POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea Magistrale

Studio di fattibilità per la realizzazione di un sistema di monitoraggio dell'attività muscolare tramite realtà aumentata

Relatore:
Marco Gazzoni

Candidato:
Francesca Martini

Correlatore:
Giacinto Cerone

A.A. 2018-2019

A mia mamma,

a mio papà,

per tutto,

per sempre

Sommario

Il sistema di prelievo elettromiografico è un esame di tipo funzionale e permette l'analisi dell'attivazione muscolare attraverso i potenziali elettrici che si sviluppano durante la contrazione. L'elettromiografia può contribuire alla diagnosi di patologie del sistema muscolare o del sistema nervoso, che compromettono il comportamento fisiologico del corpo. L'obiettivo del presente elaborato consiste nella realizzazione preliminare di un sistema atto a favorire l'analisi e lo studio dell'attività elettrica dei muscoli, tramite l'arricchimento dell'ambiente normalmente percepito con elementi virtuali. La realtà aumentata fornisce una conoscenza più approfondita dell'ambiente mediante l'inserimento di contenuti digitali, i quali corrispondono, in questa tesi, alla rappresentazione di una mappa identificativa dell'attivazione muscolare. La mappa virtuale è posta al di sopra di un marker aruco, situato sulla matrice di registrazione del segnale, ovvero un oggetto di forma quadrata, bianco e nero, riconosciuto tramite apposite librerie. L'implementazione è avvenuta mediante l'uso dell'ambiente di sviluppo Unity, il quale prevede l'utilizzo del linguaggio di programmazione orientato agli oggetti C#, e di un tool per il riconoscimento del marker, il quale inizialmente coincise con l'SDK Vuforia che fu poi sostituito dalla libreria ArUco, appartenente a OpenCV Sharp. L'architettura del sistema presenta un dispositivo EMG di prelievo, un calcolatore atto alla ricezione del segnale dalla sonda Due, eventuali dispositivi con sistema operativo android, come Smartphone e SmartGlasses, o Personal Computer. Lo scambio di dati e la ricezione del segnale emg, tra gli host coinvolti, prevedono una comunicazione basata su rete LAN, sulla quale è stato definito un protocollo utile per l'identificazione in tempo reale dei dispositivi presenti e per informare dell'avvio della trasmissione dei pacchetti. L'implementazione del codice e la verifica del riconoscimento del marker hanno evidenziato la necessità di introdurre la libreria ArUco, di OpenCV, in sostituzione a Vuforia, che fornisce un riconoscimento adeguato solo a distanze ravvicinate (5-10cm). Tale variazione ha comportato, quindi, la necessità di prediligere l'ottimizzazione del riconoscimento del marker a discapito della rappresentazione della mappa, lasciando l'opportunità di prosecuzione del progetto.

Indice

Elenco delle figure

Listings

Introduzione

Lo sviluppo tecnologico degli ultimi anni ha condotto a un miglioramento dello stile di vita delle persone, non soltanto incrementando la qualità del tempo libero ma contribuendo anche a livello lavorativo, sanitario e militare. Il potenziamento dei vari dispositivi e, allo stesso tempo, la riduzione delle loro dimensioni ha comportato una maggiore diffusione negli ambiti più diversi. Il presente elaborato è definito nell'ambito bio-ingegneristico e opera su una delle principali innovazioni degli ultimi decenni: la realtà aumentata. Essa è definita come l'introduzione di informazioni digitali all'interno dell'ambiente reale, tali contenuti possono essere di diverse tipologie come semplici elementi virtuali, animazioni o suoni. Essa compare per la prima volta nel 1968 in un lavoro di Ivan Sutherland, scienziato e informatico statunitense, il quale progettò i primi occhiali per una realtà digitalizzata. In seguito, i primi prototipi sono stati utilizzati in campo militare per lo sviluppo di visori per piloti americani, dando quindi il via negli anni '90 a innumerevoli ricerche e studi in questo ambito.

Obiettivi

Lo scopo di questa tesi è lo sviluppo di un software che, utilizzando la realtà aumentata, consente un'analisi immediata dell'attivazione muscolare. Il progetto consiste nell'implementazione di un sistema client che, ricevuti i pacchetti dati relativi all'ampiezza dell'attività muscolare, rappresenta una mappa virtuale sopra la matrice di elettrodi, adoperata per il prelievo. L'attività elettrica dei muscoli è ricavata, quindi, mediante un sistema di prelievo EMG, il quale invia i segnali registrati ad un software per l'elaborazione degli stessi. Tale software ha quindi il compito di generare i pacchetti da inviare all'applicativo client, realizzato in questo progetto. L'applicazione realizzata nel presente elaborato può essere eseguita su dispostivi diversi, purché provvisti di una fotocamera, come SmartPhone, SmartGlasses e Personal Computer. L'impiego dell'applicativo sviluppato è previsto nel percorso riabilitativo di pazienti affetti da patologie del sistema muscolare. Il trattamento di questi disturbi possono essere favoriti dall'utilizzo del presente

software, il quale restituisce una chiara raffigurazione dell'ampiezza del segnale EMG prelevato.

Struttura della tesi

La tesi è composta da 5 capitoli: i capitoli **1** e **2** rappresentano la sezione introduttiva di questo elaborato ed espongono la storia dell'arte. Il primo dei due tratta l'attivazione muscolare e i sistemi di prelievo del segnale elettromiografico. Presenta, inoltre, una sezione dedicata all'apparato muscolare e al funzionamento del biofeedback.
Il secondo capitolo approfondisce la realtà aumentata e definisce, inoltre, le altre tipologie di tecnologie che comportano una modifica dell'ambiente reale. Come è stato accennato nell'introduzione, sono innovazioni tecnologiche molto diffuse e utilizzate in applicazioni diverse. Questo capitolo espone, poi, alcuni degli ambiti di maggior utilizzo della realtà aumentata.
Nel capitolo **3** si definisce l'architettura di sistema del suddetto progetto e si approfondiscono le varie componenti software e hardware coinvolte. Tratta, inoltre, l'ambiente di sviluppo Unity, impiegato per la creazione dell'applicativo, e i tool aditi al riconoscimento del marker. Presenta brevemente l'interfaccia e i concetti fondamentali relativi a Unity e il funzionamento dei tool Vuforia e ArUco.
Il capitolo **4** definisce e approfondisce i passi implementativi che hanno condotto alla realizzazione del progetto. La struttura del capitolo è organizzata per sezioni, ognuna delle quali descrive uno step del progetto, come ad esempio il sistema di trasmissione dati.
Il capitolo **5** presenta il tool per il tracciamento dei marker introdotto in fase di sviluppo, in sostituzione di Vuforia. L'impiego di questo strumento è stato necessario per eseguire un rilevamento dei target adeguato, anche non a distanze ridotte. Ha, quindi, permesso di migliorare l'applicativo, ed in particolar modo il rilevamento del sistema di prelievo elettromiografico per mezzo di un marker.
Infine, nella **Conclusioni e Scenari Applicativi** sono definiti gli obbiettivi raggiunti, le ottimizzazioni eventualmente realizzabili e le possibili applicazioni del software sviluppato.

Capitolo 1

Attivazione Muscolare e prelievo EMG

Nel presente capitolo è possibile trovare una breve descrizione dei muscoli e del loro funzionamento; si approfondiscono, in seguito, il biofeedback e i sistemi di prelievo elettromiografici (EMG).

1.1 I muscoli

I muscoli sono organi composti da *tessuto muscolare*, ovvero un tessuto biologico che ha la capacità di contrarsi, e sono preposti al movimento del corpo umano o di alcune sue parti. In base al tipo e alla ubicazione, il muscolo conferisce mobilità allo scheletro, ad organi di senso o a strutture anatomiche. L'attività muscolare è fondamentale, non solo per permettere il movimento ma anche per mantenere le funzioni vitali come la circolazione sanguigna, la respirazione e la digestione. I muscoli sono la componente principale della massa corporea e rappresentano il 40% del corpo umano; questa percentuale varia sulla base dell'età, del genere e dal tipo di vita del soggetto. L'insieme di tali organi costituisce il sistema muscolare o apparato muscolare, il quale assieme alle ossa e alle articolazioni definisce l'apparato locomotore, che permette i movimenti, siano essi volontari o involontari.

Figura 1.1: Muscolatura Dorsale

1.1.1 Tipi di Muscoli

I muscoli possono essere classificati sulla base della loro attivazione, morfologia o funzione. Negli esseri vertebrati, è possibile distinguere i muscoli in:

- Volontari: il corpo umano ne possiede tra i 400 e i 600, sono composti da tessuto muscolare striato e sono vincolati, tramite i tendini, alle ossa; La striatura di questi organi è definita dai filamenti di actina e miosina che li compongono, i quali essendo disposti in modo regolare definiscono l'alternanza di bande chiare e bande più scure, osservabili al microscopio. Come dice il nome, sono comandati **volontariamente** dal soggetto;

- Involontari: a differenza dei precedenti, sono di natura liscia e si contraggono **involontariamente**. Inoltre, ricoprono la maggior parte delle pareti degli organi interni, come nelle arterie, nella vescica e nel tratto digestivo, e per questo sono anche chiamati *muscoli viscerali*.

A questa classificazione si esclude il muscolo cardiaco, denominato *miocardio*, che pur essendo striato appartiene al secondo gruppo, ovvero a quelli involontari.

Possono, però, anche essere distinti secondo la loro morfologia:

- Scheletrici: sono muscoli volontari e striati; Le fibre muscolari che li compongono hanno forma allungata, e il relativo citoplasma è composto da miofibrille, ovvero strutture cilindriche, che contengono i filamenti delle proteine actina e miosina, deputate alla contrazione e al rilassamento del muscolo. I fasci di filamenti delle proteine definiscono, poi, il sarcomero, il quale si accorcia quando un nervo invia degli impulsi al muscolo, sovrapponendo i filamenti di actina e miosina, e si distende al termine dello stimolo. Inoltre, le striature trasversali, presenti sulle fibre muscolari, sono piuttosto marcate, tanto che è possibile vederle ad occhio nudo.

I muscoli striati scheletrici permettono di realizzare sette movimenti differenti:

Flessione: si ha quando si avvicinano tra di loro due ossa di un'articolazione, grazie ai muscoli flessori;

Estensione: è il movimento opposto alla flessione e si ha quando si allontanano fra di loro due ossa di un'articolazione, grazie ai muscoli estensori;

Abduzione: si ha quando un arto si allontana dalla linea mediana del corpo, grazie ai muscoli abduttori;

Adduzione: è il movimento opposto dell'abduzione e si ha quando un arto si avvicina alla linea mediana del corpo, resa possibile dai muscoli adduttori;

Rotazione: si ha quando ruotiamo una parte del corpo, resa possibile dai muscoli rotatori;

Muscoli Mimici: permettono di muovere la pelle del volto per fare diverse espressioni;

Muscoli Antagonisti: muscoli che concorrono ad un movimento con azioni opposte e contemporanee.

- Cardiaci: rappresentano la componente muscolare del cuore, detta *miocardio*; Rispetto alla tipologia di muscoli precedenti possiedono, oltre alle striature trasversali, le strie intercalari, le quali sono le zone di giunzione delle fibre stesse. La sua funzione è pompare il sangue nei vasi sanguigni ed opera in maniera autonoma e ritmica. La sua cadenza è regolata dal Sistema Nervoso Autonomo o SNA: il SN simpatico aumenta il ritmo del battito, mentre quello parasimpatico lo diminuisce.
- Lisci: sono muscoli involontari, così chiamati per l'assenza di strie trasversali e, perciò, sono caratterizzati da un colore più chiaro rispetto alla muscolatura striata. Costituiscono il rivestimento interno di vari organi e si contraggono lentamente per periodi prolungati di tempo.

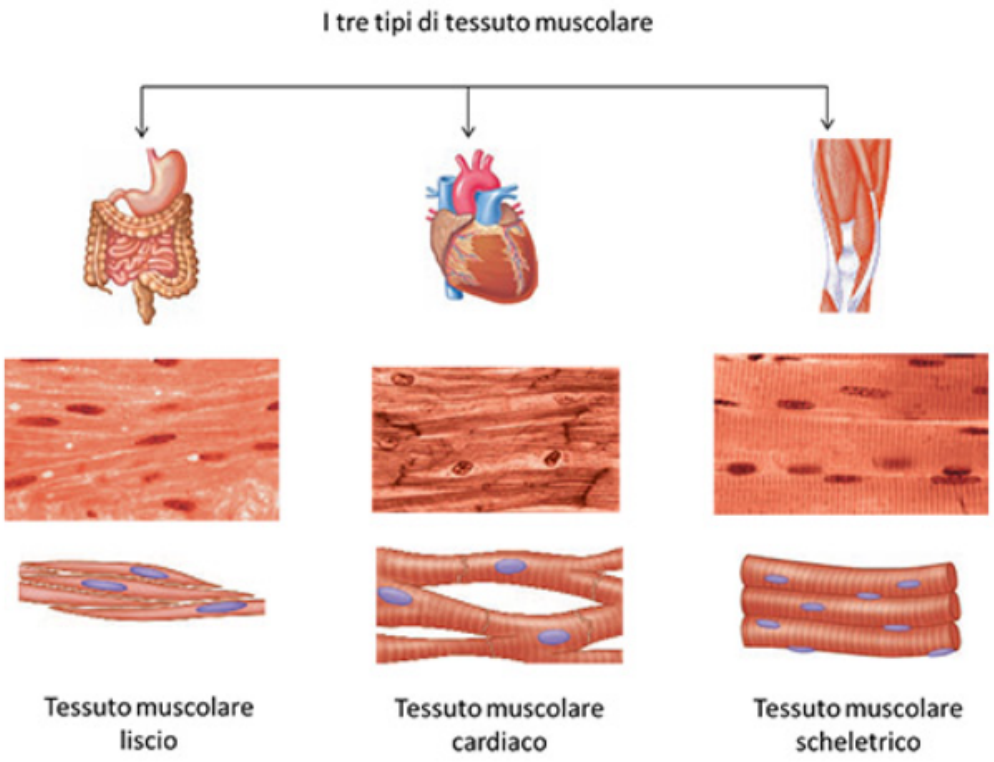

Figura 1.2: Tipi di Muscoli secondo la morfologia

Infine, i muscoli possono essere distinti anche secondo la propria funzione:

- Antagonisti: si oppongono al movimento o si rilassano. Un esempio è il bicipite rispetto al tricipite, e viceversa;
- Sinergisti: aiutano il muscolo responsabile nell'esecuzione del movimento o riducono i movimenti controproducenti;
- Fissatori: sono muscoli sinergisti specializzati.

La tabella in *Fig.* 1.3 riassume le tipologie di muscoli descritte finora:

MUSCOLO LISCIO	MUSCOLO STRIATO	
	SCHELETRICO	MIOCARDICO (cardiaco)
Involontario	Volontario	Involontario
Riveste le pareti di tutti quegli apparati devoluti alla vita vegetativa; lo troviamo nella parete dei vasi sanguigni (arterie, vene), nella parete degli organi cavi (stomaco, intestino), all'interno del globo oculare, nei muscoli erettori dei peli. La sua principale funzione è di spingere materiali dentro e fuori dal corpo.	Costituisce i muscoli scheletrici e la muscolatura di organi come il bulbo oculare e la lingua, quindi la maggior parte della muscolatura. Permette il movimento ed il mantenimento della postura; concorre a determinare le forme corporee.	E' responsabile della continua e ritmica contrattilità del cuore
E' costituito da fibre lisce, che al microscopio non presentano le striature tipiche del muscolo cardiaco o scheletrico	La particolare disposizione delle proteine contrattili conferisce al muscolo un aspetto striato, caratterizzato da striature (bande chiare e scure alternativamente ripetute); da ciò il termine Muscolo Striato.	Possiede caratteristiche funzionali e strutturali intermedie agli altri due tipi di tessuto muscolare. Per approfondire vedi: muscolo cardiaco
Contrazione molta lenta, ma prolungate e più efficiente (meno ATP richiesto).	Risponde con eccezionale velocità agli impulsi nervosi, contraendosi rapidamente ed intensamente.	
Non sono implicati nell'insorgenza della fatica muscolare.	Non possono rimanere contratti per molto tempo con elevata intensità, sono soggetti alla fatica	
Sono spesso intrinseci, e come tali, non si attaccano alle strutture scheletriche	Di norma, si collegano allo scheletro per mezzo di tendini	

Figura 1.3: Tabella classificativa dei muscoli

1.2 Attivazione Muscolare

Per *attivazione muscolare* si intende l'attività elettrica provocata dal movimento di un muscolo, indipendentemente dalla sua collocazione nel corpo. La contrazione di un muscolo inizia dalla creazione di un impulso da parte del *SNC*, *Sistema Nervoso Centrale*, tale impulso scorre sul motoneurone, o *MN*, a 60 m/s, passando da un nodo di Ranvier al successivo. Ciascun ramo del MN raggiunge, poi, una fibra muscolare in corrispondenza della giunzione neuromuscolare, che corrisponde alla zona d'innervazione dell'unità motoria. Tramite il rilascio di acetilcolina il potenziale è trasmesso, quindi, alla fibra dove si genera un nuovo potenziale di singola fibra, anche detto *Potenziale d'azione*. Tale potenziale si propaga, in entrambe le direzioni, dalla giunzione verso i tendini con una velocità di 4 m/s. Il motoneurone, assieme alle fibre muscolari da esso innervate, costituisce l'*Unità Motoria*, o UM, la quale rappresenta la più piccola unità funzionale del sistema neuromuscolare[1]. L'UM è composta da:

- il corpo cellulare del motoneurone;
- l'assone del motoneurone, che corre lungo il nervo periferico;
- la giunzione neuromuscolare tra le terminazione dell'assone e le singole fibre;
- l'insieme di fibre innervate dal MN.

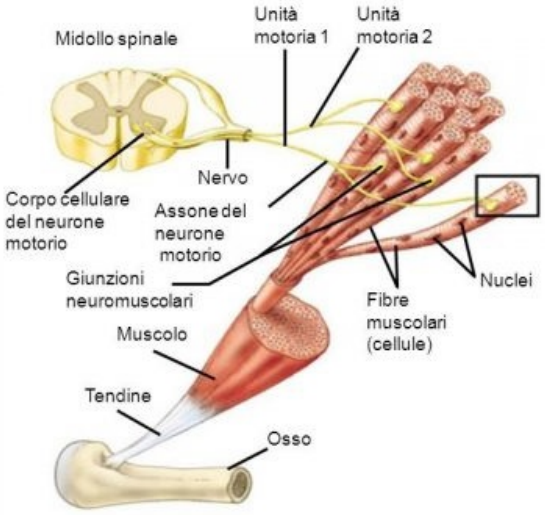

Figura 1.4: Sistema Neuromuscolare

Inoltre, il motoneurone innerva più fibre, ognuna delle quali genera un potenziale di singola fibra; la totalità dei suddetti potenziali costituisce un treno di *Potenziali d'azione dell' UM.*

Mentre, la somma dei potenziali d'azione di tutte le unità motorie prende il nome di *segnale interferente*, ed è possibile visionarlo dall'elettrodo durante l'operazione di prelievo. La contrazione delle unità motorie avviene in modo asincrono ed è definita sulla base del livello di forza che si vuole ottenere: maggiore è il livello di forza che si vuole ottenere e maggiore sarà il numero di unità motorie reclutate dal SNC; aumenta, perciò, anche il numero di fibre eccitate e, di conseguenza, i potenziali che si generano. Per cui, il livello di forza che il muscolo sviluppa è proporzionale all'ammontare di fibre che si contraggono.

1.2.1 Principio di Henneman

Le unità motorie sono reclutate dal Sistema Nervoso Centrale secondo un ordine preciso, tale procedimento è definito dal *Principio di Henneman.* Il *Principio di Henneman* definisce l'ordine di reclutamento delle UM alla contrazione di un muscolo: innanzitutto sono attivate le unità di piccole dimensione, che solitamente possiedono poche fibre a metabolismo aerobico, ed, in seguito, quelle più grandi, per una richiesta maggiore di forza, che sono anaerobiche. Durante un'attività muscolare, se è richiesto un livello di forza superiore, rispetto a quello già fornito, il SNC può reclutare altre unità motorie o aumentare la frequenza di attivazione di quelle già operanti. Questa scelta è dettata dal tipo di muscolo e dal movimento intrapreso.

Tipi di UM

Le unità motorie possiedono un'area approssimabile ad un cerchio e possono, anche, essere parzialmente sovrapposte. L'impulso da parte del SNC per le UM si basa, soprattutto, dalla tipologia di unità:

- I: sono a metabolismo aerobico e ciò comporta un minor affaticamento;
- IIA e IIB: sono reclutate per uno sforzo massimale ma si affaticano velocemente.

Le UM differiscono, inoltre, per il tipo di fibre che le costituiscono, queste possono essere rapide (FF), intermedie (FR) o lente(S).

