

UNIVERSITÀ DEGLI STUDI DI NAPOLI “PARTHENONE”

SCUOLA INTERDIPARTIMENTALE DELLE SCIENZE,
DELL'INGEGNERIA E DELLA SALUTE

DIPARTIMENTO DI SCIENZE E TECNOLOGIE

CORSO DI LAUREA IN INFORMATICA



TESI DI LAUREA TRIENNALE

**Rilevazione di persone mediante
scheletro 3D con telecamere di
profondità**

RELATORE

Prof. Ferone Alessio

CANDIDATO

Caruso Denny
matr. 0124002062

Anno Accademico 2021/2022

*Siamo come nani sulle spalle di giganti, così che possiamo vedere più cose
di loro e più lontane, non certo per l'altezza del nostro corpo, ma perché
siamo sollevati e portati in alto dalla statura dei giganti.*

Bernardo di Chartres

Sommario

Sempre più spesso e sempre in più contesti la sicurezza è messa a rischio: la sicurezza delle persone, la sicurezza degli oggetti, la sicurezza degli ambienti. Al fine di garantire il massimo grado di sicurezza in tempo reale, vengono adottate misure sempre più avanzate e, in alcuni casi, anche costose e stringenti. Uno dei metodi più convenzionali per sorvegliare una scena o un ambiente all'interno del quale si vuole garantire la sicurezza in tempo reale, è quella d'installare dei dispositivi di videosorveglianza che catturano video 2D; ovvero, senza l'informazione sulla profondità. Questo tipo di approccio, nato ormai da diversi decenni, necessita di sempre più operatori al crescere del numero di videocamere installate.

Con l'avvento dell'intelligenza artificiale è possibile automatizzare parte di questo lavoro, riuscendo a rilevare con un elevato grado di precisione eventuali anomalie, scene sospette ed eventi che potrebbero mettere a rischio la sicurezza della scena osservata. Una forte limitazione a tale approccio, però, è l'assenza dell'informazione sulla profondità all'interno della scena e di conseguenza l'assenza dell'informazione sulla profondità di oggetti e persone in essa presenti. Una possibile soluzione è quella di adottare nuovamente una soluzione che faccia uso dell'intelligenza artificiale per ricavare la profondità. Questa tipologia di approccio però, in quest'ultimo caso, comporta uno svantaggio: l'imprecisione più o meno elevata sull'informazione relativa alla profondità, a causa del fatto che si tratta di un'approssimazione ottenuta mettendo in correlazione la scena considerata, con scene simili.

Con questa tesi, si vuole dimostrare com'è possibile realizzare una soluzione di videosorveglianza 3D, che tenga conto dell'informazione sulla profondità grazie a un insieme di sensori hardware appositi, che sia in grado di rilevare più persone all'interno della scena osservata, che sia in grado di rilevare anomalie e comportamenti anomali, che sia di facile implementazione ed economicamente conveniente.

La tesi è organizzata seguendo un percorso lineare che man mano conduce all’implementazione del sistema completo. Di seguito si riporta una panoramica della tesi, con una breve descrizione per ogni capitolo. Il capitolo 1 riassume il contesto e l’obiettivo finale della tesi. Inoltre, fornisce alcune nozioni di base e poi avanzate per gli aspetti teorici affrontati nei successivi capitoli e che rappresentano le fondamenta del funzionamento del progetto di tesi. Tra le nozioni teoriche fondamentali verranno trattate le modalità di percezione della profondità da parte dell’occhio umano, il problema della stima della profondità per un dispositivo elettronico, cenni d’intelligenza artificiale, geometria epipolare, grafi e molto altro ancora. Il capitolo 1, quindi, è una sorta di capitolo preparativo che illustra gli *ingredienti* fondamentali per realizzare l’intera architettura.

Il capitolo 2 contiene tutte le scelte seguite in fase di progettazione al fine di realizzare questo particolare modulo dell’intero sistema: l’approccio usato per ricavare le informazioni sulla profondità, quello usato per l’estrazione, da un’immagine, di particolari strutture dette *skeleton*, quello impiegato per la conversione fra sistemi di coordinate, quello utilizzato per l’invio delle informazioni prodotte, la struttura architettonale e altri dettagli relativi alla fase di progettazione. Quindi, dopo aver preso in considerazione gli *ingredienti* fondamentali nel capitolo 1, nel capitolo 2 si mostrano le problematiche del dominio applicativo e le soluzioni a disposizione.

Il capitolo 3 indica le modalità d’implementazione del modulo realizzato dal candidato. Nel fare ciò, si illustrano le tecnologie che implementano gli approcci analizzati nel capitolo 2, il loro funzionamento ed eventuali alternative. Inoltre, il capitolo 3 mostra come integrare tra di loro le soluzioni scelte per ogni problema del dominio applicativo, motivando opportunamente il tutto. Infine, il capitolo 3 mostra i test e i risultati sperimentali ottenuti.

Per concludere, il capitolo 4 riassume brevemente la tesi, trae delle deduzioni sul lavoro svolto e si conclude proponendo sviluppi e miglioramenti futuri del modulo sviluppato dal candidato e dell’intero sistema realizzato.

Indice

1	Introduzione	1
1.1	Contesto	1
1.2	Obiettivo	4
1.3	Concetti preliminari	11
1.4	Aspetti teorici	16
1.4.1	Profondità e occhio umano	16
1.4.2	Profondità e videocamere RGBD	20
1.4.3	Cenni di geometria epipolare	22
1.4.4	Cenni di morfologia matematica	27
1.4.5	Intelligenza artificiale	29
1.4.6	Il percettrone	31
1.4.7	Artificial Neural Network	33
1.4.8	Convolutional Neural Network	34
1.4.9	Non Maximum Suppression (NMS)	36
1.4.10	Cenni sui Grafi	37
1.4.11	Algoritmi greedy	38
2	Progettazione	39
2.1	Depth Estimation	39
2.2	Pose Estimation	42
2.3	Trasformazione di sistemi di coordinate	51
2.4	Streaming di eventi	56
2.5	Scelte progettuali	61
2.6	Flusso di esecuzione AI Watch A1	63
2.7	Complessità computazionale	66
2.8	Ingegnerizzazione AI Watch A1	69
2.8.1	Design Pattern utilizzati	73
3	Implementazione e test	81
3.1	Intel RealSense	81
3.1.1	Sistema di coordinate tridimensionale Intel RealSense .	90

Indice

3.1.2	Hole-filling spaziale	94
3.2	Convolutional Pose Machines	95
3.2.1	OpenPose	97
3.3	Algoritmi per la trasformazione di SC	105
3.4	Apache Kafka	108
3.5	Aspetti implementativi	115
3.6	Test e risultati	119
3.7	Architettura complessiva A1	128
4	Conclusione e sviluppi futuri	131

Elenco delle figure

1.1	Telecamere Intel RealSense [68].	2
1.2	Casi d'uso videocamere [69].	3
1.3	CVPR Lab.	4
1.4	Architettura completa AI Watch.	5
1.5	Architettura completa del modulo realizzato dal candidato.	6
1.6	Posizionamento telecamera all'interno del CVPR Lab.	7
1.7	Skeleton [72].	8
1.8	CVPR Lab reale e corrispettivo gemello digitale.	10
1.9	Rappresentazione immagine digitale.	11
1.10	Spettro elettromagnetico [75].	12
1.11	Spazio colore RGB.	13
1.12	Immagine RGB.	14
1.13	Immagine RGBD.	14
1.14	Accomodazione e convergenza visiva [5].	17
1.15	Dimensioni dell'immagine e percezione della profondità [5].	18
1.16	Due fotocamere che acquisiscono un'immagine della stessa scena idealmente nello stesso istante[81].	23
1.17	Determinazione della profondità per ogni pixel dell'immagine acquisita [81].	25
1.18	Stereo-matching e disparità immagini [81].	26
1.19	Illustrazione del graduale riempimento di vuoti durante l'esecuzione dell'algoritmo [3].	28
1.20	Un neurone biologico confrontato con una rete neurale artificiale [14].	31
1.21	Rete neurale convoluzionale [82].	34
1.22	Applicazione della NMS [83].	36
1.23	Esempi di grafi [84].	37
2.1	Tecnologia Raytrix [77].	40
2.2	Vista della scena mediante microarray lens [77].	40
2.3	Processo di <i>pose estimation</i> in PoseNet [90].	43

Elenco delle figure

2.4	Modelli di skeleton ricavati con diversi approcci [29].	47
2.5	Diverse tecniche per la rappresentazione dello skeleton basate su grafi [29].	50
2.6	Immagine risultato skeleton frame 395 - Test 3.	54
2.7	Interpretazione del problema relativo alla trasformazione fra sistemi di coordinate.	55
2.8	Diagramma delle classi AI Watch A1 semplificato.	72
2.9	Esempio strutturale con Façade per risolvere diverse interdipendenze in un sottomodulo dal funzionamento complesso [49].	74
2.10	Struttura del pattern Singleton [49].	75
2.11	Struttura del pattern Command [49].	76
2.12	Struttura del pattern Strategy [49].	77
2.13	Struttura del pattern Decorator [49].	79
3.1	Viste differenti della telecamera Intel RealSense D435 [68]. . .	82
3.2	Architettura interna della telecamera Intel RealSense D430 [68].	83
3.3	Effetto Rolling Shutter [100].	84
3.4	Telecamera Intel RealSense T265 [68].	86
3.5	Pattern circolari emessi da Intel RealSense D415 e D435 [103].	87
3.6	Immagine lidar [103].	89
3.7	Tre approcci differenti per effettuare il riempimento dei fori nelle immagini acquisite con telecamera Intel RealSense [68]. .	94
3.8	Convolutional Pose Machine [52].	95
3.9	Esempio OpenPose [45].	100
3.10	Schematizzazione del funzionamento di OpenPose [45].	102
3.11	Risultati raggiunti da OpenPose [45].	104
3.12	Un esempio di utilizzo di Apache Kafka [74].	110
3.13	Componenti di base in Kafka [105].	110
3.14	Topic e partizioni in Kafka [105].	111
3.15	Struttura di un messaggio inviato da un produttore in Kafka [105].	112
3.16	Replicazione delle partizioni in Kafka [105].	113
3.17	Struttura ad albero di file e cartelle AI Watch A1.	116
3.18	Immagine risultato skeleton frame 155 - Test 2.	122
3.19	Immagine risultato skeleton frame 198 - Test 2.	123
3.20	Immagine risultato skeleton frame 341 - Test 2.	123
3.21	Immagine risultato skeleton frame 576 - Test 2.	125
3.22	Immagine risultato skeleton frame 278 - Test 3.	126
3.23	Immagine risultato skeleton frame 163 - Test 3.	126

Elenco delle figure

Elenco delle tabelle

2.1	Approcci e tecnologie per lo streaming di eventi.	60
3.1	Differenze tra telecamere orientate al “field of view” oppure al “light spectrum”.	85
3.2	Operazioni di <i>projection</i> e <i>deprojection</i> a confronto.	91
3.3	Misure reali di Renato Esposito e Denny Caruso	124
3.4	Misure rilevate di Renato Esposito e Denny Caruso	124

Elenco degli algoritmi

1	Trasformazione ascissa di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale.	106
2	Trasformazione ordinata di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale.	106
3	Trasformazione ordinata semplificata di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale (valido per formato ordinata Intel RealSense D435).	107
4	Operazioni principali di AI Watch A1.	130

Capitolo 1

Introduzione

Sicurezza: la condizione che rende e fa sentire di essere esente da pericoli, o che dà la possibilità di prevenire, eliminare o rendere meno gravi danni, rischi, difficoltà, evenienze spiacevoli, e simili [67].

1.1 Contesto

Oggigiorno si sente parlare sempre più spesso di sicurezza: sicurezza sul lavoro, in mare, in volo, in casa, di persone, di oggetti, di ambienti e così via. Per preservare e mantenere la sicurezza, è possibile adottare diversi approcci: alcuni più costosi, altri più invadenti e stringenti, altri ancora più innovativi. Un approccio classico per mantenere la sicurezza, è l'utilizzo di videocamere di sorveglianza.

