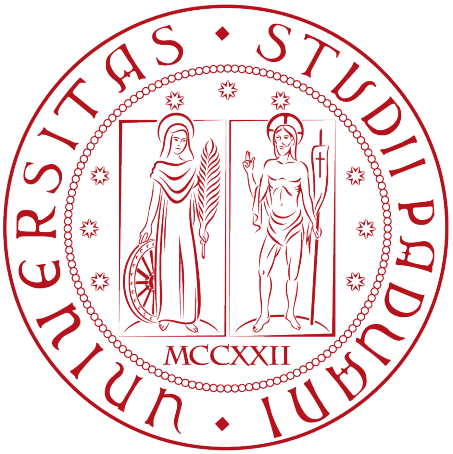
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Realtà Aumentata applicata all'Industria: gestione di una linea produttiva

Tesi di laurea triennale

Relatore

Prof. Ballan Lamberto

Laureando

Pilotto Nicholas

Anno Accademico 2021-2022

Ci sono momenti nella vita in cui qualcuno ti manca così tanto che vorresti tirarlo fuori dai tuoi sogni per abbracciarlo davvero.

— Paulo Coelho

Dedicato a nonno Pietro e Davide

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, dal laureando Pilotto Nicholas presso l'azienda Dallan S.p.A. Tale documento rappresenta non solo la conclusione dell'attività di stage, ma anche la fine del percorso di studi di Laurea triennale in Informatica.
L'obiettivo dello stage era la creazione di un'applicazione che permettesse di riconoscere delle immagini predefinite ed, attraverso Realtà Aumentata, visualizzare informazioni e parametri di una linea produttiva.

"Stay hangry, stay foolish"

— Steven Paul Jobs

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto ed il sostegno fornitomi durante la stesura del lavoro.

A Dallan S.p.A. per avermi ospitato in questa mia esperienza e per avermi dato tutto il necessario per portarla a termine.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio. Per tutto il sostegno ricevuto, non solo durante gli anni scolastici, ma in tutte le cose che ho fatto.

A mia sorella Gloria, assieme a Mauro e Victoria, un ringraziamento per la vicinanza, l'interesse per i miei studi e per avere sempre avuto una parola di conforto nei momenti in cui ne avevo bisogno.

Ho desiderio di ringraziare, infine, nonna Rosina per tutto il supporto che mi ha dato, in tutto.

Padova, Dicembre 2022 Pilotto Nicholas

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli Algoritmi

Capitolo 1

Introduzione

1.1 L'azienda

Dallan S.p.A. è un'azienda collocata a Castelfranco Veneto, nel trevigiano, nata nel 1978 con lo scopo di rivoluzionare il settore industriale. Uno dei leader mondiale nella realizzazione di linee di produzione, come macchine per la profilazione, punzonatrici o taglio laser, con clienti da svariate parti del mondo.

Figura 1.1: Logo di Dallan S.p.A.

1.2 L'offerta di Stage

I macchinari prodotti dall'azienda presentano dimensioni importanti, occupando anche intere campate di capannoni industriali, costringendo così gli operatori a percorrere tutta la lunghezza per verificare dati di interesse. L'idea è quella di sviluppare un'applicazione *mobile* che, attraverso un $marker_G$, riconosca la particolare zona della linea di produzione desiderata e visualizzi i parametri specifici. Per rendere il tutto più interattivo e gradevole, si è pensato di far apparire tali informazioni tramite Realtà Aumentata.

1.3 Organizzazione del testo

Il secondo capitolo descrive l'organizzazione dello stage;

Il terzo capitolo approfondisce l'analisi dei requisiti che l'applicazione deve rispettare;

Il quarto capitolo descrive gli il processo di progettazione ed i problemi riscontrati;

Il quinto capitolo descrive il processo di sviluppo, implementando gli studi effettuati nella sezione **§4**

Il sesto capitolo approfondisce come vengono visualizzati gli oggetti attraverso AR e come è possibile intercettare un tocco su un oggetto 3D

Nel settimo capitolo descrive le conclusioni finali dell'attività di stage

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per tutte le occorrenze dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: $parola_G$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

Dallan S.p.A. segue tutto il processo di progettazione e realizzazione delle linee di produzione, dalla parte meccanica ed elettrica fino al software che permette il corretto funzionamento.

I macchinari prodotti eseguono tutti i passaggi che portano la trasformazioni di una materia prima ad essere un prodotto finito, compresa la fase di imballaggio. Dati i vari step di lavorazione, una linea di produzione occupa un notevole spazio che si estende in lunghezza.

Tutti i comandi ed i controlli utili all'operatore finale si trovano in quello che Dallan S.p.A. chiama "pulpito": il cervello che permette di controllare e/o impostare parametri inerenti al funzionamento del processo produttivo. Si tratta di un unico schermo, posto a lato della macchina.

Essendo, per l'appunto, unico, gli operatori sono costretti compiere diversi spostamenti in caso di necessità, dovendo, in alcuni casi, percorrere tutta la lunghezza della linea di produzione. Vi è l'esigenza di semplificare la gestione e visualizzazione dei dati, parametri e comandi della macchina, senza effettuare grandi spostamenti.

L'idea di Dallan S.p.A. è quella di creare un'applicazione *mobile* che permetta la gestione della macchina direttamente da un qualsiasi dispositivo portatile, come ad esempio uno *smartphone*. Tale applicativo, inquadrando un $marker_G$ presente in ogni determinata zona della linea produttiva, la riconoscerà ed, attraverso la Realtà Aumentata, mostrerà a schermo tutte le informazioni necessarie.

La Augmented Reality (Realtà Aumentata) è una tecnologia nata alla fine degli anni '60, ma ha acquisito un maggiore interesse attorno al 2010. Vi sono varie suddivisioni, ma quella che verrà presa in considerazione per lo sviluppo del progetto è *Augmented Images* (Immagini Aumentate). L'idea che sta alla base è molto semplice: attraverso la videocamera di un dispositivo vengono riconosciute delle immagini campione, dei $marker_G$. Avvenuto il riconoscimento, viene applicato un *Layer* (Livello) nel quale vengono aggiunti degli elementi, "arricchendo" così la visione del mondo reale.

2.2 Pianificazione

La durata totale dello stage è stata di trecentoventi ore, pari ad otto settimane lavorative. Secondo il piano di lavoro iniziale definito con l'azienda, le attività sono distribuite come riportato in **Tabella 2.1**.

Settimana	Ore	Descrizione
1	12	Analisi e studio app attuale
	8	Analisi e studio piattaforma
	20	Analisi e studio $Xamarin_G$
2	40	Architettura scalabile e customizzabile
3	7	Navigazione tra pagine
	14	Comunicazione con la piattaforma esistente
	19	Creazione oggetti grafici
4	2	Creazione progetto ed importo pacchetti
	25	Creazione business logic
	13	Implementazione oggetti grafici
5	5	Interfacciamento API_G
	30	Gestione dei "Motori"
6	35	Studio Realtà Aumentata
	5	Decisione tecnologia
7	15	Test nuovo ambiente
	25	Progettazione Realtà Aumentata
8	40	Implementazione Realtà Aumentata
Totale	**320**	

Tabella 2.1: Organizzazione del lavoro

A causa di imprevisti durante la fase di sviluppo, alcune attività hanno necessitato di maggior tempo rispetto alla previsione iniziale, compensando, però, le attività che sono risultate più rapide. Le aspettative iniziali sono state rispettate, portando a termine lo sviluppo dell'applicazione.

2.3 Le tecnologie utilizzate

In questa sezione vengono elencate ed illustrate tutte le tecnologie utilizzate per la realizzazione del progetto di stage.

Visual Studio 2019 Ambiente di sviluppo integrato, di proprietà e sviluppato da *Microsoft*, ideato per la creazione di applicazioni *desktop*, *mobile* ma anche *console* in ambiente *Microsoft Windows*, *Android* e web.

Xamarin *Framework*$_G$ per lo sviluppo di app multipiattaforma seguendo il concetto *"Code once, run everywhere"*. Infatti, permette di condividere del codice su più piattaforme, generando *GUI*$_G$ native utilizzando il linguaggio *C#*$_G$.

Unity Motore grafico, utilizzato principalmente per lo sviluppo di videogiochi, ma anche per la visualizzazione di contenuti interattivi come animazioni 3D.

Unity Studio Strumento che permette la progettazione e realizzazione di prodotti software utilizzando il motore grafico *Unity*.

Samsung S20 FE Dispositivo fisico, con sistema operativo *Android*, sul quale è stata installata l'applicazione per l'esecuzione di test.

NuGet È un gestore di pacchetti progettato per consentire agli sviluppatori di condividere codice riutilizzabile.

Capitolo 3

Analisi dei Requisiti

3.1 I vincoli del progetto

Il progetto svolto durante l'attività di stage rappresentava lo sviluppo e lo studio di nuove tecnologie per Dallan S.p.A., ma esse dovevano integrarsi con il prodotto software già esistente. Tale software è scritto utilizzando il *framework*$_G$ *.NET Standard 2.0* di *Microsoft*, attraverso delle *DLL*$_G$. Questo perché, i PLC impiegati, utilizzano il sistema operativo *Microsoft Windows*.

Questo risulta essere un vincolo per la progettazione del progetto di stage, in quanto esso dovrà utilizzare tecnologie compatibili con tali *DLL*$_G$. Per rispettare questo vincolo, il codice scritto deve essere rappresentato dal linguaggio *C#*$_G$

3.2 Requisiti e obiettivi

Durante degli incontri preliminari allo stage, assieme al tutor aziendale Ing. Fuser Cristian, sono stati definiti tutti i requisiti che l'App deve rispettare.

Per identificare un requisito si è scelto di utilizzare la seguente codifica:

R[Tipologia]-[Classificazione][Identificativo]

Dove:

- **R:** definisce un requisito;
- **Tipologia:** un valore tra i seguenti
  - **F:** funzionale;
  - **Q:** qualitativo;
  - **P:** prestazionale;
  - **V:** vincolo;
- **Classificazione:** un valore tra i seguenti
  - **O:** obbligatorio;
  - **F:** facoltativo;
- **Identificativo:** codice univoco incrementale con valore iniziale pari a 1;

Codice	Descrizione	Use Case
RF-O8	Il dispositivo deve supportare la funzionalità di AR	UC1.1

Tabella 3.1: Requisiti funzionali stabiliti con Dallan S.p.A.

Codice	Descrizione	Use Case
RQ-F17	Al codice prodotto deve essere eseguita un'analisi statica	-
RQ-F18	Il codice e l'organizzazione del progetto deve essere facilmente manutenibile	-
RQ-F19	Il codice deve essere provvisto di commenti che ne documentino il funzionamento	-

Tabella 3.2: Requisiti qualitativi stabiliti con Dallan S.p.A.

Codice	Descrizione	Use Case
RP-F20	L'applicazione deve avere un tempo di avvio inferiore a 3 minuti	-

Tabella 3.3: Requisiti prestazionali stabiliti con Dallan S.p.A.

