Università degli studi di Milano
Facoltà di scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione
Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

# UN SISTEMA PER LA GESTIONE AUTOMATICA DEL PANORAMA SONORO IN FUNZIONE DELLA POSIZIONE DEGLI ASCOLTATORI

Relatore: Prof. Luca Andrea Ludovico
Correlatore: Dott. Adriano Baratè

Elaborato finale di
Davide Savarè
Matricola 749372

Anno accademico 2009 – 2010

*A mia Mamma*

*e alla mia famiglia.*

# SOMMARIO

# 1. INTRODUZIONE

Il tirocinio si è concentrato sulla progettazione e realizzazione prototipale di un programma per la gestione automatica dell'ambiente sonoro: in funzione della posizione degli ascoltatori, si modificano i parametri principali delle tracce audio che costituiscono il panorama sonoro. All'ambiente vuoto corrisponde il silenzio, l'ingresso di una persona nella scena attiva la riproduzione di un campione audio tra quelli disponibili, l'uscita dalla scena ne provoca l'interruzione.

Tale ambito di interazione tra l'uomo e gli strumenti informatici per la gestione di contenuti audio (ma anche multimediali) è applicabile a scopi artistici come le installazioni multimediali e a scopi didattici per avvicinarsi agli elementi sonori anche in rapporto agli altri sensi: la spazialità fisica dell'ambiente, percepita attraverso la vista, il tatto e il movimento, viene codificata nella spazialità musicale ed elaborata dall'udito. Ecco quindi che la percezione sonora rappresenta la propria posizione spaziale.

Il progetto è da considerarsi un primo approccio per un applicativo destinato alla gestione delle installazioni artistiche multimediali, che per ottenere l'effetto desiderato fanno leva proprio sull'interazione tra sensi diversi e in cui il fruitore finale viene messo in gioco attivamente: la propria posizione in rapporto all'installazione diviene al tempo stesso elemento di costruzione e di fruizione dell'opera.

Le fasi preliminari si sono orientate alla ricerca di metodi già esistenti in letteratura per la localizzazione e il tracking di elementi in un ambiente, cui è seguita la pianificazione del software da implementare. La progettazione è stata orientata verso due moduli funzionali, uno relativo alla localizzazione attraverso una rete di sensori ed uno relativo alla gestione dell'audio.

Nella valutazione della rete di sensori, sono stati presi in considerazione parametri di efficienza e prestazioni, utilizzo in contesti analoghi, reperibilità e costi dell'apparecchiatura, difficoltà nella messa in opera.

Per il controllo dell'audio, la scelta è ricaduta sulle librerie open source irrKlang, per la compatibilità con il linguaggio C# e la completezza delle funzioni predefinite. Il codice implementato è infine stato sottoposto ad alcune sessioni di test con filmati di prova per verificarne il funzionamento, con esito positivo.

Vengono quindi proposte alcune tendenze per gli sviluppi futuri dell'applicazione.

# 2. STUDI E RICERCHE PRELIMINARI

## 2.1. DESCRIZIONE DEL MODELLO

Si è preso come modello una stanza, chiusa su tre lati e con accesso vincolato al quarto, dotata di diffusione audio stereofonica e monitorata da una telecamera posizionata sul soffitto (pertanto tutti i riferimenti spaziali del presente testo sono da considerarsi in riferimento alla vista in pianta della stanza, considerando come lato di ingresso il lato inferiore). L'altezza a cui posizionare la telecamera e quindi l'altezza della stanza dipende dal tipo di ottica e deve consentire di riprendere tutta l'area, possibilmente senza o comunque minimizzando le distorsioni dell'immagine che potrebbero influenzare negativamente le operazioni di rilevazione; il punto di ripresa ottimale si trova al centro del soffitto con ripresa ortogonale al pavimento. La stanza non contiene al suo interno oggetti fissi o elementi architettonici tali da ingombrare la superficie calpestabile; la grandezza non è stata definita a priori ma è stata scelta in modo da ospitare il numero massimo previsto di occupanti e consentirne il movimento senza sovrapposizioni. Si è considerato un unico lato di accesso e agli utenti è consentito di muoversi liberamente con i vincoli di non correre e non toccarsi l'un l'altro nel loro percorso. Sono state osservate le condizioni ottimali di illuminazione, in modo da garantire uniformità alla scena, limitare le interferenze derivate dalle ombre e quindi non falsare la valutazione delle posizioni. Le variazioni di luminosità e contrasto infatti modificano le soglie utilizzate per tracciare gli elementi all'interno dell'area di lavoro, provocando false rilevazioni.

Per modellare correttamente gli obiettivi del sistema, si è analizzato come in generale le persone possono interagire con l'ambiente in cui si trovano e con gli altri individui presenti, e sono state individuate due modalità: la prima attraverso la propria posizione assoluta e la seconda attraverso azioni che coinvolgono altri elementi come

persone o oggetti. Alla prima appartengono l'ingresso e l'uscita nella stanza, il movimento al suo interno, l'adozione di una determinata posizione del corpo, la modalità di spostamento (correndo, saltando, camminando, etc.) e le traiettorie scelte (rette, curvilinee, spezzate, etc.). Alla seconda appartengono invece l'interazione con gli altri occupanti, come l'avvicinamento, il contatto, l'allontanamento, il dialogo, e l'utilizzo di eventuali oggetti presenti, ad esempio lo spostamento di un vaso o l'accensione di una lampada. Una ricerca similare e parallela ha permesso di individuare una serie di eventi musicali che un sistema di sonorizzazione deve necessariamente offrire: il controllo del volume e del pan, l'applicazione di effetti audio (riverberi, chorus, equalizzatori, distorsori, etc.) e la modifica dei relativi parametri, l'attivazione, l'arresto, la modifica della velocità di riproduzione e della modalità loop, la riproduzione di singoli effetti sonori. Un progetto di sonorizzazione di ambienti deve decidere quali azioni degli utenti considerare tra quelle possibili e associare ad ognuna un evento tra quelli forniti dal proprio sistema: questo è un passaggio molto importante in quanto descrive in che modo andrà ad operare il proprio modello.

Nel progetto si è scelto di focalizzare l'attenzione al rapporto degli individui con l'ambiente attraverso la codifica di volume e pan in funzione della posizione assoluta, trascurando l'interazione con altri soggetti ed elementi presenti nella stanza. Da questa analisi sono scaturite le seguenti associazioni tra azioni ed eventi sonori:

1. ingresso nella stanza: l'utente occupa una porzione di spazio e una delle tracce audio disponibili è associata alle sue coordinate, attivando la riproduzione
2. movimento verticale: modifica il volume della propria traccia in modo direttamente proporzionale alla distanza dall'ingresso; alla linea di ingresso è associato il valore minimo e al lato opposto quello massimo

3. movimento orizzontale: modifica il pan della propria traccia; la posizione centrale nel panorama sonoro è assegnata all'asse verticale della stanza mentre ai lati sinistro e destro sono assegnati gli estremi
4. uscita dalla stanza: l'utente esce dall'ambiente e la traccia a lui associata è bloccata e resa disponibile per un nuovo utente.

Le tracce audio sono diffuse nell'ambiente, in modo che gli utenti possano sentire in real-time la costruzione globale e le variazioni del panorama sonoro per poterlo influenzare a piacimento attraverso il loro movimento. Il numero dei diffusori può variare in relazione alla grandezza dell'ambiente, ma sempre in modalità stereofonica: nella configurazione minima si collocano due diffusori sulla parete opposta all'ingresso, altrimenti si creano due linee di diffusori da collocare sulle pareti laterali. Il numero massimo di occupanti deve essere definito a priori in rapporto alla grandezza dell'ambiente e al numero di tracce audio disponibili.

## 2.2. DESCRIZIONE DEL PROTOTIPO

Si è cercato di realizzare un'area di test il più possibile fedele al modello esposto in rapporto alle apparecchiature, agli ambienti e alle risorse disponibili, mantenendo comunque l'obiettivo finale. Per poter riprendere un area sufficientemente estesa da consentire buone possibilità di movimento e minimizzare la distorsione dovuta all'inclinazione dell'apparecchio di ripresa, è stato necessario posizionare una telecamera digitale nel punto medio del lato d'ingresso dell'area di ripresa a circa 7 metri di altezza all'esterno di un palazzo per riprendere il piazzale sottostante: ciò non ha dunque consentito di allestire una stanza reale, ma un area calpestabile (stage) delimitata a terra da segni di confine. Si è preferito alla soluzione real-time, un approccio off-line per l'analisi della scena; sono stati registrati dei filmati campione in formato avi non compresso di dimensione 720 x 576 pixel a 25 frame per secondo, poi trasformati in sequenze di immagini in bianco e nero di dimensione 352 x 288 pixel ed elaborate per

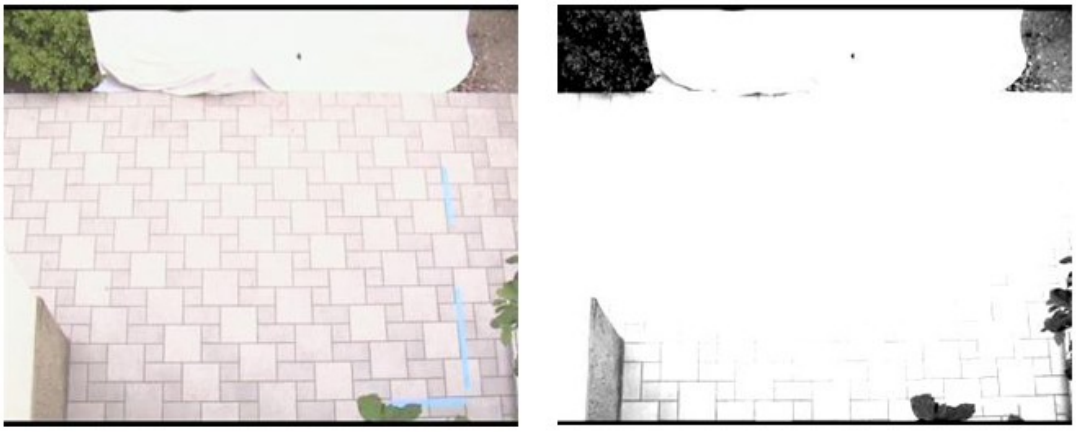

Figura 1 - L'area di ripresa prima e dopo l'applicazione degli effetti video.

garantire maggior contrasto tra sfondo ed ascoltatori, semplificando la rilevazione del movimenti (Figura 1).

L'area calpestabile è stata poi ridotta ulteriormente in fase di analisi dell'immagine per valutare unicamente lo spazio utile ed escludere le aree di scarso interesse ai fini del tracking (oggetti ai lati dello stage, elementi di disturbo come piante, aiuole, etc.). Le immagini così ottenute sono state poi analizzate dal programma che durante la sua esecuzione ha riprodotto l'ambiente sonoro relativo alla sequenza: non è stato quindi necessario allestire un impianto di diffusione per lo stage.

Per l'ambiente sonoro sono stati utilizzati file audio wav con codifica PCM non compresso a 44100Hz di frequenza di campionamento e 16 bit di quantizzazione; tutti i sample sono stati realizzati della medesima lunghezza per poter essere riprodotti in loop simultaneamente e sincronizzati rispetto ad un segnale di sincronizzazione costituito da un sample vuoto.

## 2.3. STRUMENTI PER IL TRACKING

La prima fase di ricerca è stata incentrata sulle metodologie per il tracking degli oggetti in un ambiente. Questo processo si occupa di determinare la posizione di uno o più oggetti all'interno di un ambiente e seguirne il movimento nel tempo[1][2]. Sono state prese in considerazione varie tecnologie per determinare la posizione degli ascoltatori, valutandone l'efficienza, la precisione, la reperibilità, la messa in opera e il costo. Non si è reputato necessario raggiungere particolari livelli di precisione nella determinazione delle coordinate spaziali degli ascoltatori, in quanto una maggior granularità non apporta miglioramenti sensibili alle variazioni dei parametri audio. Sono inoltre stati esclusi dalla progettazione i casi di occlusione, ossia la configurazione in cui due oggetti separati si avvicinano a tal punto da mascherarsi l'un l'altro per poi separarsi nuovamente: durante l'intervallo di sovrapposizione non è possibile discriminarli e in fase di separazione risulta difficile riassegnarne correttamente il posizionamento, se non attraverso un algoritmo di predizione [3]. Tale processo consiste nella valutazione delle posizioni precedenti per identificare l'andamento dell'oggetto secondo velocità e direzione e definire alcune possibili posizioni future; più numerose e dettagliate sono le proiezioni, maggiore è la complessità dell'algoritmo e del calcolo. Di seguito sono riportate alcune tecnologie e metodi presi in considerazione nella progettazione per effettuare il tracking.

