

Introduzione di elementi multimodali nel gioco *Pong*

Gesture Control, Realtà aumentata e Projection Mapping

Giorgia Di Tommaso ¹

Luca Marchionni ²

Stefano Sinisi ³

Progetto di Interazione Multimodale
*Docente Maria De Marsico*⁴

Version 3.00, February, 2014

¹giorgiaditom@gmail.com

²luca.marchionni89@gmail.com

³stefano.sinisi@gmail.com

⁴demarsico@di.uniroma1.it

Copyright ©2014 Giorgia Di Tommaso, Luca Marchionni, Stefano Sinisi and Contributors.

All rights reserved.

Prefazione

Capitolo 1 *Introduzione.*

Capitolo 2 *Stato dell'arte.* Espone in modo dettagliato le caratteristiche principali del gioco collocandolo nel contesto storico nel quale nasce e si sviluppa. È illustrato lo stato dell'arte del progetto, gli spunti da cui questo ha origine e vengono messi in luce i principi fortemente innovativi di tale metodo e le differenze con il progetto originale.

Capitolo 3 *Multimodalità utilizzata.* Si fornisce una descrizione delle tecniche di interazione multimodale sul quale il progetto si basa.

Capitolo 4 *Progettazione Controller.* Presenta le caratteristiche principali dell'hardware Arduino con il quale avviene l'interazione con l'utente. Infine sono esposte le componenti principali di tale dispositivo il loro funzionamento e la loro interconnessione

Capitolo 5 *Fase di analisi.* Sono illustrati i casi d'uso e i diagrammi di attività, punto di partenza per la pianificazione del progetto.

Capitolo 6 *Fase di design.* È presentata l'architettura del progetto fornendone una visione globale e una descrizione di tutte le componenti che la costituiscono.

Capitolo 7 *Implementazione.* Sono descritte e giustificate le scelte implementative effettuate considerando i problemi incontrati nello sviluppo. Sono qui esposti i criteri e il codice utilizzato per elaborare le principali funzionalità del progetto.

Capitolo 8 *Test Sperimentali.* Il progetto si conclude con esempi di utilizzo dell'applicazione illustrando e spiegando i test effettuati e gli eventuali problemi riscontrati.

Capitolo 9 *Conclusione e sviluppi futuri.* Sono esposte considerazioni riguardo il raggiungimento degli obiettivi prefissati, la qualità del percorso proposto e la funzionalità del software. Infine sono presentate eventuali soluzioni per ampliare il progetto.

Appendice A *Algoritmi di rilevamento dei bordi.* Si illustrano gli algoritmi utilizzati nel progetto.

Indice

1	Introduzione	1
2	Stato dell'arte	3
3	Multimodalità utilizzata	6
3.1	Realtà virtuale e realtà aumentata	6
3.2	Gesture control	7
3.3	Projection mapping	8
4	Progettazione Controller	10
4.1	Arduino	10
4.1.1	Arduino Uno	11
4.2	Accelerometro e Giroscopio	12
4.2.1	MPU6050 GY-521	12
5	Fase di analisi	15
5.1	Attori	15
5.2	Casi d'uso	16
5.3	Diagrammi di attività	17
5.3.1	Acquisire immagine	17
5.3.2	Giocare	18
5.4	Classi di analisi	19
5.4.1	Diagramma dei package	19
5.4.2	Diagramma delle classi	20
5.4.3	Classi del package Arduino	21
5.4.4	Classi del package GUI	23
5.4.5	Classi del package Level Detection	24
5.4.6	Classi del package Utils	25
6	Fase di design	26
6.1	Architettura di design	26
6.2	OpenCV	27
6.3	Box2D	27

7	Implementazione	28
7.1	Arduino Sketch	28
7.1.1	Filtri Kalman	29
7.2	OpenCV	29
7.2.1	Catturare lo scenario	29
7.2.2	Rilevamento rette	30
7.3	Box2D e JavaFX	30
7.3.1	Creazione mondo	30
7.3.2	Ascoltatore di collisioni	32
8	Test sperimentali	34
8.1	Rilevamento oggetti	34
8.2	Gioco	36
8.3	Proiezione livello	37
9	Conclusioni e sviluppi futuri	39
9.1	Conclusioni	39
9.2	Sviluppi futuri	39
A	Algoritmi di rilevamento dei bordi	41
A.1	Edge Detection	41
A.2	Lines Detection	44
A.2.1	Hough Line Transform	44
A.2.2	Accumulatore	44

Elenco delle figure

2.1	Pong: gameplay originale del 1972	3
3.1	Riconoscimento dei gesti	8
3.2	Riconoscimento dei gesti in un gioco	8
4.1	Arduino Board Uno	10
4.2	Gradi di libertà dell'accelerometro e del giroscopio	12
4.3	Sensore MPU6050	12
4.4	Descrizione delle connessioni tra il sensore MPU6050 GY-521 e Arduino Board Uno	14
5.1	Diagramma dei casi d'uso	16
5.2	Diagramma di attività Acquisire immagine	17
5.3	Diagramma di attività Giocare	18
5.4	Diagramma dei package	19
5.5	Diagramma delle classi	20
5.6	Classi del package Arduino	21
5.7	Classi del package Game	22
5.8	Classi del package GUI	23
5.9	Classi del package Level Detection	24
5.10	Classi del package Utils	25
6.1	Architettura del sistema	26
8.1	Immagine di esempio per l'estrazione dei bordi	34
8.2	Esempi di output al variare delle soglie	35
8.3	Immagine di esempio per l'estrazione dei bordi	36
8.4	Immagine di esempio per l'estrazione dei bordi	36
8.5	Esempi di costruzione e proiezione di un livello	37
8.6	Strumenti utilizzati per le fasi di test	38
A.1	La forma di un filtro gaussiano tipico	42
A.2	Rilevamento bordi mediante il gradiente	43

Elenco delle tabelle

4.1	Specifiche tecniche Arduino Board Uno	11
4.2	Descrizione delle porte di entrata e uscita del sensore MPU6050 GY-521	13
5.1	Attori del sistema	15
5.2	Caso d'uso acquisire immagine	16
5.3	Caso d'uso giocare	17

Elenco degli Algoritmi

A.2.1Hough Lines Transform	45
--------------------------------------	----

Capitolo 1

Introduzione

Il computer è attualmente uno strumento di lavoro irrinunciabile. Il suo crescente sviluppo non solo ha provocato un cambiamento nel nostro modo di comunicare e interagire con gli altri, ma anche nel nostro modo di giocare. Le caratteristiche di questi elaboratori elettronici, in particolare la versatilità e l'interattività, hanno offerto una buona spinta alla ricerca nel campo della ricostruzione di situazioni reali.

Il vero punto di svolta si è raggiunto con l'introduzione della multimedialità, ovvero la possibilità di comunicare coinvolgendo non solo l'udito e la vista, ma anche il tatto e l'olfatto, cercando di riprodurre sensazioni fisiche il più possibile affini alla realtà.

In questo contesto si colloca la nascita del gioco interattivo PONG¹. Quello che ci si propone con lo sviluppo di questo progetto è di garantire un'esperienza emozionale e sensoriale nuova e differente. Eliminando il confine fisico e percettivo tra l'ambiente reale e quello computerizzato l'utente da semplice spettatore diviene un partecipante attivo che costruisce in prima persona la realtà, interagendo con essa.

L'obiettivo perseguito è stato quello di convertire un gioco già esistente e molto noto, arricchendolo con l'introduzione di tecniche nuove quali lo sviluppo della realtà virtuale e il gesture control. La combinazione di queste tecniche induce, tramite un sistema più o meno immersivo, a pensare di vivere una certa realtà ingannando i nostri sensi, ma tale realtà è completamente generata dal computer.

Nel gioco in questione, infatti, si ha l'idea che la palla, caratteristica del gioco tradizionale, rimbalzi su oggetti fisici presenti nella parete scelta per

¹<http://en.wikipedia.org/wiki/Pong>

giocare.

Il gioco è stato selezionato tra l'ampia varietà di giochi per computer differenziandosi dagli altri per la sua semplicità, ma allo stesso tempo il forte coinvolgimento dell'utente, catturato dall'ambizione di accrescere il punteggio e mettere alla prova i propri riflessi. Si è cercato di riprodurre il gioco fedelmente, salvaguardando lo scopo originale.

Lo studio è partito dall'analisi di un video dell'interaction designer Sures Kumar², trovato in rete, ed è stato poi arricchito da altre fonti cartacee e multimediali.

Oltre all'approfondimento dei sistemi di realtà virtuale, allo studio e alla progettazione del gioco è stato necessario uno studio del sistema Arduino, del suo linguaggio e dei modi per interfacciarsi con l'utente e con il computer.

Da queste premesse ne è emerso un gioco interattivo che si inserisce tra quelle esperienze in cui ambiente reale e ambiente virtuale si fondono donando un'esperienza multisensoriale e unica.

Nei capitoli a seguire sarà possibile leggere nel dettaglio le scelte implementative, gli approfondimenti effettuati e tutti gli aspetti che hanno portato alla realizzazione del progetto.

²http://sureskumar.com/?page_id=114

Capitolo 2

Stato dell'arte

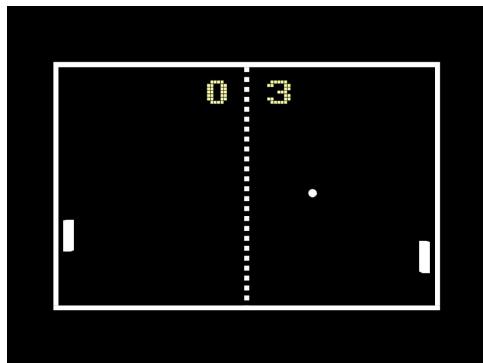


Figura 2.1: Pong: gameplay originale del 1972

Il gioco *Pong* vede la sua pubblicazione nel 1972 grazie alla casa produttrice **Atari** e risulta essere uno dei primi videogiochi commercializzati. Intuitivamente molto semplice si pone come un simulatore del classico gioco di Ping pong. La sua interfaccia, inizialmente in bianco e nero, prevede soltanto una *pallina* e due *barrette*. Le modalità di gioco previste sono due: *Single player* o *Multiplayer*. Lo scopo è oltrepassare la barretta avversaria con la pallina.

Variante successiva del gioco è quella che prevede un solo giocatore, il cui scopo è colpire la pallina senza farla cadere, nonostante questa prenda velocità e differenti direzioni a seconda degli ostacoli colpiti, presenti nella scena. Inoltre la barra si riduce ad ogni errore diminuendo le probabilità di colpire la palla per accrescere la difficoltà e mettere alla prova l'utente.