Una videocamera è un apparecchio portatile che permette di registrare su uno o più supporti, dei segnali corrispondenti a immagini e suoni; comprende un sistema ottico (in genere uno zoom con regolazione automatica dell'esposizione), un microfono, un sensore capace di trasformare i segnali luminosi in segnali elettronici, un piccolo videoregistratore e solitamente viene collegata a un monitor più o meno grande, su cui si possono osservare le immagini registrate. Una videocamera può avere anche funzionalità tecnologicamente avanzate o di rilievo, come la capacità di operare in maniera wireless, la capacità di acquisire immagini ad alta risoluzione e suoni limpidi, la capacità d'interfacciarsi con smartphone, tablet, computer e cloud, la capacità di acquisire o dedurre la profondità all'interno dell'immagine e così via.



Figura 1.1. Telecamere Intel RealSense [68].

Una videocamera di sorveglianza, o alternativamente telecamera di sorveglianza, ha tutte le caratteristiche di una videocamera. Il fine per il quale viene utilizzata però, è specificamente rivolto alla sicurezza. Anche questa tipologia di videocamere possono avere funzioni più o meno avanzate. Per esempio, in commercio sono presenti delle videocamere di sorveglianza che permettono il rilevamento di anomalie, di comportamenti inconsueti, di persone e di oggetti grazie all'utilizzo di tecniche avanzate di machine learning e intelligenza artificiale. Inoltre, esistono apposite telecamere che permettono l'acquisizione dell'informazione sulla profondità, mentre altre ancora permettono l'elaborazione d'immagini di diversa natura. Un esempio di telecamere in grado di acquisire e gestire l'informazione sulla profondità della scena inquadrata, è dato dalle telecamere Intel RealSense [68]. In figura 1.1 si riporta un'immagine che mostra questa famiglia di telecamere.

Con i progressi tecnologici degli ultimi anni, si è reso possibile utilizzare delle videocamere per la visione artificiale, nella robotica, per svolgere compiti e attività di magazzino, di logistica, in ambito agricolo, medico, industriale, per la *collision avoidance*, *gesture recognition*, *skeleton detection* e *skeleton tracking*, in ambito videoludico e in tanti altri domini applicativi. In figura 1.2 si riportano alcuni dei domini applicativi citati. Con il veloce avanzamento della tecnologia è possibile impiegare telecamere, ormai sempre più avanzate, anche per il *perceptual computing*. Con questa espressione, si

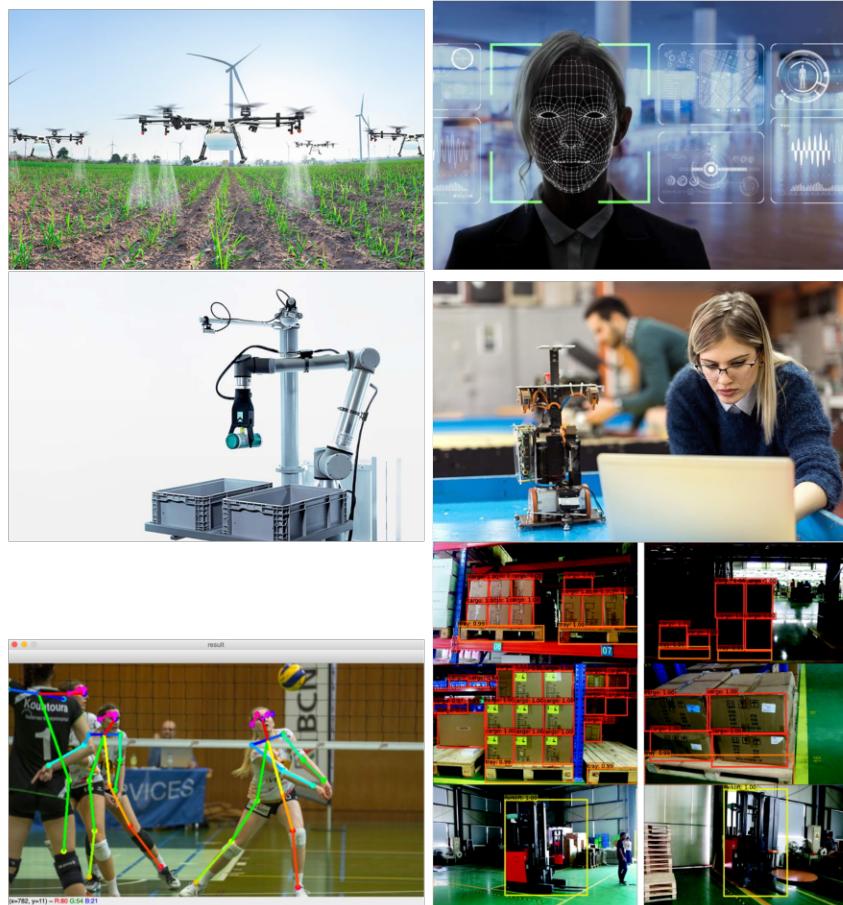


Figura 1.2. Casi d'uso videocamere [69].

fa riferimento alla capacità di un calcolatore di riuscire a percepire e/o analizzare l'ambiente circostante che lo circonda e agire di conseguenza. L'idea del *perceptual computing* è quella di sfruttare voce, gesti e altri input forniti mediante sensori per utilizzare il calcolatore, anziché usare mouse e tastiera. Quindi si tratta di un approccio del tutto *touch-less*. Molti, se non tutti questi domini applicativi finora citati, necessitano dell'informazione sulla profondità degli oggetti all'interno della scena osservata. Come si vedrà in seguito, esistono oggi diversi approcci per ricavare quest'informazione. Uno dei più importanti è quello che fa uso di telecamere con capacità di rilevamento della profondità.

1.2 Obiettivo

Ora si descriverà l'idea del lavoro di tesi in maniera sintetica, ma completa; per poi approfondirne maggiormente i dettagli in questo capitolo e in quelli che seguono. Il progetto completo si è svolto in modalità ibrida: da casa e presso il laboratorio “*Alfredo Petrosino - CVPR Lab*” dell’Università degli Studi di Napoli “Parthenope” [70]. Il laboratorio è visibile nei video reperibili al seguente riferimento [71] e nella figura 1.3.

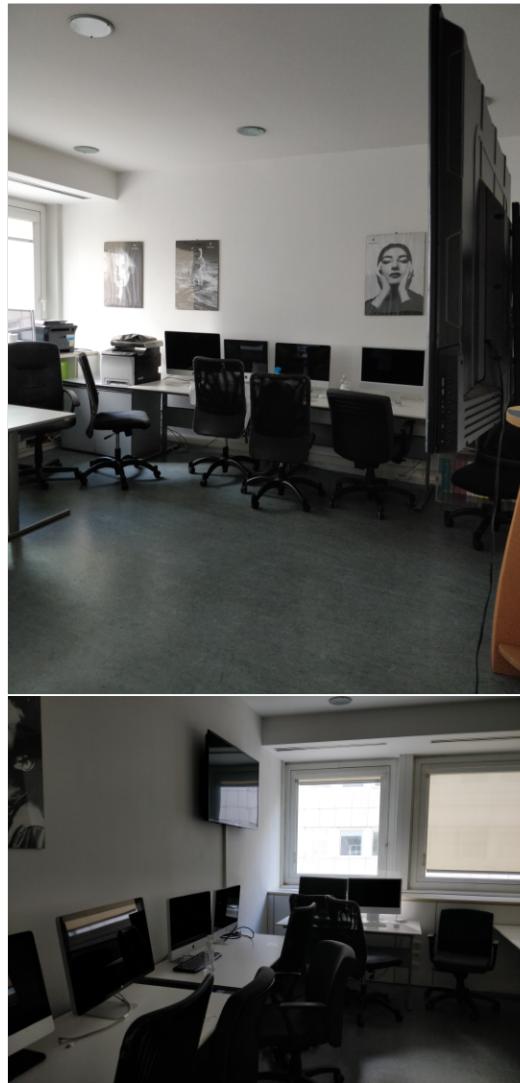


Figura 1.3. CVPR Lab.

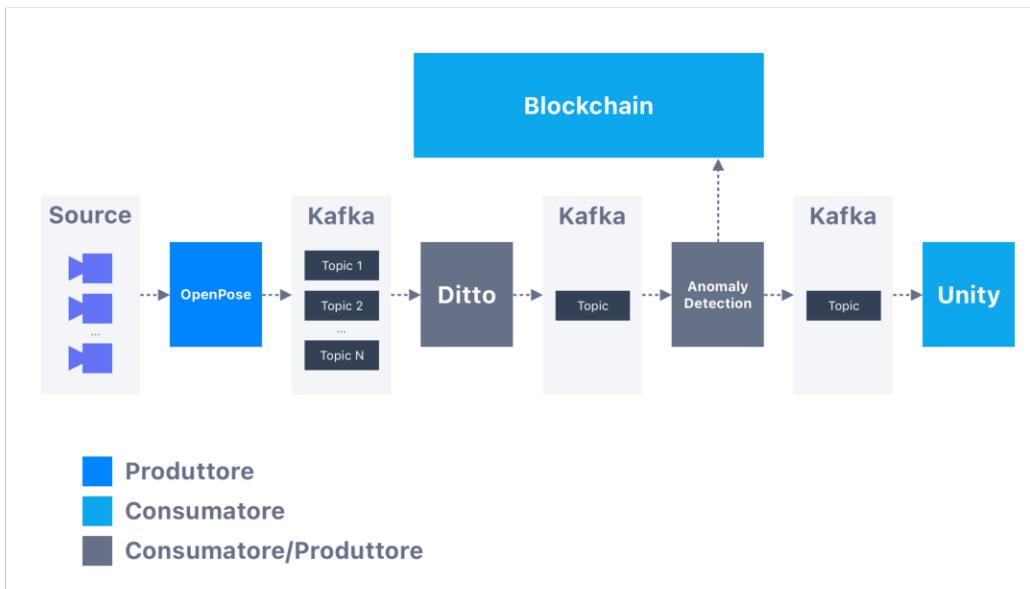


Figura 1.4. Architettura completa AI Watch.

Il progetto completo è noto col nome di “AI Watch” e al momento è articolato in quattro moduli operativi. Il candidato si è occupato del primo dei quattro moduli poc’anzi menzionati. Nella sua interezza il progetto AI Watch ha l’obiettivo di realizzare un sistema di videosorveglianza in cui le telecamere acquisiscono i frame video all’interno dei quali si rilevano le persone presenti. Successivamente si cerca di dedurre se le persone presenti nella scena stanno compiendo un’azione da considerare anomala o meno, per poi rappresentare l’intera scena in un corrispettivo ambiente virtuale. Le molteplici informazioni fondamentali relative alle anomalie vengono salvate in maniera permanente e sicura all’interno di una blockchain appositamente realizzata.

In figura 1.4, è riportata l’architettura del sistema completo e illustra l’integrazione di tutti i moduli citati. Invece, in figura 1.5, si illustra l’architettura del modulo del sistema realizzato dal candidato e discusso in questa tesi.

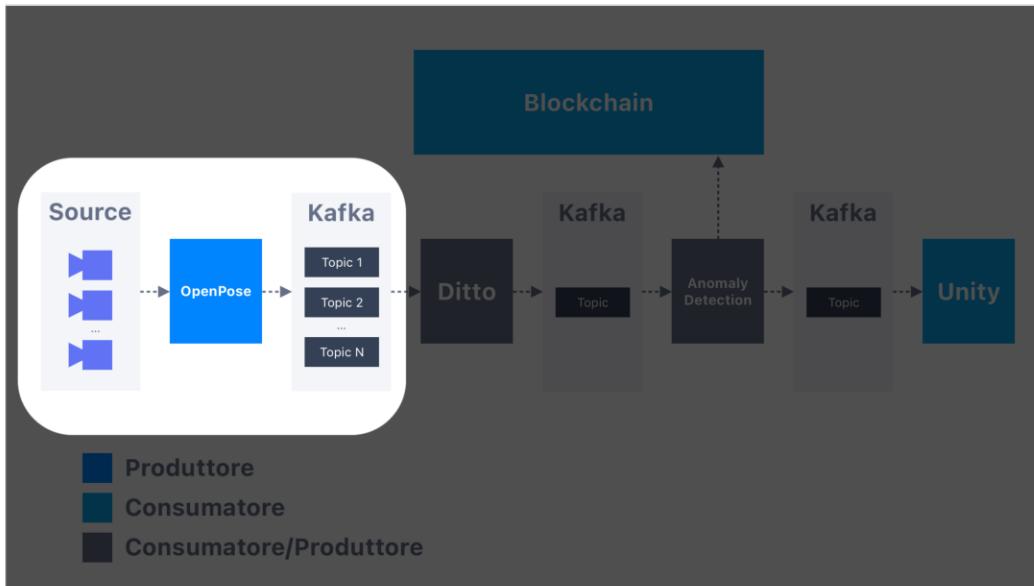


Figura 1.5. Architettura completa del modulo realizzato dal candidato.

