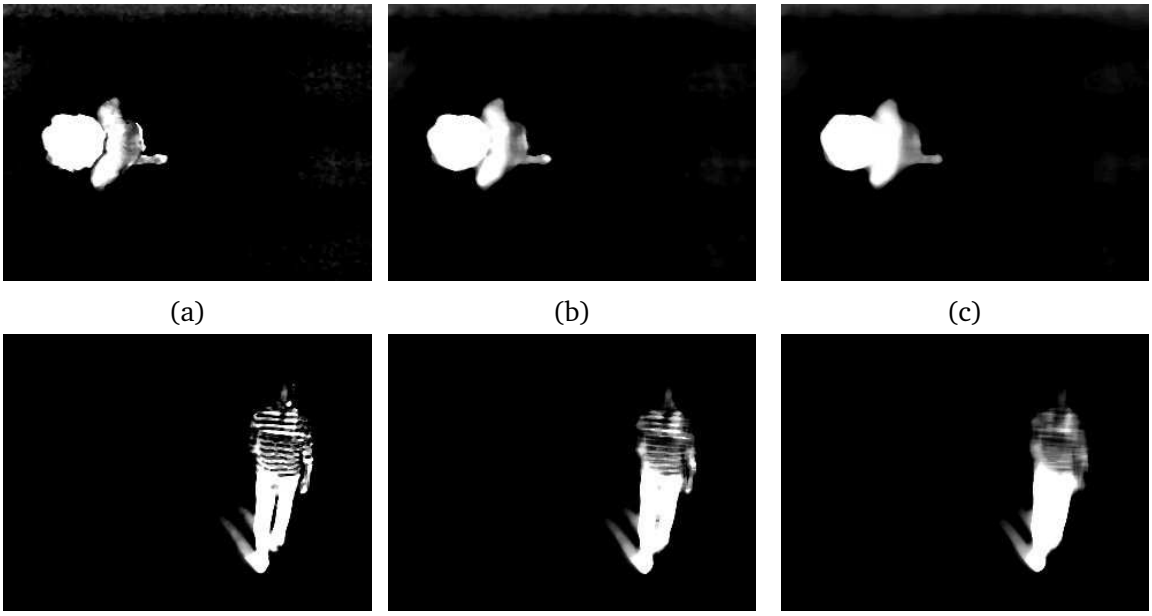
Filtro mediano [51]

Il filtro mediano è un filtro non-lineare tipicamente utilizzato nell'ambito del *signal processing* per evidenziare i pattern significativi del segnale, trascurando componenti rumorose e dettagli. La sua azione consiste nel sostituire al valore di un campione la mediana [7] tra tutti quelli presenti in una regione nelle vicinanze del campione in esame. Questa operazione si realizza attraverso una convoluzione non lineare tra l'immagine in questione e una finestra o *kernel*, tipicamente di forma quadrata. Questo tipo di filtraggio appartiene alla categoria cosiddetta di *smoothing*, ma al contrario del filtraggio di media, in cui invece al valore di un campione è sostituito il valore medio tra tutti i campioni della finestra, il filtraggio mediano riesce a preservare le informazioni sui bordi dell'immagine, condizione imprescindibile perché i successivi algoritmi di *contour finder* possano individuare correttamente i bordi della figura di interesse. Nella figura 3.5 sono riportati le immagini delle sequenze video di test mostrate in precedenza, a cui è stato applicato un filtraggio mediano utilizzando finestre di tre differenti dimensioni. Come si può notare dalle figure, la sagoma individuata dall'algoritmo di *confiance* viene resa più omogenea dal filtro mediano, ma in presenza di rumore dovuto alle fluttuazioni dell'immagine video il filtraggio riesce a rimuovere solamente componenti impulsive (tanto meglio quanto maggiore è la dimensione della finestra del filtro) mentre come previsto è impotente di fronte a estese regioni rumorose (3.5a, 3.5b, 3.5c). La libreria OpenCV fornisce una funzione per applicare una serie di filtraggi a un immagine, tra cui il filtro mediano.

Operazioni morfologiche [52]

Un altra tecnica diffusa per l'elaborazione di immagini digitali consiste nell'operare determinate trasformazioni morfologiche all'immagine di interesse. Tali trasformazioni sono innanzitutto definite per immagini binarie, ovvero in cui solo due valori di pixel sono permessi (tipicamente il valore 1 corrisponde al nero e 0 corrisponde al bianco). Le operazioni base sono la *dilatazione* e l'*erosione*, che in maniera intuitiva corrispondono rispettivamente a una espansione e a un assottigliamento della regione nera (o, in maniera duale, a un assottigliamento e un espansione della regione bianca). Da una combinazione delle due si hanno poi le operazioni di *Apertura* e *Chiusura*. Tralasciando una più approfondita trattazione matematica, tali operazioni si possono implementare nella maniera seguente.

[7] la mediana è quel numero di una serie di valori per cui una metà dei valori della serie è superiore o uguale e una metà è inferiore o uguale a esso



(a) (b) (c)

(d) (e) (f)

Figura 3.5: *Confiance image* dopo aver applicato un filtro mediano di dimensione 3X3 (figure 3.5a e 3.5d), 7X7 (figure 3.5b e 3.5e) e 11X11 (figure 3.5c e 3.5f)

Dilatazione. Si inizi da un immagine destinazione in cui tutti i pixel sono a 0 (bianchi). Si proceda con una sequenza di operazioni *OR* tra essa e l'immagine di partenza, ogni volta con una specifica traslazione, determinata da un oggetto chiamato *elemento strutturale (Sel)*. L'immagine ottenuta è quella risultante dalla dilatazione dell'immagine sorgente. Il *Sel* può essere considerato come una serie di vettori bidimensionali, che specificano le traslazioni rispetto a un *origine* (3.1). Un esempio di dilatazione, per un *Sel* di forma rotonda è riportato in figura 3.6a)

Erosione. In questo caso l'immagine di destinazione è inizialmente costituita da pixel neri (valore 1), alla quale vengono applicate una serie di operazioni logiche *AND* con l'immagine di sorgente, sempre traslandola rispetto a uno specifico elemento strutturale *Sel* (3.6b).

Apertura. Se si applica un operazione di erosione seguita da una dilatazione si ottiene un Apertura (3.6c).

Chiusura. Infine l'operazione di chiusura è ottenuta da una dilatazione seguita da da una erosione (3.6d).

Tali operazioni morfologiche possono essere generalizzate per immagini in scala di grigi. Dove le operazioni binarie *OR* e *AND* sono sostituite rispettivamente da operazioni di massimo e minimo. Nell'ambito del *image processing* le operazioni di chiusura e apertura (le quali

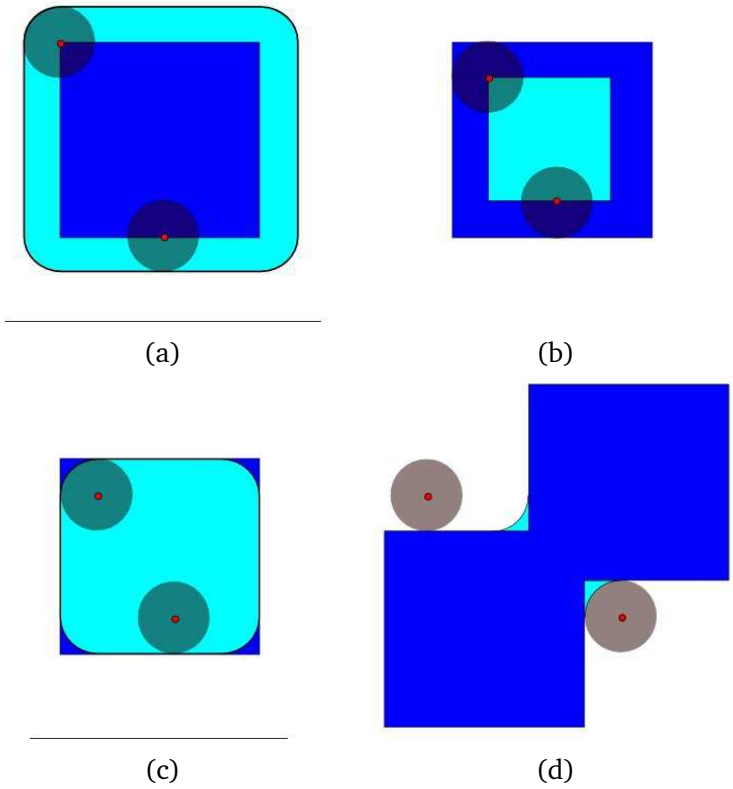


Figura 3.6: Dilatazione (3.6a), erosione (3.6a), apertura (3.6c) e chiusura (3.6d) della figura blu scuro utilizzando un elemento *Sel* a forma di disco. La figura risultante è quella in blu chiaro [53]

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

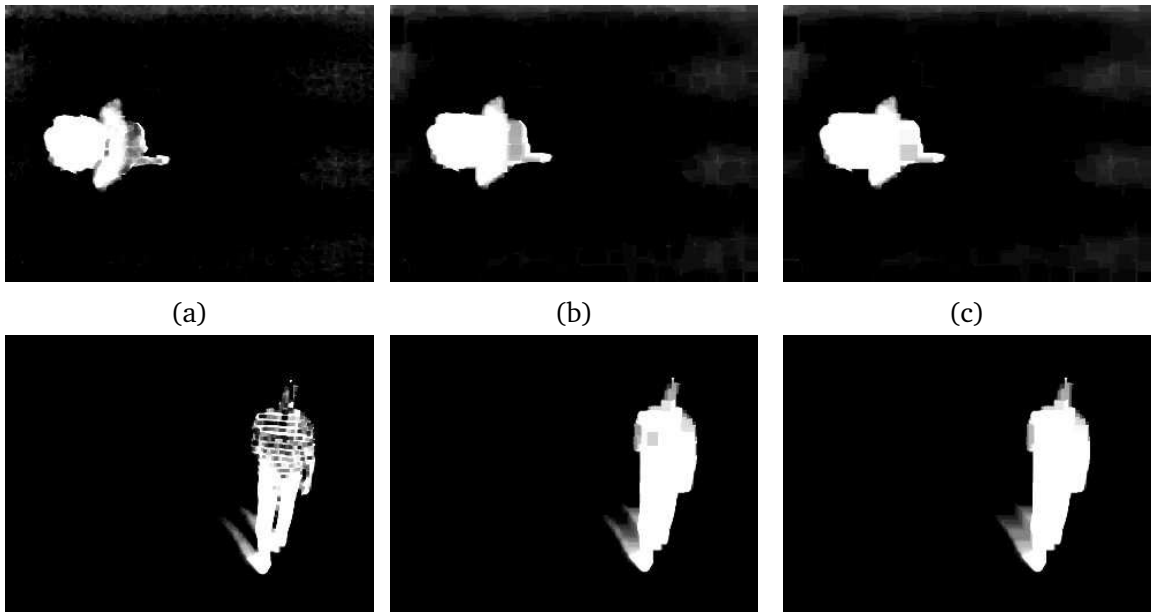
Tabella 3.1: *Sel* quadrato di dimensione 3 rappresentato sotto forma matriciale

conservano le informazioni sui bordi della figura, al contrario della dilatazione e dell'erosione) sono utilizzate per la riduzione di rumorosità morfologiche. In particolare l'apertura rimuove piccoli oggetti bianchi da una figura scura e la chiusura rimuove piccoli oggetti neri da una figura chiara.
Tornando al caso di interesse, come notato in precedenza (figure 3.4a e 3.4b), la sagoma bianca individuata dall'algoritmo di *confiance* non è uniforme, ma presenta delle regioni scure nei punti in cui l'algoritmo non è riuscito a distinguere la figura dallo sfondo. Per questo motivo è stata sperimentata l'applicazione di operazioni di chiusura, utilizzando un *Sel* standard di dimensioni $3X3$, come quello mostrato in 3.1. Le *confiance image* dopo aver applicato l'operazione di chiusura per diverse iterazioni sono mostrate in figura 3.7. Come è possibile vedere, con 3 o 5 iterazioni la sagoma della figura risulta completamente individuata dalla regione bianca, con risultati migliori di quanto ottenuto applicando un filtraggio mediano. D'altra parte però anche le regioni rumorose sono state espanse, in particolare il rumore impulsivo dell'immagine 3.4a è diventato tanto più esteso quanto maggiori sono le iterazioni. Una tecnica comunemente utilizzata ([54], [55], [56]) per la rimozione di rumore impulsivo consiste nell'applicare prima una apertura per eliminare elementi rumorosi chiari dalla figura, poi una chiusura, per ricostituire la parte chiara di interesse. Ma in casi limite, con una *silhouette* non perfettamente definita, la rimozione del rumore ottenuta dall'operazione di apertura si ha al costo dell'eliminazione di regioni della figura di interesse, rendendo quindi non percorribile tale strada (figura 3.8).