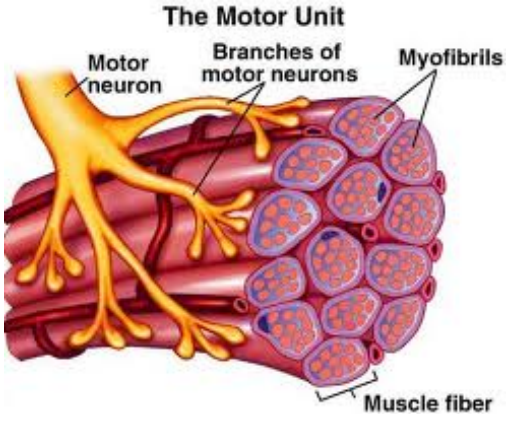

Figura 1.5: Unità motorie

1.3 Biofeedback

Il *biofeedback* è una tecnica terapeutica che ha come obiettivo l'aumento del controllo volontario[2]. Tramite precisi sistemi di strumentazione, il biofeedback permette la restituzione delle risposte fisiologiche, le quali corrispondono alle reazioni involontarie del proprio organismo. Tale procedura, in ambito riabilitativo, favorisce l'avvio, da parte di un paziente, ad intraprendere un percorso atto a migliorare il controllo delle proprie funzionalità. L'apprendimento ad autoregolarsi da parte di pazienti con pattern anomali di attivazione, i quali possono causare l'aggravarsi di disturbi fisici o psicologici già persistenti, induce i sistemi di risposta fisiologica, che presentano alterazioni, ad una situazione di equilibrio. Le apparecchiature adoperate per la misurazione e la rappresentazione delle risposte relative al funzionamento dell'organismo sono:

- degli elettrodi di registrazione;
- uno strumento di elaborazione dei segnali;
- uno monitor per la visualizzazione e il monitoraggio dei parametri.

Mentre, i parametri fisiologici più comunemente considerati sono[3]:

- la frequenza cardiaca;
- la temperatura cutanea;
- la tensione muscolare;
- la risposta psicogalvanica.

Al termine della procedura, i risultati ottenuti possono persistere senza la necessità di ricorrere ad un ulteriore intervento di *biofeedback*. I pazienti per i quali il *biofeedback* è risultato particolarmente efficace e hanno trovato maggior sollievo, sono affetti da:

- ansia;
- asma;
- disturbo dell'attenzione ed iperattività;
- dolore cronico;
- epilessia;
- incontinenza urinaria;
- insonnia;
- ipertensione;
- stress;

Gli innumerevoli benefici riscontrati, in risposta a questa tecnica, sono da attingere dai seguenti contributi:

- la persona ha un ruolo attivo nella propria guarigione;
- non è una pratica invasiva;
- sollecita le naturali reazioni di guarigione dell'organismo;
- insegna a controllare le proprie funzionalità;

1.4 Prelievo Elettromiografico

L'elettromiografia è una procedura diagnostica per valutare lo stato di salute dei muscoli e la presenza di eventuali patologie del sistema nervoso periferico, come neuropatie traumatiche o da compressione. Essa consiste nella registrazione del segnale elettrico generato durante la contrazione di un muscolo, tale segnale è proporzionale alla forza scaturita dallo stesso[4]. La sorgente del segnale EMG è il potenziale che si crea su una fibra muscolare e si propaga dal punto in cui il motoneurone raggiunge la fibra muscolare fino ai tendini. L'esecuzione di un prelievo elettromiografico è eseguita da un neurologo, su un paziente, tramite un apposito strumento denominato *elettromiografo*, esso raccoglie e registra i dati ottenuti durante l'esame.

1.4.1 Tipi di EMG

Il segnale elettrico generato da un muscolo è ricavabile da due tipologie di prelievo:

- EMG intramuscolare: è una tecnica invasiva che prevede l'infissione di un ago nel muscolo, per l'osservazione di un volume ridotto, e la distinzione dei contributi delle singole UM del segnale interferente; è prevalentemente utilizzata in fase di diagnosi sempre da parte di personale competente;
- EMG di superficie: rispetto alla precedente tecnica può essere usata anche dal paziente, restituisce una visione generale dell'attività muscolare e prevede l'utilizzo di una matrice di elettrodi da applicare sulla pelle.

La principale differenza tra queste due modalità è la possibilità, del paziente, di eseguire, o meno, l'esame presso il proprio domicilio; questa evenienza, infatti, non è permessa per la prima tecnica dell'elenco.

Inoltre, se si effettua un prelievo di superficie, rispetto a quello ad ago, tra il punto di prelievo e la sorgente del potenziale vi sono:

- tessuto muscolare;
- tessuto adiposo;
- cute;

che simulano il comportamento di un filtro passa-basso.

1.4.2 EMG di superficie

Il prelievo elettromiografico considerato in questo progetto di tesi è quello superficiale; esso consta nel posizionamento, a cavallo della zona di innervazione, di due elettrodi sulla cute in corrispondenza del muscolo da analizzare. In base alla zona di collocamento degli elettrodi è possibile distinguere due tipi di EMG di superficie[5]:

- monopolare: prevede l'utilizzo di singolo elettrodo per effettuare la misura, posto sopra il muscolo, e uno di riferimento, fuori dalla zona elettricamente attiva; si ottiene un volume di prelievo ampio, poiché sono rilevati diversi segnali elettrici nelle vicinanze del punto di prelievo, che favorisce l'insorgenza di *crosstalk*;

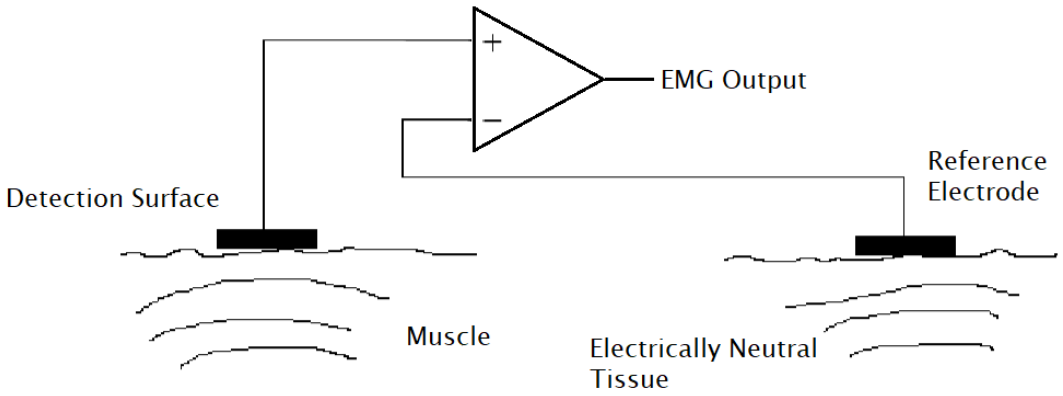

Figura 1.6: Configurazione Monopolare per il prelievo EMG

- bipolare: per la detenzione dell'attività elettrica si adopera un amplificatore differenziale con il posizionamento di due elettrodi sullo stesso muscolo, il segnale risultante è dato, quindi, dalla differenza dei potenziali ricavati dagli elettrodi.

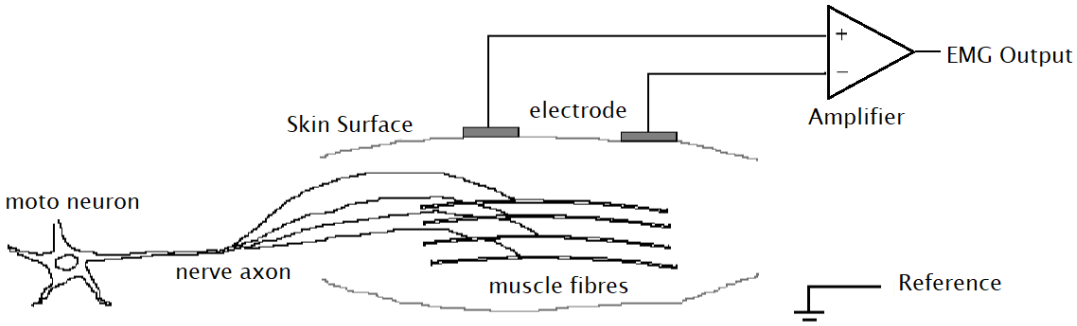

Figura 1.7: Configurazione Bipolare per il prelievo EMG

Capitolo 2

Tipi di Realtà: AR, VR e MR

> *La sensazione è che la Realtà Virtuale e quella Aumentata ci aiuteranno a dialogare meglio con l'Universo, con noi stessi, con gli altri e con le infinite possibilità del "Multiverso".*[6]

La citazione proposta è tratta dal capitolo "La Realtà Virtuale e la Realtà Aumentata in Architettura e Ingegneria" del libro "Ingegneria Elevato n – Ingegneria del Futuro o Futuro dell'Ingegneria" di Patrizia Boi e Maurizio Boi. Il libro comprende, inoltre, l'intervento di architetti specializzati ed è rivolto a coloro interessati all'evoluzione delle nuove tecnologie. Questo testo fornisce una visione del mondo odierno ed evidenzia anche le conseguenze nell'ambito della progettazione, dello sviluppo e della gestione delle opere definite dalle innovazioni tecnologiche[7].

2.1 Realtà Aumentata

Prima dei capitoli relativi alla progettazione dell'applicativo, è importante definire la Realtà Aumentata e il suo ruolo all'interno del presente progetto. Il termine Realtà Aumentata deriva dall'inglese *Augmented-Reality*, abbreviato semplicemente con AR, ed indica l'arricchimento della percezione umana, basata su i cinque sensi, tramite dati e informazioni aggiuntivi in formato digitale che sono convogliati elettronicamente.
La Realtà Aumentata permette, quindi, di potenziare le nostre possibilità con dispositivi ad alta tecnologia. Gli elementi che *aumentano* la realtà possono essere aggiunti attraverso dispositivi diversi: oltre ad un dispositivo mobile, si possono considerare gli Smart Glasses, un PC dotato di webcam o altri sensori, un dispositivo di visione (per es. occhiali a proiezione sulla retina), di ascolto (auricolari) o di manipolazione (guanti), che aggiungono informazioni multimediali alla realtà già normalmente percepita. Le informazioni aggiuntive possono consistere anche

in una diminuzione della quantità di informazioni normalmente percepibili per via sensoriale, sempre al fine di presentare una situazione più chiara, più utile o più divertente. Anche in questo caso si parla, però, di AR.

Per un maggior dettaglio, è fondamentale precisare che esistono due tipi principali di realtà aumentata:

il primo atto a dispositivi mobile, che devono essere dotati obbligatoriamente di sistema GPS (Sistema di Posizionamento Globale) e di magnetometro, deve, poi, consentire la visualizzazione di video a run-time e avere un collegamento Internet per la ricezione di dati; Il dispositivo mobile, come ad esempio uno smartphone, inquadrando l'ambiente circostante, sovrappone al mondo reale, in particolar modo a Punti di Interesse (POI) localizzati dagli elementi 3D, dei contenuti virtuali;

il secondo modello è definito per elaboratori, ed è basato sull'uso di marcatori o marker, ovvero disegni stilizzati in bianco e nero che se mostrati alla webcam, vengono riconosciuti dal computer e sovrapposti in tempo reale con contenuti multimediali: video, audio, oggetti 3D, ecc.

In entrambi i casi le informazioni circa il mondo reale, che circonda l'utente, possono diventare interattive e manipolabili digitalmente.

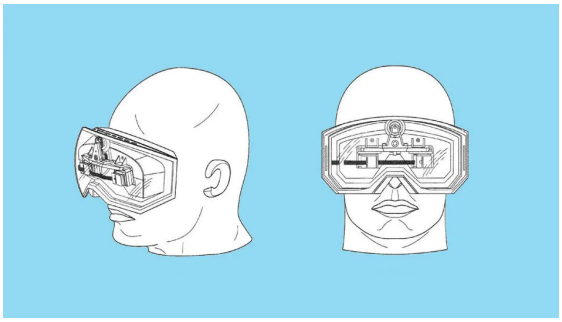

Figura 2.1: Realtà Virtuale

2.2 Realtà Virtuale

La Realtà Aumentata non coincide, però, con la Realtà Virtuale (VR). Infatti, quest'ultima crea un ambiente totalmente artificiale, e lo rende credibile avvalendosi di tecnologie che fornisco sensazioni tali da credere di trovarsi *realmente* immersi in quello scenario, come stimoli visivi, sonori e relativi agli altri sensi. Si pensi, ad esempio, ai simulatori di volo o ai visori per giochi e film in 3D.[8]

Questo totale coinvolgimento dell'utente nell'ambiente digitale creato da applicazioni di Realtà Virtuale, può causare, però, disturbi, più o meno gravi. Infatti, se una latenza eccessiva, ovvero il tempo che intercorre tra il movimento della nostra testa e l'adeguamento dell'ambiente virtuale, può *spezzare l'illusione*, generata durante l'utilizzo dei sistemi, e, al massimo, provocare nausea e vertigini(motion sickness); l'utilizzo prolungato dei supporti, quali caschi pesanti e isolanti, espongono a rischi reali.[9] Ad esempio, la permanenza prolungata in un ambiente virtuale può dissolvere il confine percettivo tra realtà e finzione, facendo si che l'utente creda di essere in grado di fare nella realtà ciò che fa agevolmente nella finzione (sport estremi per esempio), e accrescere l'asocialità nei soggetti, fino a provocare l'identificazione dell'utente con il proprio avatar.

Figura 2.2: Oculus Go

D'altro canto, la realtà virtuale potrebbe essere utilizzata nella psicoterapia delle fobie, ovviamente in modo controllato per evitare di acutizzare i timori stessi del paziente.

La realtà aumentata si avvale, invece, di quello che già esiste attorno all'utente, e che viene modificato con l'aggiunta di animazioni e contenuti digitali, ciò fa sì che l'utente non sia totalmente estraneo all'ambiente reale, e sia, di conseguenza, meno esposto a pericoli. Inoltre, i sistemi atti alla AR sono meno ingombranti e permettono libertà di movimento. Se la realtà virtuale è, dunque, una realtà artificiale, la realtà aumentata può essere definita realtà "arricchita"[8].

2.3 Realtà Mista

Tra queste due tipologie di realtà si inserisce la cosiddetta Realtà Mista, o *Mixed Reality*, la quale combina insieme la realtà fisica con quella virtuale (vedere *Fig.* 2.3). Permette di osservare il mondo reale e trarne informazioni ma, anche, di vedere e interagire con gli oggetti virtuali.

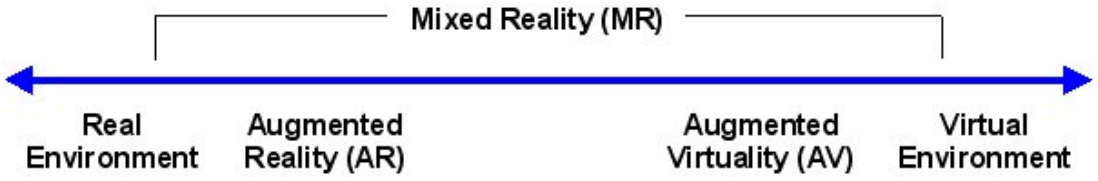

Figura 2.3: Realtà Mista

Un esempio noto di applicazione è stata progettata per Smart Glasses HoloLens di Microsoft, che presenta il mondo video ludico di Minecraft e consente di relazionarsi coi personaggi. Un ulteriore uso della realtà mista è relativo alla creazione di video utili per presentare un'applicazione virtuale e il suo contenuto: essi mostrano l'utente immerso in una realtà virtuale che interagisce con i componenti presenti. L'immagine del soggetto è catturata dal flusso video, reale, di una telecamera, mentre il mondo virtuale è quello renderizzato, e quindi generato, dal computer. In questo caso si può, perciò, dire che *realtà reale* e *realtà virtuale* coesistono.

2.4 Esempi di AR-Application

La realtà aumentata può essere utilizzata in diverse tipologie di applicazioni e soprattutto con finalità differenti; in questa sezione è possibile trovare alcuni esempi tra le più rilevanti applicazioni, andando, inoltre, a definire i principali campi di utilizzo.[10]

2.4.1 Ambiente Militare

Uno dei primi impieghi della realtà aumentata è relativo all'ambito militare, dove le nuove tecnologie possono assistere il singolo soldato. Le operazioni militari prendono forma, prevalentemente, in ambienti ostili e il soldato deve tenere traccia di molteplici informazioni, come le forze amiche e nemiche presenti, i nomi delle strade e il proprio orientamento, che rendono la missione ancora più critica. La realtà aumentata facilita la memorizzazione e l'elaborazione di tutti questi dati.[11]

Figura 2.4: Realtà Mista: utente immerso nella Realtà Virtuale

Nell'agosto del 2015, i Marines degli Stati Uniti hanno sperimentato, durante le esercitazioni con armi da fuoco, applicazioni di realtà aumentata. Il percorso di training, denominato *Augmented Team Trainer* o AITT, prevedeva l'uso di visori, sviluppati dall'Ufficio di Ricerca Navale o ONR. Esso utilizza la AR per immergere il soldato all'interno di un ambiente ostile o di conflitto e prevede, come strumentazione, un laptop, un software apposito, una fonte di alimentazione e un display sull'elmetto del soldato che potrà quindi vedere oggetti grafici sovrapposti all'immagine reale, per ricreare virtualmente uno scenario diverso.[12]

Un ulteriore esempio, in questo ambito, è l'uso di visori dotati di *head-up display*, o HUD, da parte di piloti su aerei da combattimento, che permettono di mantenere lo sguardo in avanti e allo stesso tempo di accedere ad informazioni aggiuntive, come la distanza dall'obiettivo o l'inclinazione del veicolo.[13]

2.4.2 E-Commerce

Un altro ambito che vede la diffusione di applicazioni *aumentate* è l'*e-commerce*, dove l'impiego di sistemi AR consente di personalizzare e migliorare l'esperienza di acquisto dei clienti. Diventa, perciò, possibile acquistare da sistemi mobile

o personal-computer, che simulano la compera dal vivo, facilitando la scelta del prodotto all'interno di ambienti virtuali.

Alcune grandi marche stanno, già, procedendo in questa direzione:

IKEA ha promosso la propria app *IKEA Place* che consente di vedere i mobili all'interno della propria abitazione, o in qualunque spazio ci si trovi;[14]

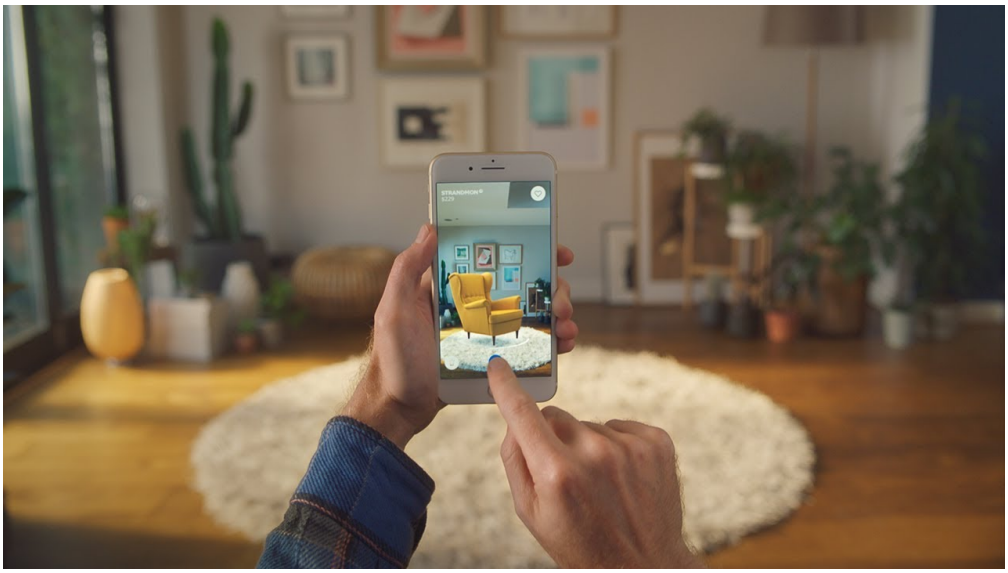

Figura 2.5: IKEA Place

Sephora, invece, ha lanciato l'app *Virtual Artist* che consente di vedersi con rossetti, ombretti e fard di tutti i tipi e colori, rendendo più facile la scelta. Taluni la preferiscono, addirittura, ad un'esperienza dal vivo[15].

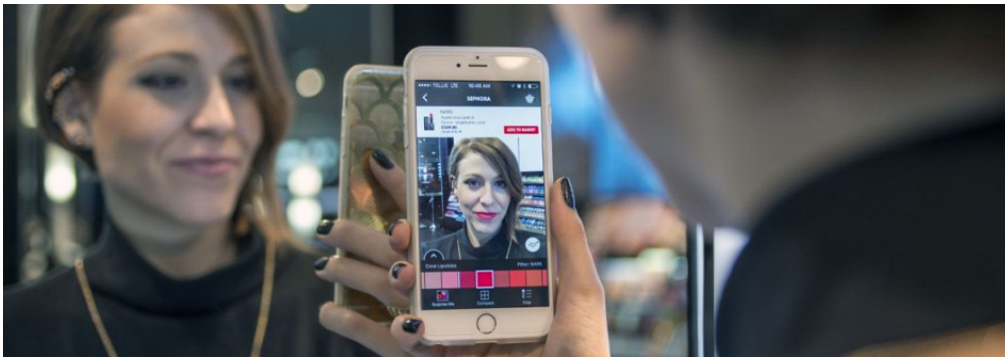

Figura 2.6: Virtual Artist di Sephora

2.4.3 Ambiente Video Ludico

L'augmented reality sta rivoluzionando, anche, il mondo dei videogiochi e sta conquistando, soprattutto, le piattaforme mobile. Una ricerca di Ericsson ha, infatti, evidenziato la crescita del settore prevalentemente in smartphone e tablet, tanto che il 26%, in media, del tempo che gli utenti spendono sullo smartphone è dedicato al gaming. La più nota applicazione in questo campo è sicuramente Pokemon GO, un gioco gratuito dedicato al famoso mondo Pokemon: è stato sviluppato per sistemi operativi Android e iOS, da Niantic, e utilizza una combinazione di realtà aumentata e geolocalizzazione con GPS. Prevede, infatti, la cattura di pokemon in base alla regione e alle condizioni atmosferiche, rendendo il gioco stimolante e mai noioso. Ha conquistato moltissimi utenti e il suo successo non sembra voler diminuire.