La complessità dei temi trattati ha portato a direzionare le ricerche verso un progetto che potesse essere divertente, ma allo stesso tempo fosse rappresentativo del concetto di multimodalità e potesse mettere alla prova le proprie

abilità. L'argomento della multimodalità è ancora molto dibattuto e ignoto, solo negli ultimi anni questa disciplina si sta sviluppando in vari ambiti come quello dei videogiochi.

L'idea del progetto proposto nasce da un video trovato in rete¹ in cui un designer Sures Kumar insieme al suo team mostra un framework basato sul concetto di realtà virtuale che cerca di sfruttare lo spazio architettonico permettendo agli oggetti virtuali di giocare con lo spazio reale secondo i gesti dell'utente. Il video è dimostrativo quindi non introduce codice o dettagli tecnici ma solo un breve elenco degli strumenti e dei software utilizzati. Questo ha consentito di approfondire gli elementi citati e dopo un'analisi valutarne la validità e nel caso sostituirli con tecnologie migliori.

Nel progetto esistente l'interazione con l'utente avviene attraverso la tecnologia Arduino, una scheda elettronica dotata di microcontrollore e circuiteria utile per creare interazioni tra macchina e utente. Alla scheda principale vengono collegati un accelerometro a tre assi, un giroscopio monoasse e un force pressure sensor.

Analogamente, anche nel progetto trattato l'interazione con l'utente viene effettuata attraverso Arduino, ma all'uso di due sensori si è preferito l'utilizzo di una scheda unica con accelerometro e giroscopio integrati, dotata dello stesso grado di precisione e ritenuta più efficiente e pratica, in quanto permette un'ottimizzazione dell'attrezzatura. Inoltre si è deciso di non utilizzare il sensore di pressione e gestire l'avvio e la chiusura del gioco direttamente dal codice.

Per interfacciare il computer con l'hardware Arduino, nel progetto d'origine viene utilizzato un software open-source basato su Java, ma dotato di funzioni ad alto livello per gestire facilmente aspetti multimediali e grafici. Tuttavia si è pensato di far prevalere la robustezza al posto della semplicità, sviluppando interamente il progetto in Java si è raggiunta una maggiore flessibilità e una maggiore capacità di riutilizzo e integrazione delle varie parti che compongono il gioco.

Per poter fare in modo che la palla interagisca con gli oggetti reali è necessario catturare la posizione e le coordinate della parete in cui si desidera proiettare il gioco. Questo nel progetto originale avviene con una tecnica manuale attraverso l'uso di uno software grafico come Paint.

In questo progetto grazie anche alla linearità di sviluppo si è riusciti ad inserire all'interno del codice degli algoritmi di *Computer Vision* che hanno consentito di gestire l'acquisizione delle immagini in modo automatizzato senza l'intervento dell'utente. Inoltre per dare alla pallina movimenti più realistici tenendo conto delle leggi fisiche del moto e della gravità si è utilizzato

¹<http://sureskumar.com/?p=362>

un motore fisico.

Passo finale comune ad entrambi i progetti è la tecnica del *projection mapping*, tecnica di proiezione approfondita nel Capitolo 3, indispensabile per dare agli oggetti virtuali l'illusione di essere parte della realtà.

Capitolo 3

Multimodalità utilizzata

Produrre effetti sul mondo simulato è uno degli elementi cruciali per l'illusione di realtà. Ma quali sono le tecniche che permettono tale illusione?

3.1 Realtà virtuale e realtà aumentata



(a) Realtà virtuale



(b) Realtà aumentata

La realtà virtuale così come la realtà aumentata è una branca della computer graphics che studia e sviluppa sistemi in grado di combinare immagini provenienti dal mondo reale con informazioni elaborate dal computer.

L'utente di un'applicazione di realtà virtuale, utilizzando opportune apparecchiature, è nella condizione di vivere un'esperienza sensoriale arricchita di informazioni ed elementi virtuali, con il quale può interagire.

Il primo a creare una macchina che permettesse una totale immersione dei sensi in un altro mondo fu Morton Heiling nel 1957 che brevettò un simulatore chiamato **Sensorama** che coinvolgesse l'utente durante la visione di un film. Solo in seguito si attribuì a questa tecnica il nome di realtà virtuale. Nello stesso periodo Tom Caudell durante il cablaggio di aeromobili Boing si avvalse dell'aiuto di un sistema che in tempo reale gli desse informazioni su come installare i cavi senza commettere errori coniando il termine di *Realtà Aumentata*. Da allora la differenza tra realtà aumentata e realtà virtuale è fonte di dibattito.

Nel corso degli anni si sono date diverse definizioni formali dalle quali si evince che le due discipline, pur attingendo allo stesso bagaglio di conoscenze, sono concettualmente diverse. Erroneamente oggi si tende ad indicare con il termine realtà virtuale tutte quelle simulazioni che consentono un qualche grado di interazione con l'ambiente descritto.

La differenza fondamentale fra realtà aumentata e virtuale consiste nel concetto di simulazione utilizzato. Si può brevemente affermare che mentre la realtà virtuale indica una realtà simulata in cui l'utente è immerso in un mondo artificiale, la realtà aumentata si interessa di "ampliare" la realtà inserendo informazioni multimediali nel mondo reale (ne sono esempi la chirurgia robotica o gli smartphone dotati di applicazioni di geolocalizzazione).

Quando, come nel caso del progetto presentato, non è possibile definire quale sia il fattore dominante, se il mondo reale o quello virtuale allora si parla di realtà mixata.

In particolare queste tecniche trovano spazio in ambito militare per le esercitazioni o in ambito medico e ingegneristico per supportare l'utilizzatore con una serie di informazioni aggiunte digitalmente alla realtà.

3.2 Gesture control

Con il termine "gesture control" si indica quell'insieme di gesti naturali e intuitivi che permettono di interagire con il computer. I gesti possono essere originati da qualsiasi posizione o movimento del corpo, ma solitamente sono prodotti dal volto o dalla mano.

Il riconoscimento dei gesti è utilizzato per elaborare informazioni non verbali. Alla base dell'interpretazione dei gesti ci sono complessi algoritmi di visione artificiale, alcuni approcci utilizzano solo fotocamere basate sul riconoscimento visivo mentre altri prevedono l'uso di dispositivi ausiliari, come i guanti, per poter catturare e interpretare il movimento.

I gesti tracciabili non sono tutti uguali né rispettano un preciso dizionario, possono essere spontanei e originali oppure soggetti ad un insieme di regole. Tra i primi troviamo i gesti naturali che accompagnano il discorso, nei secondi abbiamo i gesti deittici utilizzati per dare indicazioni, oppure i gesti del linguaggio dei segni utilizzati per comunicare con persone sordi.

Un'altra categoria di gesti nota con il nome di gesti manipolativi è quella che prevede la manipolazione di dispositivi per generare l'interazione. A una manipolazione dell'oggetto fisico corrisponde lo stesso gesto sullo schermo. Rientrano in questo insieme i gesti utilizzati nel progetto in esame in cui il paddle è controllato stabilendo una relazione tra il movimento della mano e l'entità manipolata.

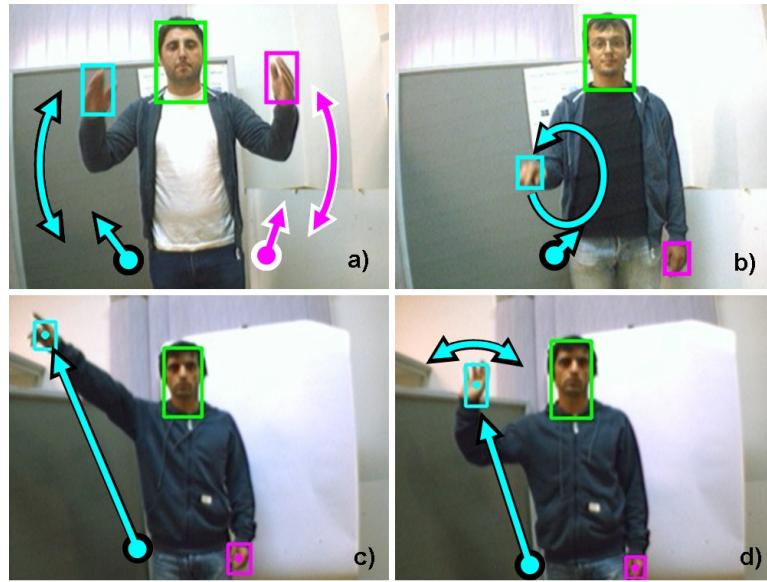


Figura 3.1: Riconoscimento dei gesti



Figura 3.2: Riconoscimento dei gesti in un gioco

3.3 Projection mapping

Con il termine projection mapping si indica una tecnica in cui l'elemento portante è il videoproiettore concepito come mezzo creativo per generare una realtà alternativa.

Non si conosce l'origine esatta di questa tecnica, uno dei suoi primi utilizzi è avvenuto nel 1969 nella Haunted Mansion a Disneyland.

Normalmente, i videoproiettori vengono utilizzati per visualizzare immagini su schermi bianchi piatti. Il projection mapping, invece, usa il videopro-

iettore per visualizzare le immagini su superfici arbitrariamente complesse (non flat). Questa tecnologia di proiezione trasforma oggetti, spesso di forma irregolare, in una superficie di visualizzazione per proiezione video. Utilizzando software specializzato, un oggetto bidimensionale o tridimensionale è spazialmente mappato sul programma virtuale che imita l'ambiente reale sul quale è proiettato.

L'impatto visivo del projection mapping trascende la normale proiezione video per offrire esperienze memorabili per qualsiasi pubblico .

Inoltre è una tecnica "economica" in quanto le immagini sono facilmente riproducibili su qualsiasi superficie o struttura (piccoli oggetti, edifici, paesaggi...).

L'immagine proiettata viene mappata per adattarsi alla superficie, non è necessario apportare modifiche fisiche alla struttura del display. L'immagine può mascherare e modificare l'aspetto di tutto ciò che si sta proiettando.