Algoritmo di soglia

Le tecniche finora illustrate agiscono indiscriminatamente sulle porzioni rumorose e regioni di interesse dell'immagine. In particolare quando il valore di *confiance* non permette di decidere con certezza se una regione faccia parte della figura di interesse o dello sfondo tali tecniche non forniscono nessuno strumento di discriminazione. Un possibile approccio a tale problema consiste nel classificare le regioni incerte in base alla loro vicinanza con pixel in cui il valore di *confiance* è del 100% [48]. Questa funzione di soglia è stata implementata attraverso un operazione di tipo convolutivo non lineare tra l'immagine di *confiance* e una maschera (*kernel*). La dimensione e la forma della maschera identificano un intorno di un pixel dell'immagine. Quindi, per K intorno del pixel di coordinate (x, y), si ha.

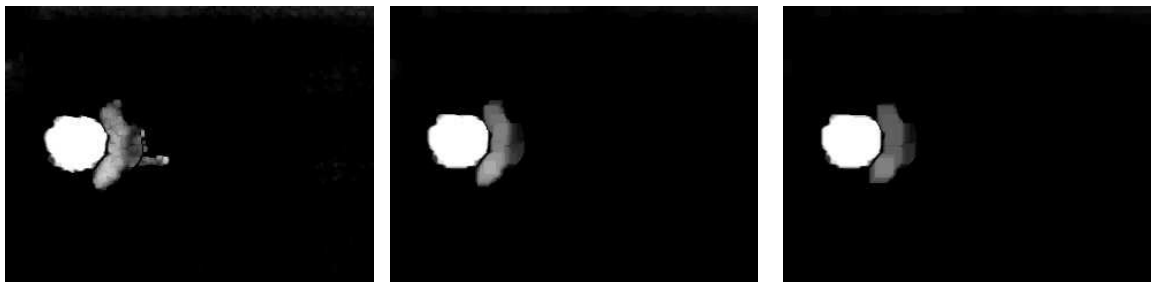
$$dst(x,y) = \begin{cases} 1 & \text{se } src(x,y) \neq 0 \wedge \exists (u,v) : src(u,v) = 1, (u,v) \ni K_{x,y} \\ 0 & \text{altrimenti.} \end{cases}$$



(a) (b) (c)

(d) (e) (f)

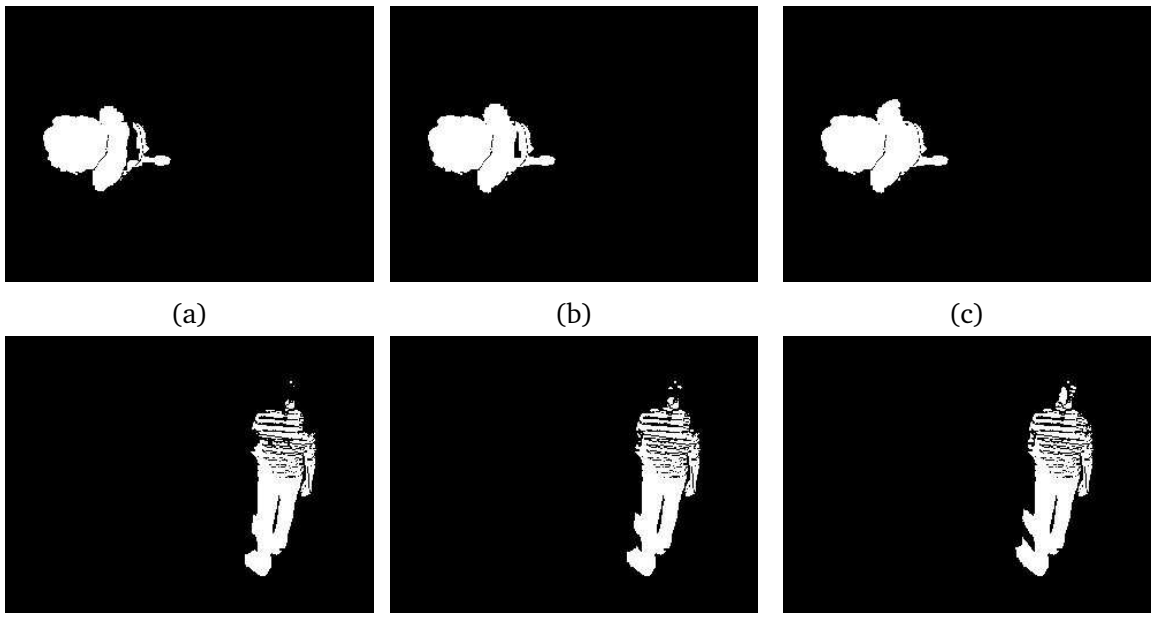
Figura 3.7: *Confiance image* dopo aver applicato l'operazione morfologica di chiusura per 1 iterazione (figure 3.7a e 3.7d), 3 iterazioni (figure 3.7b e 3.7e) e 5 iterazioni (figure 3.7c e 3.7f)



(a) (b) (c)

Figura 3.8: *Confiance image* della figura 3.4a dopo aver applicato l'operazione di chiusura per 1 (3.8a), 3 (3.8b) e 5 (3.8c) iterazioni

Dove $dst(x, y)$ rappresenta un pixel dell'immagine sorgente e $src(x, y)$ un pixel dell'immagine a cui è stata applicata la trasformazione. Un pixel incerto (con valore diverso da zero) non fa parte dello sfondo se in un suo intorno K si ha almeno un pixel con valore di *confiance* del 100%. Nella figura 3.9 sono mostrate le *confiance image* a cui è stata applicato l'algoritmo di soglia utilizzando maschere quadrate di differenti dimensioni. È possibile notare come le regioni rumorose, in cui non era presente nessun punto con valore di *confiance* 100% sono state completamente eliminate. A questo punto è stato effettuata un'operazione di chiusura per eliminare residui neri rimanenti nella figura della sagoma 3.10.
Di seguito sono mostrati *frames* video acquisiti nella Stanza Logo-Motoria in diverse condizioni, insieme alla *confiance image* e all'immagine finale dopo le operazioni di soglia con una maschera $11X11$ e di chiusura per 3 iterazioni (3.11 3.12 3.13 3.14). L'algoritmo funziona bene in tutte le condizioni tranne che per il caso illustrato in figura 3.13. In questo caso, minime differenze di colore tra la maglietta bianca della figura umana e del pavimento, insieme a una condizione di luminosità sfavorevole (luce solare), non rende possibile distinguere regione di primo piano da quella di sfondo.



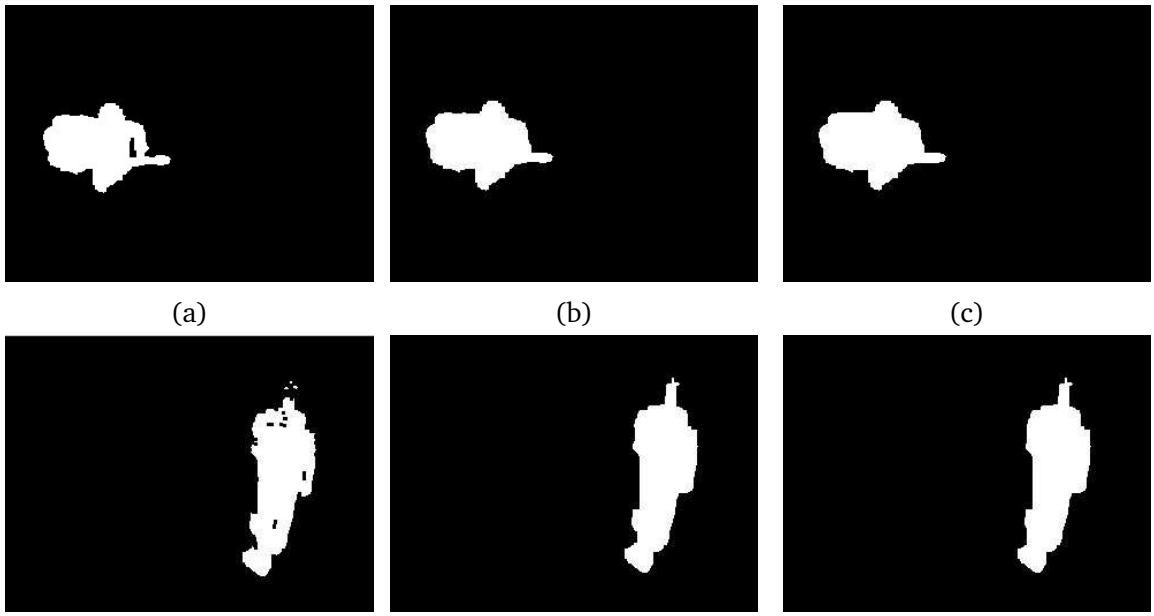
(a) (b) (c)

(d) (e) (f)

Figura 3.9: *Confiance image* dopo aver applicato l'algoritmo di soglia con una maschera di dimensione 7X7 (figure 3.9a e 3.9d), 11X11 (figure 3.9b e 3.9e) e 19X19 (figure 3.9c e 3.9f)

3.5.3 *Blob-tracking*

Gli algoritmi descritti in precedenza producono una immagine binaria dove i soggetti in primo piano sono individuati da regioni bianche, comunemente riferite come *blob*. Il passo successivo consiste nel descrivere ognuno di questi *blob* in base alla sua posizione e forma,



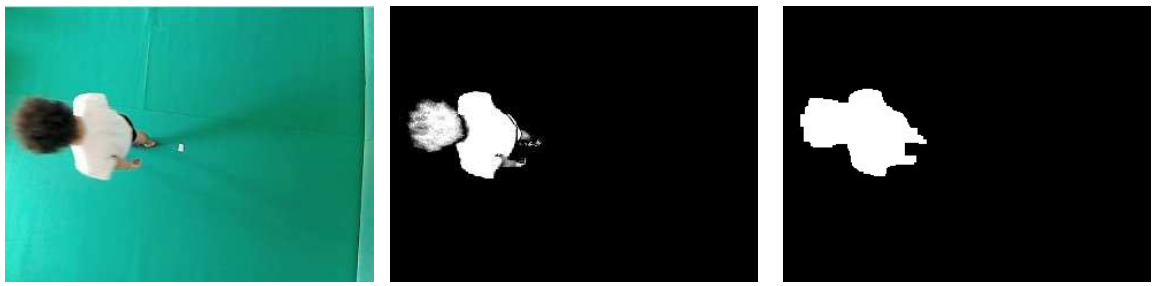
(a) (b) (c)

(d) (e) (f)

Figura 3.10: *Confiance image* a cui prima è stato applicato l'algoritmo di soglia con una maschera 11X11, quindi l'operazione di chiusura per 1 iterazione (figure 3.10a e 3.10d), 3 iterazioni (figure 3.10b e 3.10e) e 5 iterazioni (figure 3.10c e 3.10f)

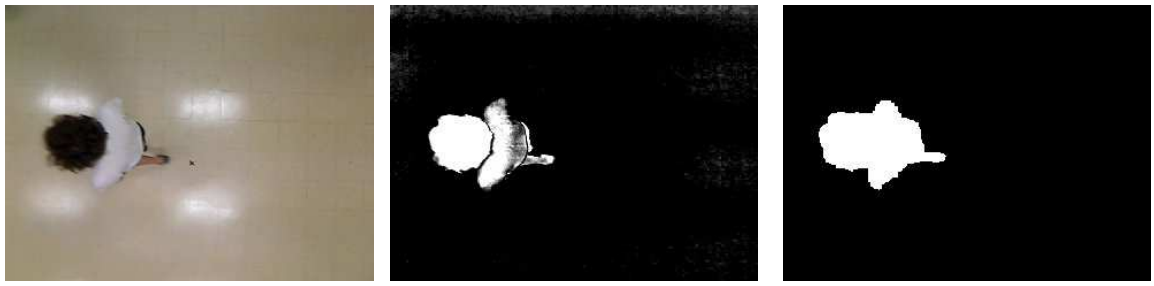
(a) (b) (c)