L’idea del primo modulo è la seguente. Data una videocamera RGBD Intel RealSense D435 [68], in grado di acquisire informazioni sia sul colore che sulla profondità, si estraе e si scinde l’informazione relativa al colore, da quella relativa alla profondità. In figura 1.6 è riportata l’organizzazione e il posizionamento della telecamera all’interno del laboratorio durante le fasi di test.

Una volta fatto ciò, si sfruttano tecniche avanzate di machine learning e intelligenza artificiale, in modo da individuare e ricavare i cosiddetti “skeleton”, ovvero gli scheletri delle persone individuate all’interno dell’immagine. Ogni scheletro è composto sostanzialmente da punti e segmenti. I punti fanno riferimento a punti di giuntura fondamentali del corpo umano, mentre i segmenti si occupano di unire opportunamente tali punti di giuntura, in modo da formare lo skeleton finale. Esempi di punti di giuntura di uno skeleton sono l’orecchio destro, l’orecchio sinistro, il naso, l’occhio destro, l’occhio sinistro, i gomiti, le ginocchia e così via. In figura 1.7, è mostrato uno skeleton con una possibile annotazione per identificare i diversi punti di giuntura.

Introduzione

Obiettivo

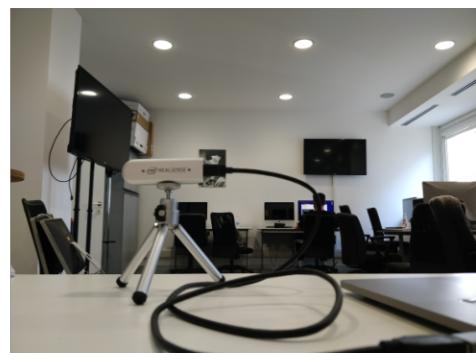
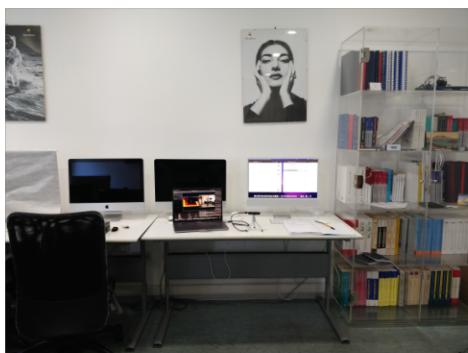


Figura 1.6. Posizionamento telecamera all'interno del CVPR Lab.

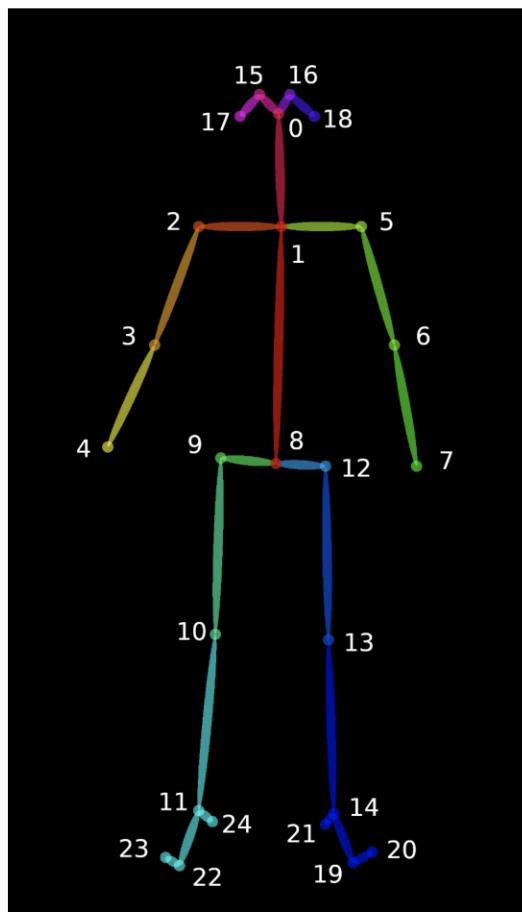


Figura 1.7. Skeleton [72].

Una volta ottenuti gli skeleton di tutte le persone presenti all'interno dell'immagine, si passa alla trasformazione del sistema di coordinate. In particolare si passa dal sistema di coordinate usato dalla videocamera RGBD a un sistema di coordinate opportunamente fissato e basato su parametri di configurazione della scena inquadrata. A questo punto si hanno a disposizione le coordinate di tutti i punti di giuntura di ogni skeleton individuato, espresse in un formato adottato per convenzione anche dai successivi moduli realizzati e che indicano per ogni punto di giuntura, la sua posizione all'interno della stanza ove è stata installata la telecamera. Tutte le informazioni finora ottenute, vengono inviate al modulo successivo che si occuperà di eseguire un controllo sulla presenza di eventuali anomalie nei movimenti eseguiti da uno o più skeleton. Per fare ciò, si fa uso ancora una volta di tecniche di machine learning e intelligenza artificiale [1].

Una volta fatto ciò, il secondo modulo si occupa d'inviare tutte le informazioni finora prodotte e le informazioni relative alle anomalie al terzo modulo, che effettuerà il rendering digitale (seguendo il modello dei digital twin) in ambiente Unity [73]. Inoltre, in tale ambiente, si mostra a video in maniera dettagliata se un determinato skeleton sta compiendo un'azione anomala o meno [2]. In figura 1.8 è illustrato parte del laboratorio riprodotto nel terzo modulo.

Infine, il quarto e ultimo modulo si occupa di salvare man mano, in maniera permanente e sicura, i dati relativi alle anomalie all'interno di una blockchain appositamente sviluppata per AI Watch.



Figura 1.8. CVPR Lab reale e corrispettivo digitale.

L'integrazione dei quattro moduli appena menzionati, avviene mediante Apache Kafka [74] e permette di mettere in funzione un sistema di rilevamento delle anomalie efficiente e multipersona con conseguente rendering video in ambiente Unity [73]. Nell'attuale elaborato di tesi, si discuterà in maniera approfondita del primo modulo, in quanto è quello relativo al lavoro di tesi del candidato. Si precisa che, in questo elaborato di tesi, si farà riferimento a questo modulo come “AI Watch A1”, o più semplicemente modulo “A1”. Il nome “A1” ha un significato ben preciso: ciò che è stato realizzato all'interno di questo modulo, è la prima delle quattro parti attualmente caratterizzanti l'intero sistema AI Watch. Di conseguenza è stata usata la lettera “A”. Invece, il numero “1” indica la versione del modulo considerato. Di conseguenza, l'unione di questi due elementi, permette d'identificare in maniera univoca il modulo del sistema e la relativa versione.

I dettagli relativi ai successivi due moduli, sono rintracciabili nei documenti citati nei riferimenti [1, 2]. Invece, al riferimento che segue, è possibile visionare due video dimostrativi che mostrano il primo modulo del sistema in esecuzione [71]. Il primo video mostra il modulo nelle sue fasi primordiali, mentre il secondo video mostra lo stato stabile raggiunto verso la fine del progetto di tesi.

1.3 Concetti preliminari

Si consideri il video in quanto elemento multimediale. Un video acquisito da una videocamera, può essere visto in prima approssimazione e in termini di rappresentazione digitale, come un insieme d'immagini e suoni. Tralasciando l'aspetto sonoro, un video può essere più o meno fluido a seconda di quanti frame per secondo, e quindi immagini, vengono acquisiti dal dispositivo incaricato.

Si consideri dapprima un'immagine in scala di grigi. Una singola immagine di questo tipo, può essere definita come una funzione bidimensionale, $f(x, y)$, dove x e y sono le coordinate spaziali (sul piano), e l'ampiezza di f in ogni coppia di coordinate (x, y) viene chiamata *intensità* o livello di grigio dell'immagine in quel determinato punto. Quando x , y e i valori dell'ampiezza assunti da f sono tutti finiti, dunque quantità discrete, si può definire l'immagine come un'immagine digitale. Ogni elemento di un'immagine è detto *picture element* o *pixel* ed è caratterizzato dall'avere una posizione e un valore all'interno dell'immagine. In figura 1.9 si schematizza quanto detto poc'anzi.

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0, N - 1) \\ f(1,0) & f(1,1) & \cdots & f(1, N - 1) \\ \vdots & \vdots & & \vdots \\ f(M - 1,0) & f(M - 1,1) & \cdots & f(M - 1, N - 1) \end{bmatrix}$$

Figura 1.9. Rappresentazione immagine digitale.

Esistono diversi dispositivi di imaging in grado di trattare e acquisire immagini a partire da componenti differenti dello spettro elettromagnetico e non. Questo elaborato di tesi si soffermerà sulle immagini acquisite a partire dalla luce visibile dello spettro elettromagnetico. In figura 1.10 si riporta l'intero spettro elettromagnetico, le lunghezze d'onda espresse in metri e la locazione della banda della luce visibile con le lunghezze d'onda espresse in nanometri.

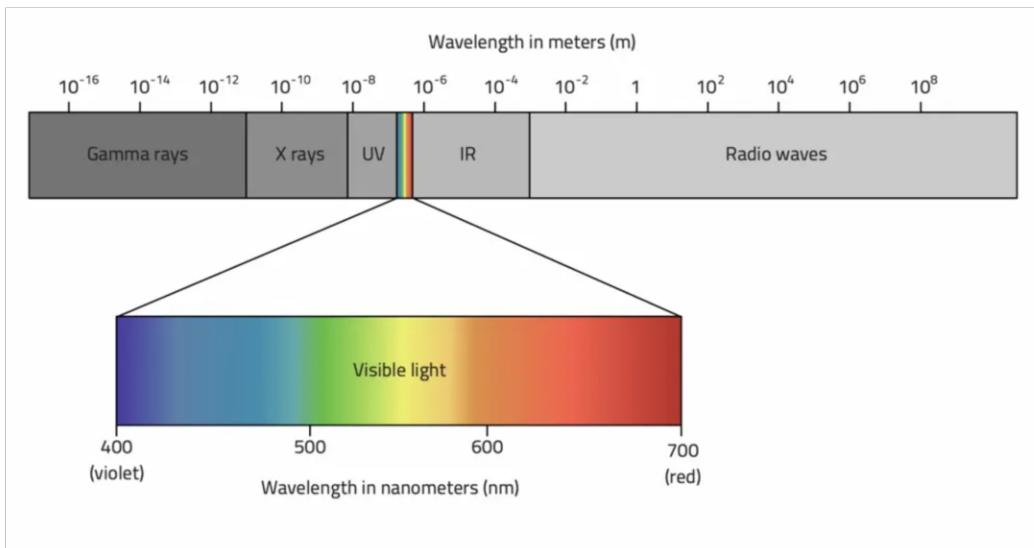


Figura 1.10. Spettro elettromagnetico [75].

Durante questo lavoro di tesi sono state adoperate tecniche dell'elaborazione delle immagini sia di basso, sia di medio, che di alto livello; integrando così sia operazioni di base, che avanzate, nell'elaborazione e nel trattamento delle immagini. Al fine di alleggerire l'elaborato, si tralasciano i dettagli relativi alla struttura e al funzionamento di un dispositivo di imaging, all'acquisizione e alla formazione di un'immagine digitale, alla risoluzione spaziale e alla risoluzione d'intensità.