Il metodo che lo caratterizza si può dividere in tre fasi:

1. Replica virtuale: dopo la scelta dell'oggetto sul quale proiettare si crea una replica virtuale di tutta la fisica della superficie grazie a specifici programmi per computer.
2. Mascheratura: il passo successivo è definito come "mascheratura", in cui le forme e le posizioni dei vari elementi della costruzione sono mascherate utilizzando modelli di opacità. Devono essere definite le coordinate con cui l'oggetto è posto in relazione al proiettore.
3. Regolazioni: infine, si specifica l'orientamento (x, y, z) e la posizione. Le immagini del proiettore si aggiungono alla scena virtuale. Le regolazioni sono necessarie e spesso prevedono un intervento manuale per modificare sia la scena fisica sia virtuale per ottenere migliori risultati.

Tale tecnica è utilizzata da artisti e inserzionisti che possono aggiungere dimensioni extra, illusioni ottiche, e movimento su oggetti precedentemente statici.

Capitolo 4

Progettazione Controller

Il controller proposto è basato sul rilevamento del movimento della mano dell’utente per permettergli di manovrare direttamente il *paddle* del gioco. È realizzato tramite una scheda *Arduino*¹ combinata con un accelerometro e un giroscopio.

4.1 Arduino

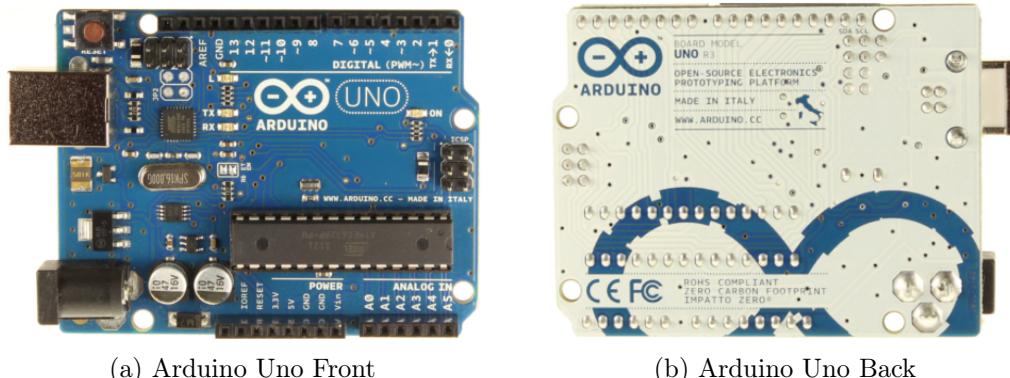


Figura 4.1: Arduino Board Uno

Arduino è il nome di una piattaforma hardware *Open Source* progettata con microcontrollori **ATMEL [-AVR]** con *chip* della serie **megaAVR**. È dotata di circuiteria di contorno che consente di creare rapidamente dispositivi per scopi hobbyistici o didattici. Ne sono un esempio controllori di luci e velocità, sensori per temperatura o per la comunicazione con più dispositivi. Il kit fornito dal produttore prevede un ambiente di sviluppo integrato per la programmazione; il software e gli schemi circuitali sono del tutto liberi.

¹<http://arduino.cc>

La scheda offre inoltre una porta I/O e un'interfaccia USB. A questo hardware è affiancato un *IDE*, ambiente di sviluppo integrato, multipiattaforma *Open Source*.

A livello concettuale, tutte le schede vengono programmate attraverso una porta seriale.

Tra le varie distribuzioni in commercio è stata presa in considerazione la **board Arduino Uno** (Figura 4.1) con le seguenti le specifiche tecniche (Tabella 4.1).

4.1.1 Arduino Uno

Microcontroller	ATmega328
Voltaggio	5V
Voltaggio in Input (raccomandato)	7-12V
Voltaggio in Input (minimo e massimo)	6-20V
Pin I/O Digitali	14
Pin I/O Analogici	6
Corrente DC per Pin I/O	40 mA
Corrente DC per 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocità di Clock	16 MHz

Tabella 4.1: Specifiche tecniche **Arduino Board Uno**

4.2 Accelerometro e Giroscopio

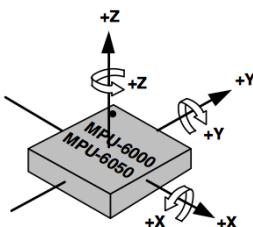


Figura 4.2: Gradi di libertà dell’accelerometro e del giroscopio

L’accelerometro è un dispositivo in grado di misurare l’accelerazione. È costituito da una massa sospesa ad alcuni elementi elastici e attraverso dei sensori ne rileva lo spostamento e lo trasforma in un segnale elettrico. Il segnale elettrico è letto via software ed è elaborato con algoritmi per la rimozione del *rumore* così come descritto nel Capitolo 7.

Il giroscopio è un dispositivo rotante che per effetto della legge di *conservazione della quantità del momento angolare* tende a mantenere il suo asse di rotazione orientato in una direzione fissa. In questo modo quando è fissato ad un supporto il suo asse rimane orientato nella stessa direzione anche se il supporto cambia orientamento. Il funzionamento hardware è analogo a quello dell’accelerometro.

Di seguito è riportata una breve descrizione delle specifiche tecniche del sensore utilizzato per la progettazione del controller.

4.2.1 MPU6050 GY-521



Figura 4.3: Sensore MPU6050

Il sensore è InvenSense MPU-6050 GY-521[Inc13] con accelerometro e giroscopio MEMS² a tre assi integrati (Figura 4.2). Contiene un buffer FIFO

²Microelectromechanical
Microelectromechanical systems

systems:<http://en.wikipedia.org/wiki/>

di 1024 byte. È stato programmato per scrivere i valori dei singoli assi nel buffer così che la board **Arduino** li possa leggere accendendovi.

Il protocollo di comunicazione è I²C, uno standard creato da **Philips**³ per permettere a più dispositivi di *interfacciarsi*. È costituito da due porte: *SDA* per i dati e *SCL* per il clock.

Nome porta	Descrizione
VCC	Alimentazione a 3.3V o 5V
GND	Massa
SCL	Clock I/O
SDA	Data I/O
XDA	Sensori esterni
AD0	-
INT	Segnale di Interrupt programmabile

Tabella 4.2: Descrizione delle porte di entrata e uscita del sensore MPU6050 GY-521

Nella Tabella ?? sono elencate le porte di entrata e di uscita del sensore MPU6050, mentre la Figura 4.4 schematizza come queste porte sono connesse alla scheda **Arduino**.

³<http://en.wikipedia.org/wiki/I2C>

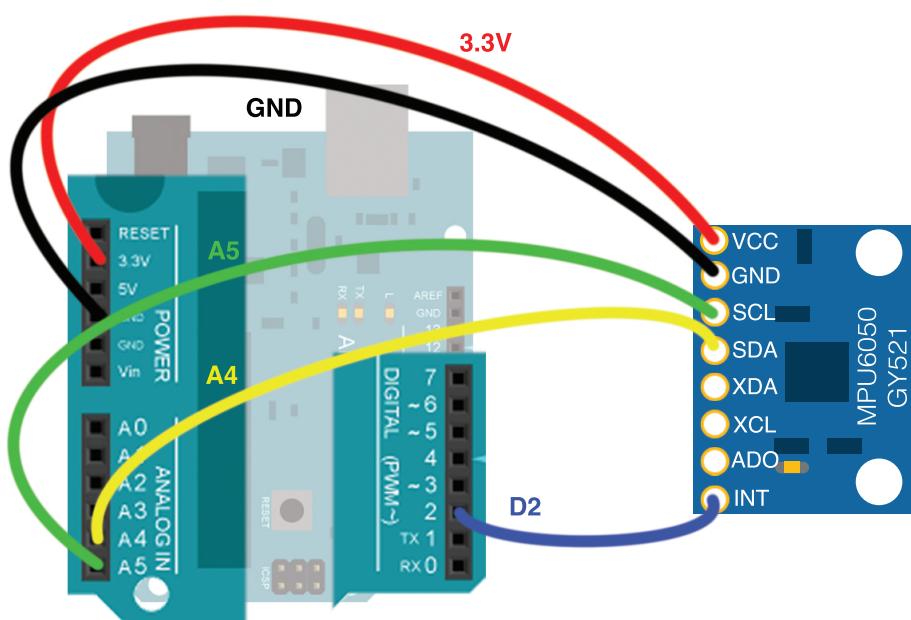


Figura 4.4: Descrizione delle connessioni tra il sensore MPU6050 GY-521 e Arduino Board Uno

Capitolo 5

Fase di analisi

In questo capitolo vengono illustrati i casi d'uso, i quali permettono di individuare e descrivere gli scenari elementari di utilizzo del sistema da parte degli attori che si interfacciano con esso. Come primo passo quindi vengono identificati gli attori e successivamente vengono mostrati i diagrammi di attività che permettono di definire i passaggi da svolgere per realizzare una data funzionalità del sistema. Essi consentono, mediante l'uso di immagini convenzionali, una visione più chiara e rappresentativa delle interazioni tra gli attori e il sistema.

Infine verrà presentato il diagramma delle classi.

5.1 Attori

Nome	Giocatore
Descrizione	Il Giocatore è l'attore che utilizza l'applicazione

Tabella 5.1: Attori del sistema

5.2 Casi d'uso

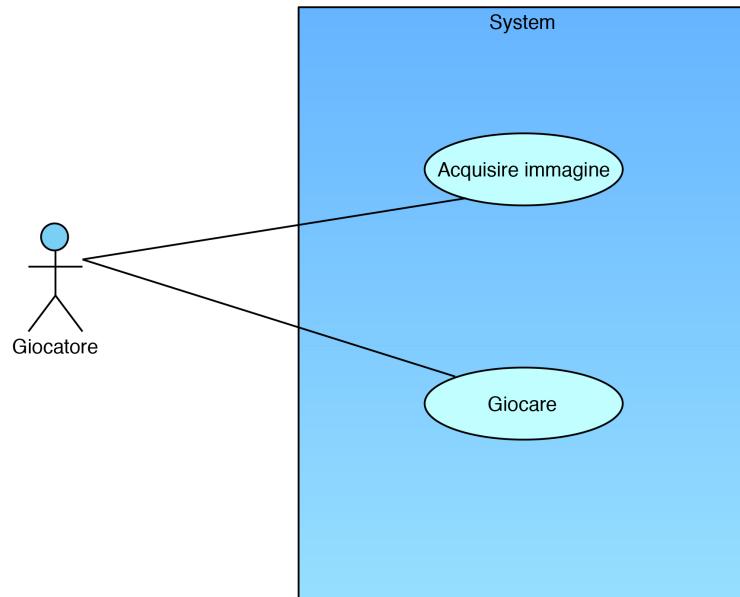


Figura 5.1: Diagramma dei casi d'uso

Caso d'uso	Acquisire immagine
ID	CU_01
Attore	Giocatore
Descrizione	L'utente richiede l'acquisizione dell'immagine che diventerà il livello del gioco
Pre-condizione	L'applicazione è pronta all'utilizzo
Post-condizione	Il sistema avvia la partita
Flusso eventi	<ol style="list-style-type: none">1. L'utente avvia l'applicazione2. Il sistema acquisisce l'immagine tramite webcam3. Il sistema proietta la schermata iniziale del gioco