Figura 3.11: *Frame* (3.11a), *confiance image* (3.11b) e immagine finale (3.11c) di una sequenza video della Stanza Logo-Motoria con l'utilizzo di un tappeto di colore uniforme



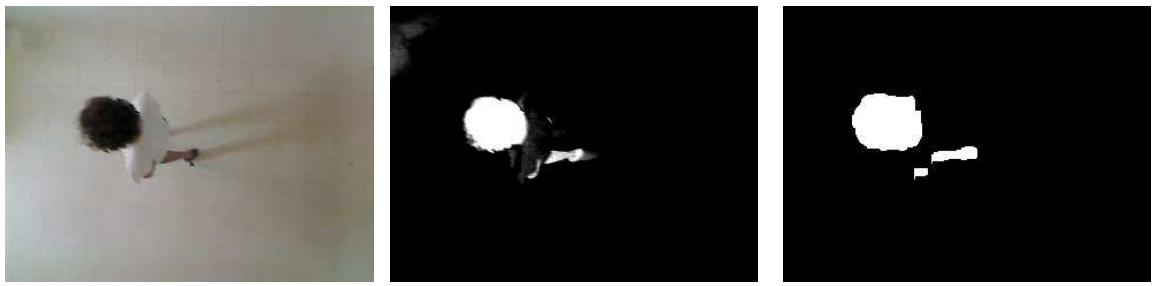
(a) (b) (c)

Figura 3.12: *Frame* (3.12a), *confiance image* (3.12b) e immagine finale (3.12c) di una sequenza video della Stanza Logo-Motoria con l'utilizzo di un tappeto di colore uniforme e in presenza di luce solare



(a) (b) (c)

Figura 3.13: *Frame* (3.13a), *confiance image* (3.13b) e immagine finale (3.13c) di una sequenza video della Stanza Logo-Motoria con senza tappeto



(a) (b) (c)

Figura 3.14: *Frame* (3.14a), *confiance image* (3.14b) e immagine finale (3.14c) di una sequenza video della Stanza Logo-Motoria senza l'utilizzo del tappeto e in presenza di luce solare

in modo da poter tracciare i suoi movimenti. A questo scopo l'*addon* di OpenCV in Openframeworks mette a disposizione le classi *ofxCvContourFinder* e *ofxCvBlob*. In particolare il metodo *findContours* in *ofxCvContourFinder*, data in ingresso un immagine binaria, utilizza alcune funzioni di OpenCV per individuare i *blob* e memorizzarne le informazioni in strutture dati di tipo *ofxCvBlob*.

Rilevazione di contorni in OpenCV [47]

OpenCV ha una specifica funzione per l'estrazione di contorni da un immagine binaria, *cvFindContours()*. Tale funzione fa uso di specifiche strutture di OpenCV per gestire l'accesso in memoria quando si ha a che fare con oggetti dinamici (*CvMemStorage*) e per creare sequenze di oggetti (*CvSeq*). Il loro obiettivo è ottenere prestazioni ottimizzate per operazioni di *Computer Vision*. I contorni sono rappresentati in OpenCV da sequenze (*CvSeq*) in cui ciascun elemento codifica l'informazione della posizione di un punto della curva del contorno. La definizione della funzione è la seguente.

```
int cvFindContours(
IplImage* img,
CvMemStorage* storage,
CvSeq** firstContour,
int headerSize = sizeof(CvContour),
CvContourRetrievalMode mode = CV_RETR_LIST,
CvChainApproxMethod method = CV_CHAIN_APPROX_SIMPLE
);
```

Il primo argomento è l'immagine di ingresso, questa deve avere un solo canale e verrà interpretata come binaria dalla funzione (tutti i pixel di valore non nullo sono considerati equivalenti l'uno all'altro). Questa immagine viene modificata dalla funzione durante la sua esecuzione, è quindi importante creare una copia dell'immagine sorgente se si intende utilizzarla successivamente. L'argomento successivo indica dove *cvFindContours* può trovare lo spazio in memoria in cui scrivere l'informazione del contorno. Questa area dati è rappresentata da un oggetto della classe *CvMemStorage*, precedentemente allocato. L'argomento successivo è *firstContour* che è un puntatore a *CvSeq* che la funzione allocherà al suo interno. È alla posizione in memoria **firstContour* che si troverà un puntatore all'informazione sul contorno. Infine la funzione restituisce in uscita il numero totale di contorni trovati. I restanti argomenti permettono di selezionare alcune opzioni riguardo al metodo utilizzato per l'estrazione dei contorni e il modo in cui vengono rappresentati tali contorni.

Le classi *ofxCvContourFinder* e *ofxCvBlob*

ofxCvBlob è una semplice classe che raggruppa i principali attributi con cui identificare forma e posizione di un *blob*: l'area, il baricentro, la lunghezza del contorno, una *bounding box* (ovvero il mimino rettangolo che contiene la superficie del blob), un vettore di punti che contiene l'informazione del contorno e una variabile booleana a segnalare se il blob ha buchi

o meno. Oltre a questi attributi la classe ha anche un metodo *draw()* per disegnare su schermo il contorno e la *bounding box* del blob. Le informazioni geometriche sono rappresentate dai tipi basici con cui Openframeworks gestisce gli oggetti grafici. In questo caso *ofRectangle* per la *bounding box*, e *ofPoint* per il baricentro e i punti del che costituiscono il contorno.
Gli elementi principali della classe *ofxCvContourFinder* sono costituiti da un vettore di oggetti di tipo *ofxCvBlob*, in cui vengono salvate le informazioni di tutti i *blob* individuati nell'immagine, e il metodo *findContours()*, la cui definizione è la seguente.

```
findContours( ofxCvGrayscaleImage& input,
              int minArea, int maxArea,
              int nConsidered, bool bFindHoles,
              bool bUseApproximation = true);
```

In cui *input* è l'immagine binaria in cui individuare i *blob* ed estrarre i contorni. *minArea* e *maxArea* permettono di specificare dei limiti di area per i *blob* da rilevare mentre *nConsidered* specifica il massimo numero di blob da rilevare. In *bFindHoles* viene specificato se ricercare anche i contorni dei buchi nei blob. Infine *bUseApproximation* è un *flag* per impostare il modo in cui viene immagazzinata l'informazione sul contorno: *true* se si vuole che la curva sia rappresentata in maniera semplificata (segmenti orizzontali, verticali e diagonali sono memorizzati utilizzando solo i punti finali) o meno.
Il comportamento della funzione si svolge nei passi seguenti.

1. Creazione di una copia dell'immagine di ingresso, se necessario (nei casi in cui il metodo è usato per la prima volta o le dimensioni dell'immagine di ingresso sono cambiate). Come spiegato in precedenza il metodo *cvFindContours* modifica l'immagine da cui si intende estrarre i contorni, in questo caso quindi l'immagine in ingresso non sarà modificata dalla funzione.

2. Allocamento degli oggetti di tipo *cvMemStorage* (attributi privati della classe *ofxCvContourFinder*) e dichiarazione di un oggetto del tipo *cvSeq*. Questi sono poi utilizzati dal metodo *cvFindContours* per immagazzinare le informazioni sui contorni.

3. Definizione dei parametri per *cvFindContours*, in accordo con il valore dei flag *bFindHoles* e *bUseApproximation* specificati alla chiamata della funzione.

4. Chiamata della funzione *cvFindContours*.

5. Se le dimensioni del blob rilevato rientrano nei limiti imposti da *minArea* e *maxArea*, sposto i suoi dati relativi al contorno (memorizzati in un unica lista *cvSeq*) in un singolo elemento dell'array *cvSeqBlobs*. Questo array è composto da elementi di tipo *cvSeq**, ognuno dei quali memorizza però un solo contorno.

6. L'array *cvSeqBlobs* viene ordinato in maniera decrescente secondo la dimensione (area) dei blob.

7. A partire dall'array *cvSeqBlobs*, i dati relativi ai *blob* (l'area, il baricentro, la lunghezza del contorno, una *bounding box*) vengono passati alle strutture *ofxCvBlob*.

8. Viene liberata la memoria allocata per gli oggetti *cvMemStorage* con la funzione *cvReleaseMemStorage()*.

Dopo l'esecuzione di tale metodo le informazioni su ogni *blob* rilevato sono quindi accessibili attraverso il vettore di *ofxCvBlob* della classe *ofxCvContourFinder* e potranno essere quindi utilizzate per successive computazioni.

Capitolo 4

L'applicazione

Nella figura 4.1 è riportato lo schema generale di tutte le classi di cui si compone il software. Seguendo una pratica diffusa nei programmi Openframeworks, la parte centrale dell'applicazione è costituita da un oggetto di una classe chiamata *testApp*. Come spiegato nella sezione 3.3, in questa classe vengono implementate le funzioni *setup()*, *update()* e *draw()* in cui si definiscono rispettivamente l'inizializzazione, l'aggiornamento e la parte relativa alla grafica del programma. Nel caso del software della Stanza Logo-Motoria, si sono poi definite due diverse classi, ognuna responsabile delle due fondamentali attività del programma e ognuna con una propria GUI.

***TrackingEngine*.** Qui viene scelto l'algoritmo di *Motion Tracking* utilizzato e vengono restituite le informazioni sui *blob* in modo che queste possano essere utilizzate dall'oggetto della classe *AppManager*. Inoltre un interfaccia grafica è disponibile (*vista setup*) per permettere eventuali manipolazioni dei parametri utilizzati dall'algoritmo di *motion-capture* e per monitorare il funzionamento di quest'ultimo.

***AppManager*.** Questa classe gestisce il funzionamento delle applicazioni utilizzabili all'interno della Stanza Logo-Motoria (per ora limitate alla sola *Resonant Memory*). Riceve le informazioni sui *blob* individuati dall'oggetto del tipo *TrackingEngine* e le restituisce all'applicazione attiva. Inoltre implementa un interfaccia grafica con cui l'utente "gestore" può scegliere tra le varie applicazioni disponibili (*vista applicazioni*).

Le definizioni delle funzioni *setup()*, *update()* e *draw()* all'interno di *testApp* si limitano ad invocare le omologhe funzioni delle classi *TrackingEngine* e *AppManager*, a fare in modo che queste possano comunicare tra loro. Inoltre nella classe *testApp* si implementa una macchina a due stati attraverso una variabile di tipo *enum*. Essa può assumere due valori: *applicationState* e *setupState* a seconda che si voglia mostrare l'interfaccia di setup o dell'applicazione.

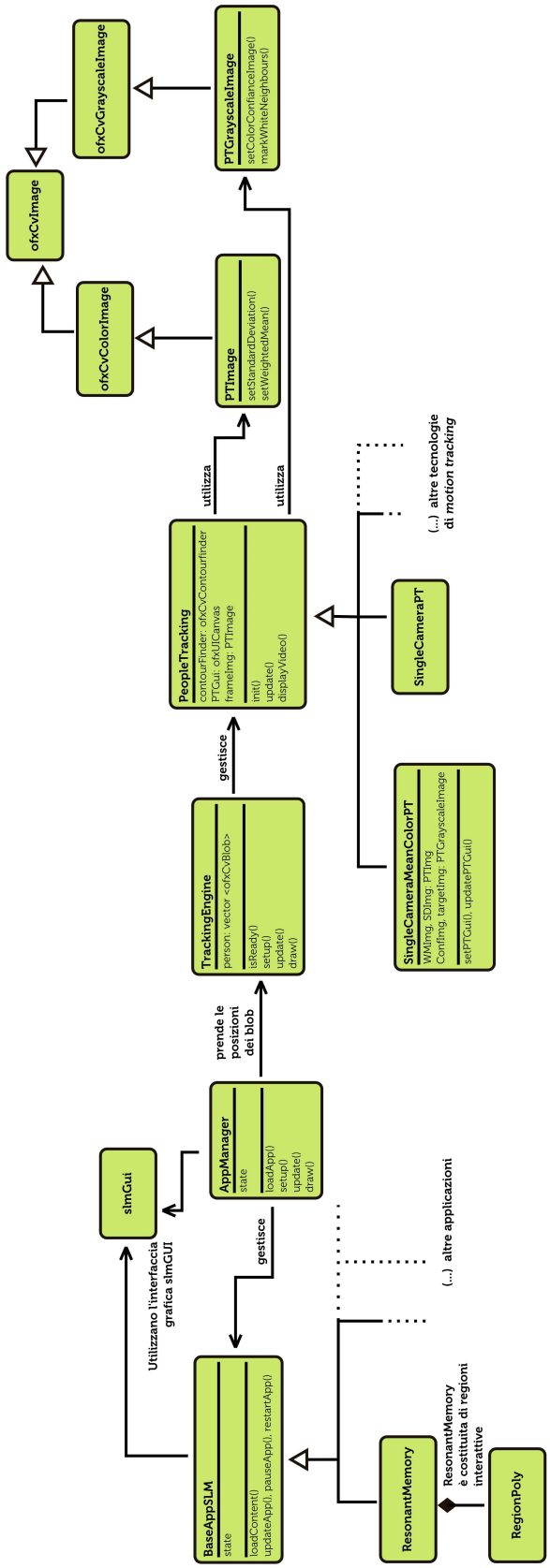


Figura 4.1: Diagramma UML delle classi del software della Stanza Logo-Motoria. Per ogni classe sono riportati gli attributi e i metodi principali.