Codice	Descrizione	Use Case
RV-O1	Il software prodotto deve essere scritto con il linguaggio $C\#_G$	-
RV-O2	Il codice prodotto deve essere in grado di comunicare con DLL_G che utilizzano il $framework_G$ *.NET Standard 2.0*	-
RV-O3	L'applicazione deve informare l'utente eventuali errori di connessione al PLC	UC3.2, UC4.2
RV-O4	L'applicazione dovrà funzionare su dispostivi con sistema operativo *Android*	-
RV-O5	Nel caso di un dispositivo *Android*, le versione del sistema operativo dovrà essere pari o superiore ad *Android Nougat*	-
RV-F6	L'applicazione dovrà funzionare su dispostivi con sistema operativo *iOS*	-

Codice	Descrizione	Use Case
RV-F7	Nel caso di dispositivi *iOS*, la versione del sistema operativo dovrà essere pari o superiore ad *iOS 12*	-
RV-O9	L'applicazione deve informare l'utente in caso il dispositivo non supporti AR	UC1.2
RV-O10	Sviluppo di una funzionalità che permetta il riconoscimento di un $marker_G$	UC2.1
RV-F11	Il riconoscimento del $marker_G$ dovrà avvenire da una distanza compresa tra i 30cm ed i 50cm	-
RV-O12	L'applicazione deve visualizzare i dati derivanti dalle DLL_G	UC3.1
RV-O13	Creazione oggetto *custom Label*	-
RV-F14	Creazione oggetto *custom Gauge*	-
RV-F15	Creazione oggetto *custom Bottone*	-
RV-F16	L'utente deve essere in grado di premere l'oggetto *custom Bottone*	UC4.1

Tabella 3.4: Requisiti di vincolo stabiliti con Dallan S.p.A.

Tipo	Obbligatori	Facoltativi
Funzionali	1	0
Qualitativi	0	3
Prestazionali	0	1
Vincolo	9	6

Tabella 3.5: Riepilogo dei requisiti

3.3 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo UML_G dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Essendo il progetto finalizzato alla creazione di un *tool* per l'automazione di un processo, le interazioni da parte dell'utilizzatore devono essere ovviamente ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto.

Per identificare un caso d'uso si è scelto di utilizzare la seguente codifica:

UC[CodicePadre].[CodiceFiglio]

È importante ribadire come questo formalismo sia gerarchico, ovvero un codice figlio può essere codice padre di un suo eventuale codice figlio. Possono essere figli le generalizzazioni e i sottocasi d'uso.

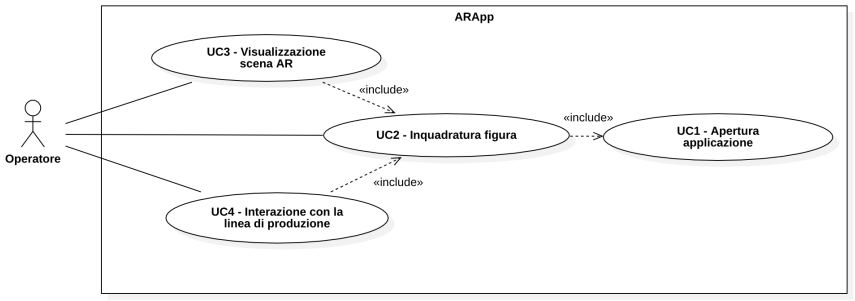


Figura 3.1: Casi d'uso: Scenario principale

3.4 UC1 - Apertura applicazione

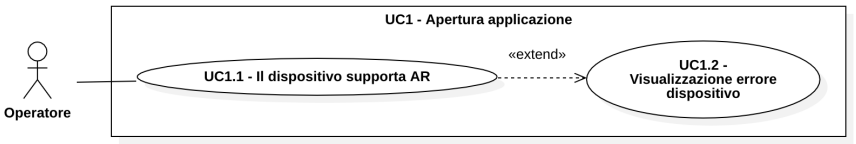


Figura 3.2: UC1: Scenario principale

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'operatore ha intenzione di aprire l'applicazione;
- **Descrizione:** L'operatore ha premuto l'icona dell'applicazione con la volontà di eseguirla sul proprio dispositivo;
- **Postcondizione:** L'applicazione viene istanziata nel dispositivo e mostrata a schermo;

3.4.1 UC1.1 - Il dispositivo supporta AR

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'operatore ha intenzione di aprire l'applicazione;
- **Descrizione:** L'applicazione effettua il controllo che il dispositivo supporti AR e che la versione del sistema operativo rispetti i requisiti;
- **Postcondizione:** L'applicazione mostra a schermo la schermata principale, pronta per l'utilizzo;

3.4.2 UC1.2 - Visualizzazione errore dispositivo

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'operatore ha intenzione di aprire l'applicazione;
- **Descrizione:** Il dispositivo non supporta AR o il sistema operativo non rispetta i requisiti;
- **Postcondizione:** L'applicazione mostra a schermo un messaggio che informa l'utente che il dispositivo non possiede i requisiti necessari e non permette l'utilizzo dell'applicazione;

3.5 UC2 - Inquadratura figura

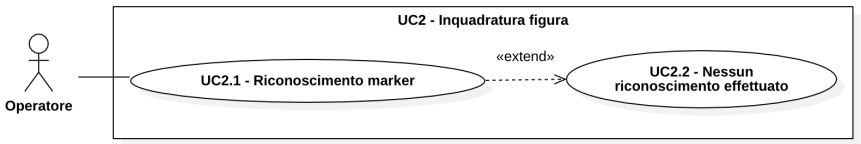


Figura 3.3: UC2: Scenario principale

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'applicazione è stata avviata ed è stato eseguito il controllo che il dispositivo supporti AR e che il sistema operativo rispetti i requisiti, controllo che ha portato esito positivo;
- **Descrizione:** L'operatore inquadra una figura che potrebbe rappresentare un $marker_G$;
- **Postcondizione:** Nel caso la figura rappresenti un $marker_G$, viene rappresentata a schermo la scena corretta;

3.5.1 UC2.1 - Riconoscimento $marker_G$

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo;
- **Descrizione:** La figura inquadrata viene riconosciuta come $marker_G$;
- **Postcondizione:** Vengono visualizzati a schermo la scena inerente al $marker_G$ riconosciuto;

3.5.2 UC2.2 - Nessun riconoscimento effettuato

- **Attori primari:**
  - Operatore;
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo;
- **Descrizione:** La figura inquadrata non viene riconosciuta come $marker_G$;
- **Postcondizione:** L'applicazione non visualizza a schermo nulla e continua il processo di *tracking*;

3.6 UC3 - Visualizzazione scena AR

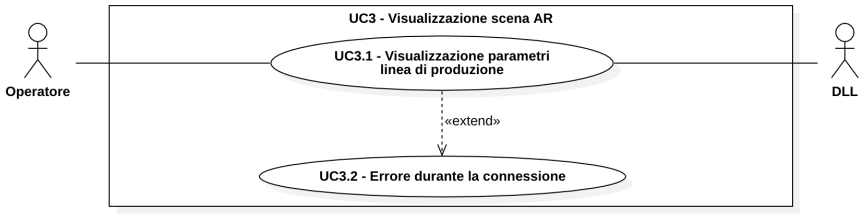


Figura 3.4: UC3: Scenario principale

- **Attori primari:**
  - Operatore;
  - DLL_G
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo, la quale è stata riconosciuta come $marker_G$;
- **Descrizione:** La figura inquadrata viene riconosciuta e si mostra la scena AR del corrispondente $marker_G$;
- **Postcondizione:** Viene mostrata la scena AR corrispondente al $marker_G$ riconsciuto;

3.6.1 UC3.1 - Visualizzazione parametri linea di produzione

- **Attori primari:**
  - Operatore;
  - DLL_G;
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo, la quale è stata riconosciuta come $marker_G$;
- **Descrizione:** La figura inquadrata viene riconosciuta come $marker_G$, si instaura una connessione con le DLL_G per la richiesta dei dati inerenti alla linea di produzione, i quali saranno visualizzati attraverso gli oggetti *Label* o *Gauge*, presenti nella scena AR associata al $marker_G$ riconosciuto;

- **Postcondizione:** Vengono visualizzati i dati della linea di produzione richiesti alle DLL_G tramite la scena AR inerente al $marker_G$ riconosciuto;

3.6.2 UC3.2 - Errore durante la connessione

- **Attori primari:**
  - Operatore;
  - DLL_G;
- **Precondizione:** LL'operatore ha inquadrato una figura con il proprio dispositivo che è stata riconosciuta come $marker_G$;
- **Descrizione:** Il tentativo di connessione alle DLL_G non è andata a buon fine;
- **Postcondizione:** L'applicazione avvisa l'operatore, tramite messaggio a schermo, che la connessione alle DLL_G non ha avuto esito positivo;

3.7 UC4 - Interazione con la linea di produzione

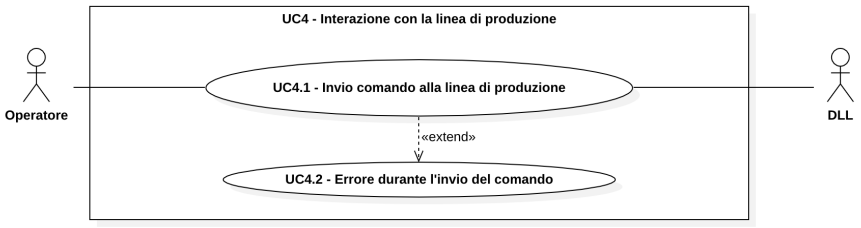


Figura 3.5: UC4: Scenario principale

- **Attori primari:**
  - Operatore;
  - DLL_G
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo, la quale è stata riconosciuta come $marker_G$;
- **Descrizione:** L'operatore vuole inviare comandi alla linea di produzione;
- **Postcondizione:** I comandi vengono inviati alla linea di produzione;

3.7.1 UC4.1 - Invio comando alla linea di produzione

- **Attori primari:**
  - Operatore;
  - DLL_G;
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo, la quale è stata riconosciuta come $marker_G$;

- **Descrizione:** L'operatore ha premuto il bottone presente nella scena AR con la volontà di inviare comandi alla linea di produzione attraverso la connessione alle DLL_G;
- **Postcondizione:** I comandi sono stati inviati alla linea di produzione ed eseguiti;

3.7.2 UC4.2 - Errore durante l'invio del comando

- **Attori primari:**
  - Operatore;
  - DLL_G;
- **Precondizione:** L'operatore ha inquadrato una figura con il proprio dispositivo che è stata riconosciuta come $marker_G$ e premuto un bottone presente nella scena AR;
- **Descrizione:** Il tentativo di connessione alle DLL_G non è andata a buon fine;
- **Postcondizione:** L'applicazione avvisa l'operatore, tramite messaggio a schermo, che la connessione alle DLL_G non ha avuto esito positivo;

Capitolo 4

Progettazione

In questo capitolo vene esaminato la fase di progettazione, analizzando le tecnologie disponibili ed i problemi sorti durante questo processo. Verrà trattato lo studio che ha portato alla risoluzione, motivando le scelte prese.