Tabella 5.2: Caso d'uso **acquisire immagine**

Caso d'uso	Giocare
ID	CU_02
Attore	Giocatore
Descrizione	Il giocatore utilizza l'applicazione
Pre-condizione	L'applicazione è in esecuzione
Post-condizione	L'applicazione termina la sua esecuzione
Flusso eventi	<ol style="list-style-type: none"> 1. Il giocatore richiede di iniziare la partita 2. Il sistema carica la partita 3. Finché la palla rimane in gioco, il giocatore può muovere la barretta per respingere la palla 4. Il sistema aggiorna il punteggio del gioco

Tabella 5.3: Caso d'uso **giocare**

5.3 Diagrammi di attività

5.3.1 Acquisire immagine

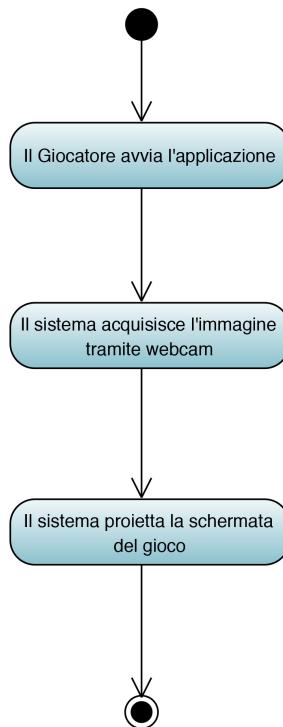


Figura 5.2: Diagramma di attività **Acquisire immagine**

5.3.2 Giocare

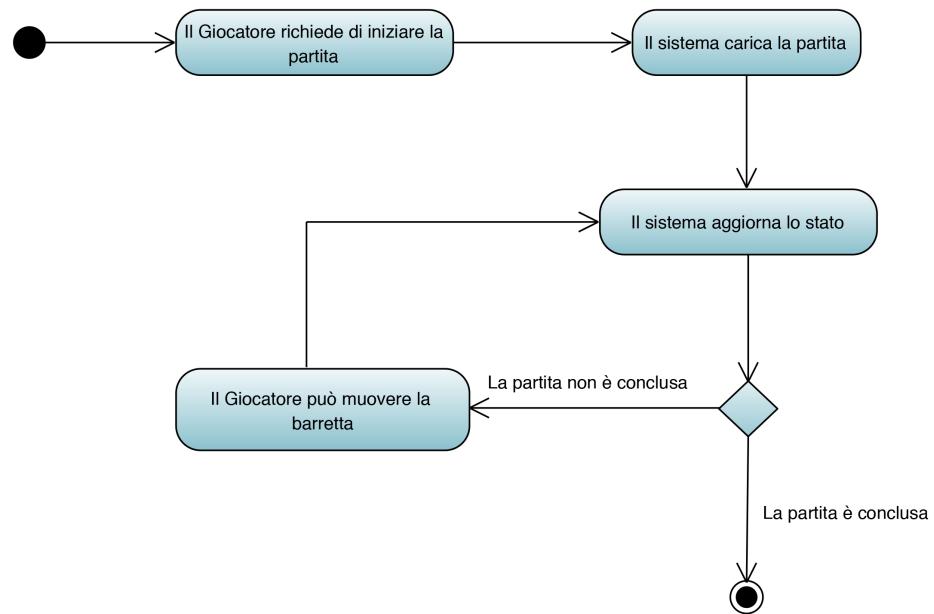


Figura 5.3: Diagramma di attività **Giocare**

5.4 Classi di analisi

5.4.1 Diagramma dei package

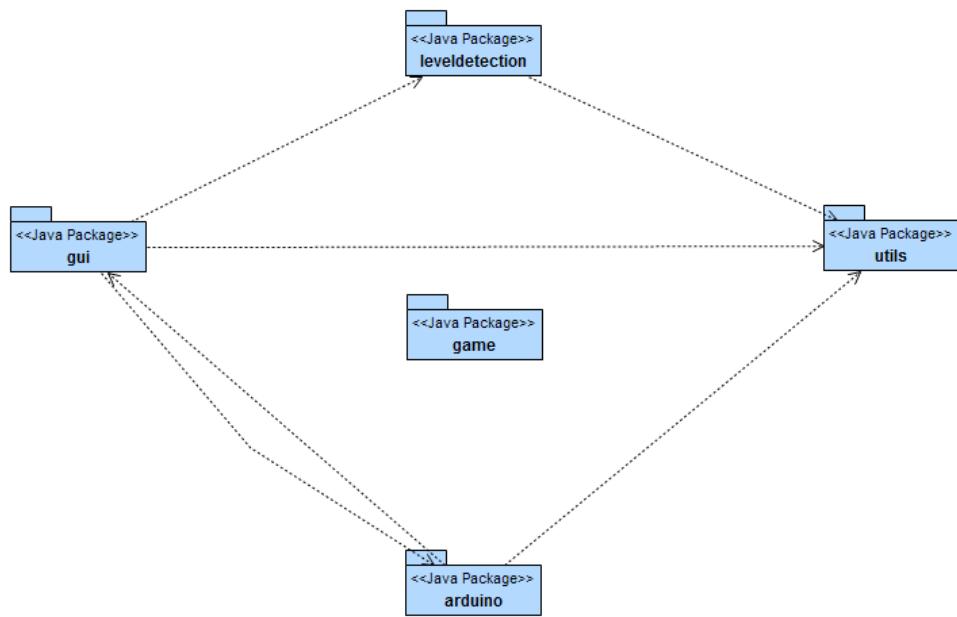


Figura 5.4: Diagramma dei package

5.4.2 Diagramma delle classi

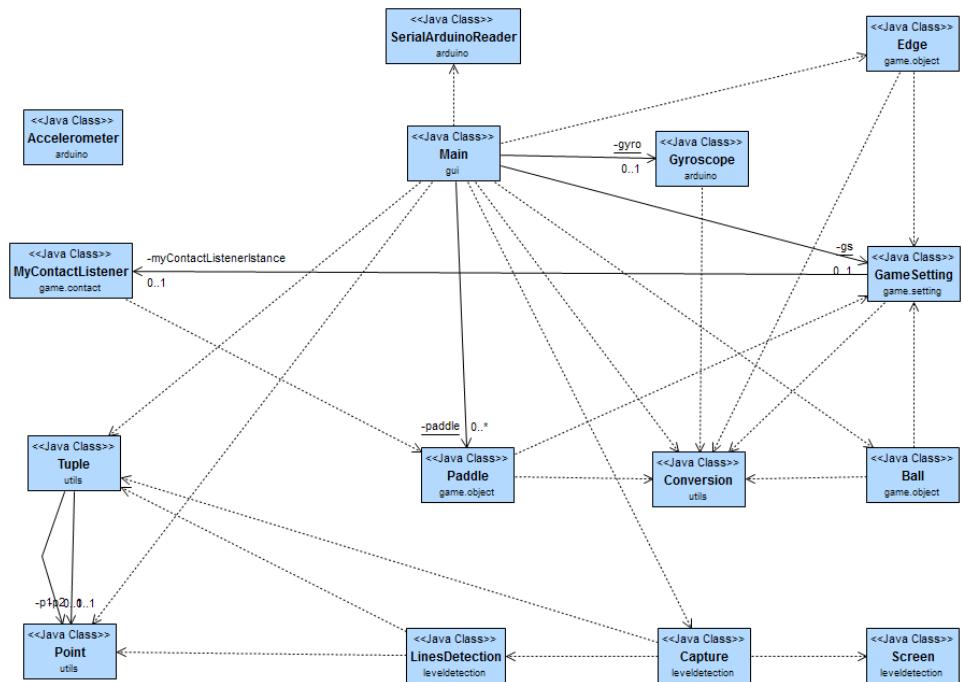


Figura 5.5: Diagramma delle classi

5.4.3 Classi del package Arduino

<pre><<Java Class>> Accelerometer arduino -accDegreeMax: float -accDegreeMin: float -xAngle: float -yAngle: float -zAngle: float +Accelerometer(float,float) +fixAccAngleX(float,float):void +fixAccAngleY(float):void +fixAccAngleZ(float):void +getxAngle():float +setxAngle(float):void +getyAngle():float +setyAngle(float):void +getzAngle():float +setzAngle(float):void</pre>	<pre><<Java Class>> SerialArduinoReader arduino ~serialPort: SerialPort -PORT_NAMES: String[] -input: BufferedReader -output: OutputStream -TIME_OUT: int -DATA_RATE: int -xGyr: float -yGyr: float -zGyr: float -xAcc: float -yAcc: float -zAcc: float -xAcc_offset: float -setted: boolean +SerialArduinoReader() +initialize():void +close():void +serialEvent(SerialPortEvent):void +run():void +main(String[]):void</pre>	<pre><<Java Class>> Gyroscope arduino -gyroDegreeMax: float -gyroDegreeMin: float -xAngle: float -yAngle: float -zAngle: float +Gyroscope(float,float) +fixGyroAngleX(float,float):void +fixGyroAngleY(float):void +fixGyroAngleZ(float):void +getxAngle():float +setxAngle(float):void +getyAngle():float +setyAngle(float):void +getzAngle():float +setzAngle(float):void</pre>
--	---	---