setup()

currentState viene inizializzata al valore di *applicationState*. All'esecuzione il programma mostrerà l'interfaccia grafica dell'applicazione. Successivamente vengono invocati i metodi *setup()* delle classi *TrackingEngine* e *AppManager*, in cui si hanno inizializzazioni di variabili e allocamenti in memoria.

update()

Qui i due oggetti delle classi *TrackingEngine* e *AppManager* vengono continuamente aggiornati e nel caso ci si trovi nello stato *applicationState*, la GUI dell'applicazione è stata implementata in modo da richiedere specificatamente il suo aggiornamento. Dopodiché si interrogano le funzioni preposte delle classi *TrackingEngine* e *AppManager* per vedere se è stato richiesto un passaggio tra la *vista applicazione* e la *vista setup* e infine, se la funzione *isReady()* della classe *TrackingEngine* restituisce un valore affermativo, la posizione del baricentro dei *blob* viene trasmessa all'oggetto *AppManager*, quindi all'applicazione della Stanza Logo-Motoria in esecuzione.

draw()

In base allo stato corrente, in questa funzione si invocano i metodi *draw()* delle due classi *TrackingEngine* e *AppManager*. Nel caso ci si trovi nella *vista setup*, delle specifiche funzioni impostano la posizione e disegnano la "mappa" dell'applicazione della Stanza Logo-Motoria in esecuzione. Per ora si tenga conto che tale procedura ha solo una finalità grafica, si rimanda comunque alle sezioni successive per una descrizione più dettagliata.

4.1 *TrackingEngine*

```
class TrackingEngine
{
private:
PeopleTracking* pt;
public:
void setup(int trackingMethod);
  void update();
  void draw();
vector<ofxCvBlob> person;
bool isReady()
  void thresholds(int tep00, int tep01, int tep02,
                  int tep03, int tep04);
  bool quitSetup();
}
```

pt

pt è un puntatore del tipo *PeopleTracking*. Nella funzione *setup()* a esso viene istanziato un oggetto di una classe derivata di *PeopleTracking*, ognuna delle quali realizza un diverso algoritmo di *motion capture*.

setup(int trackingMethod)*, *update()* e *draw()

Queste tre funzioni vengono chiamate rispettivamente nelle tre omonime *setup()*, *update()* e *draw()* della classe *testApp*.
Nella funzione *setup(int trackingMethod)* al puntatore *pt* vengono assegnati dinamicamente oggetti delle classi derivate da *PeopleTracking* ognuna identificata dal valore di trackingMethod. Successivamente viene chiamata la funzione *init()* sul puntatore *pt*, nella quale si inizializza la classe di *motion capture* (allocamento di immagini, caricamento dei parametri iniziali...).
Nel metodo *update()* il vettore di *ofxCvBlob* viene passato dall'oggetto della classe *PeopleTracking* viene passato al vettore dello stesso tipo *person*, normalizzato secondo le dimensioni dell'immagine video.
Infine nel metodo *draw()* viene invocato *displayVideo()* della classe *PeopleTracking*. Relativamente al metodo di *motion capture* utilizzato, viene mostrato un diverso menu e dei video su cui monitorare il corretto funzionamento dell'algoritmo.

person

Il vettore di *ofxCvBlob* che durante la fase di *update()* di *testApp* viene passato tramite un apposito metodo ad *AppManager*.

isReady()

Viene chiamata durante ogni ciclo di *update()* di *testApp*. Restituisce un valore booleano che, se affermativo, significa che l'oggetto *PeopleTracking* è pronto per passare le informazioni sui *blob* (non ci sono operazioni di *background learning* o simili in corso).

threshold()

Questa funzione può venire usata per modificare parametri dell'algoritmo di *motion capture* da parte di altre sezioni del programma durante la sua esecuzione. La versione finale del software prevede che modificazioni a tali parametri possano solo essere fatti dall'utente attraverso l'interfaccia grafica della vista di setup.

quitSetup()

È una funzione con un valore booleano come uscita. Viene chiamata durante ogni ciclo di *update()* di *testApp*, se il suo valore è vero si passa dalla vista di setup a quella dell'applicazione. Il suo utilizzo è legato a un apposito bottone nella GUI di setup.

4.1.1 La classe astratta *PeopleTracking*

Durante la realizzazione del software, sono state sperimentate diverse metodologie di *Motion Tracking* prima di decidere per l'adozione delle tecniche descritte nella sezione 3.5. Per questa ragione e per permettere nel corso di eventuali futuri sviluppi, di utilizzare altri algoritmi di *motion capture* si è utilizzata la classe astratta *PeopleTracking*, i cui metodi virtuali *init()*, *update()* e *displayVideo()* vengono poi implementati nelle classi derivate secondo l'algoritmo di *motion capture* che esse utilizzano (vedi figura 4.2).
Le classi derivate scrivono le informazioni riguardanti i *blob* individuati in un vettore di *ofxCvBlob* dichiarato nella classe *PeopleTracking*, eventualmente utilizzando il metodo *findContours* della classe *ofxCvContourFinder*, anch'essa istanziata nella classe *PeopleTracking*. Inoltre in questa classe è definita l'immagine di tipo *PTImage* sulla quale vengono passati i *frame* del flusso video, che tramite la modifica di una costante booleana posta in una direttiva *#define*, possono provenire da un file video presente nell'hard-disk (per effettuare test sull'algoritmo) o da una camera collegata al computer (durante l'utilizzo della Stanza Logo-Motoria). A questo proposito si hanno le speciali funzioni non virtuali *updateSource() updateFrameImg() setupPeopleTracking()*. Queste si comportano in modo differente a seconda della sorgente video e vengono chiamate dalle classi derivate per interagire con *PeopleTracking*.
La versione finale del software permette di compilare il codice utilizzando come classe che implementa l'algoritmo di tracking *SingleCameraPT* o *SingleCameraMeanColorPT*. La prima classe utilizza una tecnica di *background subtraction* statico (un solo frame dello sfondo è acquisito) e sottrae in valore assoluto tale immagine di sfondo ai *frame* successivi. La classe *SingleCameraMeanColorPT* realizza invece il metodo dell'*averaging background* con algoritmo di soglia, illustrato nel capitolo precedente.

4.1.2 *SingleCameraMeanColorPT*

SingleCameraMeanColorPT è la classe che realizza l'algoritmo di *motion capture* utilizzato nella versione finale del software, illustrato nella sezione 3.5. Di seguito ne verranno illustrati i principali metodi e attributi.

WMImg, SDImg

Oggetti della classe di immagine *PTImage*. Tale tipo, che verrà analizzato in dettaglio i seguito, fornisce alcune funzioni speciali per l'elaborazione di immagini. *WMImg* e *SDImg* rappresentano rispettivamente la media pesata (*WM* sono le iniziali di *Weighted Mean*) e

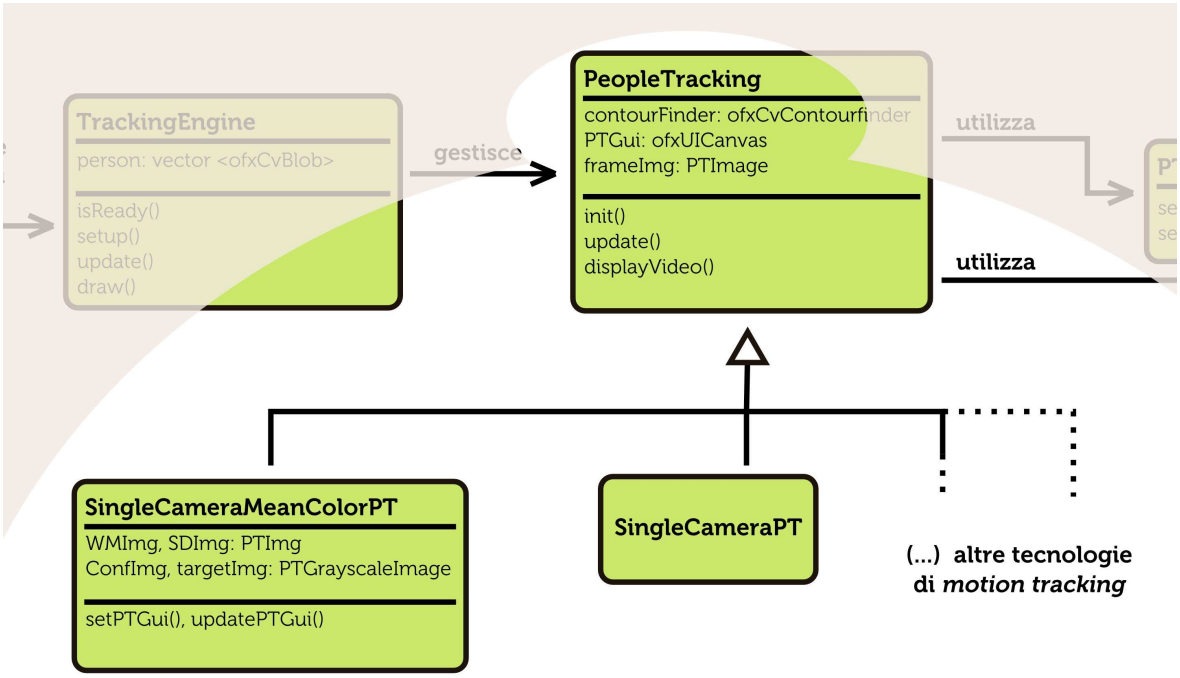


Figura 4.2: La classe astratta *PeopleTracking* e le sue classi derivate.

deviazione standard (*SD* sta per *Standard Deviation*) e il loro valore viene calcolato su una serie di *frame* dello sfondo, utilizzando funzioni appositamente create in *PTImage*.

ConfImg, targetImg

Oggetti della classe di immagine *PTGrayscaleImage*, simile alla classe *PTImage* ma per immagini di un solo canale, anche questa classe è stata creata allo scopo di fornire speciali funzioni per l'elaborazione di immagini. Dopo aver modellato lo sfondo, viene chiamata la funzione *setColorConfianceImage* sull'immagine *ConfImg*. Questa utilizza le immagini *WMImg* e *SDImg* per generare la *confiance image*. Viene successivamente creata una copia di *ConfImg* in *targetImg*, a cui vengono poi applicate funzioni che realizzano filtraggi e soglie per ottenere l'immagine binaria da cui identificare i *blob*.

init()

In questo metodo viene dapprima chiamata la speciale funzione di setup *setupPeopleTracking()* della classe base *PeopleTracking* in cui viene definito l'oggetto appropriato per la gestione del video (a seconda che si tratti di un file nell'hard disk o del flusso video da una camera). Di seguito vengono allocate le immagini *ConfImg*, *targetImg*, *WMImg*, *SDImg* relativamente alla risoluzione della camera. Si inizializzano le variabili utilizzate per il conteggio dei *frame* durante la modellizzazione dello sfondo e infine si inizializza la GUI e si caricano i parametri dell'algoritmo scelti dall'utente.

update()

In primo viene memorizzato il valore binario di uscita dal metodo *updateSource()*, che ha valore positivo quando l'oggetto video riceve un nuovo frame. Quando ciò avviene si hanno due diversi comportamenti, a seconda che ci si trovi nella fase di modellizzazione dello sfondo o meno. Nel primo caso di volta in volta si chiamano le funzioni *setWeightedMean()* e *setStandardDeviation()* rispettivamente sulle immagini *WMImg* e *SDImg*, che modificano via via il loro valore nel corso dell'acquisizione dello sfondo, per il numero di *frame* prestabilito. Dopo questa fase, prima di proseguire con il calcolo della *confiance image*, un operatore di *clipping* inferiore viene applicato all'immagine *SDImg* per evitare comportamenti erronei dell'algoritmo successivo dovuto ai valori troppo bassi dell'immagini (vedi sezione 3.5). Successivamente si calcola l'immagine *ConfImg*, fornendo le immagini di deviazione standard e media e i parametri di *threshold* alla funzione *setColorConfianceImage*, e si applicano i filtraggi ed elaborazioni alla copia di *ConfImg*, *targetImg*. Infine l'immagine *targetImg* viene passata alla funzione *findContours*, che ne estrae i *blob*, che vengono successivamente scritti nel vettore *person*.

displayVideo()

In *displayVideo()* vengono visualizzate le immagini *WMImg*, *SDImg*, *ConfImg*, *targetImg*. Attraverso il metodo *draw()* della classe *contourFinder*, all'immagine *targetImg* sono sovrapposti il contorno, la *bounding box* e il baricentro dei *blob* individuati.

setPTGui()* e *updatePTGui()

Per l'interfaccia utente della vista di setup è stato utilizzato l'*addon ofxUI*, il cui funzionamento verrà illustrato in seguito. Queste due funzioni sono previste dall'utilizzo dell'*addon* rispettivamente per: impostare la posizione e inizializzare lo stato degli oggetti grafici; associare variabili ad ogni oggetto.