### *2.3.1. Sensori di pressione a tappeto*

Si tratta di sottili superfici utilizzate anche negli impianti di sicurezza, sensibili alle variazioni di pressione. Superata una determinata soglia di pressione, si comportano come interruttori: viene inviato alla centralina di controllo un segnale per indicarne l'attivazione senza specificare l'intervallo di variazione. Una rete di questi tappeti è in grado di determinare gli spazi occupati all'interno di un ambiente con una percentuale di errore minima, in quanto l'attivazione è meccanica, e con una granularità dipendente dalle dimensioni del tappeto. A meno che non si preveda una aggiornamento ciclico,

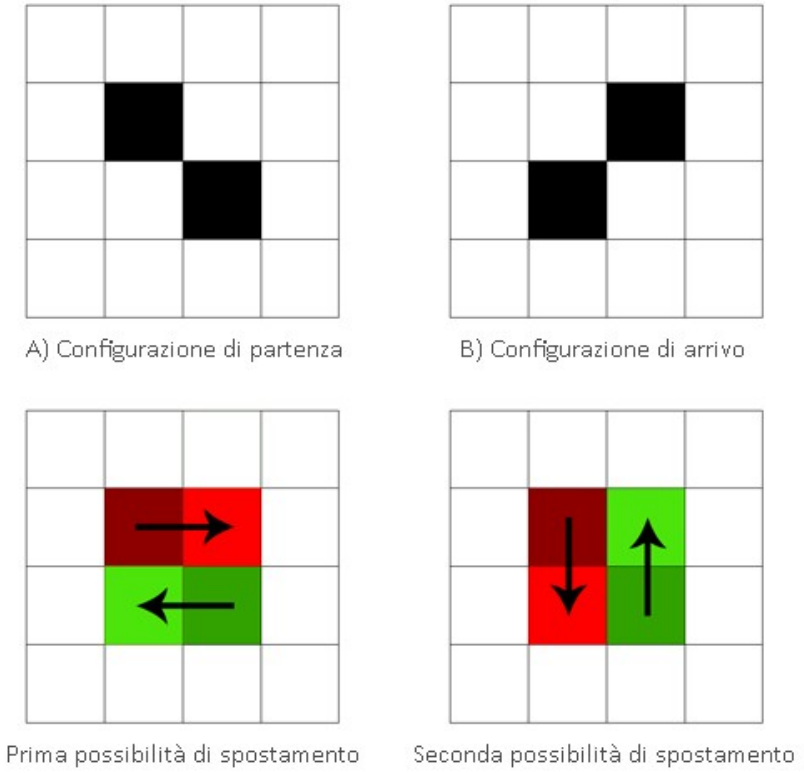


**Figura 2 - Esempio di indeterminazione**

questa configurazione segnala unicamente le variazioni di pressione, ovvero solo il cambiamento di stato di un particolare sensore e non lo il suo stato ad intervalli regolari.

Per questo motivo, per valutare gli spostamenti, è necessario applicare un algoritmo in grado di stabilire le posizioni istantanee in rapporto a quelle precedenti. In ogni modo a causa della scarsa precisione, infatti su un sensore potrebbero essere ospitate più persone e spostamenti inferiori alla grandezza del tappeto non verrebbero rilevate, il metodo si espone a situazioni di indeterminazione destinate a non essere risolte: spostamenti da e verso celle contigue verrebbero interpretati in modo aleatorio dal sistema (Figura 2). Oltre a ciò, si aggiunge il costo elevato di questi tipi di apparati, che non consente l'applicazione su ambienti medio grandi.

### *2.3.2. Sensori di temperatura*

Sono comuni sensori utilizzati negli impianti di sicurezza domestica e funzionano come interruttori alla variazione di temperatura: quando il corpo umano passa nel loro raggio d'azione viene rilavata la variazione di temperatura e inviato un segnale alla centralina di controllo. Con sensori di ultima generazione è possibile configurare il raggio d'azione e incrociando più sensori è possibile determinare le posizioni. Tuttavia risulta abbastanza complesso determinare e tracciare i movimenti, in quanto la precisione non è elevata a causa del comunque elevato raggio d'azione (a meno di installare una rete di sensori a scacchiera per determinare righe e colonne occupate) e poiché il segnale viene inviato al variare della temperatura, nel caso di figure statiche non verrebbe rilevata nessuna variazione poiché il sensore si adatta al calore del corpo umano.

### *2.3.3. Laser scanner*

I laser scanner (o laser a scansione) sono dispositivi capaci di emettere un impulso elettromagnetico (il laser) e di ricevere il segnale riflesso, misurando l'intervallo di tempo trascorso e quindi la distanza tra lo strumento ed il punto rilevato; il raggio laser viene deflesso mediante un meccanismo di specchi rotanti ed oscillanti che colpisce il terreno in punti contigui. Questo sistema opera misurando anche decine di migliaia di punti al secondo formando delle "nuvole di punti" e fornisce per ogni misurazione l'intensità del segnale di ritorno, consentendo il rilevamento di modelli tridimensionali di oggetti a scale e risoluzioni differenti. A seconda dello strumento utilizzato si ottengono precisioni e distanze massime misurabili differenti, fino ad un massimo di 4 - 6 mm a circa 100 m; alle posizioni così ottenute è sufficiente legare con un algoritmo i dati relativi all'oggetto scansionato e seguirne il movimento rispetto alle posizioni precedenti. Questa soluzione offre precisione e stabilità alla progettazione, ma ha costi molto elevati e devono essere considerate anche precauzioni sanitarie per l'utilizzo nel tracking di persone, in quanto

propaga nell'ambiente un raggio laser con rischi per la vista, molto remoti ma da considerare.

### *2.3.4. Telecamere*

Questi tipi di risorse risultano di particolare interesse nel tracking in quanto molto versatili, di facile reperibilità e con costi spesso contenuti (si pensi soprattutto alle webcam); da non trascurare il fatto che tali apparecchiature sono in grado di fornire video in formato digitale, che possono venire esaminati direttamente senza ulteriori passaggi, e che spesso sono comandabili ad accesso diretto attraverso librerie dedicate fornite dal produttore o protocolli standard forniti dal sistema operativo, come Windows Imaging Architecture (WIA)[4]. Un algoritmo analizza la sequenza di frame video e restituisce in output gli spostamenti degli oggetti al variare dei frame; esistono molti algoritmi per fare ciò e ognuno presenta punti di forza e debolezze a seconda dello scopo per cui sono utilizzati: pertanto è importante valutare il tipo di oggetti da tracciare per ottenere risultati soddisfacenti. I sistemi di tracking sono costituiti da due elementi principali, il primo che si occupa di rappresentazione e localizzare gli obiettivi, attraverso la rilevazione di parametri come contorni, variazioni rispetto ad un'immagine di riferimento, confronti tra riprese della stessa scena da angolazioni differenti (visione stereoscopica), e il secondo che associa i dati caratterizzanti gli oggetti target. Nel presente elaborato si è scelto di confrontare i frame video estratti dai filmati con un frame di riferimento contenente lo stage vuoto: le variazioni costituiscono oggetti cui verranno associati i parametri definiti dalle classi implementate.

### *2.3.5. Algoritmi per il tracking video*

Un oggetto nell'ambito della visione può essere descritto come un insieme di pixel che l'occhio considera un'entità unica, definizione che risente molto dell'interpretazione umana in grado di definire autonomamente un contesto per ciò che sta visualizzando.

Esistono quindi diversi metodi per distinguere gli oggetti in un frame e seguirne il movimento, tutti basati sul riconoscimento di una particolare caratteristica in grado di descriverlo. Per semplificare l'analisi, sono stati considerati unicamente algoritmi di segmentazione, cioè che dividono l'immagine in insiemi di pixel per semplificare o cambiare la rappresentazione dell'immagine in qualcosa che sia più significativa o comunque più semplice da analizzare. In questo modo sono raggruppati tutti i pixel che condividono le medesime caratteristiche.

Gli algoritmi di *color based tracking* [5] [6] analizzano l'immagine ricercando tutte le aree riconducibili ad un unico colore o sfumature di esso per determinare gli oggetti; le informazioni di colore infatti sono strettamente legate a porzioni di immagine relativamente grandi e l'analisi di queste ultime consente di aggregarle in oggetti più grandi. Una delle tecniche per analizzare l'immagine è quella di confrontare l'istogramma del colore ricercando i valori massimi, che rappresentano il colore predominante per un determinato pixel.

Gli algoritmi di *edge detection* [7] ricercano linee e contorni all'interno di un immagine ricercando i punti di discontinuità, ovvero con rapide variazioni della luminosità. Lo scopo di determinare questi particolari punti è quello di catturare particolari condizioni o cambiamenti nelle proprietà dell'oggetto; tali eventi sono rappresentati da discontinuità nella profondità dell'immagine o nell'orientamento della superficie, dal cambiamento di materiali o dell'illuminazione della scena. Nel caso ideale questo approccio produce un insieme di linee che rappresentano i contorni e i tratti salienti di un oggetto, riducendo il contenuto informativo a favore dei tratti essenziali.

Gli algoritmi di *region growing* o di espansione [8], come quello utilizzato nel presente elaborato, sono semplici metodi di segmentazione dell'immagine per determinare insiemi di punti comuni in un immagine; sono classificati anche come metodi in segmentazione *pixel a pixel* poiché la definizione di un'area ha inizio con

l'individuazione di un singolo punto iniziale. Definito il punto di inizio di un area, l'algoritmo esamina i pixel adiacenti per determinare quali possano essere accorpati all'oggetto; il processo è quindi iterato e con l'aggiunta successiva di altri punti si espande l'area occupata. Per determinare l'inizio e gli altri punti di un oggetto si fa riferimento a criteri definiti dall'utente secondo le necessità, come la variazione di colore rispetto ad un'immagine di riferimento.

## 2.4. STRUMENTI PER LA PROGRAMMAZIONE

Si è scelto di utilizzare come linguaggio di programmazione C# [9][10] per la stabilità e l'ampia gamma di librerie disponibili e per la compatibilità con la libreria irrKlang per la gestione dell'audio. È un linguaggio di programmazione object-oriented sviluppato da Microsoft all'interno del progetto .NET, e successivamente approvato come standard ECMA. La sintassi del C# prende spunto da quella del Delphi, del C++, da quella di Java e da Visual Basic per gli strumenti di programmazione visuale.

Nella fase iniziale sono stati eseguiti alcuni test sulle librerie disponibili, per comprendere quali fossero le scelte più adeguate per gestire le strutture dati da implementare, quali classi predefinite potessero essere utilizzate e quali invece si sarebbero dovute creare ad hoc. Sono stati testati tra gli altri le funzionalità degli array (dichiarazione, capacità statica e dinamica, accesso, inserimento ed eliminazione degli elementi), le classi List<>, Queue, Bitmap e le principali classi della libreria irrKlang.

### *2.4.1. Classe List<>*

La classe List<> consente di organizzare collezioni di oggetti del tipo specificato e fornisce numerosi metodi per l'inserimento e l'eliminazione, la ricerca e l'accesso diretto agli elementi, il confronto, il conteggio e la selezione degli elementi in essa contenuti: è stata utilizzata per gestire la memorizzazione degli oggetti identificati. Consente il

dimensionamento dinamico, feature molto utile non sapendo a priori il numero di elementi da trattare: in fase di inserimento di un nuovo elemento, viene espansa automaticamente la struttura dati e ridimensionata in caso di eliminazione, modificando di conseguenza i valori degli indici. È indipendente dai tipi contenuti, che devono essere specificati in fase di dichiarazione, e dispone di un indice di interi in base zero per accedere agli elementi.

### *2.4.2. Classe Bitmap*

Bitmap è un oggetto utilizzato per operare con immagini definite dai dati pixel e dai relativi attributi. Sono disponibili molti formati standard per il salvataggio di una bitmap in un file: BMP, GIF, EXIF, JPG, PNG e TIFF. È possibile creare immagini da file, flussi e altre origini mediante uno dei costruttori e salvarle in un flusso o nel file system con il metodo Save. Le immagini vengono disegnate sullo schermo o nella memoria mediante il metodo DrawImage dell'oggetto Graphics. Nel progetto, questa classe viene utilizzata per elaborare i frame estratti dai video: attraverso il costruttore viene costruita un'immagine a partire dai BMP dei frame estratti; il metodo GetPixel(int x,int y) restituisce un oggetto di tipo Color, costituito dai membri RBG e Alfa (per la trasparenza) ciascuno rappresentato da un intero, corrispondente al pixel della riga X e colonna Y.

### *2.4.3. Libreria irrKlang*

irrKlang[11] è una potente API (Application Programming Interface: funzioni software ad alto livello in grado di attivare software di basso livello per gestire l'hardware) per la riproduzione di suoni in applicazioni 3D e 2D come giochi e applicazioni multimediali; supporta svariati formati audio (.wav, .ogg, .mp3, .flac, e altri) estendibili ulteriormente attraverso plugin. È compatibile con le principali piattaforme (Windows 98, ME, NT 4, 2000, XP, Vista, Windows 7, Linux e Mac OS X) e portabile su differenti architetture senza dover modificare il codice; è rilasciato sotto licenza freeware per scopi

non commerciali. Dispone del supporto per suoni 3D su tutte le piattaforme e driver audio, utilizzabile nello sviluppo di software senza caricare eccessivamente la CPU e di alcuni effetti sia per l'audio 3D che 2D (chorus, compressore, distorsore, echo, flanger, gargle, riverberi ed equalizzatore). È possibile abilitare e disabilitare questi effetti e modificarne i parametri durante la riproduzione di ogni suono, senza interruzioni.