4.1 Il primo approccio

L'idea iniziale di Dallan S.p.A. era quella di integrare la funzionalità di AR nella applicazione già esistente. Questo applicativo, seguiva la filosofia *"Code once, run everywhere"*, ovvero possiede una $codebase_G$ comune per più progetti, nel caso specifico *Android* ed *iOS*.

Per lo sviluppo di questa app, viene utilizzato il $framework_G$ $Xamarin_G$, attraverso il linguaggio $C\#_G$. Di conseguenza, la progettazione delle funzionalità di AR dovevano essere studiate per il $framework_G$ in questione.

Nasce il primo problema: per utilizzare la AR, sia per il riconoscimento dei $marker_G$, che per la visualizzazione di oggetti 3D, il dispositivo deve fare uso di librerie native che, per i dispositivi *Android* si interfacciano con il $framework_G$ $ARCore_G$, mentre per i dispositivi *iOS* con $ARKit_G$.

4.1.1 *Xamarin*$_G$

$Xamarin_G$, nel corso degli anni, ha raggiunto un livello di performance paragonabile ad un codice nativo, rendendolo un approccio molto utilizzato dagli sviluppatori. Infatti, il sorgente prodotto, utilizzando *Xamarin.iOS* e *Xamarin.Android*, viene compilato producendo codice *assembly* nativo. Le API_G interrogate sono chiamate alle librerie scritte da *Apple* e *Google*, attraverso dei $wrap_G$ scritti in $C\#_G$.

Figura 4.1: Logo del *framework*$_G$ *Xamarin*$_G$

4.1.2 I problemi di *ARCore*$_G$ se utilizzato con *Xamarin*$_G$

La gestione descritta in **§4.1.1** funziona correttamente nel caso si usi *ARKit*$_G$, in quanto questo è il responsabile di tutte le operazioni di AR, dal *tracking* della camera alla visualizzazione degli oggetti 3D. Mentre, per quanto riguarda *ARCore*$_G$ le cose vengono gestite in modo diverso: il *framework*$_G$ in sé si occupa soltanto di tenere traccia del mondo reale mentre, per il *render* degli oggetti 3D, utilizza *OpenGL*$_G$, come mostrato in **Figura 4.2**.

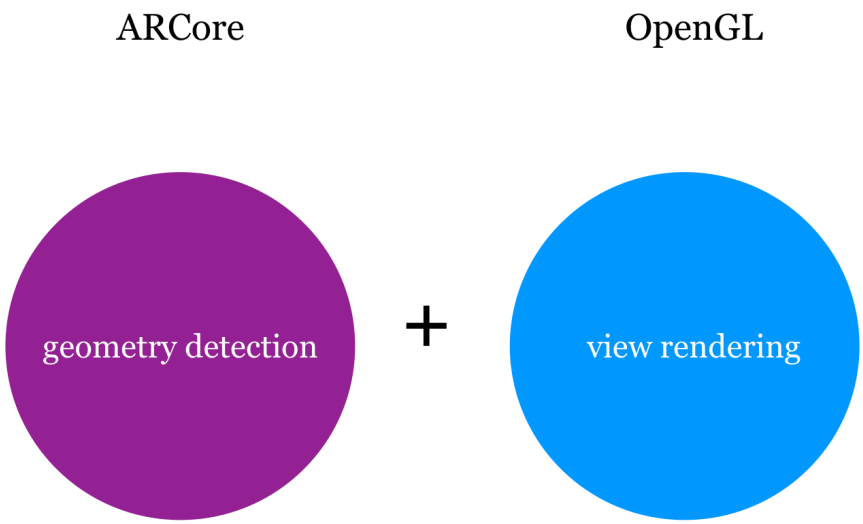


Figura 4.2: Collaborazione *ARCore*$_G$ e *OpenGL*$_G$

Il problema nasce proprio in questo scambio di compiti delle due librerie. Infatti, il *wrap*$_G$ di tutti i metodi di *ARCore*$_G$ è ben fatto e funzionate, mentre le chiamate alle varie *API*$_G$ di *OpenGL*$_G$ non sono fatte ad hoc, ma utilizzando il *bridge jni4net*.

Come verrà spiegato in **§4.1.2.1**, la traduzione delle *API*$_G$ non è automatico, ma un processo che comporta uno sforzo di alcuni programmatori. Il risultato finale

non comprende tutto il materiale necessario per l'utilizzo di queste tecnologie nel *framework*$_G$ *Xamarin*$_G$, scartando, così, questo primo approccio.

4.1.2.1 jni4net

Si tratta di un progetto *open-source*$_G$, il cui scopo è quello di creare u *bridge* (ponte) tra il linguaggio di programmazione *Java*$_G$ ed il *framework*$_G$ *.NET Standard*. Ovvero permette di eseguire chiamate ad *API*$_G$ ai metodi pubblici delle librerie *.NET* attraverso del codice *Java*$_G$, ma permette anche chiamate seguendo il percorso inverso.

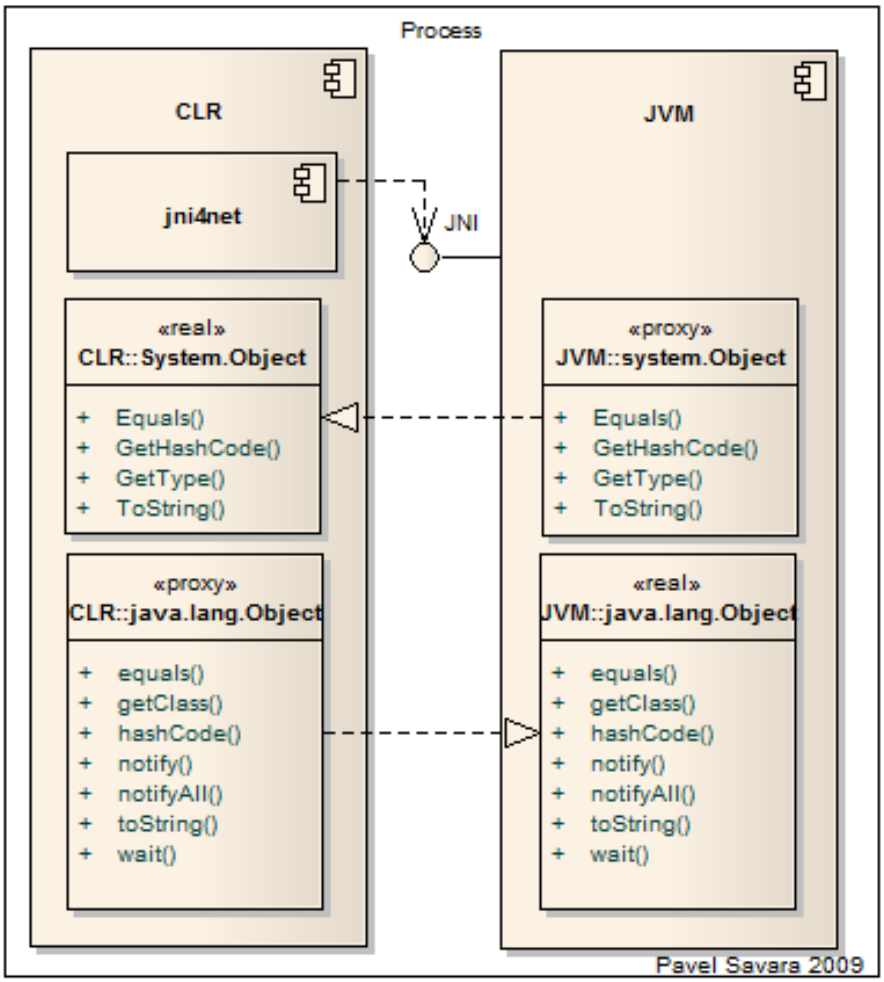


Figura 4.3: Semplificazione *bridge jni4net* [7]

Chiamate da *.NET* a *Java*$_G$ La *Java Virtual Machine* (JVM) espone le JNI, ovvero quel particolare *framework*$_G$ che permette al codice *Java*$_G$ di essere richiamato dal codice nativo di un determinato sistema operativo.

Le chiamate ai metodi avvengono sfruttando la *Java Reflection*, attraverso la quale vengono generati dei *proxy*. Questi assomigliano a delle classi *Java*$_G$, avendo gli stessi

nomi, *namespaces* e stesse firme delle funzioni. Tali oggetti non sono implementati in *.NET* richiamano l'implementazione nativa della JVM.

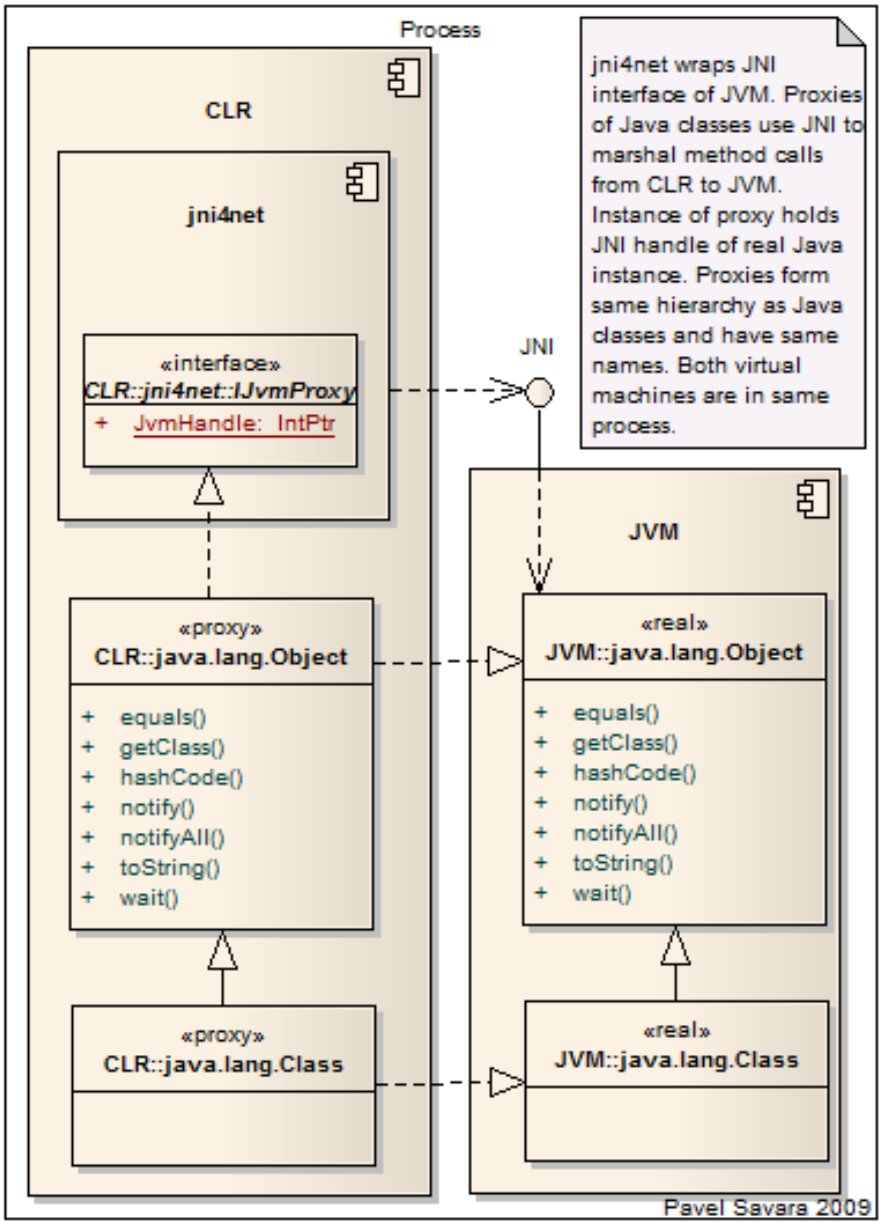


Figura 4.4: Schema delle chiamate *.NET* a $Java_G$ utilizzando *jni4net* [7]

