UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

Progettazione e sviluppo di Web-app di Realtà Aumentata a supporto dell'Industria 5.0

Design and development of Augmented Reality Web-apps to support Industry 5.0

Relatore:
Prof.ssa Paola Pierleoni

Tesi di Laurea di:
Federico Babbini

Correlatore:
Prof. Emanuele Storti

Today is only one day in all the days that will ever be. But what will happen in all the other days that ever come can depend on what you do today. It's been that way all this year. It's been that way so many times.

(Ernest Hemingway)

Sommario

Il presente elaborato vuole dimostrare come sfruttare la tecnologia della Realtà Aumentata, inserendola in un contesto industriale moderno, quale l'Industria 5.0, ed affiancandola all'utilizzo di dispositivi mobile, come smartphone e tablet, mediante un'applicazione Web.

Il lettore verrà guidato alla comprensione di questa tesi tramite una parte introduttiva, dove vedrà una panoramica della Realtà Aumentata nel *Capitolo 1 – Introduzione* e nel *Capitolo 2 – Realtà Aumentata*. Si forniranno definizioni in materia, facendo riferimenti ai contesti storici nei quali questa tecnologia è comparsa per la prima volta, e si farà chiarezza sul concetto di Realtà Virtuale contrapponendolo a quello della Realtà Aumentata.

Dato che si vuole mostrare come Realtà Aumentata e Industria 5.0 possono lavorare in simbiosi, nel *Capitolo 3 – Industria 5.0* si fornisce una descrizione di questa nuova realtà. Dopo un breve cenno storico, verranno illustrate le caratteristiche principali di questo settore, differenziando questo contesto con il precedente, quello dell'Industria 4.0.

Si procede poi con il *Capitolo 4 – Sistemi di Realtà Aumentata nei dispositivi mobile* dove si parlerà finalmente della *Mobile Augmented Reality*, un concetto chiave di questo elaborato, in quanto è la fusione tra Realtà Aumentata e dispositivi mobile. Si evidenzieranno gli sviluppi tecnologici che hanno condotto verso questa fusione ed i vantaggi che questo affiancamento ha portato.

Dal *Capitolo 5 – Analisi dei requisiti* si entra nel vivo del progetto realizzato per questo elaborato di tesi, chiamato *ARSupport*. Verranno qui analizzati e spiegati i requisiti, funzionali e non, che hanno guidato la progettazione dell'applicazione. Si intuiranno le funzioni che l'applicativo dovrà espletare e verrà fatta chiarezza sulle differenze tra applicazioni native, ibride e Web, evidenziando i motivi che hanno condotto alla decisione di realizzare una Web-app. Verranno introdotti tre casi di studio che sono stati presentati per poter apprezzare le potenzialità dell'applicativo: *ARSupport-office*, *ARSupport-product* e *ARSupport-engine*.

Gli strumenti che sono stati utilizzati per la realizzazione dell'applicazione sono illustrati nel *Capitolo 6 – Strumenti utilizzati*. In particolare, verranno descritti due framework opensource, ideali per realizzare esperienze di Realtà Aumentata nel

Web, che sono: *AR.js* e *A-Frame*. Questi strumenti, che si basano sui linguaggi JavaScript e HTML, lavorano insieme e questa loro compatibilità è stata utilizzata per poter riconoscere dei *target* nel mondo reale (che, come si vedrà, possono essere dei *marker*, delle immagini o anche degli oggetti reali), funzionali all'aggiunta di contenuti virtuali.

L'architettura dell'applicativo viene discussa nel *Capitolo 7 – Struttura dell'applicazione*, elemento fondamentale per la stesura di questo progetto. L'applicazione si inserisce in un contesto di *Internet of Things*, nel quale sono presenti tre soggetti collegati alla rete: la Web-app, dei sensori simulati e uno *storage* di dati. Le funzionalità della Web-app verranno discusse in seguito, mentre qui si descriveranno i sensori simulati, che forniscono dati riguardanti il macchinario a cui sono idealmente connessi, e il meccanismo di *storage*, ovvero un database esterno per questi dati, svolto tramite un servizio Web. Il vantaggio principale di questo approccio sta nel fatto che le informazioni ricevute dal database esterno vengono visualizzate direttamente sul dispositivo che l'utente sta utilizzando, al momento della scansione di un determinato target.

I casi di studio presentati vengono descritti in dettagli nel *Capitolo 8 – Funzionalità dell'applicazione*, dove si percorreranno i passaggi salienti che hanno portato alla realizzazione di ogni caso. Si capirà che *ARSupport* non è semplicemente un'applicazione Web, ma è piuttosto un approccio, che può essere adattato a diversi contesti, sia industriali che non. Il primo caso di studio presentato, *ARSupport-office*, infatti, si discosta dallo scenario industriale, in quanto i contenuti grafici mostrati fanno riferimento ad un ambiente d'ufficio. L'obiettivo, tuttavia, è chiaro: l'applicazione, interagendo con un servizio Web, fornisce dati all'utente riguardanti lo stato del macchinario di cui egli sta scannerizzando il *marker*, posto sul macchinario stesso, e mostra un contenuto virtuale che, in base alla risposta dei sensori, può cambiare il suo comportamento per guidare l'utente nell'attività da svolgere. *ARSupport-product*, il secondo caso di studio proposto, svolge le stesse funzioni del caso precedente, andando, però, ad 'aumentare' un macchinario reale con oggetti virtuali, oggetti che simbolizzano il materiale che dovrà essere installato nel macchinario in questione. Nell'ultimo caso di studio, *ARSupport-engine*, ci si discosta dai precedenti, eliminando l'interazione con il servizio Web. Qui lo scopo è andare ad aumentare immagini 2D, riuscendo ad offrire contenuti virtuali che hanno lo scopo di facilitare la comprensione dell'immagine stessa. Nel particolare, è stato utilizzato un manuale di un motore, che può essere quello di un'automobile o di un sistema per la produzione di energia. Tramite l'applicazione, l'utente, scansionando l'immagine, riesce a

visualizzare componenti virtuali in 3D del motore stesso, componenti che non sono immediatamente visibili dall'osservazione dell'immagine. Nonostante un'applicazione di Realtà Aumentata non richieda sforzi eccessivi dal punto di vista dell'interfaccia grafica, poiché, per definizione della stessa, la grafica che viene utilizzata è per lo più l'ambiente circostante visto dalla fotocamera del dispositivo, in questo capitolo verranno mostrati anche alcuni aspetti che sono stati realizzati per garantire all'applicazione un carattere più *user-friendly*. Il lettore potrà comprendere come la grafica realizzata aiuti l'utente che usa la Web-app a interagire con il contenuto virtuale oppure a visualizzare i dati forniti dai sensori.

L'elaborato si chiude con il *Capitolo 9* – Conclusioni. Una parte conclusiva, dove si valutano i risultati ottenuti, accennando ad alcuni miglioramenti che possono essere implementati.

Indice

Capitolo 1
Introduzione

Nel corso dell'ultimo ventennio, l'evoluzione della tecnologia informatica ha portato ad una transizione dall'ufficio stazionario con computer desktop all'utilizzo di apparecchi mobile. Le vendite di smartphone e tablet hanno superato di gran lunga quelle dei PC convenzionali, sia fissi che portatili [2], *handheld-device*[1] di gran lunga più comodi e più facili da usare rispetto ai dispositivi convenzionali.

È in questo contesto che aumenta l'interesse di includere il mondo fisico, reale, nella nostra esperienza computazionale e viceversa, situazioni fino a questo momento impossibili da implementare nei normali computer fissi. Si ottiene così l'effetto di migliorare l'esperienza stessa, rendendola più immersiva, avvincente e accessibile. Cresce, quindi, l'interesse verso ciò che chiamiamo *Realtà Aumentata* (AR, dall'inglese *Augmented Reality*).

Come verrà descritto successivamente, la storia di questa tecnologia nasce intorno la seconda metà del secolo precedente, ma l'aumento esponenziale dell'interesse a riguardo lo si può apprezzare solo negli ultimi anni. Questo grazie a tre fattori in particolare [18]:

- I miglioramenti nelle prestazioni degli *handheld-device* insieme alla loro integrazione con sensori.
- Avanzamenti nell'ambito della *Computer Vision*, tecnologia che verrà utilizzata nelle applicazioni di Realtà Aumentata.
- La nascita di dispositivi dedicati alla AR (per esempio Microsoft Hololens e Google Glass) parallelamente a potenti SDK[2] (su tutti ARCore di Google e ARKit di Apple).

[1] Dispositivo portatile
[2] Software Development Kit, ovvero, Kit per lo sviluppo di Software.

In Figura 1.1 sono mostrate le tappe storiche più importanti che hanno portato la Realtà Aumentata allo stato in cui è attualmente.

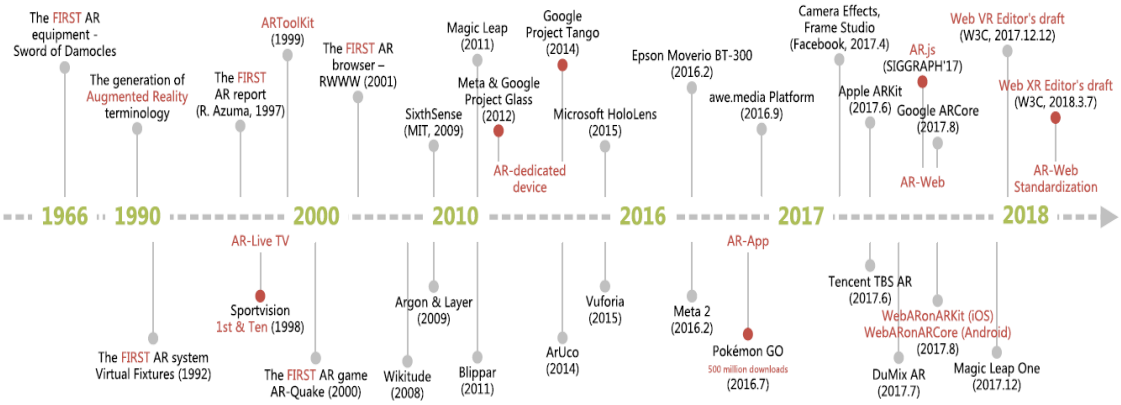


Figura 1.1: Evoluzione storica dell'AR [18].

Per avvalorare la tesi sull'importanza della Realtà Aumentata ad oggi, si può prendere come esempio l'applicazione mobile *Pokèmon GO*, un gioco in Realtà Aumentata *location-based*[3] che ha raggiunto un picco di oltre 200 milioni di utenti attivi nell'anno del suo lancio, avvenuto a luglio 2016, per poi raggiungere oggi un totale di oltre 500 milioni di download e un profitto superiore a un miliardo di dollari [19].

Tuttavia, in questa tesi ci si pone come obiettivo principale quello di mostrare che la Realtà Aumentata non è soltanto un terreno fertile per l'intrattenimento, ma può essere implementata per applicazioni in altri settori. In particolare, verrà proposta un'applicazione Web, orientata per i dispositivi mobile, che sfrutta la AR come supporto a servizi pensati per l'Industria 5.0, dal nome *ARSupport*.

La Realtà Aumentata non è ancora largamente diffusa a livello industriale, ma, data l'evoluzione tecnologica negli ambienti di produzione industriale che sono diventati sempre più digitali ed interattivi, essa possiede il potenziale di diventare parte integrante della nuova Industria.

[3] Ovvero in grado di sfruttare i sensori GPS del dispositivo per valutare la posizione corrente

Capitolo 2
Realtà Aumentata

In questo capitolo si vorrà rispondere alla domanda su cos'è la Realtà Aumentata.

Per far ciò si partirà da un contesto più generale, che comprende il mondo reale in cui viviamo, ed un mondo virtuale, opportunamente creato tramite computer, chiamato Realtà Virtuale, dall'inglese *Virtual Reality* VR, ma anche *Virtual Environment* VE (entrambi i termini sono adoperati dalla comunità scientifica[4]) [14]. Si vedrà come la Realtà Aumentata si pone all'interno di questi due contesti opposti.

Il lettore verrà poi guidato a una comprensione delle due realtà, Virtuale e Aumentata, fornendo definizioni formali, evidenziando gli aspetti in contrapposizione tra esse e illustrando alcune delle tappe storiche fondamentali per aver contribuito allo stato della tecnologia attuale, accennando, infine, ad alcune applicazioni realizzate finora mediante questi approcci.

[4] In questo elaborato si predilige l'utilizzo del termine *Virtual Reality*.

2.1 Continuum Realtà-Virtualità e Realtà Mista

Osservando la Figura 1.1 del Capitolo precedente, si può notare che la striscia temporale presa in considerazione per valutare l'evoluzione della Realtà Aumentata inizia nel 1966, con il primo esperimento che sfrutta questa tecnologia, chiamato 'Spada di Damocle', di cui si parlerà nel prossimo Paragrafo.

A quel tempo, tuttavia, non vi era una definizione chiara né di Realtà Aumentata né di Realtà Virtuale ed i due termini venivano spesso confusi [12]. Con il passare degli anni l'interesse in materia aumentava ed emerse quindi la necessità di far chiarezza.

Per rispondere a questa necessità intervenne Paul Milgram, che nel 1994 prova a fornire una tassonomia di queste tecnologie emergenti introducendo due nuovi concetti: il 'Continuum Realtà-Virtualità' (dall'inglese *Virtual-Reality Continuum*) e la Realtà Mista (dall'inglese *Mixed Reality*, MR).

In Figura 2.1 viene mostrato lo schema del Continuum, ai cui estremi si trovano due mondi diametralmente opposti: a sinistra il del mondo reale, in cui viviamo e a cui l'osservatore riesce ad accedere anche senza l'uso di opportuni display o dispositivi elettronici [11]; a destra quello della Realtà Virtuale, dove l'osservatore vi è totalmente immerso e, possibilmente, con la capacità di interagirvi [12].

Tutto ciò che si trova in mezzo a questo Continuum è chiamato Realtà Mista, un luogo dove entrambi i mondi, reale e virtuale, si mescolano tra loro e per accedervi è necessario un monitor o un qualsiasi dispositivo elettronico opportunamente designato.

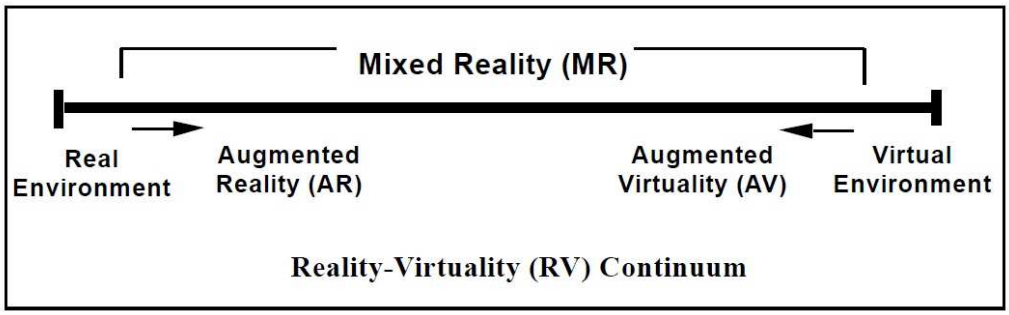


Figura 2.1: Schema del continuum Realtà-Virtualità [11].

Milgram et al. propongono una classificazione degli ambienti che appartengono alla Realtà Mista basata su 3 proprietà [11]:

- *Reality*, ovvero la proprietà di un certo ambiente di essere principalmente reale o principalmente virtuale, cioè creato artificialmente tramite computer.
- *Immersion*, ovvero quanto l'osservatore (anche chiamato, dagli autori, partecipante) è immerso all'interno dell'ambiente.
- *Directness*, ovvero se gli oggetti dell'ambiente sono visibili direttamente o per mezzo di strumenti elettronici.

Nonostante gli stessi autori affermino che il termine Realtà Mista non sia largamente diffuso, è in questo contesto che viene individuata la Realtà Aumentata[5], ossia un ambiente di realtà mista, principalmente reale, dove l'osservatore è in grado di accedere tramite uno schermo elettronico e di interagire con gli oggetti presenti (si veda il Paragrafo 2.3).

[5] In opposizione alla Realtà Aumentata si trova la Virtualità Aumentata (dall'inglese *Augmented Virtuality*, AV), un ambiente principalmente virtuale, arricchito di contenuti virtuali. La sua trattazione esula da questo elaborato ed in generale non ha trovato finora molto spazio in contesti applicativi.

Un'applicazione di Realtà Mista, per essere tale, deve possedere tre caratteristiche fondamentali, che la rendono un'applicazione grafica ed interattiva, con risposte in tempo reale. La grafica dell'applicazione si occupa di renderizzare[6] e visualizzare gli oggetti tridimensionali con i quali l'utente che usufruisce dell'applicazione interagisce. La risposta del contenuto virtuale all'azione dell'utente deve avvenire in tempo reale, così che l'applicazione sembri apparire continua agli occhi di chi la sta utilizzando.

[6] Dal verbo inglese *rendering* che si riferisce al processo di generazione dell'immagine a partire dalle primitive grafiche

2.2 Realtà Virtuale

La Realtà Virtuale, conosciuta anche come 'Realtà Artificiale', è un'esperienza totalmente immersiva, coinvolgente ed interattiva di una realtà alternativa, ottenuta mediante l'utilizzo di una qualche struttura informatica che permette all'utente di interagire con oggetti simulati all'interno di questo ambiente, come se fossero reali [13].

Come già accennato in precedenza, la nascita di questa tecnologia risale alla seconda metà del secolo scorso, in particolare quando Ivan Sutherland, nel 1965, scrisse il suo saggio *'The Ultimate Display'*. Nel trattato l'autore, proponendo un'innovativa descrizione di come si potessero realizzare dei display 'immersivi', anticipa quello che oggi consideriamo come Realtà Virtuale, e di conseguenza anche Realtà Aumentata, pur non citando né l'una né l'altra [2]. Si intuì come i computer potessero avere il "controllo dell'esistenza della materia" e interpretare il movimento dell'occhio umano, adattandosi di conseguenza. Parafrasando Sutherland, il *ultimate display*, con un'opportuna programmazione, può essere il 'Pese delle Meraviglie' per Alice, un mondo che sembra reale e risponde alle azioni dell'osservatore come tale [10].

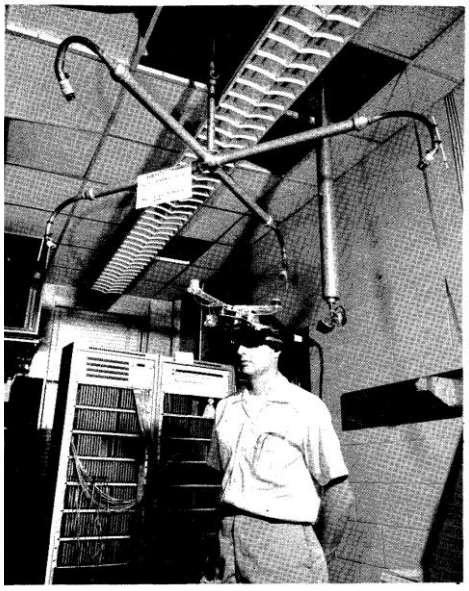


Figura 2.2: Utente che utilizza la 'Spada di Damocle' di Sutherland [2.2].

Qualche anno dopo, nel 1968, lo stesso autore realizza quello che passerà alla storia come il primo esperimento di Realtà Aumentata, chiamato 'Spada di Damocle' [7]. L'origine del nome è dovuta al fatto che la strumentazione utilizzata fosse così pesante da dover essere sostenuta dal soffitto, da cui veniva calata per essere indossata dall'utente, come si vede dalla Figura 2.2 [9].

La funzione di tale dispositivo *head-mounted*[7], il cui schema è presentato in Figura 2.3 [9], è quella di visualizzare oggetti generati tramite computer, e quindi virtuali, in un display tridimensionale, opportunamente presentati in base alla posizione dell'utente. La 'spada' è quella parte della strumentazione che viene collegata alla testa dell'utente e tiene traccia della sua posizione, inviando i dati a un computer che si occuperà di visualizzare il contenuto negli occhiali indossati dall'utente.

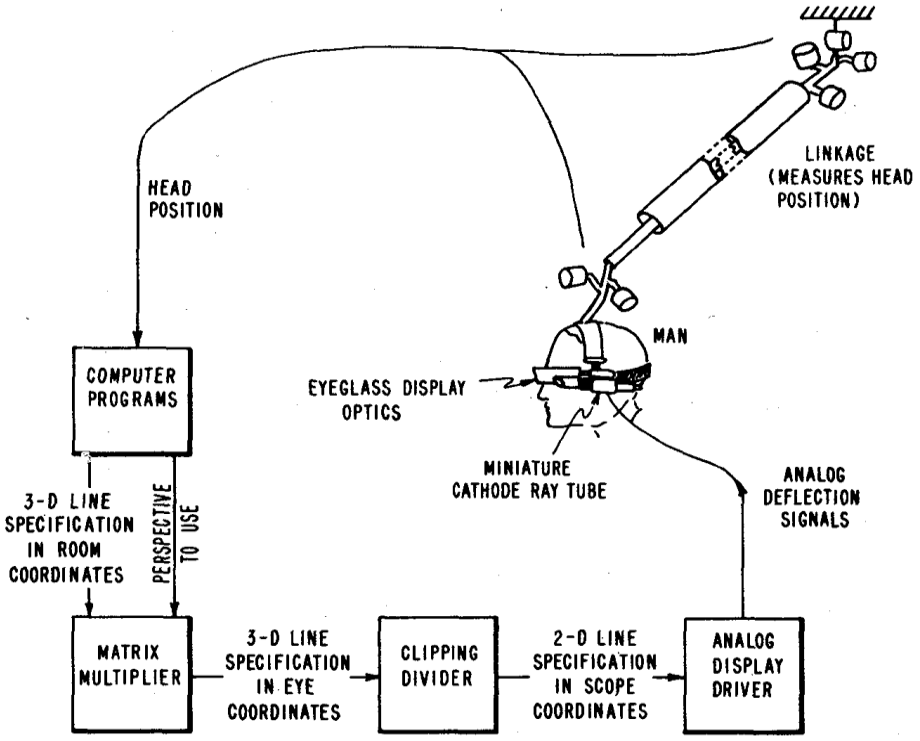


Figura 2.3: Schema della 'Spada di Damocle' [9].

[7] Dispositivo ideato per essere montato in testa dell'utente.

Decenni di progressi tecnologici hanno portato allo sviluppo non solo di computer, ma anche di smartphone e tablet delle prestazioni sempre più elevate e più accessibili all'utente medio, che al giorno d'oggi è sempre più esigente e dispositivi ingombranti come quello realizzato da Sutherland non desterebbero alcun interesse [13].

2.3 Realtà Aumentata

Se la *Virtual Reality* pone l'utente dentro un ambiente interamente generato da computer, la *Augmented Reality* ha lo scopo di mostrare contenuti virtuali che appaiono sovrapposti al mondo fisico. Essa va oltre la computazione, colmando il divario tra mondo virtuale e mondo reale. Con la AR l'informazione digitale diventa parte del mondo reale [2].

Si parla di Realtà Aumentata quando la realtà in cui viviamo, il nostro mondo fisico, viene arricchito in qualche modo da un qualcosa che al mondo fisico non appartiene, un contenuto fittizio, che va quindi ad 'aumentare' la realtà, aggiungendole qualcosa. Per poter accedere a questo contenuto e, successivamente, interagirvi è necessario, allo stato attuale della tecnologia, utilizzare un dispositivo, che può essere un normale smartphone dotato di fotocamera oppure un dispositivo più complesso, come ad esempio un visore.

Una delle più recenti definizioni di questa tecnologia è stata data da Nicolò Carpignoli in [1] che ha sostenuto, nel 2021, come la Realtà Aumentata sia un mezzo di comunicazione per visualizzare del contenuto fittizio situato nella realtà. Si intuiscono quindi i due soggetti principali che fanno parte di questa materia: il mondo reale e un contenuto non reale, fittizio appunto.

Come accennato nel paragrafo precedente, già nella seconda metà del secolo scorso, si possono apprezzare esperimenti di Realtà Aumentata, sebbene il termine non fosse stato ancora coniato. È solo nel 1992, infatti, che si parla per la prima volta di Realtà Aumentata, grazie a Tom Caudell, un ricercatore che a quel tempo lavorava per Boeing, compagnia nota per la produzione di velivoli [7].

Caudell, con l'aiuto di David Mizell, realizza un dispositivo *head-mounted*, alla stregua di quello realizzato da Sutherland nel 1968 (si veda il Paragrafo 2.2), ma perfezionandolo e rendendolo più 'portatile', grazie all'avanzamento tecnologico di cui poteva usufruire in quegli anni. In Figura 2.4 è mostrato lo schema del dispositivo che passerà alla storia come primo dispositivo pensato per la Realtà Aumentata, e si può già intuire il netto miglioramento in termini di praticità rispetto alla 'Spada di Damocle'.

L'idea rivoluzionaria dietro questo esperimento fu quella di considerare il fatto che, date le prestazioni dei microprocessori correnti, immergere l'utente in un mondo completamente virtuale sarebbe stato troppo costoso. Quello che più

conveniva fare era mostrare all'utente pochi contenuti virtuali, adattandoli in base alla sua posizione nel mondo reale. Il dispositivo, infatti, teneva traccia della posizione dell'utente (con uno strumento molto meno ingombrante di quello utilizzato nel 1968) e, tramite microprocessori, veniva visualizzato un contenuto virtuale direttamente nel display del visore indossato. Questo contenuto virtuale avrebbe guidato l'operatore nelle attività lavorative, facilitando il suo operato, favorendone l'efficienza e migliorandone in generale la qualità del lavoro [8].

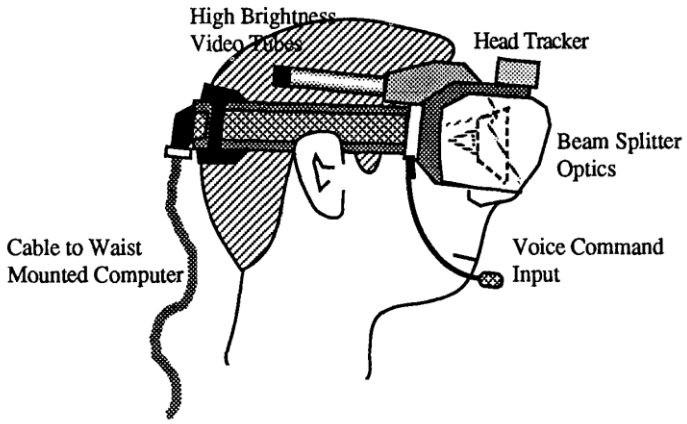


Figura 2.4: Primo dispositivo pensato per la Realtà Aumentata, creato da Caudell e Mizell [8].

Quello che voleva ottenere Caudell era favorire lo staccamento dell'utente dal pc o qualsiasi hardware statico per poter sfruttare un mix di realtà e virtualità, anticipando così quello che oggi chiamiamo *Mobile Augmented Reality*, ovvero Realtà Aumentata nei dispositivi mobile (si veda il Capitolo 4).

Da quel momento in avanti, le ricerche sulla Realtà Aumentata progredirono e si giunse a quella che oggi è la definizione maggiormente adottata nella comunità scientifica [2]. Nel 1997 infatti, Ronald Azuma, nel suo ' *A Survey of Augmented Reality* ', realizza il primo report sulla Realtà Aumentata (come si vede anche nella striscia temporale di Figura 1.1) fornendone una definizione basata su 3 proprietà, affermando, cioè, che un sistema di Realtà Aumentata deve [6]:

- combinare reale e virtuale,
- essere interattivo in tempo reale,
- essere registrato in 3D.