La riproduzione è affidata alla classe ISoundEngine che implementa un motore sonoro per la riproduzione di file audio in 2D e 3D attraverso le funzioni play2D e play3D; queste funzioni ricevono come parametro il percorso del file da riprodurre e altri valori di configurazione come l'abilitazione della riproduzione in loop e l'inizio riproduzione automatica. Questa funzione restituisce un oggetto di tipo Isound che rappresenta il suono attualmente in esecuzione e che mette a disposizione proprietà e metodi per il controllo del campione. Non è possibile creare un'istanza di Isound attraverso il suo costruttore perché non funzionerebbe, ma soltanto assegnando un oggetto tramite la funzione play2D o play3D da un ISoundEngine: questo perché ad ogni suono deve essere associato un motore sonoro per la riproduzione. In particolare sono state utilizzati i seguenti parametri della classe ISound.

1. Looped: restituisce o imposta la riproduzione in loop del suono. Nel caso in cui tale cambiamento avvenisse durante la riproduzione, il suono si mette in pausa dopo aver completato il ciclo di riproduzione corrente o viceversa continuerà la riproduzione da capo. Nel caso in cui la variazione sia applicata quando il suono è in pausa, l'invocazione di questo metodo non ha effetto.
2. Pan: imposta il valore del pan di un suono. Può assumere valori compresi tra -1 e 1 dove 0 rappresenta il pan centrale.
3. PlaybackSpeed: imposta o restituisce la velocità di riproduzione di un suono. Riproducendo un sample a velocità maggiore o minore, si aumenta o diminuisce la sua frequenza rendendo il suono più acuto o più grave. Questa

caratteristica non può essere utilizzata in coppia con il parametro enableSoundEffect del metodo Play2D e con il metodo play3D.

4. PlayPosition: imposta o restituisce la posizione attuale di riproduzione di un suono in millisecondi; nel caso in cui non fosse possibile determinare tale valore, ad esempio perché il suono è già terminato, restituisce -1. Questa caratteristica è stata utilizzata per sincronizzare tra loro le tracce del panorama sonoro: un sample vuoto viene mandato in esecuzione all'inizio del programma e fornisce il valore in millisecondi del punto di esecuzione per i nuovi suoni che devono essere riprodotti.
5. Volume: restituisce o imposta il volume del suono. Tale valore è compreso tra 0 (mute) e 1 (volume massimo) e deve essere moltiplicato al volume master del ISoundEngine collegato e agli altri parametri che concorrono alle variazioni di volume, come la distanza dell'ascoltatore nella riproduzione 3D.

# 3. PROGETTAZIONE E REALIZZAZIONE

L'algoritmo progettato, confronta i pixel di ogni frame con un'immagine dell'ambiente vuoto e determina quali sono differenti; per ognuno di essi verifica che appartenga ad oggetto già identificato nel frame, altrimenti ne crea uno nuovo. Ultimata l'analisi dell'immagine, verifica se gli oggetti trovati nel frame corrente esistono già nel frame precedente: in caso positivo vengono aggiornati i valori di quest'ultimo, altrimenti un nuovo oggetto è inserito nel bounding box (struttura dati contenente gli oggetti attivi)[12]. Ottenute le coordinate, riproduce le tracce audio calcolando i parametri di volume e pan a partire dalle coordinate x e y degli oggetti.

Per il riconoscimento viene applicato un metodo definito di espansione, un processo bottom-up in cui con l'evolversi dell'analisi si aumenta la dimensione della riquadratura di un oggetto fino a contenerlo completamente (Figura 3). Si valuta dapprima un singolo pixel, poi l'intera riga e infine si legano tra loro le informazioni delle singole righe per ricostruire un oggetto. In questo modo il primo pixel diverso dallo sfondo costituirà un riquadro degenere definito da un unico punto, che con il procedere dell'esecuzione diventerà una sequenza di pixel diversi dallo sfondo, ossia un altro tipo di riquadro degenere costituito da un punto iniziale e finale con la medesima coordinata y. Finita l'analisi dell'immagine, si valuta se ogni sequenza può essere unita ad altre per formare un riquadro vero e proprio con coordinate di inizio e fine differenti.

La scelta è ricaduta su questo metodo per la sua robustezza nei confronti di spazi vuoti (ossia pixel di un frame uguali all'immagine di riferimento) all'interno della figura da tracciare (nel caso del corpo umano sono rappresentati ad esempio dallo spazio tra braccia e busto) che sono ben presenti in oggetti di forma irregolare o variabile.

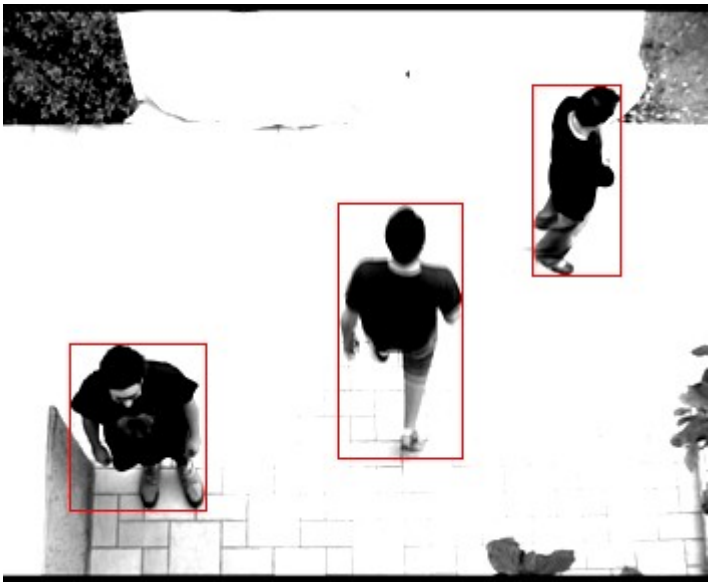

Figura 3 - Esempio di riquadratura delle figure

## 3.1. STRUTTURA GLOBALE DEL PROGRAMMA.

Il programma è diviso in due unità funzionali, che si è cercato di mantenere il più possibile separate: la prima si occupa di determinare la posizione degli ascoltatori nell'area di lavoro, la seconda di gestire le tracce audio dell'ambiente sonoro, e sono legate da una struttura dati contenente le coordinate degli oggetti. Il primo modulo scrive le coordinate nella struttura dati, chiamata *bounding box*, implementata attraverso la classe *List*<> mentre il secondo legge le coordinate e le trasforma in valori per i parametri delle tracce (volume e pan). Questa struttura è stata scelta per consentire la massima indipendenza tra l'algoritmo di riconoscimento e la gestione dell'audio, in modo da poter sviluppare in modo parallelo il software o utilizzare algoritmi di tracking differenti. Le immagini BMP da analizzare sono preparate estraendo i frame dai filmati registrati nel caso di un sistema off-line o tramite un video grabber in sistemi real-time. Il programma è costituito da un ciclo *for* che scandisce le immagini (un ciclo infinito nel caso di sistemi

real-time) e per ogni immagine sono scansionati tutti i pixel dell'area utile definita dalle variabili globali dichiarate ad inizio esecuzione e raggruppate nella variabile stage di tipo *obj*, che verrà utilizzata anche dalle funzioni di calcolo di pan e volume. Questa impostazione snellisce l'analisi diminuendo il carico computazionale, il tempo di esecuzione per ogni immagine e la possibilità di errore poiché elimina quelle porzioni di frame che sicuramente non sono di interesse.

Oltre alla dimensione dell'area di lavoro, all'inizio del codice sono definite le soglie di tolleranza per determinare gli oggetti.

1. *Range_color*: rappresenta la variazione minima per ottenere una modifica sensibile nel colore. Poiché si opera con immagini in bianco e nero, i valori dei canali R, G e B, definiti dalla classe *color*, sono sempre uguali tra loro e pertanto per riscontrare una differenza è sufficiente valutare un unico canale. Un valore troppo basso, aumenta la precisione nel determinare le variazioni rispetto all'immagine di riferimento ma al tempo stesso aumenta la possibilità di rilevare semplici disturbi come oggetti. È importante quindi testare questo valore in rapporto ad un'immagine di prova di cui si conosce il contenuto, per limitare il disturbo.
2. *Range_pos*: questa variabile è utilizzata per verificare se un oggetto è già presente nel bounding box e indica lo scostamento massimo entro il quale una coordinata è considerata identica ad un'altra; definisce quindi un intorno. Aumentando questo valore, è più semplice ritrovare l'oggetto nel frame successivo ma diminuisce la precisione rischiando di considerare due oggetti separati come uno solo; con un valore troppo piccolo si limita quest'ultima condizione ma aumenta la possibilità di confondere spostamenti di oggetti già esistenti con oggetti nuovi.
3. *Min_obj*: è la grandezza orizzontale e verticale minima perché un oggetto sia considerato tale. La variabile è stata introdotta per eliminare tutti quegli

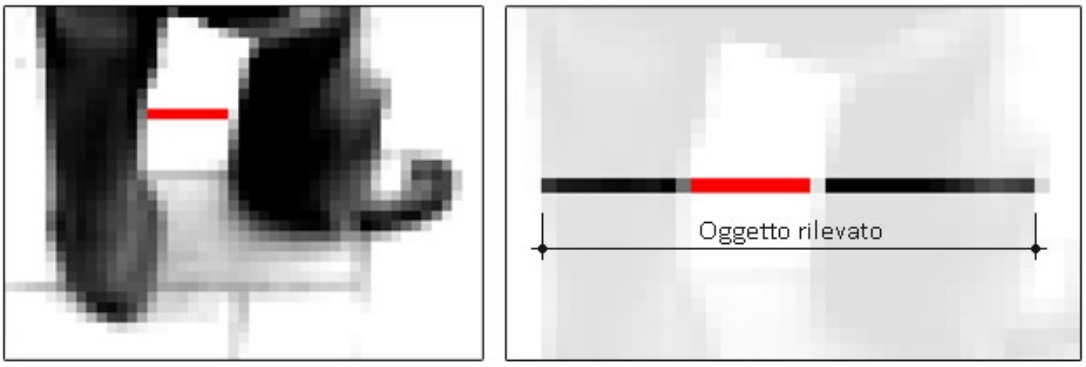


Figura 4 - L'area rossa indica il gap tra due sequenze di pixel diverse dallo sfondo

oggetti di dimensione non rilevante rispetto alla scena; per calibrare questo parametro è sufficiente analizzare la grandezza di un elemento conosciuto all'interno di un frame, come una mano.

4. *MaxGap*: nell'analisi delle righe, è la distanza massima oltre la quale due sequenze di pixel diversi dallo sfondo sono considerati separati (Figura 4). Questo serve a non segmentare in tanti elementi singole parti di un unico oggetto creando false rilevazioni, situazione che si verifica nel caso di forme non regolari. Pertanto se la distanza tra due rilevazioni su una riga è minore di *maxGap*, si costruisce un unico oggetto comprendente entrambe le sequenze, compreso lo spazio che le separa.

Tutte queste variabili devono essere calibrate *ad hoc* a seconda delle condizioni di ripresa per ottimizzare il risultato dell'algoritmo.

L'inizializzazione del programma continua con la dichiarazione di alcune funzioni pubbliche e del numero massimo di fruitori dell'installazione: lavorando con file audio infatti, non è possibile prevedere un numero illimitato di utenti per l'installazione, anche per ovvie questioni di spazio occupato, e devono essere presenti un numero sufficiente di sample da eseguire. Di seguito sono riportate le principali funzioni pubbliche e classi implementate.

## 3.2. FUNZIONI IMPLEMENTATE

### *3.2.1. Funzione isEqual(Color A, Color B)*

```
public static bool isEqual(Color a, Color b)
{
  if (b.R - range_color < a.R && a.R < b.R +range_color)
    return true;
  else return false;
}
```

**Codice 1 - Funzione isEqual**

*D*efinisce l'uguaglianza o meno di due pixel rispetto al loro colore (Codice ). Come già enunciato, le immagini sono in bianco e nero e quindi ogni pixel ha valori per i canali R,G e B uguali tra loro. Sfruttando questa proprietà è stato possibile ridurre ad un terzo il numero di confronti da effettuare per ogni pixel. La funzione riceve come parametri due oggetti di tipo *color* (contenente quattro membri: i tre canali RGB e il canale alpha della trasparenza) e verifica che il valore di B si trovi nell'intorno di A (definito come *a-range_color < b < a+range_color*). Restituisce vero in caso di uguaglianza: è quindi una sorta di overload di *operator=* a cui è stata aggiunta una tolleranza.