Nell'elaborazione delle immagini non ci si limita soltanto all'uso di determinati livelli di grigio, ma si sfrutta anche il colore. L'uso del colore nell'elaborazione delle immagini è motivato da due fattori principali. Per prima cosa, il colore è un descrittore che semplifica l'identificazione di un oggetto e la sua estrazione da una scena. In secondo luogo, gli uomini sono in grado di distinguere migliaia di gradazioni di colore e d'intensità, in confronto a solo due dozzine di tonalità di grigio. Ciò è di particolare importanza nell'analisi manuale dell'immagine. Per rappresentare le immagini a colori si sfrutta un modello colore (o spazio colore). Il suo scopo è quello di facilitare e standardizzare la specifica dei colori. In sostanza, un modello colore è un sistema di coordinate e di un sottospazio all'interno di quel sistema dove ogni colore viene rappresentato da un singolo punto. I modelli colore maggiormente noti e utilizzati sono RGB, CMY, CMYK e HSI. Il modello colore usato durante questo lavoro di tesi è quello RGB, dove il modello si basa su un sistema di coordinate cartesiane, mentre il sottospazio d'interesse è un cubo unitario. Il

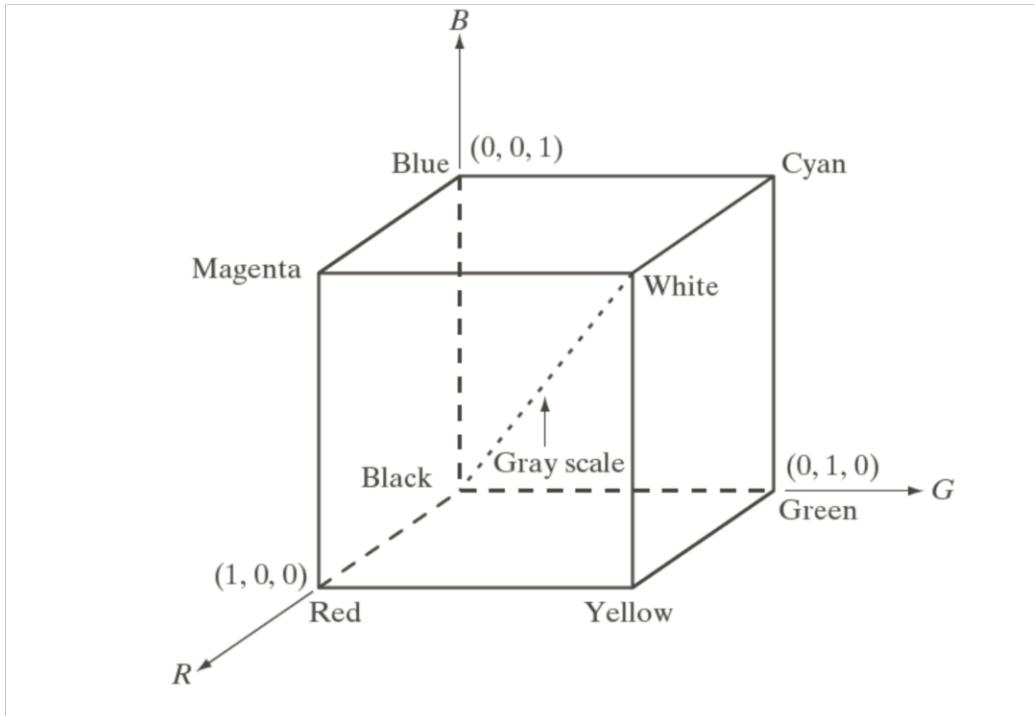
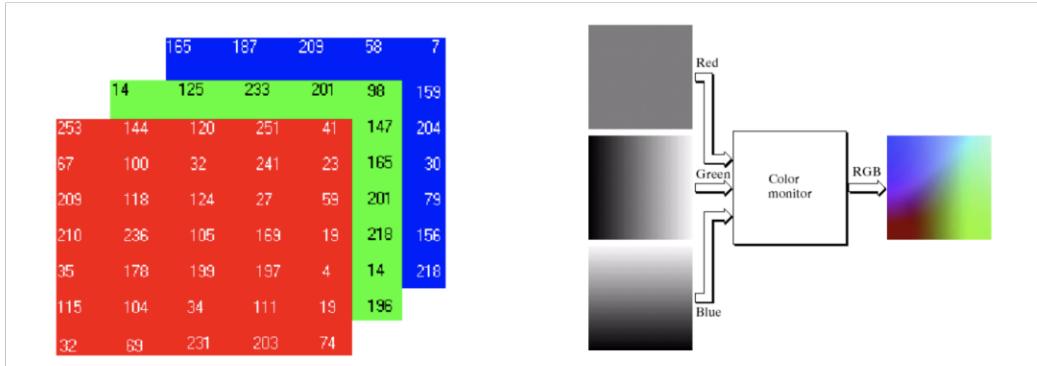


Figura 1.11. Spazio colore RGB.

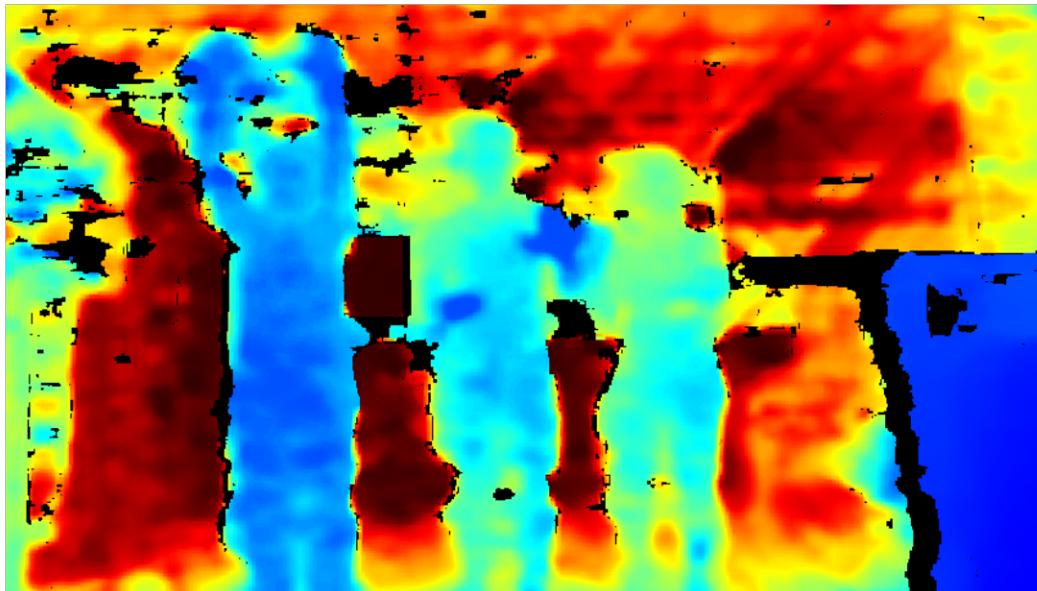
modello RGB viene mostrato in figura 1.11.

Le immagini rappresentate nel modello RGB sono formate da tre immagini, una per ogni colore primario; come mostrato in figura 1.12. Quando vengono visualizzate in un monitor RGB, queste tre immagini si combinano per produrre un'immagine a colori composta. Dal punto di vista dell'accesso in memoria, è possibile sia accedere a ognuna delle tre immagini, che all'immagine a colori nel suo complesso. Invece, dal punto di vista dell'elaborazione delle immagini e dei risultati ottenuti, in alcuni casi l'output di operazioni ripartite sui singoli canali differiscono dall'output dell'operazione effettuata sull'immagine intera [3].

Un'immagine RGBD è semplicemente un'immagine RGB combinata con una corrispondente immagine di profondità. Un'immagine di profondità è un'immagine a un canale singolo dove per ogni pixel si conserva l'informazione sulla distanza tra il piano immagine e il corrispondente oggetto nella realtà. Uno dei dispositivi storici utilizzati per catturare immagini RGBD è il Kinect [76]. Al fine di ottenere l'informazione sulla profondità in maniera accurata, senza fare ricorso a tecniche di machine learning e intelligenza ar-

**Figura 1.12.** Immagine RGB.

tificiale e senza fare ricorso alla tecnologia Raytrix [77] o tecnologie analoghe che sfruttano una solo videocamera in modo molto particolare, sono necessarie almeno due videocamere. A partire da queste videocamere, è possibile fare ricorso a nozioni di geometria epipolare, trigonometria e fisica, in modo da ricavare la profondità per ogni pixel dell'immagine acquisita. In figura 1.13, è mostrata un'immagine RGBD acquisita durante alcuni test condotti presso il laboratorio “Alfredo Petrosino - CVPR Lab” [70].

**Figura 1.13.** Immagine RGBD.

In un'immagine contenente informazioni sulla profondità degli elementi inquadrati nella scena, la visualizzazione è semplificata dal particolare uso del colore. Infatti, nel caso delle telecamere Intel RealSense D435, come si vede nella figura 1.13, si usa la cosiddetta mappa di profondità tale per cui colori caldi (e.g. giallo, arancione, rosso) indicano elementi lontani; viceversa colori freddi (e.g. azzurro, blu, viola) indicano elementi vicini. Elementi contraddistinti dal colore verde sono invece a una distanza media, dove il concetto di medio è da intendersi rispetto alla calibrazione iniziale effettuata dalla telecamera. Infine, laddove sono visibili zone di colore nero, queste intendono indicare zone ove non è stato possibile valutare la distanza o zone di colore nero ai fini interpretativi della profondità (tipicamente in presenza dei bordi di persone e oggetti). Si noti come la persona più a sinistra sia caratterizzata da un colore più freddo, mentre quella più a destra sia caratterizzata dall'avere un colore più caldo. Questo indica semplicemente che la prima persona menzionata è più vicina alla telecamera, a differenza della seconda che è più lontana.

1.4 Aspetti teorici

1.4.1 Profondità e occhio umano

Il sistema visivo umano ricava informazioni sulla profondità e sulla distanza da una varietà di fonti. Queste fonti si distinguono in fonti primarie, che derivano dai meccanismi fisiologici di accomodazione, convergenza e stereopsi e fonti secondarie (o psicologiche) [4]. Nella cultura popolare, la percezione dello spazio tridimensionale (3D) è spesso associata alla stereovisione. Tuttavia, la stereopsi è solo uno dei tanti spunti di profondità che informano il nostro sistema visivo [5], infatti, altri spunti sono disponibili anche quando si guardano immagini con un solo occhio aperto (monoculare).

Le fonti primarie sono l'accomodazione, la convergenza e la stereopsi (cioè la disparità binoculare) [78]. L'accomodazione è quel meccanismo dell'apparato visivo che consente di mettere a fuoco spostando lo sguardo da un oggetto vicino a uno lontano e viceversa. Quando la messa a fuoco non avviene in modo istantaneo, si presuppone che ci sia un problema di accomodazione [79]. La convergenza è la rotazione dei nostri due occhi, necessaria per focalizzarsi sullo stesso bersaglio. L'angolo di percezione degli occhi è correlato alla distanza tra l'osservatore e il bersaglio, come si nota nella figura 1.14. In quest'ultima figura si mostrano graficamente i concetti appena menzionati relativi ad accomodazione e convergenza visiva.

La stereopsi, di fatto, è un'evoluzione della visione binoculare e rappresenta la capacità di apprezzare la profondità delle immagini. Ciò accade quando quello che percepisce l'occhio destro differisce in maniera minima rispetto a quello che percepisce l'occhio sinistro, non creando diplopia, ovvero visione doppia, ma consentendo di vedere la tridimensionalità dell'oggetto. Questa differenza si definisce disparità, ed è misurata in secondi di arco: quanto più piccola è la disparità che si riesce a percepire, tanto migliore sarà la capacità d'individuare minime variazioni di profondità. La stereopsi, fondamentale in molte attività quotidiane come la lettura, lo sport e la guida, si sviluppa entro l'anno di vita per poi continuare ad affinarsi fino all'ottavo anno di età [80].