Mentre la prima caratteristica non aggiunge molto a ciò che si è visto in [1], lo stesso non si può dire per le altre due, che delineano in modo chiaro, evitando ogni tipo di fraintendimento, che cosa veramente è la Realtà Aumentata, fornendone un contesto di utilizzo, ovvero la sua interattività in tempo reale, e una sua visualizzazione, quella in 3D. Come lo stesso autore afferma, film come 'Jurassic Park' presentano contenuti virtuali mescolati egregiamente col mondo reale, ma mancando di interattività non possono essere classificati come sistemi di Realtà Aumentata [6]. Allo stesso modo, invece, i filtri usati per Snapchat e Instagram possono essere considerati a tutti gli effetti allineati con questa definizione [21].

2.4 Applicazioni di oggi

Sebbene questo elaborato si ponga come obiettivo quello di illustrare come la Realtà Aumentata possa affiancarsi efficacemente al settore dell'Industria 5.0, vale la pena presentare altri settori in cui questa tecnologia sta avendo positivi riscontri. Già con Caudell, nel 1992, si è visto come sfruttare la Realtà Aumentata nel settore manifatturiero, utilizzandola per guidare l'operatore nelle sue attività lavorative riguardanti la produzione di aeroplani.

Si è già parlato nel Capitolo 1 di come *PokèmonGo*, un'app che sfrutta la Realtà Aumentata per scopi ludici, abbia suscitato un forte interesse raggiungendo oltre mezzo miliardo di download. Si è già accennato a come i filtri utilizzati in *Instagram*, social network che ha raggiunto nel 2021 oltre 2 miliardi di utenti attivi [22], utilizzino un sistema di Realtà Aumentata, consolidando così il settore dell'intrattenimento come il più diffuso per questa tecnologia, sia per numero di utenti che per profitti di mercato.

Altre aree applicative di interesse sono quella medica e quella militare [6]. In campo medico, la tecnologia AR si sta facendo avanti in quanto le procedure adottate dalla chirurgia tendono ad essere sempre meno invasive. Il vantaggio principale è quello di poter visualizzare regioni di interesse quali tumori, vasi sanguini e nervi che sono spesso non visibili o oscurati da una vista diretta. In [23], ad esempio, viene proposto un sistema di Realtà Aumentata in grado di acquisire dati 3D tramite scansione a raggi-x e visualizzarli direttamente nello schermo di un *head-mounted* device indossato dal chirurgo, sovrapponendo l'immagine alla regione di interesse, guidando così l'operazione.

In campo militare la Realtà Aumentata è usata da anni come *training* per i piloti, i quali indossano un dispositivo *head-mounted* che permette loro di simulare il lancio di missili contro bersagli nemici, fornendo quindi una preparazione altrimenti impossibile [24]. Restando al combattimento terrestre, invece, si può considerare il sistema BARS, *Battlefield Augmented Reality System*, ideato per facilitare le operazioni militari terrestri. Mentre per i piloti aerei il contenuto visualizzato dal sistema di Realtà Aumentata è visualizzato direttamente tramite il dispositivo indossato, per i combattimenti terresti ci sono alcuni fattori da considerare: l'ambiente 3D è più complesso, dinamico e si perde spesso la linea di tiro [25]. L'idea del sistema BARS è quella di risolvere questi problemi, fornendo un database al dispositivo di Realtà Aumentata indossato dai soldati, che viene

aggiornato continuamente di soldati stessi, aggiornando quindi il contenuto visualizzato in base ai cambiamenti che si possono verificare per i problemi elencati sopra [25].

Capitolo 3
Industria 5.0

In questo capitolo verrà introdotta l'idea di Industria 5.0, un nuovo sistema di concepire il settore industriale, a stampo europeo, proposto per la prima volta nel 2020 [57]. Si confronterà questa rivoluzione industriale con le precedenti, concentrandosi soprattutto sull'Industria 4.0.

Si definiranno i capisaldi della nuova industria, descrivendo quali miglioramenti essa apporterà alla società, per poi concludere con un'analisi della *smart factory*, su cui si basa il progetto presentato per la realizzazione di questo elaborato.

3.1 Contesto storico

In Figura 3.1 [58] si vede una schematizzazione delle quattro rivoluzioni industriali avvenute fino ad oggi. Se da un lato la prima, avvenuta nel Regno Unito durante la seconda meta del '700, è stata caratterizzata dalla meccanizzazione attraverso l'acqua e la forza del vapore, e la seconda, diffusasi in Europa nella seconda metà dell'800 ha avuto come concetto centrale la produzione di massa, la terza rivoluzione industriale ha gettato le basi per i progressi che oggi ci portano a parlare dell'Industria 5.0. Essa, avvenuta a partire dalla metà del XX secolo, consiste infatti nell'ascesa dei computer e dell'automazione, che raggiungono l'apice della loro importanza con l'Industria 4.0, dove connessione e digitalizzazione sono i soggetti principali, necessari allo scopo di costruire fabbriche intelligenti.

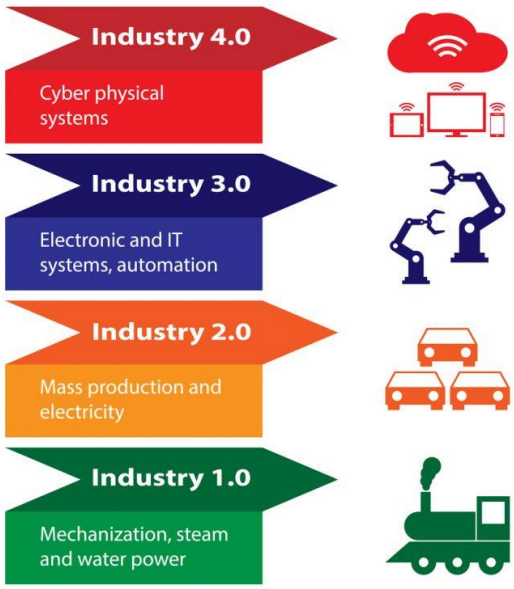


Figura 3.1: Le quattro rivoluzioni industriali con i loro soggetti principali [58].

Il termine 'Industria 4.0' è stato coniato per la prima volta in Germania, nel 2011 [58], come strategia che doveva essere adottata dalle imprese e dalla scienza, fortemente legata all'ambito *high-tech*. Questa rivoluzione è figlia degli avvenimenti che caratterizzarono l'inizio del XXI secolo, dove l'avanzamento

tecnologico ne è l'esponente principale. A testimonianza di ciò, in Figura 3.2 [58], sono mostrate le 10 società più grandi, in ordine di capitalizzazione totale, nel 2009 e nel 2019. Si può vedere come nella prima decade del nuovo secolo soltanto una società tecnologica fosse nella classifica, mentre successivamente questo settore si è fatto avanti, occupando la quasi totalità della classifica.

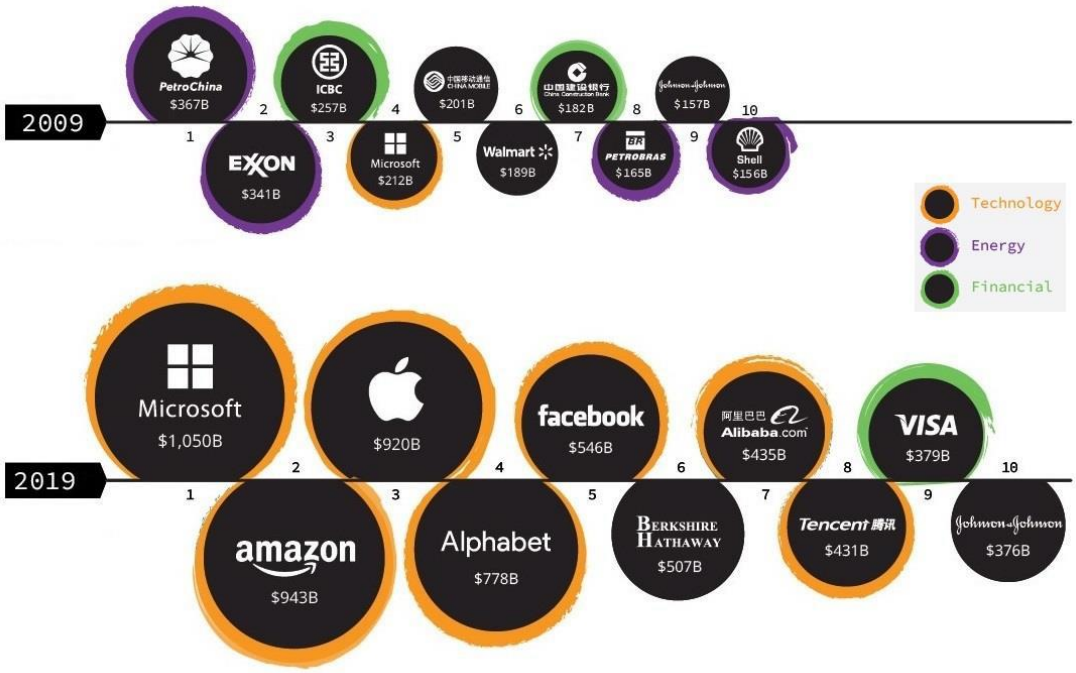


Figura 3.2: Capitalizzazione delle società nel 2009 (in alto) e nel 2019 (in basso) [58].

Il progresso tecnologico degli ultimi anni non sarebbe stato neanche immaginabile senza i miglioramenti in aree come l'automazione, la digitalizzazione e la connettività. Ruolo chiave dell'Industria 4.0 è, infatti, l'impatto che ha avuto l'*Internet of Things* (IoT) nell'organizzazione della produzione, la quale viene sconvolta da una nuova interazione tra uomo e macchina. Ed è proprio questa la parte che vuole colmare il quinto passaggio della rivoluzione industriale. Se il focus originario dell'Industria 4.0 comprendeva anche equità sociale e sostenibilità, negli ultimi anni la sua rotta ha seguito un percorso prevalentemente ed esclusivamente incentrato sulla digitalizzazione [58], e l'Industria 5.0 vuole

cambiare questa direzione, riavvalorando quei principi cardine persi dalla precedente rivoluzione industriale.

3.2 Capisaldi dell'Industria 5.0

La quinta rivoluzione industriale[8] è, perciò, un completamento dell'Industria 4.0, dalla quale prende il progresso tecnologico, donandone uno stampo più *human-centric* [58]. La recente crisi del Covid-19 ha evidenziato la necessità di reinventare i metodi e gli approcci lavorativi esistenti, necessità già presa in considerazione con l'Industria 4.0. Si vogliono perciò colmare queste vulnerabilità, ponendo un occhio di riguardo sia al benessere dell'operatore industriale, sia all'ambiente, cercando quindi di adottare una visione *Green* alla produzione.

Sono dunque questi i capisaldi dell'Industria 5.0, mostrati in Figura 3.3: approccio *human-centric*, sostenibilità e resilienza.

Figura 3.3: Industria 5.0 e suoi capisaldi [58].

[8] Il termine Industria 5.0 compare nella letteratura già dal 2019 [57], ma una definizione formale è stata fornita nel 2021 dalla Commisione Europea.

Approccio human-centric.
La differenza sostanziale tra le ultime due rivoluzioni industriale sta nelle domande che la società si pone a priori. Piuttosto che chiedersi cosa si può fare con la nuova tecnologia, ci si chiede cosa la nuova tecnologia può fare per noi. Piuttosto che prendere la tecnologia come punto di partenza, la nuova industria pone i bisogni degli esseri umani come cuore del processo di produzione. È la tecnologia emergente che si adatta ai lavoratori e non il contrario, per guidarli ed addestrarli nella loro attività lavorativa. Si pone fine a quel timore, nato con l'Industria 4.0, di un'inevitabile sostituzione dell'uomo con le macchine.

Sostenibilità.
La nuova industria deve rispettare i limiti planetari. Deve sviluppare processi circolari che riusino, reinventino e riciclino risorse naturali, riducendo così lo spreco e l'impatto ambientale. Sostenibilità significa ridurre il consumo di energia e delle emissioni di gas serra, per evitare l'esaurimento e la degradazione di risorse naturali, assicurando i bisogni della generazione di oggi senza mettere a rischio quelli della generazione del domani. Tecnologie come l'intelligenza artificiale e la produzione additiva[9] possono svolgere un ruolo importante in questo caso, ottimizzando l'efficienza delle risorse e riducendo al minimo gli sprechi.

Resilienza.
Per far fronte a crisi come la pandemia di Covid-19 degli ultimi anni, si deve sviluppare un più alto livello di robustezza nella produzione industriale, preparandola al meglio contro le interruzioni e i malfunzionamenti. I cambiamenti geopolitici e le crisi naturali hanno evidenziato la fragilità della corrente produzione globale. Le catene di produzione adesso dovranno essere sviluppate con strategie resilienti, che provvedano alla capacità di adattare la produzione alla flessibilità dei processi industriali, specialmente in quei settori che forniscono i beni di prima necessità dell'essere umano, come l'assistenza sanitaria o la sicurezza.

[9] Processo industriale che parte da modelli 3D virtuali per fabbricare oggetti reali, aggiungendo uno strato dopo l'altro, diversa dalla classica produzione sottrattiva, che parte da un materiale e rimuove strati da esso per realizzare l'oggetto finale.

L'Industria 5.0 riconosce il potere della realtà industriale di raggiungere obiettivi sociali, al di là dell'occupazione e della crescita professionale, per diventare un fornitore resiliente di prosperità, facendo in modo che la produzione rispetti i confini del nostro pianeta e ponendo il benessere del lavoratore dell'industria al centro del processo produttivo [58].

In questo contesto si dispone questo elaborato di tesi, considerando la Realtà Aumentata come una di quelle tecnologie in grado di esaltare l'aspetto *human-centric* della nuova industria, guidando l'operatore nella sua attività lavorativa, cercando di facilitare, tramite una Web-app, quelle che sono le sue mansioni all'interno del processo di produzione.

3.3 *Internet of Things*

Il termine *Internet of Things*, che letteralmente si traduce come Internet delle Cose, venne coniato nel 1999 da Kevin Ashton [81], riferendosi a quegli oggetti unicamente identificabili (*things*) e alla loro rappresentazione virtuale in una struttura simile a Internet. Esistono diverse definizioni di questo concetto, ma per capire cosa veramente significhi l'IoT si deve partire da cos'è, ovvero, un nuovo paradigma di comunicazione: "un mondo dove le cose possono automaticamente comunicare con computer e tra loro, fornendo servizi a beneficio dell'essere umano" [82].

Le applicazioni dell'*Internet of* Things sono molteplici, basti considerare che al giorno d'oggi esistono più oggetti connessi a Internet che persone [83]. In Figura 3.3 vengono mostrati i vari settori di utilizzo in cui questo nuovo approccio si è diffuso.

L' *Internet of Things* è composto dagli *smart objects*, oggetti intelligenti in grado di percepire e interpretare l'ambiente circostante e di interagire con altri oggetti o scambiare informazioni con persone. Questi oggetti sono equipaggiati da: sensori o attuatori[10], un piccolo microprocessore, un dispositivo per la comunicazione e una batteria. I sensori, o attuatori, conferiscono all'oggetto l'abilità di interagire con il mondo fisico, prendendo informazioni da esso o traducendo queste in operazioni sul mondo fisico. Il microprocessore è la componente che rende l'oggetto capace di interpretare il mondo fisico in maniera computazionale, traducendo quindi i dati catturati dai sensori, seppur con prestazioni limitate in quanto lo *smart object* non è pensato per ricevere dati complessi. Il dispositivo di comunicazione permette all'oggetto sia di inviare i dati raccolti tramite sensore sia ricevere informazioni da altri *smart object*. La batteria, infine, è la fonte di energia che permette all'oggetto di svolgere il suo lavoro. Ogni oggetto deve essere unico ed avere un indirizzo in Internet.

[10] I sensori sono responsabili di tradurre informazioni provenienti dal mondo fisico in segnali elettrici, gli attuatori svolgono il lavoro opposto, traducendo il segnale elettronico in lavoro meccanico.

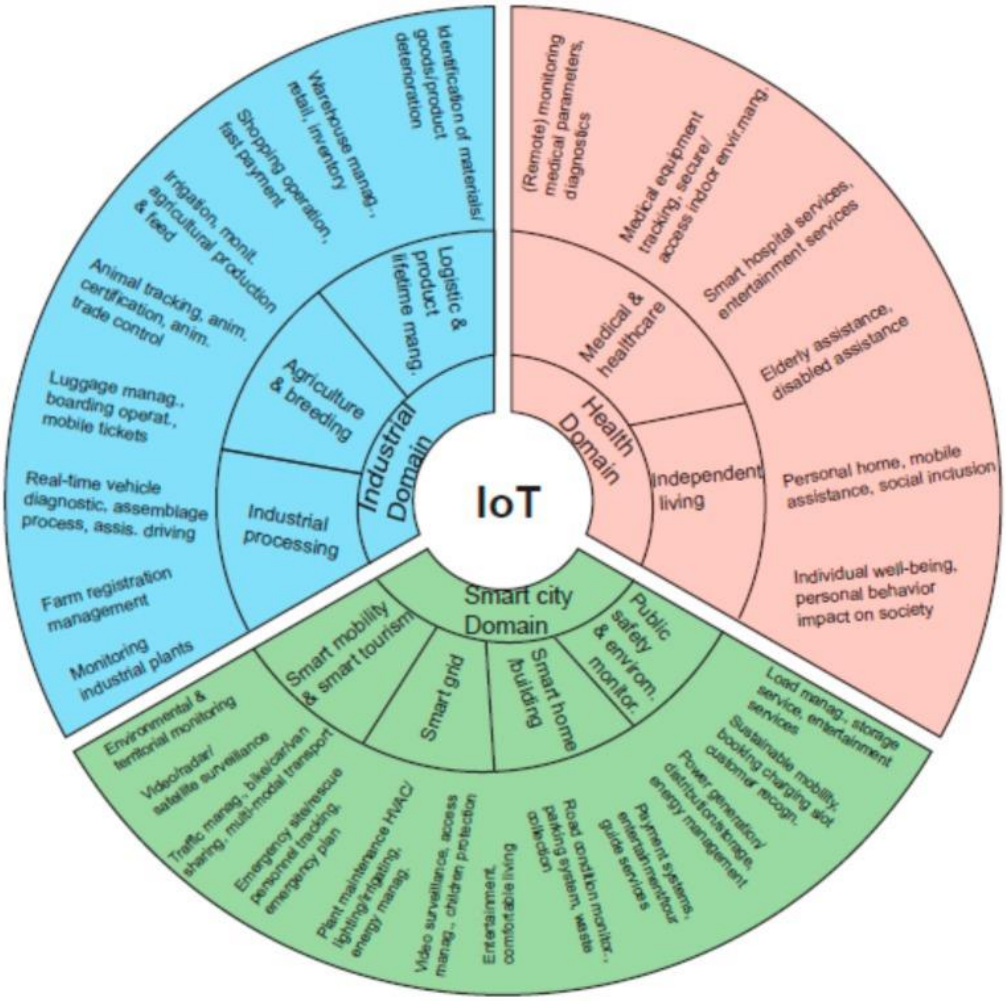


Figura 3.3: Settori dove l'IoT è utilizzato.

Gli oggetti fisici che possono far parte di questa architettura sono svariati, dai più familiari utilizzati in casa, quali ad esempio le lampadine, alle risorse in ambito sanitario, come dispositivi medici indossabili [69].

Nel progetto presentato in questo elaborato, si simulano dei sensori IoT che forniscono informazioni relative allo stato di un certo macchinario industriale e l'utente, tramite l'utilizzo di una Web-app, è in grado di ricevere queste informazioni direttamente nel suo dispositivo smartphone o tablet.

3.4 *Smart factory*

La nuova industria diventa quindi intelligente, prendendo anche il nome di *smart factory* per mettere in evidenza la combinazione tra il mondo fisico della produzione e quello digitale [79]. In Figura 3.4 viene mostrato uno schema dettagliato di come è strutturata una smart factory, dal quale si intuisce che l'architettura di questa nuova realtà è composta di quattro livelli.

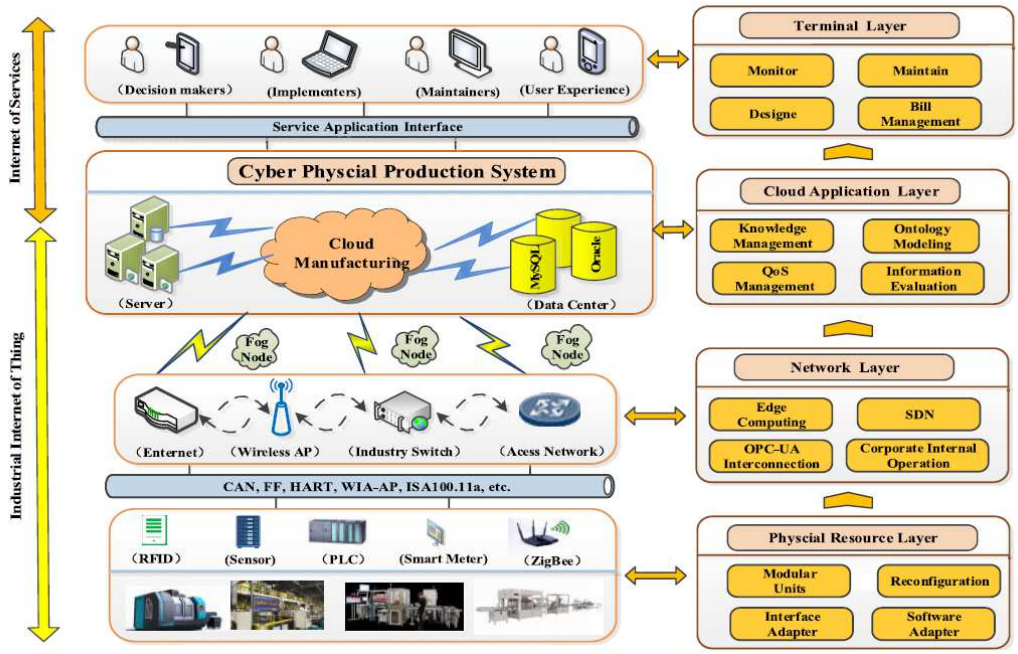


Figura 3.4: Schema gerarchico della smart factory [79].

Physical Resource Layer.
Il livello fisico, dove è presente l'attività produttiva, con i suoi macchinari e materiali. Se prima questi oggetti svolgevano solamente la loro funzione di produzione, ora essi vengono resi intelligenti, *smart*, avendo la capacità di comunicare tra loro attraverso la rete industriale. Tutto l'equipaggio fisico della nuova industria necessita di avere supporto per l'acquisizione di informazioni in tempo reale, così da poter costruire le fondamenta per la struttura *Internet of Things* della nuova realtà aziendale. I dispositivi devono essere in grado di

adattarsi e riconfigurarsi automaticamente in base alle informazioni che ricevono, le quali devono essere trasmesse ad alta velocità.

Industrial Network Layer.
Il livello della rete industriale realizza un'importante infrastruttura che non solo permette la comunicazione tra i componenti dell'attività produttiva, ma connette anche il livello fisico con il *cloud* aziendale. Viene introdotto il concetto di *Industrial Wireless Sensor Networks*, i sensori industriali di rete, che provvedono a una maggior efficienza rispetto alla classica rete Ethernet [80], che non riesce ad adattarsi a pieno alla volatilità della nuova industria. Inoltre, nuove tecnologie, come ad esempio il *Open Platform Communications Unfied Architecture* (OPC-UA), che consente la comunicazione tra i macchinari di informazioni contenute nei macchinari stessi, o i *Software defined Networks* (SDNs), vengono introdotte per garantire una miglior qualità di servizio della rete in termini di affidabilità.

Cloud Layer.
Le informazioni prodotte dagli *smart object* della nuova infrastruttura industriale possono essere di enormi quantità e della loro gestione se ne occupa il livello cloud. In questo livello i dati, provenienti dai sensori industriali di rete, vengono salvati in database e analizzati attraverso la tecnologia del *cloud computing*. Si inizia qui a parlare di *Internet of Services*: lo spazio di archivio viene ridimensionato e i dati analizzati a seconda delle necessità, per poi essere mostrati al livello finale, dove è presente l'operatore.

Terminal Layer.
Anche chiamato livello di supervisione e controllo, questo livello collega le persone alla smart factory. Attraverso terminal, che possono essere PC, tablet o smartphone, gli utenti possono accedere ai dati forniti dal cloud per applicare una diversa configurazione ai macchinari, occuparsi di attività di manutenzione e diagnosi o visualizzare la statistica dei dati.

Capitolo 4
Sistemi di Realtà Aumentata nei dispositivi Mobile

In questo capitolo verrà affrontato l'argomento della Realtà Aumentata nei dispositivi mobile (dall'inglese *Mobile Augmented Reality*, MAR). Una tecnologia relativamente nuova, ma che sta attraendo l'interesse di ricercatori sia dal mondo accademico che da quello industriale [27].

Si farà luce su come questi device portatili riescano a fornire esperienze di Realtà Aumentata in maniera efficace e del perché stiano giocando un ruolo chiave sullo sviluppo di questa tecnologia. Si parlerà di ARToolKit, libreria gratuita scritta in liguaggio C, largamente utilizzata nei sistemi di Realtà Aumentata, specialmente per quelli implementati nei dispositivi mobile e storicamente importante (come si può vedere, è citata anche in Figura 1.1 del Capitolo 1).

Verrà poi trattato il tema della Realtà Aumentata nel Web (dall'inglese *Web Augmented Reality*, Web AR) considerandone gli aspetti positivi e negativi. L'interesse nelle Web-app è in aumento, basti pensare che già nel 2014, per la prima volta, il numero globale di utenti che accedono al Web mediante dispositivi mobile sorpassa quello di coloro che vi accedono tramite un computer desktop [26].

Nasce quindi l'idea di voler proporre in questo elaborato, come si vedrà nel prossimo capitolo, una Web-app, pensata per dispositivi mobile (ma funzionante anche su computer desktop), per mostrare la potenzialità della Realtà Aumentata in questi apparecchi.

4.1 Realtà Aumentata nei dispositivi Mobile

I dispositivi mobile, specialmente smartphone e tablet, hanno visto un ampio avanzamento tecnologico durante l'ultimo ventennio. Nati come un mero mezzo di comunicazione, i telefoni cellulare sono ormai diventati un 'hub' per l'intrattenimento, la fotografia, la navigazione, l'uso di internet e in particolare dei social media, per nominarne alcuni [5]. L'utente di oggi si trova nelle mani un computer a tutti gli effetti, dotato di microprocessori e capace di elaborare audio e video in maniera più che efficace. Oltre che strumenti come fotocamera, necessaria per fare da ponte tra mondo reale e mondo virtuale, sensori di ultima generazione quali localizzatore GPS, bussola, accelerometro di movimento [16], rendono lo smartphone di oggi uno strumento ideale per avere consapevolezza del mondo fisico circostante, nonostante le sue limitate prestazioni computazionali.

Già con Caudell [8] si presupponeva la necessità di staccarsi dal pc desktop fisso per poter interagire con il mondo fisico che ci circonda, ma la soluzione proposta, quella di indossare un *headset-mounted-device*, seppur innovativa a quel tempo, oggi apparirebbe ingombrante e scomoda, nonché costosa. Una tecnologia come la Realtà Aumentata, che prevede l'elaborazione di pochi dati virtuali (quindi non necessita di processori con prestazioni particolarmente elevate), può meglio essere utilizzata con *handled-device* [15], e gli smartphone sono l'equilibrio perfetto tra costi e comodità, in quanto hanno prestazioni adatte alla causa, con prezzi accessibili e fanno ormai parte della nostra vita quotidiana

A prova di questo si può guardare la Figura 4.1, dove si nota un notevole aumento di ricerche online con la parola chiave 'augmented reality' a partire dal 2009. Fu proprio in quell'anno che vennero installati i primi sensori di localizzazione e bussola nei telefoni cellulari e questo riaccese l'interesse per la Realtà Aumentata [17].