### *3.2.2. Funzione matchLine(obj A,obj B)*

```
public static bool matchLine(obj a, obj b)
{
if(
  Math.Min(a.fine.x, b.fine.x) - Math.Max(a.inizio.x, b.inizio.x) > 0
  && a.inizio.y <= (b.fine.y + range_pos)
  )
  return true;
else
  return false;
}
```

**Codice 2 - Funzione matchLine**

Per rilevare un oggetto in un frame, vengono ricercate in ogni riga dell'immagine tutte le sequenze di pixel adiacenti diversi dallo sfondo e inserite in una lista, una per ogni riga dell'immagine. Queste sequenze sono salvate attraverso la classe obj in qualità di oggetti degeneri, con coordinata di inizio e fine coincidente (oggetto grande un pixel), con valore di x uguale nelle coordinate di inizio e fine (riga orizzontale). Terminata la scansione delle righe si scansionano le liste così ottenute per combinare più sequenze in

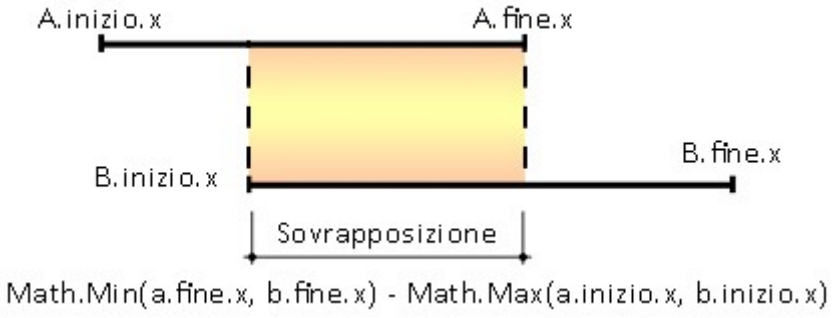


Figura 5 - Sovrapposizione

oggetti reali più grandi.

Questa funzione (Codice 2 ) quindi calcola la sovrapposizione orizzontale (Figura 5) di due sequenze di pixel adiacenti diversi dallo sfondo e restituisce vero se le condizioni sono verificate. Per fare ciò si sottrae al più piccolo valore x di fine sequenza il più grande valore x di inizio sequenza: il risultato indica di quanti pixel sono sovrapposti due oggetti; in caso di valore uguale a zero, due oggetti hanno un estremo coincidente, se minore di 0 non sono sovrapposti. È stata posta inoltre la condizione per cui l'oggetto B si trovi in un intorno verticale inferiore di A, per evitare di eseguire controlli su oggetti troppo distanti tra loro e quindi sicuramente differenti.

### 3.2.3. Funzione matchObj(obj A, obj B)

```
public static bool matchObj(obj a, obj b)
{
  if (b.centre.x - range_pos < a.centre.x &&
      a.centre.x < b.centre.x + range_pos &&
      b.centre.y - range_pos < a.centre.y &&
      a.centre.y < b.centre.y + range_pos )
    return true;
  else
    return false;
}
```

**Codice 1 - Funzione MatchObj**

La funzione matchObj (Codice 3) ha lo scopo di controllare che l'oggetto B si trovi nell'intorno dell'oggetto A; è utilizzata per determinare se un oggetto riscontrato al frame *n* sia già presente nel bounding box: in caso positivo si procede all'aggiornamento, mentre in caso negativo di provvede all'inserimento.

## 3.3. CLASSI IMPLEMENTATE

### *3.3.1. Classe Obj*

Un oggetto in un frame è un'area di immagine occupata, ossia un insieme di punti contigui diversi dallo sfondo. Poiché il corpo umano è mobile e non compatto, l'area occupata è di forma irregolare e frastagliata. Si è quindi resa necessario un sistema che potesse regolare tale forma comprendendola interamente e facilitando la definizione con una geometria uniforme a tutti gli oggetti. La classe obj definisce quindi il rettangolo che riquadra la forma dell'oggetto (bounding box): per fare ciò sono sufficienti due coordinate corrispondenti ai vertici superiore sinistro e inferiore destro, chiamati *inizio* e *fine*. La prima è costruita calcolando il minimo valore di x e y incontrato nella forma, la seconda determinando i massimi: ciò garantisce che non esistano porzioni di oggetto al di fuori dell'area così definita.

Per pilotare la componente audio e compattare il metodo di riferimento, è stata introdotta anche la coordinata *centre*, che definisce il punto centrale ed è costruita calcolando la media aritmetica dei membri inizio e fine. Per contrassegnare univocamente gli oggetti attivi nello stage e associarli ad una traccia audio è stato introdotto un id numerico: questo membro è *private* per non ammettere modifiche dirette che potrebbero essere dannose al comportamento del sistema. Poiché gli id collegano direttamente oggetti e tracce, sono contenuti in una struttura dati globale di tipo FIFO (First In First Out) implementata attraverso la classe *Queue* e sono tanti quanti i sample utilizzabili. Questo significa che i sample sono prelevati e resi nuovamente disponibili dopo la scomparsa dell'oggetto cui sono associati: l'utilizzo di una coda permette una gestione agevole di tale meccanismo perché evita di dover gestire la disponibilità dei sample attraverso un indice e inoltre consente un maggior ricambio nel loro utilizzo, favorendo minor ripetitività all'ambiente sonoro.

La classe obj prevede anche un membro "di servizio" utilizzato per segnalare l'eliminazione di un oggetto del bounding box, evitando di dover effettuare un ulteriore ciclo di scansione (*for*) per determinare quali oggetti devono essere eliminati perché non più presenti. È una variabile private di tipo bool chiamata *delete*.

La classe è stata implementata con numerosi costruttori (Codice 4) per andare incontro alle esigenze di programmazione che di volta in volta richiedevano metodi di definizione diversi a seconda delle variabili disponibili (sono quindi presenti costruttori con uno, due e quattro parametri). Allo stesso modo sono state implementate funzioni per la modifica di tutti o parte dei membri della classe.

```
public obj(coord a, coord b)
{
  this.inizio = new coord(a);
  this.fine = new coord(b);
  this.centre = new coord();
  this.centre.setMid(a, b);
  this.delete = false;
}
```

**Codice 2 - Costruttore della classe obj**

*setObj(obj A), setInizio(coord a) e setFine(coord a)* modificano le coordinate di *inizio* e *fine* calcolando al tempo stesso i valori della coordinata *centre*; *setDelete()* e *setId*() attraverso due overload consentono di impostare id o delete di un oggetto invocando la funzione ad un solo parametro oppure di restituire tali valori tramite la funzione senza parametri.

*Update(obj B)* (Codice 5) è la funzione di aggiornamento di un oggetto: poiché questo è definito per mezzo del quadrato che lo contiene, l'aggiornamento consiste nella modifica delle coordinate di inizio e fine, e automaticamente del centro. Si calcola il minimo valore x e y tra l'oggetto invocante e quello passato per parametro per aggiornare inizio, e il massimo per aggiornare fine; poi viene ricalcolato centre. Più che

aggiornare, questa funzione opera una fusione di due oggetti incrementando le dimensioni della riquadratura intorno all'oggetto.

```
public void update(obj b)
{
  this.inizio.setCoord(
  Math.Min(
    this.inizio.x,b.inizio.x),Math.Min(this.inizio.y,b.inizio.y));
  this.fine.setCoord(
    Math.Max(this.fine.x, b.fine.x), Math.Max(this.fine.y, b.fine.y));
  this.centre.setMid(this.inizio, this.fine);
}
```

**Codice 3 - Funzione Update, classe Obj**

La funzione *fakeObj(int)* controlla che un oggetto abbia le dimensioni minime indicate dalla variabile passata per parametro, che sarà tipicamente la costante min_obj. Questo valore va modificato in rapporto alla distanza e all'angolo di ripresa: diverse condizioni di zoom, determinano una variazione dell'area occupata dal medesimo oggetto; per stimare quale deve essere la minima grandezza richiesta, è opportuno riferirsi ad un area di cui si conosce la dimensione reale presente nell'immagine, come la grandezza di una testa, di una mano, di un oggetto, etc.

### *3.3.2. Classe Coord*

Definisce la posizione di un punto dell'immagine attraverso riga (y) e colonna (x). I membri della classe sono solo due interi x, y e alcune funzioni per il loro controllo. Oltre ai costruttori, sono state implementate le funzioni per la modifica come *setCoord()* e per calcolare il valore medio come *setMid(coord min, coord max).* Questa classe ha un numero limitato di funzioni in quanto è un semplice contenitore di valori numerici.

### *3.3.3. Classe track*

Definisce gli oggetti per la gestione dei file audio ed è il fulcro del secondo modulo di programmazione; proprio per la delicatezza del compito svolto, non possiede membri

public. Contiene al suo interno le coordinate dell'oggetto associato e dello stage, l'id della traccia da riprodurre che corrisponde anche all'identificativo dell'oggetto, i due oggetti della libreria irrKlang per la riproduzione: il motore sonoro *ISoundEngine* e l'oggetto *ISound*. Il primo contiene le informazioni dei parametri di riproduzione, file da riprodurre e fonte sonora; il secondo quelle del suono riprodotto: volume e pan, velocità di riproduzione, play – pause – stop. Poiché una traccia inizia la sua riproduzione all'ingresso di un soggetto nell'ambiente e la termina alla sua uscita, il costruttore (Codice 6) funge anche da abilitazione al play. Prende come parametri l'oggetto obj relativo all'ascoltatore a cui deve associarsi, l'oggetto obj relativo allo stage e l'oggetto *ISound* per la sincronizzazione.

```
public track(obj position, obj stg, ISound sync)
{
this.id = position.setId();
this.pos = new coord(position.centre);
this.stage = new obj(stg);
string sampleName = sample + this.id + ".wav";
this.engine = new ISoundEngine();
this.engine.SoundVolume = 1;
this.instrument =
    engine.Play2D(sampleName,true,true,StreamMode.AutoDetect,true);
this.instrument.Volume = setVol(position.centre);
this.instrument.Pan = setPan(position.centre);
this.instrument.PlayPosition = sync.PlayPosition;
this.instrument.Paused = false;
}
```

**Codice 4 - Costruttore della classe track**

Dapprima vengono salvate alcune informazioni utili come ID, posizione e stage nei membri interni ed è ricostruito il percorso del file da caricare. Quindi si istanzia il motore sonoro e infine viene creato l'oggetto *ISound* attraverso il metodo *Play2D* dell'oggetto ISoundEngine; la sincronizzazione avviene assegnando al membro *instrument* dell'oggetto invocante, il valore restituito da *PlayPosition*: in questo modo la riproduzione delle tracce si troverà allo stesso millisecondo.

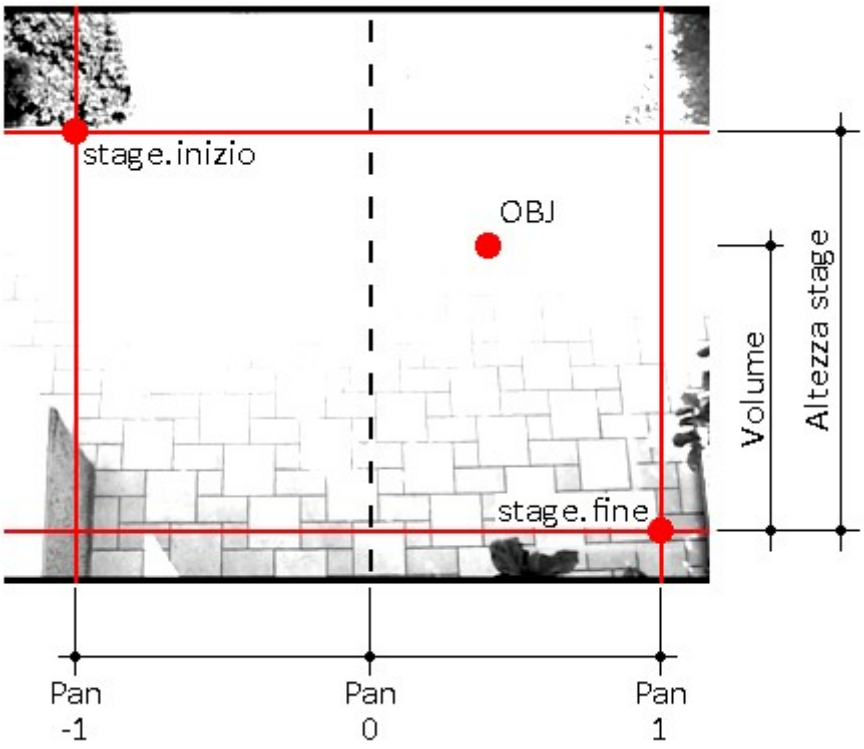


Figura 6 - Descrizione dello stage

Accanto alla creazione di una nuova traccia, si è dovuto gestire anche la sua modifica al variare della posizione degli oggetti nel tempo: questo processo non può essere effettuato assegnando direttamente i valori di x e y ai parametri della traccia, in quanto operano su scale differenti (Figura 6), l'uno con minimi e massimi definiti dallo stage, l'altro con quelli della classe irrKlang (0 e 1 per il volume, -1 e 1 per il pan). Sono state quindi implementate due funzioni per portare alla scala corretta le coordinate.