Chamate da $Java_G$ a *.NET* All'interno di *jni4net* vi è un *tool* che trasforma tutti i metodi pubblici del $framework_G$ *.NET* in classi *proxy* in $Java_G$. Eseguendo questo cambio di linguaggio, vengono modificati anche i nomi dati agli oggetti, in modo tale da rispettare tutti gli stili di nomenclatura tipici di $Java_G$: ad esempio, ciò che $C\#_G$ definisce come *proprietà* vengono convertiti in metodi *set* e *get*, secondo lo standard

JavaBean.

In questo modo, si ha la possibilità di richiamare funzionalità scritte in *.NET* direttamente da $Java_G$.

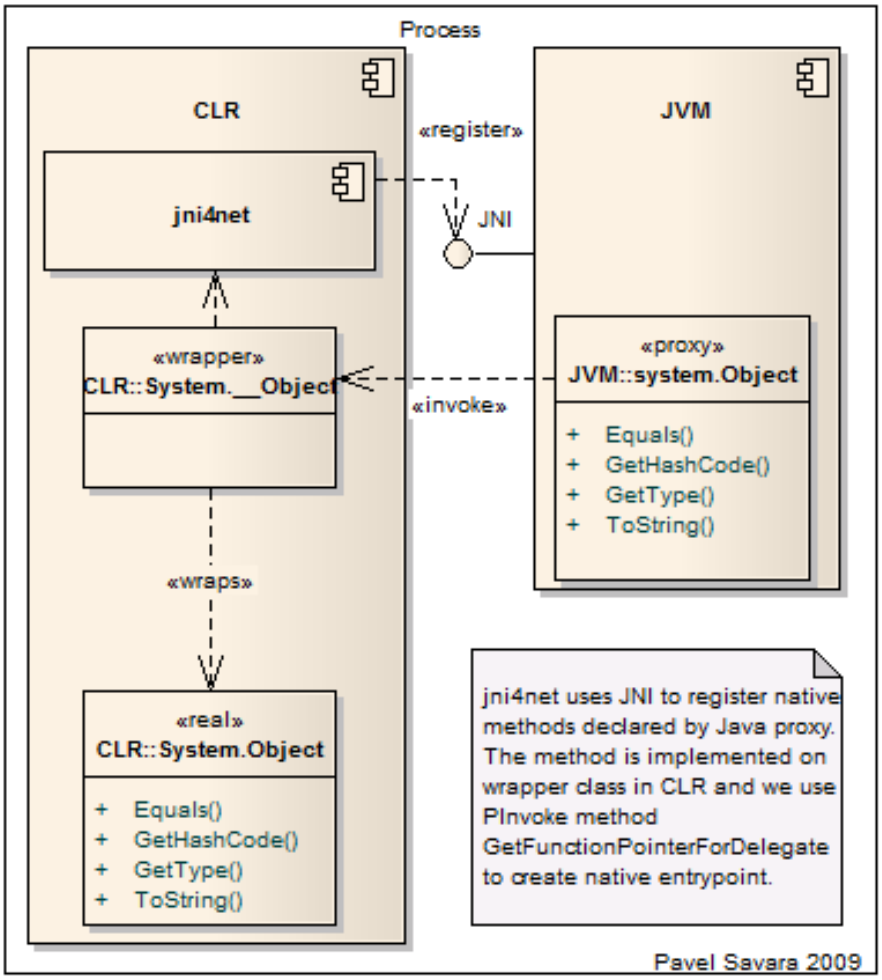


Figura 4.5: Schema delle chiamate $Java_G$ a *.NET* utilizzando *jni4net* [7]

4.2 Il secondo approccio

Date le problematiche descritte in **§4.1**, assieme al tutor aziendale Ing. Fuser Cristian, si è deciso di optare per altre alternative.

La prima strada pensata era la più ovvia, ovvero quella di creare un'applicazione nativa, scritta interamente in $Java_G$. Questa soluzione risolveva il problema sopracitato, ma comportava la realizzazione di due applicazioni distinte nel caso Dallan S.p.A. decidesse di produrre lo stesso software per dispositivi *iOS*.

La seconda strada pensata, che sarà poi quella implementata, è l'utilizzo di *Unity*.

4.2.1 *Unity*

Unity [19] è un motore grafico che permette la realizzazione di contenuti interattivi e visualizzazione di oggetti 3D in tempo reale. Il suo impiego principale è per lo sviluppo di videogiochi, ma, ultimamente, sta prendendo piede anche nella creazione di applicazioni.

Figura 4.6: Logo *Unity*

La sua caratteristica principale è la possibilità di produrre software multipiattaforma, inglobando anche le ultime tecnologie quali *Oculus Rift*[1], $ARCore_G$ ed $ARKit_G$ [20].

Per effettuare la codifica utilizzando *Unity*, si possono utilizzare i linguaggi *C/C++* e $C\#_G$.

Proprio grazie a queste due caratteristiche, tale motore grafico era perfetto per la realizzazione dell'applicazione che Dallan S.p.A. intendeva produrre.

4.2.2 *Unity* ed $ARCore_G$

Unity ha la possibilità di utilizzare $ARCore_G$ a pieno delle sue potenzialità, in quanto *Google* fornisce pieno supporto a tale motore grafico.

Nella pagina della documentazione di $ARCore_G$ [3], viene spiegato come, attraverso SDK_G *AR Foundation* (**§4.2.3**) di *Unity*, sia possibile sviluppare applicazioni che utilizzino $ARCore_G$.

Oltre a spiegazioni dettagliate, un altro fattore a favore di questo approccio è il sostegno della *community* [2], la quale fornisce spiegazioni, esempi ed aiuto per tutto ciò che riguarda lo sviluppo secondo questa tecnologia. Supporto che a $Xamarin_G$, se si fosse seguita quella strada, sarebbe mancato, non essendoci molti forum che trattano la questione.

[1] visore per la Realtà Virtuale (VR), progettato e prodotto da *Oculus VR*, oggi sotto il controllo di *Meta*. Da tale acquisizione il nome cambia in *Meta Quest*.

[2] sostantivo, comunità

4.2.3 AR Foundation

AR Foundation è un pacchetto che permette di sfruttare le potenzialità della AR, scrivendo codice eseguibile su multi-piattaforma, tutto all'interno di *Unity*.

Questo pacchetto fornisce agli sviluppatori soltanto un'interfaccia: infatti, *AR Foundation* non implementa nessun metodo, ma riconosce il dispositivo *target* di compilazione, per poi utilizzare dei *plug-in*[3] per il sistema operativo di interesse.

I dispositivi che sono supportati sono:

Dispositivo	**Sistema Operativo**	**SDK AR**
iPhone, iPad	iOS/iPadOS	$ARKit_G$
Android Phones	Android	$ARCore_G$
Magic Leap	Lumin OS	LuminSDK
HoloLens	Microsoft Windows	Mixed Reality Toolkit SDK

Tabella 4.1: Dispositivi supportati da *AR Foundation* [2]

Per quanto riguarda $ARCore_G$ e $ARKit_G$, le funzionalità disponibili sono:

Funzionalità	$\boldsymbol{ARCore_G}$	$\boldsymbol{ARKit_G}$
Device tracking	✓	✓
Plane tracking	✓	✓
Point clouds	✓	✓
Anchors	✓	✓
Light estimation	✓	✓
Environment probes	✓	✓
Face tracking	✓	✓
2D Image tracking	✓	✓
3D Object tracking		✓
Meshing		✓
2D & 3D body tracking		✓
Collaborative participants		✓
Human segmentation		✓

[3]"modulo aggiuntivo". Termine usato per definire dei programmi (software) che vanno aggiunti ad applicazioni più complesse al fine di estenderne le funzionalità. [6]

Funzionalità	$ARCore_G$	$ARKit_G$
Raycast	✓	✓
Pass-through video	✓	✓
Session management	✓	✓
Occlusion	✓	✓

Tabella 4.2: Funzionalità di $ARCore_G$ e $ARKit_G$ supportate da *AR Foundation* [1]

4.2.3.1 Architettura di *AR Foundation*

L'architettura di *AR Foundation* [2] è l'insieme di due componenti:

- **AR Subsystem**;
- **AR Components**;

Il modulo *AR Subsystem* è colui collega le interfaccie di *Unity* con i metodi del $framework_G$ del dispositivo (come spiegato precedentemente in **§4.2.3**).

Mentre, *AR Components* contiene tutti gli oggetti utili a gestire gli eventi che vengono generati dall'applicazione. I più importanti sono:

- **ARSession:** gestisce il ciclo di vita di tutte le attività di AR;
- **ARPlaneManager:** gestisce il riconoscimento delle superfici piane;
- **ARRaycastManager:** gestisce tutti gli eventi legati al tocco da parte dell'utente sull'ambiente AR;
- **ARTrackedImageManager:** gestisce il riconoscimento di immagini;

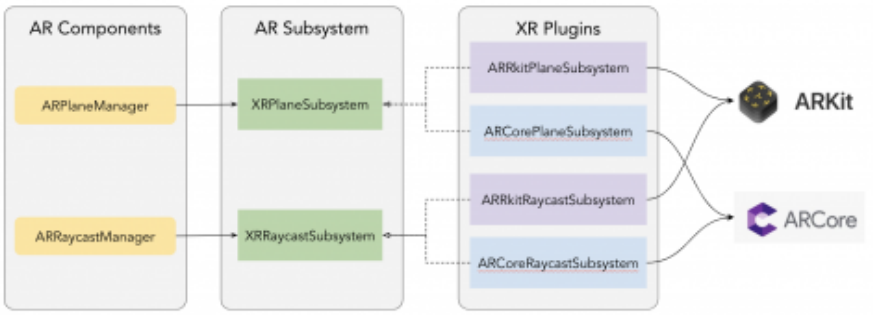


Figura 4.7: Schema dell'architettura di *AR Foundation* [1]

Capitolo 5

Sviluppo

In questo capitolo viene affrontato lo sviluppo dell'applicazione, quindi la scrittura del codice utile per la realizzazione del progetto richiesto da Dallan S.p.A., utilizzando le tecnologie studiate nel capitolo 4, ma anche la progettazione di tutte le cose utili al completamento di tale prodotto.