Figura 2.7: Pokemon GO

Questi sono solo alcuni degli ambiti nei quali la AR si sta diffondendo e sicuramente avrà una notevole crescita negli anni che seguono. Tanto che gli esperti di TechCrunch dichiarano:

> *La VR sarà grande, AR sarà più grande e richiederà più tempo. [..] Il nostro nuovo caso di base è che l'AR mobile potrebbe diventare il principale elemento di un mercato VR/AR da $108 miliardi entro il 2021 [...], con AR che fa la parte del leone con $83 miliardi e VR $25 miliardi.*

Capitolo 3

Progettazione

3.1 Architettura del Sistema

Una delle parti fondamentali di questa tesi è stata, ovviamente, la definizione dell'architettura del Sistema. L'obiettivo del presente elaborato è lo sviluppo di un software che permetta la comunicazione tra un host principale, denominato Server, e i vari dispositivi client, i quali ricevuti i pacchetti di dati rappresentano una mappa codificata in colori, identificativa dell'attivazione muscolare del soggetto. La connessione e la comunicazione tra i vari dispositivi coinvolti è realizzata mediante rete WLAN, specificando gli indirizzi IP e le porte di ingresso.

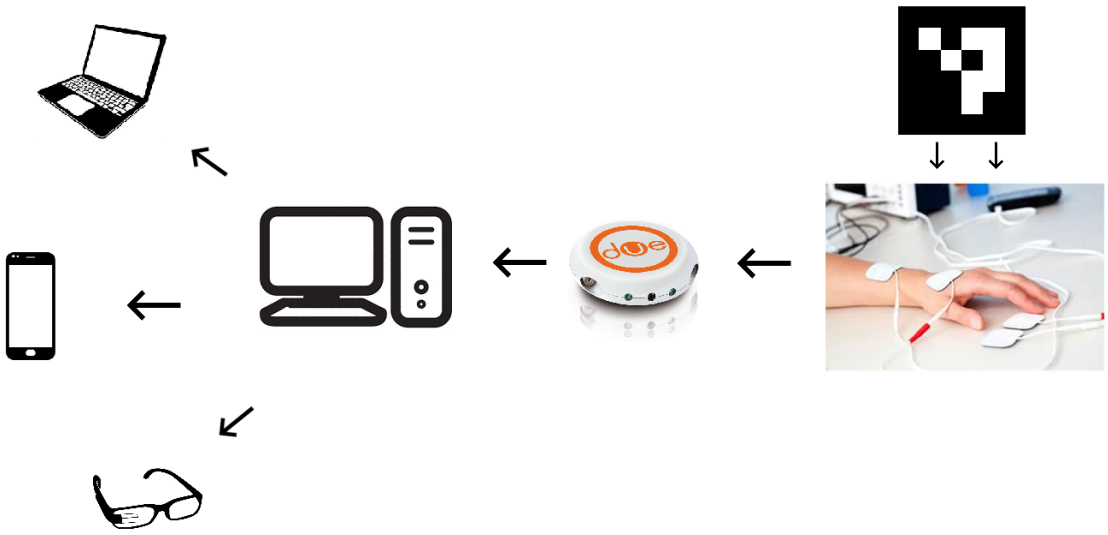

Figura 3.1: Architettura del Sistema

L'immagine in *Fig.*3.1 presenta l'architettura di sistema, i cui componenti sono introdotti nell'elenco sottostante:

- Marker: marcatore fiduciale, che viene posto sulla matrice di elettrodi per permetterne l'identificazione;
- Sistema EMG di prelievo: dispositivo indossabile, per il prelievo del segnale elettromiografico, che comunica al sistema principale i segnali elettromiografici; essi sono rilevati tramite l'impiego della sonda Due che permette un prelievo elettromiografico superficiale;
- Main System: calcolatore, denominato *Server*, atto all'elaborazione dei dati ricevuti dal dispositivo di prelievo e li invia ai sistemi per la rappresentazione;
- Client: dispositivi per consentire il riconoscimento del marker, ed in seguito la generazione e la collocazione della mappa virtuale sopra la matrice, la quale è rilevata tramite il marcatore posto su di essa; gli host *client* possono essere:
  - Portable PC: computer portatile, deve necessariamente possedere una fotocamera;
  - Sistemi Mobile: SmartPhone o Tablet con SO android, devono avere una fotocamera;
  - Smart Glasses: occhiali smart, oggetto principale della tesi.

I sistemi client proposti in elenco sono aditi alla rappresentazione della matrice e possono essere usati contemporaneamente o meno, purché almeno uno sia adoperato. Nei paragrafi seguenti si andranno ad approfondire ciascuno dei dispositivi citati in elenco.

3.1.1 Marker

L'utilizzo di un marcatore sopra il sistema di prelievo EMG è fondamentale affinché i dispositivi client identifichino dove posizionare la mappa dell'attività elettrica. Col termine *marker* ci si riferisce ad un qualsiasi oggetto target da rilevare nell'ambiente; ma si impiegano tipologie diverse di marcatori a seconda del tool usato per il *detection*. Infatti i target possono, ad esempio, coincidere con immagini o foto, come per Vuforia, oppure con marker fiduciali, come per il tool ArUco. Le applicazioni in AR operano su molteplici tipi di marcatori e le tipologie principalmente coinvolte sono i *template marker* ed i *barcode marker 2D*.

Conseguentemente, i sistemi di identificazione dei target che trovano maggior utilizzo sono relativi ai modelli di marker sopracitati:

i *matching system*, utilizzati per i *template marker*;

i *decoding system*, per i marker 2D a barre.

Gli algoritmi di *matching* rilevano soltanto il marker, mentre quelli di *decoding* decrittano i dati codificati nel marcatore. Inoltre, la tecnica di *matching* descritta necessita di un database dei marcatori da tracciare, ognuno dei quali deve possedere un numero identificativo; questa caratteristica non è prevista, invece, negli algoritmi di decodifica.

3.1.2 Il sistema di prelievo

Il dispositivo considerato per il prelievo EMG di superficie è il modello DuePro[16], il quale è stato progettato dalla ditta OT Bioelettronica in collaborazione con il lab. LISIN (Laboratory for Engineering of the Neuromuscular System) del Politecnico di Torino.

Il sistema di prelievo utilizzato è un dispositivo portatile, alimentato a batteria, che permette una connessione diretta con sistemi diversi e l'acquisizione di dati. Per acquisire i dati si utilizzano le sonde Due o DueBio, il sistema li visualizza e li registra in tempo reale per mezzo di un elaboratore. Inoltre, sfruttando l'attivazione simultanea di 7 sonde, registra fino a 14 segnali EMG bipolari a 2 segnali ausiliari (es. forza, angolo).

In questo progetto, si utilizza per il prelievo dei segnali la sonda Due[17], le cui specifiche sono definite in *Fig.*3.2. Tale sonda esegue un'elettromiografia superficiale, ovvero non invasiva e può acquisire fino a due segnali EMG di superficie attraverso due coppie di elettrodi bipolari (CDE - Elettrodi bipolari diam.15mm con connettore concentrico).

Due	
Number of sEMG signals	2
Gain	200 V/V
Bandwidth	10 ÷ 500Hz
Input noise	< 2 μV_{RMS}
CMRR	>100 dB
Sampling frequency	2048 Hz
Input Range	±8mV (RTI – Referred To Input)
A/D converter resolution	16 bits
Receiver	PC (Windows), Smartphone or Tablet (Android)
Wireless transmission protocol	Bluetooth 4.0
Power supply	3.7V LiPo battery
Charging mode	Inductive
Battery lifetime during continuous transmission	13 hours
Weight	12g
Size	21mm (Radius) x 11mm (thickness)

Figura 3.2: Specifiche Sonda Due

La sonda Due permette di collegare due coppie di elettrodi bipolari, tramite due connettori concentrici di ingresso del sistema DuePro. Volendo acquisire solamente un segnale EMG, si consiglia di utilizzare il canale 1 (ovvero il connettore accanto al LED rosso) con lo scopo di minimizzare il rumore in ingresso e ridurre l'interferenza.

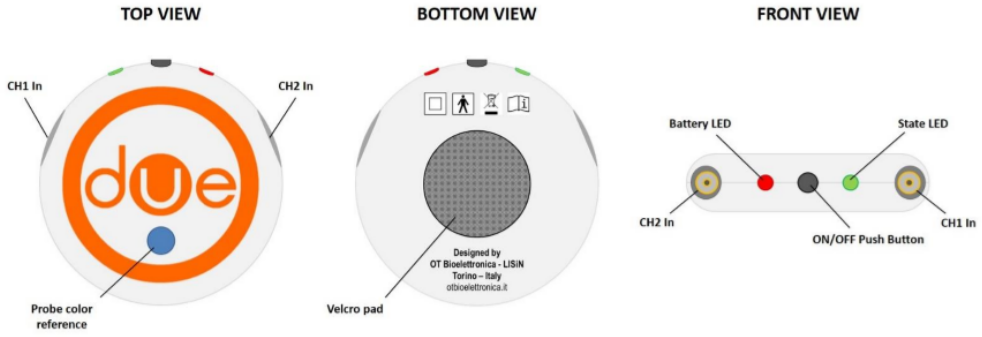

Figura 3.3: Due

La Figura 3.3 mostra la parte superiore, inferiore e frontale della sonda, nella vista dal basso è possibile, inoltre, notare un pad in velcro atto al fissaggio della sonda al soggetto. Relativamente al pacchetto dati, trasmesso dalla sonda, esso è costituito

da 4 Byte di intestazione, 64 Byte di dati e 8 Byte di terminazione. L'intestazione, o *Start Code*, è composta da una sequenza costante di 4 Byte: (253, 253, 0, 0), e identifica l'inizio di un nuovo pacchetto. I dati, invece, posti nella parte centrale del pacchetto, costituiscono la componente principale e sono formati da 32 campioni, o *Samples*, da una *word* ciascuno(2 Byte). Infine, la coda del pacchetto è composta da 7 elementi :

- SysID: definisce il tipo di sonda (Due o DueBio) e la configurazione (solo per DueBio);
- PID: rappresenta il codice identificativo della sonda;
- Ramp: è un contatore incrementale (di 2 Byte), necessario per sapere se si verifica perdita di pacchetti durante la trasmissione;
- DC1: elemento di verifica, deve corrispondere al prodotto dell'Id di sistema per l'Id della sonda;
- DC2, DC3 e DC4: Byte riservati per un eventuale uso futuro, per ora impostati a 0.

Nella Figure sottostanti, si mostra la struttura del pacchetto e le indicazioni delle varie componenti:

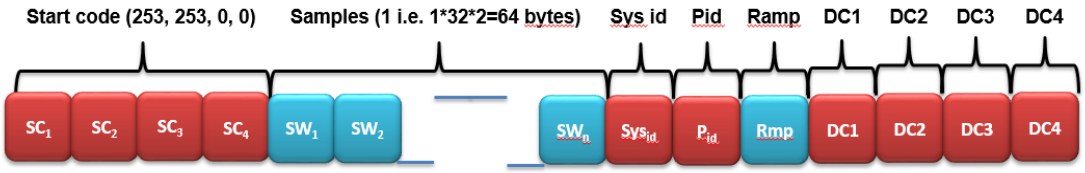

Figura 3.4: Payload della sonda Due

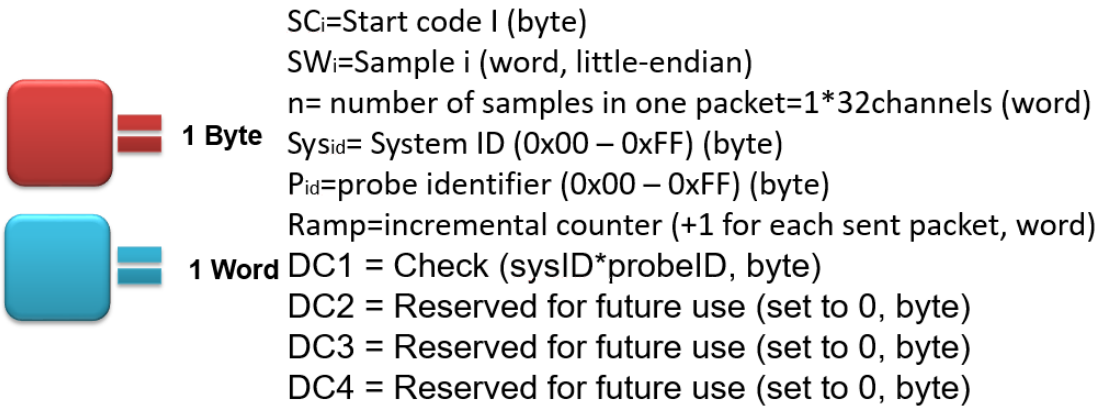

Figura 3.5: Componenti del Payload

3.1.3 Server & Client

Il server consiste in un sistema di elaborazione che ha il compito di riceve i dati del prelievo elettromiografico dalla sonda, e, una volta elaborati, inviarli, sempre tramite comunicazione WLAN, al dispositivo o ai dispositivi mobile. L'elaborazione dei dati avviene tramite un software, che corrisponde al sistema principale. Come è stato indicato in precedenza, il server può comunicare, contemporaneamente, con diversi dispositivi, identificati tramite i rispettivi indirizzi IP, e ciò comporta l'uso di un thread differente per ogni dispositivo con cui è in contatto. Per quanto riguarda, invece, i dispositivi client, le caratteristiche essenziali sono una scheda di rete (wireless o non) e una fotocamera (integrata o meno), indipendentemente dalla tipologia del dispositivo.

Quindi, una volta posizionate le matrici di elettrodi sul paziente è possibile iniziare a registrare il segnale EMG tramite la sonda Due; questa invia i segnali ottenuti dal prelievo al server che, una volta elaborati, li trasmette ai dispositivi client. Questi avranno il compito di calcolare le tensioni, sulla base dei valori di scala e livello ottenuti tramite payload, e definire, tra i valori minimi e massimi, una scala di colori. Perciò, ad ogni livello sarà associato un colore in RGB, un modello di colore basato su colori primari addittivi: Red (rosso), Green (verde) e Blue (blu).

Tra il server e i dispositivi client è stato definito un protocollo di comunicazione, che sarà però, analizzato nel capitolo 4.

3.1.4 Smart Glasses

Come è già stato chiarito precedentemente, nonostante l'architettura proposta permetta l'utilizzo di tecnologie alternative come SmartPhone e Tablet, tale elaborato si focalizza sull'utilizzo di dispositivi Smart Glasses, per attività riabilitative e di analisi, tramite la rappresentazione dell'attività muscolare del soggetto.

I dispositivi *Smart Glasses* sono occhiali "intelligenti", o più semplicemente *smart*, che aggiungono informazioni digitali a ciò che l'utente vede o lo immergono in ambienti virtuali[18]. Per ottenere la sovrapposizione di informazioni sul campo visivo, la struttura dell'occhiale prevede un display ottico montato sulla testa (OHMD) o occhiali wireless incorporati con display, HUD (Head -Up Display) o AR, che hanno la capacità di proiettare immagini[19]. Mentre i primi modelli possono svolgere attività di base, come nel caso degli smartglasses che utilizzano la tecnologia cellulare o Wi-Fi, gli occhiali intelligenti moderni sono effettivamente computer indossabili che possono eseguire autonomamente applicazioni mobile. Alcuni sono anche vivavoce e possono comunicare con Internet tramite comandi vocali in linguaggio naturale, mentre altri utilizzano i pulsanti a sfioramento.

Come altri sistemi di elaborazione, gli smartglasses possono raccogliere informazioni da sensori interni o esterni, controllare o recuperare dati da altri strumenti o computer, e supportare tecnologie wireless come Bluetooth, Wi-Fi e GPS . Però, solo un numero minore di modelli avvia un sistema operativo mobile e invia file audio e video all'utente tramite un auricolare Bluetooth o WiFi[20][21].

Moverio BT-300

Il modello adoperato (vedere *Fig.*3.6), durante la realizzazione di questo progetto, è Moverio BT-300 di Epson: sfrutta la microtecnologia di visualizzazione digitale Si-OLED (Organic Light Emitting Diode) a base di silicio, che rende il visore binoculare, a lenti trasparenti (*optical-see-through*), il più leggero sul mercato e con un'elevata qualità delle immagini. La visualizzazione in HD e l'elevata luminosità garantiscono, infatti, immagini nitide e colori vivaci[22]. Questo modello di occhiali smart è stato scelto poiché rappresenta un ottimo compromesso in termini di prestazioni/costo; inoltre rispetto agli *SmartGlasses* HoloLens forniti da Microsoft consentono un angolo di visione più ampio.

Figura 3.6: Moverio BT-300, Epson

L'area di visualizzazione non interferisce con la visione dell'utente e l'elevato rapporto di contrasto consente una fusione ottimale tra i contenuti digitali e il mondo reale. Moverio BT-300 è, inoltre, dotato, di sensori di movimento integrati nel visore stesso e nel controller.

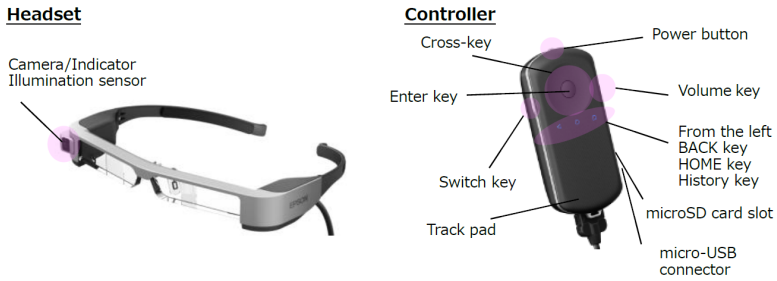

Figura 3.7: Product Appearance and Hardware Configuration

Tale sistema Moverio è definito, principalmente, da 2 componenti: un *headset* e un *controller*. L'headset presenta la camera con un indicatore di luminosità, mentre il controller ha diversi pulsanti, un trackpad, un ingresso USB e uno micro-SD. La figura sottostante approfondisce le specifiche che hanno condotto alla scelta del suddetto modello di occhiali:

Item		Specifications
System	Processor	Intel Cherry Trail, Atom x5, Quad core, 1.44 GHz
	Architecture	x86 (ABI 32-bit)
	Software	Android 5.1 API Level 22
Memory	RAM	2 GB
	Internal storage	16GB
RF	Wi-Fi	IEEE 802.11a/b/g/n/ac, Wi-Fi Direct
		Wi-Fi Miracast Sink/Source w/UIBC
	Bluetooth	Bluetooth 4.1 (Bluetooth Smart Ready certified)
Display	Resolution	1280RGB x720
	Color reproduction	24 bit color
	Screen density	mdpi
	Screen orientation	Fixed at Landscape
Codec	Still image format	BMP, JPEG, PNG, GIF
	Movie format	MP4, VP8
	Audio format	WAV, MP3, AAC
External I/F	USB Type	Micro USB Type-B, USB 2.0 (host/device)
	Vendor ID	0x17EF
	SD card	microSD, microSDHC (MAX 32 GB)
UI	Track pad	Multi-touch supported
	Buttons	Power button, HOME key, BACK key, History key, Volume key, Switch key
	Vibrations	Available
Audio I/O	Output	Stereo earphones
	Input	Microphone
Sensor	Headset	9-axis, ALS
	Controller	9-axis
Camera	Resolution	5 M pixels
GPS		Available

Figura 3.8: Specifiche Moverio BT-300

3.2 Ricerca dell'ambiente di sviluppo

Inizialmente si è effettuato uno studio per la ricerca del software più adatto per lo sviluppo del programma. Tale ricerca è stata eseguita considerando la totalità del progetto e definendo gli elementi essenziali che deve possedere il software per la sviluppo dell'applicativo.

Le caratteristiche essenziali per la scelta del software, quindi, sono:

- Riconoscimento Marker;
- Compatibilità con dispositivi Smart Glasses;
- Interfacciamento con periferiche Wifi;
- Interfacciamento con altri programmi.

I software soddisfacenti le specifiche ricercate sono:

- Android Studio (Java)
- Xamarin
- ARToolKit
- Unity
- Pikkart AR
- Wikitude SDK
- Vuforia SDK
- Moverio AR SDK
- Augumenta Studio & SDK
- SDK Android Standard.

Android Studio è un ambiente di sviluppo integrato per piattaforme Android, nonostante sia utilizzabile in ambiti diversi è più adatto a Smart Glasses Google.
Xamarin è un framework cross-platform per lo sviluppo di app native, è utilizzabile su due diversi IDE (Xamarin e Visual Studio) ma richiede una sottoscrizione gratuita a DreamSpark.
ARToolKit è una libreria open-source per la creazione di applicazioni in AR(Augmented-Reality), dove è possibile sovrapporre immagini virtuali alla visione del mondo reale.
Unity è un applicativo multi-piattaforma per la creazione di videogiochi 3D o altri contenuti interattivi.

Pikkart AR fornisce componenti diversi per il riconoscimento dei target, la creazione di versioni differenti per lo stesso marcatore, il recupero di immagini tramite Cloud e la gestione di flussi diversi di lavoro.
Wikitude SDK è la piattaforma di realtà aumentata più completa sul mercato, migliora le applicazioni con riconoscimento e tracciamento degli oggetti, permette di creare più tracker ma di attivarne uno solo per volta.
Vuforia SDK è un kit di sviluppo software di realtà aumentata (SDK) per dispositivi mobile.
Moverio SDK, ottimizzato per i device Moverio, è disponibile solamente la versione beta, questo fa si che sia poco stabile.
Augumenta Studio & SDK, compatibile con dispositivi SmartGlasses differenti, permette di inserire e manipolare i dati tramite controlli dell'interfaccia come pulsanti, manopole e cursori, ecc.
SDK Android Standard è adatto per dispositivi Moverio e compatibile con Wikitude, Vuforia e Android Studio.