Un'altra fonte primaria è la parallasse di movimento, che utilizza le trasformazioni dell'immagine derivanti dal movimento dell'osservatore. Durante il movimento, infatti, gli oggetti vicini si muovono più velocemente di quelli più lontani. Più qualcosa appare lontano, più lentamente verrà percepito dall'osservatore.

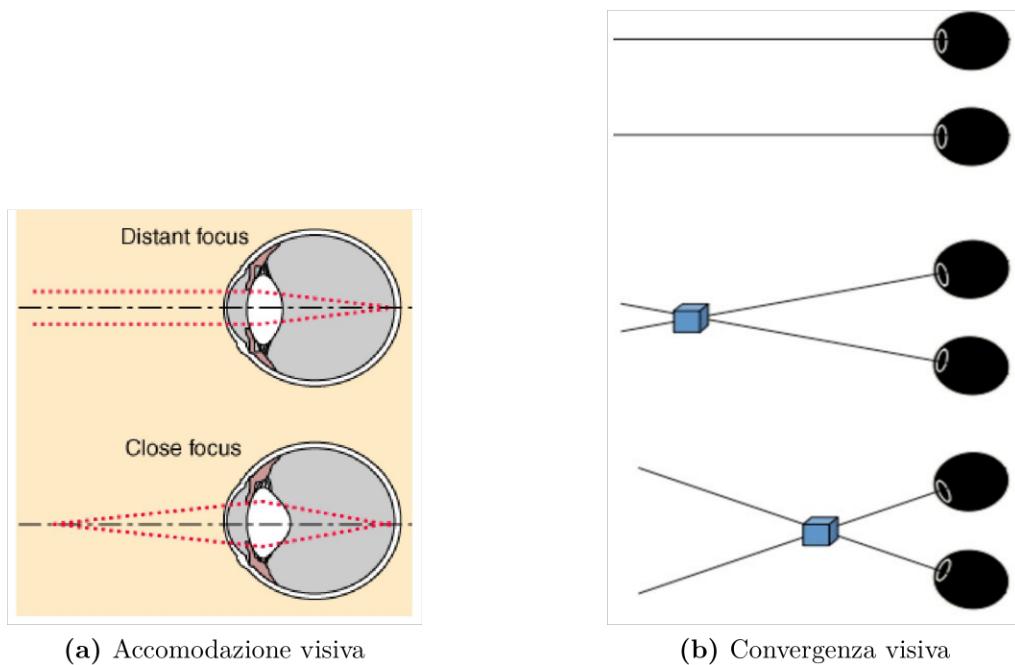


Figura 1.14. Accomodazione e convergenza visiva. (a) Nell'immagine di sinistra è possibile visionare il concetto di accomodazione visiva. Per spostare l'attenzione dell'occhio da un bersaglio lontano (in alto) a un bersaglio vicino (in basso), i muscoli dell'occhio che controllano lo spessore della lente ricevono segnali efferenti [5]. (b) Nell'immagine di destra è possibile visionare il concetto di convergenza visiva: l'angolo di convergenza aumenta man mano che gli occhi spostano lo sguardo da un obiettivo lontano (in alto) a obiettivi progressivamente più vicini (in mezzo e in basso) [5].



Figura 1.15. Dimensioni dell’immagine e percezione della profondità. Se coerente con altri spunti di prospettiva lineare (a), la dimensione dell’immagine è un forte spunto per capire la profondità dell’oggetto. Se si osserva l’oggetto isolatamente (b), le dimensioni dell’immagine diventano più ambigue, anche con oggetti di dimensioni note o simili [5].

Le fonti secondarie sono prettamente psicologiche e sono la dimensione dell’immagine retinica, la prospettiva lineare, la trama dell’oggetto, l’occlusione, la prospettiva aerea, le sfumature e le ombre. La dimensione dell’immagine retinica è un potente indizio quando si considerano elementi di dimensioni identiche note o presunte, ma può essere ambigua, se considerata come fonte isolata. Un esempio è visibile nella figura 1.15.

La prospettiva lineare, cioè le linee parallele convergenti all’orizzonte e l’altezza nel piano dell’immagine sono correlate alla profondità. Strade e piste creano trapezi in fotografie di ambienti normali, e la base di un oggetto più vicino è più lontana dall’orizzonte rispetto alla base di uno lontano. Le tessiture forniscono una scala continua della profondità. La trama superficiale di un oggetto diventa più dettagliata man mano che ci si avvicina a esso. Di conseguenza, gli oggetti con trama liscia sono tipicamente percepiti come più lontani. L’occlusione è un’altra potente indicazione visiva della profondità: gli oggetti più vicini ostruiscono gli oggetti più distanti lungo la linea visiva.

Per quanto riguarda le sfumature e le ombre, quando si conosce la posizione di una sorgente luminosa e si vedono oggetti che proiettano ombre su altri oggetti, il cervello è in grado di apprendere che l'oggetto che ombreggia l'altro è più vicino alla sorgente luminosa. L'ombreggiatura, di solito, dà alcune informazioni sui cambiamenti di profondità all'interno di una superficie o in un particolare regione dell'immagine. Infine, la prospettiva aerea (o atmosferica) è limitata agli oggetti visti a distanze abbastanza elevate; in quanto dipende dall'effetto che l'aria, tra l'osservatore e gli oggetti, ha sulla qualità dell'immagine [6].

1.4.2 Profondità e videocamere RGBD

Le videocamere capaci di determinare la profondità degli oggetti e delle persone presenti all'interno della scena inquadrata, hanno avuto negli ultimi tempi sempre più importanza, in quanto forniscono ai dispositivi la possibilità di vedere, capire, interagire e imparare dall'ambiente osservato. L'output prodotto da questa tipologia di telecamere è dato da immagini a colori i cui elementi, e più in generale regioni, con una stessa profondità all'interno dell'immagine hanno un colore simile, indipendentemente dal colore che possiedono all'interno della scena. È possibile migliorare la resa delle immagini catturate andando a calibrare la telecamera all'avvio, rispetto alla scena inquadrata e impostando determinati parametri in maniera sperimentale, in modo da avere dei miglioramenti sui singoli frame (e.g. riempimento buchi, riduzione rumore, potenza emissione laser, e così via). Inoltre, è possibile applicare tecniche di post-processing delle singole immagini acquisite, così da migliorare ulteriormente il risultato finale.

È utile fare una prima distinzione tra il concetto di “depth”, rispetto a quello di “range”. Il primo vocabolo fa riferimento alla distanza memorizzata all'interno dell'immagine, mentre il secondo fa riferimento alla distanza effettiva nella scena. Questi due valori possono differire a seconda di vari fattori: telecamera utilizzata, versione del firmware della telecamera, stabilità dell'installazione del supporto fisico, luci, ombre e altre specifiche della scena inquadrata.

Un aspetto fondamentale, al fine di determinare il valore di profondità per ogni pixel dell'immagine, è il concetto di riflettanza. In ottica, la riflettanza misura la capacità di riflettere parte della luce incidente su una data superficie o materiale. Essendo quindi il rapporto tra intensità del flusso radiante riflesso e intensità del flusso radiante incidente, è una grandezza adimensionale. Un oggetto che ha una bassa riflettanza produce immagini più rumorose e con maggiore incertezza sul valore della distanza dalla telecamera, anche detto “depth confidence”. Viceversa, oggetti con una riflettanza più elevata, permettono di avere meno rumore, qualità più elevata e informazioni sulla distanza più accurate. Queste due considerazioni sulla riflettanza possono creare dei problemi in situazioni particolari, in quanto spesso si sfruttano tecnologie laser o infrarossi al fine di dedurre la distanza. Una delle cause delle zone nere visibili nei frame contenenti informazioni sulla distanza, è proprio quella citata poc'anzi. Inoltre, a parità di distanza dalla telecamera che inquadra la scena, un oggetto con maggiore riflettanza sarà maggiormente definito e limpido nell'immagine risultante, a differenza di un oggetto con

minore riflettanza. Di conseguenza all'aumentare della distanza dalla telecamera, gli oggetti con riflettanza maggiore riescono a conservare meglio il grado di chiarezza col quale verranno rappresentati nelle immagini risultanti. Chiaramente, quando si parla di questi aspetti di basso livello, si fa riferimento a immagini reali e non a immagini sintetiche prodotte al computer.

L'informazione sulla profondità all'interno di un'immagine può tornare utile anche ai fini della segmentazione. Infatti, questo tipo di operazione è di gran lunga facilitata avendo a disposizione anche quest'ulteriore informazione. In linea generale alcune problematiche dell'elaborazione delle immagini sono più semplici se affrontate in RGBD, anziché in RGB; oppure è possibile affrontarle in maniera più efficace. Per esempio, applicare il deep learning su immagini RGBD, tendenzialmente porta a risultati migliori rispetto a usare semplici immagini RGB, perché la rete neurale non deve sforzarsi più di tanto per segmentare i singoli oggetti: basta trovare i pixel che fra di loro sono più vicini in termini di distanza e considerarli come appartenenti allo stesso oggetto.

Un aspetto importante da precisare è che le immagini catturate con una telecamera di profondità non saranno mai perfette, nemmeno con la visione stereoscopica di una potenziale telecamera. La motivazione è che non è possibile vedere "tutto" con i soli due "occhi" della telecamera di profondità. Per esempio, le regioni della scena ai lati non saranno visibili a seconda dell'ampiezza del campo visivo della telecamera. L'errore che si commette è lo stesso errore di quando si pone un dito davanti agli occhi e si cerca di vedere oltre. Di norma, l'effetto prodotto è quello di vedere due dita, anziché uno; ma la visione umana, aiutata dal cervello, unisce le due viste e ne vede uno soltanto. Una telecamera digitale, benché si possano usare algoritmi molto avanzati, non è in grado di farlo completamente e in ogni scenario. Di conseguenza, in alcuni casi si generano degli artefatti.

Inoltre, è possibile ricavare l'informazione sulla profondità per ogni pixel grazie alla tecnologia laser o infrarossi applicata alle telecamere. In entrambi i casi, qualora tali telecamere dovessero essere impiegate in presenza di luce diretta del Sole, la determinazione della distanza e la sua accuratezza potrebbero essere leggermente inibite. In ogni caso, per migliorare la precisione e integrare le misure delle distanze rilevate, si applicano delle nozioni importanti di geometria epipolare.

1.4.3 Cenni di geometria epipolare

La geometria epipolare è la geometria che lega due immagini acquisite da due punti di vista differenti. Le relazioni che intercorrono tra le immagini, tuttavia, non dipendono dalla scena osservata, ma dipendono solamente dai parametri intrinseci delle fotocamere e dalle posizioni relative. Per ogni punto osservato della scena, il piano epipolare è il piano formato da quel punto espresso in coordinate del mondo e dai due centri ottici. La geometria epipolare è quindi la geometria che descrive la visione stereoscopica (o binoculare); descrive le relazioni e i vincoli geometrici che legano due immagini bidimensionali della stessa scena tridimensionale, catturata da due fotocamere con posizione e orientamento distinti.

L'essere umano riesce a percepire la distanza grazie al fatto che ha a disposizione due occhi. Con un solo occhio è possibile ricavare qualche indizio, capire approssimativamente la distanza in base all'esperienza umana; ma la vera visione tridimensionale e la percezione della profondità avviene con entrambi gli occhi. Il cervello sa dove sono fisicamente situati gli occhi sul viso e quindi riesce a elaborare e capire come interpretare la scena osservata dagli occhi. Invece, nel caso di un dispositivo elettronico stereoscopico, quest'ultimo deve essere messo al corrente di dove sono situate le due telecamere. Si precisa che si parlerà di due telecamere, ma in realtà si fa riferimento a una sola telecamera con due sensori fotosensibili separati. I concetti sono applicabili anche nel caso di telecamere del tutto distinte, con le opportune osservazioni [5].

Si supponga di avere due telecamere situate in un cubo unitario, rivolte verso una delle pareti, con una determinata angolazione e posizionate a una certa altezza. Idealmente, la prima telecamera acquisisce un'immagine nello stesso istante in cui la seconda telecamera acquisisce un'altra immagine. Questo scenario è mostrato in figura 1.16. A questo punto è necessario trovare i punti corrispondenti tra le due immagini. Qualora non conoscessimo dove sono situate le telecamere all'interno del cubo, la complessità computazionale e lo spazio di ricerca, in generale, aumenterebbero vertiginosamente (e.g. angoli che appaiono più volte, punti confusi, sovrapposti e così via. Per esempio, un angolo potrebbe apparire più volte quando si inquadra un libro).