5.1 Organizzazione dello sviluppo

Durante lo sviluppo non vi è soltanto la fase di codifica del codice, ma anche la progettazione e la realizzazione di tutte quelle componenti utili al funzionamento dell'applicazione.
In particolare, lo sviluppo si è suddiviso nel seguente ordine:

1. Creazione immagini $marker_G$;
2. Creazione oggetto *Label*;
3. Creazione oggetto *Gauge*;
4. Creazione *prefab*;
5. Riconoscimento $marker_G$;
6. Creazione oggetto *Bottone*;
7. Collegamento e recezione dati da DLL_G;
8. Refactoring del codice seguendo un analisi statica;

5.1.1 Creazioni immagini $marker_G$

Un parte fondamentale dell'applicazione era il riconoscimento di immagini univoche. Queste immagini, per l'appunto i $marker_G$, verranno, successivamente, applicate alle sottozone delle linee di produzione messe in commercio da Dallan S.p.A.. Per tale motivo, queste non devono solamente essere univoche, ma anche piacevoli alla vista e rappresentare in modo stilizzato, ciò sulla quale sono poste.

Il design di questi $marker_G$ è stato effettuato usando il software di disegno vettoriale *Sketch©* [1]

Figura 5.1: Logo di Sketch©

Durante questa fase, sono state progettate due immagini, rappresentanti le sottozone *"Aspo"* e *"Rulli"*.

Aspo Nell'industria, l'aspo è una macchina che serve ad avvolgere del materiale formandone una matassa. Nella fattispecie, Dallan S.p.A. chiama aspo quella particolare sottozona che ha il compito di spiegare la materia prima - solitamente una lamina di metallo - con una velocità controllata, per permettere alla linea di produzione di lavorare tale materiale.
Per tale sottozona, è utile conosce la velocità di rotazione.

Figura 5.2: $Marker_G$ rappresentante un aspo

Rulli La sottozona rulli rappresenta i nastri di trasporto, utili a spostare del materiale, che può essere semilavorato oppure già imballato, verso un'altra sottozona della linea di produzione.

[1] link al sito ufficiale di Sketch©: https://www.sketch.com

Per tale sottozona è utile conoscere la velocità di rotazione.

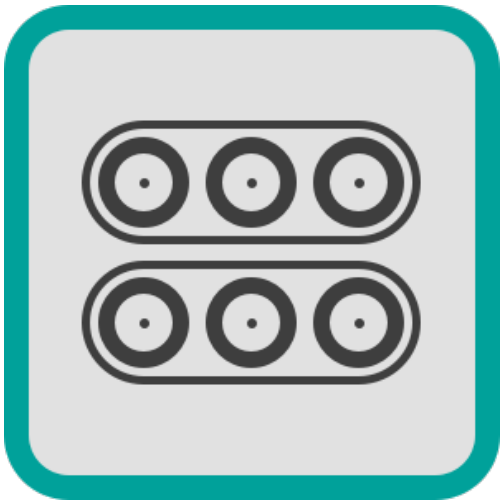


Figura 5.3: *Marker*$_G$ rappresentante un aspo

5.1.1.1 Unicità di un *marker*$_G$

Per essere perfettamente riconoscibile, un *marker*$_G$ deve essere il più univoco possibile.

Per essere sicuri che un'immagine sia riconoscibile da *ARCore*$_G$, *Google* ha messo a disposizione degli sviluppatori un *tool* chiamata *arcoreimg tool*. Questo è uno strumento che, una volta installato [2], è possibile utilizzare tramite terminale.

Comando per sistema operativo MacOS:

```
$ ./arcoreimg eval-img --input_image_path=image.jpg
```

Comando per sistema operativo Microsoft Windows:

```
$ arcoreimg.exe eval-img --input_image_path=image.png
```

Tale comando restituirà un valore intero nell'intervallo [0 - 100]: 0 indica che l'immagine non è per nulla riconoscibile, 100 indica che l'immagine non potrà mai essere confusa. Ovviamente 100 è il valore ottimale, ma *ARCore*$_G$ accetta immagini che hanno una valutazione superiore o pari a 75 [14].

5.1.1.2 Creazione oggetto *Label*

L'oggetto *Label* (N.d.R. "etichetta" in inglese) è un oggetto grafico che ha il computo di contenere e visualizzare del testo. Per tale motivo, è uno degli elementi fondamentali durante il design di qualsiasi prodotto software. La realizzazione di tale elemento era specificata dal vincolo **RV-O13** nella sezione **§3.2**.

Per la realizzazione, è stata disegnata un'immagine (utilizzando *Sketch*©): un rettangolo bianco, un bordo sottile di colore verde e gli angoli smussati. Tale immagine viene utilizzata come sfondo dell'oggetto grafico.

[2] link per il download: https://github.com/google-ar/arcore-android-sdk/tree/master/tools/arcoreimg

Dal punto di vista di implementazione su *Unity*, tale oggetto è rappresentato da un *Canvas*, ovvero un contenitore generico che raggruppava un'immagine, appunto come sfondo ed il testo effettivo. È stata utilizzata questa tecnica per avere un testo dinamico, ovvero avere la possibilità di cambiare il testo.

Una possibile alternativa sarebbe stata quella di disegnare il testo sull'immagine di background, per poi applicarla all'oggetto. Facendo così, però, si ha un costo, dal punto di vista delle prestazioni, molto più elevato e l'effetto finale non risultava piacevole.

Figura 5.4: Oggetto *Label*

5.1.1.3 Creazione oggetto *Gauge*

L'oggetto *Gauge* è un elemento semicircolare il quale indica un valore compreso tra un intervallo [minimo - massimo]. Il valore corrente viene indicato attraverso la rotazione di una lancetta. Il suo utilizzo è molto diffuso nella meccanica, in forma digitale od analogica: basti pensare all'indicatore di velocità di un'automobile. a realizzazione di tale elemento era specificata dal vincolo **RV-O14** nella sezione **§3.2**.

Per la realizzazione, sono state disegnate due immagini (utilizzando *Sketch*©): la prima, a forma di semicerchio di color verde, la quale rappresentava la scala dei valori rappresentabili, la seconda era la lancetta per la segnalazione del valore corrente.

Dal punto di vista dell'implementazione su *Unity*, tale oggetto era rappresentato da un *Canvas*, il quale conteneva le due immagini e alcuni oggetti testuali che indicavano:

- Il valore di minimo dell'intervallo;
- Il valore che rappresentava 1/4 dell'intervallo;
- Il valore che rappresentava la metà dell'intervallo;
- Il valore che rappresentava 3/4 dell'intervallo;
- Il valore di massimo dell'intervallo;
- Il valore corrente;

Per indicare il valore corrente, si associa al valore di minimo e di massimo il grado di rotazione e, attraverso la seguente formula si calcola il grado di rotazione del valore corrente:

$$\tilde{y} = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) \cdot (x_2 - x_1) + \tilde{x}$$

Dove:

- $\tilde{y}$: grado di rotazione del valore corrente;
- y_2: grado di rotazione del valore massimo dell'intervallo;
- y_1: grado di rotazione del valore minimo dell'intervallo;
- x_2: valore massimo dell'intervallo;
- x_1: valore minimo dell'intervallo;
- $\tilde{x}$: offset [3];

5.1.1.4 Creazione *Prefab*

Unity utilizza degli oggetti particolari, chiamati *Prefabs*. Un *Prefab* è un componente che permette di creare, configurare e salvare un *GameObject*[4].

Sono pensati per configurare una sola volta particolari *GameObjects*, i quali vengono riutilizzati più volte non solo nella scena ma in tutto il progetto. Questo permette di evitare ripetizioni di codice, ripetizioni di configurazioni, virando il progetto verso una strada mirata ad una manutenzione più facile.

Nel nostro caso, i *Prefabs* vengono utilizzati per la creazione di ogni scena che poi verrà mostrata attraverso AR. Nella fattispecie, ad ogni $marker_G$ verrà associato un unico *Prefab*, in modo tale che, nel momento in cui avviene il riconoscimento, tale evento sappia cosa mostrare all'utente.

5.1.1.5 Riconoscimento $marker_G$

Il riconoscimento di un $marker_G$ è una delle fasi più importanti dello sviluppo del progetto. Infatti, senza di questa funzione, tutto il resto non sarebbe possibile, in quanto si tratta dell'*entry point* per la visualizzazione degli oggetti utilizzando AR. Per tale motivo, questa operazione è coperta dal requisito **RV-O10**, specificato nella sezione **§3.2**.

Per l'esecuzione di tale operazione, è stata scritta la funzione **Algoritmo 5.1**, la quale controlla tutti i *frame* che la camera del dispositivo registra e controlla se l'*input* corrisponde ad un $marker_G$. Tale funzione fa parte della class *MarkTrackedImage*, la quale da delle variabili:

- *trackedImageManager*: di tipo privato *ARTrackedImageManager*, è colui che si occupa di controllare i *frame* provenienti dalla camera del dispositivi;
- *arPrefabs*: di tipo pubblic *GameObject[]*, è una lista che contiene tutti i *Prefabs* presenti nel progetto;
- *instantiatedPrefabs*: di tipo privato *Dictionary<string, GameObject>*, associa il *Prefab* con il nome dell'immagine corrispondente;

La funzione ha anche argomento, ovvero *ARTrackedImagesChangedEventArgs eventArgs*, in quanto essa viene associata all'evento *OnEnable* della scena principale. Significa che quando l'applicazione viene aperta, verrà eseguita tale funzione fintanto che l'applicazione non viene chiusa.

[3] Essendo un semicerchio, il valore minimo non corrisponde ad un valore [0/360; 90; 180; 270;], ma avrà una differenza da questi valori, per l'appunto, un offset. Nel nostro caso corrisponde a 35 gradi.