Sebbene alcuni di questi applicativi siano specifici per Smart Glasses Moverio, la scelta è ricaduta su Unity con Vuforia SDK per la forte stabilità che fornisce e le molteplici e versatili operazioni che permette.

3.2.1 Unity & Vuforia SDK

Come esposto nella sezione precedente, per lo sviluppo dell'applicativo è stata preferita la combinazione del software Unity con l'SDK Vuforia.

Unity

Unity è un ambiente di sviluppo integrato a run-time, è ormai utilizzato anche in campi diversi e permette di sviluppare applicazioni cross-platform[23][24]. Inizialmente pensato per la creazione di videogiochi 3D, Unity è oggi impiegato per realizzare contenuti interattivi vari come visualizzazioni architettoniche o animazioni 3D in tempo reale. La sua popolarità è senz'altro data dalla possibilità di esportare i propri progetti su piattaforme desktop (Mac, Windows e Linux), Web, e per diversi marketplace e device (Windows Store, Windows Phone, iOS, Android, Blackberry, Wii U, PlayStation e XBOX)[25].
Questo software è composto da un motore grafico 3D, un motore fisico, componenti per l'audio, componenti per l'animazione e un IDE grafico per la gestione dei progetti[26][27]. La peculiarità di Unity è la compatibilità con svariati linguaggi di programmazione, come C#, C, C++, javascript, ecc; questo amplia ovviamente le possibilità progettuali.

L'ambiente di sviluppo visuale semplifica il processo di creazione dell'applicativo, rendendolo più veloce.

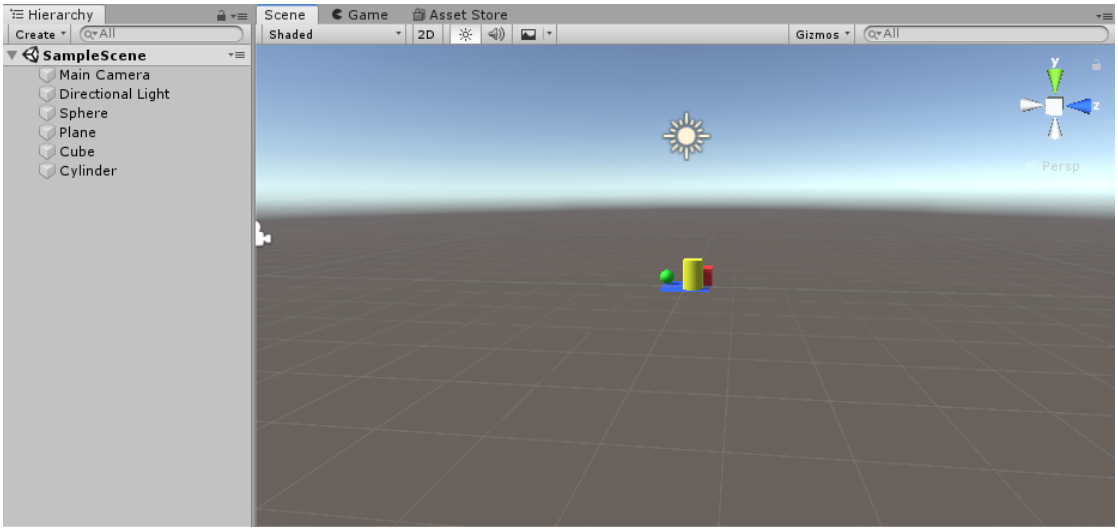

Figura 3.9: Unity - Sample Scene

Interfaccia Unity L'ambiente Unity è costituito da differenti pannelli, ognuno dei quali svolge specifiche funzioni[28]:

- *Hierarchy:* si trova di default a sinistra della finestra *Scene* e contiene l'elenco di tutti gli oggetti presenti nella scena, i quali da qui possono essere creati, per mezzo del pulsante *Create*, organizzati e selezionati. Tra questi elementi si può definire una relazione di parentela di tipo padre-figlio: occorre solamente trascinare l'oggetto che si desidera essere il figlio all'interno dell'altro. Tutte le modifiche apportate al padre, come spostamenti o scalamenti, coinvolgeranno anche il figlio.
- *Scene:* mostra la vista sulla scena su cui si sta operando e gli oggetti in essa contenuti, come telecamere, modelli 3D, ecc. Essa permette di modificare gli oggetti in relazione alla loro posizione, rotazione e scalamento. Tale pannello prevede come vista predefinita quella 3D, ma per mezzo di un bottone con l'etichetta *2D* si può passare ad una visualizzazione in 2D. Un applicativo realizzato con Unity può contenere più scene.
- *Game:* simula l'esecuzione dell'applicativo, qualunque oggetto ripreso dalla camera viene mostrato in questa finestra, che viene aperta automaticamente all'avvio.

- *Inspector:* presenta tutte le proprietà ed i componenti di un oggetto selezionato nella scena. Per aggiungere un nuovo componente ad un oggetto occorre utilizzare il pulsante *Add Component*; se si inserisce uno script di codice, come componente, questo verrà eseguito all'avvio dell'applicazione se l'oggetto, a cui è collegato, è attivo.
- *Project:* visualizza gli *assets* presenti nel progetto, come librerie, script e animazioni.
- *Console:* rende visibili i messaggi di sistema, gli errori e i warnings generati durante la compilazione.

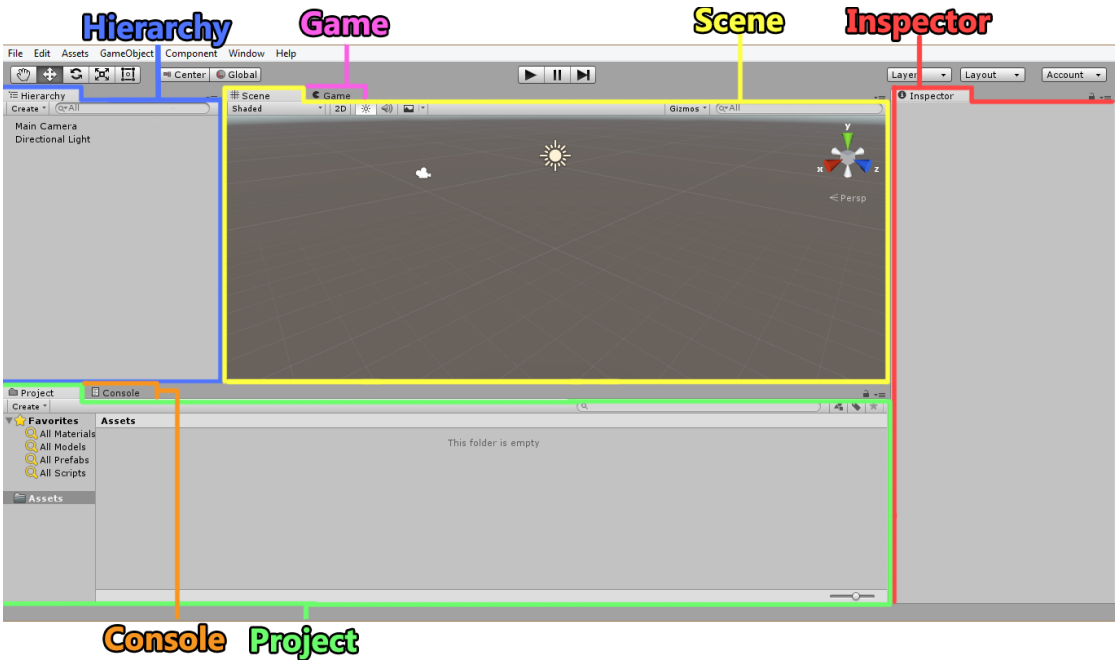

Figura 3.10: Interfaccia Unity

Concetti Fondamentali Alla creazione di un progetto Unity, il software deve fornire tutte le risorse di cui l'applicativo necessita, come scene, script o prefab; Come è stato introdotto precedentemente, una scena è un ambiente virtuale che contiene i *Game Object* inseriti e definisce un livello gerarchico dell'applicazione. Al caricamento di una scena, o al passaggio da una scena a quella successiva, il motore alloca tutte le risorse in memoria, ed eventualmente dealloca quelle non più in utilizzo[29]. I tre elementi principali in Unity sono presentati nel seguito:

Game Object, corrisponde all'elemento primario del motore; ad esso è possibile aggiungere dei *Components* per definire comportamenti e proprietà ad esso

relativi. Questa tipologia di oggetto presenta un elemento denominato *Transform*, il quale ne stabilisce la posizione, la rotazione e lo scalamento rispetto ad un sistema di riferimento (x, y, z).

Component, rappresenta le caratteristiche applicabili ad un *Game Object*; tra i componenti più utilizzati vi sono, per esempio, *Animation*, *Mesh Collider*, *Box Collider* e Script. Qualsiasi componente può essere attribuito ad un oggetto tramite il pulsante di aggiunta nell'*Inspector*.

Prefab, questo elemento esegue il salvataggio di oggetti per tipologia e crea un collegamento tra di essi, cosicché una modifica su un elemento venga apportata anche agli altri dello stesso tipo.

Inoltre, Unity consente uno sviluppo dinamico degli applicativi mediante la creazione di script ed è compatibile con diversi tool, come Visual Studio e Blender. È, quindi, necessario impostare su Unity quale tool si intende impiegare per la scrittura del codice mediante il percorso: Edit -> Preferences -> External Tools -> External Script Editor.

Vuforia Engine

Vuforia Engine, invece, è un kit di sviluppo software (SDK) per dispositivi mobili, che consente la creazione di applicazioni di realtà aumentata. Utilizza la tecnologia computer vision per riconoscere e tracciare, in tempo reale, le immagini (Image Targets) e semplici oggetti 3D. Questa sua capacità consente agli sviluppatori di posizionare e orientare oggetti virtuali, come modelli 3D e altri media, in relazione alle immagini del mondo reale, quando sono visualizzate attraverso la fotocamera di un dispositivo mobile. L'oggetto virtuale traccia, quindi, la posizione e l'orientamento dell'immagine reale, in modo che la prospettiva dell'osservatore sull'oggetto corrisponda a quella del Target Image. Rendendo, perciò, l'oggetto virtuale parte della scena del mondo reale.

L'SDK Vuforia supporta, inoltre, target diversi, 2D o 3D, compresi i target *markerless*, ovvero senza l'utilizzo di immagini adite al riconoscimento dell'ambiente reale, e le configurazioni Multi-Target, come i VuMark. Questo motore fornisce, anche, Application Programming Interfaces (API) in C++, Java, Objective-C++ (un linguaggio che utilizza una combinazione di C++ e sintassi Objective-C) e in .NET, attraverso un'estensione del motore di gioco Unity. In questo modo, l'SDK supporta sia lo sviluppo nativo per iOS e Android sia lo sviluppo di applicazioni AR in Unity che sono facilmente portabili su entrambe le piattaforme.
Le applicazioni AR sviluppate, utilizzando Vuforia, sono quindi compatibili con una vasta gamma di dispositivi mobili tra cui iPhone, iPad, smartphone e tablet con sistema operativo Android 2.2 o versioni successive e un processore ARMv6 o 7

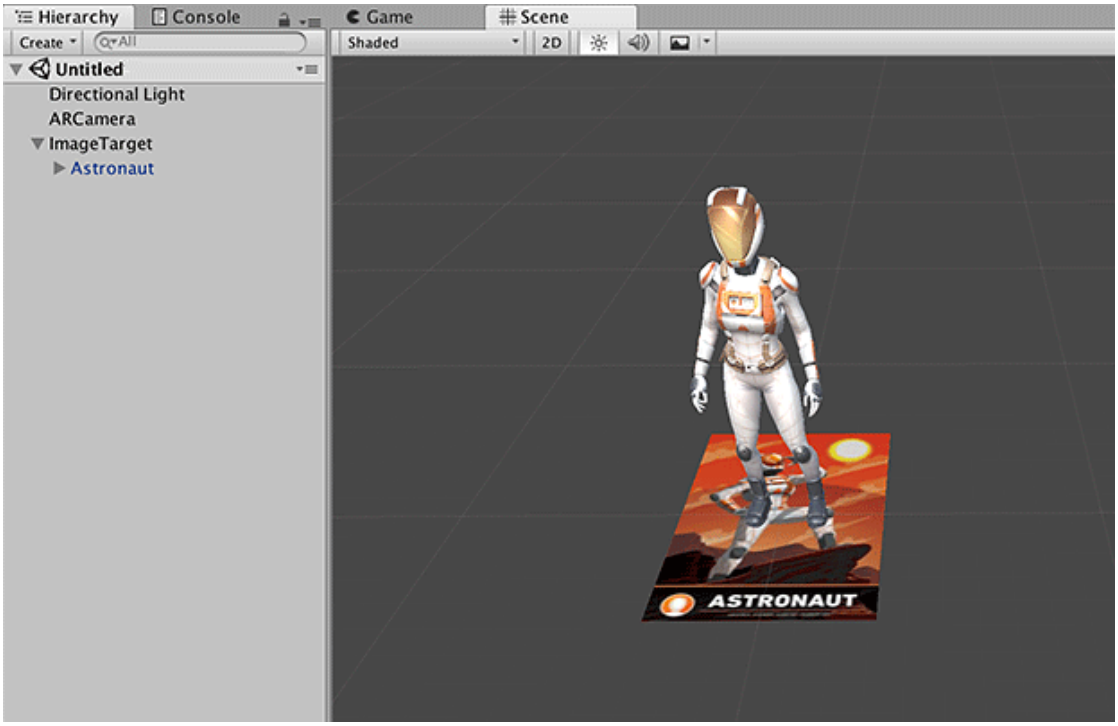

Figura 3.11: Vuforia - Esempio di utilizzo

con capacità di elaborazione FPU (Floating Point Unit). Durante la realizzazione del progetto si è, però, giunti alla conclusione che occorresse un tool aggiuntivo per migliorare il riconoscimento del marker, perciò è stato inserito ArUco che sarà approfondito nella sezione seguente.

3.2.2 ArUco

L'SDK finora analizzato presenta un inconveniente che ha condotto alla necessità di introdurre un tool più efficiente. Il difetto riscontrato consta nella difficoltà di un tracciamento valido del marker per marcatori di dimensioni ridotte. Infatti, Vuforia richiede, per un buon rilevamento, un target stampato di almeno 12 cm di altezza[30]; poichè lo scopo ultimo di questa tesi è il posizionamento di un marker su una matrice di elettrodi, è facilmente intuibile che tale dimensione non soddisfa i requisiti previsti. A tal proposito, si propone in questo paragrafo il tool ArUco, il quale fornisce un miglior rilevamento del target immagine rispetto a Vuforia. Si è, quindi, scelto di introdurre ArUco, in sostituzione al precedente motore, perché è ottimizzato anche per marcatori di dimensioni ridotte, consente un rilevamento più rapido ed è compatibile con Unity.

ArUco è una libreria Open Source, scritta in C++ e vincolata a OpenCV (*Open Source Computer Vision Library*[31]), è stata sviluppata per applicazioni in Realtà Aumentata. Essa rileva indicatori fiduciali di forma quadrata da immagini statiche o dalla ripresa video di una videocamera[32]. Esistono diversi tipi di indicatori che sono classificati in database, i quali devono essere specificati in fase di scrittura di codice per indicare alla libreria quale set ricercare nella scena. Nonostante ArUco abbia definito e consenta l'utilizzo di un proprio dizionario, può rilevare target anche appartenenti a librerie diverse come Chiltags, AprilTags, ARToolKit+.

Marker ArUco

I marcatori fiduciali di ArUco sono di forma quadrata con un bordo nero che delimita una regione interna, essa identifica un pattern binario che consente la differenziazione dei marker rendendoli univoci.

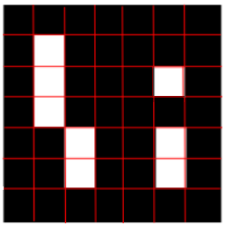

Figura 3.12: Marcatore Fiduciale

I marcatori possono essere distinti in base al numero di bit: un numero elevato diminuisce la probabilità di errore nel tracciamento, tuttavia comporta la necessità di una miglior risoluzione grafica.

Oltre ai marker fiduciali standard trovano utilizzo i marker cosiddetti *Enclosed* e le *Marker Map*. I primi sono marcatori circoscritti, ovvero marker circondati agli angoli da un ulteriore bit. Essi implicano un rilevamento leggermente più complesso ma permettono un'accuratezza migliore in termini di subpixel, cioè in termini degli elementi minimi del singolo pixel. Le mappe, invece, vengono utilizzate per ottenere un miglior rilevamento; poiché il riconoscimento di un singolo marker potrebbe fallire per motivi diversi, come una scarsa illuminazione o un movimento rapido durante la ripresa, esse sono costituite da più marker posti in posizioni note[33].

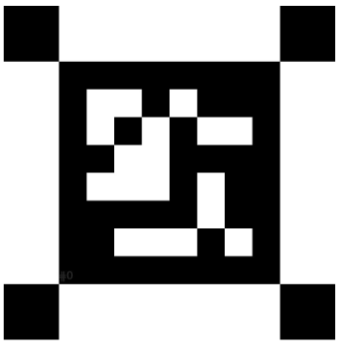

Figura 3.13: Marker Circoscritto

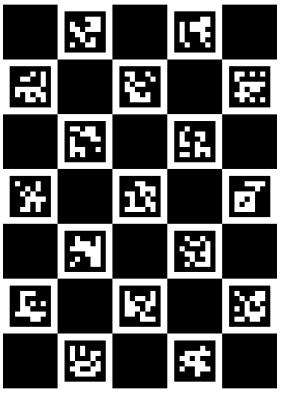

Figura 3.14: Mappa di Marker

Procedura di Rilevamento

Il processo di riconoscimento di un target ArUco prevede inizialmente la rilevazione di marcatori candidati, in questa prima fase l'immagine è, quindi, analizzata per trovare forme quadrate riconducibili a possibili marker[34].
La procedura è composta dai seguenti punti:

- Applicazione di una *Adaptive Thresholding* per ottenere i bordi: la definizione della soglia consente di distinguere i pixel interni da quelli nell'intorno: se il pixel ha un valore superiore alla soglia è definito *object pixel*, altrimenti *background pixel*;

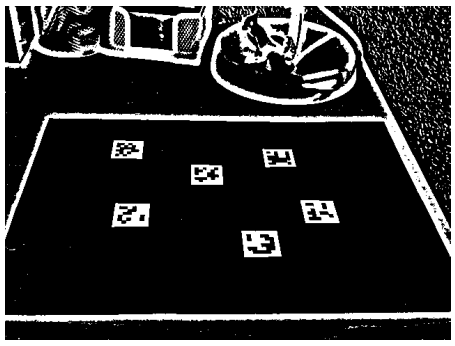

Figura 3.15: Immagine sottoposta a Adaptive Thresholding

- Rilevamento dei contorni: alcuni elementi vengono scartati, dopo essere stati estratti dall'immagine, se non sono convessi o di forma quadrata;
- Applicazione di filtri aggiuntivi atti alla rimozione di contorni troppo ravvicinati o di dimensioni eccessive (piccole o grandi).

- Analisi della codifica interna per determinare se le forme preservate finora sono marker a tutti gli effetti:

  – Estrazione dei bit dai marker: si rimuove la prospettiva di proiezione in modo da ottenere una vista frontale dell'immagine e si prosegue adoperando *Otsu*[35], metodo di sogliatura automatica dell'istogramma nelle immagini digitali, per separare i bit bianchi e neri;

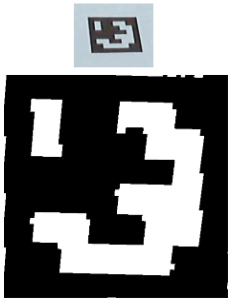

Figura 3.16: Vista Frontale del marker

  – L'immagine è suddivisa in celle, la cui dimensione è data dalla grandezza dell'indicatore e del bordo, e per ognuna di esse si calcola la quantità di pixel bianchi o neri per ricondurla a un bit bianco o nero;

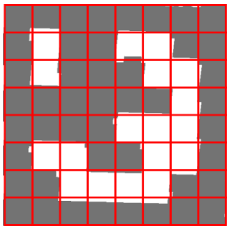

Figura 3.17: Suddivisione in celle

– Analisi dei marker per stabilire se appartengono a un dizionario.

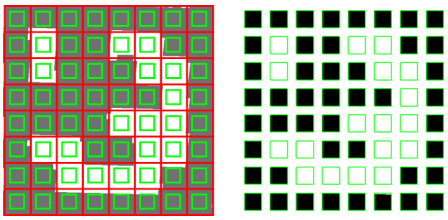

Figura 3.18: Rilevamento del colore dei bit per tracciamento

Capitolo 4

Step di Implementazione

L'implementazione dell'applicativo client è stata realizzata per step, i quali definiscono le funzionalità principali del progetto. In questo capitolo si analizzano, quindi, le differenti fasi che costituiscono la totalità del programma, presentando estratti di pseudo-codice per una maggior completezza e comprensione. Nell'immagine sottostante è presentato il modello gerarchico che descrive l'algoritmo implementato: la classe iniziale è denominata *Thread Class* e genera i due thread principali, ovvero quelli relativi al riconoscimento del marcatore e alla connessione col server.