### 3.3.4. Funzione setVol(coord obj)

```
public float setVol(coord obj)
{
return
   (float)(this.stage.fine.y - obj.y) /
   (float)(this.stage.fine.y - this.stage.inizio.y);
}
```

Codice 5 – Funzione setVol, classe Track

Opera sui valori y per determinare la distanza verticale di un oggetto dall'inizio dello stage (che può non coincidere con l'inizio dell'immagine) e metterla in rapporto alla dimensione totale dell'area di lavoro (Codice 7). Si ottiene in questo modo un valore (*float*) compreso tra 0 e 1, in grado di rappresentare il volume.

### 3.3.5. Funzione setPan(coord obj)

```
public float setPan(coord obj)
{
return
   (2 * (float)(obj.x - stage.inizio.x) /
   (float)(stage.fine.x - stage.inizio.x)) - 1;
}
```

Codice 6 - Funzione setPan, classe Track

Per calcolare il valore del pan (Codice 8), si mette in rapporto tra loro la distanza x dell'oggetto dall'inizio dello stage e il range massimo del parametro pan, cioè 2 (da -1 a 1). Al risultato ottenuto è sottratto -1 per impostare correttamente il valore del pan: -1 corrisponde al canale sinistro, 0 pan centrale e 1 canale destro.

## 3.4. CODICE DEL PROGRAMMA PRINCIPALE

L'esecuzione del programma ha inizio con la definizione e l'inizializzazione di tutte le variabili necessarie al corretto funzionamento del sistema. Sono state definite le strutture dati per contenere ed elaborare gli oggetti rilevati: *List<List<obj>> righe* è il contenitore di tutte le sequenze di pixel diverse dallo sfondo divise per riga ed è utilizzato per confrontare e costruire gli oggetti. Sempre di classe *List* sono le due strutture dati per gli oggetti: *bbox_temp* contiene tutti gli oggetti rilevati nel frame corrente, compresi gli oggetti degeneri e quelli troppo piccoli che verranno eliminati, mentre *bbox_main* contiene gli oggetti effettivamente utilizzati. La definizione dei sample da utilizzare è affidata alla coda *instrument*, implementata attraverso la classe *Queue<int>*, una struttura dati di tipo FIFO (First In First Out): gli elementi sono inseriti in coda tramite la funzione *Enqueue(int)* ed estratti dalla testa con la funzione *Dequeue()*. In questo modo all'ingresso di una nuova persona un id è prelevato dalla coda e assegnato ad un oggetto, mentre all'uscita viene nuovamente inserito nel fondo della coda pronto per essere riutilizzato, garantendo in questo modo un ricambio dei sample riprodotti rispetto all'utilizzo di altre strutture dati, come le pile o gli array. Infine sono definiti la cartella contenente le immagini da processare, il file dell'immagine di riferimento con la quale effettuare i confronti e alcune variabili di servizio. In ultima istanza prima di iniziare l'analisi delle immagini, si attiva la riproduzione della traccia di sincronizzazione: questa fa riferimento ad un sample vuoto, ma della stessa lunghezza degli altri sample. Utilizzando il parametro PlayPosition, è possibile sapere a quale millisecondo si trova la riproduzione di *sync* e di conseguenza far partire la riproduzione di un nuovo sample dal medesimo punto.

Ha quindi inizio l'analisi dei frame, con due cicli iterativi *for* che scandiscono rispettivamente righe e colonne per confrontare tutti i pixel del frame con l'immagine di riferimento: l'obiettivo di questa prima fase è quello di determinare per ogni riga tutte le sequenze di pixel diversi dallo sfondo consecutivi. Si seleziona quindi la prima riga

dell'immagine e si confronta il singolo pixel con l'immagine di riferimento: la differenza indica un punto nell'ambiente occupato da un oggetto, ma non è in grado di fornire ulteriori informazioni. Si considera allora anche la condizione del pixel precedente e si determinano configurazioni standard in grado di definire le sequenze di pixel su una riga, come riportato in Tabella 1.

| **Pixel *n-1*** | **Pixel *n*** | **Descrizione** | **Azione** |
|---|---|---|---|
| uguale | uguale | Pixel non appartenente all'oggetto | Conteggio *gap* |
| diverso | uguale | L'oggetto è terminato al pixel precedente o inizia uno spazio vuoto nell'oggetto (*gap*) | Conteggio *gap* |
| uguale | diverso | Inizio di un nuovo oggetto o fine un *gap* | Inserimento nuovo oggetto o aggiornamento coordinata *fine* |
| diverso | diverso | Continuazione di un oggetto | Aggiornamento coordinata *fine* dell'oggetto |

**Tabella 1 - Configurazioni standard per definire le sequenze di pixel in una riga dell'immagine**

Nel caso di uguaglianza di due pixel consecutivi o di uguaglianza per il pixel corrente e disuguaglianza del precedente, non si effettua nessuna operazione sugli oggetti, ma si calcola la lunghezza della sequenza di pixel uguali allo sfondo; tali sequenze infatti potrebbero comunque appartenere ad un oggetto in qualità di spazio tra due porzioni del medesimo (si veda Figura 4) a patto che non superino il valore massimo definito dalla variabile globale *maxGap*. Il caso di disuguaglianza si configura un aggiornamento della coordinata fine dell'oggetto, mentre la disuguaglianza del pixel corrente con l'uguaglianza del precedente implica l'inserimento di un nuovo oggetto nel caso in cui il valore di *gap* sia maggiore del massimo consentito dalla variabile globale *maxGap* altrimenti si procede con l'aggiornamento della coordinata *fine* dell'oggetto. Di seguito è riportato il diagramma di flusso (Figura 7).

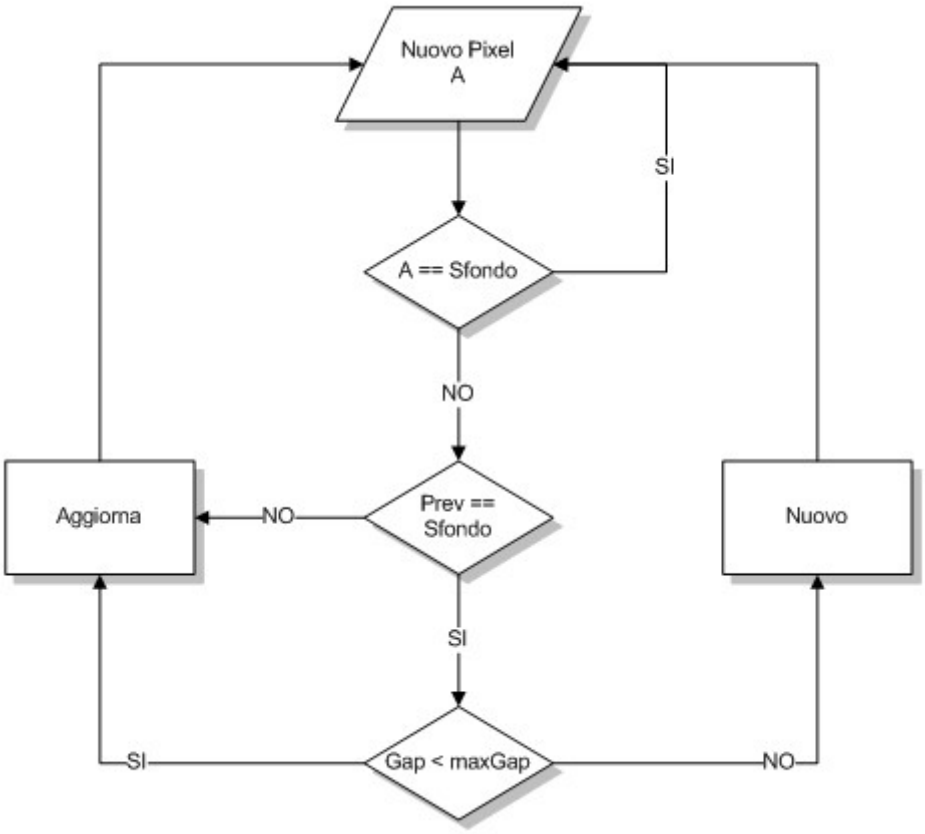


Figura 7 - Diagramma di flusso dell'analisi dei pixel

A questo punto il ciclo for relativo alla scansione dei pixel ha terminato la sua esecuzione e il prodotto della scansione di una riga è una lista contenente tutti gli oggetti riconosciuti (si ricorda che in questa fase si parla ancora di oggetti degeneri, in quanto sequenze di pixel) che devono ora essere inseriti nel *bbox_temp* per costruire gli oggetti finali. In fase di inserimento (Figura 8), si determina se nel *bbox_temp* è già presente un oggetto con coordinate attigue a quelle che si sta per inserire attraverso la funzione *matchLine()* in grado di valutare la sovrapposizione orizzontale di due elementi. In caso di esito positivo, si aggiorna la coordinata di fine oggetto, in quanto l'elemento inserito aggiungerà una porzione inferiore di riquadro all'oggetto poiché la scansione dell'immagine avviene dal margine superiore a quello inferiore. In caso contrario si inserisce un nuovo oggetto. La funzione di *bbox_temp* è duplice: ospita tutti gli oggetti

riscontrati in un frame, compresi quelli degeneri che andranno eliminati, consente il confronto con la configurazione nel frame precedente, garantendo la possibilità di

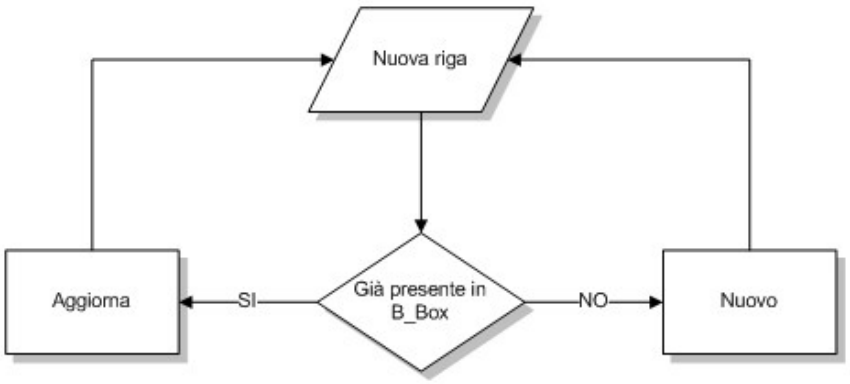


Figura 8 - Diagramma di flusso della gestione delle righe

aggiornare le posizioni di oggetti già presenti.

Terminata anche la scansione di tutte le righe di un frame, si scorre la lista *bbox_temp* per eliminare tutti gli oggetti degeneri e inserire gli altri nel *bbox_main*. Anche in questo contesto (Figura 9), l'inserimento è vincolato al controllo sulla presenza di un oggetto con coordinate simili, attraverso la funzione *matchObj*, per determinare un operazione di aggiornamento piuttosto che una nuova immissione.

In ultima istanza si eliminano gli oggetti ormai non più presenti nel frame: si sfoglia *bbox_main* in cerca di tutti gli oggetti con membri *delete* uguale a *true*. L'esecuzione di questo ciclo è vincolata ad un controllo sul numero di istanze presenti nel *bbox_main* e nel *bbox_temp*: qualora il numero di elementi presente nel primo fosse maggiore a quelli presenti nel secondo, ci sarebbero sicuramente oggetti rimasti salvati dall'analisi dei frame precedenti da cancellare.

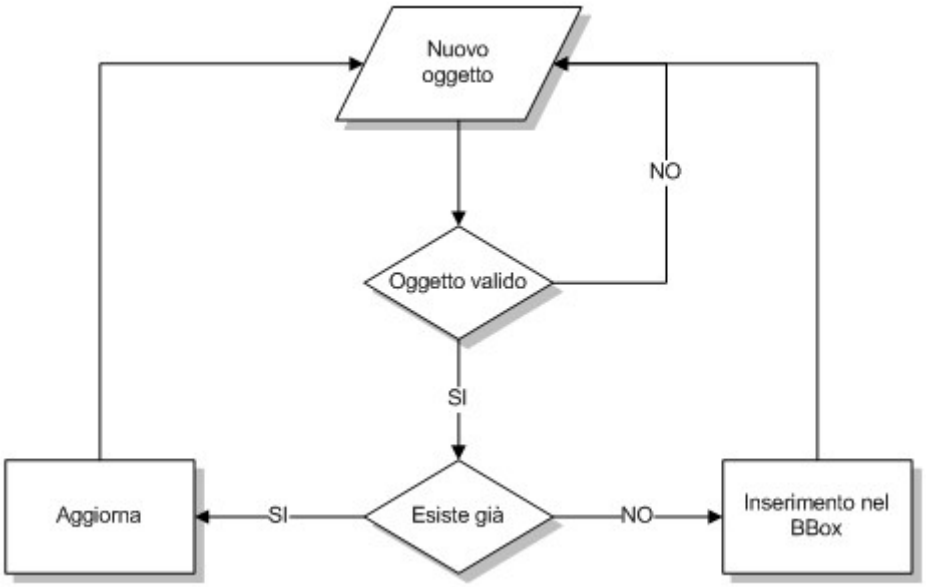


Figura 9 - Diagramma di flusso della gestione del Bounding Box

La gestione delle tracce audio si integra all'interno delle operazioni di inserimento, modifica ed eliminazione degli oggetti nel *bbox_main*. In corrispondenza di ogni inserimento, si richiama il costruttore della classe track per iniziare la riproduzione relativa al nuovo elemento. Dapprima si estrae dalla coda *instrument* un nuovo id da assegnare all'oggetto e alla traccia: tale valore è anche l'indice dell'array *ascoltatori* in cui verrà salvata l'istanza di track relativa all'oggetto. In seguito si imposta il membro *delete* uguale a *false*, per non consentire la cancellazione; infine l'oggetto è inserito nel *bbox_main* e si istanzia un nuovo oggetto della classe track attraverso il costruttore (si veda Codice 6) che dà inizio alla riproduzione.