[4] Un *GameObject* è la classe base per tutti i gli oggetti che possono essere utilizzati in *Unity*

Algoritmo 5.1 Funzione *OnTrackedImagesChanged*

```
1.  for all trackedImage in eventArgs.added do
2.    imageName ← trackedImage.referenceImage.name;
3.    for all curPrefab in arPrefabs do
4.      imageName ← trackedImage.referenceImage.name;
5.      if curPrefab.name == imageName and instantiatedPrefabs not
        Contains(imageName) then
6.        newPrefab ← Instantiate(curPrefab);
7.        instantiatedPrefabs[imageName] ← newPrefab;
8.      end if
9.    end for
10. end for
11.
12. for all trackedImage in eventArgs.updated do
13.   if trackedImage.trackingState == TrackingState.Tracking then
14.     instantiatedPrefabs[trackedImage.referenceImage.name].active ← true;
15.   end if
16. end for
17.
18. for all trackedImage in eventArgs.removed do
19.   Destroy(instantiatedPrefabs[trackedImage.referenceImage.name]);
20.   instantiatedPrefabs.Remove(trackedImage.referenceImage.name);
21. end for
```

5.1.1.6 Creazione oggetto *Bottone*

L'oggetto *Bottone* è un oggetto grafico con la possibilità di essere premuto. Tale operazione genera un evento, al quale corrisponde un azione, che verrà eseguita. Nella fattispecie, ciò che richiedeva Dallan S.p.A. era di poter inviare comandi, che poi la linea di produzione avrebbe eseguito. Tale operazione era specificato dal vincolo **RV-F15**, specificato nella sezione **§3.2**.

Le immagini per i bottoni non sono state disegnate durante l'attività di stage, ma sono state fornite da Dallan S.p.A..

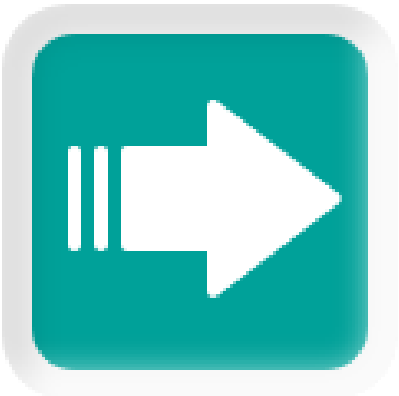


Figura 5.5: Oggetto *Button*

Il riconoscimento del tocco da parte dell'utente avviene tramite *Raycast* (**§6.2**). Quando il dispositivi rilevava un'interazione con l'oggetto *Button*, veniva invocato un metodo, precedentemente associato, il quale comunicava alla linea di produzione il comando da eseguire attraverso richiesta alle DLL_G.

5.1.1.7 Collegamento e recezione dati da DLL_G

Tutto il progetto verte sulla visualizzazione dei parametri provenienti da una linea di produzione. Per rendere possibile questa operazione, è necessario il collegamento alle DLL_G fornite da Dallan S.p.A.. Tale operazione era specificata dal vincolo **RV-O12**, specificato nella sezione **§3.2**.

Come spiegato precedentemente, tali DLL_G sono scritte utilizzando il linguaggio $C\#_G$, seguendo lo standard *.NET 2.0*.

La gestione delle interazioni e delle richieste viene affidata alla classe *PLCManager*: seguendo il *pattern Singleton*, essa fornisce tutti i metodi per la connessione, l'autentificazione e la richiesta di dati verso le DLL_G.

ConnectToDCU Avvia un tentativo di connessione alle DLL_G, restistuendo un oggetto *bool* che indica l'esito di tale operazione.

- **Input:** -
- **Output:**
  - *bool:* oggetto che rappresenta l'esito della connessione;
- **Visibilità:** pubblico;

DisconnectToDCU Esegue una disconnessione dalle DLL_G.

- **Input:** -
- **Output:** -
- **Visibilità:** pubblico;

GetAuthenticationToken Restituisce un *Token* di autentificazione temporaneo.

- **Input:**
  - *user:* oggetto stringa che rappresenta l'utente;
  - *password:* oggetto stringa che rappresenta la password dell'utente;
- **Output:**
  - *TokenResponse:* oggetto che rappresenta il *Token* di autentificazione;
- **Visibilità:** privato;

CheckApiAuthentication Controlla che si sia in possesso di un *Token* o che questo non sia scaduto.

- **Input:** -
- **Output:**
  - *bool:* oggetto che rappresenta lo stato dell'autorizzazione;
- **Visibilità:** protetto;

CreateClientFactory Crea un *client* per la connessione alle DLL_G.

- **Input:**
  - *serviceUrl:* oggetto stringa che rappresenta l'indirizzo di connessione;
- **Output:**
  - *RestClient:* oggetto che rappresenta il *client* appena creato per la connessione;
- **Visibilità:** pubblico statico;

GetVariableValue Metodo che esegue l'effettiva richiesta dei dati.

- **Input:**
  - *layoutName:* oggetto stringa che rappresenta l'oggetto alla quale eseguire la richiesta;
  - *variablePath:* oggetto stringa che rappresenta il percorso nella quale ritrovare i dati;
  - *clientId:* oggetto *Guid* che rappresenta il codice identificativo univoco del *client* che sta effettuando la richiesta;
- **Output:**
  - *object:* oggetto generico che rappresenta i dati richiesti;
- **Visibilità:** pubblico;

Login Metodo che esegue i controlli sull'autentificazione alle DLL_G e crea un *client*.

- **Input:** -
- **Output:**
  - *Guid?:* oggetto che rappresenta il codice identificativo univo del *client* creato;
- **Visibilità:** pubblico;

Logout Metodo che esegue la disconnessione alle DLL_G.

- **Input:**
  - *clientId:* oggetto Guid che rappresenta il codice identificativo univoco del *client* da disconnettere;
- **Output:** -
- **Visibilità:** pubblico;

5.1.1.8 Refactoring del codice seguendo un analisi statica

La pulizia del codice e la sua organizzazione possono risultare, nelle prime fasi di sviluppo, un processo inutile e complesso. Questa idea, con il corso del tempo, paga gli sforzi restituendo un codice semplice, snello e facilmente manutenibile.

Una prassi molto utilizzata è quella dell'analisi statica del codice: il controllo o analisi statica è una tecnica che consente di analizzare il codice di un programma attraverso un processo di valutazione basato sulla struttura, forma e contenuto. Non richiede dunque l'esecuzione del programma che si sta esaminando [10].

Tale operazione era specificata dal vincolo **RV-F17**, specificato nella sezione **§3.2**.

Per eseguire un'analisi statica del codice, non sono stati utilizzati strumenti di terze parti, ma si è scelto di utilizzare il *tool* integrato in *Visual Studio 2019*, con la configurazione di default. Durante la scrittura del codice veniva eseguito il controllo e, per tutto il codice che non rispettava i parametri, venivano segnalati con una leggera sottolineatura color verde ed un messaggio informativo, il quale suggeriva uno snippet[5] con che riportava le correzioni necessarie.

[5] Frammento di codice

Capitolo 6

Il funzionamento della Realtà Aumentata

In questo capitolo vengono analizzare ed approfondire gli aspetti tecnici che stanno alla base di tutto il funzionamento della Realtà Aumentata. Nello specifico verranno scorporati i calcoli che permettono la visualizzazione degli oggetti 3D e come è possibile intercettare un tocco su un oggetto 3D

6.1 Come vengono disegnati gli oggetti

Il mondo reale, per quanto riguarda il movimento ed il posizionamento degli oggetti, si presenta come uno spazio vettoriale a 3 dimensioni. Si ha, quindi, un vettore [*x, y, z*], dove x e y rappresentano rispettivamente larghezza ed altezza, mentre la variabile z rappresenta la profondità. Proprio la profondità e la caratteristica che contraddistingue un oggetto 3D da uno 2D.

6.1.1 Le coordinate di *OpenGL*$_G$

Allo stesso modo, *OpenGL*$_G$ utilizza un sistema di coordinate [*x, y, z*], il cui valore è compreso nell'intervallo [-1, 1]. Tutti gli oggetti che posseggono coordinate con valori che fuoriescono da tele intervallo, non saranno visibili.

Ciò che *OpenGL*$_G$ fa è ricevere in input delle coordinate che si riferiscono ad uno spazio definito dal programmatore e le trasforma in *normalized device coordinates* (NDC), ovvero le trasforma in coordinate 2D, sistema utilizzato dai schermi dei dispositivi.

Questa trasformazione avviene seguendo vari step. Vi sono 5 tipi di coordinate:

- Local space (spazio dell'oggetto)
- World space
- View space (spazio identico)
- Clip space
- Screen space

In tutti questi stati, i vertici dell'oggetto vengono trasformati, per poi essere disegnati a schermo.

6.1.2 Trasformazioni dei vertici di un oggetto

Le trasformazioni citate nella sezione **§6.1.1** avvengono tramite moltiplicazioni tra matrici oppure tra matrici ed un vettore.

La definizione di una matrice utile per il calcolo della trasformazione è:

$$A \in \mathbb{R}^{n \cdot n}, n = 4$$

Mentre la definizione di un vettore è:

$$v \in \mathbb{R}^{n \cdot 1}, n = 4$$

Il vettore possiede ha ordine $n = 4$, a differenza di un $n = 3$ come è facile pensare, perché, oltre alle coordinate $[x,\ y,\ z]$, possiede una variabile w tale che:

- se $w = 1$, il vettore $v = [x,\ y,\ z,\ 1]$ rappresenta un punto nello spazio;
- se $w = 0$, il vettore $v = [x,\ y,\ z,\ 0]$ rappresenta una direzione;

Le matrici più importanti sono *model, view* e *projection* le quali, moltiplicate tra loro restituiscono le NDC.

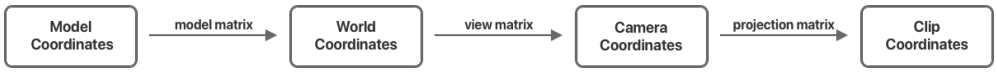


Figura 6.1: Trasformazione da coordinate del mondo reale a coordinate di un dispositivo

6.1.3 Model matrix

La *model matrix* trasforma la posizione in un modello in posizione nel mondo reale. Contiene tutte le traslazioni, rotazioni e scalatura che l'oggetto dovrà subire. In questo modo, il calcolo non verrà suddiviso per ogni operazione desiderata, ma, avendo una matrice che contiene tutti i dati necessari, basterà eseguire un singolo calcolo, incrementando notevolmente l'efficienza.

$$\text{Model Coordinates} \xrightarrow{\text{model matrix}} \text{World Coordinates}$$

6.1.3.1 Traslazione

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice T traslerà un vettore di T_x nella direzione x, T_y nella direzione y e T_z verso la direzione z.

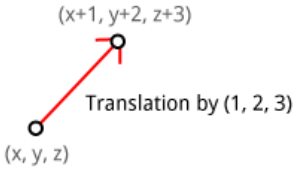


Figura 6.2: Esempio di traslazione di un punto

Esempio: se volessimo traslare il vettore colonna $v = [10, 10, 10, 1]$ di 10 unità rispetto all'asse x, avremmo [16]:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

6.1.3.2 Rotazione

Per questo calcolo, in realtà, sono necessarie tre matrici: una per la rotazione per l'asse x, per l'asse y e per l'asse z.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice R_x ruoterà un vettore di α gradi nell'asse x in senso antiorario.

$$R_y = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice R_y ruoterà un vettore di α gradi nell'asse y in senso antiorario.

$$R_z = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice R_z ruoterà un vettore di α gradi nell'asse z in senso antiorario.