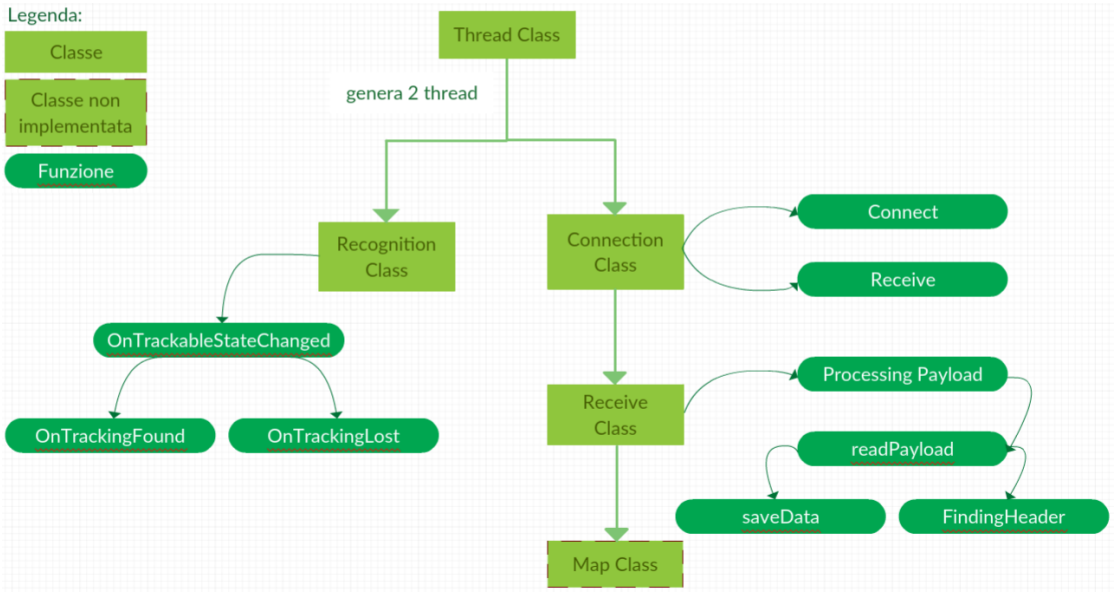

Figura 4.1: Modello gerarchico delle classi

La classe Map, rappresentata con il bordo tratteggiato, è stata implementata solo parzialmente, come già chiarito in precedenza, per privilegiare l'operazione del rilevamento del marker. Dalla figura *Fig.*4.1 è possibile intuire che ogni classe è stata realizzata su un thread differente, con lo scopo di eseguire più task parallelamente e aumentare l'efficienza.

4.1 Riconoscimento del Marker

Il task prioritario dell'elaborato è il *marker detection* che consiste nel rilevamento e riconoscimento del marker, sul quale sovrapporre la mappa relativa all'attivazione elettrica dei muscoli. Nel presente paragrafo si descrive il procedimento necessario per ottenere un sistema atto al tracciamento dei target. Esistono differenti tool per il rilevamento di marcatori e in questa fase dell'implementazione si utilizza l'SDK Vuforia, compatibile con l'ambiente di sviluppo. Vuforia, rispetto ad altri software, permette la definizione del proprio marcatore che, perciò, può essere un indicatore tradizionale, come aruco, o più in generale un'immagine personale.

Figura 4.2: Teiera Virtuale su Marker

Vuforia Engine è, infatti, in grado di rilevare e tracciare oggetti, appartenenti ai tipi disponibili nel proprio tool, chiamati *Image Target*. Essi, differentemente dai codici data matrix e dai QR, non richiedono regioni in bianco e nero per essere riconosciuti. Per l'esecuzione del tracciamento il motore agisce mediante *features*, le quali sono le caratteristiche naturali dell'immagine, confrontandole con un database di risorse di destinazione noto.

Riconosciuto il target, l'immagine è tracciata finché si trova, anche solo parzialmente, nel campo visivo della telecamera.[36]

Figura 4.3: Features del logo del Politecnico di Torino

4.1.1 Procedura di Abilitazione

L'utilizzo di Vuforia comporta la necessita di eseguire varie operazioni, oltre alla scrittura di codice. Innanzitutto, occorre creare il progetto su Unity, importare l'SDK, abilitarlo e registrare il marker sul portale di Vuforia; l'elenco sottostante definisce per punti le azioni da svolgere:

- Creazione del progetto;
- Inserimento, per eventuale diffusione, del *Company Name* e *Product Name* nelle Settings del progetto, abitualmente concordi, rispettivamente, col proprio nome e col nome del progetto;
- Definire in *Other Settings* il livello minimo API, è stato scelto Android 7.0 per questo applicativo perché è il livello minore ancora in uso, e il *Package Name* secondo la struttura
  `com.CompanyName.ProductName`;
- Abilitare Vuforia in *XR Setting*;
- Eliminare la Main Camera, camera standard di Unity, e importare la AR Camera, specifica per applicazioni in realtà aumentata;
- Proseguire sul portale Developer.Vuforia per sviluppatori:

  - Ottenere la chiave identificativa del database, tramite il bottone `Get Development Key`: definito il nome della licenza confermare e copiare la chiave restituita;
  - Nella sezione TargetManager, aprire il DB corrispondente al progetto e creare un Target mediante `Add Target`;
  - Scaricare il DB prodotto, relativo al progetto, come Unity Editor;
- Su Unity, incollare la chiave nelle impostazioni di configurazioni di Vuforia, sotto la dicitura *App License Key*
- Aggiungere nella scena, utilizzando il tasto destro del mouse, l'oggetto *ImageTarget* di Vuforia Engine;
- Aprire il menù Asset e scegliere *Import Package* per importare il DB *costum*, ottenuto dal portale Vuforia;
- Verificare, in *AR Camera/Datasets*, che il DB sia stato caricato correttamente;
- In ImageTarget, sotto *Behaviour/DataBase* selezionare il proprio database.

La procedura, eseguita su Developer.Vuforia (punto 5 dell'elenco), crea un database a cui il rilevamento del marker fa fede, di conseguenza è possibile inserire più immagini nello stesso database e il motore traccerà, contemporaneamente, tutte quelle presenti nella scena.

L'impiego di Vuforia ha permesso un riconoscimento automatico del target, ma per poter eseguire operazioni su di esso è necessario operare dinamicamente sul codice. Nonostante sia possibile verificare l'avvenuto riconoscimento del target tramite il cambio di coordinate relative all'oggetto nella scena, si è deciso di inserire un elemento *Text* a vista per operazioni di debug, che indicasse se il riconoscimento del target fosse avvenuto o meno.

4.1.2 Recognition Class

Relativamente alla scrittura di codice, si è definita una classe *Recognition Class*, che deriva dall'interfaccia `ITrackableEventHandler` per la gestione del cambiamento di stato e deve contenere un oggetto `TrackableBehaviour`, corrispondente al target presente nella scena.

Nel metodo *Start*, antecedente al *marker detection*, vengono eseguite alcune operazioni di inizializzazione tra cui la definizione di due metodi di callback, invocati alla conclusione della funzione corrente e all'interruzione dell'applicazione, del formato del frame della camera in termini di PIXEL e di un nuovo gestore degli eventi per

il *tracking*. I metodi di callback sopracitati svolgono la funzione invocata su un thread distinto rispetto a quello in esecuzione.

```
void Start()
{
    Stampa("Recognition Class");
    //Frame della camera a 3 byte per ogni pixel:
    SettaFormatoFrameCamera(PIXEL_FORMAT.RGB888);

    var vuforia = OttieniIstanza(Vuforia);

    //Metodi di Callback:
    vuforia.VuforiaStartedCallback(OnVuforiaStarted);
    vuforia.OnPauseCallback(OnPaused);

    DetectionMarkerText.text = "Detected Marker?";
    TrackableBehaviourObj = OttieniComponente();
    if (TrackableBehaviourObj != null)
    {
        TrackableBehaviourObj.TrackableEventHandler(this);
    }

    GameObject rcobj = GameObject.TrovaPerNome("ImageTarget");
}
```

Listing 4.1: Metodo Start definito in Recognition Class

I metodi `OnVuforiaStarted` e `OnPaused` impostano la modalità di focus della camera a *Continuos* mediante il seguente comando:

```
Camera.SetFocusMode(FOCUSMODE.CONTINUOUSAUTO);
```

L'interfaccia, sopracitata, comporta l'inserimento di tre metodi per indicare quali azioni eseguire al ritrovamento o alla perdita del marker.

Il primo di essi è `OnTrackableStateChanged`[37], che riceve come parametri due variabili corrispondenti allo stato precedente e attuale del target, e che in base al cambiamento di stato ridireziona alle altre 2 funzioni:

- se il marker è rilevato richiama la funzione `OnTrackingFound`;
- se, invece, viene perso interpella `OnTrackingLost`.

```
1  public void OnTrackableStateChanged(Status previousStatus, Status newStatus)
2  {
3      if (newStatus == DETECTED || newStatus == TRACKED || newStatus ==
       EXTENDED_TRACKED)
4      {
5          //Se il marker e' rilevato:
6          OnTrackingFound();
7          Stampa("Il Marker e' stato riconosciuto.");
8          Stampa("Trackable " + TrackableBehaviourObj.Name + "found");
9      }
10     else if (newStatus == NO_POSE || newStatus == LIMITED)
11     {
12         //Se il marker viene perso:
13         OnTrackingLost();
14         Stampa("Il Marker e' stato perso.");
15         Stampa("Trackable " + TrackableBehaviourObj.Name + " lost");
16     }
17 }
```

Listing 4.2: Metodo OnTrackableStateChanged in Recognition Class

I due metodi richiamati ottengono i componenti `Renderer`[38] e `Collider`[39], per eventuali impieghi futuri di rappresentazione, e utilizzano l'oggetto `Text` per mostrare a video lo stato del tracciamento.

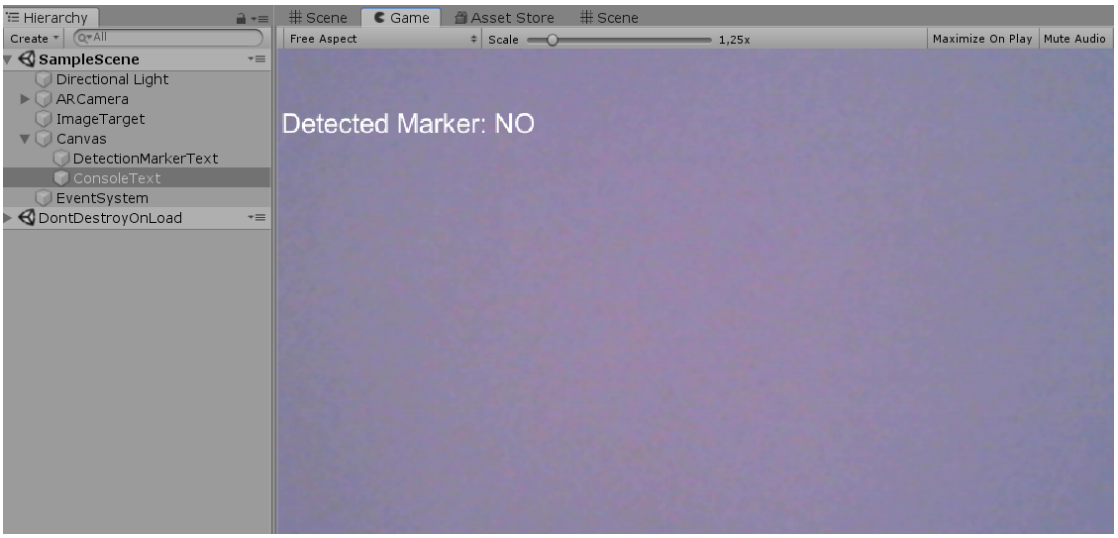

Figura 4.4: Marker Non Rilevato

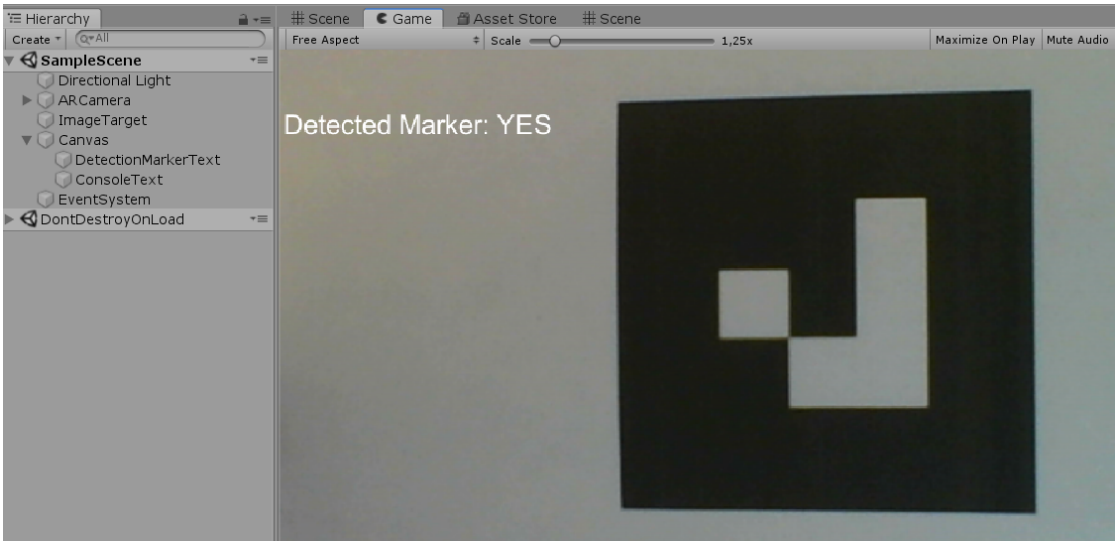

Figura 4.5: Marker Rilevato

L'elemento `Renderer` consente di far apparire l'oggetto sullo schermo, che esso sia una *mesh* o un sistema di particelle, modificando il boolean di abilitazione a *true* o renderlo invisibile con *false*. Un `Collider`, invece, definisce la forma dell'oggetto, a cui è associato, per gestire le collisioni fisiche. Nel codice sottostante si nota che tali oggetti vengono disabilitati alla scomparsa del marker, per poi essere ripristinati al ritrovamento. L'oggetto di testo, con il messaggio per l'utente, ha come padre un *Canvas* ovvero una tela appartenente all'intefaccia UI di Unity, la quale viene creata automaticamente per un qualsiasi oggetto di tipo *User Interface*.

```
1  private void OnTrackingFound()
2  {
3      List<Renderer> renderers = OttieniRenderer();
4      List<Collider> colliders = OttieniCollider();
5
6      //Rendering abilitati:
7      foreach (Renderer component in renderers)
8      {
9          AbilitaRenderer(component,true);
10     }
11
12     //Colliders abilitati:
13     foreach (Collider component in colliders)
14     {
15         AbilitaCollider(component,true);
16     }
17
18     DetectionMarkerText.text = "Detected Marker: YES";
19 }
```

Listing 4.3: Metodo OnTrackingFound in Recognition Class

```
private void OnTrackingLost()
{
    List<Renderer> renderers = OttieniRenderer();
    List<Collider> colliders = OttieniCollider();

    //Rendering disabilitati:
    foreach (Renderer component in renderers)
    {
        DisabilitaRenderer(component, false);
    }

    //Colliders disabilitati:
    foreach (Collider component in colliders)
    {
        DisabilitaCollider(component,false);
    }

    DetectionMarkerText.text = "Detected Marker: NO";
}
```

Listing 4.4: Metodo OnTrackingLost in Recognition Class

I metodi appena mostrati consentono di informare l'utente del ritrovamento e della scomparsa del target a video; nel metodo `OnTrackingFound` sarebbe, poi, stata richiamata la classe per la creazione della mappa, dell'attività elettrica, se non si fosse sostituito Vuforia.

Nel capitolo che segue è possibile trovare l'introduzione e la descrizione della libreria ArUco appartenente ad OpenCVSharp. Esso è stato introdotto per potenziare il rilevamento del target e a differenza di Vuforia lavora su marker fiduciali, e non su immagini.

4.2 Connessione col Server e scambio di dati

Il macro argomento affrontato, in questo paragrafo, è la connessione col sistema principale e il conseguente scambio di dati: eseguito il prelievo del segnale, tramite la sonda Due, questo viene inviato al server che genera i payload per gli host. Essi contengono informazioni come l'identificativo del sistema e della sonda, ed elementi di controllo (per un maggior dettaglio vedere il capitolo 3 inerente alla progettazione).

La comunicazione tra i dispositivi avviene tramite rete WLAN: il protocollo di trasmissione utilizzato è il TCP/IP (Transmission Control Protocol/Internet Protocol), il quale assicura un trasferimento affidabile dei dati. Prima di avviare la comunicazione tra i dispositivi è indispensabile eseguire la connessione tra gli stessi, ciò è attuabile specificando l'indirizzo IP, per identificare il dispositivo, e il numero di porta per realizzare una connessione univoca. Suddetto protocollo genera una

porta IP virtuale, sarà poi compito dei componenti hardware e software della rete incanalare correttamente i dati in input e output[40].

4.2.1 ConnectionClass

All'avvio della classe adita alla connessione, è richiamato in primo luogo il metodo *Connessione*[41], il quale connette un socket asincrono a un sistema di rete. Si è impiegato un socket asincrono per non rendere la comunicazione bloccante, ovvero far sì che il client inviato un messaggio non rimanga in attesa di una risposta, ma prosegua con la propria esecuzione. Per fare ciò si interpella la funzione *IniziaConnessione*, passandole come parametri:

- un EndPoint che identifica il dispositivo di rete;
- un metodo callaback per la connessione;
- un oggetto di tipo *Socket* che tiene traccia dello stato tra le chiamate asincrone.

Per quanto riguarda la creazione del *Socket* è richiesto il tipo, ad esso relativo, e il tipo di protocollo, mentre per l'EndPoint[42] servono l'indirizzo IP e il numero di porta. Al termine di questa procedura, viene restituito l'oggetto *Socket* per le successive operazioni di invio e ricezione.

```
1  private static Socket Connessione(Socket client)
2  {
3      String hostname = OttieniHostname();
4      Stampa("hostname: ", hostname);
5
6      IPAddress IP = OttieniIndirizzoIP(ipAddress)
7      Debug.Log("IP: " + IP.ToString());
8
9      // Crea un EndPoint:
10     IPEndPoint remoteEP = CreaEndPoint(IP, port);
11     Stampa("IP End Point Creato.");
12
13     // Crea un socket TCP/IP:.
14     client = CreaSocket(OttieniAddressFamily(IP), SocketType.Stream, Tcp);
15
16     client.IniziaConnessione(remoteEP, new AsyncCallback(ConnessioneCallback),
        client);
17     Aspetta();
18     Stampa("Metodo di connessione terminato");
19
20     return client;
21 }
```

Listing 4.5: Metodo Connessione in Connection Class

Il metodo per la connessione *ConnessioneCallback* viene implementato attraverso il delegato *AsyncCallback*, cosicché i risultati della funzione asincrona siano elaborati su un thread distinto,per ottenere uno sviluppo più efficiente. In questa routine si segnala al thread che la connessione è completa, impostando *ManualResetEvent*[43] su *connectDone*.

```
private static void ConnessioneCallback(IAsyncResult ar)
  {
     try
     {
        // Recupera il socket dall'oggetto di stato asincrono:
        Socket client = OttieniSocket(ar.AsyncState);

        // Completa il tentativo di connessione:
        client.TerminaConnessione(ar);

        Stampa("Socket connesso a ", client.RemoteEndPoint);


        // Segnala che la connessione e' completata:
        Set(connectDone);
        Stampa("Connessione Completata");
     }
     catch (Exception e)
     {
        Stampa("Eccezione generata");
     }
}
```

Listing 4.6: Metodo ConnessioneCallback in Connection Class

4.2.2 Protocollo di Comunicazione

Il protocollo di comunicazione rappresenta la modalità con cui il server e il client scambiano messaggi, essi sono indispensabili per ottenere informazioni riguardanti i dispositivi coinvolti. I comandi, dal server verso il client, sono numeri in esadecimale, ed ognuno di essi rappresenta una richiesta specifica. Il primo di questi identifica l'inizio della comunicazione, `0x80`, e non attende una risposta. I comandi successivi possono richiedere il nome del dispositivo o la versione del software, rispettivamente con `0x81` e `0x82`; mentre l'ultimo invio corrisponde al pacchetto, la cui struttura è definita nel capitolo di Progettazione. Il diagramma in *Fig.*4.6 descrive graficamente il flusso dei messaggi tra il server e il client, le sezioni in azzurro indicano i messaggi a video per l'utente.

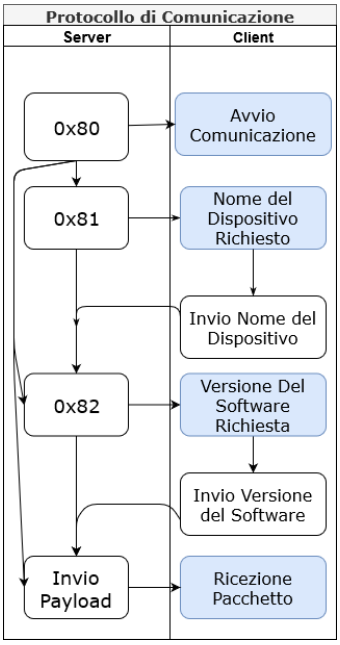

Figura 4.6: Swimlane Diagram del Protocollo di Comunicazione

4.2.3 ReceiveClass

Alla ricezione del primo comando dal server, ovvero di quello che indica l'avvio della comunicazione, è richiamata la classe *Receive Class* su un thread separato. Tale classe gestisce l'arrivo dei pacchetti e li analizza per determinare che tipo di dato contengono. Innanzitutto, verifica la lunghezza del pacchetto:

- se è grande 1 Byte lo confronta, prima, con le stringhe di comando; in seguito, se tutti i costrutti `if` falliscono, richiama il metodo di lettura del pacchetto *readPayload*;
- se è maggiore di 1 Byte invoca direttamente il metodo di lettura.