Figura 5.6: Classi del package Arduino

Classi del package Game

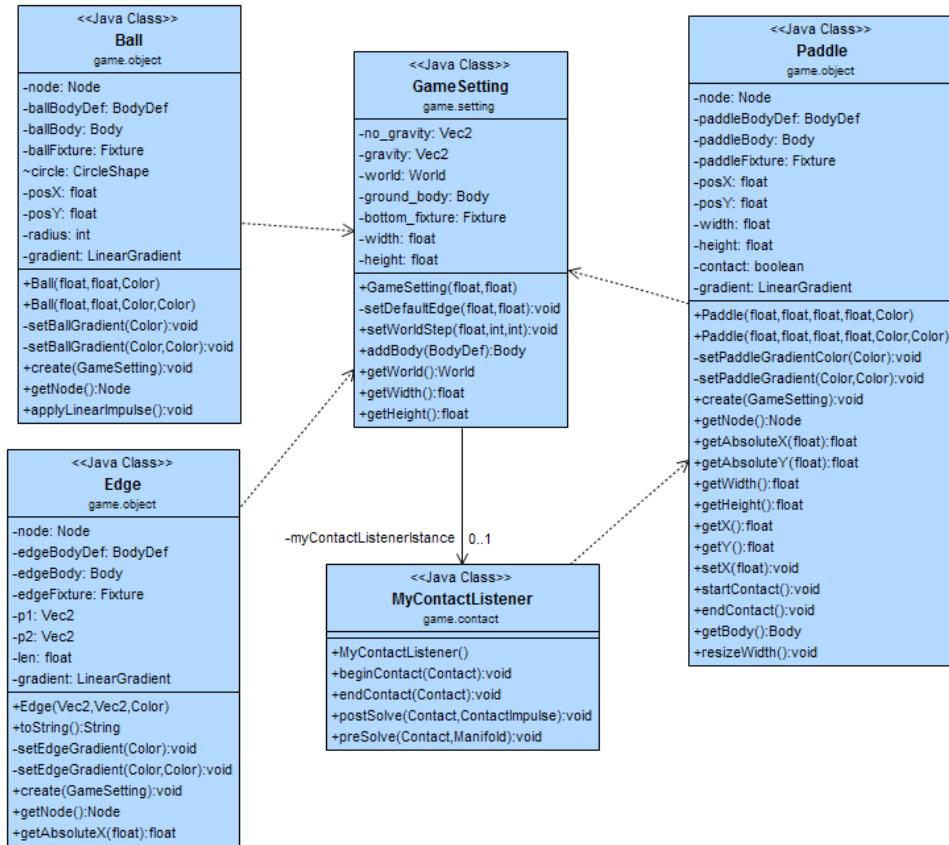


Figura 5.7: Classi del package Game

5.4.4 Classi del package GUI

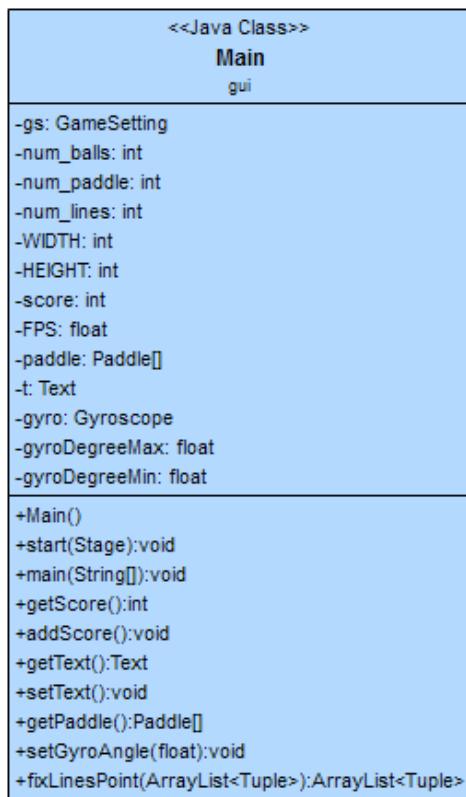


Figura 5.8: Classi del package GUI

5.4.5 Classi del package Level Detection

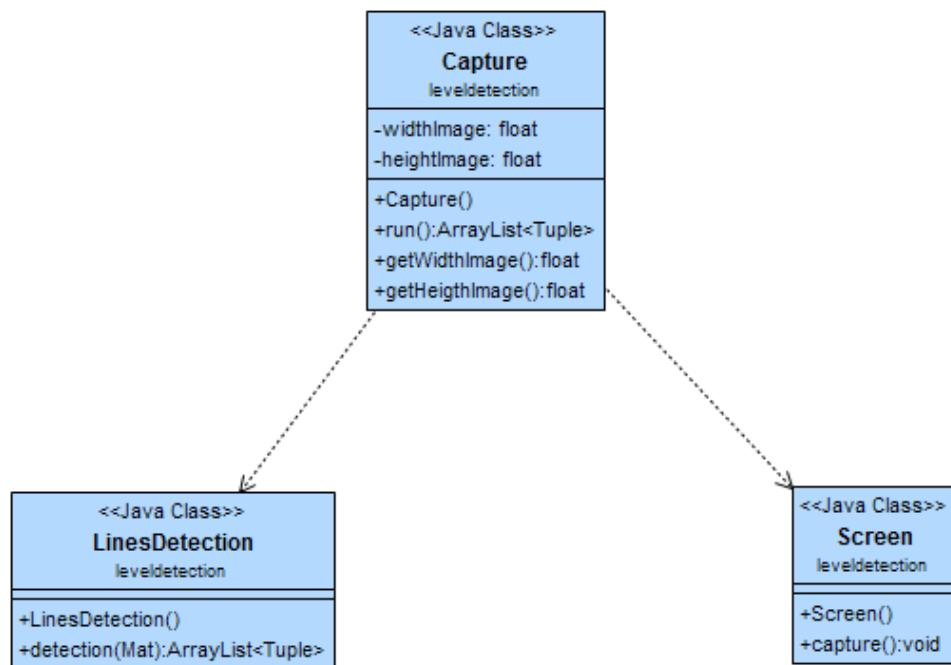


Figura 5.9: Classi del package Level Detection

5.4.6 Classi del package Utils

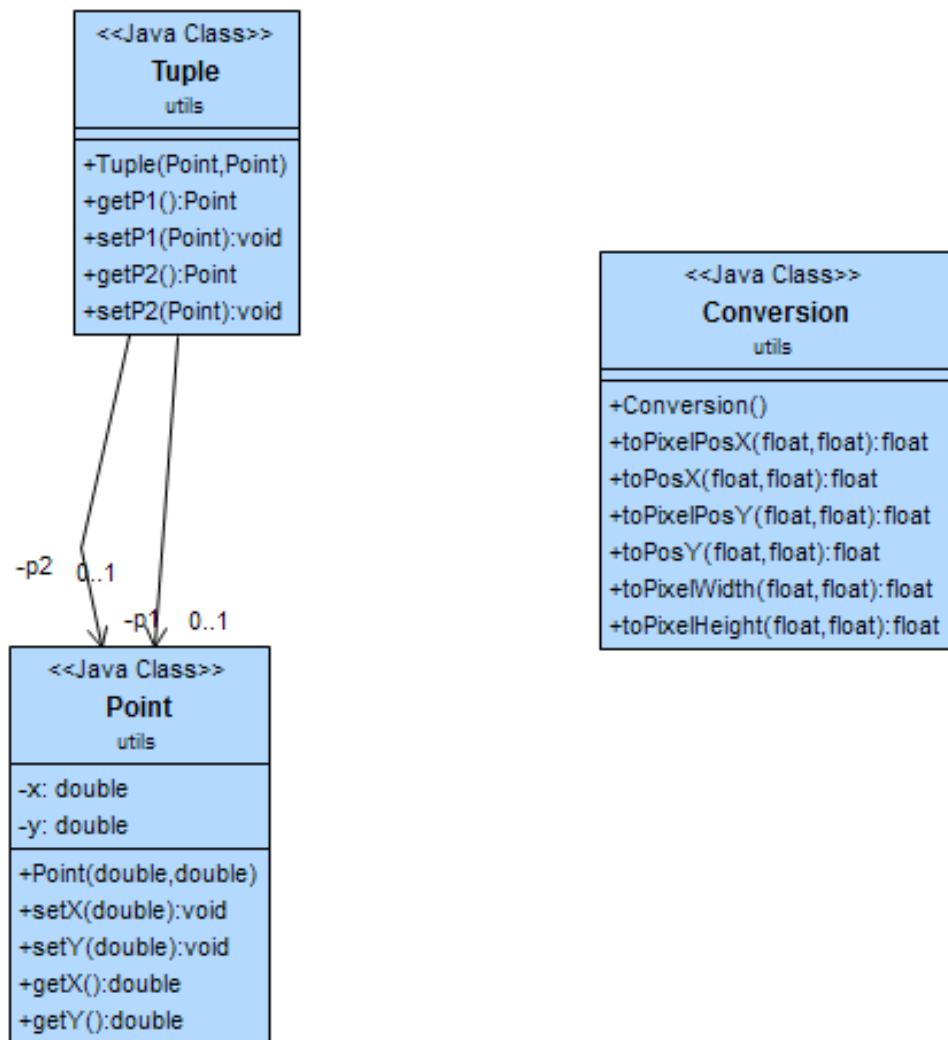


Figura 5.10: Classi del package Utils

Capitolo 6

Fase di design

6.1 Architettura di design

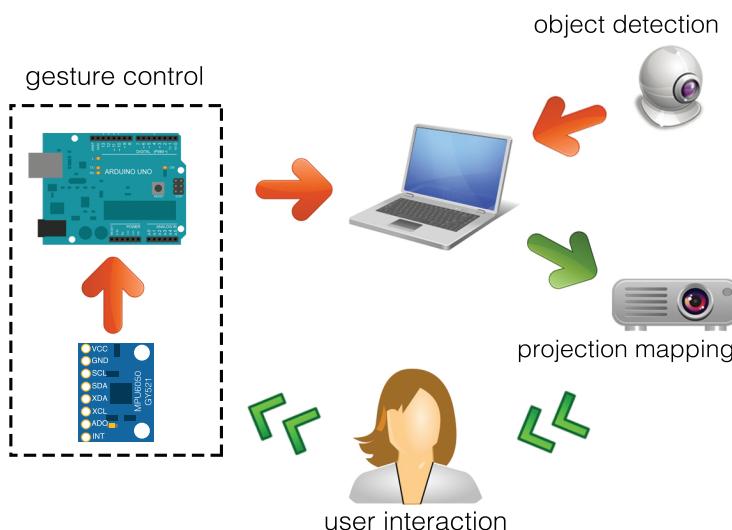


Figura 6.1: Architettura del sistema

L'utente interagisce con il gioco tramite il controller introdotto nel Cap. 4. Il controller programmato in C++ invia le informazioni raccolte (quali per esempio gli angoli assunti dal giroscopio) tramite una porta seriale al sistema, che le preleva e le elabora nel gioco [Mon11, McR13]. L'applicazione è sviluppata in Java e utilizza il framework Box2D per gestire le collisioni che avvengono tra gli oggetti del gioco.
Per la costruzione del livello è necessario rilevare i contorni degli oggetti del mondo reale e si è cercato un *framework* che potesse svolgere tale compito in modo efficiente. La scelta è ricaduta su OpenCV, libreria contenente una

vasta raccolta di algoritmi di *Visione Artificiale* e *Apprendimento Automatico* maggiormente utilizzati.