4.1.3 Le classi *PTImage* e *PTGrayscaleImage*

Per poter più agevolmente manipolare le immagini secondo le necessità dell'applicazione, sono state create le classi *PTImage* e *PTGrayscaleImage*. Queste ereditano rispettivamente da *ofxCvColorImage* e *ofxCvGrayscaleImage*, immagini a *8 bit* (i pixel assumono valori da 0 a 255) di tre (*RGB*) e un canale (scala di grigi). Esse a loro volta derivano entrambe dalla classe *ofxCvImage*, ovvero una famiglia di immagini introdotte dall'*addon* di OpenCV in Openframeworks. In questo modo tali immagini possono essere utilizzate con le funzioni di OpenCV e con quelle previste dall'*addon* oltre ad avere a disposizione altre le funzioni appositamente create per l'utilizzo in algoritmi di *motion capture*. Quelle utilizzate nella classe *SingleCameraMeanColorPT* sono descritte di seguito.

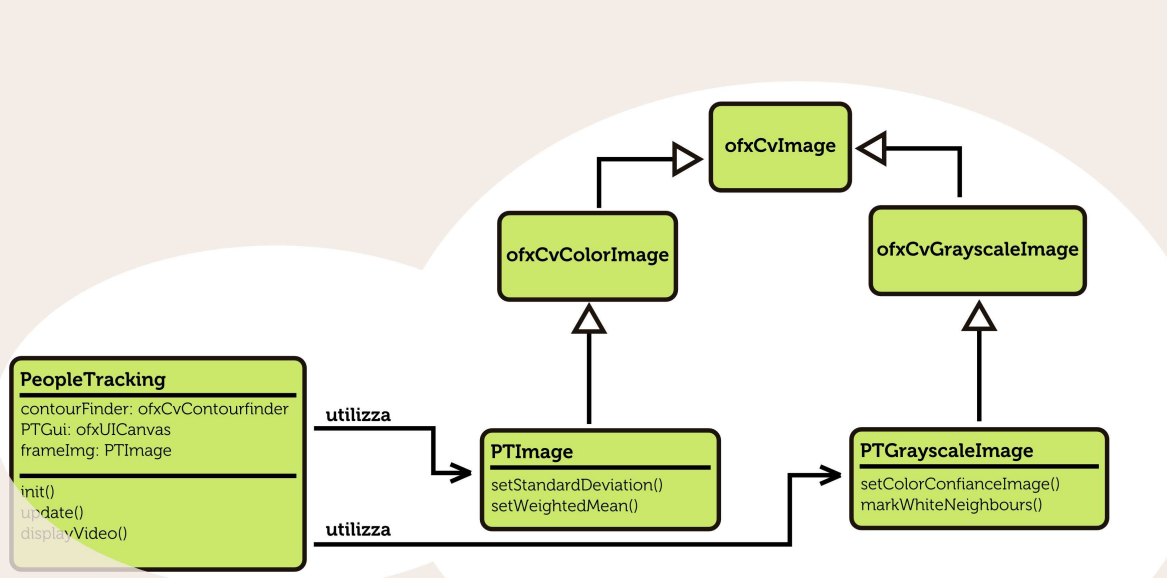


Figura 4.3: Le immagini *PTImage* e *PTGrayscaleImage* ereditano rispettivamente da *ofxCvColorImage* e *ofxCvGrayscaleImage* che a loro volta derivano entrambe dalla classe *ofxCvImage*.

setWeightedMean()

setWeightedMean() utilizza l'immagine del frame corrente (*img*), della media calcolata fino al frame corrente (*oldWMImg*) e un *coefficiente di apprendimento*, per calcolare la nuova media pesata di una serie di immagini. L'espressione utilizzata è la seguente:

$$\mu_t = \alpha x_t + (1-\alpha)\mu_{t-1}$$

in cui μ_t è un pixel dell'immagine della nuova immagine media, x_t il corrispondente pixel dell'immagine del *frame* corrente, μ_{t-1} il corrispondente pixel dell'immagine media precedente e α è il *coefficiente di apprendimento*. Tale calcolo è effettuato per tutti i tre canali di un immagine a colori, la funzione è perciò definita in *PTImage*.

setStandardDeviation()

Questa funzione utilizza l'immagine del *frame* corrente (*img*), l'immagine della media (*WMImg*), della precedente immagine di deviazione standard (*oldSDImg*) e il *coefficiente di apprendimento* (*weight*) per calcolare la deviazione standard di una serie di immagini utilizzando l'espressione

$$\sigma_t^2 = \alpha(x_t - \mu_t)^2 + (1-\alpha)\sigma_{t-1}^2.$$

α è il suddetto *coefficiente di apprendimento*, x_t è il valore di un pixel del *frame* corrente, μ_t il valore del corrispondente pixel della media e σ_{t-1} il valore del corrispondente pixel nell'immagine della deviazione standard. Come la precedente anche in questa funzione il

calcolo viene effettuato indipendentemente per i tre canali di un immagine a colori, anche questa funzione è definita in *PTImage*.

setColorConfianceImage()

Dopo aver costruito il modello statistico dello sfondo, l'algoritmo implementato nella classe *SingleCameraMeanColorPT* prevede la costruzione della *confiance image*. A questo scopo è stata scritta una specifica funzione, definita nella classe *PTGrayscaleImage* (la *confiance image* è composta di un solo canale). Essa riceve il *frame* corrente (*img*), l'immagine della media e della deviazione standard del modello dello sfondo (*WMImg* e *SDImg*) insieme a due parametri di soglia (*minTh* e *maxTh*) e costruisce la *confiance image* applicando, per ogni pixel di ogni canale delle immagini in ingresso, l'espressione vista nella sezione 3.5 del capitolo precedente.

$$C^c = \begin{cases} 1 & \text{se } |x_t - \mu_t| > M\sigma \\ 0 & \text{se } |x_t - \mu_t| < m\sigma \\ \frac{D-m\sigma}{M\sigma-m\sigma} & \text{altrimenti.} \end{cases}$$

Dopodiché l'immagine in scala di grigi (singolo canale) di *confiance* viene creata prendendo per ogni pixel il massimo dei tre canali e scalata in valori da 0 a 255.

markWhiteNeighbours()

Dopo aver calcolato la *confiance image* si applicano alcune ulteriori trasformazioni per migliorare l'aspetto delle figure rilevate. Tra queste l'algoritmo di soglia descritto in 3.5, che è stato implementato per la classe *PTGrayScaleImg*. Come già visto questo realizza la formula seguente.

$$dst(x,y) = \begin{cases} 1 & \text{se } src(x,y) \neq 0 \wedge \exists (u,v) : src(u,v) = 1, (u,v) \ni K_{x,y} \\ 0 & \text{altrimenti.} \end{cases}$$

L'intorno $K_{x,y}$ è definito di forma quadrata e di centro il pixel corrente. Il parametro *size* non rappresenta direttamente il lato del quadrato, ma viene modificato in $size = 2 * size + 1$ per assicurarsi che sia un numero dispari e che quindi il quadrato dell'intorno sia esattamente centrato nel pixel corrente.

Altre funzioni e filtri

Nelle classi *PTImage* e *PTGrayscaleImage* sono state implementate anche altre classi, per ulteriori operazioni di filtraggio e *image processing*. Perlopiù il loro funzionamento si riduce a permettere l'accesso alle immagini dei tipi *PTImage* e *PTGrayscaleImage*, alcune funzioni di OpenCV, allo stesso modo con cui ciò è realizzato dalle classi di Openframeworks *ofxCvColorImage* e *ofxCvGrayscaleImage*. Di seguito le definizioni di tali funzioni.

```
void openMorph(int iter);
void closeMorph(int iter);
void median(float level);
void minClip(int val);
```

openMorph() e *closeMorph()* realizzano le operazioni morfologiche di apertura e chiusura utilizzando il *Sel* quadrato di dimensioni $3X3$ default della funzione corrispondente di OpenCV, e con il parametro *iter* a specificare il numero di volte in cui l'operazione viene ripetuta.
median() è una funzione di *wrapping* del filtro mediano implementato in OpenCV, in cui $level * 2 + 1$ specifica le dimensioni della maschera.
minClip() limita inferiormente il valore dei pixel dell'immagine al valore *val* (da 0 a 255).

4.1.4 GUI della *vista setup* e gestione delle configurazioni

Nel capitolo precedente è stata presentata una procedura ottimale per ottenere un immagine binaria da cui estrarre i blob. Nel software finale tale configurazione è presente come scelta di *default*, ma è comunque lasciata all'utente la possibilità di creare la propria configurazione. Per questo l'applicazione dei vari filtri è strutturata con *statement if* e i loro parametri sono controllati una serie variabili. Per la gestione delle varie funzioni di *image processing* nella *vista setup* del software è stato utilizzato l'addon di Openframeworks ofxUI[1] il quale fornisce una serie di oggetti grafici e la possibilità di associarli a variabili all'interno dell'applicazione. L'utilizzo di questa libreria consiste nella definizione di puntatore a un oggetto della classe *ofxUICanvas* e nella definizione e implementazione di due funzioni.

```
void setPTGui()
void updatePTGui(ofxUIEventArgs &e)
```

In *setPTGui()* si istanzia l'oggetto *ofxUICanvas*, che rappresenta una schermata della GUI, e in esso vengono posizionati i vari oggetti grafici, assegnando un nome a quelli attivi. Ad esempio, il bottone con cui si esce dalla *vista setup* è impostato nella maniera seguente.

```
PTGui->addWidgetDown(new ofxUIButton(dim, dim, false,
                                     "Quit setup"));
```

PTGui è il nome del puntatore alla classe *ofxUICanvas*, *addWidgetDown* specifica che tale oggetto va posizionato al di sotto del precedente, *ofxUIButton* richiede che sia un oggetto di tipo "bottone", *dim* specifica le dimensioni dell'oggetto e la stringa *"Quit setup"* il suo nome. Dopo aver posizionato tutti gli oggetti viene chiamata la seguente funzione.

```
ofAddListener(PTGui->newGUIEvent,this,
              &SingleCameraMeanColorPT::updatePTGui);
```

[1] L'addon non è incluso nella distribuzione di Openframeworw, ma si trova in un *repository Git* all'indirizzo web github.com/rezaali/ofxUI

Nella quale viene passata una referenza alla funzione *updatePTGui*, che si occuperà di associare i comportamenti degli oggetti grafici alle corrispondenti variabili, definita nello stesso *scope*.
Il corpo della funzione *updatePTGui* è invece strutturato nella seguente maniera.

```
string name = e.widget->getName();
if(name == "Quit setup")
{
    ofxUIButton *button = (ofxUIButton *) e.widget;
    bQuitSetup = button->getValue();
}
```

Viene chiamata ogni qual volta si presenta un evento *e* del tipo *ofxUIEventArgs*, ovvero ogni qualvolta si ha un interazione con uno degli oggetti grafici presenti nel "canvas". L'oggetto viene poi identificato in base al suo nome e rispetto al suo tipo ne viene estratto un valore che viene poi memorizzato in una variabile precedentemente definita.