Il metodo *readPayload* restituisce un flag che definisce se si è all'interno o all'esterno del pacchetto, questa operazione è necessaria perché un payload può essere ricevuto in blocchi diversi. Se la variabile booleana, o *boolean*, *insideFlag* è *true* ci si trova all'interno del pacchetto e si prosegue nel salvataggio dei byte tramite la funzione *SalvaDati*; se, invece, è settato a *false* si è all'inizio del pacchetto e occorre appurare la correttezza dell' *header*. Come illustrato nel capitolo precedente, l'header è un'intestazione, di 4 Byte, che rappresenta l'inizio dei pacchetti, al fine di poterli distinguere l'uno dall'altro.

La funzione di lettura del payload invoca la funzione *FindingHeader*, la quale tramite un array di 4 boolean registra quali Byte dell'intestazione sono stati trovati:

- se sono stati riscontrati tutti e 4, viene settato l'*insideFlag* a *true* e si esegue la funzione di salvataggio;
- altrimenti memorizza quali Byte sono stati trovati insieme all'indice del vettore di boolean, per la prossima ricezione.

Lo pseudo-codice sottostante mostra il procedimento appena descritto:

```
private void ElaborazionePacchetto{
   try{
       while(true) {
         Inizializzo(payload);
         Ricevo(payload);
         Attesa;
         if (payload == null)
         {
            Stampa("Errore nella ricezione del pacchetto");
         }
         else if(payload.Length==1)
         {
            if(payload[0] == start_cmd)
            {
               Stampa("Ricevuto 0x80: il processo e' terminato");
               return;
            }
            else if(payload[0] == DevNameReq_cmd)
            {
               Stampa("Nome del Dispositivo richiesto")
               Invio(NomeDispositivo);
               Attesa;
            }
            else if(payload[0] == SWVerReq_cmd)
            {
               Stampa("Versione del Software richiesta");
               Invio(VersioneSoftware);
               Attesa;
            }
            else
            {
               Stampa("Lettura Pacchetto");
               readPayload(payload);
            }
         }
         else
         {
            Lettura Pacchetto;
            readPayload(payload);
         }
      }
   }
   catch(Exception e)
   {
      Stampa("Eccezione generata", e);
   }
}
```

Listing 4.7: Metodo ElaborazionePacchetto in Receive Class

I comandi in esadecimale introdotti nella sezione precedente e utilizzati in questo estratto di pseudo-codice sono:

`0x80` è lo `start_cmd`, ovvero il comando di avvio e arresto della trasmissione;

`0x81` è lo `DevNameReq_cmd`, che corrisponde al comando per la richiesta del nome del dispositivo client;

`0x82` è lo `SWVerReq_cmd`, il quale è il comando di per richiedere la versione del software in uso.

L'inserimento del costrutto `try-catch` è necessario nel caso di problemi verificatosi durante la trasmissione dei pacchetti, come ad esempio l'arresto di uno degli host coinvolti o l'assenza di connessione. Questo blocco tenta (*try*) l'esecuzione del codice e se fallisce cattura (*catch*) l'eccezione generata per informare l'utente.

```
public bool readPayload(payload)
{
   // Restituisce la lunghezza del pacchetto come numero di byte:
   int length = payload.Length;

   int ind_payload = 0;

   if(insideflag == false)
   // Inizio del pacchetto, ricerca dell'header:
   {
      TrovaHeader(payload);
      // Se si trova l'header, si e' dentro il pacchetto, salvataggio dati:
      if (TrovatoHeader())
      {
         SalvaDati(payload);
         insideflag = true;
      }
      else
      {
         if (!TrovatoPrimoByteHeader())
         {
            Stampa("Pacchetto danneggiato, headerflag non trovato.");
            return false;
         }
         else
         {
            Stampa("HeaderFlag trovato parzialmente.");
            return false;
         }
      }
   }
   // Si e' gia' all'interno del pacchetto:
   else
   {
      // Si prosegue il salvataggio:
      SalvaDati(payload);
   }
   return false;
}
```

Listing 4.8: Metodo readPayload in Receive Class

Le routine di ricezione ed invio dei dati operano analogamente al metodo *Connessione* della *ConnectionClass* (per un maggior dettaglio vedere il paragrafo precedente al *Listing* 4.5); richiamano, infatti, una funzione callback in modo asincrono per ricavare i risultati su un thread differente rispetto a quello corrente[44].

4.3 Posizionamento di un oggetto virtuale sopra il marker

Successivamente alle discussioni affrontate sulla connessione col server e sul rilevamento del marker, occorre definire come posizionare un oggetto virtuale sopra il target. La procedura di collocamento varia in base al tipo di oggetto virtuale che si utilizza; nei seguenti paragrafi, si descrivono i due procedimenti esaminati per questo progetto con l'uso di Vuforia. In entrambe le procedure rintracciato il marcatore, il motore pone automaticamente l'elemento predisposto sull'*ImageTarget*. Il capitolo successivo (*cap.* 5), invece, mostra il procedimento per la creazione e il posizionamento dell'elemento virtuale con l'impiego di ArUco.

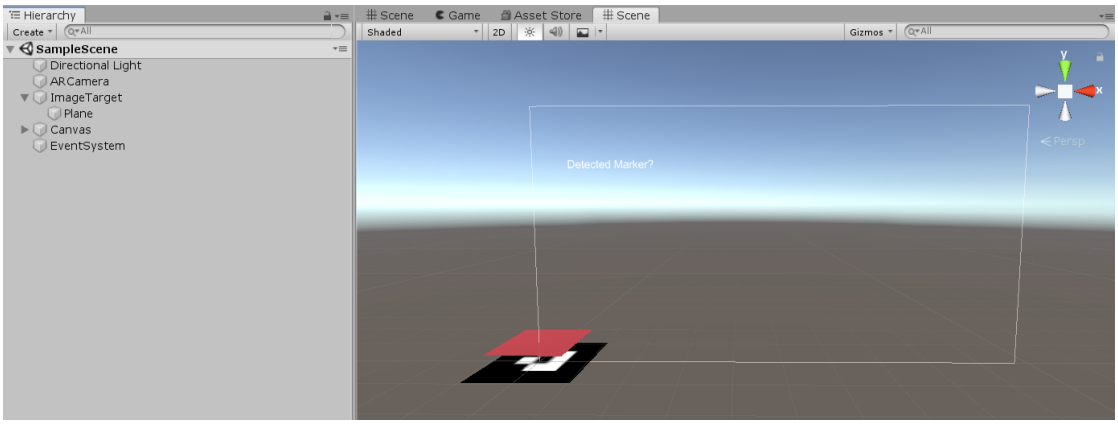

Figura 4.7: Unity: Piano sul marcatore

4.3.1 Game Object

Il primo metodo si fonda sulla creazione, dinamica o meno, di un *Game Object* di Unity nella scena: esso può avere diverse forme, come un cubo, un cilindro, una sfera o altro, ma, in questo caso, si impiega un piano.

Affinché, il motore ponga il piano sopra il marker è necessario posizionare un *Game Object* di tipo *Plane* nell'ambiente di sviluppo, alle coordinate concordi con quelle del marcatore.

Sviluppo Dinamico

Se si opta per uno sviluppo dinamico, soluzione più adatta per apportare variazioni in seguito, serve generare una classe che esegua i seguenti passi:

- creare un oggetto specificandone il tipo[45];
- porre il nuovo oggetto come figlio del target immagine;
- impostare le dimensioni del piano;
- impostare la posizione in termini di coordinate (x, y, z), concordi con quelle impostate del target immagine.

Lo script, contenente la suddetta classe, deve essere aggiunto come *Component* ad un *Game Object* come la *ARCamera*, affinché venga eseguito all'avvio dell'applicativo (per dettagli più approfonditi vedere la sottosezione 3.2.1 nel capitolo 3).

```
public bool Piano(dim)
{
    piano = creaPiano();
    if(piano == null)
    {
        Stampa("Errore nella creazione dell'oggetto");
        return false;
    }

    // Definisce una relazione gerarchica:
    ImageTarget.SettaFiglio(piano);

    // Setta la dimensione:
    SettaDim(piano, dim);

    // Ottiene la posizione del marker:
    pos = OttieniPos(ImageTarget);

    // Setta la posizione:
    SettaPos(piano, pos);

    // Modifica il colore:
    SettaColore(piano.OttieniRenderer().materiale);

    return true;
}
```

Listing 4.9: Metodo Piano per sviluppo dinamico

Lo pseudo-codice appena proposto mostra il metodo necessario per uno sviluppo dinamico, cioè tramite la scrittura di codice.

Per modificare il colore, sempre avvalendosi del codice, bisogna associare un oggetto di tipo `Renderer` al piano (per un maggior dettaglio vedere la sottosezione 4.2.3 del presente capitolo), tramite il quale si può accedere al materiale e, quindi, al colore.

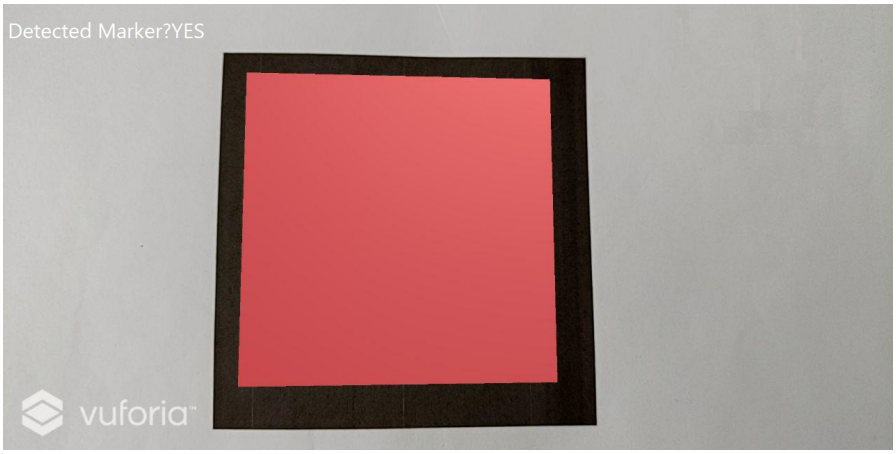

Figura 4.8: Test su SmartPhone con SO Android

Sviluppo Statico

Se, invece, si predilige non ricorrere alla scrittura di codice e operare direttamente sull'ambiente Unity, occorre creare un oggetto *Plane* nella gerarchia accanto alla *Scene*, mediante il tasto destro del mouse. Si prosegue, poi, spostando e ridimensionando il piano premendo su di esso nella scena Unity e apportando le appropriate modifiche nell'*Inspector*.

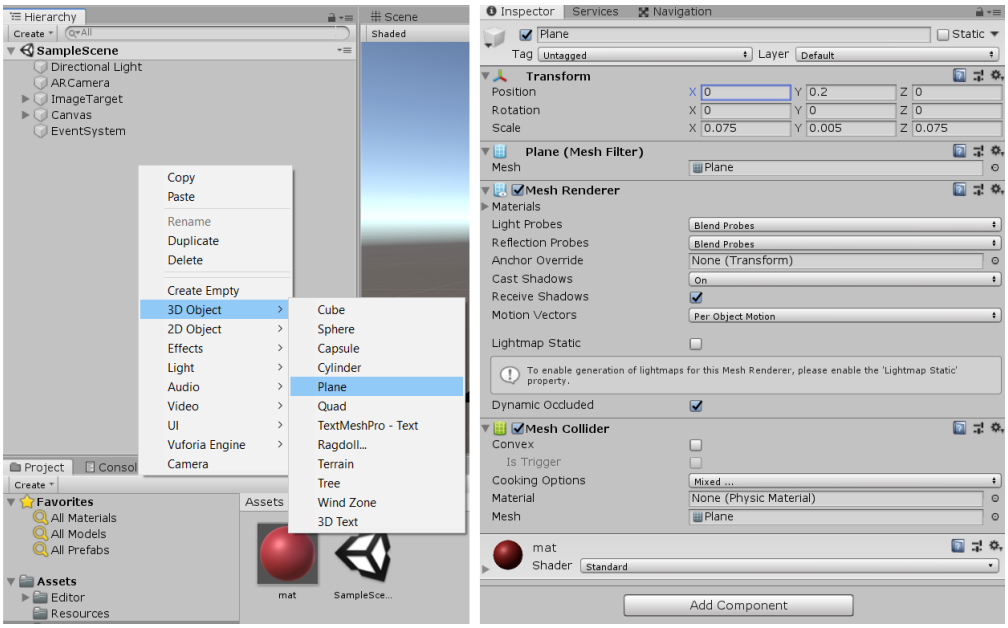

Figura 4.9: Unity: Creazione e modifica del piano

4.3.2 Mesh

Il secondo metodo permette di creare figure non preimpostate, chiamate *Mesh*, in quanto si definiscono dinamicamente le coordinate degli angoli del poligono senza dover ricorrere alle primitive di oggetti di Unity. Poiché la geometria *Mesh* approssima solamente la forma, è, solitamente, associata alle *Texture*[46]: esse sono immagini bitmap applicate sulla superficie della maglia. Tale procedura ha, quindi, il vantaggio di consentire la raffigurazione non solo di quadrati e rettangoli, ma di qualsiasi parallelogramma purché si conoscano le coordinate degli angoli. Questa geometria può contenere più vertici e molteplici array di triangoli e propone la realizzazione di una qualsiasi poligono per mezzo dell'accostamento di 2 o più triangoli[47].

Perciò, per creare una *Mesh* è necessario:

- assegnare i vertici del poligono;
- assegnare i triangoli costituenti la maglia.

La *Fig.* 4.10 mostra la realizzazione di un quadrato tramite l'uso di tale geometria: La figura mostra i vertici dei triangoli con le rispettive coordinate in (x, y).

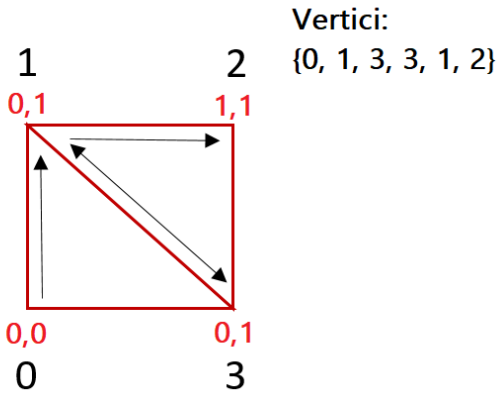

Figura 4.10: Quadrato generato dall'accostamento di due triangoli

Nel vettore, di vertici della maglia, i vertici sono definiti in senso orario; per cui:

il vertice con coordinate (0, 0) ha indice 0;

il vertice con coordinate (0, 1) ha indice 1;

il vertice con coordinate (1, 1) ha indice 2;

il vertice con coordinate (0, 1) ha indice 3.

I vertici sono, poi, richiamati per essere collegati da una retta e costituire effettivamente due triangoli, di cui: il primo è costituito dai vertici con indici 1, 0 e 3, mentre il secondo dai vertici con indici 3, 1 e 2. L'ordine con cui questi sono invocati nel vettore non è invariabile e può essere modificato; l'essenziale, però, è che non si escludano dei punti, lasciandoli vacanti.

Relativamente allo sviluppo, si procede creando un *Game Object* vuoto come figlio dell'*ImageTarget*. Come è stato descritto nella sottosezione precedente, tale azione può essere dinamica o no, nel primo caso si utilizza il metodo per creare l'oggetto, nel secondo si richiama il metodo per identificarlo nella scena.

```
public bool Piano(dim)
{
    piano = TrovaPiano("Plane");
    if(piano == null)
    {
        Stampa("Errore nella creazione dell'oggetto");
        return false;
    }


    // Ottiene la posizione del marker:
    pos = OttieniPos(ImageTarget);

    piano = CreaPoligono(piano, ImageTarget, pos);

    // Modifica il colore:
    SettaColore(piano.OttieniRenderer().materiale);

    return true;
}
```

Listing 4.10: Metodo Piano per sviluppo dinamico della Mesh

Nello pseudo-codice soprastante, si cerca l'oggetto nella scena e lo si passa alla funzione atta alla rappresentazione. Rammentando che l'obbiettivo è la creazione di un piano, alla funzione *CreaPoligono* si passa, come parametro, il centro del marker ma in essa verranno, poi, calcolate le coordinate dei quattro punti, coincidenti con gli angoli. Il metodo *CreaPoligono* è visionabile nella pagina seguente, al *Listing.* 4.11.

```
public GameObject CreaPoligono(piano, ImageTarget, pos)
{

    if(!piano.Possiede(MeshFilter) && !piano.Possiede(MeshRenderer))
    {
        piano.Aggiungi(MeshFilter);
        piano.Aggiungi(MeshRenderer);
    }

    l = OttieniLarghezza(piano);
    a = OttieniAltezza(piano);

    Mesh.vertici =
    {
        (pos.x - l/2, pos.y . a/2),             //Basso - Sinistra
        (pos.x - l/2, pos.y + a/2),             //Alto - Sinistra
        (pos.x + l/2, pos.y + a/2),             //Alto - Destra
        (pos.x + l/2, pos.y - a/2),             //Basso - Destra
    };

    Mesh.triangoli = new int[] { 0, 1, 3, 3, 1, 2 };

    MeshFilter.Mesh = Mesh;

    return piano;
}
```

Listing 4.11: Metodo CreaPoligono richiamato in Piano

Come proposto nel codice, la realizzazione di una *Mesh* richiede due elementi *Mesh Components*[48]:

- il primo è il *Mesh Filter*, esso definisce la forma dell'oggetto tramite le coordinate dei vertici[49];
- il *Mesh Renderer*, invece, riceve la *Mesh*, che il *Mesh Filter* ha ricavato dagli assets, per il rendering sullo schermo[50].

Quindi, un qualsiasi poligono, generato da una maglia, prende forma dall'unione di più triangoli; il comando `Mesh.vertici{...}` del codice dà vita a un insieme di punti, che però necessitano di essere collegati tra loro per costruire una forma. Tale forma, risulta quasi immediata se è un triangolo, altrimenti, come già indicato, si accostano più triangoli; la figura sottostante mostra alcuni esempi di sagome realizzabili con questa metodologia.

Figura 4.11: Alcuni esempi di forme realizzabili tramite Mesh

La soluzione *Mesh*, rispetto a quella implementata con un *Game Object*, è sicuramente più flessibile poiché non limita la rappresentazione a figure standard dell'ambiente Unity, come i piani e le sfere. D'altro canto necessita della scrittura di codice per la definizione dei triangoli, mentre la prima proposta è facilmente realizzabile nella scena Unity. In fase di testing delle due soluzioni è stato rilevato che entrambe sono eseguite correttamente e raffigurano gli oggetti richiesti. Relativamente alle prestazioni non occupano memoria, se non la minima indispensabile per l'esecuzione.

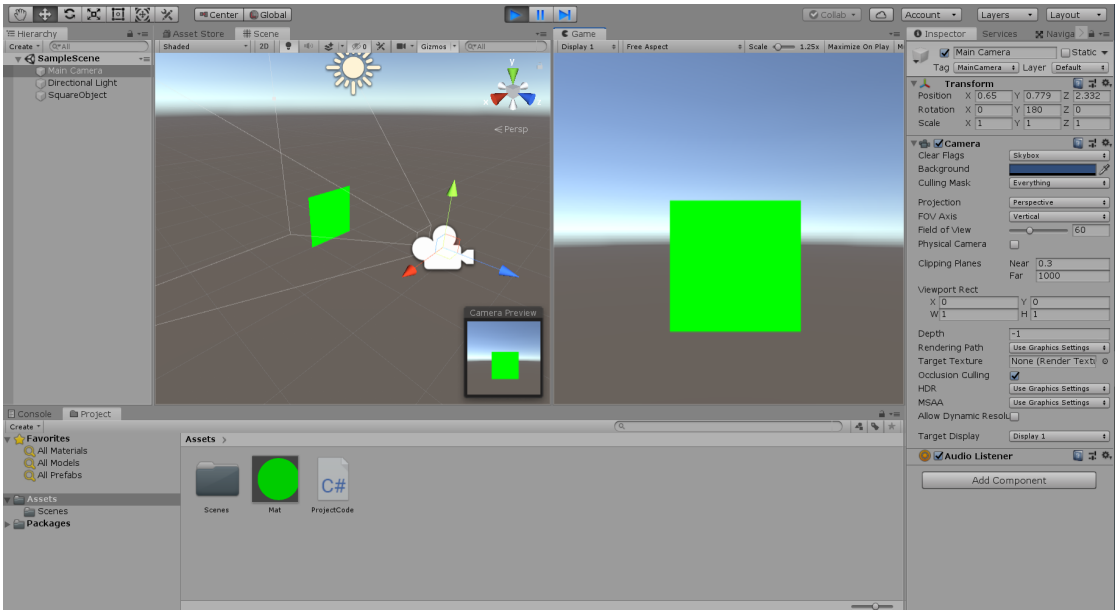

Figura 4.12: Realizzazione del piano tramite Mesh

Capitolo 5

Rivalutazione del Tool di Implementazione

Il tool ArUco, col quale è stato sostituito Vuforia, è già stato introdotto nel capitolo 3. In questo capitolo, invece, si procede ad un'analisi dal punto di vista implementativo e progettuale.