Gli oggetti del mondo reale che compongono lo scenario sono catturati tramite una *webcam* e sono passati direttamente agli algoritmi di OpenCV come spiegato nel Capitolo 7.

Infine viene proiettato il gioco sulla parete tramite tecniche di Projection Mapping, in modo tale da dare l'illusione che gli oggetti del gioco rimbalzino sugli oggetti reali.

Il risultato finale dell'architettura proposta è in Figura 6.1.

6.2 OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria open source finalizzata soprattutto alla visione artificiale e all'apprendimento automatico. Oltre ad essere multilinguaggio, è anche multipiattaforma. OpenCV contiene numerosi algoritmi che vanno da quelli per rilevare i volti umani a quelli per identificare gli oggetti, da quelli per effettuare l'eye-tracking a quelli per estrarre dei modelli 3D. In questo progetto sono due gli algoritmi di OpenCV utilizzati: con l'algoritmo di Canny [Can86] vengono rilevati i bordi, mentre con l'algoritmo di Hough [Hou62] è possibile, partendo dall'output del primo algoritmo, identificare le rette degli oggetti. Si rimanda all'Appendice A e a [GW08] per ulteriori approfondimenti su entrambi gli algoritmi.

6.3 Box2D

Box2D è un motore fisico open source per la simulazione dei corpi rigidi in 2D, sviluppato in C++. Viene usato in numerose applicazioni ludiche in quanto permette di gestire la fisica del gioco, come per esempio la forza di gravità e l'attrito. Tramite il suo utilizzo è possibile rilevare e gestire facilmente le collisioni per ogni oggetto presente nel gioco. Il sistema di rilevamento delle collisioni opera in tre fasi: una prima fase in cui viene effettuata una scansione degli oggetti, una seconda in cui avviene un continuo rilevamento delle collisioni e un'ultima fase in cui viene utilizzato un algoritmo in grado di gestire tutte le possibili collisioni [Par13].

Capitolo 7

Implementazione

7.1 Arduino Sketch

```
int16_t accX, accY, accZ;
int16_t tempRaw;
int16_t gyroX, gyroY, gyroZ;
uint8_t i2cData[14]; // Buffer for I2C data

accX = ((i2cData[0] << 8) | i2cData[1]);
accY = ((i2cData[2] << 8) | i2cData[3]);
accZ = ((i2cData[4] << 8) | i2cData[5]);
tempRaw = ((i2cData[6] << 8) | i2cData[7]);
gyroX = ((i2cData[8] << 8) | i2cData[9]);
gyroY = ((i2cData[10] << 8) | i2cData[11]);
gyroZ = ((i2cData[12] << 8) | i2cData[13]);
```

Il codice qui riportato è parte dello Sketch caricato su Arduino. In generale serve a leggere i valori dell'accelerometro e del giroscopio ricevuti dal sensore e a scriverli sulla porta seriale.

```
Serial.print(accXangle); Serial.print("\t");
Serial.print(gyroXangle); Serial.print("\t");
Serial.print(compAngleX); Serial.print("\t");
Serial.print(kalAngleX); Serial.print("\t");

Serial.print("\t");

Serial.print(accYangle); Serial.print("\t");
Serial.print(gyroYangle); Serial.print("\t");
Serial.print(compAngleY); Serial.print("\t");
Serial.print(kalAngleY); Serial.print("\t");
```

Questi valori sono letti dal bus I²C come spiegato nel Capitolo 4 e sono convertiti in angoli con il seguente codice per comodità di implementazione.

```
uble gyroXrate = (double)gyroX / 131.0;
uble gyroYrate = -((double)gyroY / 131.0);
roXangle += gyroXrate * ((double)(micros() - timer) /
    1000000); // Calculate gyro angle without any filter
roYangle += gyroYrate * ((double)(micros() - timer) /
    1000000);
```

7.1.1 Filtri Kalman

Prima di scrivere i valori sulla porta seriale, applichiamo dei filtri per ridurne il rumore. Uno dei filtri più usati è il filtro di Kalman. Il codice usato per implementare i filtri è stato fornito da [KL12].

7.2 OpenCV

7.2.1 Catturare lo scenario

```
VideoCapture cap = new VideoCapture(0);

if(!cap.isOpened())
{
    System.out.println("Webcam non aperta.");
}
else
{
    System.out.println("Webcam trovata.");
    return;
}

Mat frame = new Mat();
cap.retrieve(frame);

Highgui.imwrite("image.jpg", frame);
cap.release();
```

Quella che viene mostrata è la parte di codice che si occupa di catturare l'immagine dello scenario del livello tramite webcam. Per prima cosa viene creato un oggetto **VideoCapture** a cui viene passato l'id della webcam da usare. Il valore 0 sta ad indicare l'utilizzo di quella di default. Se la webcam selezionata è attiva, viene scattata un'immagine e salvata in un oggetto di

tipo `Mat` che contiene tutte le informazioni contenute in essa. Infine viene chiuso il device utilizzato.

7.2.2 Rilevamento rette

```
Mat src = new Mat();
src.create(image.size(), image.type());
Imgproc.Canny(image, src, 100, 200);
Mat lines = new Mat();
lines.create(image.size(), image.type());

Imgproc.HoughLinesP(src, lines, 1, Math.PI/270,
                     threshold, minLineSize, lineGap);

ArrayList<Tuple> linee = new ArrayList<Tuple>();

for(int j=0; j < lines.cols(); j++)
{
    double[] vec = lines.get(0, j);
    double x1 = vec[0],
           y1 = vec[1],
           x2 = vec[2],
           y2 = vec[3];
    Point pt1 = new Point(x1,y1);
    Point pt2 = new Point(x2,y2);
    linee.add(new Tuple(pt1,pt2));
}

return linee;
```

L'immagine prelevata dalla webcam viene passata come parametro all'Algoritmo di Canny insieme ai due valori soglia. L'output viene salvato in un oggetto di tipo `Mat` e utilizzato come input per l'algoritmo di Hough Lines. Le rette identificate da questo algoritmo vengono salvate in un elenco e passate alla classe che si occupa della creazione del livello.

7.3 Box2D e JavaFX

7.3.1 Creazione mondo

```
final Group root = new Group(); //Create a group for
                               holding all objects on the screen
final Scene scene = new Scene(root, WIDTH, HEIGHT, Color.
                               WHITE);

gs = new GameSetting(WIDTH, HEIGHT);
```

```

final Ball ball = new Ball;
final Paddle paddle = new Paddle;
final Edge[] lines = new Edge[num_lines];

```

Il primo passo per la creazione del mondo è creare un oggetto di tipo **Group** che conterrà tutti gli oggetti del livello e un oggetto di tipo **Scene** che sarà la scena del gioco. Per crearla gli vengono passate le dimensioni e il colore dello sfondo. Inoltre sono istanziati gli oggetti: Ball, Paddle e un array di Edge.

```

ball = new Ball(Conversion.toPosX(WIDTH/2, gs.getWidth()
()), Conversion.toPosY(560, gs.getHeight()), Color.
BISQUE, Color.BLUE);
ball.create(gs);
ball.applyLinearImpulse();

```

Successivamente per inizializzare gli oggetti è chiamato il metodo **create()**.

```

public void create(GameSetting gs){

    Circle ball = new Circle();
    ball.setRadius(radius);
    ball.setFill(gradient);

    ball.setLayoutX(Conversion.toPixelPosX(posX, gs.
        getWidth()));
    ball.setLayoutY(Conversion.toPixelPosY(posY, gs.
        getHeight()));

    ball.setCache(true);

    ballBodyDef = new BodyDef();
    ballBodyDef.type = BodyType.DYNAMIC;
    ballBodyDef.position.set(new Vec2(posX,posY));

    ballBody = gs.getWorld().createBody(ballBodyDef);

    circle.m_radius = radius * 0.1f;

    // Create a fixture for ball
    FixtureDef ballShapeDef = new FixtureDef();
    ballShapeDef.shape = circle;
    ballShapeDef.density = 1.0f;
    ballShapeDef.friction = 1.0f;
    ballShapeDef.restitution = 1.2f;
}

```

```

    ballFixture = ballBody.createFixture(ballShapeDef);
    ball.setUserData(ballBody);

    node = ball;
}

```

Tale metodo crea una palla utilizzando sia l'oggetto `Circle` di *JavaFX*, utilizzato per la parte grafica, e sia `CircleShape` di *Box2D*, utilizzato per le collisioni. All'oggetto di tipo `Circle` vengono impostate le coordinate in pixel mediante i metodi `toPixelPosX()` e `toPixelPosY()`. Per creare un oggetto palla in *Box2D*, viene creato un oggetto dinamico di tipo `BodyDef` e una fixture di tipo `FixtureDef`, alla quale vengono assegnate le caratteristiche fisiche della palla (quali per esempio la densità e la frizione).

Un tale procedimento viene ripetuto in modo analogo sia per la barretta che per le rette.

```

EventHandler<ActionEvent> ae = new EventHandler<
    ActionEvent>() {
    public void handle(ActionEvent t) {
        gs.setWorldStep(1.0f/FPS, 8, 3);
        //Move balls to the new position computed by
        JBox2D
        Body body = (Body)ball.getNode().getUserData
            ();
        float xpos = Conversion.toPixelPosX(body.
            getPosition().x, gs.getWidth());
        float ypos = Conversion.toPixelPosY(body.
            getPosition().y, gs.getHeight());
        ball.getNode().setLayoutX(xpos);
        ball.getNode().setLayoutY(ypos);

    }
};

```

Una volta aggiunti gli oggetti nel mondo, viene utilizzato un `EventHandler` per aggiornare la posizione della palla in seguito alle sue collisioni e spostamenti.

7.3.2 Ascoltatore di collisioni

```

public void beginContact(Contact c) {
    Paddle p;
    Object o = c.getFixtureA().getBody().
        getUserData();

```

```

        if(o instanceof Paddle)
        {
            p = (Paddle)o;
            p.startContact();
        }

        o = c.getFixtureB().getBody().getUserData();
        if(o instanceof Paddle)
        {
            p = (Paddle)o;
            p.startContact();
        }
    }
}

```

Il metodo `beginContact` nella classe `MyContactListener` si occupa di rilevare l'inizio di una collisione. Nel caso in cui uno dei due oggetti della collisione è di tipo `Paddle`, allora viene richiamato il metodo `startContact()` che si occupa di aggiornare il punteggio.