Un'ulteriore problematica è che ci saranno sezioni della scena visibili a una telecamera, ma non all'altra e magari si tratta di sezioni appartenenti a uno stesso oggetto. Quello che si fa inizialmente, in linea generale, è cali-

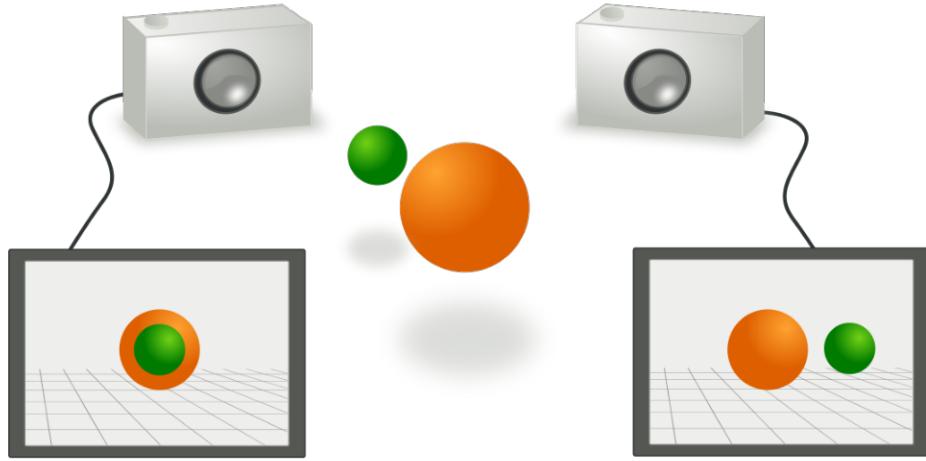


Figura 1.16. Due fotocamere che acquisiscono un'immagine della stessa scena idealmente nello stesso istante[81].

brare le due telecamere. Questo permette, date le due telecamere supposte poc'anzi, di capire il loro posizionamento nello spazio. Per fare ciò, si acquisisce un'immagine da entrambe le telecamere idealmente nello stesso istante, sebbene il concetto di contemporaneità esatta non sempre è possibile. In linea generale, è necessario che il tempo che intercorre tra le due acquisizioni sia il più contenuto possibile; altrimenti, la scena cambierebbe e non sarebbe possibile procedere con i passaggi successivi. A questo punto, si hanno a disposizione due immagini che sono dette “left view” e “right view”: rispettivamente l'immagine acquisita dalla telecamera sinistra, che verrà indicata come telecamera A , e quella acquisita dalla telecamera destra, che verrà indicata come telecamera B .

Come determinare la distanza per ogni pixel di un'immagine? Si consideri la figura 1.17: l'oggetto che si suppone di fotografare è quello che viene rappresentato dal punto X . Si supponga di avere a disposizione due fotocamere pin-hole: A e B . Asserire ciò, presuppone che tutti i raggi ottici che concorrono alla formazione dell'immagine siano “transitati”, a un certo tempo del loro cammino, per un unico punto, detto centro di proiezione. Il termine pin-hole nasce proprio dal fatto che si suppone che la luce che impressiona la lastra o i sensori di una fotocamera digitale, attraversi un foro di piccole dimensioni, tanto piccole da potersi considerare un punto nell'accezione geometrica del termine [78].

Le due fotocamere A e B , sono centrate rispettivamente nei punti O_L e O_R . Il punto X in questione verrà rispettivamente proiettato sul piano immagine della fotocamera sinistra in x_L e in x_R nel piano immagine della fotocamera destra. Si denota con a il segmento che congiunge il punto O_L con X e con b il segmento che congiunge il punto O_R con X . Il valore della distanza fornito dalla fotocamera A , potrebbe essere un valore qualsiasi lungo il segmento a od oltre. Analogamente per la fotocamera B con il segmento b . Si tratta di un problema non da poco. Si supponga quindi, di essere a conoscenza del fatto che un pixel nell'immagine acquisita da A è equivalente a un pixel nell'immagine acquisita da B . In tal caso, effettuando delle proiezioni e con l'aiuto della trigonometria, si ricava la profondità del punto nella scena. Purtroppo, non si ha l'informazione relativa al pixel di equivalenza tra le due immagini acquisite da A e B . Infatti, questo pixel può differire nel tempo, può non essere visibile a entrambe le fotocamere e così via. Per risolvere questo problema piuttosto complesso, trovare i punti di corrispondenza tra le immagini acquisite, estendere il discorso a ogni pixel e ricavare la profondità; si sfrutta la geometria epipolare [7].

Si consideri il punto O_L di A e il punto X nella scena inquadrata. Per rendere più semplice la ricerca nell'immagine di B e al fine di dedurre la profondità a cui si trova X , si prendono in considerazione questi due punti poc'anzi menzionati e il punto O_R di B , come appartenenti a un unico grande triangolo. Si immagini che dalla fotocamera B escano tanti segmenti che vanno a intersecarsi con il segmento a generando una serie di punti, che nell'immagine acquisita da B risultano essere presenti su un'unica retta. Questa retta è detta retta epipolare. Modellando il problema in questo modo, la difficoltà di risoluzione cala vertiginosamente. Infatti, sapendo dove sono situate le fotocamere e con l'informazione relativa alla retta epipolare, si deduce che la corrispondenza tra x_L e x_R , è possibile trovarla solo e unicamente sulla retta epipolare poc'anzi descritta. C'è quindi un insieme limitato di pixel da dover andare a controllare e fra questi determinare quello più simile a x_L , triangolarizzare come nell'approccio ideale e determinare di conseguenza l'informazione sulla profondità dell'oggetto nella scena. Questo tipo di approccio è possibile solo perché si è a conoscenza di dove sono situate le fotocamere; altrimenti, se così non fosse, sarebbe necessario cercare nell'intera immagine e ciò sarebbe costosissimo. Questo problema è definito come *problema di corrispondenza*.

Siccome nel caso di una telecamera Intel RealSense, di cui si discuterà nella sottosezione 3.1, le due telecamere sono allineate e ben calibrate all'avvio, il lavoro di triangolarizzazione per ricavare la profondità è sempli-

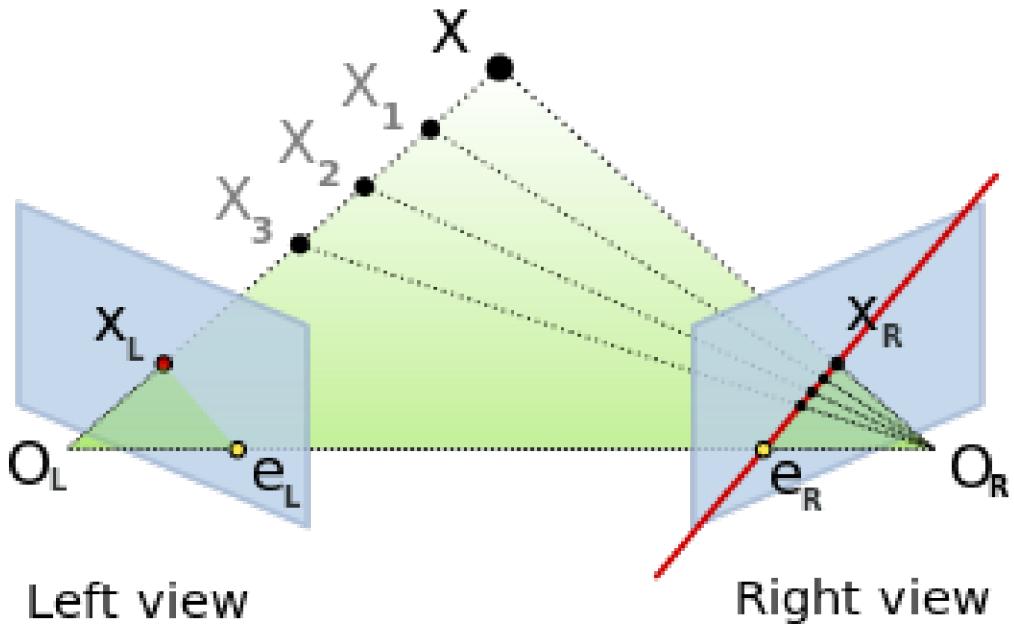


Figura 1.17. Determinazione della profondità per ogni pixel dell’immagine acquisita [81].

ficato. Inoltre, si precisa che durante il procedimento di triangolarizzazione, avvengono anche alcuni passaggi di smoothing per eliminare il rumore e altri dettagli meno rilevanti. In molti casi vengono applicate anche tecniche di post-processing per migliorare la qualità dell’immagine finale.

Quando si hanno due o più telecamere gli approcci maggiormente noti sono due: il primo approccio sfrutta la geometria epipolare. Questo modo permette una precisione e un’efficienza elevata. Il secondo approccio fa uso della minimizzazione delle somme delle distanze al quadrato (quantità nota anche come SSD). Quest’ultimo approccio, però, risulta essere abbastanza costoso e poco preciso. Di conseguenza non verrà considerato in questo documento. Infatti, l’aspetto fondamentale da tenere a mente quando è necessario lavorare con due telecamere, è quello di considerare le disparità tra i pixel delle due immagini acquisite e abbinarli in modo che corrispondano alla stessa regione, oggetto, persona. Tra tutte le possibili scelte, va fatta ovviamente quella che permette di avere le disparità più piccole tra tutti i pixel dell’immagine. Solo agendo in questo modo si riesce a ottenere un’immagine di profondità qualitativamente soddisfacente [8].

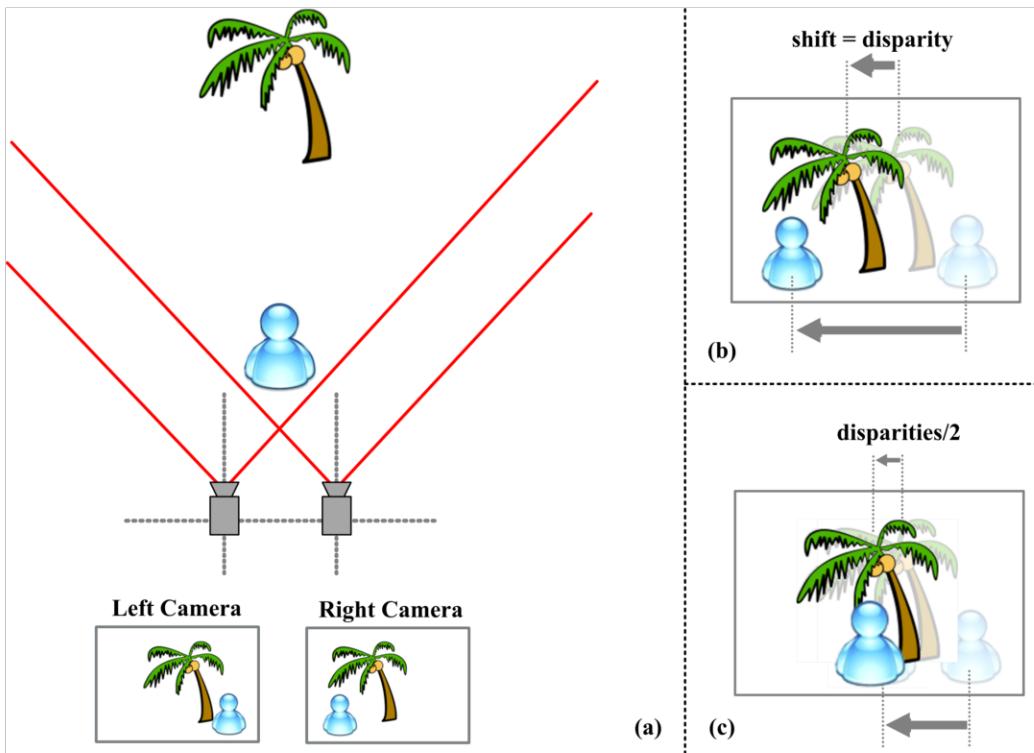


Figura 1.18. Stereo-matching e disparità immagini [81].