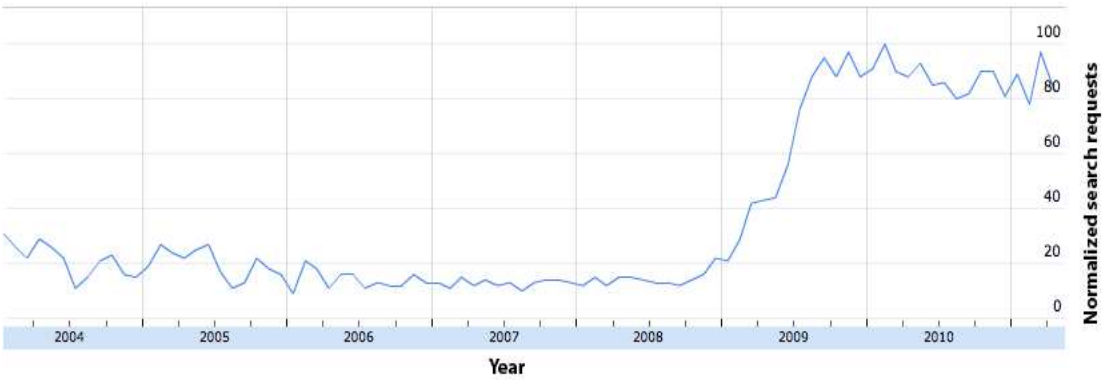


Figura 4.1: Ricerche Google con parola chiave 'augmented reality' tra gli anni 2004-2011. L'asse delle ordinate mostra il numero di ricerche normalizzato nell'intervallo da 0 a 100 [17].

I sistemi di *Mobile Augmented Reality* sono sistemi che rispecchiano a pieno la definizione di Realtà Aumentata fornita da Azuma (si veda il Capitolo 2), possedendo le caratteristiche di fusione tra realtà e virtualità, visualizzazione e interazione in tempo reale. In particolare, un sistema mobile di Realtà Aumentata, per essere definito tale, deve svolgere le seguenti funzioni chiave [27]:

- *Tracking and Registration*
- *Object Detection and Recognition*
- *Calibration*
- *Model Rendering*

4.1.1 *Tracking and Registration*

Il tracciamento della posizione e dell'orientazione (*tracking*) dell'osservatore e la registrazione del contenuto virtuale (*registration*) sovrapposto alla scena reale, sono fasi essenziali per ogni sistema di Realtà Aumentata, a maggior ragione per i sistemi MAR, dove l'utente cambia spesso posizione e orientazione. Ogni volta che l'osservatore si sposta, infatti, il sistema deve ricalcolare la sua posizione per allineare il contenuto virtuale alla scena reale, ottenendo come risultato quello di far sembrare all'osservatore che il contenuto sia parte del mondo fisico. Questo si ottiene con il processo di tracciamento [43]. Il processo di registrazione, invece, si

occupa della sovrapposizione dell'oggetto virtuale alla scena reale, che si allinea alla posizione ottenuta dal tracciamento [30].

Entrambi i due processi vengono svolti dai sistemi mobile di Realtà Aumentata tramite la fotocamera installata nel dispositivo, utilizzando tecniche che verranno illustrate più avanti.

4.1.2 *Object Detection and Recognition*

Letteralmente, rilevamento e riconoscimento di oggetti. Questo processo è la fase cruciale di ogni sistema MAR, non esistendo ancora uno standard su come realizzarlo è ancora in fase di sperimentazione [29], ed alcune tecniche verranno illustrate in seguito.

Lo scopo di tale processo è riconoscere il *target* nella scena a cui poi sovrapporre il contenuto virtuale ed è diviso in due step. Il primo, *object detection*, svolge la funzione di rilevamento e classificazione [44] ed è in questa fase che vengono riconosciute forme e caratteristiche principali degli oggetti nella scena. Come si vedrà nei successivi paragrafi, il riconoscimento oggetti è quella tecnologia per la quale il sistema rileva i *markers* presenti nella scena. Il secondo, *Object Recognition*, svolge la funzione di *image matching*, ovvero immagazzina le caratteristiche dell'immagine rilevata e delle sue informazioni aggiuntive in un database esterno. In questo modo si ottiene un tracciamento migliore, riconoscendo la scena corrente tramite corrispondenza di immagini [27].

Nei dispositivi mobile l'*Object Detection* è svolto dalla fotocamera, che cattura le immagini presenti nella scena corrente, le quali vengono poi processate tramite l'*Object Recognition*.

4.1.3 *Calibration*

La calibrazione (*calibration*) è quel processo che completa il tracciamento della posizione e orientazione, aggiungendo informazioni relative alle coordinate dello strumento utilizzato per visualizzare la scena; nel caso dei sistemi di Realtà Aumentata mobile è la fotocamera del dispositivo.

È in questa fase che avviene il 'calcolo della posa' (*pose estimation*), ovvero il calcolo, di cui si discuterà in seguito, delle coordinate dell'oggetto virtuale in

relazione alla posizione della fotocamera, utilizzando le informazioni ottenute dal processo di *tracking* [40].

4.1.4 *Model Rendering*

Il *Model Rendering* è il processo finale per un sistema di Realtà Aumentata, si occupa della visualizzazione del contenuto virtuale 3D tenendo in considerazione i dati raccolti dalle fasi precedenti ed utilizzando librerie esterne pensate per la grafica 3D, in modo che l'oggetto virtuale appaia più fotorealistico possibile [45]. A tal proposito, nei i sistemi in Realtà Aumentata per dispositivi mobile, è largamente usata la libreria grafica *OpenGL ES*.

Questa API[11], cross-platform, prodotta da 'Khronos Group', è stata pensata come standard per oggetti 3D utilizzati in sistemi mobile (come ad esempio smartphone, tablet e console) e tutti quei sistemi in generale che hanno prestazioni più scarse rispetto a un computer desktop [28]. OpenGL ES lavora sfruttando la GPU (graphics processing unit) del dispositivo, salvando i dati ottenuti in memoria per poi utilizzarli nel *task* da svolgere attraverso la CPU (central processing unit), come schematizzato in Figura 4.2 [27].

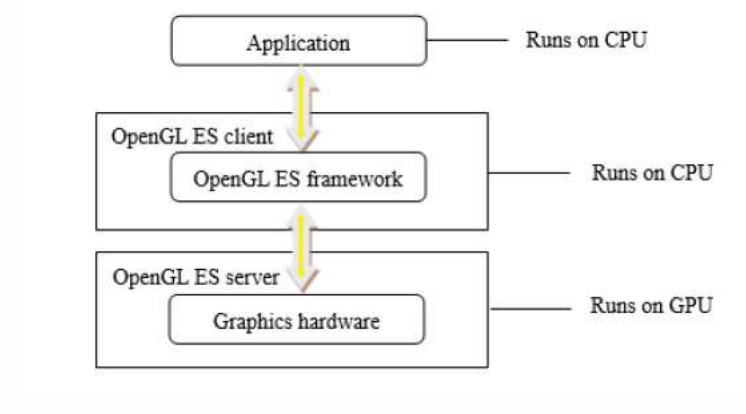


Figura 4.2: Schema di funzionamento dello standard OpenGL ES [27].

[11] *Application Programming Interface*, ovvero interfaccia di programmazione delle applicazioni, un insieme di definizioni e protocolli per la creazione e integrazione di software applicativi [73].

4.1.5 WebGL

La compagnia citata nel paragrafo precedente, 'Khronos Group', è responsabile anche di aver creato *WebGL*, un'API JavaScript, cross-platform e senza licenza, per il rendering di grafica 2D e 3D su Web browser [74]. Esso si basa OpenGL ES, visto in precedenza, ed è scritto in un linguaggio chiamato OpenGL ES shading (GLSL ES), un linguaggio simile a C e C++, che consente di implementare elementi in 2D e 3D senza richiedere alcun plug-in ed eseguirli direttamente nella GPU del computer.

WebGL è una DOM (Document Object Model) API, questo significa che può essere usato con ogni altro linguaggio DOM-compatibile, su tutti il linguaggio JavaScript. Il DOM è un API per documenti HTML e XML che ne definisce sia la struttura logica sia il modo in cui si può accedere e modificare il documento stesso. Il principale linguaggio di programmazione usato nel DOM è proprio JavaScript, presente sulla maggior parte delle pagine Web. JavaScript, infatti, consente di modificare in modo dinamico il DOM, ad esempio animando, spostando e nascondendo determinati elementi HTML e, come si vedrà, è stato utilizzato per la realizzazione del progetto di questa tesi.

WebGL è ormai supportato dai maggiori venditori di browser, su tutti Google (con Chrome), Apple (con Safari) e Mozilla (con Firefox).

La natura di basso livello di WebGL, che da solo contribuisce poco alla realizzazione rapida di grafica 3D, ha contribuito alla creazione di librerie che sono tipicamente usate per costruire esperienze in 3D. Anche semplici attività di base, come il caricamento di scene grafiche e di oggetti 3D nei formati più comuni, non sono direttamente previste. Per questo sono state create librerie JavaScript, anche da altri linguaggi, così da aggiungere tali funzioni. Tra le varie librerie si può citare A-Frame, ovvero il framework utilizzato nella realizzazione del progetto di questo elaborato.

4.2 ARToolKit

Come visto nel paragrafo precedente, i sistemi di Realtà Aumentata, sia mobile che non, per riuscire nel loro obiettivo, devono partire dal processo di *tracking* e *registration*. Questo processo, nei dispositivi mobile, può essere realizzato con due metodi differenti: il metodo *sensor-based* e quello *vision-based* [29]. Il primo si basa sull'utilizzo di sensori installati nel dispositivo, impiegando campi elettromagnetici o onde radio per determinare il calcolo della posa. Di maggior interesse è però il secondo metodo, che è quello utilizzato in questo elaborato, dove le informazioni ottenute provengono dalle immagini catturate dalla fotocamera. Questo approccio può offrire due diverse esperienze di Realtà Aumentata, le quali differiscono a seconda dell'immagine catturata dalla fotocamera: *marker-based* e *natural-feature-tracking*.

ARToolKit, sviluppata da Kato e Billinghurst nel 1999, è la prima libreria *open-source*, scritta in linguaggio C, che consente agli sviluppatori di creare esperienze in Realtà Aumentata, all'inizio soltanto tramite i cosiddetti *fiducial markers* [34], ma poi è stata migliorata anche per riconoscere immagini 2D in generale (*ARToolKit NFT*). Questa libreria segna una tappa fondamentale per la storia della Realtà Aumentata (è citata anche in Figura 1.1), grazie alla quale sono state create numerose applicazioni [34].

Come si vedrà, *AR.js*, la libreria *open-source* utilizzata per realizzare il progetto del presente elaborato, si basa proprio su ARToolKit, precisamente sulla versione Plus, sviluppata nel 2004 da Wagner e Schmalsteig e pensata proprio per i dispositivi mobile [47].

Per precisazione, ARToolKit, è stato distribuito come progetto open-source nel 2001 e rimase tale fino alla sua vendita, nel 2015 [72]. Nel 2017 i precedenti CEO e CTO di ARToolworks, l'organizzazione che supportava il progetto, Ben Vaughan e Phil Lamb, crearono *artoolkitX*. Questo nuovo software open-source si sostituisce, quindi, al vecchio ARToolKit, garantendone il funzionamento e il mantenimento.

4.3 Sistemi *Marker-based*

I marker (dove un esempio è mostrato in Figura 4.3), anche chiamati *fiducial markers* [34], sono oggetti o immagini reali di forma quadrata, che si trovano nel mondo fisico e che vengono riconosciuti dal sistema di Realtà Aumentata tramite la videocamera del dispositivo. Il riconoscimento di tali marker servirà poi per il calcolo della posa, descritto in precedenza, dell'oggetto virtuale da visualizzare [35].

Figura 4.3: Esempio di un marker [32].

Il sistema di Realtà Aumentata, per riconoscere il marker mediante la videocamera, utilizza moderne tecniche di *computer vision*, convertendo prima l'immagine in una scala di grigi, per accelerare l'algoritmo di processamento immagini [37], e riconoscendo poi i bordi della sagoma quadrata (si veda Figura 4.5). Avere forma quadrata è infatti mandatorio per un marker, in quanto il sistema, per effettuare il riconoscimento, scarta tutte quelle forme curve o convesse presenti nella scena. L'immagine viene prima catturata dalla fotocamera, poi, attraverso un rilevatore di linee e angoli, vengono riconosciuti i quattro lati del quadrato. Queste informazioni serviranno poi per il calcolo della posa [36]. Lo schema di questo procedimento è illustrato in Figura 4.4.

Riconoscere un certo marker piuttosto che un altro è possibile mediante due tecniche: *template matching* e riconoscimento tramite algoritmi di decodifica [38] [39]. Per la prima tecnica è necessario precaricare nel sistema il contenuto del

marker, ovvero il suo *pattern* identificativo, che verrà poi riconosciuto per corrispondenza immagini. Questo è il metodo adoperato da ARToolKit [42]. La seconda tecnica è più complicata dal punto di vista computazionale, ma consente di avere maggiore velocità nelle prestazioni ed evita il precaricamento del marker nel sistema. Il contenuto del marker viene codificato, mediante opportuni algoritmi, poi il sistema ne effettua la decodifica e riconosce il marker. Entrambe le tecniche consentono comunque di poter lavorare con più marker contemporaneamente, ottenendo quindi un sistema *multi-markers*.

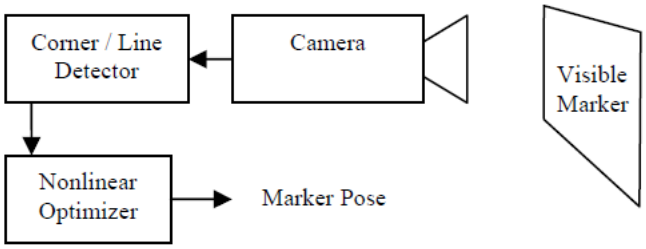


Figura 4.4: Schema per l'acquisizione dell'immagine e calcolo della posa [36].

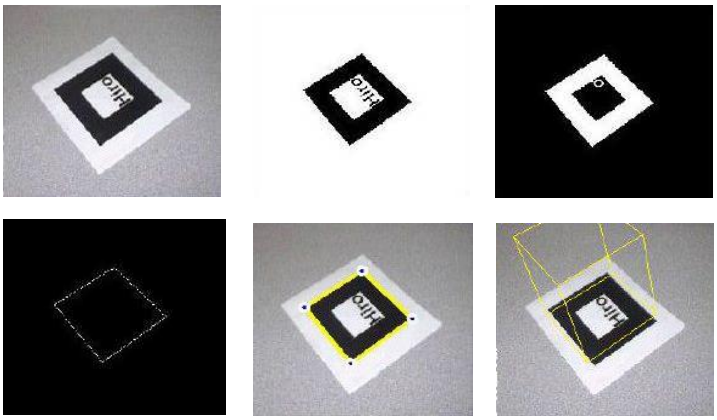


Figura 4.5: Le diverse fasi del processo di elaborazione visuale di ARToolKit: Acquisizione, Ricerca quadrati, Estrazione Contorni, Calcolo della Posizione,Rendering dell'oggetto virtuale [51].

In Figura 4.6 è mostrato il processo di tracciamento e registrazione utilizzato da ARToolKit. Il sistema, in questo caso, acquisisce l'immagine dalla videocamera e, applicando una soglia binaria e un adattamento parziale delle linee [48], riesce a riconoscere i bordi delle figure, considerando come potenziali markers tutte quelle regioni racchiuse entro quattro linee di contorno. Da qui si procede al calcolo della matrice di trasformazione a sei gradi di libertà, considerando le tre coordinate spaziali del marker in relazione alle tre coordinate della videocamera. Nota questa matrice, il marker viene rettificato, così da farlo apparire parallelo alla videocamera ed avviene il riconoscimento. Il sistema, infatti, confronta il *pattern* individuato con quelli già precaricati nel suo database. Se c'è corrispondenza il sistema ricava l'orientazione del marker rispetto al suo asse verticale e a questo punto avviene la registrazione dell'oggetto virtuale.

Uno schema dettagliato di come avviene la binarizzazione e l'applicazione di una soglia binaria è mostrato in Figura 4.6 [49].

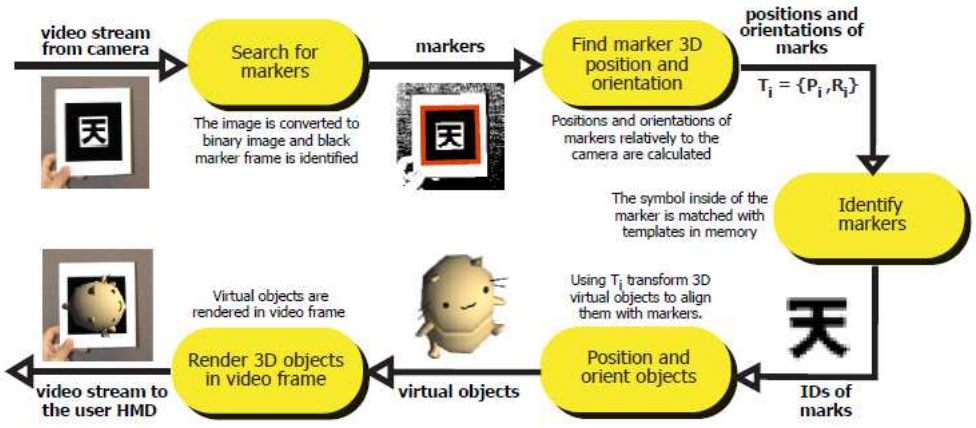


Figura 4.6: processo di tracking e registration utilizzato da ARToolKit [48].

Per questo progetto sono stati realizzati *pattern* tramite un generatore apposito fornito dallo stesso creatore di *AR.js*, Jeromme Etienne [50]. In Figura 4.8 è mostrato il risultato ottenuto, prendendo uno dei marker utilizzati in un caso di studio, caricandolo nel generatore e ottenendo il file .patt corrispondente. Come si vede questo file è una rappresentazione della forma e del contenuto del marker, dove ogni pixel viene codificato da una matrice [16x16] con un valore da 0 a 255. Questo file .patt è il file che viene precaricato nel sistema e che permetterà di

riconoscere il marker per corrispondenza immagini, dopo averlo rettificato dall'immagine catturata.

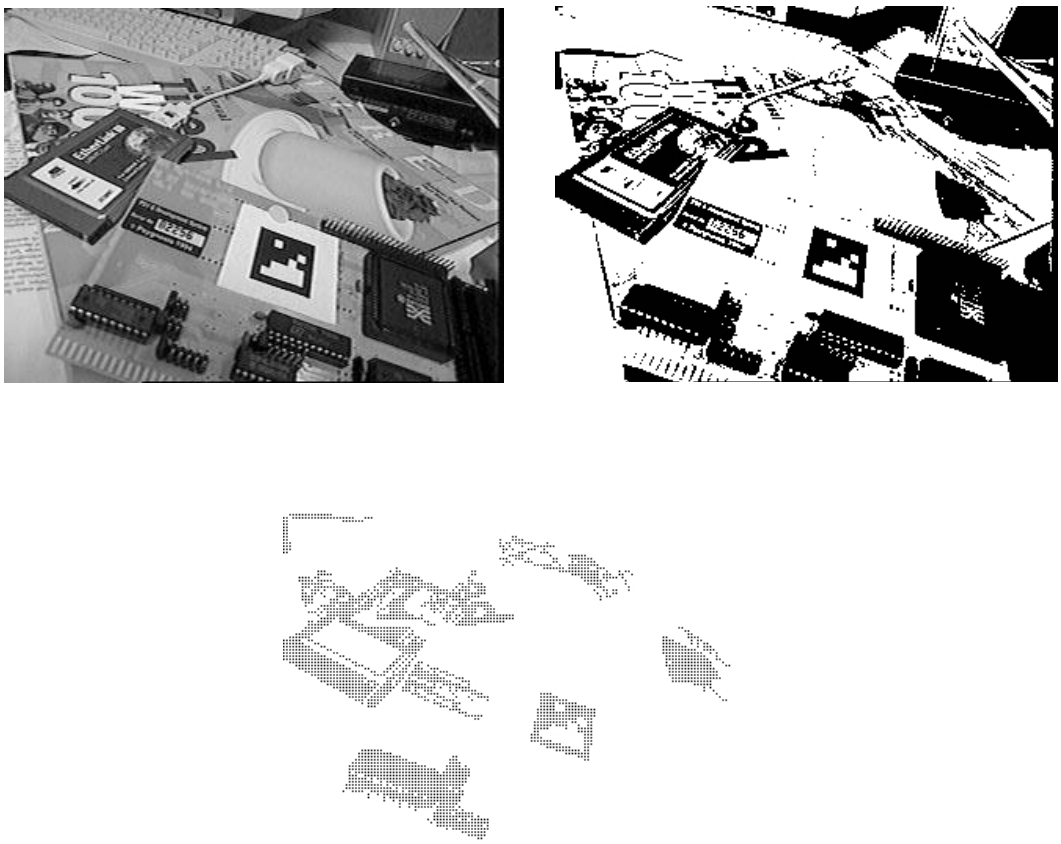


Figura 4.7: Riconoscimento delle forme per un sistema di Realtà Aumentata mobile: immagine originale (in alto a sinistra), binarizzazione (in alto a destra) e binarizzazione dopo l'applicazione della soglia (in basso) [49].

255	254	255	255	255	254	255	255	254	254	254	255	254	254	254	254
254	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	0	13	13	13	13	13	13	0	255	255	255	255
255	255	255	255	0	255	255	255	255	255	255	0	255	255	255	255
254	253	0	0	0	0	0	0	0	0	0	0	0	0	254	254
254	254	0	0	0	0	0	0	0	0	0	0	0	33	254	254
255	254	0	0	0	0	0	0	0	0	0	0	0	0	255	255
254	254	0	0	0	0	0	0	0	0	0	0	0	0	254	254
254	254	0	0	0	255	255	255	254	254	254	0	0	0	254	254
254	254	0	0	0	255	255	255	255	254	255	0	0	0	255	255
254	254	255	255	0	255	0	0	0	0	254	0	253	254	254	254
254	254	255	255	0	15	15	15	15	15	15	0	255	254	255	254
254	254	255	254	255	255	255	255	254	254	254	255	254	254	254	254
254	254	255	255	255	255	255	255	254	254	254	255	254	254	254	254
254	254	254	255	255	255	255	255	254	254	254	254	254	246	250	245

Figura 4.8: Immagine del marker a sinistra e rappresentazione del pattern del marker a destra.

I marker sono utili in quelle situazioni dove il riconoscimento oggetti non è applicabile per mancanza di dati sufficienti, o dove serve una determinazione della posa con alta affidabilità, cosa che difficilmente avviene con il *natural-feature-tracking*. Ad ogni modo, la struttura dei marker deve essere in grado di mantenersi invariata nel tempo, condizione necessaria per un'esperienza di Realtà Aumentata *marker-based* [38].

Nonostante la popolarità raggiunta, ARToolKit ha alcune carenze. L'adattamento parziale di linee che viene utilizzato per riconoscere le forme dei soggetti in una scena reale, è molto suscettibile alle occlusioni [34]. A causa di questa limitazione, basta poco per far sì che il marker non venga più riconosciuto dal sistema.

Un altro problema di questo approccio è percepibile nelle esperienze *multi-markers* accennate in precedenza. Dato che ARToolKit utilizza il metodo di *template matching* per riconoscere i marker presenti nella scena, questo provoca rallentamenti a run-time. Considerando N marker visibili dalla fotocamera e M marker noti, precaricati nel sistema, sono richieste 4xNxM operazioni di corrispondenza immagini e questo rallenta le prestazioni [47].

4.3.1 Stima della posa

Una volta identificato e riconosciuto un marker, si procede alla stima della posa, essenziale per sovrapporre correttamente un oggetto virtuale e far sembrare che esso sia effettivamente parte della scena reale, caratterista peculiare di ogni sistema di Realtà Aumentata.

In Figura 4.8 è mostrato lo schema disegnato da Billinghurst e Kato in [40] per proporre il loro sistema di calcolo della matrice di trasformazione e conseguente stima della posa, che verrà utilizzato da ARToolKit.

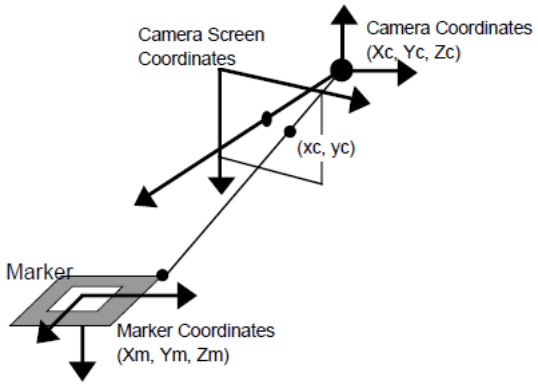


Figura 4.9: Relazione tra coordinate del marker e della fotocamera.

Riconosciuto e identificato il marker con i metodi illustrati sopra, le quattro linee di contorno della superficie quadrata e i suoi quattro vertici sono utilizzati come parametri per calcolare la matrice di trasformazione $\boldsymbol{T_{cm}}$. Per identificare il marker il sistema esegue una trasformazione prospettica della regione individuata intorno le quattro linee di contorno ed esegue una corrispondenza immagini con ciò che vede. Questo processo di normalizzazione, che, come risultato, permette alla fotocamera di osservare il marker come se fosse parallelo ad essa, si effettua tramite l'eq. 1, dove x_c e $\boldsymbol{y_c}$ sono le coordinate dello schermo con il quale l'utente vede la scena e X_m,Y_m sono le coordinate ricavate dai quattro vertici del marker rilevato [40].

$$\begin{bmatrix} hx_c \\ hy_c \\ h \end{bmatrix} = \begin{bmatrix} N_{11} & N_{12} & N_{13} \\ N_{21} & N_{22} & N_{23} \\ N_{31} & N_{32} & N_{33} \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ 1 \end{bmatrix} \quad \text{(eq. 1)}$$

Proiettando due lati paralleli di un marker quadrato sull'immagine, l'equazione di ognuno si presenta, considerando le coordinate dello schermo menzionate prima, come [42]:

$$a_1x + b_1y + c_1 = 0 \qquad a_2x + b_2y + c_2 = 0 \quad \text{(eq. 2)}$$

Come accennato, questi parametri sono ottenuti dal processo di riempimento linea per ogni marker. Data $\mathbf{P}$ matrice di proiezione prospettica, ottenuta dalla calibrazione della fotocamera in eq. 3, le equazioni cartesiane dei piani che includono i due lati risultano in eq. 4, sostituendo le coordinate X_c e $\boldsymbol{Y_c}$ della fotocamera in eq. 3 con le coordinate x e y di eq.2. Il risultato è mostrato in eq. 4 [40].

$$\boldsymbol{P} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{22} & P_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} hx_c \\ hy_c \\ h \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad \text{(eq. 3)}$$

$$A_1P_{11}X_c + (a_1P_{12} + b_1P_{22})Y_c + (a_1P_{13} + b_1P_{23} + c_1)Z_c = 0$$
$$A_2P_{11}X_c + (a_2P_{12} + b_2P_{22})Y_c + (a_2P_{13} + b_2P_{23} + c_2)Z_c = 0 \quad \text{(eq. 4)}$$

Considerando n_1 e n_2 i versori normali dei piani individuati, il versore dei due lati paralleli della superficie quadrata è ottenuto dal prodotto vettoriale $\mathrm{n}_1 x\, n_2$. Dati u_1 e u_2, versori delle due coppie di lati paralleli, questi dovrebbero essere perpendicolari, tuttavia, a causa di errori del processamento immagini, le

perpendicolarità non sarà esattamente verificata. Per compensare a questo errore, due versori v_1 e v_2 sono definiti nel piano che include v_1 e v_2 come mostrato in Figura 4.9.