Capitolo 8

Test sperimentali

8.1 Rilevamento oggetti

I test per rilevamento degli oggetti si sono concentrati nell'individuare i corretti valori delle due soglie dell'algoritmo di Canny. Tali valori sono stati trovati sperimentalmente facendo variare entrambe le soglie e analizzando la correttezza dell'output ottenuto. Ponendo delle soglie troppo basse, si rischia di ottenere un output con un'elevata quantità di rumore. Al contrario ponendole troppo alte, si ha il rischio che molti bordi non vengano calcolati. La Figura 8.1 è presa in esame per l'estrazione dei bordi.

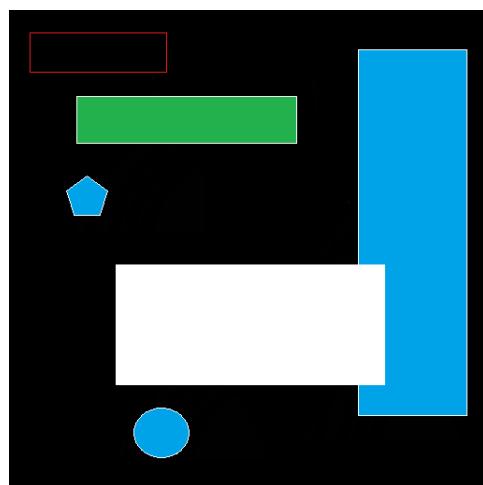


Figura 8.1: Immagine di esempio per l'estrazione dei bordi

Le immagini successive mostrano quattro diversi output ottenuti facendo variare le due soglie dell'algoritmo di Canny:

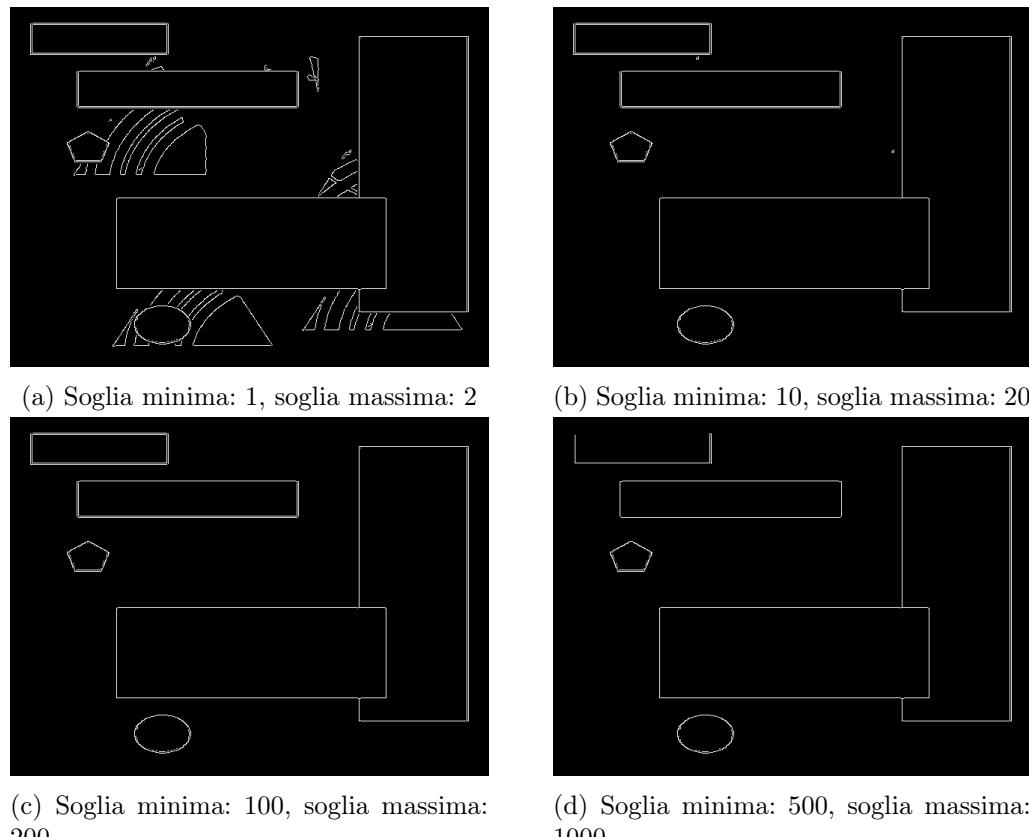


Figura 8.2: Esempi di output al variare delle soglie

I valori che hanno restituito un risultato migliore sono stati quelli in Figura 8.2c. Un ulteriore esempio di output ottenuto mediante queste soglie è il seguente.



Figura 8.3: Immagine di esempio per l'estrazione dei bordi

Al contrario della Figura 8.2 precedente, questa contiene del rumore dovuto alla presenza di linee sulla pagina posteriore del foglio.

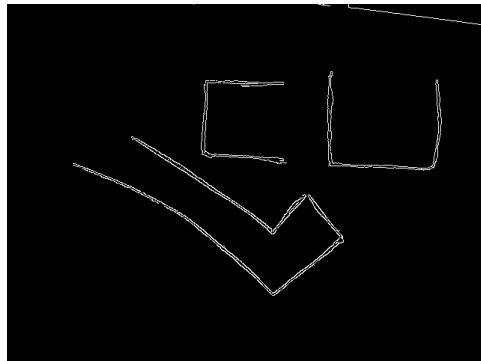


Figura 8.4: Immagine di esempio per l'estrazione dei bordi

Come è possibile vedere dalla Figura 8.4 di output, tale rumore non è calcolato e si ottiene il risultato desiderato.

8.2 Gioco

I test relativi al gioco hanno avuto come obiettivo la correttezza dell'aggiornamento dello stato del sistema durante una partita. Quindi nello specifico sono stati fatti dei test riguardanti:

- la gestione delle collisioni;
- l'aggiornamento della posizione della barretta attraverso il dispositivo Arduino;

- l'aggiornamento del punteggio solamente quando a collidere è la palla con il paddle;
- l'interruzione della partita solo nel caso che la palla cada al di là del paddle.

8.3 Proiezione livello

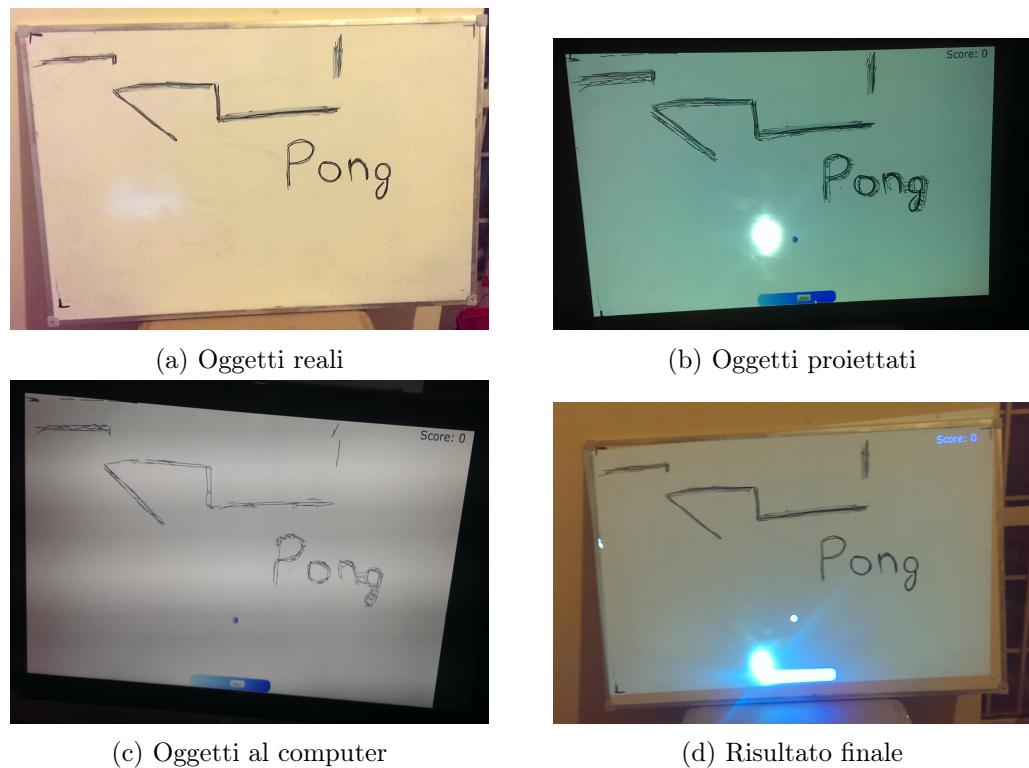
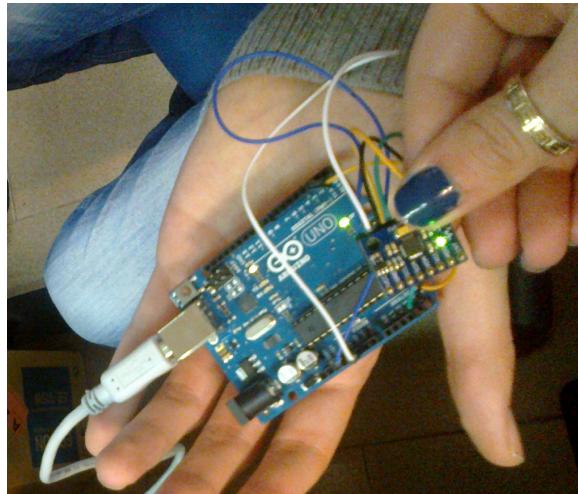


Figura 8.5: Esempi di costruzione e proiezione di un livello

Per ottenere un risultato soddisfacente di *Projection Mapping*, si proietta l'immagine nella stessa locazione in cui è stata scattata la foto. Il vero problema nasce dalla visuale differente che ha la webcam rispetto a quella del proiettore. È proprio questo il punto su cui sono stati effettuati gli ultimi test. La posizione dove mettere la webcam è stata stabilita sperimentalmente, in modo da far coincidere gli oggetti del mondo reale con i loro contorni all'interno del gioco. In questo modo si ottiene l'illusione che la palla del gioco rimbalzi sugli oggetti del mondo reale.