La modifica di tracce già in esecuzione è invece gestita dalla funzione *setPos* che aggiorna i valori di volume e pan; per identificare l'istanza da modificare è sufficiente richiamare l'elemento dell'array *ascoltatori* alla posizione indicata dall'id dell'oggetto da modificare.

Infine l'eliminazione di una traccia è contestuale alla scansione del *bbox_main* per determinare gli oggetti da eliminare: attraverso il distruttore della classe track si interrompe la riproduzione e si libera lo spazio occupato nel *garbage collector* insieme al reinserimento dell'id nella coda e all'eliminazione dell'istanza dal *bbox_main*.

# 4. CONCLUSIONI

Il progetto si può considerare globalmente riuscito in quanto ha raggiunto tutti gli obiettivi prefissati: il sistema infatti è in grado di identificare correttamente le posizioni di alcune persone all'interno di uno streaming video, seguirne i movimenti e provvedere alla sonorizzazione dell'ambiente. Considerando che non sono stati rilevati errori durante tutti i test eseguiti, questo tipo di approccio mostra la sua robustezza e apre la strada ad un possibile sviluppo effettivo, soprattutto nel campo delle installazioni multimediali, che hanno caratteristiche ambientali comuni a quelle utilizzate nel progetto: ambienti medio – grandi con un affluenza di persone all'interno limitata o che sia comunque stabilita a priori. L'idea di poter raffigurare nel panorama sonoro la posizione spaziale, risulta molto suggestiva e coinvolge maggiormente l'utente in ciò che sta osservando, spingendolo ad interagire per influenzare direttamente la sua percezione sonora. Questa circostanza porta in primo piano l'esperienza sonora anche in un ambito, come quello dell'immagine e dell'arte visiva, che ha ormai prevaricato nella società moderna l'attenzione nei confronti dell'elemento musicale. Allo stesso tempo si accentua l'attenzione verso il proprio modo di porsi nei confronti dell'ambiente in cui ci si trova, valorizzando l'idea di costruirsi un proprio spazio in rapporto a ciò che ci circonda.

Sono state eseguite alcune sessioni di test con video registrati appositamente per testare il comportamento del sistema in presenza di un numero di persone variabile da uno a tre; dapprima è stato chiesto ad un solo individuo di eseguire alcuni percorsi lungo gli assi verticale ed orizzontale per testare il comportamento di base del sistema: questo tipo di prova ha permesso di valutare il corretto funzionamento del sistema e la corretta configurazione dei parametri in rapporto alle posizioni riscontrate nei frame. Successivamente si è testato il sistema con più persone cui è stato chiesto inizialmente di eseguire movimenti prestabiliti e poi di muoversi liberamente nell'area di lavoro per verificare il funzionamento in condizioni di normale utilizzo.

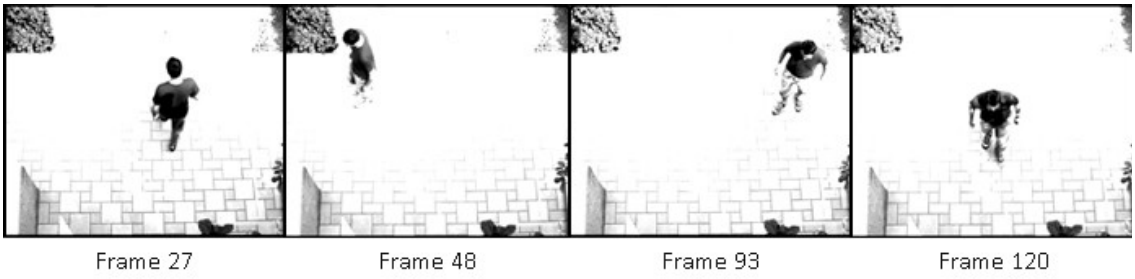


Figura 11 - Alcuni frame della prima sequenza di test

Il primo test (Figura 11) ha previsto un percorso con inizio al centro della scena dal margine inferiore dello stage verso quello superiore, per mostrare la variazione del volume, cui è seguito lo spostamento prima verso il margine sinistro e poi verso quello destro per le variazioni al pan; tornato al centro, il soggetto è uscito di scena dal margine inferiore. La sonorizzazione corrispondente ha seguito perfettamente gli spostamenti mostrando dapprima un aumento di volume, lo spostamento del pan verso sinistra e poi verso destra a volume costante, per poi tornare centrale a terminare con un *fade out*. Poiché il test prevedeva un solo soggetto, è stato utilizzato solo un file audio e precisamente il primo inserito nella coda *instrument*. Questa prima sequenza non ha mostrato errori di rilevazione e posizionamento degli oggetti, attestando la sostanziale stabilità del programma.

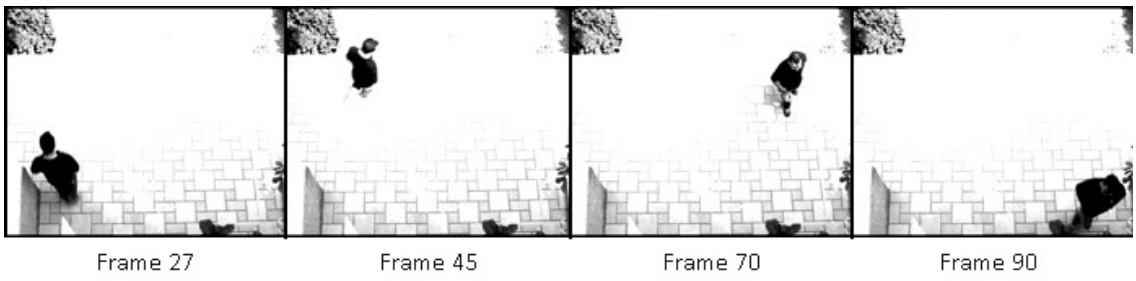


Figura 10 - Alcuni frame della seconda sequenza di test

Il secondo video di test (Figura 10) ha visto coinvolto ancora un solo individuo, che ha iniziato il suo percorso dal margine inferiore sinistro, salendo verticalmente verso

quello superiore; è seguito uno spostamento orizzontale verso destra per poi uscire di scena dal margine inferiore destro. Questa sessione è stata utilizzata per testare l'inizio della sonorizzazione di un elemento da un punto che avesse un pan diverso dalla posizione di default (centro). Anche in questa sessione di test il programma ha risposto correttamente con una sonorizzazione che è iniziata con un *fade in* e pan tutto a sinistra, per poi spostarsi tutto a destra a volume costante e terminare con un *fade out*. Anche questa sessione di test non ha riscontrato nessun errore in fase di esecuzione.

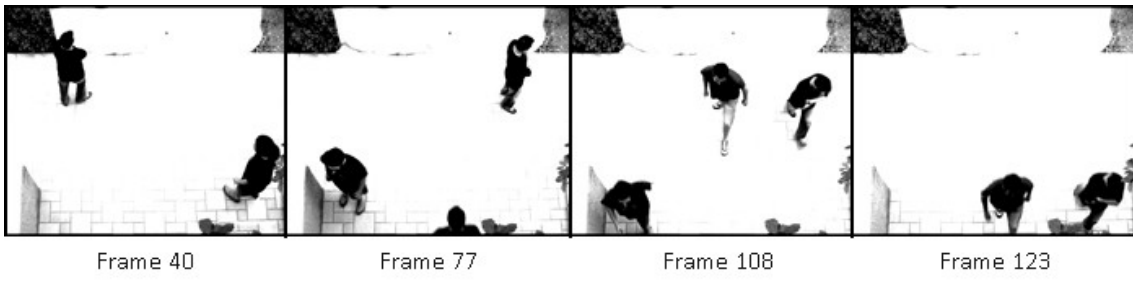


**Figura 12 - Alcuni frame della terza sequenza di test**

La terza fase di test (Figura 12) ha utilizzato tre soggetti in scena con movimenti concordati a priori, per mettere alla prova il sistema nella gestione parallela di più tracce. Due persone sono entrate nell'area di lavoro contemporaneamente dal margine inferiore, uno a destra e uno a sinistra e raggiungendo due punti verticali differenti; il sistema ha risposto attivando la riproduzione di due tracce con un *fade in* iniziale e pan uno a destra e uno a sinistra. Successivamente i soggetti si sono spostati verso il margine opposto, provocando la variazione contemporanea dei rispettivi pan che in sostanza si sono scambiati l'uno con l'altro; infine il terzo elemento è entrato in scena dal centro del margine inferiore provocando il *fade in* di una nuova traccia con pan centrale. Per concludere, tutti gli occupanti sono usciti dall'area di lavoro, sempre dal margine inferiore dell'area di lavoro. Durante l'esecuzione di questo test è stato riscontrato un errore relativo ad una falsa rilevazione di un oggetto ai frame 75 e 76: questo malfunzionamento è stato causato da una scorretta impostazione dei parametri di rilevamento da impostare

ad inizio programma. Il programma infatti ha classificato come oggetto l'ombra proiettata di un elemento effettivamente presente nell'area di lavoro. Per ovviare al problema è stata calibrata nuovamente la grandezza dello stage poiché l'area che ha generato l'errore si trovava al di fuori dell'area utile. Questo esempio mette in luce l'importanza fondamentale del setup delle variabili a seconda della scena e delle forme da trattare: condizioni particolari di luce e contrasto, aree definite, variazioni nel colore possono causare difficoltà nel funzionamento. Per questo motivo si è cercato di mantenere il più possibile costanti le condizioni di ripresa, correggendo eventuali variazioni di luce causate dal fatto che si è stati costretti ad acquisire immagini all'esterno: la luce artificiale aiuta sicuramente a mantenere più stabile l'ambiente. Queste considerazioni risultano comunque in linea con gli obiettivi iniziali, che prevedevano lo studio di un prototipo destinato principalmente ad installazioni artistiche multimediali.

In ultima analisi sono state sperimentate le condizioni di utilizzo generiche, con i soggetti interessati che hanno avuto la possibilità di muoversi liberamente nell'ambiente. Il sistema ha risposto egregiamente, calcolando correttamente le posizioni e generando la sonorizzazione corretta: non sono stati rilevati errori in questa fase.

Alla luce dei test effettuati e qui presentati, si può dire che il progetto ha risposto egregiamente alle aspettative iniziali, pur tenendo conto delle restrizioni imposte inizialmente. La scelta di non considerare il fenomeno dell'occlusione può sembrare molto restrittiva, ma se si pensa ad ambienti medio – grandi la sovrapposizione di più elementi nelle riprese video risulta essere un caso particolare. Anche il posizionamento della videocamera può certamente aiutare a limitare questo fenomeno: una ripresa perfettamente ortogonale dal centro dell'area considerata evidenzia meglio i singoli oggetti ed elimina anche le deformazioni dovute all'angolo di ripresa, diminuendo le situazioni di mascheramento. Allo stesso modo le restrizioni relative al numero massimo di tracce, dovuta all'utilizzo di file audio, può sembrare un ostacolo, ma bisogna considerare che l'area di lavoro scelta avrà sempre un limite di abitabilità; comunque

questa situazione può essere facilmente superata selezionando un numero molto maggiore di campioni riproducibili.

La scelta di lavorare su un sistema *off-line* è stata vincolata ad alcune difficoltà tecniche quali il tempo di esecuzione dell'algoritmo per ogni frame e la disponibilità di una videocamera che permettesse l'estrazione diretta dei frame tramite una libreria. Il tempo di esecuzione dell'algoritmo è di circa 180ms per ogni frame su un notebook con processore Intel Centrino da 2,00 GHz con 2 GB di memoria RAM, che non consente di gestire in tempo reale il frame rate del video (25 frame secondo). In alternativa sarebbe stato possibile diminuire il frame rate o analizzare un frame ogni cinque ma avrebbe comportato, come hanno dimostrato alcuni test, un numero maggiore di false rilevazioni per l'impossibilità di avere un passaggio fluido tra una posizione e la successiva di un oggetto. Una macchina più potente come la maggior parte di quelle attualmente in commercio con una GPU dedicata [13] per l'analisi dei frame migliorerebbe sicuramente le prestazioni consentendo l'approccio real-time. Da non trascurare anche la necessità di elaborare seppur in maniera minima le immagini per aumentare il contrasto e mettere quindi in risalto gli oggetti da tracciare; questo processo è stato eseguito dopo aver registrato i video di test ma nel caso di un sistema real-time dovrebbe essere gestito direttamente dalla videocamera o dal sistema di estrazione dei frame: in quest'ultimo caso infatti bisogna considerare i tempi di esecuzione di tale modifica.