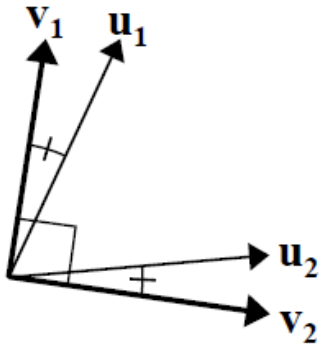


Figura 4.10: versori perpendicolari v_1 e v_2 calcolati a partire da u_1 e u_2.

Dato v_3 versore perpendicolare ad ambo v_1 e v_2, la componente di rotazione V_{3x3} della matrice di trasformazione $\boldsymbol{T_{cm}}$ è pari a $[V_1^t \quad V_2^t \quad V_3^t]$. In eq. 5 è mostrata la relazione che lega questa matrice alle coordinate della camera e del marker, dove W_x W_y W_z sono le componenti di traslazione che si ottengono, equivalentemente a quelle di rotazione, da eq. 3, ottenendo così il sistema a sei gradi di libertà utilizzato da ARToolKit.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = \quad \text{(eq. 5)}$$

$$= \begin{bmatrix} V_{3x3} & W_{3x3} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix} = T_{cm} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

Questa matrice è adatta alla libreria grafica OpenGL (da cui è stata creata OpenGL ES per i dispositivi mobile) usata dai creatori di ARToolKit per i loro esperimenti [42].

4.4 Natural Feature Tracking

Basato sullo stesso principio del *template matching* usato per i marker, il sistema *natural-feature-tracking*, anche chiamato più semplicemente *image recognition* [66], consente di realizzare esperienze di Realtà Aumentata *markerless*, precaricando nel sistema delle *features*, ovvero dei punti d'interesse, di un'immagine opportunamente 'allenata' [52]. ArtoolKit alla sua nascita, nel 1999, possedeva soltanto la capacità di rilevare marker posti nella scena reale. Furono gli stessi autori che, nel 2003, proposero il primo metodo per rilevare immagini reali tramite il sistema di *natural-feature-tracking*, creando le fondamenta della libreria usata per un caso studio di questo elaborato: ARToolKit NFT.

Il processo di rilevamento dell'oggetto e della stima della posa segue un percorso simile a quello per i sistemi *marker-based*, la differenza principale è che qui viene inserita un'immagine 2D piuttosto che un marker. Il sistema di coordinate utilizzato per effettuare il calcolo è mostrato in Figura 4.11.

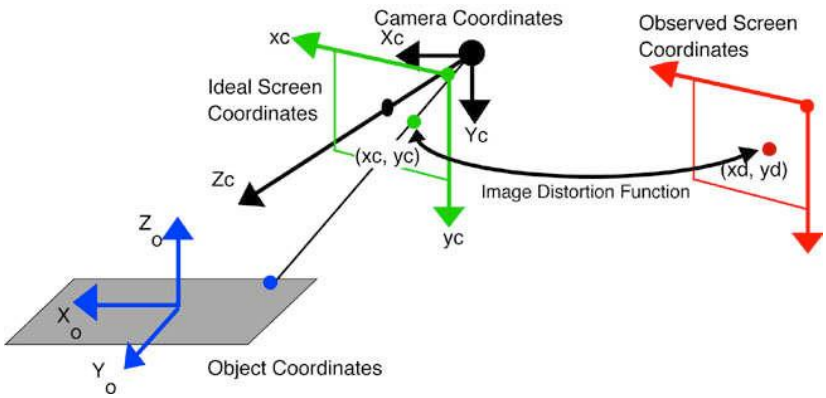


Figura 4.11: Relazione tra coordinate dell'oggetto e della fotocamera [53].

Tracciato l'oggetto, le sue coordinate X_0 e Y_0, centrate su di esso, giacciono nel piano individuato dall'oggetto, mentre Z_0 è perpendicolare alla superficie. La fotocamera possiede le sue coordinate e c'è una relazione prospettica tra queste e le coordinate dello *screen*. Fino a qui niente di particolarmente differente rispetto al caso discusso in precedenza; tuttavia, l'innovazione apportata in questo metodo sono le coordinate x_d e y_d osservate nello screen. Mentre prima non era presente questa distinzione, qui è stato utilizzato questo stratagemma per

compensare alla distorsione che è intrinseca nella lente della fotocamera. Si vedono le coordinate dello *screen* ideale, indicate come x_c e y_c, che mantengono la trasformazione prospettica con le coordinate della fotocamera, e le coordinate dello *screen* osservato, appena menzionate, che rappresentano l'immagine distorta dalla fotocamera. La trasformazione delle coordinate dell'oggetto (X_0, Y_0, Z_0) in quelle dello schermo ideale sono rappresentate in eq. 6 [53], dove è evidente la somiglianza con eq. 5 vista in precedenza.

$$h\begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ 0 & C_{22} & C_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix} = \boldsymbol{C} \cdot \boldsymbol{T_{co}} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ \mathbf{1} \end{bmatrix} \quad \text{(eq. 6)}$$

Dove $\mathbf{C}$, che svolge lo stesso ruolo di $\mathbf{P}$ in eq. 3, contiene i parametri intrinseci della fotocamera: lunghezza focale, fattore di scala, centro ottico, proporzioni e fattore di inclinazione basato sul modello di proiezione prospettica. La matrice $\boldsymbol{T_{co}}$ rappresenta la trasformazione tra le coordinate dell'oggetto nelle coordinate della fotocamera e consiste di una componente traslazione e di una componente rotazione, l'equivalente di $\boldsymbol{T_{cm}}$ in eq. 5.

Il punto (x_c, y_c) nelle coordinate dello screen ideale, è trasformato nel punto (x_d, y_d) nelle coordinate dello *screen* osservato, tramite la funzione di distorsione:

$$x = s(x_c - x_{d0}), \qquad y = s(y_c - y_{d0})$$
$$d^2 = x^2 + y^2$$
$$p = \{1 - f d^2\}$$
$$x_d = px + x_{d0}, \qquad y_d = py + y_{d0} \qquad \text{(eq. 7)}$$

Dove (x_{d0}, y_{d0}) è il centro dello schermo osservato, s è il parametro di scalatura e f è il fattore di distorsione. Questa trasformazione è una funzione non lineare rappresentata in eq. 8:

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = \boldsymbol{F} \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} = \boldsymbol{F^{-1}} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \qquad \text{(eq. 8)}$$

A questo punto, effettuato il tracciamento, quello che rimane è stimare la posa calcolando la matrice di trasformazione $\boldsymbol{T_{co}}$.

Considerando n punti caratteristici, o *features*, con coordinate del piano dell'oggetto (X_{oi}, Y_{oi}, Z_{oi}) dove *i=0,1…n-1*, questi vengono proiettati nelle coordinate dello *screen* osservato (x_{di}, y_{di}), bisogna considerare una funzione di errore rappresentata nelle coordinate dello *screen* ideale (eq. 9).

$$err^2 = \frac{1}{n}\sum_{i=0}^{n-1}\{(x_{ci} - \tilde{x}_{ci})^2 + (y_{ci} - \tilde{y}_{ci})^2\} \qquad \text{(eq. 9)}$$

$$\begin{bmatrix} x_{ci} \\ y_{ci} \\ 1 \end{bmatrix} = \boldsymbol{F^{-1}} \begin{bmatrix} x_{di} \\ y_{di} \\ 1 \end{bmatrix}, \qquad h\begin{bmatrix} \tilde{x}_{ci} \\ \tilde{y}_{ci} \\ 1 \end{bmatrix} = \boldsymbol{C \cdot T_{co}} \begin{bmatrix} X_{oi} \\ Y_{oi} \\ Z_{oi} \\ 1 \end{bmatrix} \qquad \text{(eq. 10)}$$

La matrice $\boldsymbol{T_{co}}$ è quella che minimizza la funzione d'errore e se n è maggiore o equivalente a 3 questo calcolo è possibile.

Per calcolare valori consecutivi per $\boldsymbol{T_{co}}$, è necessario rilevare posizioni nelle coordinate dello screen osservato che corrispondono a n punti caratteristici nelle coordinate dell'oggetto. Per questo si usa un *template matching*, o corrispondenza d'immagini.

L'oggetto tracciato esiste nel piano $X_o - Y_o$, quindi i punti *features* possono essere rappresentati come $(X_{oi}, Y_{oi}, 0)$ per $i=0,1,\ldots,n$. Data la matrice di trasformazione dalle coordinate dell'oggetto alle coordinate della camera $\boldsymbol{T_{co}}$, le coordinate dello *screen* osservato (x_{di}, y_{di}) corrispondenti a $(X_{oi}, Y_{oi}, 0)$ possono essere calcolate da eq. 6 e eq. 7. Per generare il *template* nel quale (x_{di}, y_{di}) è centrato, è necessario conoscere la trasformazione dalle coordinate dello *screen* osservato alla superficie $X_o - Y_o$ nelle coordinate dell'oggetto. Dato Z_o=0, si ottiene la seguente equazione.

$$h\begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} = \boldsymbol{C} \cdot \boldsymbol{T_{co}} \begin{bmatrix} X_o \\ Y_o \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ 1 \end{bmatrix} = \boldsymbol{P} \begin{bmatrix} X_o \\ Y_o \\ 1 \end{bmatrix} \qquad \text{(eq. 11)}$$

Quindi, tramite eq. 9:

$$\frac{1}{h}\begin{bmatrix} X_o \\ Y_o \\ 1 \end{bmatrix} = \boldsymbol{P}^{-1} \begin{bmatrix} x_c \\ y_x \\ 1 \end{bmatrix} = \boldsymbol{P}^{-1} \cdot \boldsymbol{F}^{-1} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \qquad \text{(eq. 12)}$$

Tramite eq. 12, le coordinate dell'oggetto, per ogni pixel dell'immagine *template*, sono calcolate e i valori dei colori, provenienti dall'immagine dell'oggetto tracciato, possono essere sostituiti nel *template*.

Per effettuare il *matching* dei *template*, gli autori in [53] hanno utilizzato una correlazione normalizzata (eq. 13) considerando x_i come valore del pixel, $\tilde{x}$ come la media dei valori dei pixel totali, y_i come il valore del *template* e $\tilde{y}$ come la media dei valori *template*.

$$s = \frac{\sum_{i=1}^{N}(x_i - \tilde{x}) \cdot (y_i - \tilde{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \tilde{x})^2} \sqrt{\sum_{j=1}^{N}\left(y_j - \tilde{y}\right)^2}} \qquad \text{(eq. 13)}$$

Il valore ottenuto appartiene all'intervallo [-1,1]. La posizione dove si trova il valore massimo ottenuto determina la corrispondente posizione (x_{di}, y_{di}) nello *screen* osservato con la posizione del punto *feature* $(X_{oi}, Y_{oi}, 0)$ nelle coordinate dell'oggetto. Ottenendo una corrispondenza per 3 posizioni, $\boldsymbol{T_{co}}$ può essere trovata per il frame contenente l'immagine corrente.

Questo metodo, permettendo un miglior tracciamento e una miglior registrazione grazie alla funzione di distorsione, è stato poi adottato anche per i sistemi *marker-based*, aggiungendo uno *screen observed* allo schema già mostrato e discusso di Figura 4.8, come si può vedere in Figura 4.12 [54].

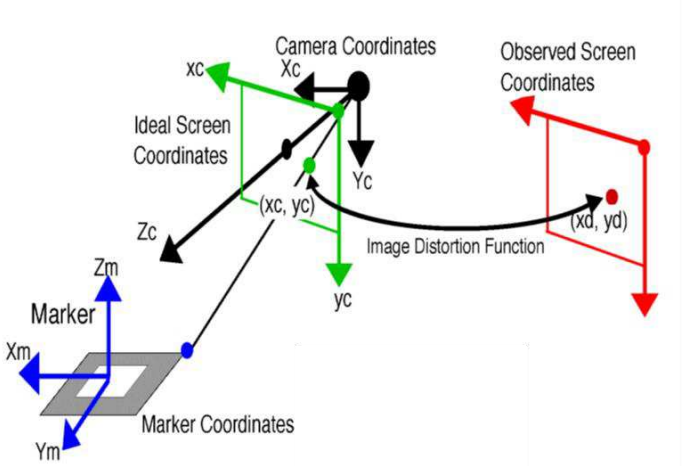


Figura 4.12: Sistema di coordinate con marker, fotocamera, schermo ideale e schermo orsservato.

4.4.1 Punti d'interesse

I punti d'interesse, o *features*, di un'immagine sono, come si è visto, alla base del *image matching* e della stima della posa per i sistemi di *Natural Feature Tracking*, qui si discuterà come questi punti vengono selezionati.

ARToolKit utilizza un algoritmo che si basa sul sistema SIFT (*Scale Invariant Feature Transform*), ideato da David G. Lowe nel 1999 [76][77]. Lo scopo di questo algoritmo è quello di identificare dei punti in un'immagine che sono invarianti

rispetto alla traslazione, rotazione, ridimensionamento e variazione di luce dell'immagine stessa.

Per ottenere questo risultato devono essere selezionati punti di interesse[12] adatti e per farlo si applica una funzione Gaussiana all'immagine che deve essere allenata. La suddetta funzione, a una dimensione, è mostrata in eq. 14 e sarà applicata in direzione orizzontale e verticale.

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-x^2/2\sigma^2} \qquad \text{(eq. 14)}$$

Il ruolo di questa funzione è quello di distorcere l'immagine, simulando ciò che potrebbe avvenire in un ambiente reale. I *key point* che vengono estratti derivano dall'utilizzo della piramide Gaussiana: una struttura a livelli dove alla base si trova l'immagine iniziale e ogni livello consiste in un campionamento di questa.

All'immagine iniziale viene convoluzionata la funzione Gaussiana usando $\sigma = \sqrt{2}$, dove sigma è la deviazione standard, e si ottiene come risultato un'immagine A. Questo poi viene ripetuto con un ulteriore incremento della deviazione standard di $\sqrt{2}$, ottenendo una nuova immagine B, che ha un σ =2. S i procede poi sottraendo l'immagine B all'immagine A ottenendo la differenza di Gaussiana (DoG, dall'inglese Difference of Guassian). Per costruire il successivo livello della piramide, si ricampiona l'immagine B usando un'interpolazione bilineare con una separazione di pixel pari a 1.5 in ogni direzione, per minimizzare il problema dell'*aliasing*[13].

A questo punto si devono individuare i punti minimi o massimi della DoG, che si determinano confrontando ogni pixel nella piramide con i suoi pixel vicini. Prima si confronta ogni pixel con gli 8 pixel più vicini ad esso nello stesso livello della piramide. Se questo è un massimo o un minimo viene calcolata la posizione del pixel più vicino al livello più basso della piramide, tenendo conto del ricampionamento di 1.5. Se il pixel rimane più alto o più basso del pixel più vicino e dei suoi 8 vicini, il test viene ripetuto per il livello superiore. In questo modo verranno eliminati un gran numero di pixel.

Per osservare un risultato che si può ottenere con questo algoritmo, si può considerare il generatore di marker NFT utilizzato da ARToolKit e consultabile in

[12] Chiamati da Lowe anche *key point*
[13] Perdita di informazione dovuta al campionamento

[78]. L'immagine in input viene convertita in scala di grigi e da questa si applica l'algoritmo per estrarre i punti d'interesse. Il risultato è quello di ottenere un file *.iset*, contenente l'immagine in bianco e nero, e due file *.fset* e *.fset2* contenenti i punti d'interesse.

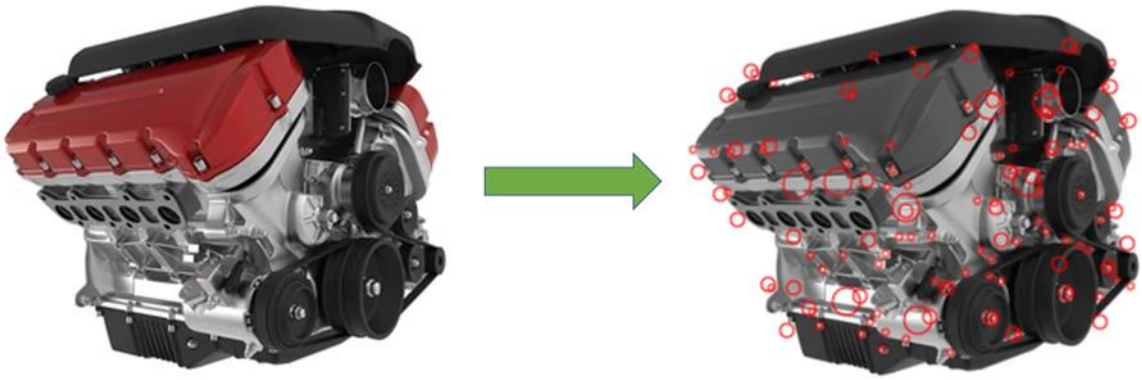


Figura 4.13: Immagine originale e immagine in scala di grigi, con i suoi punti d'interesse in evidenza

Un esempio è mostrato in Figura 4.13, dove si può intuire come i punti d'interesse di un'immagine sono quei pixel che formano il 'descrittore' di una certa *feature* [77]. Un descrittore è un vettore che descrive l'intorno di una *feature* in modo che sia indipendente dal sistema di riferimento dal quale si osserva l'immagine. Si vede dalla figura come questi punti siano il più possibile diversi tra loro.

Capitolo 5
Analisi dei requisiti

In questo capitolo si illustrerà la prima fase dello sviluppo dell'app in questione, ovvero l'analisi dei requisiti. Questa è una parte cruciale nella progettazione non solo di ogni applicazione mobile, ma anche di ogni software o sito Web in generale. È in questo momento che si gettano le basi per la bozza del progetto, individuando le funzionalità che l'app deve fornire all'utente.

L'analisi dei requisiti è spesso preceduta da una raccolta delle informazioni, dalla quale emerge la *specifica dei requisiti*, che descrive tutte le funzionalità desiderate nel progetto. La raccolta delle informazioni è spesso affiancata ad un'intervista al cliente che commissiona l'applicazione, tuttavia, in questo elaborato, non avendo un cliente specifico, si è ipotizzato di rivolgersi ad un generico proprietario di un'azienda di produzione industriale.

Si procederà quindi con una raccolta informazioni in questo contesto, illustrando, infine, i requisiti funzionali e non che l'applicazione dovrà soddisfare.

5.1 Raccolta delle informazioni

Prima della definizione dei requisiti del sistema, occorre recuperare quante più informazioni utili al fine di avere un quadro completo del contesto in cui si sta operando. Generalmente tali informazioni vengono raccolte attraverso delle interviste al committente e agli utenti dell'eventuale tecnologia che si sta sostituendo. Inoltre, qualora quest'ultima esista, è opportuno analizzarne i punti di forza e di debolezza per capire quali aspetti mantenere nel nuovo progetto, quali migliorare e quali aggiungere.

Come già accennato, il committente di questo progetto di tesi è un ipotetico imprenditore che vuole migliorare l'attività produttiva della sua azienda. Per farlo vuole servirsi di un'applicazione che possa guidare l'operatore sul lavoro da svolgere, ottenendo come risultato un miglioramento dell'efficienza dell'operatore stesso, e quindi dell'attività produttiva in generale.

Desiderio principale dell'operatore è quello di 'digitalizzare' la sua impresa, trasformandola quindi in un'industria 5.0 (si veda il Capitolo 3 – Industria 5.0) per stare al passo con i tempi. Ha quindi investito risorse nel settore in quest'ottica, riuscendo a provvedere un moderno dispositivo smartphone o tablet ad ogni operatore. La sua richiesta è di utilizzare questi dispositivi per sfruttare la tecnologia di Realtà Aumentata affiancandola ad attività di produzione industriale. Tramite questo approccio l'operatore, sfruttando la fotocamera del dispositivo che sta utilizzando, verrà guidato dai contenuti virtuali, che appariranno nello schermo, nell'attività da svolgere. La produzione è quindi migliorata sia dal punto di vista dell'imprenditore, che vedrà una maggiore efficienza dei suoi dipendenti, sia dal punto di vista degli operatori stessi, che saranno facilitati nelle loro mansioni.

Si è anche ipotizzato che l'azienda in questione avesse già installata un'architettura di *Internet of Things*, nella quale dei sensori installati su macchinari di produzione forniscono informazioni riguardanti lo stato del macchinario stesso. Il progetto da realizzare dovrà essere in grado di interfacciarsi con questi sensori e mostrare il contenuto all'utente che ne fa uso, direttamente sul suo smarthpone o tablet.

5.2 Specifica dei requisiti

Dopo aver terminato la raccolta delle informazioni si procede ad una traduzione di queste in una lista di requisiti che l'applicazione dovrà soddisfare, fase che prende il nome di specifica dei requisiti. Questi requisiti servono a delineare meglio le funzioni che l'applicazione dovrà soddisfare e si dividono in requisiti funzionali e non funzionali. Con i primi si intende la totalità delle caratteristiche che il progetto finito dovrà implementare, i secondi invece vanno a delineare quegli aspetti trasversali che riguardano per lo più il lato tecnico della realizzazione del progetto. Entrambi sono allo stesso modo di cruciale importanza per lo sviluppo dell'applicazione.

5.2.1 Requisiti funzionali

Le funzionalità che l'applicazione dovrà implementare sono riassunte in Figura 5.1, dove si vede lo schema gerarchico che esse seguono. Alcune funzionalità, infatti, determinano il corretto funzionamento di altre.

Permessi fotocamera e sensori di movimento: un'applicazione di Realtà Aumentata presuppone l'utilizzo di fotocamera, per mostrare all'utente l'ambiente circostante in cui si trova, e di sensori di movimento, per poter scalare ed orientare il contenuto virtuale che verrà visualizzato a seconda della posizione dell'osservatore. Quindi, funzionalità principale dell'applicativo è quella di mostrare una schermata iniziale, dalla quale l'utente riuscirà a concedere i permessi necessari a garantire il corretto funzionamento di ogni altra funzionalità dell'applicativo.

Riconoscimento target: questa funzione consentirà all'applicazione di individuare e riconoscere alcuni *target*, che possono essere marker, immagini o oggetti reali. Ovviamente, il riconoscimento avverrà attraverso la fotocamera del dispositivo, perciò questa funzionalità dipende dalla concessione di tale permesso descritta in precedenza.

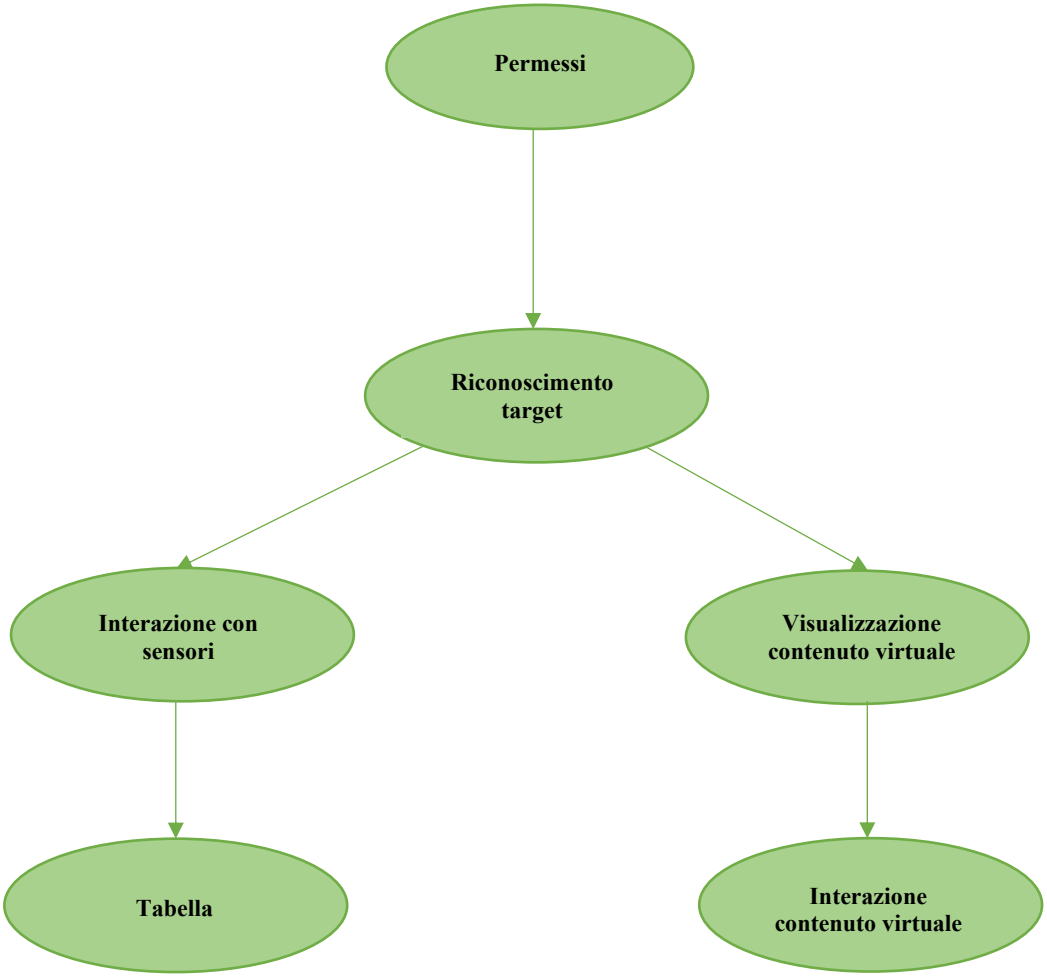


Figura 5.1: Schema requisiti funzionali.

Interazione con sensori: questa funzione consentirà all'applicazione di poter interagire con dei sensori simulati, i quali forniscono informazioni riguardanti lo stato e l'attività di alcuni macchinari industriali. Queste informazioni saranno salvate in un servizio Web e la chiamata a tale servizio al riconoscimento del target. In questo modo l'utente potrà interagire con il servizio ogni volta che lo ritiene necessario, basterà visualizzare il target attraverso la fotocamera del dispositivo.

Tabella: i dati raccolti dai sensori dovranno poter essere visualizzati dall'utente. Questa funzione mostrerà una tabella a tale scopo, raccogliendo i dati e mostrandoli all'utente.

Visualizzazione contenuto virtuale: la Realtà Aumentata presuppone, per definizione, la visualizzazione di un contenuto tridimensionale. Con questa funzionalità l'applicazione si occuperà della resa di tale contenuto virtuale basandosi sul riconoscimento del target, dalla cui posizione e orientamento ne adatterà l'oggetto in questione.

Interazione con contenuto virtuale: l'utente, per essere guidato nelle sue attività lavorative, oltre a vedere il contenuto aumentato, sovrapposto alla realtà aziendale in cui si trova, dovrà anche poter interagire con esso. Questo facilita un maggior coinvolgimento dell'operatore con il servizio che sta utilizzando, oltre che un miglior approccio nel guidare il suo lavoro. L'interazione con il contenuto avverrà tramite *touch* da parte dell'utente sul contenuto stesso, il quale cambierà comportamento a seconda dei dati mostrati nella tabella descritta precedentemente.