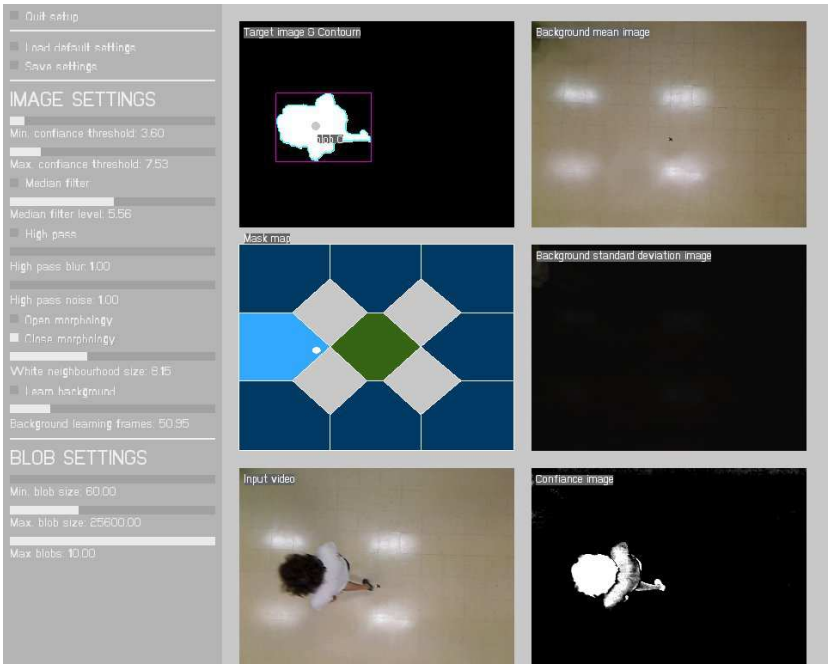


Figura 4.4: Screenshot del software nella *vista setup*.

4.2 *AppManager*

```
class AppManager
{
private:
    BaseAppSLM* appSLM;
    state currentState;
    slmPage appMenuGUI;
    void setupGUI();
public:
    void setup();
    void update();
    void updateGui();
    void draw();
    bool showSetup();
    bool loadApp(string appName);
    void passPositions(vector <ofPoint>& personCentroid);
    void drawMap(float _x, float _y, float _w, float _h );
};
```

appSLM

appSLM è un puntatore alla classe astratta *BaseAppSLM*. Quando viene selezionata un applicazione per la Stanza Logo-Motoria, viene creato un oggetto della classe associata all'applicazione, che quindi si comporterà secondo le sue caratteristiche.

currentState

Il comportamento della classe *AppManager* è determinato da questa variabile *enum*, che può assumere due valori: *applicationState* e *applicationMenuState*, a seconda che ci si trovi nel menu delle applicazioni o che un applicazione sia stata lanciata.

appMenuGUI* e *setupGUI()

La *vista applicazione* utilizza un interfaccia grafica *slmGUI* che è stata appositamente realizzata. *appMenuGUI* è l'oggetto associato a tale interfaccia, relativo al menu delle applicazioni. Nella funzione *setupGUI()* vengono definiti gli oggetti grafici che utilizza. In seguito verrà fatta una trattazione approfondita di questa libreria (vedi 4.2.4).

setup()*, *update()*,*updateGui()* *draw()

Come per le corrispondenti funzioni della classe *TrackingEngine*, anche queste vengono eseguite negli omonimi metodi all'interno della classe *testApp*. Una minima differenza la si ha per *updateGui()*, che viene chiamata soltanto quando è attiva la *vista applicazione*.

La funzione *setup()* è responsabile dell'inizializzazione della variabile *currentState* della classe al valore *applicationMenuState*, che corrisponde alla vista in cui si ha il menu delle varie applicazioni della Stanza Logo-Motoria da cui scegliere, e dell'inizializzazione della GUI attraverso la funzione *setupGUI()*.

showSetup()

showSetup() è una funzione con valore booleano di uscita. A essa è associato un bottone nelle interfacce grafiche della *vista applicazione* attraverso il quale l'utente può scegliere di accedere alla *vista setup*.

loadApp(string appName)

Questa funzione è responsabile del caricamento di una applicazione, quando viene richiesto dall'utente nel menù delle applicazioni.

passPositions(const vector <ofPoint>& personCentroid)

Attraverso questa funzione *AppManager* acquisisce le posizioni dei baricentri dei blob nell'immagine, quindi delle persone nello spazio fisico della Stanza Logo-Motoria. In seguito verranno passati alle applicazioni, mediante la classe *BaseAppSLM*.

drawMap()

Questa classe ha solo una funzione grafica. La sua funzione è visualizzare la mappa in cui è sono rappresentati gli utenti dello spazio fisico della Stanza Logo-Motoria mentre si muovono in essa.

4.2.1 La classe astratta *BaseAppSLM*

BaseAppSLM è una classe astratta che definisce la struttura delle applicazioni per la Stanza Logo-Motoria (vedi figura 4.5). I suoi principali metodi *loadContent()*, *updateApp() updateGui()*, *pauseApp()*, *restartApp()* e quelli relativi alla GUI *showAppMenu()* e *showSetup()* sono virtuali e vengono poi ridefiniti nelle classi derivate.

4.2.2 *Resonant Memory*

La classe *ResonantMemory* è l'unica applicazione definita nel presente stato del software della Stanza Logo-Motoria. Questa applicazione è costituita da due parti

Mappa della superficie. Ovvero una struttura composta da un certo numero di oggetti "poligono" (rappresentati dalla classe *RegionPoly*) che suddividono lo spazio fisico di interazione (la superficie) e che sono associati ognuno a uno specifico suono. Tra questi

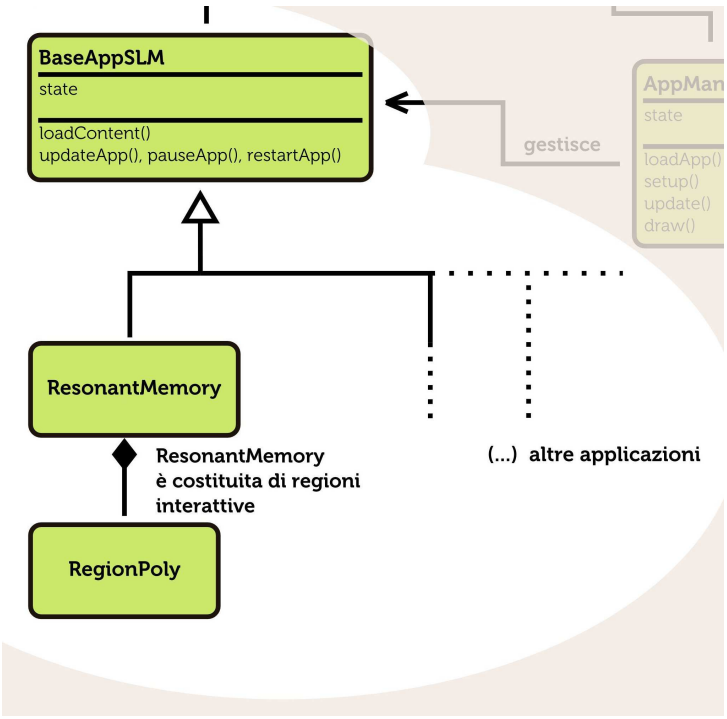


Figura 4.5: La classe astratta per le applicazioni della Stanza Logo-Motoria *BaseAppSLM*, e la classe applicazione *ResonantMemory* derivata da essa. L'applicazione *Resonant Memory* ha la struttura di una mappa interattiva in cui oggetti di tipo *RegionPoly* sono le parti costituenti.

oggetti ce n'è uno particolare, etichettato come la regione *start story*. Il suono di questa regione è la registrazione vocale di una narrazione, e viene riprodotta solo quando tutte le altre regioni sono state esplorate.

***Engine* dell'applicazione.** Ha il compito di caricare il contenuto didattico prescelto (vedremo di seguito come tale funzione viene espletata), ricevere il vettore con la posizione dei *blob* rilevata dal *Tracking Engine*, etichettare come esplorate le varie regioni della mappa e di aggiornare lo stato dell'applicazione e gestire la riproduzione dei suoni.

Di seguito sono illustrati e spiegati i suoi principali metodi e attributi.

currentState e *lastActiveState*

Il funzionamento della classe può essere rappresentato da una macchina a tre stati, definiti dalla variabile *enum* chiamata *state*. Il valore di tale variabile influenza il comportamento della logica dell'applicazione e della rappresentazione grafica della mappa della superficie di interazione. Lo stato corrente è memorizzato nella variabile *currentState*m mentre *lastActiveState* viene utilizzata per memorizzare l'ultimo stato attivo quando l'applicazione è in pausa.

exploreState In questo stato è richiesto all'utente di muoversi attraverso tutte le zone e ascoltare i suoni associati ad ognuna di esse, prima di incominciare l'attività vera e propria dell'applicazione *Resonant Memory*. Durante questa fase le regioni attraversate vengono marcate come "esplorate" e ne viene riprodotto il suono, a meno che non si tratti della regione *start story*, che in questa fase è inattiva.

playState Una volta che tutte le zone sono state esplorate si passa in questo stato, in cui la zona *start story* si attiva e, quando essa viene attraversata ha inizio la riproduzione audio della storia.

pauseState Quando tramite l'interfaccia grafica dell'applicazione, il sistema viene messo in pausa, la logica dell'applicazione si ferma e lo stato precedente viene memorizzato in *lastActiveState* in modo che, usciti dallo stato di pausa, i sistema riprenda da dove era stato interrotto.

Il costruttore *ResonantMemory()*

In questa funzione vengono inizializzati al valore di *esploreState* le variabili *lastActiveState* e *currentState*, viene eseguita la funzione *setupGUI()* in cui è definita l'interfaccia grafica dell'applicazione e vengono inizializzate alcune variabili relative anch'esse all'interfaccia grafica.

loadContent(string contentName)

Attraverso l'interfaccia grafica viene selezionato un contenuto per applicazione, tra quelli disponibili, ed il presente metodo viene invocato. Il suo compito è quello di leggere il file XML corrispondente al contenuto, e creare la struttura della mappa associando ad ogni regione un oggetto della classe *RegionPoly*, assegnandogli la forma ed il suono corrispondente ed eventualmente indicandolo come regione *start story*.

updateApp(const vector<ofPoint>& personCentroid)

La classe *AppManager()* passa all'applicazione corrente il vettore contenente i baricentri dei *blob* rilevati. All'interno di questa funzione tali posizioni vengono confrontate con gli oggetti *RegionPoly* che costituiscono la mappa, e attraverso un *hit test* viene determinato se i *blob* sono all'interno di una regione e quale. Il comportamento della regione viene quindi determinato dallo stato dell'applicazione.

drawMap(int x, int y, int w, int h)

Gli oggetti *RegionPoly* hanno un metodo *draw()*, che viene evocato all'interno della funzione *drawMap()*. La mappa completa costituita da tutte le regioni viene visualizzata quindi sullo schermo alle posizione *x*,*y* all'interno di una regione rettangolare di dimensioni *w* (larghezza) e *h* (altezza).

setupGUI(), rmGUI, showAppMenu(), showSetup(), pauseApp(), restartApp()

Queste funzioni si occupano di gestire l'interfaccia grafica dell'applicazione e di eseguire le istruzioni dell'utente. *setupGUI()* è il metodo in cui vengono definiti gli oggetti grafici che la GUI definita con l'oggetto *rmGUI* della classe *slmPage* utilizza. *showAppMenu()*, *showSetup()*, *pauseApp()* e *restartApp()* sono funzioni che eseguono i comandi dell'utente rispettivamente di: "mostra il menu delle applicazioni", "mostra la vista setup", "metti in pausa l'applicazione" e "fai ripartire l'applicazione".

maskMapXML

maskMapXML è un oggetto del tipo *ofxXmlSettings*. Un *addon* di Openframeworks che fornisce alcune funzioni per scrivere e leggere da file di tipo XML. I contenuti di *Resonant Memory* sono organizzati utilizzando questo formato, perciò tale oggetto viene utilizzato nel metodo *loadContent* per caricare il contenuto richiesto dall'utente.