5.1 Impiego di ArUco in Unity

Come è stato indicato nel capitolo 3 alla sezione 3.2.2, ArUco è una libreria sviluppata in C++; ma l'ambiente di Unity prevede la scrittura di codice in C#. Nonostante sia ammissibile l'introduzione di file esterni in C++, si è preferito importare l'asset *OpenCV + Unity*[51] dallo store di Unity, il quale è una risorsa gratuita e *open source* che contiene la libreria *OpenCVSharp*, essenziale per l'impiego di ArUco. Questa scelta implementativa è stata presa per limitare eventuali problemi di compatibilità tra linguaggi diversi.

5.1.1 Riconoscimento del Marker

Il riconoscimento richiede, innanzitutto, la stampa su carta dei marcatori che si intendono utilizzare; mentre per l'aspetto di sviluppo occorre definire a quale dizionario, tra i vari disponibili, si intende fare riferimento. ArUco consente il tracciamento di più target simultaneamente, ma essi devono appartenere allo stesso set di marker.
Nella funzione di inizializzazione di Unity, *Start*, si ha, perciò, la precisazione del dizionario su cui si intende operare:

```
void Start()
{
    // Inizializzazioni:
    [...]

    // Definizione del dizionario e assegnazione dei parametri:
    dizionario = Aruco.OttieniDizionario(NomeDizionario);
    param = CreaParametriTracciamento();

    [...]
}
```

Listing 5.1: Metodo Start in Recognition Class cn ArUco

Il metodo di *Update*, presente nello pseudo-codice *lst.* 5.2, definisce i parametri necessari per il processo di rilevamento, come la costante per l'operazione di *thresholding*, il numero massimo di iterazioni per il processo di rifinitura degli angoli o il numero di bit del bordo dei marker[34]. Per il settaggio di questi parametri si può ricorrere ai valori di default, come in questo caso, oppure si possono personalizzare in base alle proprie esigenze.
Nella funzione *Update*, invece, si richiama il metodo atto al riconoscimento dei marker e, se si ottiene un riscontro positivo, si procede con eventuali attività sul target. Nell'estratto al *lst.* 5.2, ad esempio, al riscontro positivo relativo al tracciamento si calcolano il centro e la dimensione del marker, e mediante la funzione *Piano* si realizza l'oggetto virtuale.

```
void Update()
{
    [...]

    // Ricerca dei marker:
    Aruco.TracciamentoMarker(img, dizionario, out angoli, out id, param, out
     scartati);

    // Se almeno un marker e' stato trovato:
    if(id.Length > 0)
    {
        for(int i = 0; i < id.Length; i++)
        {
            centro[i] = CalcoloCentro(angoli);
            dim[i] = CalcolaDimensione(angoli));
        }

        Piano(dim[i]);
    }
    [...]
}
```

Listing 5.2: Metodo Update in Recognition Class con ArUco

Il metodo `Aruco.TracciamentoMarker(...)`, al *lst.* 5.2, riga 6, prevede diversi parametri:

- l'immagine su cui effettuare il *detection* deve essere statica, ma il procedimento è facilmente adattabile ad una ripresa video procedendo con uno screen della scena;
- il dizionario;
- le coordinate degli angoli dei marker trovati; questo elemento è una matrice di punti passata by reference mediante l'uso della parola chiave *out*;
- un vettore contenente gli identificatori dei marker rintracciati, anch'esso ritorna by reference;
- i parametri di configurazione istanziati nel metodo di inizializzazione;
- un vettore di angoli degli oggetti rigettati, passato per riferimento.

Successivamente al riconoscimento, nel metodo soprastante, si calcola, prima, il centro e la dimensione dei marker, tramite le coordinate degli angoli, ed infine si crea una *Mesh* per realizzare la forma del *Game Object* denominato *Plane*. L'oggetto virtuale è, quindi, sovrapposto al marcatore (fare riferimento al codice al *lst.* 5.6). In breve, lo pseudo-codice presentato finora mostra che il metodo *Start* si occupa delle inizializzazioni, mentre la funzione *Update*, che viene chiamata automaticamente dal sistema ad ogni cambiamento nella scena, analizza il frame e se rileva il marker calcola il centro e la dimensione del marcatore nel frame video. Inoltre, se il target è riconosciuto si procede con il metodo *Plane*, il quale, a sua volta, invoca la funzione *CreaPoligono* per la generazione della *Mesh* (per ulteriori informazioni vedere la sotto-sezione 4.3.2), sul marcatore.

Inoltre, un'altra sostanziale differenza, imposta dal tool descritto in questo capitolo, è l'impiego di un oggetto di tipo piano, chiamato camPlane, per renderizzare il frame e consentirne la vista all'utente, visionare il *lst.* 5.3, per una miglior comprensione. È importante evidenziare che il *Game Object* dedicato alla visualizzazione del frame è un oggetto differente rispetto al piano colorato da sovrapporre al marker.

Rispetto all'implementazione della *Mesh* con Vuforia, nello sviluppo con ArUco si sono riscontrate delle difficoltà nel posizionamento dell'oggetto virtuale sul marker, non avendo un collocamento automatico. Infatti, le coordinate degli angoli restituite da ArUco appartengono ad un sistema di riferimento locale al frame, per cui non coincidono con quelle necessarie, per il posizionamento del piano, nel sistema di riferimento globale dell'ambiente Unity. La soluzione proposta per risolvere il suddetto problema è stato l'utilizzo di due variabile `kx` e `ky`, per altezza

e larghezza rispettivamente, rappresentative dell'offset tra il sistema di riferimento del frame e quello dell'ambiente Unity. Tali variabili vengono sommate agli angoli, in coordinate, ricavati dal tool. I due offset sono ottenuti dal rapporto tra la dimensione dell'oggetto camPlane e la misura del frame, con la somma del fattore di scala, che è assegnato al piano renderizzatore alla sua creazione: 20 per le ascisse e 10 per le ordinate.

```
float kx = Dim_camPlane.x / dimFrame.width;
float ky = Dim_camPlane.y / dimFrame.height;


m.vertices = new Vector3[]
{
new Vector3(-(tl.X*kx)+20f, -tl.Y*ky+10f),              //TOP-RIGHT
new Vector3(-(tr.X*kx)+20f, -tr.Y*ky+10f),              //TOP-LEFT
new Vector3(-(br.X*kx)+20f, -br.Y*ky+10f),              //BOTTOM-RIGHT
new Vector3(-(bl.X*kx)+20f, -bl.Y*ky+10f)               //BOTTOM-LEFT
};
```

Il segno negativo posto prima delle coordinate è adoperato per ottenere la stessa direzione degli assi per i due sistemi di riferimento.

In alternativa all'impiego di una *Mesh*, ArUco offre l'utilizzo di alcuni metodi che rappresentano delle forme geometriche sul marcatore; il rettangolo mostrato in figura è creato con la funzione `Rectangle` ed è necessaria una nuova finestra per la raffigurazione.

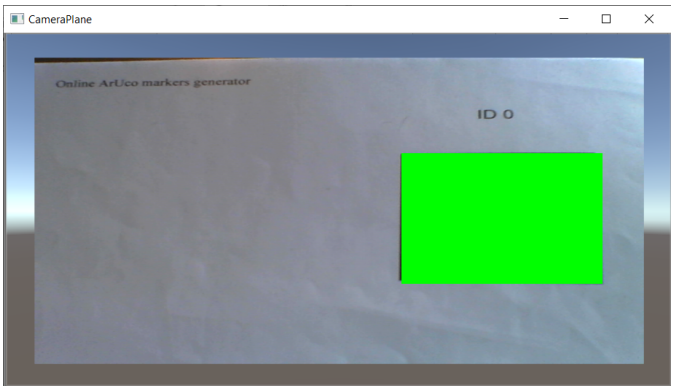

Figura 5.1: Marker Rilevato con Aruco

5.2 Variazioni nel Progetto

In questo paragrafo si analizzano, quindi, le variazioni, che sono state introdotte a causa della sostituzione del tool, per il *marker detection*. Vuforia ha il vantaggio di avere un'implementazione completa, nella quale le principali operazioni da eseguire, per il riconoscimento, sono gestite automaticamente. Al contrario, la libreria introdotta, pur consentendo molteplici task, richiede un'implementazione manuale, come ad esempio l'avvio della video camera e l'inizializzazione della stessa. Il principale cambiamento è l'inserimento di una *WebCameraTexture*[52], la quale consente di ottenere il frame dalla video camera; ed esso è, in seguito, renderizzato su un *Game Object* per la visualizzazione, denominato camPlane. Quest'ultimo simula, pertanto, le funzionalità di un display ed è collocato nella scena davanti alla *Camera*[53], affinché possa essere riprodotto all'avvio dell'applicativo. Il precedente motore fornisce, invece, un oggetto *AR Camera* che renderizza il frame, senza la necessità di un piano per permettere la visione all'utente. Relativamente al codice, le modifiche apportate sono introdotte sulla classe assegnata al rilevamento dei marcatori, ovvero la *Recognition Class*, citata precedentemente nel capitolo 4 alla sezione 4.1.2. La routine *Start*, della suddetta classe deve perciò:

- ottenere il dizionario;
- impostare i parametri del processo di riconoscimento;
- istanziare una *WebCameraTexture*;
- creare o ottenere un piano che renderizzi il frame video ottenuto dalla *texture*;
- posizionare il piano davanti all'oggetto *Camera*.

```
1  void Start()
2  {
3      // Inizializzazioni:
4      [...]
5
6      // Definizione del dizionario e assegnazione dei parametri:
7      dizionario = Aruco.OttieniDizionario(NomeDizionario);
8      param = CreaParametriTracciamento();
9
10     // Definizione della WebCameraTexture:
11     webCamTexture = CreaWebCameraTexture(NomeDispositivo);
12
13     // Acquisizione del piano dalla scena:
14     camPlane = TrovaPiano("CameraPlane");
15     // Rendering del frame video:
16     camPlane.SettaTexture(webCamTexture);
17
18     // Positionamento nell'origine:
19     SettaPos(camPlane, OttieniPos(Camera));
20 }
```

Listing 5.3: Metodo Start

La *WebCameraTexture*, dalla quale si ottengono i vari frame, restituisce un elemento *texture*; poiché il metodo adibito al rilevamento dei marker richiede in input un oggetto di tipo *Mat*[54][55], ovvero un array numerico n-dimensionale, occorre eseguire una conversione. La libreria *OpenCVSharp* fornisce due valide funzioni di conversione:

- `TextureToMat` per convertire una *Texture* in una *Mat*, è il metodo utilizzato qui;
- `MatToTexture` per ottenere una *Texture* da una *Mat*.

L'*AR Camera*, nominata precedentemente, ha anche il vantaggio di essere compatibile con dispositivi diversi, come i personal computer, i sistemi mobile e gli occhiali *smart*. Senza l'impiego di questa componente è inevitabile scindere la routine *Update* a seconda del dispositivo in uso. La principale differenza tra questi sistemi è il tipo di immagine che renderizzano[56]:

- gli applicativi mobile catturano immagini monoculari, cioè catturate da una singola telecamera e, conseguentemente, da un solo punto di vista;
- gli *Smart Glasses*, d'altro canto, sono composti da un display trasparente binoculare e necessitano che qualsiasi oggetto virtuale introdotto sia disegnato separatamente per ciascun occhio, perché hanno punti di vista differenti.

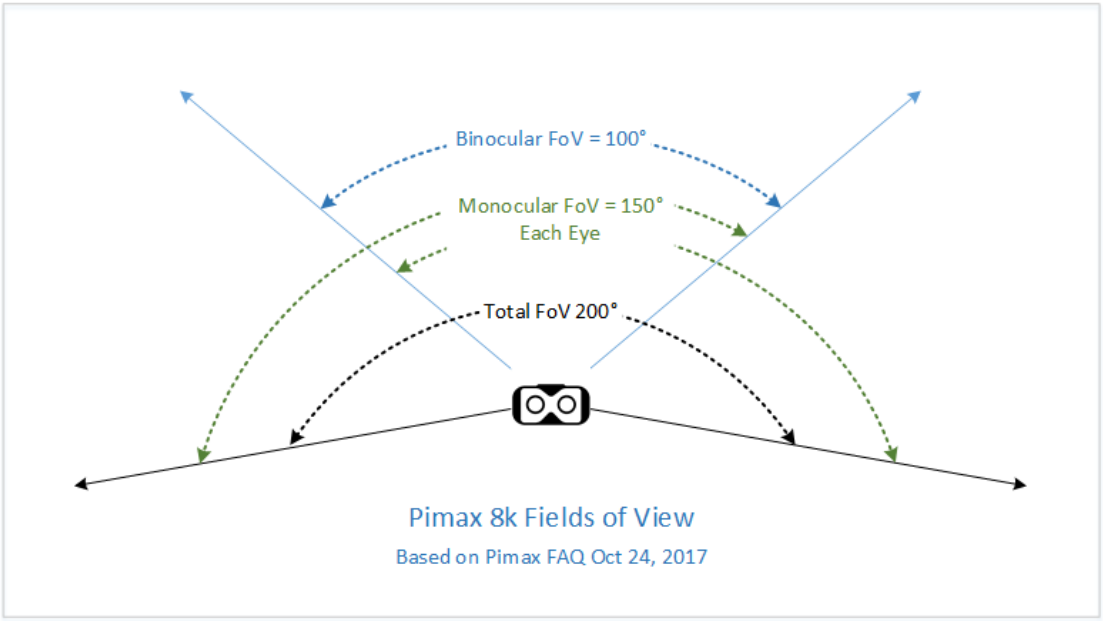

Figura 5.2: Vista Monoculare vs Vista Binoculare

Infatti, operando su dispositivi mobile o su elaboratori il software deve presentare il video, coincidente con quello della camera, con l'aggiunta della mappa virtuale sul marker.

Per l'esecuzione su *Smart Glasses*, nei quali si vuole consentire la vista naturale dell'ambiente e inserire digitalmente solo la mappa, si sostituisce la *texture*, rappresentante il frame video, con una *texture* nera, poiché i pixel neri risultano trasparenti. Affinché l'utente si avvalga del proprio campo visivo, senza l'uso di un video, e allo stesso tempo il marker sia posto nelle coordinate definite rispetto ai propri occhi, è fondamentale l'inserimento di un fattore di calibrazione.

Questa ulteriore distinzione, rispetto ai sistemi mobile, comporta l'uso di un offset, definito alla sotto-sezione 5.1.1 del capitolo corrente, per stabilire le coordinate del marker, relativamente alla vista dell'utente, a partire da quelle ricavate dal frame video.

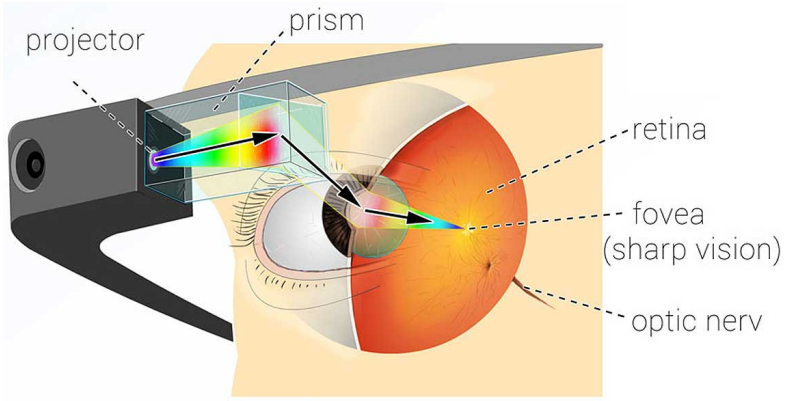

Figura 5.3: Rapporto tra l'occhio e l'occhiale smart

Lo pseudo-codice sottostante riporta le modifiche appena descritte:

```
void Update()     // Use this for initialization
   {
      if(webCameraTexture.FrameAggiornato() == true && webCameraTexture.Attivata()
     == true)
      {
         // Conversione in Mat:
         img = TextureToMat(webCamTexture);

         // Ricerca dei marker:
         Aruco.TracciamentoMarker(img, dizionario, out angoli, out id, param, out
     scartati);

         // Se almeno un marker e' stato trovato:
         if(id.Length > 0)
         {
            for(int i = 0; i < id.Length; i++)
            {
               centro[i] = CalcoloCentro(angoli);
               dim[i] = CalcolaDimensione(angoli));
            }

         }

         if(OttieniDispositivo("SmartGlasses"))
         {
            Stampa("Dispositivo riconosciuto: SmartGlasses in uso.");
            SettaTexture(Color.Black);
            k = OttieniOffset();

            Piano(dim[i], k);
         }
         else
         {
            Stampa("Dispositivo riconosciuto: Mobile in uso.");
            Piano(dim[i], 0);
         }

         [...]
      }
}
```

Listing 5.4: Metodo Update

Il metodo *Piano* mostrato nel seguito è analogo al metodo omonimo presentato al *lst.* 4.10, ma invoca la funzione aggiornata *CreaPoligono* (*lst.* 5.6). Inoltre, per la realizzazione della mappa virtuale sul marker mediante SmartGlasses si somma alla variabile `pos`, contenente le coordinate degli angoli, l'offset di calibrazione.

```
public bool Piano(dim, k)
{
   piano = TrovaPiano("Plane");
   if(piano == null)
   {
      Stampa("Errore nella creazione dell'oggetto");
      return false;
   }

   //Per l'offset di calibrazione dgli SmartGlasses
   if(k != 0)
   {
      pos = pos+k;
   }

   // Ottiene la posizione del marker:
   pos = OttieniPos(ImageTarget);

   piano = CreaPoligono(piano, ImageTarget, pos);

   // Modifica il colore:
   SettaColore(piano.OttieniRenderer().materiale);

   return true;
}
```

Listing 5.5: Metodo Piano per sviluppo dinamico della Mesh con ArUco

```
public GameObject CreaPoligono(piano, ImageTarget, pos)
{

   if(!piano.Possiede(MeshFilter) && !piano.Possiede(MeshRenderer))
   {
      piano.Aggiungi(MeshFilter);
      piano.Aggiungi(MeshRenderer);
   }

   l = OttieniLarghezza(piano);
   a = OttieniAltezza(piano);
   kx = camPlane.w / frame.w;
   ky = camPlane.h / frame.h;

   Mesh.vertici =
   {
      (-(pos.BS.x*kx) + scaleX, -(pos.BS.y*ky)+scaleY),     //Basso - Sinistra
      (-(pos.AS.x*kx) + scaleX, -(pos.AS.y*ky)+scaleY),     //Alto - Sinistra
      (-(pos.AD.x*kx) + scaleX, -(pos.AD.y*ky)+scaleY),     //Alto - Destra
      (-(pos.BD.x*kx) + scaleX, -(pos.BD.y*ky)+scaleY),     //Basso - Destra
   };

   Mesh.triangoli = new int[] { 0, 1, 3, 3, 1, 2 };

   MeshFilter.Mesh = Mesh;

   return piano;
}
```

Listing 5.6: Metodo CreaPoligono richiamato in Piano

Il metodo *CreaPoligono* soprastante riporta le modifiche necessarie per il posizionamento del marker; i vertici della *Mesh* sono generati con l'applicazione degli offset. Questi ultimi, come presentato nella sezione 5.1.1, sono dati dal rapporto tra la dimensione del piano renderizzante il frame e la misura del frame stesso.

5.3 Considerazioni

Nelle sezioni precedenti si è introdotto il nuovo tool, poi adoperato, per eseguire il tracciamento dei marcatori in modo più efficiente anche a distanze non ravvicinate. In questa ultima sezione, invece, si realizza un confronto tra le due librerie presentate in questo elaborato; si analizzano, perciò, gli aspetti positivi e negativi di entrambe per ottenere un sunto delle motivazioni che hanno condotto all'inserimento di questo capitolo.

Il primo SDK utilizzato, e poi sostituito, è Vuforia: esso offre innumerevoli possibilità, in ambito applicativo, e diversi vantaggi, come l'operazione automatica di *detection*.

I punti di interesse di Vuforia rilevati in questo sviluppo sono:

- *Marker Detection* è automatico, dopo la definizione dei database di immagini;
- la possibilità di introdurre il *Game Object AR Camera*, la quale è conforme a tutti i *device*: computer, sistemi mobile e visori;
- conseguentemente al punto antecedente, non richiede la scissione del codice secondo il tipo di dispositivo;
- nonostante non sia stato necessario, consente il riconoscimento non solo di target immagine ma anche di target multipli e oggetti;

D'altra parte, poiché questo motore opera su immagini, le quali vengono analizzate tramite *features*, si ha una riduzione di performance nel tracciamento dei marker, che, invece, presentano pochi elementi spigolosi.[36]

La riduzione prestazionale di Vuforia è definita dalla possibilità di rilevare soltanto marker molto vicini, questo svantaggio riduce, quindi, le applicazioni sviluppabili con questo tool: ovvero tutti gli applicativi che richiedono, di specifica, un riconoscimento a distanze maggiori.