5.2.2 Requisiti non funzionali

I requisiti funzionali riguardano quegli aspetti trasversali che l'applicativo deve soddisfare, ma che non influiscono direttamente all'erogazione delle sue funzionalità principali. Essi riguardano principalmente gli aspetti di prestazioni e design, responsabili di offrire un'esperienza migliore all'utente che utilizza l'applicazione.

Lo scopo principale del progetto è, infatti, realizzare un'applicazione Web che sia in grado di aiutare non solo l'operatore di un'industria 'digitalizzata' nelle sue mansioni lavorative, ma un occhio di riguardo è stato posto anche dal punto di vista del datore di lavoro. L'applicativo, forte di una elevata semplicità, è in grado di essere facilmente intuibile e utilizzabile dall'utente, ma anche facilmente modificabile dal datore di lavoro, o da chiunque in azienda abbia accesso al sistema, per adattarlo a nuove esigenze, ad esempio un cambio di macchinari o un nuovo tipo di dato raccolto dai sensori.

Di seguito vengono riportati in dettaglio i requisiti non funzionali che il progetto si pone di soddisfare.

Cross-platform: la necessità di offrire un approccio che sia utilizzabile da ogni dispositivo, sia esso smartphone o tablet, ma anche computer desktop (sebbene non necessario per la sua impraticabilità nel presente contesto) e che funzioni in ogni sistema operativo, sia esso Android o iOS, nasce dal momento che l'ipotetico

committente del progetto non ha specificato quale tipo di device ha acquistato per la sua azienda. Fornire un servizio *cross-platform* consente di far fronte a questa mancanza, prevenendo anche qualsiasi futuro acquisto di device differenti.

Usabilità: la visualizzazione dei dati erogati dai sensori dovrà essere chiara e visibile all'utente. Poiché egli, per tutto il tempo in cui sta utilizzando l'applicazione, manterrà la fotocamera aperta, i dati devono essere mostrati in modo da non occultare la visuale del mondo reale, ma riuscendo comunque ad essere chiaramente visibili, in quanto l'utente deve conoscerne il contenuto e capire a cosa esso è riferito.

Inoltre, l'interazione con il contenuto virtuale deve essere effettuata tramite *touch* dell'utente sul contenuto virtuale stesso, che diventa l'unica azione richiesta all'utente per poter apprezzare le funzionalità dell'app. Il pulsante diventa così il contenuto virtuale stesso, che è ben visibile da chi sta utilizzando l'applicazione.

Reattività: l'utilizzo della Realtà Aumentata comporta un notevole sforzo dal punto di vista delle prestazioni del dispositivo e questo può influire sulla reattività con cui l'applicazione, dopo aver scansionato il marker o un'immagine, mostra il contenuto virtuale e risponde all'interazione dell'utente. Inoltre, gli utenti mobile sono in generale meno pazienti di quelli che utilizzano un computer desktop; è quindi importante che l'app risponda alle richieste in tempi brevi, favorendo la *user-experience*.

Accessibilità: l'applicazione deve essere intuibile e di facile utilizzo. Deve svolgere la sua funzione nel modo più chiaro possibile e l'utente non deve impiegare tempi considerevoli nell'apprendere come utilizzarla né tantomeno nella sua installazione.

5.3 Applicazioni native, ibride e Web: un confronto

Dai requisiti visti nelle sezioni precedenti nasce la scelta di realizzare una Web-app, considerando questa come la migliore alternativa per soddisfare ogni specifica. In questo paragrafo verranno valutati i motivi di questa scelta, illustrando un confronto tra applicazioni native, applicazioni ibride e applicazioni Web.

I telefoni cellulari, dalla loro nascita, erano stati pensati per svolgere poche funzioni, quali, in primis, effettuare chiamate, poi nel corso degli anni si aggiunsero anche i messaggi di testo, la capacità di fare foto e di connettersi a Internet. Con Apple e l'avvento del suo primo iPhone, venne rivoluzionato il concetto classico di smartphone e l'aspettativa dell'utente riguardo le esperienze che potessero essere usufruite da questi dispositivi cambiò radicalmente [31].

Si inizia quindi a parlare di applicazioni mobile, *mobile application*, ovvero software dedicati ai dispositivi mobile, progettati secondo le caratteristiche hardware e dei sistemi operativi dei dispositivi stessi [56]. Proprio a causa dei diversi sistemi operativi che ci sono in circolazione, su tutti Android ed iOS, le applicazioni diventano native, nel senso che sono sviluppate per uno specifico sistema operativo e tramite un determinato linguaggio di programmazione.

5.3.1 Applicazioni Native

Le app native sono applicazioni sviluppate per una specifica piattaforma, così da riuscire ad interfacciarsi al sistema operativo nel modo più completo possibile. Esse, infatti, riescono ad interagire del tutto con le API della piattaforma, coinvolgendo interamente l'hardware e il software del dispositivo. Di conseguenza si avrà accesso a tutte le componenti e delle loro funzionalità (fotocamera, GPS, giroscopio, etc.) integrandole all'applicativo. Questo rende le app native più veloci ed affidabili in quanto a prestazioni, oltre al fatto di esaltare la *user experience*. Potendo utilizzare elementi grafici nativi della piattaforma, si possono realizzare applicazioni maggiormente intuitive dal punto di vista dell'utente, il quale troverà un *look and feel* comune con altre app già utilizzate.

Tuttavia, da tenere in considerazione, sono i costi che sviluppare un'app nativa comporta. Dovendo utilizzare un linguaggio specifico per la sua progettazione, se si vuole raggiungere un numero maggiore di utenti si dovrà effettuare uno sforzo

per scrivere l'app in diversi linguaggi, rendendola quindi accessibile ad ogni piattaforma. Questo ha un notevole costo dal punto di vista dello sviluppo, in quanto, pur mantenendo intatta la logica dell'applicazione, architettura e grafica dovranno essere cambiate. Stesso discorso vale per la manutenzione e l'aggiornamento: ogni app nativa avrà bisogno di essere mantenuta e aggiornata in maniera diversa, a seconda del sistema operativo per il quale è stata creata.

Per ovviare a questo problema, si è iniziato a pensare allo sviluppo *cross-platform*, nel quale si trovano due soluzioni possibili: applicazioni ibride e applicazioni Web.

5.3.2 Applicazioni ibride

Le App ibride sono realizzate per adattarsi a dispositivi differenti. Spesso sono sviluppate come app native provviste di interfacce HTML pari alle Web app. Esse rappresentano un buon compromesso in termini di funzionalità tra app native e Web. Sono più rapide da manutenere delle app native, in quanto il processo di sviluppo è unico, anche se le performance sono inferiori e presentano una stabilità ridotta.

Le app ibride, tuttavia, posseggono uno svantaggio tipico di qualsiasi app nativa: è necessaria la sua installazione nel dispositivo. Questo comporta uno sfruttamento della memoria interna del device, oltre al tempo d'attesa necessario all'installazione. Al contrario, le Web-app, specialmente negli ultimi anni, stanno suscitando un notevole interesse, proprio per la facilità nella loro distribuzione e la loro pesantezza pressoché nulla in memoria.

5.3.3 Applicazioni Web

Le applicazioni Web sono accessibili per l'utente mediante la rete e si adattano ad ogni piattaforma, poiché mediante il *browser* riescono a comunicare con il sistema operativo [31], risparmiando così notevoli risorse evitando di scrivere l'app in diversi linguaggi. Per la stessa ragione riescono a funzionare anche in ogni tipo di dispositivo, sia esso uno smartphone, un tablet o un computer desktop e non richiedono alcuna installazione da parte dell'utente, che può quindi risparmiare memoria locale.

I vantaggi di una Web-app in Realtà Aumentata sono apprezzabili, perciò, non solo dal punto di vista dell'utente, ma anche dal punto di vista dello sviluppatore. Quest'ultimo, infatti, con questo approccio, evita i numerosi sforzi tipici nella

pubblicazione di un'app nativa (validazione, tempo di pubblicazione) e può usare tecnologie ben consolidate, tipiche di una pagina Web e ben noti, quali Javascript, HTML e CSS [32]. Ogni aggiornamento, inoltre, è direttamente lanciato in tempo reale, così come ogni *bug fixing* e manutenzione generale [26], e questo viene apprezzato anche dal lato utente, che vede quindi ogni modifica in tempo reale.

Inoltre, la quasi totalità dei browser attualmente in commercio consentono l'accesso alle componenti hardware del dispositivo, previo consenso degli utenti.

Questi vantaggi si pagano con una penalizzazione delle prestazioni, che qui dipendono non solo dalle capacità del dispositivo, ma anche dalla connessione disponibile. Inoltre, alcune delle funzionalità native vanno perse, come ad esempio l'esecuzione di task in background, la ricezione di notifiche ed il salvataggio di dati.

Figura 5.2: Confronto app native, Web e ibride [68]

5.4 Realtà Aumentata nel Web

Se da un lato, come si è visto nel Capitolo 4 – Sistemi di Realtà Aumentata nei dispositivi mobile, i componenti che troviamo nello smartphone di oggi lo rendono un terreno ideale per lo sviluppo di applicazioni in realtà aumentata, dall'altro un progresso continuo nella connessione di rete, che trova attualmente il suo apice nel 5G [20], favorisce una simbiosi tra AR e Web. Si parla quindi di *Web AR*, realtà aumentata nel Web, un approccio che ha tutte le potenzialità di cambiare il nostro modo di approcciarci al mondo fisico che ci circonda [18].

Un sistema di Realtà Aumentata può sfruttare il Web per due ragioni distinte. La prima è per sfruttare un database esterno nel quale immagazzinare i contenuti virtuali da visualizzare, la seconda è prettamente indirizzata all'uso di dispositivi mobile, per offrire all'utente un'esperienza senza dover necessariamente installare alcuna applicazione.

Per progettare un sistema di Realtà Aumentata in un dispositivo mobile, specialmente uno smartphone, come affermato più volte, bisogna considerare le limitate capacità computazionali del device. Questa limitazione è però giustificata dall'elevata portabilità di questi dispositivi e può essere compensata tramite una connessione ad Internet. Utilizzare un *cloud surrogate* è la soluzione migliore per far fronte al problema del *rendering* e dell'immagazzinamento dei contenuti virtuali che saranno utilizzati per 'aumentare' la realtà [55]. Lo schema generale del funzionamento di questa tecnologia è riportato in Figura 5.3 [29].

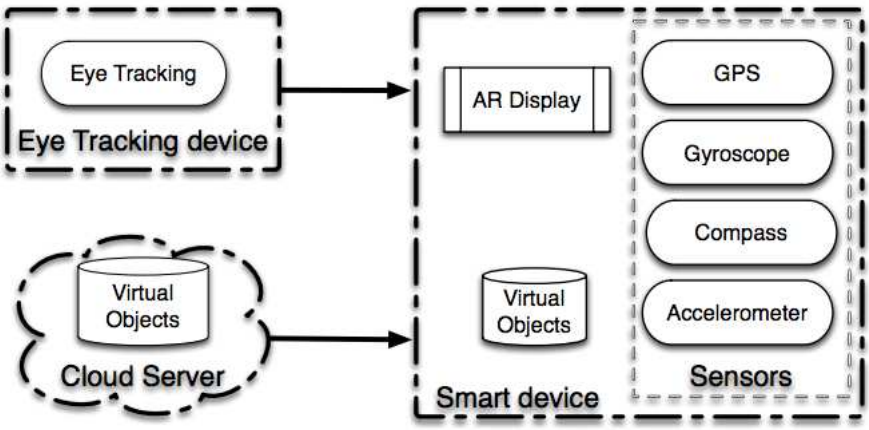


Figura 5.3: Schema generale del funzionamento di un sistema MAR che utilizza il Web per immagazzinare i dati in un server cloud.

Nello schema, si intuisce come la maggior parte dei componenti responsabili dell'esecuzione del sistema di Realtà Aumentata nei dispositivi mobile siano i sensori installati nel dispositivo stesso. Un *cloud server* si occupa solamente dello *storing* degli oggetti virtuali che verranno poi utilizzati dal *client* (in questo caso lo smartphone) per la resa dei contenuti. Il device per il tracciamento dell'occhio umano non è sempre necessario, ma alcuni device come Google Glass possono connettersi al dispositivo mobile per visualizzare gli oggetti virtuali ed offrire un'esperienza più immersiva all'utente [29]. Per la realizzazione di questo elaborato, il ruolo di cloud server è stato ricoperto da una repository GitHub (si veda Appendice B – GitHub).

Capitolo 6
Strumenti utilizzati

Nel capitolo che segue verranno presentati in dettagli gli strumenti che sono stati utilizzati per la progettazione del progetto, cuore di questo elaborato di tesi. Si farà luce sulle motivazioni che hanno spinto ad effettuare queste scelte, evidenziandone i loro vantaggi.

Per completezza, in Appendice A – Altri framework, è riportata una serie di framework che si sarebbero potuti tenere in considerazione per realizzare questo progetto.

6.1 IDE

Di seguito verrà presentata una panoramica per spiegare il concetto degli IDE. Verranno descritte le loro caratteristiche principali, evidenziando l'importanza che rivestono in fase di sviluppo di un codice. Poi si analizzerà l'IDE selezionato per la realizzazione di questo progetto di tesi, valutando le motivazioni che hanno condotto a tale scelta

Un IDE, ovvero *Integrated Development Environment*, o, tradotto, ambiente di sviluppo integrato, è un software progettato per supportare gli sviluppatori in fase di programmazione, aggregando tutti gli strumenti di cui egli ha più necessità in un'unica interfaccia utente.

Gli elementi principali che costituiscono un IDE sono i seguenti:

Editor di codice sorgente.
Si tratta di un editor di testo che, in quanto tale, assiste l'utente durante la scrittura di un codice, evidenziando errori di sintassi e proponendo suggerimenti adeguati, completando parti del testo in maniera specifica al linguaggio di programmazione e notificando eventuali bug di scrittura.

Compilatore.
Le istruzioni scritte nel codice sorgente in un determinato linguaggio di programmazione devono essere poi tradotte in istruzioni di un altro linguaggio, il linguaggio oggetto o linguaggio macchina. Questo processo è eseguito dal compilatore ed il codice generato è poi passato al linker.

Linker.
Il *linker* è quel programma che effettua il cosiddetto *linking*, ovvero il collegamento tra il codice oggetto (descritto sopra) e tutti quei moduli a cui un programma fa riferimento, che possono essere sottoprogrammi o librerie.

Debugger.
Insieme al compilatore, il *debugger* è uno degli strumenti più importanti a disposizione di un programmatore, anche se non necessariamente sempre presente. La funzione principale di questo programma o software è quella di analizzare ed eventualmente eliminare i cosiddetti *bug*[14] all'interno di un codice.

[14] Anomalia all'interno di un software che non ne permette il corretto funzionamento.

Una volta individuato un *bug*, il compito del *debugger* è quello di mostrare all'utente il frammento di codice che genera il problema, evidenziandolo. Il codice può essere mostrato nella sua forma nativa, tradotto in linguaggio macchina o anche sotto forma di codice sorgente nel linguaggio di programmazione in cui il programma analizzato è stato scritto.

Un programma eseguito in modalità debug richiede una compilazione con una maggior quantità di istruzioni rispetto alla compilazione normale. Questo penalizza la velocità di esecuzione del programma, che risulta più lenta rispetto al caso in cui viene effettuata direttamente sul processore per cui il programma è stato sviluppato.

Generalmente i debugger consentono l'esecuzione del programma da analizzare a piccoli passi, in modo da rendere più chiara l'individuazione del problema all'utente. L'esecuzione presenta quindi delle interruzioni, che possono avvenire passo passo ad ogni singola istruzione oppure possono essere impostate dall'utente, in questo caso si parla di *breakpoint*. In ogni interruzione viene mostrato il codice sorgente relativo all'istruzione corrente e lo stato attuale della CPU, quindi il valore o lo stato delle variabili in gioco e le rispettive celle di memoria.

6.2 Visual Studio Code

La scelta dell'IDE per la realizzazione del progetto di questa tesi è ricaduta su Visual Studio Code e di seguito verranno spiegate le motivazioni di questa scelta.

Visual Studio Code è un editor di codice sorgente multipiattaforma, sviluppato da Microsoft nel 2015 [65]. Esso offre agli sviluppatori gli strumenti necessari per i loro cicli di codifica, compilazione e debugging, combinando semplicità ed efficienza. VS Code è nativamente supportato dai sistemi Windows, macOS e Linux e permette di lavorare con una vasta gamma di linguaggi di programmazione, come JavaScript, C#, C++, PHP, Java, HTML, R, CSS, JSON, XML e Python.
Poiché gli sviluppatori non spendono tutto il loro tempo soltanto scrivendo codice, ma anche muovendosi avanti e indietro tra codifica e debugging, quest'ultima è spesso la caratteristica che gli sviluppatori vogliono in un IDE per avere un'esperienza di codifica più snella. Il debugging in Visual Studio Code è tra le sue funzionalità più popolari, fornendo un'esperienza integrata e semplice, con supporto per Node.js.

I motivi della scelta di Visual Studio Code sono quindi evidenti. Innanzi tutto, è un software libero e gratuito, anche se la versione ufficiale è sotto una licenza proprietaria. Inoltre, la sua capacità di poter lavorare con una vasta gamma di linguaggi lo rende ideale alla realizzazione di una Web-app, la quale è, come nella maggior parte dei casi, stata scritta utilizzando 3 linguaggi: HTML, per la struttura della pagina Web, CSS per lo stile e JavaScript per la logica dell'applicativo.
VS Code inoltre offre anche la possibilità di poter lanciare la propria Web-app direttamente dal browser, equivalentemente a quanto si fa con un'app nativa tramite l'emulatore dell'IDE, ma molto più rapidamente.

6.3 AR.js

AR.js è un framework puramente *Web-based* fondato nel 2017 da Jerome Etienne, ora mantenuto da un'organizzazione GitHub, e scritto interamente in JavaScript [32]. Questo rende possibile eseguire ogni applicazione di Realtà Aumentata implementata mediante AR.js direttamente eseguibile nel browser, rendendola quindi indipendente dalla piattaforma con la quale si sta lavorando. L'unico requisito è che gli standard WebGL e WebRTC siano supportati (si veda la sezione 4.1.5), ma, come accennato in precedenza, questa è ormai una condizione comune per la stragrande maggioranza dei browser.

AR.js consente di sviluppare applicazioni in Realtà Aumentata nel Web, integrando tre funzionalità in particolare: *Marker tracking*, *Image Tracking* e *Location based*. Dei primi due approcci se ne è già discusso in precedenza nel Capitolo 4 – Sistemi di Realtà Aumentata nei dispositivi mobile. Questa libreria si poggia sugli elementi di ARToolKit visti nel Paragrafo 4.2.

La Realtà Aumentata *Location Based* usa le coordinate spaziali di punti nel mondo reale per usufruire esperienze di realtà aumentata legate al sensore GPS del dispositivo. Pur essendo oggetto di interesse, questo approccio è più adatto per situazioni di applicazioni *outdoor* che esulano dal presente elaborato. Inoltre, i risultati di questo approccio dipendono fortemente dalle capacità dei sensori di localizzazione del dispositivo [66].

Si è scelto di utilizzare questo framework perché si presta bene a soddisfare sia i requisiti funzionali e non funzionali dell'applicativo. Essendo, come già affermato, una libreria basata interamente nel Web, essa si presta ad essere sostenuta da ogni sistema operativo, sia esso Android, iOS e Windows e a funzionare in ogni piattaforma smartphone, tablet o desktop. Essendo open-source, il costo di realizzare un'applicazione tramite questo framework è pari a zero e anche questo aspetto ha influito sulla scelta, oltre che il fatto di poter accedere al codice sorgente ed estenderlo per lo specifico progetto in questione [66].

Tuttavia, questa scelta porta con sé alcuni svantaggi. Un software open-source è infatti soggetto a manutenzione discontinua, basata sul livello di attività della *community*, e può capitare che risorse e competenze aggiuntive siano richieste durante l'implementazione.
Un altro aspetto che è stato tenuto in considerazione è lo stesso linguaggio JavaScript con il quale è scritto il framework. Questo rende possibile aggiungere funzionalità di Realtà Aumentata anche ad una applicazione Web già esistente,

semplicemente importando due librerie JavaScript. Inoltre, questo linguaggio negli ultimi anni è stato migliorato sino a diventare paragonabile ai linguaggi di programmazione nativi [45].
AR.js, infine, offre la possibilità di visualizzare i contenuti grafici tramite A-Frame, uno script basato su HTML che verrà descritto nel prossimo paragrafo.

6.4 A-Frame

A-Frame [33] è un framework Web open-source che nasce originariamente come strumento per sviluppare in maniera semplice ed efficace esperienze di Realtà Virtuale, creato dal gruppo VR di Mozilla e attualmente mantenuto da sviluppatori di Supermedium e Google. Si basa sul linguaggio HTML, semplificando e rendendo più accessibile iniziare a lavorarci, in quanto non è necessaria alcuna installazione. Inoltre, è di natura *cross-platform*, permettendo quindi agli sviluppatori di implementare applicazioni per una vasta di visori o *headset* quali, ad esempio, Rift, Vive e Windows Mixed Reality, ma semplicemente anche smartphone e tablet.
Tuttavia, A-Frame non è soltanto un linguaggio di *markup*[15], ma possiede una struttura dichiarativa, estendibile e componibile grazie al suo nucleo *entity-component*, struttura tipica della programmazione ad oggetti. Ogni elemento della scena è, infatti, una *entity*, rappresentato tramite il tag `<a-entity>`, a cui è possibile assegnare valori e attributi.

Questa sua struttura lo rende un potente framework per three.js, dove gli sviluppatori creano scene 3D usando il linguaggio HTML (per un approfondimento si consiglia la lettura dell'Appendice A – Altri Framework). Con la nascita di AR.js l'utilizzo di questo framework è stato adattato anche per la Realtà Aumentata, riuscendo a creare esperienze con risultati apprezzabili fino a 60fps[16].

L'efficienza di A-Frame è stata provata anche dal fatto di essere stato utilizzato da compagnie leader del settore quali ad esempio Google, Disney, Samsung, Toyota, Ford, Chevrolet, Amnesty International, CERN, NPR, Al Jazeera, The Washington Post, NASA. Alcuni miglioramenti sono stati apportati anche dalle stesse compagnie.

[15] Anche chiamato linguaggio di formattazione, è uno standard che descrive un insieme di regole per l'impaginazione e la rappresentazione di un testo
[16] Frame per secondo

Capitolo 7
Struttura dell'applicazione

In questo capitolo verrà mostrata la struttura dell'applicazione, che andrà ad inserirsi in uno scenario industriale con architettura di tipo *Internet of Things*. Si discuterà delle componenti coinvolte all'interno di questo scenario, illustrandone le loro funzioni.

Si procederà poi con la descrizione del servizio Web realizzato per simulare il comportamento di sensori reali in questo contesto e si mostreranno alcuni frammenti di codice utilizzati per raccogliere i dati provenienti da questo servizio.

7.1 Scenario

Come indicato nei capitoli precedenti, l'applicazione realizzata ha lo scopo di affiancare l'attività lavorativa di un operatore in un'industria 5.0, e per farlo si poggia su un'architettura *Internet of Things* (discussa nel Paragrafo 3.3) di cui è mostrato uno schema in Figura 7.1.

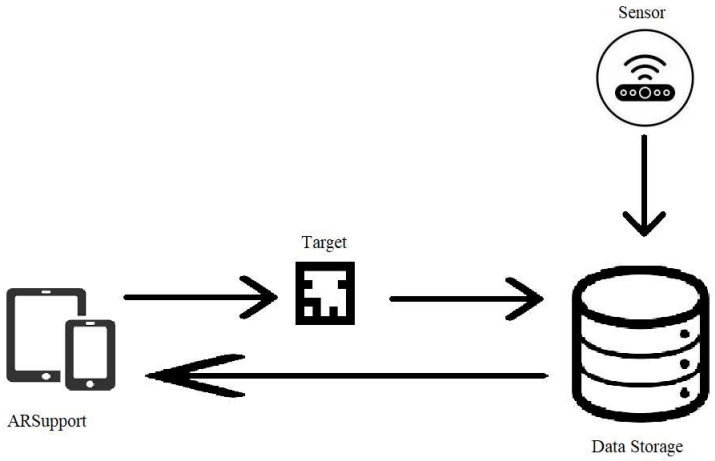


Figura 7.1 Schema generale dove si inserisce l'applicativo.

La struttura presentata nello schema si basa su quella di una smart factory, di cui un'analisi è fornita nel Paragrafo 3.4.

I soggetti che compongono la struttura dell'applicazione sono i seguenti: sensori, server e applicativo (*ARSupport*). I sensori (simulati) sono collegati ad un macchinario o ad un generico dispositivo e ne raccolgono continuamente informazioni, per poi salvare i dati ottenuti, mediante connessione alla rete, nel server che si occupa dello *storage* dei dati. L'applicativo entra in gioco dal momento che l'utente, tramite l'app, scansiona un *target*, che può essere un marker o un oggetto reale, ed effettua la chiamata al servizio Web dove sono contenuti i dati.

7.2 Protocollo HTTP

Per realizzare la chiamata al servizio Web utilizzato per lo storage dei dati ottenuti da sensori simulati, si effettua una richiesta HTTP. Il protocollo HTTP [70], ovvero *Hypertext Transfer Protocol* (tradotto: protocollo di trasferimento ipertesto) è un protocollo a livello applicativo che si basa sulla trasmissione di informazioni attraverso il Web.

La sua architettura è di tipo *client-server*, si veda la Figura 7.2, dove un terminale (client) si connette, via Internet, ad un server per usufruire di un certo servizio. È un protocollo *stateless*, ovvero si limita a fornire un servizio o svolgere una determinata funzione senza alcuna conoscenza e riferimento alle transazioni avvenute in passato; quindi, i dati realizzati dal client in una sessione non vengono salvati e non vi è memoria di essi nelle sessioni future.

Lo scambio di informazioni tra client e server avviene in questo modo: il client invia una richiesta (*request*) al server che, se la richiesta è soddisfatta, invia la risposta (*response*). Nel progetto realizzato per questo elaborato, il ruolo di client è svolto dall'applicazione Web stessa, o meglio dal browser attraverso il quale l'app viene utilizzata, mentre il ruolo di server viene assunto dalla macchina su cui risiede il servizio Web, che in questo caso sono i server dell'Università Politecnica delle Marche.

La forza del protocollo HTTP risiede nel fatto che ogni connessione instaurata tra client e server viene chiusa al momento in cui la richiesta è soddisfatta. Questo comportamento lo rende ideale nel World Wide Web, le pagine molto spesso contengono dei collegamenti (*link*) a pagine ospitate da altri server diminuendo così il numero di connessioni attive limitandole a quelle effettivamente necessarie con aumento quindi di efficienza (minor carico e occupazione) sia sul client che sul server.

Ogni richiesta HTTP è formata da un metodo, da un *URI* e dalla versione del protocollo. L'URI, *uniform resource identifier* (identificatore univoco di risorsa), indica l'oggetto della richiesta (ad esempio la pagina Web che si intende ottenere). I metodi più frequentemente utilizzati sono:

-GET, usato per ottenere il contenuto della risorsa indicata nell'URI.

-HEAD, analogo al metodo GET, ma restituisce solo i campi dell'header, utile per verificare la data di modifica di un file.

-POST, serve per inviare informazioni al server e in questo caso l'URI indica cosa si sta inviando.