4.2.3 La classe *RegionPoly* e la gestione di contenuti di *Resonant Memory*

Come si è avuto modo di vedere precedentemente, la classe *ReasonantMemory* utilizza degli oggetti di una classe chiamata *RegionPoly* e dei file di tipo XML per caricare i contenuti e per

eseguire le sue attività. Di seguito verrà analizzata la struttura del file XML in cui vengono codificati tali contenuti e il processo di caricamento di contenuti, che ci porterà ad illustrare la struttura della classe *RegionPoly*.

Il file XML di contenuto

XML è un linguaggio di *markup*, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo. I marcatori dell'XML sono chiamati *tag* che identificano un campo di contenuto [57]. Tali campi di contenuto possono a loro volta contenere altri campi, in maniera annidata.
La struttura del file XML che rappresenta un contenuto dell'applicazione *Resonant Memory* è la seguente.

```
<CONTENTNAME>FORESTA MAGICA</CONTENTNAME>
<REGION>
<REGIONID>0</REGIONID>
<SOUNDFILE>sounds/01.wav</SOUNDFILE>
<STARTSTORYREGION>0</STARTSTORYREGION>
<VERTEX>
<X>0</X>
<Y>0</Y>
</VERTEX>
<VERTEX>
<X>0.3</X>
<Y>0</Y>
</VERTEX>
<VERTEX>
<X>0.3</X>
<Y>1</Y>
</VERTEX>
<VERTEX>
<X>0</X>
<Y>1</Y>
</VERTEX>
</REGION>
<REGION>
.
.
.
</REGION>
```

Il file XML ha come primo campo `CONTENTNAME` all'interno del quale è definito il nome del contenuto. I restanti campi sono le regioni (`REGION`)che costituiscono la mappa.

A ognuna di esse è associata una geometria (definita dai vertici VERTEX di un poligono, una coppia di valori X e Y), da un suono (il cui percorso relativo è definito in SOUNDFILE) e un valore binario, vero se la regione è del tipo *start story*. I poligoni delle regioni sono definiti all'interno di un quadrato di lato unitario. Le coordinate dei loro vertici hanno quindi un valore compreso tra 0 e 1. L'insieme delle regioni forma la mappa sonora che rappresenta la struttura di un contenuto dell'applicazione *Resonant Memory*.

La classe *RegionPoly*

La funzione *loadContent()* ha il compito di leggere il file XML a cui è associato un contenuto e creare tanti oggetti del tipo *RegionPoly* quante sono le regioni della mappa. Questa classe ha a serie di attributi attraverso i quali è possibile impostare la geometria (il vettore di *ofPoint vertices*), il suono associato (caricandolo sull'oggetto della classe *ofFmodSoundPlayer*, *sp*), e identificarlo o meno coma la regione *start story* (il booleano *bStartStory*).
Successivamente, nella fase di *update()* dell'applicazione, la posizione dei *blob* viene passata al metodo *setOccupied()*, che restituisce un valore vero nel caso il *blob* sia all'interno della regione. Tale valore, rispettivamente allo stato corrente dell'applicazione, viene utilizzato per la riproduzione del suono associato, o ha soltanto effetti grafici.

4.2.4 La libreria *slmGUI*

Come accennato in precedenza, la classe *AppManager* e l'applicazione *ResonantMemory* utilizzano una libreria grafica appositamente realizzata 4.6. Le esigenze di GUI del programma

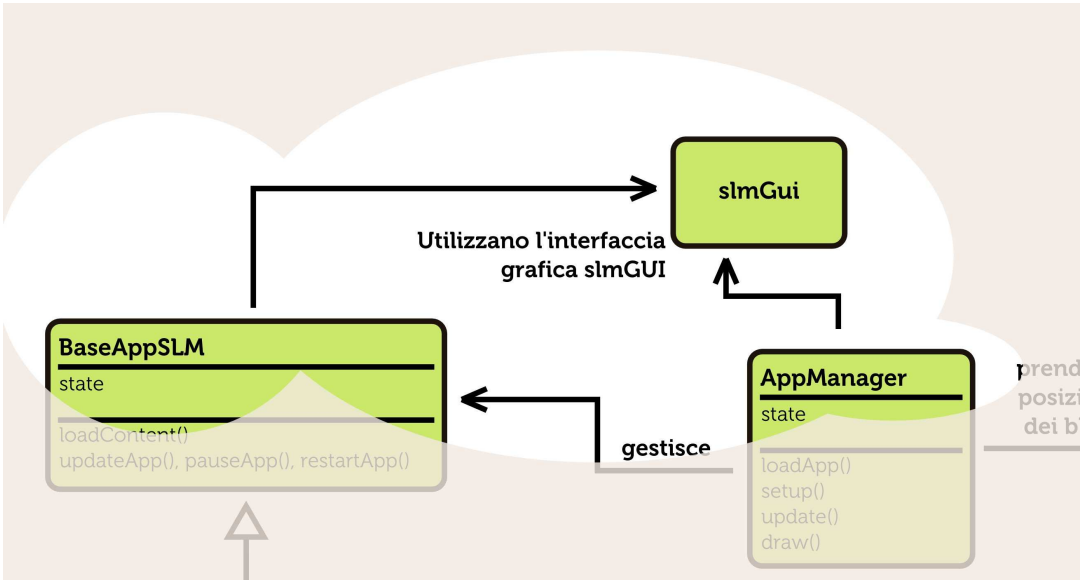


Figura 4.6: Le classi *BaseAppSLM* e *AppManager* utilizzano entrambe per la loro interfaccia grafica la libreria *slmGui* appositamente realizzata.

non sono sembrate tali da richiedere l'utilizzo di librerie grafiche avanzate, ma gli addon

per interfacce grafiche di Openframeworks (come *ofxUI*, utilizzato per la *vista setup*) non permettono grande libertà nel design visuale dei *widget*.
Per i widget interattivi *slmGUI* utilizza l'*addon ofxMSAInteractiveObject* [2], una classe i cui oggetti sono delle regioni nello schermo (rettangolari), sensibili a eventi di mouse e tastiera.
Una GUI è costituita da un oggetto della classe *slmPage*, la quale ha come attributo un vettore di puntatori alla classe astratta *slmWidget*. Da quest'ultima derivano tutti gli oggetti grafici della libreria (in figura 4.7 la struttura della libreria con tutti gli oggetti grafici presenti). Per "costruire" l'interfaccia si procede in due fasi. Prima si definisce il *widget* che

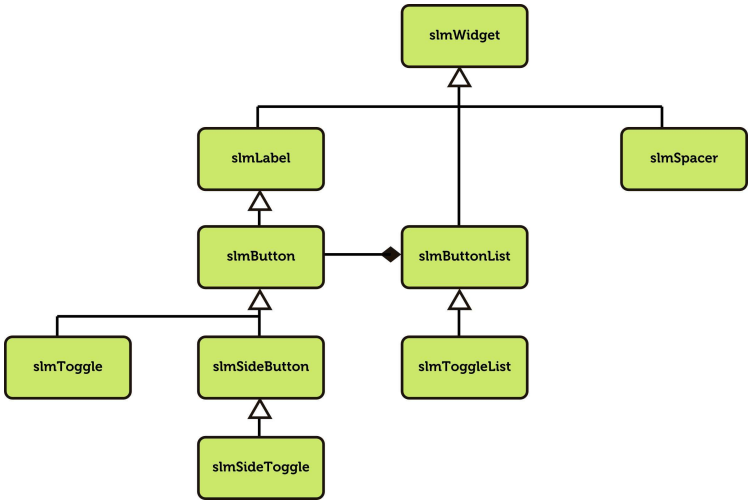


Figura 4.7: La struttura della libreria *slmGUI* secondo la notazione UML per i diagrammi delle classi.

si intende posizionare, istanziando un oggetto della classe corrispondente e, se interattivo, passandogli un puntatore alla variabile associata. Poi per ogni *widget* si invoca il metodo *addWidget(slmWidget* w)* della classe *slmPage*, il cui argomento per polimorfismo può essere uno qualsiasi degli oggetti grafici della libreria.

[2]Disponibile dal repository Git https://github.com/memo/msalibs/tree/master/ofxMSAInteractiveObject

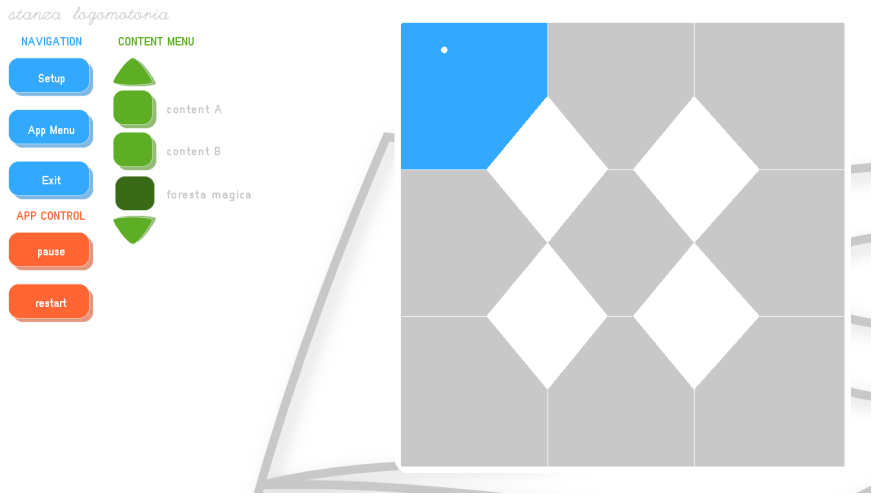


Figura 4.8: Screenshot del software nella *vista applicazione*. È in esecuzione l'applicazione *Resonant Memory*, ed è vilibile la mappa suddivisa in nove regioni, ed il cerchio bianco che rappresenta la posizione di un utente nella superficie

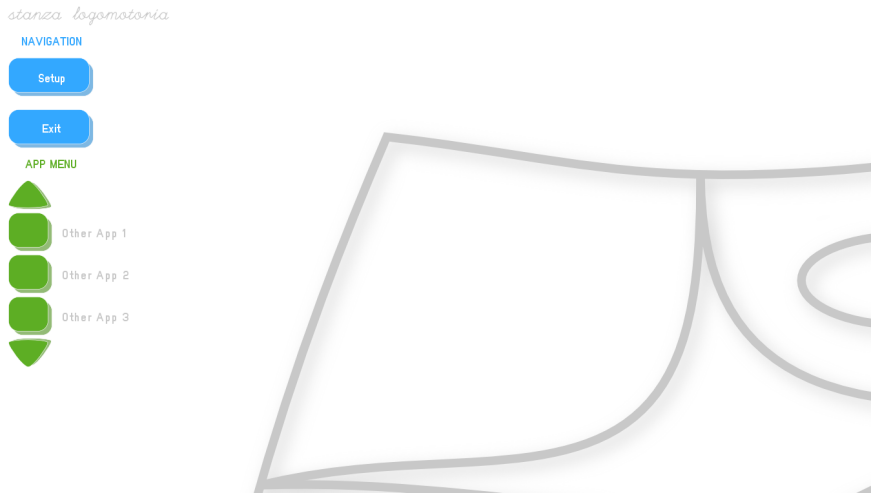


Figura 4.9: Screenshot del software nella *vista applicazione*. Menu delle applicazioni

Capitolo 5

Conclusioni

Come visto nel capitolo precedente, l'algoritmo di *background subtraction* incontra alcune difficoltà nel identificare i soggetti nella scena, quando le condizioni sono particolarmente sfavorevoli 3.5.2. Inoltre la possibilità di utilizzare il sistema con più di un utente è resa difficoltosa dal fatto che due o più persone in contatto tra loro verrebbero identificate dal sistema con un unico *blob*, con la conseguenza di compromettere il corretto funzionamento dell'applicazione. È necessario quindi che i successivi lavori si focalizzino su questi aspetti, in modo da rendere il sistema robusto sotto le più svariate condizioni e adatto a funzionare per una maggiore varietà di applicazioni. Possibili soluzioni possono venire sia dalla componente hardware che software del sistema. Ad esempio coadiuvando l' *averaging background subtraction* attualmente utilizzato con tecniche di *frame differencing* potrebbe essere una soluzione per garantire il corretto funzionamento sotto differenti condizioni di luce, oppure si potrebbe pensare all'adozione di sistemi più avanzati di modellizzazione dello sfondo come quelli che prevedono l'utilizzo di *codebook* [47]. Per quanto riguarda la possibilità di utilizzare il sistema in modalità multi-utente, si possono ottenere buoni risultati con l'utilizzo di filtri di Kalman [58], o l'algoritmo *optical flow* [59]. Un altra soluzione potrebbe derivare dall'utilizzo di tecnologie di acquisizione diverse dalla videocamera per il *motion tracking*, alcune delle quali sono state illustrate nella sezione 3.2, di cui però andrebbero valutati accuratamente i costi, che secondo le linee guida del progetto devono essere accessibili a un utenza quanto più ampia possibile.