Sempre relativamente alle attrezzature da utilizzare, si sottolinea la necessità di un ottica per la videocamera tale da consentire l'inquadratura dell'intera area ad un altezza che sia consentita da uno spazio espositivo, senza la necessità di dover collocare l'apparecchio ad altezze troppo elevate. Gli obiettivi grandangolari rispondono a queste esigenze già ad un'altezza di circa 3 metri, contro i 7 metri utilizzati nel prototipo che non consentirebbero facilmente l'installazione in un ambiente chiuso se non scegliendo una location specifica.

# 5. SCENARI DI SVILUPPO

Considerando l'esito positivo delle sessioni di test sul prototipo, il passo successivo è rappresentato dalla realizzazione del sistema in tempo reale, per testare sul campo l'interazione uomo ambiente nella costruzione del panorama sonoro, predisponendo un modulo di estrazione e modifica dei frame e un impianto di diffusione audio. Per fornire un quadro di risposta completo a tutte le possibili situazioni, è opportuno provvedere all'implementazione di un sistema di tracking che gestisca anche i mascheramenti: questa configurazione è sufficiente a garantire la messa in opera in un ambiente reale, come un'installazione multimediale, che altrimenti sarebbe vincolata al rispetto di alcune restrizioni per garantire il corretto funzionamento.

Oltre a queste modifiche strutturali, è auspicabile l'inserimento di un'interfaccia utente che consenta di visualizzare filmato e bounding box identificati per monitorare l'andamento dell'esecuzione. Questo front-end dovrebbe consentire di impostare direttamente anche le variabili per il riconoscimento degli oggetti e per la grandezza dell'area di lavoro, oltre che avviare ed arrestare il sistema, consentendo di testare ulteriormente il programma per colmare più carenze possibili. Per aumentare le possibilità di interazione e rendere più complesso il panorama sonoro, si potrebbe aumentare il numero di azioni consentite agli utenti da associare ad altrettanti eventi sonori, come descritto in precedenza nella descrizione del modello: le posizioni reciproche degli utenti potrebbero essere mappate sull'abilitazione e variazione di effetti sonori da applicare alle tracce; all'ingresso e all'uscita potrebbero corrispondere effetti sonori prestabiliti; l'interazione con eventuali oggetti potrebbe attivare la riproduzione di ulteriori file audio. Le modifiche di questo tipo sono pressoché illimitate e legate all'estro e alle esigenze degli ideatori dell'installazione, con cui ci deve essere una stretta collaborazione per chiarire gli obiettivi e gli effetti che si vogliono ottenere. Anche le limitazioni al numero di utenti potrebbero essere superate attraverso l'utilizzo di

sequenze MIDI al posto di file audio: in questo caso si tratterebbe di creare un numero prefissato di pattern, melodici e ritmici, da poter applicare a diversi suoni sintetizzati. Considerando che i banchi MIDI sono composti da 128 suoni differenti, si sarebbe in grado di gestire un numero sostanzialmente illimitato di utenti, tenendo conto delle esigenze di abitabilità dell'ambiente.

Oltre alle modifiche relative al miglioramento delle prestazioni, si può prospettare un'ulteriore applicazione nel campo didattico, per far avvicinare alla percezione sonora bambini e ragazzi: l'utilizzo di un mezzo molto fisico come la propria figura può aiutare a porre l'attenzione su un aspetto in realtà molto più etereo e raffinato come la percezione sonora, che soprattutto oggi viene soffocata da un ascolto improntato alla quantità più che alla qualità [14].

Colgo l'occasione per ringraziare il Dott. Adriano Baratè che mi ha seguito con costanza, disponibilità e puntualità nel corso dell'elaborazione del progetto, ringrazio inoltre il Prof. Luca Andrea Ludovico e tutto il LIM per la professionalità dimostratami.

Un ringraziamento particolare anche al Dott. Pietro Cerri del laboratorio VisLab (Artificial Vision and Intelligent Systems Laboratory) del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma, per il supporto fornitomi sulla visione artificiale.

# APPENDICE

# 6. CODICE DEL PROGRAMMA

## 6.5. PROGRAM.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using IrrKlang;

namespace DetectImage2
{
 class Program
 {
  // Definizione parametri dell'area di lavoro dell'immagine
  public const int minX = 45, maxX = 326, minY = 30, maxY = 260;

  // Definizione parametri per discriminare gli oggetti
  public const int range_color = 60, range_pos = 23, minObj = 15, maxGap
= 20;

  // Definizione parametri delle tracce
  const int MAX_TRACK = 3;
  public static track[] ascoltatori = new track[MAX_TRACK];
  /// <summary>
  /// Determina se i colori a e b sono uguali, con una tolleranza
definita da "range_color"
  /// </summary>
  /// <param name="a">Colore a</param>
  /// <param name="b">Colore b</param>
  /// <returns></returns>
  public static bool isEqual(Color a, Color b)
  {
   if (b.R - range_color < a.R && a.R < b.R + range_color)
    return true;
   else return false;
  }
  /// <summary>
  /// Dati due oggetti, valuta se sono sovrapposti orizzontalmente e
compresi nel range definito da range_pos verticalmente
  /// </summary>
  /// <param name="a">Oggetto a</param>
  /// <param name="b">Oggetto b</param>
  /// <returns>Restituisce vero se gli oggetti si sovrappongono</returns>
  public static bool matchLine(obj a, obj b)
  {
   if (
     (Math.Min(a.fine.x, b.fine.x) - Math.Max(a.inizio.x, b.inizio.x)) >
0 &&
      a.inizio.y <= (b.fine.y + range_pos)
     )
    return true;
```

```
    else
     return false;
   }
   /// <summary>
   /// Determina se due oggetti coincidono, con una tolleranza definita da
range_pos
   /// </summary>
   /// <param name="a">Oggetto a</param>
   /// <param name="b">Oggetto b</param>
   /// <returns>Restituisce vero se coincidono</returns>
   public static bool matchObj(obj a, obj b)
   {
    if (b.centre.x - range_pos < a.centre.x && a.centre.x < b.centre.x +
range_pos &&
        b.centre.y - range_pos < a.centre.y && a.centre.y < b.centre.y +
range_pos)
     return true;
    else return false;
   }

   // Funzioni di servizio
   public static void Print(obj a)
   {
    a.print();
   }
   public static void Print(List<obj> a)
   {
    Console.WriteLine("=====");
    Console.WriteLine("Riga " + a[1].inizio.y);
    a.ForEach(Print);
   }

   static void Main(string[] args)
   {
    List<List<obj>> righe = new List<List<obj>>();
    List<obj> bbox_temp = new List<obj>();
    List<obj> bbox_main = new List<obj>();
    Queue<int> instrument = new Queue<int>();
    instrument.Enqueue(0);
    instrument.Enqueue(1);
    instrument.Enqueue(2);
    obj stage = new obj(minX, minY, maxX, maxY);
    int idIndex = 0;

    //Variabili per la gestione delle righe
    const string IMG = "D:/_TESI/BMP/C/";
    Bitmap sfondo = new Bitmap(IMG + "Frame1.bmp");
    Bitmap IMG_name;
    bool found = false;
    bool matched = false;
    bool first = true;
    bool firstB = true;
    int gap = 0;

    // Attivazione della traccia sync
    ISoundEngine syncEngine = new ISoundEngine();
    syncEngine.SoundVolume = 0;
```

```
   ISound sync = syncEngine.Play2D("D:/_TESI/sample/sync.wav", true,
false, StreamMode.AutoDetect, true);

   for (int frame = 2; frame < 133; frame++)
   {
    IMG_name = new Bitmap(IMG + "Frame" + frame + ".bmp");
    for (int y = minY; y < maxY; y++)
    {
     righe.Add(new List<obj>());
     for (int x = minX; x < maxX; x++)
     {
      switch (isEqual(IMG_name.GetPixel(x, y), sfondo.GetPixel(x, y)))
      {
       // Pixel attuale vuoto: punto uguale allo sfondo
       case true:
        {
         gap++;
         found = false;
         break;
        }
       // Pixel attuale occupato: punto diverso dallo sfondo
       case false:
        {
         // Pixel precedente vuoto e pixel attuale occupato
         if (!found)
         {
          if (gap < maxGap && righe.Last().Count > 0)
righe.Last().Last().setFine(x, y);
          else righe.Last().Add(new obj(x, y, x, y));
         }
         // Pixel precedente occupato e pixel attuale occupato
         else
         {
          righe.Last().Last().setFine(x, y);
         }
         found = true;
         gap = 0;
         break;
        }
      }
     }
     // Inserimento della riga nel bbox_temp
     if (first)
     {
      for (int i = 0; i < righe.Last().Count; i++)
      {
       bbox_temp.Add(righe.Last()[i]);
      }
      first = false;
     }
     else
     {
      for (int i = 0; i < righe.Last().Count; i++)
      {
       for (int v = 0; v < bbox_temp.Count; v++)
       {
        if (matchLine(righe.Last()[i], bbox_temp[v]))
```

```
    {
     bbox_temp[v].update(righe.Last()[i]);
     matched = true;
    }
    if (matched)
     break;
   }
   if (!matched)
   {
    bbox_temp.Add(righe.Last()[i]);
   }
   matched = false;
  }
 }
}
// Eliminazione degli oggetti degeneri e aggiornamento del bbox_main
if (firstB)
{
 for (int q = 0; q < bbox_temp.Count; q++)
 {
  if (bbox_temp[q].fakeObj(minObj))
  {
   bbox_temp.Remove(bbox_temp[q]);
   q--;
  }
  else
  {
   idIndex = instrument.Dequeue();
   bbox_temp[q].setId(idIndex);
   bbox_temp[q].setDelete(false);
   bbox_main.Add(bbox_temp[q]);
   ascoltatori[idIndex] = new track(bbox_main.Last(), stage, sync);
  }
 }
 firstB = false;
}
else
{
 for (int q = 0; q < bbox_temp.Count; q++)
 {
  if (bbox_temp[q].fakeObj(minObj))
  {
   bbox_temp.Remove(bbox_temp[q]);
   q--;
  }
  else
  {
   for (int i = 0; i < bbox_main.Count; i++)
   {
    if (matchObj(bbox_temp[q], bbox_main[i]))
    {
     bbox_main[i].setObj(bbox_temp[q]);
     bbox_main[i].setDelete(false);
     ascoltatori[bbox_main[i].setId()].setPos(bbox_main[i].centre);
     matched = true;
    }
    if (matched) break;
```