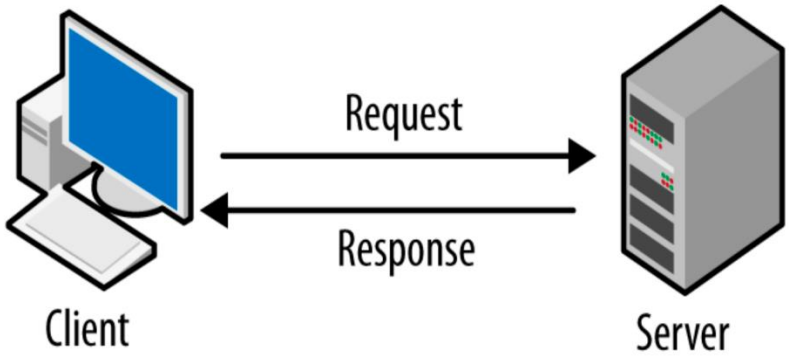


Figura 7.2: Architettura client-server.

La risposta ottenuta è di tipo testuale e, oltre al contenuto (body), presenta anche una riga di stato. In questa riga di stato è presente un codice di tre cifre, di cui i più frequenti sono quelli aventi la prima cifra 2, ad indicare che la richiesta è stata soddisfatta, e 4, quando la richiesta è sbagliata e non può essere soddisfatta. I codici di stato HTTP più famosi sono 200 'OK', che sta a significare la corretta fruizione del contenuto nella sezione body, e 404 'Not Found', che incorre quando la risorsa richiesta non è stata trovata.

7.3 Formato JSON

La risposta ottenuta dal servizio Web utilizzato in questo progetto è di tipo JSON, che sta per JavaScript Object Notation. Questo formato di testo è adatto allo scambio di dati in un'architettura di tipo client/server, essendo di facile lettura e scrittura per gli utenti e, per quanto riguarda il lato macchina, semplice da generare e analizzare. È proprio la sua semplicità ed il suo linguaggio basato in JavaScript che ne hanno decretato la sua rapida diffusione.

La struttura dei dati in JSON si basa sugli *object* (oggetti), set non ordinati di coppie chiave/valore, racchiusi tra parentesi graffe {}. I tipi di dati supportati sono booleani (`true` e `false`), interi, stringhe (racchiuse da doppi apici) e array, ovvero sequenze ordinate di valori separati da virgola (,) e racchiusi tra parentesi quadre.

7.4 Servizio Web

Come affermato più volte, il ruolo dei sensori nello scenario dove si colloca l'applicazione è simulato. Questa simulazione è stata effettuata attraverso un servizio Web, utilizzando i server dell'Università Politecnica delle Marche, che fornisce come risposta un oggetto in formato JSON. Sono stati realizzati tre ipotetici sensori differenti, ognuno dei quali avente un *id* identificativo.

7.4.1 Sensore temperatura

Questo sensore fornisce dati riguardanti l'impianto di climatizzazione di una stanza. La chiamata al servizio Web che lo simula è effettuata tramite l'URL riportato di seguito:

https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=1

La risposta ottenuta in formato JSON è riportata in Figura 7.3, realizzata tramite JSON Viewer[17].

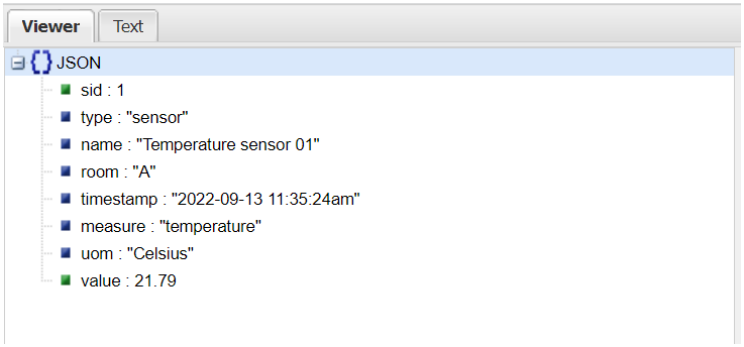


Figura 7.3: Oggetto JSON ottenuto dal sensore temperatura.

Come si vede dalla figura, la risposta consiste in un oggetto JSON (racchiuso tra parentesi graffe) con una serie di coppie chiave/valore che sono:

- *"sid"*: codice identificativo del sensore.
- *"type"*: fa riferimento al tipo del dispositivo, che può essere un sensore o il macchinario di cui esso raccoglie i dati.
- *"name"*: nome del sensore.

17 http://jsonviewer.stack.hu/

- *"room"*: stanza dove si trova il sensore e di conseguenza anche l'operatore che ne sta leggendo i dati, poiché, come si vedrà, i dati verranno forniti direttamente nel dispositivo mobile dell'utente tramite il riconoscimento, mediante fotocamera, di un *target* posto sul macchinario di cui il sensore misura lo stato.
- *"timestamp"*: data e ora correnti.
- *"measure"*: si riferisce all'oggetto della misura, in questo caso la temperatura.
- *"uom"*: unità di misura del dato fornito, che in questo caso è Celsius.
- *"value"*: valore fornito.

7.4.2 Sensore stampante

Questo sensore fornisce dati riguardanti lo stato di un'ipotetica stampante, fornendo come dato il livello di inchiostro presente nel macchinario ed eventuali messaggi d'errore se vengono riscontrati problemi. La chiamata al servizio Web si effettua tramite il seguente URL:

https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=3

In Figura è mostrata la risposta ottenuta, allo stesso modo del caso precedente.

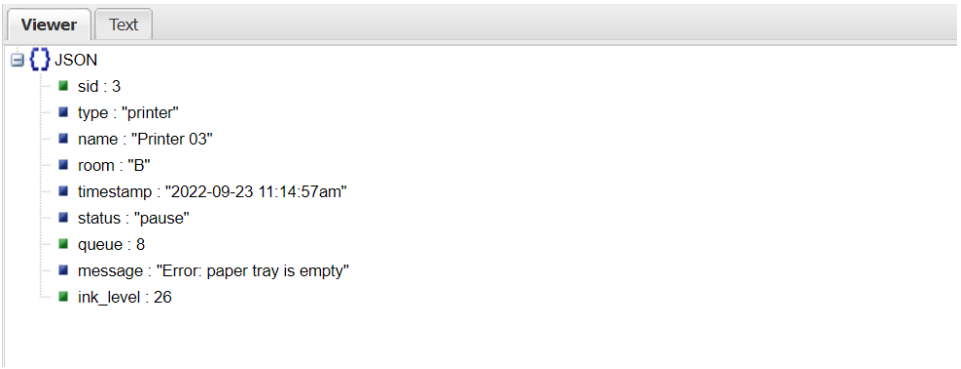


Figura 7.4: Oggetto JSON ottenuto dal sensore stampante.

Anche qui la risposta ottenuta è, come si vede in Figura, un oggetto JSON. Alcune informazioni hanno lo stesso significato del caso precedente e per questo non verranno descritte ulteriormente. Di rilevante si possono osservare:

- *"status"*: questo dato indica lo stato della stampante, che può essere *idle*, quando la stampante è pronta all'uso, *busy*, quando la stampante è occupata, ma funziona correttamente, o *pause*, quando la stampante ha riscontrato qualche problema ed è in stato di pausa.
- *"queue"*: indica il numero di stampe in coda. Questo dato sarà 0 quando lo stato del macchinario è *idle*, mentre sarà un numero random da 1 a 15 nel caso degli altri due stati.
- *"message"*: un messaggio viene riportato quando la stampante riscontra dei problemi, dunque nel caso in cui il suo stato riporta il valore *pause*. I messaggi che si possono incontrare sono: *"Error: paper jam. Please contact the administrator"*, *"Error: paper tray is empty"*, "*Warning: internal error*". Dal contenuto di questi messaggi, come si vedrà in seguito, varierà il comportamento dell'oggetto virtuale visualizzato.
- *"ink_level"*: questo valore indica il livello di inchiostro contenuto nella stampante ed assume un valore intero random da 5 a 200.

7.4.3 Sensore macchinario

Questo sensore è pensato essere posto all'interno di un macchinario industriale. Fornirà all'operatore un codice identificativo del prodotto che la linea di produzione deve realizzare. In Figura 7.5 è mostrato l'oggetto JSON fornito dal servizio Web.

Figura 7.5: Oggetto JSON ottenuto dal sensore temperatura.

Di rilevante, in questo caso, è da considerare il dato *"code"* che può assumere due valori random *A* e *B* che simboleggiano i codici prodotto della linea di produzione.

7.5 Metodo *fetch()*

La Web-app che si vuole realizzare in questo progetto, poiché deve interfacciarsi con servizi Web per la fruizione di dati in rete, svolge il ruolo di client in un'architettura client/server. Per richiedere dati ad un server tramite JavaScript esistono diversi metodi, ma il più moderno e versatile è considerato essere il metodo *fetch()* [46], usato anche per la realizzazione della suddetta applicazione.

Il metodo fetch() riceve come argomento mandatorio il percorso della risorsa di rete dalla quale si vuole attingere dati, processo che è chiamato, appunto, *fetching* dei dati. Ritorna una *promise*, letteralmente una promessa che la richiesta effettuata verrà risolta. Questo rende il metodo fetch() adatto alla programmazione asincrona, in quanto l'oggetto promise rappresenta l'eventuale completamento o fallimento di un'operazione asincrona, che in questo caso è la raccolta dei dati di un servizio Web. Una promessa è infatti un proxy[18] di un dato che non è ancora necessariamente conosciuto quando la promessa è stata creata. Questo permette a un metodo asincrono di ritornare valori come un metodo sincrono: invece di ritornare immediatamente il valore finale, il metodo asincrono ritorna una promessa di fornire tale valore a un certo punto in futuro.

Una promessa può avere uno dei seguenti stati:

- *pending*: stato iniziale, né soddisfatto né rifiutato.
- *fulfilled*: quando l'operazione è stata completata con successo.
- *rejected*: quando l'operazione è fallita.

In questo modo anche se il server restituisce uno stato di errore HTTP, la richiesta viene comunque eseguita.

Ottenuta la risposta, l'oggetto che si riceve non sarà direttamente in formato JSON, come si desidera, ma è una rappresentazione dell'intera risposta HTTP. Perciò, per estrarre il corpo JSON della risposta, si utilizza il metodo *json()*, che ritorna a sua volta una promessa, che si risolve con il risultato dell'analisi della corpo del testo della risposta come JSON.

Lo script *fetch-sensor.js* creato per l'applicazione Web di questo progetto svolge proprio questa funzione. Come si vede in figura N.3, il metodo fetch() è utilizzato

[18] Server che memorizza una copia locale degli elementi Web richiesti per poterli fornire nuovamente senza dover accedere di nuovo al server

all'interno della registrazione di una nuova componente in A-Frame, sfruttando la compatibilità di questo framework con il linguaggio JavaScript e contiene come parametro il percorso della risorsa Web che si vuole analizzare.

```
AFRAME.registerComponent('sensor1', {
  init: function () {
    this.el.addEventListener('markerFound', () => {

      fetch("https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.
php?sid=1")
        .then((response) => response.json())
```

Figura 7.6 Metodo fetch in ARSupport-office.

La componente qui creata è chiamata 'sensor1' e verrà registrata come attributo in una componente del file index.html (ulteriori dettagli nel Capitolo 9 - Funzionalità dell'applicazione). Il metodo fetch() viene chiamato all'interno di un *event-listener*[19], ovvero solamente quando l'evento `'markerFound'` si verifica, e la risposta ottenuta verrà convertita in formato JSON. Il vantaggio di questo metodo sta nel fatto che i dati vengono ottenuti e, in seguito, mostrati direttamente sul dispositivo con cui l'utente sta utilizzando l'applicazione, ogni volta che egli scansiona il marker. Tutto ciò non sarebbe possibile senza la tecnologia della Realtà Aumentata. È grazie ad AR.js infatti che i marker, o comunque i target in generale, vengono riconosciuti ed è possibile quindi utilizzarli come listener e determinare il comportamento dell'applicazione.

[19] Metodo che si riferisce letteralmente a 'ascoltatore di eventi'. Il listener è quella funzione che risponde a un determinato evento (event), che può essere ad esempio un click dell'utente.

Capitolo 8

Funzionalità dell'applicazione

In questa sezione si mostrerà come i requisiti, visti nel Capitolo 5 – Analisi dei requisiti, vengono soddisfatti dall'applicativo e quali funzionalità esso svolge. Per soddisfare ogni requisito si è scelto di presentare tre diversi casi studio che coprono diversi contesti dove l'applicazione può essere utilizzata, non solo il contesto industriale. Ad ogni caso studio è assegnato un diverso titolo dell'applicativo come segue:

- *ARSupport-office* dove l'applicazione sfrutta la Realtà Aumentata in un contesto d'ufficio.
- *ARSupport-product* dove la Realtà Aumentata aiuta l'utente ad utilizzare un certo macchinario industriale, sovrapponendo contenuti virtuali ad esso.
- *ARSupport-engine* dove si va ad aumentare un'immagine 2D di un manuale, che può essere usato anche in un contesto industriale.

Si capirà che in realtà *ARSupport* non è una semplice Web-app, ma più che altro un approccio che può essere adottato in un'impresa digitalizzata per sfruttare al meglio le funzionalità offerte dalla tecnologia della Realtà Aumentata.

Verrà posta attenzione anche per quanto riguarda la progettazione grafica. Nonostante in un'applicazione di Realtà Aumentata l'interfaccia grafica (dall'inglese *Graphical User Interface*, GUI) non sia di particolare rilevanza in quanto il soggetto principale di ogni schermata dell'applicazione è per lo più l'ambiente esterno circostante, arricchito da alcuni contenuti 3D, essa è comunque una componente fondamentale per ogni applicativo. I dati raccolti dai sensori con i quali l'app si interfaccia devono essere mostrati all'utente, in maniera chiara e semplice in modo che egli non si trovi in difficoltà nel leggerli. Un design confuso e poco *userfriendly* allontanerebbe l'utente dall'utilizzo dell'applicazione,

per questo la parte grafica va progettata con cura, tanto quanto quella funzionale, seppur minimale in questo caso.

In questo capitolo si mostrerà e si spiegherà la grafica realizzata per i tre casi di studio presentati in questo elaborato. Verranno inseriti degli screenshot effettuati da smartphone, mentre l'applicazione è in funzione, che sono stati realizzati per chiarire al lettore cosa si vuole ottenere con l'applicativo. Non sono scenari di reale utilizzo dell'applicazione, ma per lo più un chiaro esempio di cosa si è in grado di fare con essa.

Non avendo avuto un reale contatto con una realtà industriale, ma, come affermato più volte, soltanto ipotetico, gli scenari tenuti in considerazione si basano sui modelli 3D che si sono potuti reperire gratuitamente nel Web, riadattandoli poi al contesto dell'applicativo. Il progetto in questione non si pone come obiettivo quello di lavorare con contenuti virtuali realizzati a regola d'arte e di grafica elevata, ma vuole soltanto enfatizzare la potenzialità della Realtà Aumentata affiancata ad attività lavorative che si possono incontrare in un'industria 5.0.

Il lettore interessato a testare le funzionalità dell'app può trovarle, rispettivamente, ai seguenti link:

1)https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-office
2)https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-product
3)https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-engine

8.1 Schermata iniziale

All'utente, nel momento in cui accede all'applicazione, viene immediatamente richiesto se concedere i permessi per l'utilizzo della fotocamera, senza la quale non avrebbe senso utilizzare l'app. La schermata iniziale che vede l'utente è una schermata di default realizzata dagli sviluppatori di AR.js, che si presenta come in Figura 9.1 ed è identica per ogni caso di studio.

Figura 8.1: Schermata iniziale con richiesta dei permessi.

Nella grafica sono mostrati due bottoni la cui funzione è facilmente intuibile grazie al testo che essi contengono. L'utente non deve far altro che concedere all'applicazione l'accesso alla fotocamera, cliccando sul bottone "Allow".

8.2 Simulazione scenario

L'applicativo realizzato è stato testato in un ambiente che simula una situazione di smart factory, non avendo accesso a un contesto industriale vero e proprio. In particolare, si è cercata una soluzione per ottenere risposte da ipotetici sensori, non esistenti realmente.

È stato realizzato un file chiamato *query.php* (il cui codice è mostrato in Appendice C – Codici) per ottenere un oggetto JSON (si veda il Paragrafo 7.3), che simulasse la risposta di alcuni ipotetici sensori. Al file si accede tramite un servizio Web effettuando la seguente chiamata:
https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php

Esistono in tutto quattro diverse chiamate che corrispondono a quattro diversi sensori:
https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=1
https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=2
https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=3
https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=4

Dalla chiamata stessa, tramite il metodo GET di HTTP, il file *query.php* ricava l'id del sensore e in base a questo genera la risposta JSON, che viene restituita al servizio Web e mostrata all'utente.

8.3 Importazione AR.js e A-Frame

La Web-app realizzata si presenta come una qualsiasi pagina Web: un file HTML che gestisce la struttura della pagina, uno o più file CSS per lo stile e la grafica e uno o più script JavaScript per la logica. Si possono anche scrivere i diversi file all'interno di un unico file HTML, ma questo potrebbe generare disordine e confusione; perciò, in quest'applicazione è presente un file *index.html* in cui vengono importati gli eventuali file necessari.

Per poter utilizzare le funzioni di AR.js ed A-Frame nella propria applicazione Web, occorre, infatti, importare la libreria opportuna, scritta in JavaScript, come suggerisce la documentazione ufficiale di AR.js [32]. In Figura 8.2 è mostrato questo procedimento, come appare nel file *index.html* dell'app.

```
<script
src="https://raw.githack.com/AR-js-
org/AR.js/master/aframe/build/aframe-ar.js">
</script>
```

Figura 8.2: Importazione della libreria AR.js per permettere all'applicazione di svolgere funzioni marker-based.

Questo script consente di utilizzare sia le funzioni di AR.js sia quelle di A-Frame. Senza questo passaggio non sarebbe possibile riconoscere i marker né far visualizzare contenuti virtuali tramite A-Frame.

Per quanto riguarda la funzione di riconoscimento *target* tramite Natural Feature Tracking (discusso nel Paragrafo 4.4), l'importazione da effettuare è mostrata in Figura 8.3. Grazie a questa libreria si riescono ad implementare funzioni di riconoscimento immagini, oltre che l'estensione A-Frame come sopra.

```
<script
src="https://raw.githack.com/AR-js-
org/AR.js/master/aframe/build/aframe-ar-nft.js">
</script>
```

Figura 8.3: Importazione della libreria AR.js per permettere all'applicazione di svolgere funzioni di Natural Feature Tracking.

8.4 Caso di studio 1: ARSupport-office

Nel primo caso di studio presentato, il contesto ipotizzato è un contesto d'ufficio, dove un operatore sfrutta la Realtà Aumentata tramite l'approccio *marker-based* menzionato in precedenza. L'idea è quella di predisporre il marker sul dispositivo con cui l'utente ha bisogno di interagire di modo che, scansionandolo tramite la Web-app, egli riceve informazioni riguardanti il dispositivo.

Il lettore che volesse testare può utilizzare il link, già inserito in precedenza, qui riportato per comodità:
https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-office

Come si vedrà nel prossimo paragrafo, è possibile anche scannerizzare un QRcode per usufruire direttamente dell'applicazione.

8.4.1 Marker

In questo caso di studio sono stati utilizzati due marker, mostrati in Figura 8.4, il cui file `.patt` è stato realizzato tramite un tool online, realizzato dallo stesso fondatore di AR.js, Jerome Etienne, consultabile in [67].

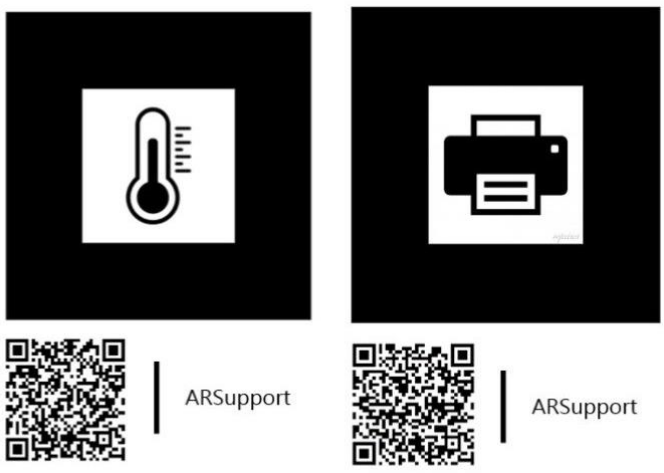


Figura 8.4: Marker e qrcode utilizzati in ARSupport-office.

Come si vede in figura i marker utilizzati sono evocativi dello strumento con la quale l'utente deve interagire. In particolare, il marker che raffigura un termometro fa riferimento a un ipotetico sensore di temperatura, mentre l'altro marker raffigura una stampante. È stato realizzato anche un qrcode ad ogni marker, per favorire l'accessibilità dell'applicazione, in quanto l'utente, semplicemente aprendo la fotocamera (almeno per i telefoni più moderni), sarà indirizzato al link della Web-app e, dopo aver concesso i permessi, potrà visualizzare il contenuto virtuale semplicemente scansionando il marker.

Per fare in modo che il marker venga riconosciuto dall'applicazione, si è precaricato il suo file .patt nella repository di GitHub per poi settare l'attributo `url` del tag `<a-marker>` di A-Frame con l'indirizzo di questo file, come si vede in Figura 8.5. Gli attributi `type` e `pattern` sono settati in modo da poter utilizzare un marker personalizzato, diverso da quelli di default disponibili in AR.js. All'interno di questo tag si setta anche la componente sensor, creata nello script *fetch-sensor.js*, oggetto di discussione della Sezione 7.5.

```
<a-marker
      preset='custom'
      type='pattern'
      url="https://raw.githubusercontent.com/Bbbgl/ARsupport.github.
io/main/ARsupport/assets/markers/pattern-print_icon.patt"
      raycaster="objects: .clickable"
      emitevents ="true"
      sensor
    >
```

Figura 8.5: Frammento di codice nel file index.html di ARSupport-office per evidenziare gli attributi del tag <a-marker>.

In questo modo la chiamata al servizio Web avviene semplicemente visualizzando il marker tramite fotocamera del dispositivo ed i dati ottenuti vengono mostrati nella tabella corrispondente.

All'interno dello script *fetch-sensor.js* è presente anche un *listener* all'evento `'markerLost'`. Questo procedimento fa sì che, una volta persa la visuale del marker, i dati presenti nella tabella si resettano.

```
this.el.addEventListener('markerLost', () => {

        var element = document.querySelector('#entity')
        element.setAttribute('animation-mixer',"clip:Static Pose")
        document.getElementById("myDevice").innerHTML = "N/A";
        document.getElementById("myType").innerHTML = "N/A";
        document.getElementById("myName").innerHTML = "N/A";
        document.getElementById("myRoom").innerHTML = "N/A";
        document.getElementById("myTime").innerHTML = "N/A";
        document.getElementById("myMeasure").innerHTML = "N/A";
        document.getElementById("myMeasure2").innerHTML = "N/A";
        document.getElementById("myMeasure3").innerHTML = "N/A";
        document.getElementById("myInkLevel").innerHTML = "N/A";
        document.getElementById("myInstruction").innerHTML = "";

      })
    }
  });
```

Figura 8.6: Frammento di codice nel file fetch-sensor.js.

Come si vede dalla Figura 8.6, alla perdita della visuale sul marker, oltre che a resettare i valori nella tabella, viene anche reimpostato in contenuto virtuale (in questo caso la stampante) settando il suo attributo `'animation-mixer'`. Il valore di questo attributo fa riferimento al modello 3D utilizzato e varia da modello a modello, a piacere del suo autore.

8.4.2 Chiamata al servizio Web

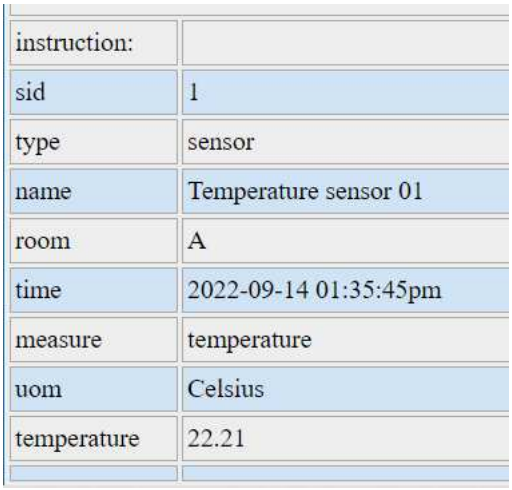
ARSupport-office, tramite il procedimento illustrato nel Capitolo 7 – Struttura dell'applicazione, effettua la chiamata al servizio Web per ottenere informazioni riguardanti lo stato del macchinario. In questo caso di studio vengono utilizzati i sensori temperatura e stampante visti nel suddetto capitolo, i cui URL per effettuare la chiamata al servizio Web sono riportati, rispettivamente, di seguito:

https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=1

https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=3

La risposta che ritorna è, in entrambi i casi, un oggetto JSON, il cui contenuto andrà a riempire la tabella mostrata in Figura 8.7, soddisfando così i requisiti di interazione con i sensori e visualizzazione dati.

La tabella in cui i dati provenienti dal servizio Web vengono raccolti è stata realizzata in HTML e CSS. Il suo corpo, con le righe e le colonne, e il testo che definisce ogni tipo di dato è presente nel file `index.html` mentre il colore e lo stile della tabella sono definiti nel file `tablestyle.css`.



instruction:	
sid	1
type	sensor
name	Temperature sensor 01
room	A
time	2022-09-14 01:35:45pm
measure	temperature
uom	Celsius
temperature	22.21

Figura 8.7: Tabella per raccolta dati, come appare dopo che l'applicazione ha effettuato la chiamata al servizio Web, nel caso del sensore temperatura.

Nel caso del sensore di temperatura, mostrata in Figura 8.7, non c'è interazione tra utente e interfaccia grafica, lo scopo della tabella è soltanto mostrare i dati e visualizzare un messaggio che guida l'utente nell'operazione da eseguire. Per quanto riguarda la tabella che si ottiene quando si interagisce con il sensore stampante si osservi la Figura 8.8.

instruction:	
sid	3
type	printer
name	Printer 03
room	B
time	2022-09-26 11:50:54am
status	pause
queue	11
message	Error: paper tray is empty
ink level	147

Figura 8.8: Tabella per raccolta dati, come appare dopo che l'applicazione ha effettuato la chiamata al servizio Web, nel caso del sensore stampante.

Da tenere a mente che i sensori, non trovandosi a lavorare in un vero contesto industriale, sono stati realizzati mediante il servizio Web discusso in precedenza, che restituisce una risposta random per alcuni valori ad ogni chiamata. Questo è stato pensato per poter sfruttare al meglio le potenzialità dell'approccio di questo applicativo. Un sensore reale difficilmente restituirebbe risposte diverse ad ogni chiamata.
L'applicazione riesce qui a far visualizzare i dati relativi a un sensore legato ad un certo macchinario direttamente sul dispositivo, smartphone o tablet, dell'operatore.

8.4.3 Visualizzazione contenuti virtuali

Un altro requisito funzionale che l'applicazione deve soddisfare è quello di far visualizzare all'utente un contenuto 3D, anche per definizione della stessa Realtà Aumentata.

Come accennato in precedenza, non avendo avuto un diretto contatto con un'impresa per questo progetto, non si sono potuti reperire modelli 3D che riguardassero un determinato caso di studio e che fossero realizzati a regola d'arte. Perciò, si è scelto di ripiegare su modelli scaricabili online[20].

[20] Per questo progetto, i modelli 3D utilizzati sono stati scaricati da Sketchfab [https://sketchfab.com/]

In Figura 8.9, sono mostrati degli *screenshot* dell'app in funzione mentre vengono mostrati i contenuti virtuali utilizzati. Da notare come anche i dati presenti nella tabella siano aggiornati.