La figura 1.18 riassume in maniera grafica i concetti presi in considerazione finora.

1.4.4 Cenni di morfologia matematica

Il termine morfologia si riferisce di solito a quella branca della biologia che ha a che fare con la forma e la struttura di animali e piante. Si utilizza lo stesso termine nel contesto della morfologia matematica, come mezzo per estrarre le componenti di un'immagine utili per la rappresentazione e la descrizione della forma di regioni come bordi, scheletri e superfici convesse. Inoltre, ai fini di questa tesi, si porrà particolare attenzione alle tecniche morfologiche per realizzare pre- o post-elaborazioni come il filtraggio morfologico. Sebbene le tecniche che verranno analizzate sono adattabili e riformulabili per immagini a scala di grigi o a colori; per semplicità, si presenteranno assumendo di lavorare con immagini binarie.

La riflessione di un insieme B , indicata con \hat{B} , è definita nell'equazione 1.1.

$$\hat{B} = \left\{ w \mid w = -b, \text{ per } b \in B \right\} \quad (1.1)$$

Se B è l'insieme dei pixel (punti 2-D) che rappresentano un oggetto in un'immagine, allora \hat{B} è semplicemente l'insieme dei punti in B le cui coordinate (x, y) sono state sostituite da $(-x, -y)$. La traslazione di un insieme B tramite un punto $z = (z_1, z_2)$, denotata da $(B)_z$, viene definita nell'equazione 1.2.

$$(B)_z = \left\{ c \mid c = b + z, \text{ per } b \in B \right\} \quad (1.2)$$

Se B è l'insieme dei pixel che rappresentano un oggetto in un'immagine, allora $(B)_z$ è l'insieme dei punti in B le cui coordinate (x, y) sono state sostituite da $(x + z_1, y + z_2)$. La riflessione e la traslazione di un insieme vengono ampiamente impiegate in morfologia matematica per formulare operazioni basate sui cosiddetti elementi strutturanti (SE, Structuring Elements): piccoli insiemi o sottoimmagini usati per esplorare (sondare) un'immagine riguardo alle proprietà d'interesse.

Se A e B sono insiemi in Z^2 , la dilatazione di A attraverso B , indicata come $A \ominus B$, viene definita nell'equazione 1.3.

$$A \oplus B = \left\{ z \mid (\hat{B})_z \cap A \neq \emptyset \right\} \quad (1.3)$$

L'equazione 1.3 si basa sulla riflessione di B rispetto alla sua origine e sulla traslazione di questa riflessione attraverso z . Quindi, la dilatazione di A attraverso B è l'insieme di tutti gli spostamenti z , tali che B e A si sovrappongano almeno per un elemento.

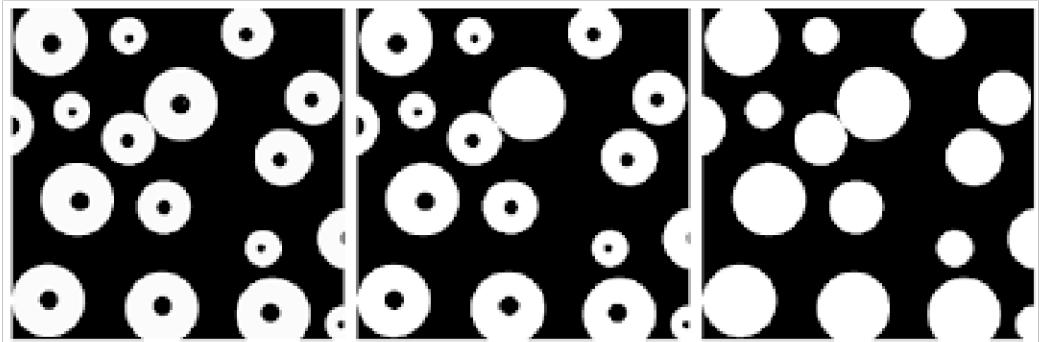


Figura 1.19. Illustrazione del graduale riempimento di vuoti durante l'esecuzione dell'algoritmo [3].

Un vuoto (hole) può essere definito come una regione di sfondo circondata da un bordo connesso di pixel del foreground (primo piano). Si fa uso di un algoritmo basato su dilatazione, complemento e intersezione per il riempimento di tali vuoti in un'immagine. Si assuma che A indichi un insieme i cui elementi siano bordi 8-connessi, ognuno dei quali racchiude una regione di sfondo (cioè, un vuoto). Dato un punto in ciascun vuoto, l'obiettivo è riempire tutti i vuoti (con valori 1). Per iniziare, si forma una matrice, X_0 , di valori 0 (della stessa dimensione della matrice contenente A), eccetto che per le posizioni in X_0 corrispondenti al dato punto in ogni vuoto, che invece vengono posti a 1. La procedura riportata nell'equazione 1.4 riempie tutti i vuoti con valore 1.

$$X_k = (X_{k-1} \oplus B) \cap A^C \text{ per } k = 1, 2, 3, \dots \quad (1.4)$$

Nell'equazione 1.4 si indica con B l'elemento strutturante. L'algoritmo termina all'iterazione k se $X_k = X_{k-1}$. L'insieme X_k contiene, dunque, tutti i vuoti riempiti. L'unione degli insiemi X_k e A contiene tutti i vuoti riempiti e i loro bordi. In figura 1.19, si mostrano i passi dell'algoritmo poc'anzi descritto.

Per maggiori informazioni in merito alla morfologia matematica, consultare il seguente riferimento al Capitolo 9 [3]. Infine, per avere un'idea operativa dei vuoti riscontrabili in un'immagine, è possibile visionare la figura 1.13. In quel caso si tratta di un'immagine RGBD e tipicamente questo tipo di immagini presentano tali artefatti in corrispondenza di zone d'incertezza, di confusione o al fine di rappresentare la profondità nell'immagine.

1.4.5 Intelligenza artificiale

L'intelligenza artificiale si propone di sviluppare delle macchine dotate di capacità autonome, di apprendimento, di adattamento e che siano ispirate ai modelli di apprendimento umani. Diversi anni fa ci furono delle scoperte che, nonostante un periodo di scoraggiamento, hanno rivoluzionato il mondo della tecnologia. Da sempre l'uomo ha cercato di costruire macchine pensanti, prendendo spunto dal funzionamento del cervello umano. Le unità responsabili del passaggio d'informazioni nel cervello sono i neuroni e in uno studio del 1943, Warren Sturgis McCulloch e Walter Pitts, introdussero un neurone artificiale [9]. Tale studio schematizzò un combinatore lineare a soglia con dati binari in entrata e un singolo dato binario in uscita. Le prime ipotesi di apprendimento furono introdotte dallo psicologo canadese Donald Olding Hebb, che propose un modello di neurone basato sul funzionamento complesso dei neuroni del cervello umano [10]. Il primo schema di rete neurale per il riconoscimento e classificazione di forme, antesignano delle attuali reti neurali, è da attribuire a Frank Rosenblatt [11]. Esso fornì un'interpretazione dell'organizzazione generale dei sistemi biologici ed era in grado di apprendere in modo simile a una rete neurale, rafforzando o indebolendo le connessioni tra neuroni vicini e interconnessi. Inoltre, utilizzava funzioni booleane linearmente separabili. L'interesse e l'euforia di tali studi, tuttavia, nel 1969 furono notevolmente ridimensionate da Marvin Minsky e Seymour A. Papert, i quali mostrarono i limiti operativi dell'apprendimento delle semplici reti a due strati basate sul percepitrone [12].

A causa di queste limitazioni, seguì un periodo di diffidenza durante il quale tutte le ricerche in questo campo, non ricevettero più alcun finanziamento dai governi. Ciò portò a far ristagnare la ricerca per oltre un decennio. Nonostante molti anni prima del 1982, il matematico Paul Werbos nella sua tesi dimostrò come addestrare le reti MLP (Multi-Layers Perceptron), solo l'intervento di John Hopfield, nel suo studio sui modelli di riconoscimento di pattern molto generali, riaprì degli spiragli per la ricerca nel campo dell'intelligenza artificiale. Pochi anni più tardi, David E. Rumelhart, Geoffrey Hinton e Ronald J. Williams introdussero uno dei metodi più noti ed efficaci per l'addestramento delle reti neurali: il cosiddetto algoritmo di retropropagazione dell'errore (error backpropagation). Esso modifica sistematicamente i pesi delle connessioni tra i nodi, così che la risposta della rete si avvicini sempre di più a quella desiderata. Tale lavoro fu prodotto riprendendo il modello creato da Werbos [13]. L'addestramento di una rete neurale avviene in due diversi fasi: forward-pass e backward-pass. Nella prima fase, i vettori in input sono elaborati dai nodi in ingresso con una propagazione in avanti

dei segnali attraverso ciascun livello della rete. Durante questa fase i valori dei pesi della rete sono tutti fissati. Nella seconda fase la risposta della rete viene confrontata con l'output desiderato ottenendo l'errore di classificazione. Questo errore è propagato nella direzione inversa rispetto a quella del passo forward-pass. Infine, i pesi vengono modificati in modo da minimizzare la differenza tra l'uscita attuale e l'uscita desiderata. Attraverso questo algoritmo si consente di superare le limitazioni del modello introdotto da McCulloch e Pitts. Infatti, viene risolto il problema della separabilità non lineare.

1.4.6 Il percettore

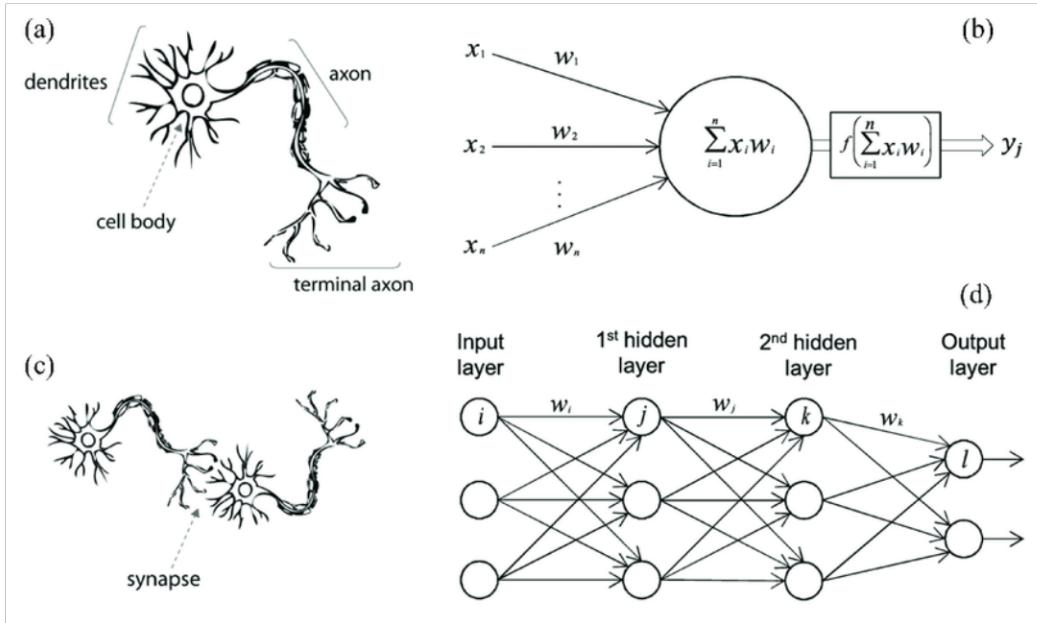


Figura 1.20. Un neurone biologico confrontato con una rete neurale artificiale: (a) neurone umano; (b) neurone artificiale; (c) sinapsi biologica; (d) sinapsi ANN [14].