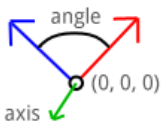


Figura 6.3: Esempio di rotazione di un punto

Esempio: riprendendo R_z, avremmo [15]:

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) \cdot x - \sin(\alpha) \cdot y \\ \sin(x) \cdot x + \cos(\alpha) \cdot y \\ z \\ 1 \end{bmatrix}$$

6.1.3.3 Scalatura

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice S scalerà un vettore di S_x nella direzione x, S_y nella direzione y e S_z verso la direzione z.

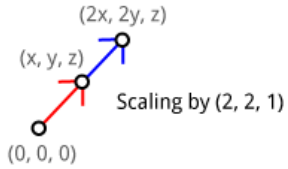


Figura 6.4: Esempio di scalatura di un punto

Esempio: se volessimo scalare un vettore colonna v di 2.0 unità rispetto a tutti gli assi, avremmo [16]:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2z \\ w \end{bmatrix}$$

Unendo le operazioni citate, la risultante *model matrix* si otterrà:

$$M = T \cdot S \cdot R$$

6.1.4 View matrix

OpenGL$_G$, di default, utilizza l'asse z come punto di vista della telecamera. Questo significa che uno spostamento in alto/basso è parallelo all'asse y mentre uno spostamento a destra/sinistra è parallelo all'asse x.

La *view matrix* controlla il modo in cui viene vista la scena e simula lo spostamento della camera. Il punto della camera è fisso, non può essere spostato, ed è posizionato nel punto [0, 0, 0]. Il fatto che la camera non può essere mossa implica che, per restituire all'utente il senso di movimento, è il mondo che si sposta o ruota attorno alla camera.

$$\text{World Coordinates} \xrightarrow{\text{view matrix}} \text{Camera Coordinates}$$

Una *view matrix* V moltiplicata per la *model matrix* M e per il vettore colonna v otteniamo:

$$v' = V \cdot M \cdot v$$

6.1.5 Projection matrix

In *OpenGL*$_G$ un oggetto ha le stesse dimensioni, indipendentemente dal punto di vista della camera. Questo rende la visualizzazione errata in quanto, nel mondo reale, se ci avviciniamo ad un oggetto le sue dimensioni vengono percepite più grandi, mentre vengono percepite più piccole se ci allontaniamo.

$$\text{Camera Coordinates} \xrightarrow{\text{projection matrix}} \text{Clip Coordinates}$$

Per ovviare a questo problema viene utilizzata una *orthographic projection*. Questa non modifica le dimensioni dell'oggetto ma ingrandisce o rimpicciolisce il volume. Tale tecnica è possibile se abbiamo oggetti di forma come il cubo, parallelepipedo.
Un esempio può essere:

$$P = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{near-far} & -\frac{near+far}{near-far} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dove: *top, bottom, left, right, near, far* rappresentano le posizioni dei piani di ritaglio.

Un'altra matrice che restituisce una visualizzazione simile al mondo reale è *projection matrix*, utilizzata per tutti gli oggetti con forma diversa rispetto a ciò che è stato spiegato sopra.

$$P = \begin{bmatrix} \frac{2 \cdot near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2 \cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{near-far} & -\frac{2 \cdot near+far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Una *projection matrix* viene, solitamente, specificata attraverso quattro parametri:

- *viewing angle* o *field of view* (FOV)
- *aspect*
- *near* e *far*

Dove: *near* e *far* rappresentano i punti dei piani di ritaglio, rispettivamente più vicini e più lontani, mentre:

$$top = near \cdot \tan\left(\frac{\pi}{180} \cdot \frac{FOV}{2}\right)$$

$$bottom = -top$$

$$right = top \cdot aspect$$

$$left = -right$$

Dopo aver applicato una trasformazione attraverso la *projection matrix* si ottiene un vettore contenente le *clip coordinates*. Queste, però, non possono ancora essere utilizzate per diventare NDC.

Prima di tale operazione, si deve applicare la *perspective division*: il vettore colonna contenente le *clipping coordinates* risultanti dalla *projection matrix* ha la coordinata w con valore diverso da 1 (e da 0). Tale numero rispecchia l'effetto di vicinanza o lontananza dalla camera. Ora basta dividere le coordinate *x, y, z* per la coordinata w, in modo da ottenere un vettore normalizzato:

$$v_{normalized} = \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{bmatrix}$$

Ora il vettore colonna $v_{normalized}$ conterrà le variabili *x, y* con valori compresi nell'intervallo [-1, 1], come specificato nella sezione **§6.1**.

Una *projection matrix* P moltiplicata per la *model matrix* M, una *view matrix* V e per il vettore colonna v otteniamo:

$$v' = P \cdot V \cdot M \cdot v$$

Ora si ha un vettore contenente tutte le informazioni riguardanti le coordinate dei vertici dell'oggetto che si desidera visualizzare a schermo.

6.2 Interagire con un oggetto 3D

Oltre alla visualizzazione di oggetti, AR permette anche l'interazione con questi. Per interazione si intende la possibilità di "toccarli", magari lanciando un evento al tocco, trattando l'oggetto come se fosse un bottone visualizzato in 3D.

L'interazione, però, non è un'operazione banale: per determinare se un oggetto è stato toccato, si devono conoscere le sue coordinate del mondo reale. Infatti, quando un utente preme sullo schermo del suo dispositivo con l'intenzione di relazionarsi con

la scena 3D, le informazioni che si hanno sono soltanto le coordinate 2D inerenti al tocco sullo schermo, ma non si conosce ciò che è presente nel mondo reale.

Per determinare se un oggetto 3D è stato toccato si utilizza una tecnologia chiamata *Raycast*.

6.2.1 Raycast

Come suggerisce il nome, al momento del tocco di un utente nello schermo del suo di dispositivo, viene generato un raggio. Un raggio è una linea immaginaria, che ha come punto di origine la camera del dispositivo, e si propagano in una traiettoria dritta. Proprio come se fosse un raggio, nel momento in cui incontra un ostacolo, ritorna le informazioni raccolte, come la distanza dell'oggetto ed i dati dell'oggetto.

L'origine del raggio è definita da un vettore contenente le tre coordinate x, y, z definito come $v_{origin} = [x_O,\ y_O,\ z_O]$. Anche la direzione è definita come un vettore contenente le tre coordinate: $v_{direction} = [x_D,\ y_D,\ z_D]$.

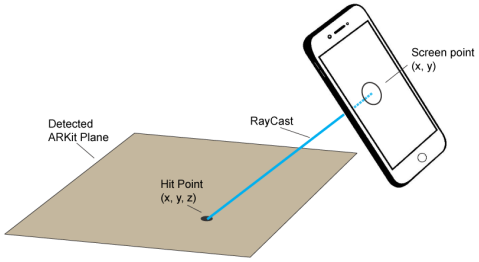


Figura 6.5: Esempio di Raycast stilizzato

6.2.2 RaycastHit

RaycastHit è una classe speciale. Essa infatti è la classe che contiene tutte le informazioni del risultato del *raycasting*, ovvero contiene tutte le informazioni che si generano al momento della collisione tra il raggio immaginario generato e l'oggetto - od un piano - colpito.

6.2.3 Codice per intercettare un oggetto 3D

Unendo le informazioni descritte nelle sezioni precedenti, il codice per determinare il tocco su un oggetto 3D risulta essere:

Algoritmo 6.2 Algoritmo che esegue un *raycasting*

```
1.  touch ← Input.touch
2.  if touch.phase ≠ Began then
3.      return
4.  end if
5.  hits ← {ARRaycastHit}
6.  if RaycastManager.Raycast(touch.position, hits) then
7.      hit ← hits[0]
8.      if hit.name == SearchedObjectName then
9.          // ...
10.     end if
11. end if
```

Il codice descritto in **Algoritmo 6.2** mostra come richiamare le funzioni di *raycasting*. Il primo passo è istanziare la variabile *touch*, contenente le informazioni del tocco, per poi controllare che il tocco sia nella fase iniziale. Questo è necessario per avere la certezza di eseguire il corpo dell'algoritmo solo una volta e non per ogni fase del tocco. Se la fase è diversa da *Began* l'algoritmo viene interrotto.

Si può notare che per generare il raggio immaginario viene utilizzata la classe *RaycastManager*. Questa classe fa parte delle API_G presenti in *AR Foundation* (**§4.2.3**).

Questo *Manager* è responsabile per tutto ciò che riguarda la creazione e gestione delle funzionalità di *raycasting*. Nel codice descritto viene utilizzato per la creazione del raggio, passando come parametri il vettore contenente le coordianate 2D, inerenti al tocco sullo schermo, e una variabile *hits* di tipo *{ARRaycastHit}*, ovvero una lista contenente elemento di tipo *ARRaycastHit*. Questa variabile viene passata in quanto, se viene rintracciata una collisione, conterrà le informazioni di tutti gli elementi con la quale è avvenuta la collisione.

La funzione *Raycast* ritorna un tipo *booleano*, ovvero *vero* se viene incontrato un oggetto nella strada del raggio, *falso* altrimenti. Per questo motivo viene richiamata all'interno di un *if-statement*.

Operazione molto importante è l'assegnazione *hit* ← *hits[0]*: facendo questo ci si interessa dell'oggetto più vicino al dispositivo che si è incontrato.

Fatto ciò, si controlla che il nome dell'oggetto *hit* corrisponda con il nome dell'oggetto che si sta cercando, in modo tale da avere la certezza di interagire con ciò che si sta effettivamente cercando. Nel caso questo sia diverso, non viene eseguito nulla.

Seguendo questo piccolo pezzo di codice si è potuto creare l'oggetto *Button* richiesto da Dallan S.p.A.. Infatti, ogni oggetto 3D visualizzato tramite AR può essere trattato come se fosse un classico bottone 2D.

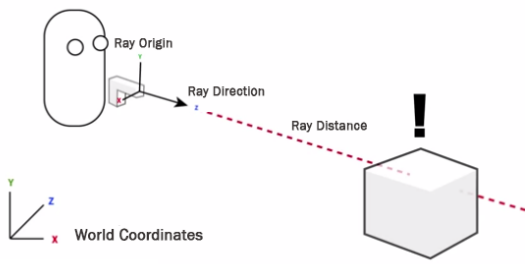


Figura 6.6: Esempio di RaycastHit

Capitolo 7

Conclusioni

7.1 Consuntivo finale

In questa sezione si riportano le ore effettive consuntivate durante l'attività di stage. Esse sono state schematizzate come segue:

Ore	Settimana	Descrizione
40	1	Studio applicazione attuale e codifica della pagina di visualizzazione delle sottozone di una linea di produzione
40	2	Creazione ed implementazione della pagina che consente la lettura di un *QR Code* e l'apertura della pagina inerente alla sottozona
32	3	Creazione oggetti *custom* come *TextBox*, *NumberBox* e *Button* per l'applicazione esistente
8	3	Inizio studio tecnologie di AR
40	4	Approccio di codifica dell'applicazione utilizzando *Xamarin*$_G$
32	5	Installazione *Unity* e studio del *framework*$_G$
8	5	Creazione struttura progetto
16	6	Creazione, design ed implementazione oggetto *Label*
24	6	Creazione, design ed implementazione oggetto *Gauge*
8	7	Design immagini per i *marker*$_G$
32	7	Creazione, design ed implementazione oggetto *Button*

Ore	Settimana	Descrizione
32	8	Import ed implementazione delle DLL_G proprietarie di Dallan S.p.A.
8	8	Pulizia del codice, analisi statica ed organizzazione della struttura del progetto
Totale: 320		

Tabella 7.1: Tabella di consuntivazione delle ore lavorate.