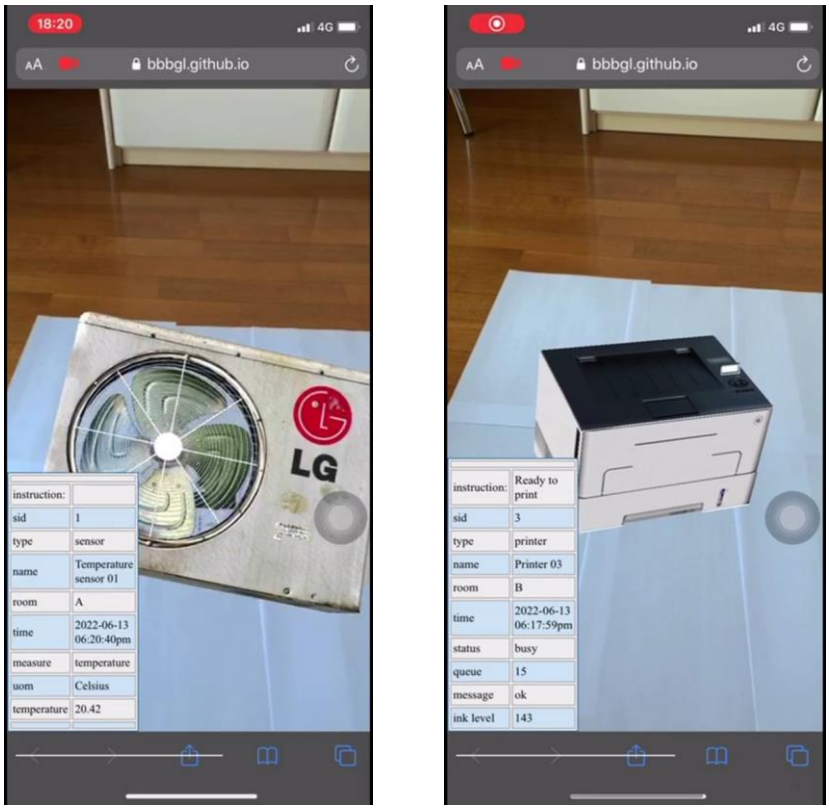


Figura 8.9: A sinistra: modello 3D di un condizionatore e tabella dei dati aggiornata relativa a un sensore di temperatura. A destra: modello 3D di una stampante e tabella dei dati aggiornata relativa a un sensore posto nella stampante stessa.

Per far visualizzare il contenuto si è lavorato con l'elemento `<a-entity>` di A-Frame, andando a settare il suo attributo `gltf-model`, come si vede in Figura 8.10. I modelli caricati, in questo caso, in formato glTF compatibile con A-Frame, trattato nella sezione 4.1.5.

```
 <a-entity
        id="entity"
  gltf-model=
"https://raw.githubusercontent.com/Bbbgl/ARsupport.github.io/main/AR
support/assets/devices/animated_engine/scene.gltf"
        position="0.80293 -0.01835 -0.11365"
        scale="1 1 1"
        rotation="-21.958034540593673 -94.13295503542834
66.51123268996649"
        class="clickable"
        clicker
        gesture-handler
        animation-mixer="clip:Take 001"

      >
      </a-entity>
```

Figura 8.10: Frammento di codice nel file index.html di ARSupport-office, dove viene messo in evidenza l'elemento `<a-entity>` *che mostra il contenuto virtuale, in questo caso una stampante.*

8.4.4 Interazione con il contenuto virtuale

Per soddisfare il requisito di interazione con il contenuto virtuale si è dovuto innanzi tutto rendere l'applicazione rispondibile all'evento *touch*. Per far questo si sono importati due script, *gesture-detector.js* e *gesture-handler.js*, creati da Fabio Cortes in [59]. Il primo script registra una componente A-Frame, chiamata `gesture-detector`, tramite lo stesso approccio visto per la chiamata dei sensori. La componente viene settata nel tag `<a-scene>`, così da far diventare la scena sensibile al tocco dell'utente. Grazie al secondo script, registrando la componente `gesture-handler` e settandola come attributo dell'elemento entity, come si vede in Figura 8.10, l'utente diventa abile a ruotare e zoommare a piacimento il contenuto virtuale.

Lo script *clickers.js* è stato realizzato per questo progetto e permette di interagire con il contenuto virtuale semplicemente tramite *touch*. Questo script realizza una componente A-Frame, allo stesso modo dei precedenti, chiamata `clicker`, visibile anche in Figura 8.10. Scopo di questa funzione è guidare l'operatore, animando il contenuto virtuale in base ai dati ottenuti dai sensori. Ad esempio, se il messaggio ricevuto in *'message'* è *"Error: paper tray is empty"* l'utente, cliccando

sul contenuto virtuale, vedrà aprirsi il cassetto della carta, intuendo quindi che è lì dove deve agire (si veda Figura 8.11).

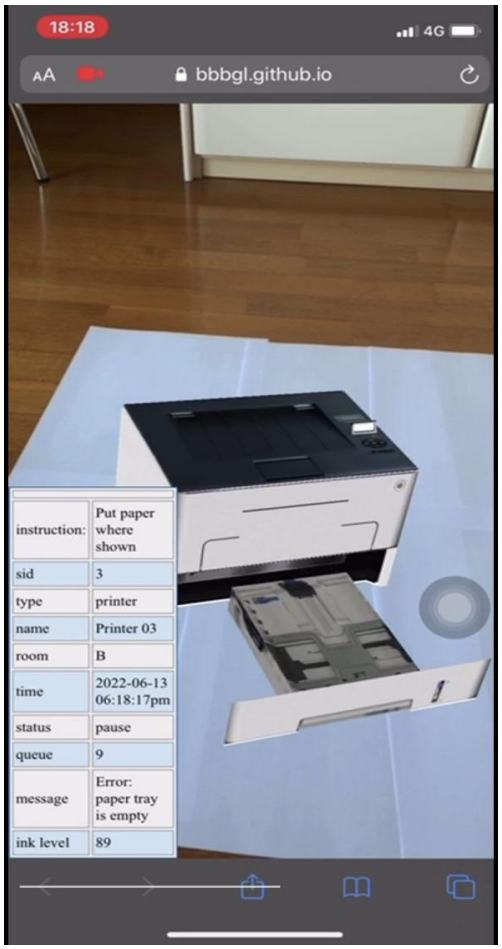


Figura 8.11: Contenuto virtuale che interagisce al click dell'utente a seconda del valore in "message".

In questo modo il requisito di interazione con il contenuto virtuale è soddisfatto, come anche la funzionalità del contenuto virtuale stesso, il quale andrà a guidare l'operazione da svolgere in base alla lettura dei dati provenienti dal servizio Web. L'usabilità e l'accessibilità sono elevate in quanto l'utente, per usufruire di tutte le funzionalità dell'app, non deve fare altro che accendere la videocamera del proprio dispositivo e interagire con il contenuto virtuale tramite *touch*.

8.5 Caso di studio 2: ARSupport-product

Nel caso di studio visto in precedenza, sebbene i requisiti delle specifiche vengano tutti soddisfatti, si è notata la mancanza della Realtà Aumentata nel senso proprio del termine. Il contenuto virtuale infatti non andava, appunto, ad 'aumentare' la realtà vista dall'operatore attraverso il suo dispositivo, ma consisteva per lo più in un modello 3D del macchinario con il quale si sta lavorando, con il quale l'utente può interagire e ricevere informazioni.

Il secondo caso di studio presentato si pone l'obiettivo di risanare questa mancanza, andando a sovrapporre effettivamente un contenuto virtuale su una scena reale. *ARSupport-product* entra in un più verosimile contesto industriale, lavorando con veri macchinari industriali e codici prodotto che si possono incontrare in una linea di produzione.

Si è ipotizzato di dover lavorare con un macchinario che richiede del materiale avvolto attorno a una struttura cilindrica. Questo materiale viene simulato dal contenuto virtuale, che si sovrappone nel punto del macchinario dove deve essere installato. A seconda del codice prodotto ricavato dalla lettura dei sensori, il materiale necessario alla produzione potrebbe essere differente. L'utente, quindi, viene guidato nella selezione del materiale interagendo con il contenuto virtuale, che cambia aspetto (in questo caso colore) a seconda del codice prodotto corrente. Un'indicazione aggiuntiva è anche simulata da una freccia virtuale che indica il verso di installazione dell'oggetto.

Il lettore che volesse testare può utilizzare il link, già inserito in precedenza, qui riportato per comodità:
https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-product

8.5.1 Natural Feature Tracking

Si è utilizzato, in questo caso di studio, un approccio *markerless*, ovvero basato sul metodo di *Natural-Feature-Tracking* discusso nel Paragrafo 4.3. In Figura 8.12 viene mostrata la sezione del macchinario utilizzata come *target* per azionare il sistema di Realtà Aumentata dell'applicativo. Questo approccio è stato utilizzato sia con l'oggetto reale si con un'immagine 2D dello stesso.

Il contenuto virtuale che apparirà è stato posizionato e ruotato accuratamente tramite lo strumento *Visual 3D Inspector* fornito dal framework A-Frame. Questo strumento consente di regolare le coordinate del contenuto 3D per poi settare i rispettivi attributi nell'elemento corrispondente in HTML.

Figura 8.12: Sezione di un macchinario industriale, utilizzata come target per l'applicazione.

8.5.2 Chiamata al servizio Web

Viene utilizzato anche qui lo stesso approccio discusso nel precedente caso di studio per richiedere informazioni al servizio Web. L'URL della chiamata è qui mostrato: https://kdmg.dii.univpm.it/iot/mobile/ar/example/query.php?sid=4 .

Si vede come la risposta che si ottiene è più essenziale, degno di nota è il dato *"code"* che sarà fondamentale per il comportamento del contenuto virtuale, come si vedrà nel prossimo paragrafo.

È stato utilizzato lo stesso script *fetch-sensor.js* del caso di studio precedente; in questo modo la chiamata avviene quando l'utente visualizza con la fotocamera del dispositivo la sezione del macchinario.

8.5.3 Interazione con il contenuto virtuale

I contenuti virtuali vengono mostrati allo stesso modo del caso di studio precedente, per questo non si aggiunge niente di nuovo. L'unica differenza sta nel tag di A-Frame in quanto, mentre prima si utilizzava il tag `<a-marker>`, qui è utilizzato il tag `<a-nft>`.

Di diverso approccio è invece l'interazione con il contenuto. Essendo l'oggetto 3D la rappresentazione virtuale del materiale che deve essere installato nella stessa posizione dove compare nel dispositivo dell'utente, si è preferito evitare di interagire con esso tramite *touch* per non rischiare di spostarlo dalla sua posizione ideale. Piuttosto, è stato aggiunto un bottone, in HTML, come mostrato in Figura 8.13. Cliccando sul bottone il contenuto virtuale cambierà di colore e di orientazione, indicando all'utente cosa e come dovrà installare nel macchinario per procedere alla produzione. I risultati ottenuti sono visibili in Figura 8.14.

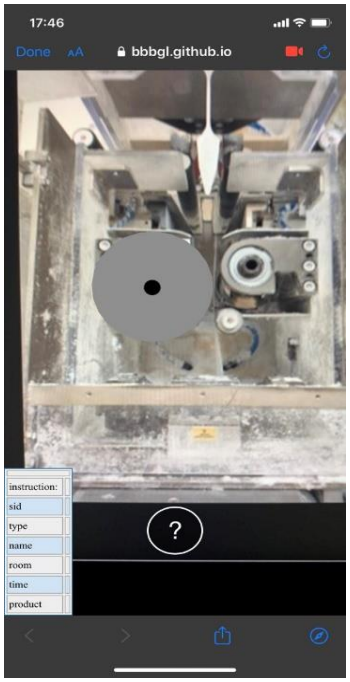


Figura 8.13: Stato iniziale del contenuto 3D predisposto sul macchinario.

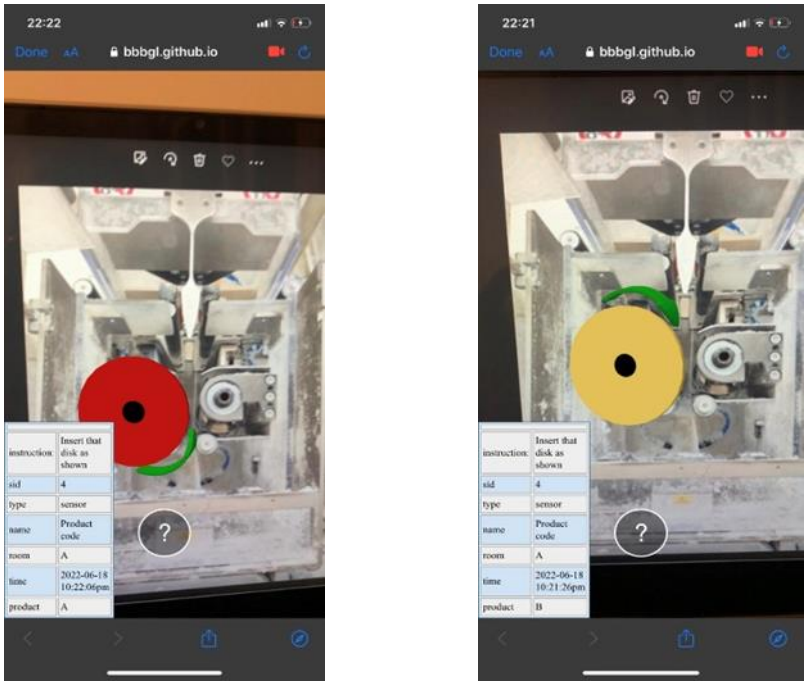


Figura 8.14: A sinistra: cambiamento del contenuto al click del bottone con il codice prodotto "A". A destra cambiamento del contenuto al click del bottone con il codice prodotto "B".

Le prestazioni ottenute in questo caso di studio sono inferiori rispetto al primo caso presentato. L'approccio *markerbased*, almeno per quanto visto con AR.js, consente di visualizzare contenuti virtuali in maniera più reattiva e più precisa, per quanto riguarda la loro posizione, che con l'approccio *natural-feature-tracking*. Inoltre, avendo testato quest'app sia sul macchinario reale, sia sull'immagine 2D dell'oggetto stesso, si è visto come il secondo caso sia di gran lunga più affidabile rispetto al primo, sebbene carente di elevata utilità.

8.6 Caso di studio 3: ARSupport-engine

L'ultimo caso di studio presentato si discosta leggermente dai precedenti in quanto non presenta l'interazione con un database esterno. Si è scelto di presentare anche questa parte di progetto considerando una situazione che può essere presente in svariati contesti, compreso anche quello industriale. *ARSupport-engine* ha come scopo quello di 'aumentare' un'immagine 2D di un motore, che può essere un motore di un veicolo o un qualsiasi motore energetico, mostrata in Figura 8.15.

L'applicazione vuole quindi mostrare all'utente parti dell'oggetto in questione che non sono visibili, e magari neanche facilmente intuibili, osservando una semplice immagine, seppur esplicativa. L'utente quindi, tramite fotocamera, potrà vedere alcune componenti del motore in questione, nonché l'intero motore nella sua installazione pezzo dopo pezzo.

Il lettore che volesse testare può utilizzare il link, già inserito in precedenza, qui riportato per comodità:
https://github.com/Bbbgl/ARsupport.github.io/tree/main/ARSupport-engine

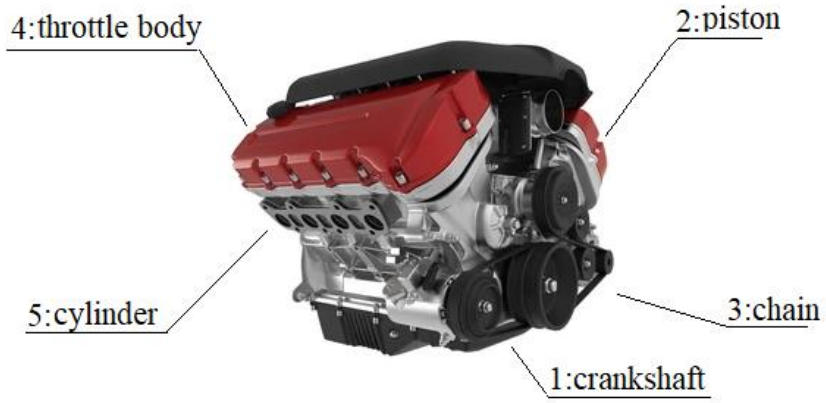


Figura 8.15: Motore con indicazioni delle sue componenti.

8.6.1 Natural Feature Tracking

È evidente che l'approccio utilizzato è di tipo *markerless*, e l'immagine utilizzata per essere 'allenata' è la stessa vista in 4.4, riportata in Figura 8.16 per comodità.

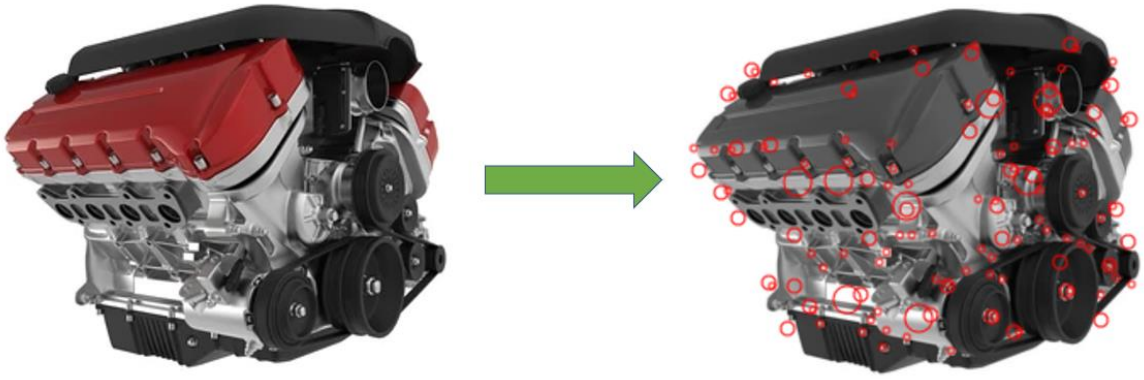


Figura 8.16: Immagine utilizzata nel progetto e sua versione 'allenata' per realizzare un riconoscimento di tipo Natural Feature Tracking.

I file utili al processo di *Natural Feature Tracking* discussi nella Sezione 4.4, sono stati salvati in una cartella nella repository GitHub creata per questo progetto, il cui link andrà a settare l'attributo `url` dell'elemento `<a-nft>` nel file index.html. Questo passaggio è visibile in Figura 8.15.

```
 <a-nft
      type="nft"
      url="https://raw.githubusercontent.com/Bbbgl/ARsupport.github.
io/main/engine_nft/engine"
    >
```

Figura 8.15: Frammento di codice che evidenzia il percorso della cartella contenente i file utili al riconoscimento dei punti caratteristici dell'immagine

8.6.2 Interazione con il contenuto virtuale

Di maggiore rilievo qui, rispetto ai casi di studio precedente, è la parte grafica. In Figura 8.18 viene mostrato il contenuto virtuale visualizzato e si vedono dei pulsanti nella parte inferiore della schermata. Sono stati realizzati, infatti, dei bottoni in codice HTML, che determineranno il comportamento del contenuto virtuale. Il numero presente in ogni bottone fa riferimento ai numeri visibili in Figura 8.15, ai quali corrisponde una determinata componente del motore. È presente anche un bottone 'reset' che mostra il motore durante la sua installazione, contenuto mostrato anche al momento della prima scansione dell'immagine tramite app. In Figura 8.19 si vedono le varie reazioni del contenuto al click dell'utente sui vari pulsanti. In questo modo il lettore può rendersi conto del vantaggio di poter apprezzare una componente del motore in maggior dettaglio rispetto a quanto si avrebbe visualizzando una semplice immagine.

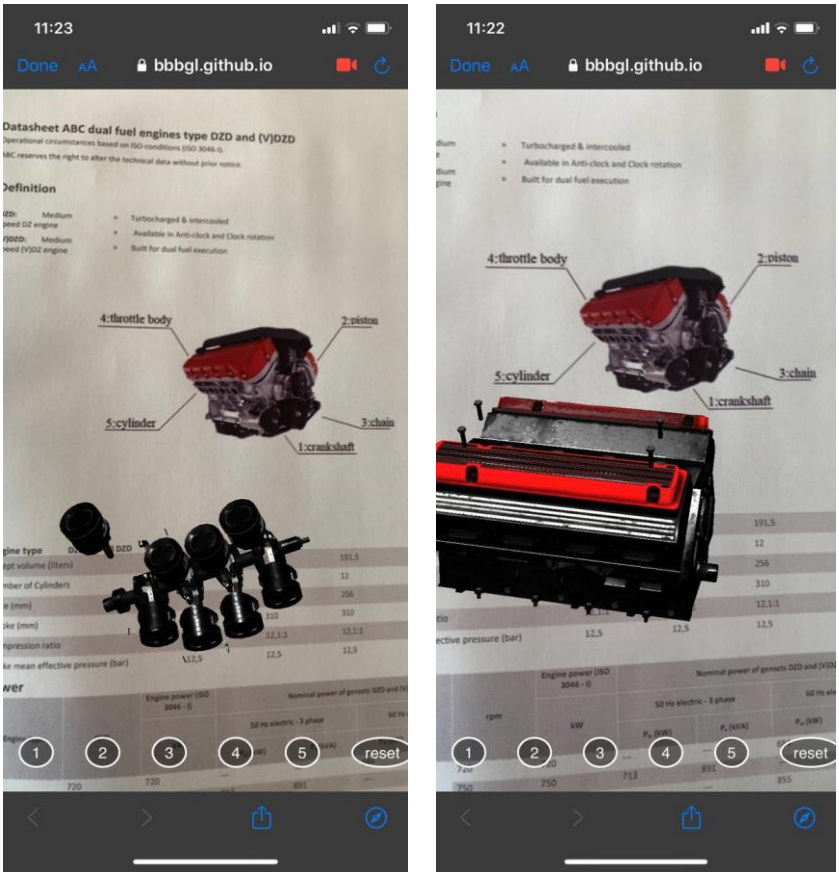


Figura 8.18: Contenuto virtuale visualizzato in ARSupport-engine, in animazione, e pulsanti, in basso, per interagire con esso.

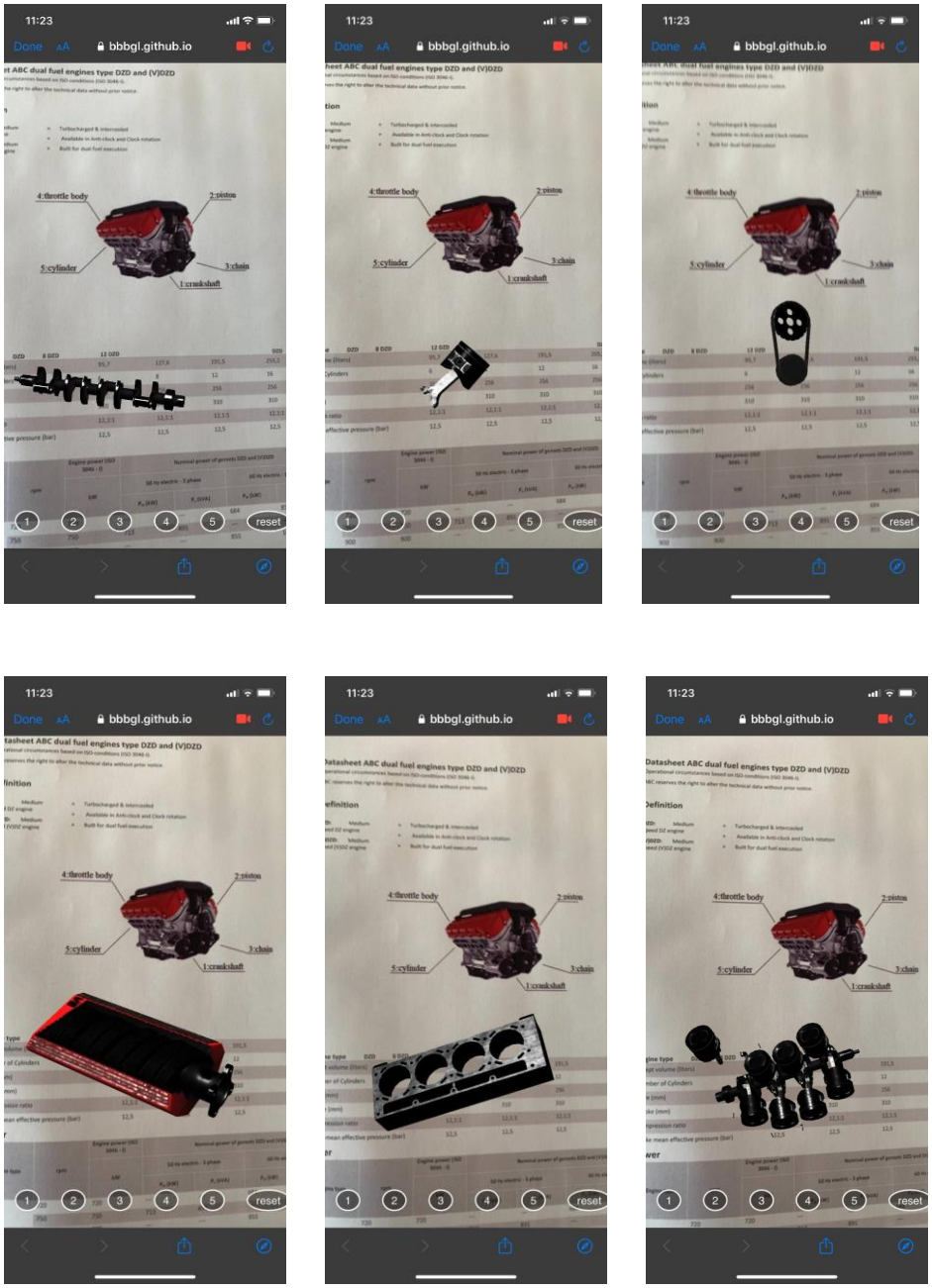


Figura 8.19: Reazione del contenuto virtuale al tocco sui vari pulsanti presenti, in senso orario fino al pulsante 'reset'.

Nel codice, sono stati importati gli *script gesture-handler.js* e *gesture-detector.js* già discussi in precedenza, di modo che l'utente possa quindi interagire con il contenuto visibile, ruotandolo e regolando le sue dimensioni semplicemente tramite *pinch* delle dita.

Il vantaggio di utilizzare questo approccio sta proprio nel poter interagire con le singole componenti dell'oggetto 3D, per apprezzarle nella loro interezza e comprenderne meglio alcuni dettagli che, osservando una semplice immagine 2D, possono risultare nascosti o poco chiari.

8.7 Implementazioni future

Una delle maggiori limitazioni incontrata nella realizzazione di questo progetto è stata quella di non aver avuto accesso a modelli grafici 3D da adattare a un caso di studio specifico. I modelli utilizzati sono stati trovati in rete o realizzati tramite software di modellazione 3D. Per sopperire a questa mancanza il codice, in particolare quello realizzato per *ARSupport-office*, è predisposto per essere riadattato a diversi contenuti virtuali.

L'approccio *marker-based* offre la possibilità di poter lavorare con nuovi oggetti 3D, in quanto basta inserire questi nella repository GitHub e il comportamento dell'applicazione rimane invariato. Se il committente non lo richiede specificatamente, i marker utilizzati possono restare invariati. Si è ancora limitati, invece, per quanto riguarda l'approccio *markerless*, dove si è vincolati a dover mantenere una certa correlazione tra il contenuto reale e quello virtuale. Sarebbe controproducente e poco intuitivo produrre un'applicazione che mostra oggetti 3D su contenuti reali di diversi contesti.