5.1 Nuove applicazioni

Seguendo le intenzioni del progetto di ricerca della Stanza Logo-Motoria, che prevedono la realizzazione di ulteriori applicazioni per il sistema oltre a *Resonant Memory*, il software realizzato per il presente lavoro di tesi è strutturato in modo da prevedere la sua estensione con nuove porzioni di software associate con possibili nuove applicazioni. Come illustrato nel capitolo relativo alla struttura del programma, attualmente aggiungere una nuova applicazione al sistema si realizza attraverso la scrittura di una classe C++, secondo la struttura della classe astratta *BaseAppSLM*. Questa soluzione prevede che chi si occupi dello sviluppo

di una applicazione abbia accesso al codice dell'intero programma, con conseguenti rischi di corruzione. Una soluzione alternativa, più robusta, potrebbe consistere nel rendere il programma un effettivo *host* per applicazioni, che verrebbero sviluppate all'esterno di essa come dei *plugin* (come avviene ad esempio per la tecnologia VST).

5.2 Nuovi contenuti

Ogni applicazione della Stanza Logo-Motoria può essere utilizzata in diversi ambiti disciplinari e per diverse attività didattiche. È stata illustrata la struttura con cui le varie applicazioni caricano e utilizzano i vari contenuti 4.2.3 ma tuttavia il software non prevede un metodo standard per la creazione di contenuti, che attualmente consiste nel creare una copia di una delle cartelle associate con i contenuti tra quelle presenti, modificare il file XML al suo interno e caricare i file audio per il nuovo contenuto. In futuro, per poter permettere più facilmente agli insegnanti di creare i propri contenuti didattici si potrebbe creare un apposito software che automatizza tale procedura.

Bibliografia

[1] J. Noble. *Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks*. O'Reilly Media, Inc., 1st edition, 2009.

[2] S. Rafaeli. Interactivity: From new media to communication. *Advancing communication science: Merging mass and interpersonal processes*, 16:110–134, 1988.

[3] S Kiousis. Interactivity: a concept explication. *New Media & Society*, 4(3):355–383, September 2002.

[4] J.S. Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42(4):73–93, 1992.

[5] J.F. Jensen. Interactivity: Tracing a new concept in media and communication studies. *Nordicom Review*, 19:185–204, 1998.

[6] J.B. Walther and J.K. Burgoon. Relational communication in computer- mediated interaction. *Human Communication Research*, 19(1):50–80, 1992.

[7] R. Moreno and R. Mayer. Interactive multimodal learning environments. *Educational Psychology Review*, 19:309–326, 2007. 10.1007/s10648-007-9047-2.

[8] P. Barker. Designing interactive learning. In *Design and Production of Multimedia and Simulationbased Learning Material*. T. de Jong and L. Sarti, eds. Kluwer Academic, Dordrecht., 1994.

[9] G. R. Amthor. Multimedia in education: An introduction. *Int. Business Mag.*, 1992.

[10] J. Dewey. *Democracy and Education*. Free Press, New York, 1966.

[11] J. Piaget. *To Understand is to Invent: The Future of Education*. Grossman, New York, 1973.

[12] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, 1980.

[13] A. Rizzo and M. Palmonari. Context and consciousness: Activity theory and human computer interaction, bonnie a. nardi (ed.). *User Model. User-Adapt. Interact.*, 8(1-2):153–157, 1998.

[14] R Moreno. In *Technology-based education: Bringing researchers and practitioners together*. Information Age Publishing, 2005.

[15] J. J. Gibson. The ecological approach to visual perception. 1979. Houghton Mifflin.

[16] D. A. Norman. *The Design of Everyday Things*. Basic Books, New York, reprint paperback edition.

[17] S. Zanolla, S. Canazza, A. Rodà, and G. De Poli. A learning environment based on movement and sound interaction. In *Emerging Software for Interactive Interfaces, Database, Computer Graphics and Animation: Pixels and the New Excellence in Communicability, Cloud Computing and Augmented Reality*. Blue Herons, 2012. In press.

[18] J. Bruner. *Toward a theory of instruction*. Belknap Press of Harvard University Press, 1966.

[19] J. Bruner. *Processes of cognitive growth: Infancy.* Clark University Press, Worcester, MA, 1968.

[20] F. Varela. *The Embodied Mind: Cognitive science and human experience*. MIT Press, 1991.

[21] Sito web del progetto Enactive Network. www.enactivenetwork.org. Accessed: 24/09/2012.

[22] A. Paivio. *Mental representations: A dual coding approach*. Oxford University Press, 1986.

[23] H. Gardner. *Frames of Mind: The Theory of Multiple Intelligences*. Basic, 1983.

[24] R. E. Mayer. *Multimedia learning*, volume 41 of *Psychology of Learning and Motivation*. New York: Cambridge University Press., 2001.

[25] M. Krueger, T. Gionfriddo, and K. Hinrichsen. Videoplace - an artificial reality. human factors in computing systems. *ACM press*, 1985.

[26] R. Gehlhaar. SOUND=SPACE: An interactive musical environment. *Contemporary Music Review*, 6(1):59–72, 1991.

[27] G. Davenport and L. Friedlander. Contextual media. chapter Interactive transformational environments: wheel of life, pages 1–25. MIT Press, Cambridge, MA, USA, 1995.

[28] A. F. Bobick, S. S. Intille, J. W. Davis, F. Baird, C. S. Pinhanez, L. W. Campbell, Y. A. Ivanov, A. Schütte, and A. Wilson. The kidsroom: A perceptually-based interactive and immersive story environment. In *PRESENCE*, pages 367–391, 1999.

[29] S. Gumtau, P. Newland, C. Creed, and S. Kunath. Mediate: a responsive environment designed for children with autism. In *Proceedings of the 2005 international conference on Accessible Design in the Digital World*, Accessible Design'05, pages 14–14, Swinton, UK, UK, 2005. British Computer Society.

[30] M. C. Johnson-Glenberg, D. Birchfield, P. Savvides, and C. Megowan-Romanowicz. Semi-virtual Embodied Learning. real World STEM Assessment. In *Serious Educational Game Assessment: Practical Methods and Models for Educational Games, Simulations and Virtual Worlds*, pages 225–241. Sense Publications, Rotterdam, 2010.

[31] Quest to learn. www.instituteofplay.org/work/projects/quest-to-learn. Accessed: 24/09/2012.

[32] A. Camurri, S. Canazza, C. Canepa, A. Rodà, G. Volpe, S. Zanolla, and G. L. Foresti. The *stanza logo–motoria*: an interactive environment for learning and communication. In *Proc. of Sound and Music Computing Conference*, pages 353–360, Barcelona, July 2010.

[33] S. Zanolla, A. Rodà, F. Romano, F. Scattolin, G. L. Foresti, S. Canazza, C. Canepa, P. Coletta, and G. Volpe. Teaching by means of a technologically augmented environment: the Stanza Logo-Motoria. In *Proceedings of INTETAIN 2011 Conference*, 2011.

[34] S. Zanolla, F. Romano, F. Scattolin, A. Rodà, S. Canazza, and G. L. Foresti. When sound teaches. In S. Zanolla, F. Avanzini, S. Canazza, and A. de Götzen, editors, *Proceedings of the SMC 2011 - 8th Sound and Music Computing Conference*, pages 64–69, 2011.

[35] DT Pham, M. Al-Kutubi, Z. Ji, M. Yang, Z. Wang, and S. Catheline. Tangible acoustic interface approaches. *Manufacturing Engineering Centre, Cardiff University, UK and Laboratoire Ondes et Acoustique, ESPCI, Paris, France*, 2005.

[36] S.J. Campbell. Play+ space: an ultrasonic gestural midi controller. *Generate+ Test*, pages 43–49, 2005.

[37] B. Raj, K. Kalgaonkar, C. Harrison, and P. Dietz. Ultrasonic doppler sensing in hci. *Pervasive Computing, IEEE*, 11(2):24–29, 2012.

[38] G.W. Raes. Gesture controlled virtual musical instruments. logosfoundation.org/ii/gesture-instrument.html, 1999. Accessed: 24/09/2012.

[39] Pagina relativa allo strumento elettronico Theremin di Wikipedia in lingua italiana. it.wikipedia.org/wiki/Theremin. Accessed: 24/09/2012.

[40] J. Smith, T. White, C. Dodge, J. Paradiso, N. Gershenfeld, and D. Allport. Electric field sensing for graphical interfaces. *Computer Graphics and Applications, IEEE*, 18(3):54–60, 1998.

[41] T.G. Zimmerman, J.R. Smith, J.A. Paradiso, D. Allport, and N. Gershenfeld. Applying electric field sensing to human-computer interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 280–287. ACM Press/Addison-Wesley Publishing Co., 1995.

[42] Y. Visell, S. Smith, A. Law, R. Rajalingham, and J.R. Cooperstock. Contact sensing and interaction techniques for a distributed, multimodal floor display. In *3D User Interfaces (3DUI), 2010 IEEE Symposium on*, pages 75–78. IEEE, 2010.

[43] T. Murakita, T. Ikeda, and H. Ishiguro. Human tracking using floor sensors based on the markov chain monte carlo method. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 4, pages 917–920. IEEE, 2004.

[44] Sito ufficiale della piattaforma e linguaggio Processing. processing.org/. Accessed: 30/09/2012.

[45] Intervista a Zach Lieberman, fondatore di Openframeworks. www.instituteofplay.org/work/projects/quest-to-learn. Accessed: 01/10/2012.

[46] Sezione about del sito web di Openframeworks. www.openframeworks.cc/about/. Accessed: 30/09/2012.

[47] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Incorporated, 2008.

[48] S. Jabri, Z. Duric, H. Wechsler, and A. Rosenfeld. Detection and location of people in video images using adaptive fusion of color and edge information. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 4, pages 627–630. IEEE, 2000.

[49] T. Horprasert, D. Harwood, and L.S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. In *IEEE ICCV*, volume 99, pages 256–261, 1999.

[50] J. Gonzàlez, F.X. Roca, and J.J. Villanueva. Hermes: A research project on human sequence evaluation. In *ECCOMAS Thematic Conference on Computational Vision and Medical Image Processing (VipIMAGE)*. Taylor and Francis, 2007.

[51] Pagina relativa al filtro mediano di Wikipedia in lingua inglese. en.wikipedia.org/wiki/Median_filter. Accessed: 11/10/2012.

[52] Morfologia. definizioni e spiegazioni riportate da Leptonica. un sito web open source di indirizzo pedagogico riguardante software e conoscenze nell'ambito del processamento di immagini. www.leptonica.com/binary-morphology.html e http://www.leptonica.com/grayscale-morphology.html. Accessed: 11/10/2012.

[53] Morfologia matematica. Wikipedia in lingua inglese. en.wikipedia.org/wiki/Mathematical_morphology. Accessed: 12/10/2012.

[54] L. Vincent. Morphological area openings and closings for grey-scale images. *NATO ASI Series F Computer and Systems Sciences*, 126:197–197, 1994.

[55] K. Chinnasarn, Y. Rangsanseri, and P. Thitimajshima. Removing salt-and-pepper noise in text/graphics images. In *Circuits and Systems, 1998. IEEE APCCAS 1998. The 1998 IEEE Asia-Pacific Conference on*, pages 459–462. IEEE, 1998.

[56] D. Ze-Feng, Y. Zhou-Ping, and X. You-Lun. High probability impulse noise-removing algorithm based on mathematical morphology. *Signal Processing Letters, IEEE*, 14(1):31–34, 2007.

[57] Pagina relativa al linguaggio XML di Wikipedia in lingua italiana. it.wikipedia.org/wiki/XML. Accessed: 16/10/2012.

[58] X. Yun and E.R. Bachmann. Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking. *Robotics, IEEE Transactions on*, 22(6):1216–1227, 2006.

[59] D. DeCarlo and D. Metaxas. The integration of optical flow and deformable models with applications to human face shape and motion estimation. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 231–238. IEEE, 1996.