(a)



(b)

Figura 8.6: Strumenti utilizzati per le fasi di test

Capitolo 9

Conclusioni e sviluppi futuri

9.1 Conclusioni

L’obiettivo dell’elaborato è stato la realizzazione di un videogioco interattivo basato sui concetti di realtà virtuale e realtà aumentata e sull’uso di dispositivi di gestione control che permettano all’utente di vivere un’esperienza unica e di divenire parte integrante del gioco. Il lavoro svolto ha consentito di approfondire lo studio delle interfacce multimodali e del loro funzionamento, di scoprire nuovi mezzi per comunicare e comprendere in che modo questi siano veicolati verso l’utente e le sue necessità.

Il progetto ha favorito l’arricchimento delle conoscenze e l’autonomia nel cercare e studiare nuovi algoritmi e nuovi linguaggi al fine di poter raggiungere gli obiettivi prefissati.

Alle nozioni teoriche sono state aggiunte quelle pratiche che hanno permesso di apprendere il funzionamento e l’uso di una piattaforma di prototipazione elettronica, **Arduino**, basata su un linguaggio di programmazione mai utilizzato.

Inoltre il lavoro svolto non è stato fine a stesso, infatti è possibile migliorare non solo il gioco a livello qualitativo, ma utilizzare la tecnologia adoperata per altri possibili sviluppi futuri.

9.2 Sviluppi futuri

Si sono ipotizzati possibili sviluppi futuri, sempre concordi ai concetti di multimodalità per portare ad un miglioramento del gioco.

- Inserimento di led e vibrazione: è possibile inserire nel gioco ulteriori feedback sia visivi, come l’accensione di led in caso di collisione della palla o in caso di *game over*, sia sensoriali, come l’introduzione di un sensore che provochi una vibrazione in determinati stati del gioco.

- Grafica 3D: inserire una grafica 3D in cui la pallina possa rimbalzare e muoversi anche in profondità renderebbe il gioco ancora più reale accrescendo il senso di fusione tra gioco e giocatore. Questo sviluppo potrebbe ottenersi grazie all'uso di motori grafici 3D e l'integrazione di più webcam, poste in differenti posizioni, in modo da calibrare e riprodurre in modo esatto l'ambiente circostante.
- Integrazione con altre interfacce: oltre alla tecnologia di gestione control si potrebbe pensare a un'integrazione con altre interfacce. Utilizzare tecniche di riconoscimento vocale per muovere il paddle garantirebbe la facoltà di gioco anche a chi presenta problemi motori.
- Multiplayer: prevedere una modalità di gioco multiplayer in cui i due giocatori possano sfidarsi renderebbe il gioco più entusiasmante e più coinvolgente.
- Sviluppo di ulteriori giochi: possono essere sviluppati altri giochi con l'utilizzo di questa tecnica come ad esempio *Snake*, *flipper* e *Pacman*.

Appendice A

Algoritmi di rilevamento dei bordi

Il primo passo per rendere possibile l'interazione con gli oggetti del mondo reale è rilevare i loro bordi mediante tecniche di visione artificiale. Nello specifico ciò che viene fatto è combinare l'algoritmo di Canny con quello di Hough.

A.1 Edge Detection

L'algoritmo di Canny [Can86] si svolge in 5 passaggi:

1. smoothing;
2. finding gradients;
3. non-maximum suppression;
4. double thresholding;
5. edge tracking by hysteresis.

Smoothing Viene applicato all'immagine originale un filtro Gaussiano¹ (Figura A.1) affinchè venga ridotto il rumore presente nell'immagine preservandone la forma dei contenuti. Un esempio di filtro di questo tipo è il seguente:

$$K = \frac{1}{159} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

¹it.wikipedia.org/wiki/Filtro_gaussiano

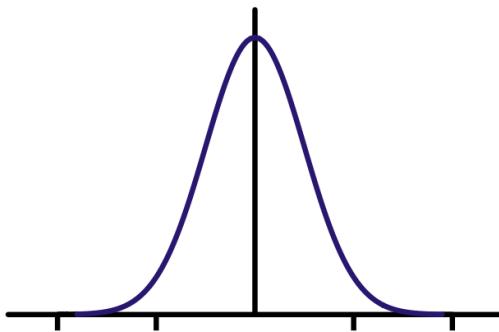


Figura A.1: La forma di un filtro gaussiano tipico

Finding gradients Così come nell'algoritmo dell'operatore di Sobel² sono applicate due maschere di convoluzione lungo le direzioni x e y .

$$\nabla_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$\nabla_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Mediane queste matrici è calcolata la direzione e l'intensità del gradiente

$$\nabla = \sqrt{\nabla_x^2 + \nabla_y^2}$$

$$\theta = \arctan \left(\frac{\nabla_y}{\nabla_x} \right)$$

²http://it.wikipedia.org/wiki/Operatore_di_Sobel

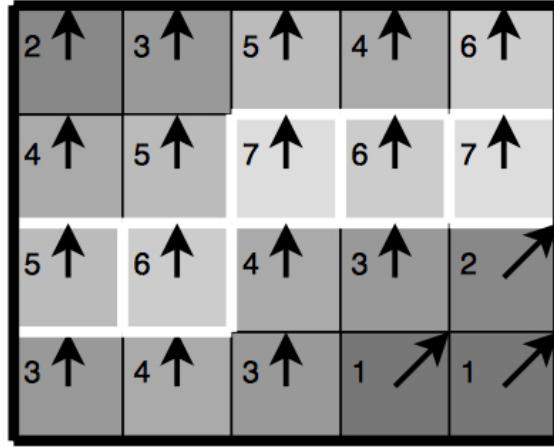


Figura A.2: Rilevamento bordi mediante il gradiente

Non-maximum suppression Con le informazioni ottenute nel passo precedente, si considerano bordi solamente quei pixel che appartengono ad un massimo locale. Ciò può essere visto facilmente in Figura A.2, dove solamente i pixel con i contorni bianchi vengono considerati appartenenti ai bordi.

Double thresholding Non tutti i pixel rimanenti appartengono realmente a dei bordi. Per questo motivo viene fissata una soglia minima e una massima. Tutti i pixel con un'intensità inferiore alla soglia minima vengono scartati; quelli con un valore maggiore della soglia massima sono marcati come pixel forti, mentre quelli compresi tra le due soglie come pixel deboli.

Edge tracking by hysteresis Ogni pixel marcato come forte viene incluso nella soluzione finale; al contrario i pixel deboli vengono inclusi solamente se sono connessi a pixel forti.

A.2 Lines Detection

A.2.1 Hough Line Transform

In questa fase è stato utilizzata la tecnica *Hough Transform*, sviluppata da Paul Hough nel 1962 in [Hou62].

La forma più semplice di *Hough transform* è *Hough line transform* ed è utile a dare una descrizione matematica delle rette che combaciano con i bordi degli oggetti presenti in un'immagine.

Il primo passo è avere un metodo per descrivere una retta, come il seguente:

$$y = mx + q \quad (\text{A.1})$$

Data l'equazione (A.1) di una retta generica, si possono considerare tutte le rette che passano per un determinato punto (x, y) . Fissato m si ricava facilmente il valore di q :

$$b = y - mx$$

Tuttavia, questo metodo non è molto utile e presenta dei problemi quando m e b crescono all'infinito. Un metodo più stabile per descrivere una retta è:

$$x \cos \theta + y \sin \theta = \rho \quad (\text{A.2})$$

Questa equazione descrive una retta che passa per il punto (x, y) che è perpendicolare alla retta che passa per l'origine e per $(\rho \cos \theta, \rho \sin \theta)$. Per ogni punto (x, y) sulla retta, θ e ρ sono costanti.

Quindi per un punto $P(x, y)$ dato, si possono ottenere le rette passanti per P risolvendo ρ e θ . Iterando su tutti i possibili valori dell'angolo θ si può trovare ρ con la formula (A.2).

A.2.2 Accumulatore

Per far sì che le rette combacino con i bordi degli oggetti nell'immagine si discretizza l'intervallo dei valori in cui è definito ρ e θ . Una volta scelta la granularità della discretizzazione si procede iterando per valore dell'angolo θ .

In particolare, per ogni angolo θ si risolve la (A.2) e si incrementa il valore associato al punto (ρ, θ) .

L'algoritmo termina restituendo i massimi locali dei valori associati ai vari punti (ρ, θ) i quali rappresentano i parametri per generare la retta cercata.

L'algoritmo A.2.1 rappresenta lo *pseudocodice* della tecnica descritta.

Algoritmo A.2.1: Hough Lines Transform

Data:

$E \leftarrow$ immagine binaria $M \times N$ dove i pixel hanno valore 1 se appartengono ad un bordo o 0 altrimenti,

$\pi \leftarrow$ soglia

$\delta\rho, \delta\theta \leftarrow$ step di campionamento per la discretizzazione

Result: $linee \leftarrow$ un elenco di linee degli oggetti

begin

$\rho \in [0, \sqrt{N^2 + M^2}]$

$\theta \in [0, \pi]$

$\rho_d, \theta_d \leftarrow$ discretizza $(\rho, \theta, \delta\rho, \delta\theta)$

$R \leftarrow \rho_d.size$

$T \leftarrow \theta_d.size$

 /* inizializzo contatore di partenza

 */

$A[R, T] \leftarrow 0$

for $i \leftarrow 1$ **to** M **do**

for $j \leftarrow 1$ **to** N **do**

$pixel \leftarrow E[i, j]$

if $pixel == 1$ **then**

for $h \leftarrow 1$ **to** T **do**

$\rho \leftarrow i \times \sin(\theta_d[h]) + j \times \cos(\theta_d[h])$

$k \leftarrow$ indice che rende minima la differenza tra

$\rho_d[k]$ e ρ

$A[k, h] = A[k, h] + 1$

 /* trovo tutti i massimi locali (k_p, h_p) , dove

$(k_p, h_p) > \pi$

 */

massimiLocali

$linee \leftarrow$ aggiungi una linea per ogni $(\rho_d[k_p], \theta_d[h_p])$

return $linee$

Bibliografia

- [Can86] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.
- [GW08] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson/Prentice Hall, 2008.
- [Hou62] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.
- [Inc13] InvenSense Inc. Mpu-6000 and mpu-6050 product specification revision 3.4. <http://invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf>, August 19 2013.
- [KL12] TKJ Electronics Kristian Lauszus. Kasbot v2 - kalman filter module. <http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it>, 2012.
- [McR13] M. McRoberts. *Beginning Arduino*. SpringerLink : Bücher. Apress, 2013.
- [Mon11] S. Monk. *Programming Arduino Getting Started with Sketches*. Tab electronics. McGraw-hill, 2011.
- [Par13] I. Parberry. *Introduction to Game Physics with Box2D*. Taylor & Francis, 2013.