7.2 Raggiungimento degli obiettivi

Durante l'attività tutti i requisiti obbligatori sono stati soddisfatti, mentre per quanto riguarda i vincoli facoltativi, non è stata sviluppata l'applicazione per dispositivi *iOS*. Tutta la pianificazione la si può trovare alla sezione **§2.2**.

Il raggiungimento degli obiettivi, rappresentato dal grado di implementazione dei requisiti, è schematizzato in **Tabella 7.2** e **Tabella 7.3**.

Codice	Stato
RV-F6	non implementato
RV-F7	non implementato
RF-O8	implementato
RV-F11	implementato
RV-F14	implementato
RV-F15	implementato
RV-F16	implementato
RQ-F17	implementato
RQ-F18	implementato
RQ-F19	implementato
RP-F20	implementato

Tabella 7.2: Stato dei requisiti facoltativi richiesti

Codice	Stato
RV-O1	implementato
RV-O2	implementato
RV-O3	implementato
RV-O4	implementato
RV-O5	implementato
RV-O9	implementato
RV-O10	implementato
RV-O12	implementato
RV-O13	implementato

Tabella 7.3: Stato dei requisiti obbligatori richiesti

Il dato sulla percentuale dei requisiti richiesti completati è dato da **Tabella 7.4**.

Requisito	Totale	Percentuale completamento
Obbligatorio	9/9	100%
Facoltativo	9/11	81.8%

Tabella 7.4: Completamento dei requisiti

7.3 Conoscenze acquisite

Durante l'attività di stage ho avuto modo di toccare con mano tecnologie mai utilizzate prima di allora. Un esempio può essere il *framework*$_G$ *Xamarin*$_G$ per lo sviluppo di applicazioni multipiattaforma.

Lo studio principale, però, si e concentrato sulla Realtà Aumentata: lo sviluppo di un'applicazione che permettesse l'implementazione di questo tipo di interazione con l'utente è stata un'esperienza nuova per me, dandomi nozioni che prima non conoscevo, partendo dai diversi motori grafici fino ad arrivare a come vengono calcolati i vertici di un oggetto per poi disegnalo a schermo.

In otto settimane ho avuto anche la possibilità di frequentare un vero luogo di lavoro, vivendo a pieno la realtà del lavoro, avendo orari e scadenze. Questo comporta anche uno studio più approfondito riguardo le tecnologie da implementare per lo sviluppo di un prodotto, non pensando soltanto al risultato finale, ma avendo una visione più ampia su tutto ciò che riguarda tempistiche - e di conseguenza i costi - di un progetto, pensando anche al target di mercato verso la quale ci si vuole posizionare.

Nonostante questo, ho avuto la possibilità di utilizzare la mia creatività e le mie idee per mettere un tocco personale su un progetto il quale ha la potenzialità di essere prodotto commercialmente.

7.4 Valutazione personale

Tale esperienza, dal mio punto di vista, non può che essere valutata positivamente. Partendo dall'idea di progetto, utile ed interessante, ma anche grazie all'attenzione e l'aiuto che Dallan S.p.A. mi ha riservato.

Aver dovuto affrontare delle problematiche mi ha aiutato ad imparare come, nel mondo del lavoro, ci sia un'alta probabilità di inconvenienti e la loro risoluzione deve essere il più veloce possibile. Dallan S.p.A., non avendo conoscenze in materia di Realtà Aumentata ha cercato di fornirmi tutti i materiali possibili, lasciando a me, però, il compito di ragionare su quali soluzioni fossero le migliori, ascoltando i loro vincoli e requisiti. Questo mi ha dato una piccola autonomia che ha fatto crescere le mie conoscenze tecniche e la mia consapevolezza delle responsabilità.

Acronimi ed Abbreviazioni

API Application Program Interface. 47

App Applicazione. 7

AR Augmented Reality (Realtà Aumentata). ix, 2, 3, 8–16, 21, 22, 27, 38, 40, 41, 47

DLL Dynamic Link Library. 47

GUI Graphical User Interface. 47

Ing. Ingegnere. 7, 19

JNI Java Native Interface. 17

JVM Java Virtual Machine. 17, 18

NDC Normalized Device Coordinates. 33, 34, 38

PLC Programmable Logic Controller. 7, 8

S.p.A. Società per Azioni. xii, 1, 3, 7–9, 15, 19, 20, 23, 24, 28, 29, 40, 42, 44

SDK Software Development Kit. 21, 48

UML Unified Modeling Language. 48

Glossario

API In informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 4, 15–17, 40, 48

ARCORE *framework*$_G$ motore grafico da *Google* per la progettazione e codifica di applicativi che permettano l'utilizzo di AR su dispositivi *Android*. ix, xi, xii, 15, 16, 20–22, 25

ARKIT *framework*$_G$ motore grafico da *Apple* per la progettazione e codifica di applicativi che permettano l'utilizzo di AR su dispositivi *iOS*. xii, 15, 16, 20–22

CODEBASE In informatica, con il termine *"codebase"* si indica l'insieme di tutto il codice sorgente, utile per la costruzione di applicazioni o determinati componenti, ed i file di configurazione e proprietà. 15

C# Linguaggio di programmazione, sviluppato da *Microsoft*. È utilizzato nel framework *.NET Standard*, ma anche per lo sviluppo di applicazioni web, desktop, mobile e videogiochi. 5, 7, 8, 15, 18, 20, 29, 48

DLL In informatica, una *DLL, Dynamic Link Library* è una libreria software che non viene collegata staticamente ad un eseguibile in fase di compilazione, ma che viene caricata dinamicamente in fase di esecuzione. Indica anche l'estensione dei file che le rappresentano in ambiente Microsoft Windows. x, 7–9, 12–14, 23, 29–31, 42

FRAMEWORK In informatica e specificamente nello sviluppo software, un framework (che può essere tradotto come struttura o quadro strutturale) è un'architettura logica di supporto sulla quale un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. xi, 5, 7, 8, 15–18, 22, 41, 43, 47, 48

GUI In informatica è un tipo di interfaccia utente che consente l'interazione uomo-macchina in modo visuale utilizzando rappresentazioni grafiche piuttosto che utilizzando i comandi tipici di un'interfaccia a riga di comando. 5

Java In informatica *Java* è un linguaggio di programmazione ad alto livello, orientato agli oggetti ed a tipizzazione statica, il quale si appoggia sull'omonima piattaforma software di esecuzione, specificamente progettato per essere il più possibile indipendente dalla piattaforma hardware di esecuzione.. xi, 17–19

MARKER Segnaposto utile per il riconoscimento della zona interessata, zona di ancoraggio per gli elementi 3D in Realtà Aumentata. ix–xi, 1, 3, 9, 11–15, 23–25, 27, 41

OPENGL (*Open Graphics Library*) Definisce delle API_G per diversi linguaggi e piattaforme per la realizzazione di computer grafica 2D e 3D. È la più diffusa nello sviluppo software date le alte prestazioni che riesce a fornire. x, xi, 16, 33, 37

Open-Source In informatica, con il termine *Open-Source* si indica un sistema di sviluppo software decentralizzato basato sulla condivisione dei file sorgenti. Tipicamente tale condivisione avviene senza scopo di lucro, ma sotto speciali licenze. 17

SDK In informatica, un *SDK* è un kit per lo sviluppo di software. Contiene la documentazioni e strumenti che consentono l'utilizzo di un componente o di una libreria all'interno di un ambiente di programmazione. 20

UML In ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. 9

WRAP In informatica, con il termine wrap si indica un modulo software che "riveste" (da qui la parola inglese wrap) un altro, facendo da tramite tra i soggetti esterni ed il modulo "rivestito". 15, 16

XAMARIN $Framework_G$ che attraverso un codice scritto con il linguaggio $C\#_G$, permette di scrivere applicazioni native Android, iOS e Windows con interfacce utenti native, e condividendo il codice su diverse piattaforme. ix, xi, 4, 15–17, 20, 41, 43

Bibliografia

Riferimenti bibliografici

[10] Eleonora Pirri. *Un metodo di Analisi Statica di Qualità del Software*. https://amslaurea.unibo.it/20576/1/pirri_eleonora_tesi.pdf. Università di Bologna, 2019/2020.

Siti web consultati

[1] *About AR Foundation*. URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@5.0/manual/index.html.

[2] *AR Foundation in Unity*. URL: https://www.kodeco.com/14808876-ar-foundation-in-unity-getting-started#toc-anchor-002.

[3] *ARCore Documentation*. URL: https://developers.google.com/ar/develop.

[4] *Chapter 3: Viewing*. URL: https://www.glprogramming.com/red/chapter03.html.

[5] *Coordinate System*. URL: https://learnopengl.com/Getting-started/Coordinate-Systems.

[6] *Glossario dei termini informatici più comuni*. URL: https://vitolavecchia.altervista.org/risorse-e-altro/glossario-informatica/.

[7] *jni4net - bridge between Java and .NET*. URL: http://jni4net.com.

[8] *Matrices - projection, view, model*. URL: https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/.

[9] *Physics Raycast*. URL: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html.

[11] *Prefabs*. URL: https://docs.unity3d.com/Manual/Prefabs.html.

[12] *Realtà Aumentata*. URL: https://it.wikipedia.org/wiki/Realt%C3%A0_aumentata.

[13] *Realtà Aumentata - Cos'è?* URL: https://www.intelligenzaartificiale.it/realta-aumentata/.

[14] *The arcoreimg tool*. URL: https://developers.google.com/ar/develop/augmented-images/arcoreimg.

[15] *Transformations*. URL: https://learnopengl.com/Getting-started/Transformations.

[16] *Tutorial 3: Matrices*. URL: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/.

[17] *Unity Documentation - GameObject*. URL: https://docs.unity3d.com/ScriptReference/GameObject.html.

[18] *Unity Lean - Prefabs*. URL: https://learn.unity.com/tutorial/prefabs-e.

[19] *Unity (motore grafico) - Wikipedia*. URL: https://it.wikipedia.org/wiki/Unity_(motore_grafico).

[20] *Unity Real-Time Development Platform*. URL: https://unity.com.