Sviluppo futuro di notevole interesse per questo progetto consiste nell'estendere il suo funzionamento anche su dispositivi VR e *smartglasses* discussi nel Capitolo 2 – Realtà Aumentata. A-Frame, come già affermato nel corrispondente paragrafo, nasce proprio per la creazione di esperienze in Realtà Virtuale, consentendo quindi la possibilità di lavorare anche con i dispositivi sopracitati, oltre che ai semplici smartphone e tablet. AR.js, grazie alla sua continua evoluzione e alla compatibilità con A-Frame, consente l'implementazione per VR. È sufficiente settare il corrispondente attributo nel tag `<a-scene>` e comparirà, nell'interfaccia dell'app, un pulsante che consente di passare alla Realtà Virtuale. Il vantaggio sta nell'ulteriore immersione che l'utente potrà apprezzare se avesse la possibilità di interagire con i suoi ambienti lavorativi tramite visori di Realtà Virtuale [32].

Capitolo 9
Conclusioni

Nel presente elaborato di tesi si è discusso della progettazione e della realizzazione di un'applicazione Web che si inserisce in un contesto di Industria 5.0, sfruttando la tecnologia della Realtà Aumentata.

Dopo aver presentato un'introduzione generale della Realtà Aumentata e dell'Industria 5.0, si è poi presentato il progetto in questione, partendo da un'ipotetica intervista al committente dell'app da cui è derivata la specifica dei requisiti. Da questa si sono delineate le funzionalità che l'applicativo si sarebbe proposto a soddisfare, funzionalità che hanno portato alla scelta della strumentazione discussa di seguito.

Delineando la struttura dell'applicazione e la sua grafica, si sono presentati i 3 casi di studio che mostrano la potenzialità di questo approccio sia in contesti industriali, ma anche in contesti più comuni. *ARSupport-office, ARSupport-product* e *ARSupport-engine*, i nomi attribuiti ad ogni applicativo, soddisfano i requisiti, funzionali e non, che le specifiche richiedevano ed è stato spiegato in che modo lo fanno. Con questo approccio l'utente riesce a ricevere informazioni provenienti da ipotetici sensori, disposti su macchinari o strumenti elettronici, direttamente dal suo dispositivo smarthpone o tablet, per poi poter interagire con contenuti grafici che guidano e illustrano l'utente stesso sull'attività da svolgere, in base alle informazioni ricevute.

L'applicazione possiede le funzionalità necessarie a soddisfare le specifiche richieste. I risultati ottenuti sono accettabili, sia in termini di usabilità che di reattività. L'app, infatti, risponde bene per tutti i casi di studio proposti, interagendo con servizi Web esterni e visualizzando contenuti virtuali sia con dispositivi smartphone che tablet. Essa, infatti, è stata testata nei seguenti dispositivi: iPhone XR (iOS 14), iPad Air 2 (iOS 10) e Huawei P20L (Android 11).
L'unica limitazione sta nel suo uso con oggetti reali, soprattutto per quanto riguarda la posizione degli oggetti 3D. Questi, infatti, difficilmente si sovrappongono all'oggetto reale e sono più soggetti rispetto ad un sistema

marker-based ai movimenti dell'utente, che sono ovviamente frequenti. Il contenuto virtuale tende a spostarsi e a scomparire dallo schermo poiché il sistema perde in continuazione il riferimento con l'immagine da scansionare, dovendo perciò ricalcolare la posa e l'orientazione del modello.

ARSupport in generale rimane un'idea nel suo stato embrionale, i casi di studio proposti sono per lo più esempi per evidenziare le potenzialità della Realtà Aumentata nei moderni dispositivi smartphone. La stessa strumentazione utilizzata, AR.js in particolare, si presta molto bene alle soluzioni proposte, ma fa fatica a fornire un livello di usabilità adatto a contesti reali. L'approccio adottato rimane efficiente, ma può essere migliorato utilizzando framework per realizzare esperienza di Realtà Aumentata più potenti e più moderni.

Appendice A
Altri Framework

In questa appendice si elencheranno alcuni *framework* che svolgono la stessa funzione di *AR.js*, ovvero offrire la possibilità di sviluppare app in Realtà Aumentata nel Web.

Di principale importanza è la distinzione tra software opensource e software con licenza, quindi a pagamento. Nel progetto di questo elaborato di tesi, infatti, si è optato per l'utilizzo di strumenti opensource, volendo risparmiare al massimo le risorse. Per un confronto diretto si devono considerare tutti quei software che appartengono a questa categoria, ma in questa appendice sono stati inseriti comunque dei framework a pagamento per completezza.

A.1 Framework opensource

Di seguito si presenteranno i principali strumenti opensource per realizzare esperienze di Realtà Aumentata nel Web.

A.1.1 Argon.js

Argon.js [60] è un framework dedicato alla Realtà Aumentata e pensato per lo sviluppo di questa nel Web, tanto che nasce per essere utilizzato in un browser apposito, Argon4. Il progetto si è poi ampliato per comprendere anche il funzionamento di tutti i browser. Si basa sulle librerie Three.js (di cui si accennerà più avanti) per il rendering, ma può anche essere integrato in A-Frame per semplificare la creazione di esperienze AR *Web-based*. Non è quindi molto lontano dalle funzioni che offre AR.js, la sua differenza sostanziale sta nelle prestazioni più scarse e nell'installazione del software di sviluppo.

La scarsa documentazione e la poca attività della sua community hanno portato alla decisione di scartare Argon.js per la realizzazione di una Web-app.

A.1.2 WebXR

L'acronimo XR, che non è raro incontrare in letteratura, è qui utilizzato per riferirsi allo spettro di hardware, applicazioni e tecniche utilizzati sia per la Realtà Virtuale che per la Realtà Aumentata. WebXR, proposto dal W3C Consortium, dove partecipano contributori provenienti da Google, Microsoft e Mozilla [61], si pone come obiettivo proprio quello di offrire 'immersività' all'utente, con tutte le opportunità e sfide che ne conseguono.

Sebbene, infatti, interagire direttamente con un hardware immersivo, come ad esempio uno schermo *head-mounted*, possa offrire esperienze allettanti, si deve porre un occhio anche alla sicurezza che concerne queste piattaforme, specialmente sul Web. Questa API pone si pone come obiettivo quello di fornire agli sviluppatori le interfacce necessarie per costruire applicazioni sì immersive, ma anche confortevoli e sicure.

Un vantaggio di WebXR rispetto al discusso AR.js sta nel fatto di poter posizionare oggetti 3D nella scena reale che posso rimanere allocati nella scena. Con AR.js non si riusciva ad ottenere tale risultato, dal momento che la fotocamera perdeva la visuale del target il contenuto spariva. Si è comunque deciso di non utilizzare WebXR poiché questo suo vantaggio non era necessario al progetto di questa tesi, inoltre non c'è compatibilità con il browser di iOS Safari, costringendo gli utenti Apple a dover scaricare un altro browser. Per questo motivo si è preferito AR.js.

A.1.3 Three.js

Se i precedenti framework erano diretti concorrenti di AR.js, *three.js* può essere considerata un'alternativa valida all'utilizzo di A-Frame per la resa grafica, dal momento che AR.js è compatibile con questa libreria.

Three.js è una libreria JavaScript utile per il *rendering* 3D direttamente su Web. È molto versatile e si può utilizzare per molti progetti diversi, da semplici grafici a videogiochi. Lo scopo di questo progetto è quello di avere una libreria grafica 3D facilmente accessibile, adatta ad ogni browser e basata sullo standard WebGL [63].

Ad essere precisi, A-Frame si basa su three.js, che ne è un'astrazione. Mantenendo, quindi, le stesse prestazioni che si otterrebbero lavorando direttamente con three.js, il framework scelto offre una maggior semplicità di sviluppo, motivo per cui si è preferito utilizzarlo per la realizzazione del progetto presentato.

A.2 Framework non opensource

Di seguito saranno presentati i principali framework a pagamento che si sono incontrati e analizzati durante la progettazione dell'applicativo di questo elaborato che, sebbene interessanti, sono stati scartati per la loro natura non opensource.

A.2.1 Vuforia

Vuforia è un kit di sviluppo software (SDK) per la creazione di esperienze di Realtà Aumentata in dispositivi mobile [75]. Usa una tecnologia di *computer vision* per riconoscere immagini 2D o oggetti 3D reali, consentendo agli sviluppatori di posizionare ed orientare contenuti virtuali in relazione all'ambiente reale in cui essi si trovano quando sono visti dallo schermo del device. L'oggetto virtuale viene così scalato in tempo reale a seconda della posizione e della prospettiva dell'utente che sta utilizzando l'applicativo. Questo software consente di realizzare applicazioni di Realtà Aumentata sia marker-based, attraverso l'utilizzo di *fiducial markers* chiamati VuMark, sia *markerless*, avendo quindi come target immagini 2D o modelli 3D. Offre inoltre API in C++ e in Java, riuscendo ad essere compatibile sia per lo sviluppo in Android che in iOS.

Viene principalmente utilizzato nelle industrie, cliente principale di PTC Inc. che acquistò Vuforia nel 2015. All'acquisto del kit viene anche offerto un servizio di consulenza per capire come la Realtà Aumentata possa essere utilizzata per migliorare la produzione nell'industria in questione.

A.2.2 8th Wall

Leader del settore per applicazioni *Web-based* in Realtà Aumentata è sicuramente 8th Wall, di recente acquisito da Niantic per avvicinarsi ai nuovi orizzonti della Realtà Virtuale, specialmente il Metaverso [1].

8th Wall permette agli sviluppatori di creare, collaborare e pubblicare esperienze Web AR in grado di funzionare su ogni browser [62]. Costruito interamente conforme agli standard JavaScript e WebGL, offre funzionalità di localizzazione e *mapping*, *world tracking*, tracciamento immagini e *face effects*.

Offre inoltre un servizio di *Cloud Editor* che permette di sviluppare progetti Web AR esaltandone tutte le possibili funzionalità, anche collaborando con altri utenti in tempo reale e un servizio di *hosting*, garantendo la possibilità di salvare ogni progetto in maniera affidabile e sicura. Questa piattaforma si integra bene con le librerie grafiche A-Frame e three.js.

Appendice B
GitHub

La piattaforma Github [64] è pensata per aiutare gli sviluppatori ad archiviare e gestire i loro codici. Essendo un servizio Web e *cloud-based* si riesce a tener traccia e controllare le modifiche apportate al proprio codice.

Github si basa su due principi fondamentali: il controllo delle versioni e il sistema Git.

Per controllo delle versioni si intende il supporto che consente agli sviluppatori di tracciare e gestire le modifiche al codice da essi progettato. Gli utenti di questo servizio riescono così a lavorare in sicurezza potendo far uso, in particolare, di due funzionalità: il *branching* (ramificazione) e il *merging* (fusione).

Il controllo delle versioni aiuta gli sviluppatori a tracciare e gestire le modifiche al codice di un progetto software. Esso permette agli sviluppatori di lavorare in sicurezza attraverso il *branching* (ramificazione) e il *merging* (fusione). Tramite il *branching*, uno sviluppatore può 'clonare' parte o la totalità del codice presente in una *repository*, avendo quindi la possibilità di poter apportare in modo sicuro le modifiche che desidera, senza influenzare il 'codice sorgente'. Completate le modifiche, se funzionanti, possono essere 'fuse', tramite il *merging*, nel codice principale, rendendo quindi ufficiale il l'apporto dello sviluppatore al progetto.

Il sistema Git consiste in un controllo versioni distribuito, dove l'intero codice sorgente e la cronologia delle modifiche apportate sono visibili ad ogni sviluppatore, facilitando quindi la creazione di ramificazioni e fusioni.

Il vantaggio di utilizzare GitHub è il servizio cloud che offre, tramite il quale si effettua un *hosting* di *repository* Git. Individui o team di sviluppatori fanno uso di questo servizio per il controllo delle versioni e la collaborazione. L'interfaccia della piattaforma offre la possibilità di visualizzare le varie ramificazioni, i *branch*, del progetto a cui si sta lavorando, oltre che tutti i suoi *commit*, ovvero le modifiche

apportate da ogni singolo utente. Per apportare modifiche a un codice si effettua una *pull request*, ovvero una richiesta, rivolta all'utente responsabile di un determinato progetto, di includere la modifica suggerita.

Appendice C
Codici

Di seguito verrà mostrato il codice scritto per realizzare il file *query.php*, tramite il quale viene generata la risposta JSON utilizzata nell'applicazione.

```php
<?php
if(isset($_GET['room'])){
    $room = $_GET['room'];
    if($room=="A")
        $data = ["room"=>"A","devices"=>[1,2]];
    else if($room=="B")
        $data=["room"=>"B","devices"=>[3]];
    else $data=["error"=>"The specified room does not exists"];
}

else if(isset($_GET['sid'])){

$sensor = $_GET['sid'];
date_default_timezone_set("Europe/Rome");

// Temperature sensor
if ($sensor == 1) {
    $data = ["sid"=>1,
             "type"=>"sensor",
             "name"=>"Temperature sensor 01",
             "room"=>"A",
             "timestamp"=>date("Y-m-d h:i:sa"),
             "measure"=>"temperature",
             "uom"=>"Celsius",
             "value"=>rand(2000,2300)/100.0];

}
else if ($sensor == 2) {
```

```
    $data = ["sid"=>2,
             "type"=>"sensor",
             "name"=>"Pressure sensor 02",
             "room"=>"A",
             "timestamp"=>date("Y-m-d h:i:sa"),
             "measure"=>"pressure",
             "uom"=>"hPa",
             "value"=>rand(10030,10240)/10.0];
}

else if ($sensor == 3) {
    $message="ok";
    $status =  ["idle","busy","pause"];
    $errorMessages=["Error: paper jam. Please contact the
administrator","Error: paper tray is empty", "Warning: internal
error"];
    if(rand(0,2)==0){
        $status = "idle";
        $num=0;
        }
    else {
        $num = rand(1,15);
        if(rand(1,2)==1)
            $status="busy";
        else { $status="pause";
               $message=$errorMessages[rand(0,2)];
        }
    }

    $data = ["sid"=>3,
             "type"=>"printer",
             "name"=>"Printer 03",
             "room"=>"B",
             "timestamp"=>date("Y-m-d h:i:sa"),
             "status"=>$status,
             "queue"=>$num,
             "message"=>$message,
             "ink_level"=>rand(5,200)];
}


else {
    $data = [ "error"=>"The sid does not exists" ];
```

```
}

}
else {$data=["error"=>"Malformed URL exception"];}

header('Content-type: application/json');
header('Access-Control-Allow-Origin: *');
echo json_encode( $data );
?>
```

Bibliografia

[1] Nicolo Carpignoli: Seminario "Aumentare la Realtà", Space13 Legnago, 2021 [online: https://www.youtube.com/watch?v=3TR2-M4qH18].

[2] Dieter Schmalstieg e Tobias Hollerer: “Augmented Reality, Principles and Practice”, Giugno 2016

[3] Maximilian Speicher: “What is augmented reality, anyway?”, 2018

[4] Kate Raynes-Goldie: “Augmented reality furniture and other signs we're living in the future”, 2017

[5] Mark Claydon: “Alternative realities: from augmented reality to mobile mixed reality”, 2015

[6] Ronald Azuma: “A survey of augmented reality. Presence: Teleoperators and Virtual
Environments”, 6, 02 1997. doi: 10.1162/pres.1997.6.4.355.

[7] Damiano Oriti: “Progettazione e Sviluppo di Ambienti di Realtà Virtuale e Aumentata Multiutente per l’Entertainment”, 2018

[8] Caudell, Thomas & Mizell, David: “Augmented reality: An application of heads-up display technology to manual manufacturing processes”, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences. 2. 659 - 669 vol.2. 10.1109/HICSS.1992.183317.

[9] Ivan E. Sutherland: “A head-mounted three-dimensional display”, in AFIPS Conference Proceedings (1968) 33, I, pages 757–764, 1968.

[10] Sutherland, I.E: “The Ultimate Display”, 1965. In Proceedings of IFIP 65 (vol 2, pp. 506-508).

[11] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino: “Augmented reality: A class of displays on the reality-virtuality continuum”, 1994. Telemanipulator and Telepresence Technologies, 2351, 01 1994. doi: 10.1117/12.197321.

[12] P. Milgram and F. Kishino: "A taxonomy of mixed reality visual displays", IEICE (Institute of Electronics, Information and Communication Engineers) Transactions on Information and Systems, Special issue on Networked Reality, Dec. 1994.

[13] Amandeep Kour: “ A Survey on virtual world”. International Journal of Scientific and Research Publications, Volume 5, Issue 4, April 2015 1 ISSN 2250-3153

[14] Mazuryk, Tomasz & Gervautz, Michael: “Virtual Reality - History, Applications, Technology and Future”,1999.

[15] Barba, E., MacIntyre, B., Rouse, R., & Bolter, J.: “Thinking inside the box: making meaning in a handheld AR experience”, 2010. In Mixed and Augmented Reality - Arts, Media, and Humanities (ISMAR-AMH), 2010 IEEE International Symposium On (pp. 19-26). IEEE.

[16] Nokia Research Centre (NRC) (Finland): “Mobile Mixed Reality, The Vision”, 2009.

[17] Procházka, David & Stencl, Michael & Popelka, Ondřej & Stastny, Jiri.: "Mobile Augmented Reality Applications", 2011.

[18] Qiao, Xiuquan & Pei, Ren & Dustdar, Schahram & Liu, Ling & Ma, Huadong & Junliang, Chen.: "Web AR: A Promising Future for Mobile Augmented Reality - State of the Art, Challenges, and Insights", 2019. Proceedings of the IEEE. 107. 1-16. 10.1109/JPROC.2019.2895105.

[19] Mansoor Iqbal: "Pokémon Go Revenue and Usage Statistics", 2022.

[20] J. G. Andrews et al., "What will 5G be?". IEEE J.Sel. Areas Commun., vol. 32, no. 6, pp. 1065–1082, Jun. 2014.

[21] Daisy, Anjali: "Relationship Marketing Through Virtual Reality and Augmented Reality", 2020 10.4018/978-1-7998-2874-7.ch001.

[22] Mansoor Iqbal: "Instagram Revenue and Usage Statistics", 2022.

[23] Ha, Ho-Gun & Hong, Jaesung: "Augmented Reality in Medicine. Hanyang Medical Reviews", 2016 36. 242. 10.7599/hmr.2016.36.4.242.

[24] Mathavan, Suresh: "Augmented Reality in Military Applications". International Journal of Engineering and Advanced Technology", 2019 9. 51-54. 10.35940/ijeat.A1010.1091S19.

[25] Mark A. Livingston, Lawrence J. Rosenblum, Dennis G. Brown, Gregory S. Schmidt, Simon J. Julier, Yohan Baillot, J. Edward Swan II, Zhuming Ai, and Paul Maassel: "Military Applications of Augmented Reality", 2011.

[26] Tandel, Sayali & Jamadar, Abhishek: "Impact of Progressive Web Apps on Web App Development", 2018 10.15680/IJIRSET.2018.0709021.

[27] Junwei Yu, Lu Fang and Chuanzheng Lu: "Key technology and applicationresearch on mobile augmented reality", 2016.

[28] Khronos Group : "OpenGL ES Overview", 2003.

[29] D. Chatzopoulos, C. Bermejo, Z. Huang and P. Hui: "Mobile Augmented Reality Survey: From Where We Are to Where We Go" in IEEE Access, vol. 5, pp. 6917-6950, 2017, doi: 10.1109/ACCESS.2017.2698164.

[30] WANG Shoujue, Y.Jiang, L. Tan: "A Point-to-Point Tracking Algorithm Based on the Theory of Floating Pixels" Chinese Journal of Electronics,vo1.25,no.Ol,pp.1-5, 2016.

[31] A. Charland and B. Leroux: "Mobile application development: Web vs. native," Commun. ACM, vol. 54, no. 5, pp. 49–53, May 2011.

[32] AR.js Documentation https://ar-js-org.github.io/AR.js-Docs/

[33] What is A-Frame? https://aframe.io/docs/1.3.0/introduction/#what-is-a-frame

[34] Billinghurst, M., Clark, A., & Lee, G: "A survey of augmented reality. Foundations and Trends in Human–Computer Interaction", 2014 8(2–3), 73–272.

[35] D. Khan et al.: "Robust Tracking Through the Design of High Quality Fiducial Markers: Optimization Tool for ARToolKit", IEEE Access, vol. 6, pp. 22421-22433, 2018.

[36] R. M. Freeman, S. J. Julier and A. J. Steed: "A Method for Predicting Marker Tracking Error", 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, 2007, pp. 157-160, doi: 10.1109/ISMAR.2007.4538841.

[37] Ćuković, Saša & Gattullo, Michele & Pankratz, Frieder & Devedzic, Goran & Carrabba, Ernesto & Baizid, Khelifa: "Marker Based vs. Natural Feature Tracking Augmented Reality Visualization of the 3D Foot Phantom", 2015.

[38] M. Fiala: "Designing highly reliable ducial markers", IEEE Trans. Pat-tern Anal. Mach. Intell., vol. 32, no. 7, pp. 13171324, Jul. 2010.

[39] David Forsyth: "Computer Vision: A Modern Approach", Pearson Education, 2012.

[40] H. Kato and M. Billinghurst: “Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System”, Proc. Second IEEE and ACM Int’l Workshop Augmented Reality, vol. 85, pp. 20-21, Oct. 1999.

[41] Nicolò Carpignoli: “AR.js — Quando il web incontra la Realtà Aumentata (medium)”, 2018.

[42] Billinghurst, M., Kato, H.: “Collaborative Mixed Reality”, proceedings of ISMR '99. Springer Verlag. pp. 261-284.

[43] W.Wang, H.Wan: “Real-time camera tracking using hybrid features in mobile augmented reality”, J.Science China(Information Sciences), vol 58, no.ll, pp.211-223, 2015.

[44] M. XU,Y. Song, S.Lv, Z.Liu, C.Huang: “Design of Objects Tracking System Based on ARM Embedded Platform”, J .Computer Aided Drafting,Design and Manufacturing,vo1.24,no.03,pp.65-69,2014.

[45] X. HOU, C. Ll: “Research on the Three-dimensional Modeling and Optimization of a Virtual Tower Crane Based on 3DS Max, Solidworks and EON Professional”, J. International Journal of Plant Engineering and Management, vo1.0 1,no.03,pp. 15-19,2013.

[45] Oberhofer, Christoph & Grubert, Jens & Reitmayr, Gerhard: “Natural feature tracking in JavaScript”, 2012 113-114. 10.1109/VR.2012.6180908.

[46] https://developer.mozilla.org/en-US/docs/Web/API/fetch

[47] Wagner, Daniel & Schmalstieg, Dieter: “ARToolKitPlus for Pose Tracking on Mobile Devices”, 2007.

[48] Mark Billinghurst, Ivan Poupyrev, Hirokazu Kato, and Richard May: “Mixing realities in shared space: An augmented reality interface for collaborative computing”, in Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, volume 3, pages 1641–1644. IEEE, 2000.

[49] Rekimoto, Jun: “Matrix: A Realtime Object Identification and Registration Method for Augmented Reality”, 1998 63 - 68. 10.1109/APCHI.1998.704151.

[50] Alexandra Etienne: “How To Create your Own Marker?”, 2017 [online: https://medium.com].

[51] http://www.hitl.washington.edu/artoolkit/

[52] https://kalwalt.github.io/artoolkit-docs/3_Marker_Training/marker_nft_training.html

[53] Kato, Hirokazu & Tachibana, Keihachiro & Billinghurst, Mark & Grafe, Michael: “A registration method based on texture tracking using ARToolKit”, 2003 IEE Review. 77 - 85. 10.1109/ART.2003.1320435.

[54] Kato, Hirokazu: “Inside ARToolKit”, 2004 Hiroshima City University.

[55] Z. Huang, W. Li, P. Hui, and C. Peylo “CloudRidAR: A cloud based architecture for mobile augmented reality”, in Proc. Work-shop Mobile Augmented Reality Robotic Technol.-Based Syst. (MARS), New York, NY, USA, 2014, pp. 2934.

[56] ZOE S.R.L.S.: “3 Tipologie di app per dispositivi mobile. Quale è adatta al tuo progetto?”, 2020 [Online: https://www.zoewebsolutions.it/3-tipologie-di-app-perdispositivi-mobile-quale-%C3%A8-adatta-al-tuo-progetto].

[57] European Commission, Directorate-General for Research and Innovation, Breque, M., De Nul, L., Petridis, A.: “Industry 5.0 : towards a sustainable, human-centric and resilient European industry”, Publications Office, 2021 [online: https://data.europa.eu/doi/10.2777/308407].

[58] Ilaria Sassone: “Industria 5.0: cosa c’è da sapere e come impatterà le aziende”, 2021 [online: https://universeit.blog/].

[59] Fabio Cortes: “Manipulate your 3D content with gestures in AR.js”, 2020. [online: https://medium.com]

[60] https://www.argonjs.io/

[61] https://immersive-web.github.io/webxr/

[62] https://www.8thwall.com/docs/web/#introduction

[63] https://threejs.org/docs/

[64] https://github.com/

[65] Tobias Kahlert and Kay Giza · Microsoft Germany: "Visual Studio Code-Tips & Tricks", Vol.1, 2016.

[66] Berger, Cosima & Gerke, Markus: "Comparison of Selected Augmented Reality Frameworks for Integration in Geographic Citizen Science Projects", 2022. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLIII-B4-2022. 10.5194/isprs-archives-XLIII-B4-2022-223-2022.

[67] https://jeromeetienne.github.io/AR.js/three.js/examples/marker-training/examples/generator.html

[68] https://stablekernel.com/article/native-web-hybrid-lets-talk-about-mobile-apps/

[69] https://www.redhat.com/it/topics/internet-of-things/what-is-iot

[70] https://httpwg.org/specs/rfc9110.html

[71] https://www.json.org/json-en.html

[72] http://www.artoolkitx.org/

[73] https://www.redhat.com/it/topics/api

[74] https://www.khronos.org/webgl/wiki/Getting_Started

[75] https://www.ptc.com/en/products/vuforia

[76] Lowe G. David: "Object recognition from local scale-invariant features", 1999. *Proceedings of the International Conference on Computer Vision*. Vol. 2. pp. 1150–1157. doi:10.1109/ICCV.1999.790410.

[77] Lowe G. David: "Distinctive Image Features from Scale-Invariant Keypoints", 2004. *International Journal of Computer Vision*. **60** (2): 91–110. CiteSeerX 10.1.1.73.2924. doi:10.1023/B:VISI.0000029664.99615.94. S2CID 221242327.

[78] https://kalwalt.github.io/artoolkit-docs/3_Marker_Training/marker_nft_training.html

[79] B. Chen, J. Wan, L. Shu, P. Li, M. Mukherjee and B. Yin, "Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges" in *IEEE Access*, vol. 6, pp. 6505-6519, 2018, doi: 10.1109/ACCESS.2017.2783682.

[80] Wang, Shiyong & Wan, Jiafu & Li, Di & Zhang, Chunhua, 2016: "Implementing Smart Factory of Industrie 4.0: An Outlook". International Journal of Distributed Sensor Networks. 2016. 1-10. 10.1155/2016/3159805.

[81] Gillis, Alexander, 2021: "What is internet of things (IoT)? ". *IOT Agenda*. Retrieved 17 August 2021.

[82] Gil D, Ferrández A, Mora-Mora H, Peral J. Internet of Things: A Review of Surveys Based on Context Aware Intelligent Services. Sensors (Basel). 2016 Jul 11;16(7):1069. doi: 10.3390/s16071069. PMID: 27409623; PMCID: PMC4970116.

[83] Cisco IBSG, April 2011.