```
        }
        if (!matched)
        {
         idIndex = instrument.Dequeue();
         bbox_temp[q].setId(idIndex);
         bbox_temp[q].setDelete(false);
         bbox_main.Add(bbox_temp[q]);
         ascoltatori[idIndex] = new track(bbox_main.Last(), stage, sync);
        }
        matched = false;
       }
      }
      if (bbox_main.Count > bbox_temp.Count)
      {
       for (int i = 0; i < bbox_main.Count; i++)
       {
        if (bbox_main[i].setDelete())
        {
         ascoltatori[bbox_main[i].setId()].Dispose();
         instrument.Enqueue(bbox_main[i].setId());
         bbox_main.RemoveAt(i);
         i--;
        }
        else bbox_main[i].setDelete(true);
       }
      }
     }
     bbox_temp.Clear();
     righe.Clear();
     IMG_name.Dispose();
     first = true;
     matched = false;
    }
    Console.Read();
   }
  }
}
```

### 6.6. CLASS1.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using IrrKlang;

namespace DetectImage2
{
 /// <summary>
 /// Definisce la posizione di un punto attraverso i valori X e Y
 /// </summary>
 class coord
 {
  /// <summary>
  /// Coordinata X
  /// </summary>
  public int x;
  /// <summary>
  /// Coordinata Y
  /// </summary>
  public int y;
  /// <summary>
  /// Costruttore di default: imposta i valori x e y a zero
  /// </summary>
  public coord()
  {
   this.x = 0;
   this.y = 0;
  }
  /// <summary>
  /// Costruttore: assegna ad x e y i valori della coordinata indicata
  /// </summary>
  /// <param name="a">Coordinata con i valori da assegnare</param>
  public coord(coord a)
  {
   this.x = a.x;
   this.y = a.y;
  }
  /// <summary>
  /// Costruttore: imposta x e y con i valori indicati
  /// </summary>
  /// <param name="x_c">Valore di x</param>
  /// <param name="y_c">Valore di y</param>
  public coord(int x_c, int y_c)
  {
   this.x = x_c;
   this.y = y_c;
  }
  /// <summary>
  /// Assegna ad x e y i valori indicati
  /// </summary>
  /// <param name="x_c">Valore di x</param>
  /// <param name="y_c">Valore di y</param>
  public void setCoord(int x_c, int y_c)
```

```
  {
   this.x = x_c;
   this.y = y_c;
  }
  /// <summary>
  /// Modifica i valori di x e y con quelli della coordinata indicata
  /// </summary>
  /// <param name="a">Coordinata con i valori da assegnare</param>
  public void setCoord(coord a)
  {
   this.x = a.x;
   this.y = a.y;
  }
  /// <summary>
  /// Imposta il valore medio delle coordinate indicate
  /// </summary>
  /// <param name="min">Coordinata minimo</param>
  /// <param name="max">Coordinata massimo</param>
  public void setMid(coord min, coord max)
  {
   this.x = (min.x + max.x) / 2;
   this.y = (min.y + max.y) / 2;
  }
  /// <summary>
  /// Stampa i valori di x e y
  /// </summary>
  public void print()
  {
   Console.WriteLine(this.x + " - " + this.y);
  }
 }
 /// <summary>
 /// Definisce un oggetto attraverso il rettangolo che lo contiene.
 /// </summary>
 class obj
 {
  /// <summary>
  /// Coordinata del vertice superiore a sinistra del rettangolo
contenente l'oggetto.
  /// Contiene i minimi x e y dell'oggetto.
  /// </summary>
  public coord inizio;
  /// <summary>
  /// Coordinata del vertice inferiore a destra del rettangolo contenente
l'oggetto.
  /// Contiene i massimi x e y dell'oggetto.
  /// </summary>
  public coord fine;
  /// <summary>
  /// Coordinata del punto medio del rettangolo contenente l'oggetto.
  /// Contiene i valori medi x e y dell'oggetto.
  /// </summary>
  public coord centre;
  /// <summary>
  /// Identificativo dell'oggetto
  /// </summary>
  private int id;
```

```
  /// <summary>
  /// Flag che determina l'eliminazione dell'oggetto
  /// </summary>
  private bool delete;
  /// <summary>
  /// Costruttore di default: imposta i valori delle coordinate inizio,
fine, centro e del flag di eliminazione
  /// </summary>
  public obj()
  {
   this.inizio = new coord(0, 0);
   this.fine = new coord(0, 0);
   this.centre = new coord(0, 0);
   this.delete = false;
  }
  /// <summary>
  /// Costruttore: assegna ad inizio e fine i valori delle coordinate a e
b,
  /// calcola la coordinata centro,
  /// imposta delete a false
  /// </summary>
  /// <param name="a">Coordinata inizio</param>
  /// <param name="b">Coordinata fine</param>
  public obj(coord a, coord b)
  {
   this.inizio = new coord(a);
   this.fine = new coord(b);
   this.centre = new coord();
   this.centre.setMid(a, b);
   this.delete = false;
  }
  /// <summary>
  /// Costruttore: assegna ad inizio e fine le coordinate dell oggetto
indicato
  /// </summary>
  /// <param name="a">Oggetto con le coordinate da assegnare</param>
  public obj(obj a)
  {
   this.inizio = new coord(a.inizio);
   this.fine = new coord(a.fine);
   this.centre = new coord(a.centre);
   this.delete = false;
  }
  /// <summary>
  /// Costruttore: imposta inizio e fine con i valori indicati,
  /// calcola la coordinata media,
  /// imposta delete a false
  /// </summary>
  /// <param name="x_i">Valore x di inizio</param>
  /// <param name="y_i">Valore y di inizio</param>
  /// <param name="x_f">Valore x di fine</param>
  /// <param name="y_f">Valore y di fine</param>
  public obj(int x_i, int y_i, int x_f, int y_f)
  {
   this.inizio = new coord(x_i, y_i);
   this.fine = new coord(x_f, y_f);
   this.centre = new coord();
```

```
   this.centre.setMid(this.inizio, this.fine);
   this.delete = false;
  }
  /// <summary>
  /// Modifica i valori di inizio e fine con quelli dell'oggetto indicato
  /// </summary>
  /// <param name="a">Oggetto con i valori da assegnare</param>
  public void setObj(obj a)
  {
   this.inizio.setCoord(a.inizio);
   this.fine.setCoord(a.fine);
   this.centre.setCoord(a.centre);
  }
  /// <summary>
  /// Modifica il valore di inizio con quelli indicati e aggiorna il
centro
  /// </summary>
  /// <param name="x_pos">Valore x di inizio</param>
  /// <param name="y_pos">Valore y di inizio</param>
  public void setInizio(int x_pos, int y_pos)
  {
   this.inizio.setCoord(x_pos, y_pos);
   this.centre.setMid(this.inizio, this.fine);
  }
  /// <summary>
  /// Modifica il valore di fine con quelli indicati e aggiorna il centro
  /// </summary>
  /// <param name="x_pos">Valore x di fine</param>
  /// <param name="y_pos">Valore y di fine</param>
  public void setFine(int x_pos, int y_pos)
  {
   this.fine.setCoord(x_pos, y_pos);
   this.centre.setMid(this.inizio, this.fine);
  }
  /// <summary>
  /// Imposta i valori minimi e massimi su inizio e fine valutando tra i
valori correnti e quelli di b,
  /// aggiorna il valore centro
  /// </summary>
  /// <param name="b">Oggetto di riferimento</param>
  public void update(obj b)
  {
   this.inizio.setCoord(Math.Min(this.inizio.x, b.inizio.x),
Math.Min(this.inizio.y, b.inizio.y));
   this.fine.setCoord(Math.Max(this.fine.x, b.fine.x),
Math.Max(this.fine.y, b.fine.y));
   this.centre.setMid(this.inizio, this.fine);
  }
  /// <summary>
  /// Valuta se un oggetto ha le dimensioni minime per essere considerato
tale.
  /// </summary>
  /// <param name="minObj">Valore minimo di x e y perchè l'oggetto sia
considerato valido</param>
  /// <returns>Restituisce true se l'oggetto non è valido</returns>
  public bool fakeObj(int minObj)
  {
```

```
   if (this.fine.x - this.inizio.x < minObj || this.fine.y -
this.inizio.y < minObj)
    return true;
   else return false;
  }
  /// <summary>
  /// Assegna ad id il valore indicato
  /// </summary>
  /// <param name="val">Valore da assegnare ad id</param>
  public void setId(int val)
  {
   this.id = val;
  }
  /// <summary>
  /// Restituisce il valore di id
  /// </summary>
  /// <returns>Restituisce il valore di id </returns>
  public int setId()
  {
   return this.id;
  }
  /// <summary>
  /// Imposta delete con il valore indicato
  /// </summary>
  /// <param name="val">Valore da assegnare a delete</param>
  public void setDelete(bool val)
  {
   this.delete = val;
  }
  /// <summary>
  /// Restituisce il valore del flag delete
  /// </summary>
  /// <returns>Restituisce il valore del flag delete</returns>
  public bool setDelete()
  {
   if (this.delete == true) return true;
   else return false;
  }
  /// <summary>
  /// Stampa le coordinate di un oggetto
  /// </summary>
  public void print()
  {
   /*Console.Write("Inizio: ");
   this.inizio.print();
   Console.Write("Fine: ");
   this.fine.print();
   Console.Write("Centro: ");
   this.centre.print();
   Console.WriteLine("ID: " + this.id);*/
   Console.Write("ID: " + this.id + " - Center: ");
   this.centre.print();
  }
  public void printLine()
  {
   Console.WriteLine("OBJ : " + this.inizio.x + " - " + this.fine.x + " -
" + this.centre.x);
```

```
  }
 }
 /// <summary>
 /// Classe per la gestione delle tracce audio
 /// </summary>
 class track
 {
  /// <summary>
  /// Valori dell'area di immagine da considerare
  /// </summary>
  private obj stage;
  /// <summary>
  /// Coordinata centrale dell'oggetto associato alla traccia
  /// </summary>
  private coord pos;
  /// <summary>
  /// Path della cartella contenente i sample
  /// </summary>
  private const string sample = "D:/_TESI/sample/";
  /// <summary>
  /// Identificativo della traccia, inizializzato a -1
  /// </summary>
  private int id = -1;
  /// <summary>
  /// Motore sonoro della traccia
  /// </summary>
  private ISoundEngine engine;
  /// <summary>
  /// Rappresenta il suono attualmente in riproduzione
  /// </summary>
  private ISound instrument;
  /// <summary>
  /// Costruttore: imposta i parametri della traccia e avvia la
riproduzione
  /// </summary>
  /// <param name="position">Coordinata dell'oggetto associato alla
traccia</param>
  /// <param name="stg">Coordinate dell'area di immagine attiva</param>
  /// <param name="sync">Traccia per la sincronizzazione</param>
  public track(obj position, obj stg, ISound sync)
  {
   this.id = position.setId();
   this.pos = new coord(position.centre);
   this.stage = new obj(stg);
   string sampleName = sample + this.id + ".wav";
   this.engine = new ISoundEngine();
   this.engine.SoundVolume = 1;
   this.instrument = engine.Play2D(sampleName, true, true,
StreamMode.AutoDetect, true);
   this.instrument.Volume = setVol(position.centre);
   this.instrument.Pan = setPan(position.centre);
   this.instrument.PlayPosition = sync.PlayPosition;
   this.instrument.Paused = false;
  }
  /// <summary>
  /// Libera le risorse della traccia
  /// </summary>
```

```
  public void Dispose()
  {
   this.instrument.Stop();
   this.instrument.Dispose();
  }
  /// <summary>
  /// Ferma la riproduzione
  /// </summary>
  public void stop()
  {
   this.instrument.Stop();
   this.instrument.Dispose();
  }
  /// <summary>
  /// Imposta i valori di coordinata, del volume e del panpot
  /// </summary>
  /// <param name="position">Coordinata dei valori da impostare</param>
  public void setPos(coord position)
  {
   this.instrument.Volume = setVol(position);
   this.instrument.Pan = setPan(position);
   this.pos.setCoord(position);
  }
  /// <summary>
  /// Stampa la coordinata indicata
  /// </summary>
  /// <param name="position">Coordinata da stampare</param>
  public void print(coord position)
  {
   Console.WriteLine("Label: " + this.id);
   Console.WriteLine("X: " + position.x + " - Y: " + position.y);
   Console.WriteLine("|==========|");
  }
  /// <summary>
  /// Calcola il valore del parametro volume per l'oggetto indicato
  /// </summary>
  /// <param name="obj">Oggetto di cui calcolare il volume</param>
  /// <returns>Restituisce il valore del parametro volume</returns>
  public float setVol(coord obj)
  {
   return (float)(this.stage.fine.y - obj.y) / (float)(this.stage.fine.y
- this.stage.inizio.y);
  }
  /// <summary>
  /// Calcola il valore del parametro pan per l'oggetto indicato
  /// </summary>
  /// <param name="obj">Oggetto di cui calcolare il pan</param>
  /// <returns>Restituisce il valore del parametro pan</returns>
  public float setPan(coord obj)
  {
   return (2 * (float)(obj.x - this.stage.inizio.x) /
(float)(stage.fine.x - this.stage.inizio.x)) - 1;
  }
 }
}
```

# BIBLIOGRAFIA


[1] Yaakov Bar-Shalom and Xiao-Rong Li, Multitarget-multisensor tracking: Principles and techniques. Storrs, CT: University of Connecticut, 1995.

[2] Alper and Javed Yilmaz and Omar and Shah Mubarak, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13, 2006.

[3] A.G. Bors and I. Pitas, "Prediction and tracking of moving objects in image sequences," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1441--1445, 2000.

[4] Microsoft. (2004, Agosto) Windows hardware developer central. [Online]. http://www.microsoft.com/whdc/device/stillimage/WIA-arch.mspx

[5] P. Fieguth and D. Terzopoulos, "Color-based tracking of heads and other mobile objects at video frame rates," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Puerto Rico, 1997, p. 21.

[6] V. Cappellini, K. Plataniotis, and A. Venetsanopoulos, "Application of color image processing," in *Proceedings of the international conference on digital signal processing*, Limassol, Cyprus, 1995.

[7] Lei Zhai, Dong Shouping, and Ma Honglian, "Recent Methods and Applications on Image Edge Detection," in *Proceedings of the 2008 International Workshop on Education Technology and Training & 2008 International Workshop on Geoscience and Remote Sensing - Volume 01*, 2008, pp. 332--335.

[8] F.Y. Shih and S. Cheng, "Automatic seeded region growing for color image segmentation," *Image and Vision Computing*, vol. 23, no. 10, pp. 877--886, 2005.

[9] Microsoft. (2010) MSDN - Microsoft Developer Network. [Online]. http://msdn.microsoft.com/en-us/library/kx37x362.aspx

[10] Andrew Troelsen, *Pro C# 2008 and the.NET 3.5 Platform, Fourth Edition*.: apress, 2007.

[11] Ambiera e.U. (2009) Ambiera Software Development. [Online]. http://www.ambiera.com/irrklang/index.html

[12] Y. Zhou and S. Suri, "Analysis of a bounding box heuristic for object intersection," *Journal of the ACM (JACM)*, vol. 46, no. 6, pp. 833--857, 1999.

[13] Minglun Gong, Aaron Langille, and Mingwei Gong, "Real-time image processing using graphics hardware: A performance study," in *Second International Conference, ICIAR 2005, Toronto, Canada, September 28-30, 2005. Proceedings*, Toronto, 2005, pp. 1217-1225.

[14] R. Murray Schäfer, *Il paesaggio sonoro*, Seconda edizione ed. Lucca, Italia: Ricordi LIM, 1998.