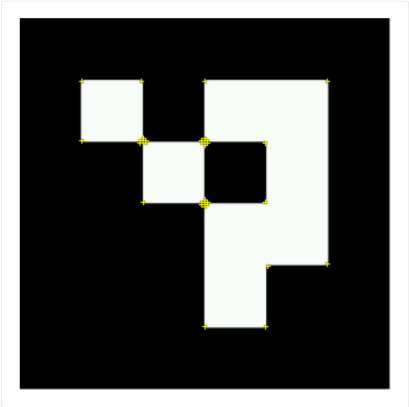

Figura 5.4: Features sul marker con ID 0 di Aruco

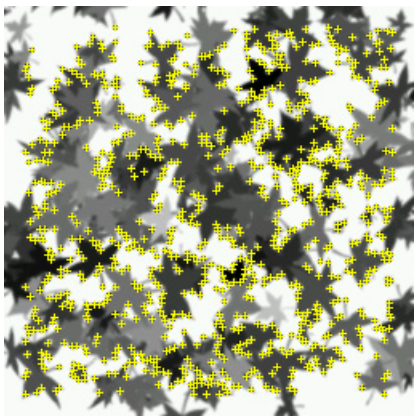

Figura 5.5: Features su immagine

Contrariamente ArUco interagisce solo con identificatori fiduciali, in bianco e nero; questa caratteristica riduce sicuramente le possibilità implementative ma, a prescindere dal fatto che nello sviluppo del presente applicativo non era previsto il riconoscimento di immagini, offre prestazioni elevate nello svolgimento di questo singolo task.

Lo svantaggio, ad ogni modo risolubile, dato dall'introduzione di ArUco è la necessità di scindere il codice, a seconda del dispositivo in uso (vedere il metodo al *lst.* 5.4), e dall'utilizzo di una *WebCameraTexture* per il rendering del video, che ha comportato l'uso di un offset per il posizionamento del piano del marcatore, a causa dei differenti sistemi di riferimento del frame e dell'ambiente Unity.

5.3.1 Test

Come già indicato, ArUco consente un miglior riconoscimento del marker, dimostrato in fase di test, ed esso corrisponde ad una caratteristica fondamentale per questo applicativo.

In fase di testing sono stati utilizzati due marcatori di dimensioni, rispettivamente, di 1.5 cm e 2 cm. Nelle immagini sottostanti è possibile vedere il sistema di prelievo, posto su un arto superiore, con la Sonda Due e il marker collocato su un elettrodo. Inoltre al tracciamento da parte di ArUco, il marcatore viene colorato di rosso.

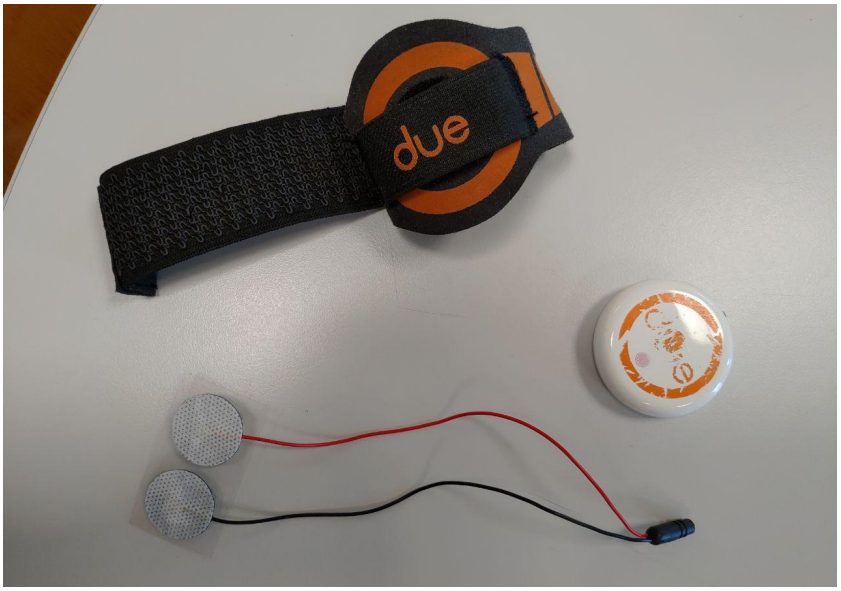

Figura 5.6: Sistema di prelievo con sonda Due

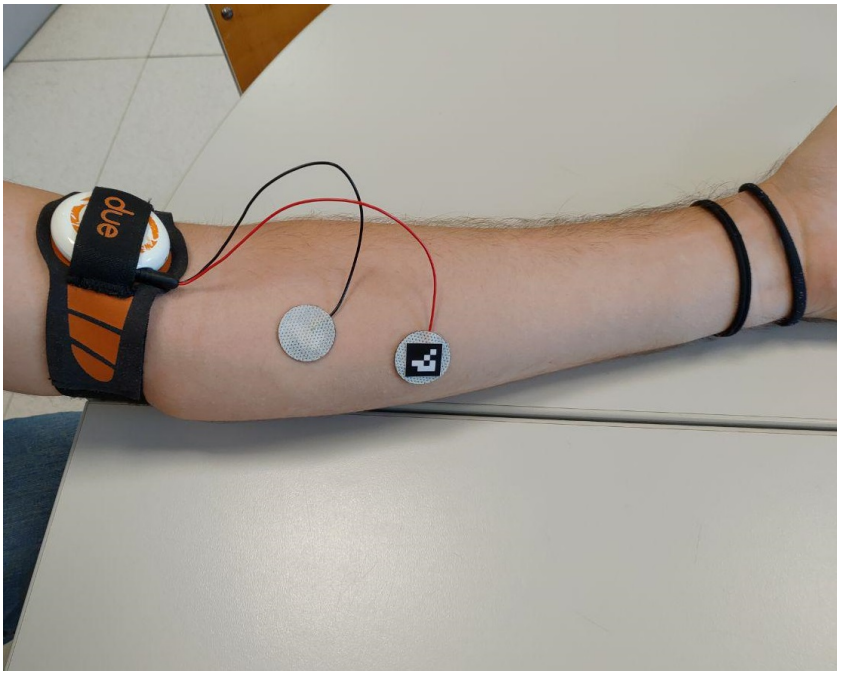

Figura 5.7: Sistema di prelievo con sonda Due e marker sull'elettrodo, arto superiore

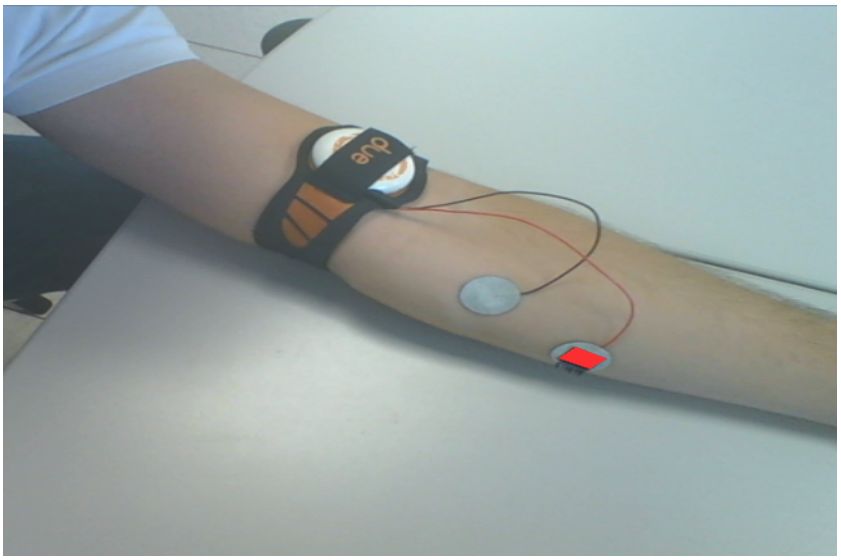

Figura 5.8: Sistema di prelievo con sonda Due e marker sull'elettrodo rilevato, arto superiore

Figura 5.9: Sistema di prelievo con sonda Due e marker sull'elettrodo rilevato, soggetto in piedi

Conclusioni e Scenari Applicativi

La realtà aumentata nei prossimi anni avrà sempre maggior diffusione, consentendo la realizzazione di software innovativi in molteplici ambiti di utilizzo, come quello scolastico, medicale, militare e gestionale. Queste nuove tecnologie comporteranno, quindi, un miglioramento nello stile di vita degli utenti e un supporto nelle attività quotidiane. L'aumento dello sviluppo di applicativi in AR è, inoltre, giustificato dai vari dispositivi su cui queste applicazioni possono essere eseguite: SmartPhone, Tablet, SmartGlasses, console video-ludiche, ecc.

L'obiettivo del presente progetto di tesi è stato la realizzazione di un sistema atto a fornire un feedback in realtà aumentata relativo al livello di attivazione muscolare. Il feedback previsto è ottenuto dall'analisi dell'attività elettrica generata dai muscoli, durante una contrazione (elettromiografia), e se ne prevedeva la rappresentazione mediante una mappa di codice colore.

Per lo sviluppo del software sono stati utilizzati l'ambiente Unity e due tool per il riconoscimento del marker. L'analisi e il testing di entrambi i tool hanno permesso l'ottimizzazione del *marker detection*. L'applicativo è stato progettato per essere eseguito su sistemi hardware diversi come SmartPhone, SmartGlasses e Personal Computer. La fase di testing ha rilevato il corretto funzionamento della trasmissione dati e della rappresentazione della mappa, identificativa del livello di attivazione muscolare, sul marcatore, sui Personal Computer. Mentre per SmartPhone e SmartGlasses si è constato l'avvio dell'applicativo ma non il posizionamento corretto del piano, questo problema è stato identificato come la necessità di ridimensionare la *texture* che renderizza il frame. Ad ogni modo il task di rilevamento funziona correttamente su tutti i dispostivi citati.

In fase di testing su Personal Computer si è utilizzato, inoltre, il gestore delle attività, fornito dal SO Windows, e si è constatato, all'avvio del software, un impiego medio della CPU del 21.6%, un'occupazione della memoria pari a 436.2MB e un consumo elettrico moderato. Mentre, in fase di rilevamento del marcatore si è notato un aumento di questi valori: l'utilizzo medio della CPU è salito al 23.8%, l'occupazione della memoria a 439.8MB ed il consumo elettrico, seppur mediamente moderato, ha avuto picchi più elevati. Inoltre, la prove effettuate sulla connessione ed il trasferimento di comandi tra il server ed il client hanno evidenziato una comunicazione affidabile, senza perdita di pacchetti e che permette un feedback dell'attivazione muscolare in real-time. Il task di

rilevamento è stato validato, con successo, su due marker di dimensioni molto ridotte, rispettivamente di 1.5 cm e 2 cm.

Il presente progetto non ha potuto completare lo sviluppo dell'intero sistema per motivi di tempistiche, ma ha realizzato una versione parziale di un software che potrà avere grande utilizzo in futuro in ambito riabilitativo. I problemi e le limitazioni riscontrate, in fase di sviluppo del software, possono essere risolte ed il progetto può, sicuramente, essere migliorato relativamente all'impiego su piattaforme diverse e al completamento della raffigurazione della mappa, del livello dell'attivazione muscolare. Inoltre, è anche necessaria un'importante fase di test, maggiore rispetto a quella finora eseguita, su pazienti differenti in termini di età, genere, ecc.

Gli scenari applicativi di questo software sono previsti in ambito riabilitativo su pazienti affetti da disturbi del sistema muscolare. Questo sistema può, quindi, essere utilizzato come tool di biofeedback per la riabilitazione motoria o come strumento di valutazione, prevenzione e monitoraggio clinico, fornendo informazioni aumentate al terapeuta e al paziente, durante una sessione di riabilitazione.

Bibliografia

[1] A. Pala, "Appunti di bioingegneria della riabilitazione." Centro Stampa CopySprinter, 2016-2017.

[2] "Centro phoenix: Psicologia, neuropsicologia, riabilitazione e psicoteriapia." [Online]. Available: https://www.centrophoenix.net/adulti/biofeedback/

[3] "Istituto di terapia cognitiva e comportamentale: il biofeedback." [Online]. Available: https://www.itcc.it/psicologia/biofeedback/

[4] M. Bracale, "Appunti del corso di elettronica biomedica: Elettromiografia."

[5] S. Passigli, "Tesi: L'elettromiografia nella valutazione funzionale dell'esercizio terapeutico." [Online]. Available: http://www.fisiobrain.com/web/2008/lelettromiografia-nella-valutazione-funzionale-dellesercizio-terapeutico

[6] M. B. Patrizia Boi, "La realtà virtuale e la realtà aumentata in architettura e ingegneria," in *Ingegneria Elevato n – Ingegneria del Futuro o Futuro dell'Ingegneria*, 2007.

[7] "Patrizia boi - ingegneria del futuro o futuro dell'ingegneria?" [Online]. Available: http://www.raiscuola.rai.it/articoli/patrizia-boi-ingegneria-del-futuro-o-futuro-dell%E2%80%99ingegneria/39665/default.aspx

[8] "Realtà aumentata: cos'è, come funziona, esempi." [Online]. Available: https://www.robotiko.it/realta-aumentata-cose/

[9] M. D. Mori, "Realtà virtuale, aumentata o mista?" [Online]. Available: https://www.focus.it/tecnologia/digital-life/realta-virtuale-aumentata-o-mista

[10] F. Giovinazzo, "Un algoritmo di marker tracking basato su local descriptors e image matching per applicazioni di realtà aumentata." in *Tesi di Laurea Magistrale in Tecnologie Informatiche*, 2010-2011.

[11] M. A. Livingston, L. J. Rosenblum, D. G. Brown, G. S. Schmidt, S. J. Julier, Y. Baillot, J. E. Swan, Z. Ai, and P. Maassel, "Military applications of augmented reality," in *Handbook of augmented reality*. Springer, 2011, pp. 671–706.

[12] "La realtà aumentata sbarca in campo militare grazie ai marines." [Online]. Available: http://www.experenti.eu/realta-aumentata/la-realta-aumentata-sbarca-in-campo-militare-grazie-ai-marines

[13] D. Vergun, "Heads-up display to give soldiers improved situational awareness," 2017. [Online]. Available: https://www.army.mil/article/188088/heads_up_display_to_give_soldiers_improved_situational_awareness/

[14] Ikea mobile apps. [Online]. Available: https://www.ikea.com/gb/en/customer-service/mobile-apps

[15] A. Carman, "Sephora's latest app update lets you try virtual makeup on at home with ar."

[16] "Bioelettronica: sistema duepro." [Online]. Available: https://www.otbioelettronica.it/index.php?option=com_content&view=article&id=90:duepro&catid=18:strumentazione-portatile&Itemid=101&lang=it

[17] "Bioelettronica: sonda due." [Online]. Available: https://www.otbioelettronica.it/index.php?option=com_content&view=article&id=96:due&catid=18&Itemid=251&lang=it

[18] P. A. Rauschnabel and Y. K. Ro, "Augmented reality smart glasses: An investigation of technology acceptance drivers," *International Journal of Technology Marketing*, vol. 11, no. 2, pp. 123–148, 2016.

[19] P. Gallos, C. Georgiadis, J. Liaskos, and J. Mantas, "Augmented reality glasses and head-mounted display devices in healthcare," *Studies in health technology and informatics*, vol. 251, pp. 82–85, 01 2018.

[20] P. RIdden, "Epson announces second-gen moverio smart glasses."

[21] A. Voica, "Smart glasses: The first wave of wearable and connected devices integrating imagination ip."

[22] "Smart glasses: Epson moverio bt-300." [Online]. Available: https://www.epson.it/products/see-through-mobile-viewer/moverio-bt-300

[23] "Unity: The world's leading real-time creation platform." [Online]. Available: https://unity3d.com/

[24] "Unity: The world's leading real-time creation platform." [Online]. Available: https://unity3d.com/unity?_ga=2.183376111.833212708.1563000920-749219006.1555420118

[25] P. Patil and R. Alvares, "Cross-platform application development using unity game engine," *IJARCSMS, Issue 4*, vol. 3, p. 19, 04 2015.

[26] R. H. Creighton, "Unity 3d game development by example beginner's guide," 09 2010.

[27] M. Marinelli, "Realizzazione di un gioco mediante unity e blender: Relazione in computer graphics," in *CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE*, 2015-2016.

[28] "L'interfaccia di unity unity bootstrap: Corso gratuito sviluppo giochi con unity 3d. le basi di unity dalla a alla z." [Online]. Available: https://www.xcoding.it/lezione/interfaccia-di-unity-3d/

[29] M. Sasso, "Game programming: Sviluppo di un gioco," in *CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE*, 2012-2013.

[30] "Vuforia: Optimizing target detection and tracking stability," 2018. [Online]. Available: https://library.vuforia.com/articles/Solution/Optimizing-Target-Detection-and-Tracking-Stability

[31] "Opencv website." [Online]. Available: https://opencv.org/

[32] F. Romero Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing*, vol. 76, 06 2018.

[33] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern Recognition*, vol. 51, 10 2015.

[34] "Documentazione opencv: Aruco." [Online]. Available: https://docs.opencv.org/3.4.0/d5/dae/tutorial_aruco_detection.html

[35] J. Zhang and J. Hu, "Image segmentation based on 2d otsu method with histogram analysis," in *2008 International Conference on Computer Science and Software Engineering*, vol. 6. IEEE, 2008, pp. 105–108.

[36] "Library vuforia: Imagetarget." [Online]. Available: https://library.vuforia.com/articles/Training/Image-Target-Guide

[37] "Vuforia: How to use the trackable base class." [Online]. Available: https://library.vuforia.com/content/vuforia-library/en/articles/Solution/How-To-Use-the-Trackable-Base-Class.html

[38] "Documentazione unity: Renderer." [Online]. Available: https://docs.unity3d.com/ScriptReference/Renderer.html

[39] "Documentazione unity: Collider." [Online]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html

[40] "A brief overview of tcp/ip communications." [Online]. Available: http://www.taltech.com/datacollection/articles/a_brief_overview_of_tcp_ip_communications

[41] "Documentazione microsoft: Uso di un socket client asincrono." [Online]. Available: https://docs.microsoft.com/it-it/dotnet/framework/network-programming/using-an-asynchronous-client-socket

[42] "Documentazione microsoft: Ipendpoint class." [Online]. Available: https://docs.microsoft.com/it-it/dotnet/api/system.net.ipendpoint?view=netframework-4.8

[43] "Documentazione microsoft: Manualresetevent class." [Online]. Available: https://docs.microsoft.com/it-it/dotnet/api/system.threading.manualresetevent?view=netframework-4.8

[44] "Documentazione microsoft: Asynccallback delegate." [Online]. Available: https://docs.microsoft.com/it-it/dotnet/api/system.threading.manualresetevent?view=netframework-4.8

[45] "Documentazione unity: Createprimitive." [Online]. Available: https://docs.unity3d.com/ScriptReference/GameObject.CreatePrimitive.html

[46] "Documentazione unity: Texture." [Online]. Available: https://docs.unity3d.com/Manual/Textures.html

[47] "Documentazione unity: Mesh." [Online]. Available: https://docs.unity3d.com/ScriptReference/Mesh.html

[48] D. Beloosesky, "Meshfilter vs meshrenderer." [Online]. Available: https://medium.com/@davidbeloosesky/mesh-filter-vs-mesh-renderer-8f4db9e2f966

[49] "Documentazione unity: Meshfilter." [Online]. Available: https://docs.unity3d.com/Manual/class-MeshRenderer.html/

[50] "Documentazione unity: Meshrenderer." [Online]. Available: https://docs.unity3d.com/560/Documentation/Manual/class-MeshRenderer.html

[51] "Asset store: Opencv + unity." [Online]. Available: https://assetstore.unity.com/packages/tools/integration/opencv-plus-unity-85928

[52] "Documentazione unity: Webcameratexture." [Online]. Available: https://docs.unity3d.com/ScriptReference/WebCamTexture.html

[53] "Documentazione unity: Camera." [Online]. Available: https://docs.unity3d.com/ScriptReference/Camera.html

[54] "Documentazione opencv:mat." [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html

[55] "Documentazione opencv: cv::mat class reference." [Online]. Available: https://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html#details

[56] J. Linowes and K. Babilinski, *Augmented Reality for Developers: Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia.* Packt Publishing Ltd, 2017.

Ringraziamenti

Ringrazio il mio Relatore ed il mio Correlatore per il tempo impiegato a seguirmi in questo progetto di tesi, li ringrazio per la loro esperienza e gli insegnamenti che ho ricevuto.

Ringrazio, inoltre, *mia mamma* per tutte le ore passate a farmi compagnia al telefono, a parlare di nulla il più delle volte, e per il suo supporto incondizionato.

Ringrazio *mio papà* per tutti i sacrifici che ha fatto per permettermi di arrivare fino a qui, ma soprattutto per le parole dolci che mi ha riservato nell'ultimo periodo, sono state di grande conforto e fonte di stimolo ad andare avanti.

Ringrazio *Liby* per aver sempre avuto un consiglio e una parola di incoraggiamento, per avermi guidata nelle scelte più importanti di questo percorso, e non solo. Sei il mio modello.

Ringrazio *Ciube* per avermi detto di provare Ingegneria sette anni fa, ci avevi visto giusto! Sei un genio.

Ringrazio *Simo* per avermi fatto capire che alcune cose sono importanti, ma altre di più.

Ringrazio *Jacopo* per essere qui oggi, questo è il miglior regalo che potesse farmi.

Ringrazio i miei amici *Francy*, *Gabry* e *Shelly*, insieme a *Marco* e *Maura* per avermi rallegrato le serate e alleggerito il cuore nei periodi di stress.

Ringrazio *Teo* per avermi preparato la cena tutte le volte che non avrei mangiato presa dall'ansia, per la pazienza che ha messo negli ultimi mesi, per tutti gli abbracci ricevuti quando sentivo di non farcela e avrei voluto abbandonare tutto. Lo ringrazio per aver sempre creduto nelle mie capacità.

Ci tengo a ringraziare ancora *Nicola*, *Ilaria* e *Stefano* per i confronti e gli insegnamenti che mi hanno dato ad ogni abbuffata fatta insieme.

Ringrazio *me stessa* per non aver mollato mai.

Grazie tutti! Vi adoro.