I neuroni artificiali sono le unità di base delle reti neurali artificiali. La loro struttura cerca di somigliare quanto più possibile ai neuroni biologici. Questi neuroni artificiali ricevono in input degli stimoli e durante l'elaborazione vengono moltiplicati per un opportuno valore detto *peso*. Il risultato delle moltiplicazioni viene sommato e se la somma supera una certa soglia, il neurone si attiva restituendo un output. Questo peso serve a quantificare l'importanza di uno stimolo rispetto agli altri. Per esempio, se più neuroni comunicano fra loro e solo alcune connessioni vengono maggiormente utilizzate, allora tali connessioni avranno un peso maggiore.

$$f(x) = k \left(\sum_{i=1}^m w_i x_i \right) \quad (1.5)$$

L'equazione 1.5 si riferisce al neurone introdotto da McCulloch e Pitts e si può osservare la funzione di attivazione k , la quale permette di eccitare o inibire l'informazione in transito all'altro neurone [9]. Le quattro funzioni di attivazione più utilizzate sono le seguenti. Si precisa che w_i rappresenta il peso i -esimo, mentre x_i l'input i -esimo.

- La funzione soglia.

$$Y = \begin{cases} 0, & \text{se } \sum_{i=1}^m w_i x_i < 0 \\ 1, & \text{se } \sum_{i=1}^m w_i x_i \geq 0 \end{cases} \quad (1.6)$$

- La funzione sigmoide.

$$Y = \frac{1}{1 + e^{(-\sum_{i=1}^m w_i x_i)}} \quad (1.7)$$

- La funzione ReLU.

$$Y = \max \left\{ 0, \sum_{i=1}^m w_i x_i \right\} \quad (1.8)$$

- La funzione tangente iperbolica.

$$Y = \frac{1 - e^{-2 \sum_{i=1}^m w_i x_i}}{1 + e^{-2 \sum_{i=1}^m w_i x_i}} \quad (1.9)$$

Se si considerano più livelli di neuroni, si crea la rete MLP [13]. La retropropagazione dell'errore, in inglese *back-propagation of error*, ma solitamente abbreviato in backpropagation, è un algoritmo per l'addestramento delle reti neurali artificiali, usato in combinazione con un metodo di ottimizzazione, per esempio il gradiente discendente. Gli output desiderati per ogni input della rete, sono contenuti solitamente in una *ground truth* e in base agli output ottenuti dalla rete, si calcola il gradiente. Questa operazione viene maggiormente utilizzata nei metodi di apprendimento supervisionato, sebbene venga anche utilizzata in metodi non supervisionati. Inoltre, la retropropagazione richiede che la funzione d'attivazione usata dai neuroni artificiali sia differenziabile. Una delle principali difficoltà quando si utilizza il gradiente, è il problema noto come *scomparsa del gradiente*. Esso è dovuto alla presenza di funzioni di attivazione non lineari che causano una diminuzione esponenziale del valore del gradiente all'aumentare della profondità della rete neurale. Il funzionamento del gradiente discendente è molto semplice: esso permette di trovare un minimo locale di una funzione in uno spazio a N dimensioni. Ottimizzando i parametri del modello, esso minimizza la funzione *costo* e aumenta l'accuratezza del modello apprendendo le combinazioni non lineari delle caratteristiche in input.

1.4.7 Artificial Neural Network

Una rete neurale artificiale imita la struttura del cervello umano basandosi su neuroni artificiali disposti a strati e collegati tra di loro mediante l'invio di segnali. Dato un neurone artificiale dello strato i -esimo, vanno definiti un peso, una soglia, un certo numero d'input e un certo numero di output. L'approccio classico è quello di una rete *feed forward*, in cui l'output dei neuroni di uno strato della rete costituisce l'input dei neuroni dello strato successivo. Le ANN sono caratterizzate dall'avere un'elevata accuratezza, migliorano col tempo e sono ideali quando il fattore tempo è fondamentale nel dominio applicativo considerato.

Nel caso classico i neuroni di una ANN sono i percetroni, mentre in altri contesti si possono adottare i neuroni sigmoidi, ovvero dei neuroni che hanno valori compresi fra zero e uno sia per l'input che per l'output, anziché assumere soltanto valore zero oppure uno. Tale tipo di neuroni sono utili quando il problema neurale è non lineare. La funzione di accuratezza misura quanto si adatta (o generalizza) il modello scelto. La convergenza della funzione di accuratezza è raggiunta quando si raggiunge un punto di minimo, generalmente mediante il metodo del gradiente discendente. Un'ANN fa ampio uso della back-propagation analizzata nella sottosezione 1.4.6.

1.4.8 Convolutional Neural Network

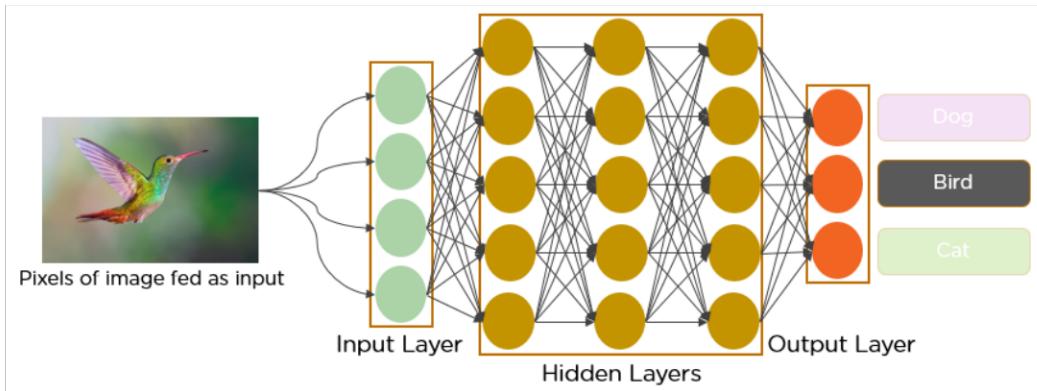


Figura 1.21. Rete neurale convoluzionale [82].

Le reti neurali convoluzionali (o convolutive) dette CNN (Convolutional Neural Network), sono identificate dai loro loop di feedback. Ci si avvale di questo approccio di apprendimento soprattutto quando si utilizzano dati di serie temporali per fare delle previsioni su risultati futuri (e.g. mercato azionario o previsioni prezzo di vendita). Alla base di una CNN vi è il concetto di convoluzione tra funzioni. Si può immaginare una CNN di base come formata da un livello convoluzionale, un livello di pooling e un livello completamente connesso. In una CNN si fa ampio uso di conversioni e filtri per analizzare gli input forniti di volta in volta.

Il campo che ha reso importante il deep learning è quello della computer vision. Tale successo è dovuto all'introduzione, da parte di Lecun nel 1998, della rete neurale convoluzionale [15]. Questo studio ha permesso di ottenere risultati ottimali, perché consente alle macchine di vedere o percepire il mondo come fanno gli esseri umani in una moltitudine di attività. Nella figura 1.21 viene mostrato un esempio dell'architettura di una rete neurale convoluzionale. L'input della CNN è un'immagine da cui si apprendono i pesi e i bias (o threshold), differenziando vari aspetti degli oggetti contenuti in essa, al fine ultimo di classificare l'intera immagine. Il preprocessing richiesto in una CNN è molto inferiore rispetto a quello che viene eseguito negli algoritmi di machine learning. Infatti, in questi ultimi, i filtri sulle immagini sono progettati a mano per il caso specifico, mentre con un buon addestramento la CNN in esame riesce ad apprendere filtri in grado di estrarre determinate caratteristiche discriminanti.

L’immagine è una matrice di pixel contenenti dei valori e la CNN è in grado di catturare con successo le dipendenze spaziali e temporali in un’immagine attraverso l’applicazione di filtri pertinenti. Una componente principale di questa rete è il layer di convoluzione. Esso permette di eseguire sull’immagine l’operazione di convoluzione attraverso un kernel, ovvero una matrice. Nel caso l’immagine fosse a più canali, per esempio RGB, il kernel ha la stessa profondità dell’immagine e tutti i risultati vengono poi sommati con il bias. Convenzionalmente, il primo layer di convoluzione è responsabile dell’acquisizione delle caratteristiche di basso livello, ad esempio: i bordi, il colore, l’orientamento del gradiente e così via. Nei successivi layer, l’architettura si adatta anche alle caratteristiche di alto livello. Oltre al layer di convoluzione, un altro layer ha molta importanza nella rete CNN: il layer di pooling. Esso è responsabile della riduzione delle dimensioni spaziali dell’immagine con l’obiettivo di diminuire la computazione richiesta per elaborare i dati (in altri casi si va a ridurre il numero di *feature*). Inoltre, questo livello è utile anche per estrarre le caratteristiche dominanti che sono invarianti sia per rotazione, che per posizione. Questi due layer insieme formano l’i-esimo strato di una rete neurale convoluzionale e il loro numero può essere aumentato a seconda della complessità delle immagini acquisendo così ulteriori dettagli di alto livello, ma bisogna pagare il costo di una maggiore complessità di calcolo in quanto aumentano i parametri. Dopo aver eseguito i vari livelli del modello CNN, l’output prodotto si appiattisce e viene inserito in una normale rete neurale MLP per scopi di classificazione [16].

In una rete neurale tipica del deep learning [17], le immagini RGB vengono immesse nella rete a strati singoli. A questo punto nel primo layer della rete si possono effettuare alcune operazioni quali: una convoluzione per unire le informazioni, operazioni di filtraggio, segmentazione, determinazione di fattori decisionali e così via. Alcune di queste operazioni, alternativamente, vengono effettuate negli strati successivi della rete. Nel caso d’immagini RGBD l’approccio è analogo, ma ora vi è un altro strato relativo alla profondità per ogni pixel dell’immagine. Alla rete non interessa che ci siano tre o quattro canali: semplicemente si rimodula il modo di agire del primo livello della rete che prenderà in input quattro canali, anziché tre. In altri casi la rete neurale ha due *primi livelli*: in uno dei due si prendono in input le informazioni relative a RGB e nell’altro solo l’informazione relativa alla profondità. L’approccio da scegliere dipende dalle performance ottenute, dalle risorse a disposizione, dal tempo impiegato per un determinato dominio applicativo e così via.

1.4.9 Non Maximum Suppression (NMS)

La Non Maximum Suppression (NMS) è una tecnica utilizzata in numerose attività di computer vision. È una classe di algoritmi per selezionare un'entità (e.g. bounding box) da molte entità sovrapposte. È possibile scegliere i criteri di selezione per arrivare ai risultati desiderati. I criteri sono solitamente una qualche forma di misura della probabilità e una qualche forma di misura di sovrapposizione. In figura 1.22 si mostra un esempio di applicazione della NMS.

La maggior parte degli algoritmi di rilevamento di oggetti utilizza la NMS per ridurre il numero dei molti, se non troppi, bounding box rilevati. Di base, la maggior parte dei rilevatori di oggetti genera una qualche forma di finestra. Vengono generate migliaia di finestre di varie dimensioni e forme. Queste finestre presumibilmente contengono un solo oggetto, e un classificatore viene utilizzato per ottenere una probabilità/punteggio per ogni classe. Se l'algoritmo di rivelazione emette un gran numero di caselle di delimitazione è necessario filtrare quelle che sono maggiormente significative. Adottare NMS è pratica comune per questo compito.



Figura 1.22. Applicazione della NMS [83].

1.4.10 Cenni sui Grafi

Un grafo orientato G è una coppia (V, E) , dove V è un insieme finito ed E è una relazione binaria in V . L'insieme V è detto *insieme dei vertici* di G e i suoi elementi sono detti *vertici*. L'insieme E è detto *insieme degli archi* di G e i suoi elementi sono detti *archi*. La figura 1.23b mostra un grafo non orientato, mentre la figura 1.23a mostra un grafo orientato.

In un grafo non orientato $G = (V, E)$ l'insieme degli archi E è composto da coppie di vertici *non ordinate*, anziché da coppie ordinate. Ovvero, un arco è un insieme $\{u, v\}$, dove $u, v \in V$ e $u \neq v$. Per convenzione, si utilizza la notazione (u, v) per indicare un arco. Nel caso di un grafo non orientato, le notazioni (u, v) e (v, u) identificano lo stesso arco. In un grafo non orientato i cappi non sono ammessi; viceversa in un grafo orientato. Un cappio è un arco che entra ed esce nello stesso vertice. Di conseguenza in un grafo non orientato, ogni arco è composto esattamente da due vertici distinti. In figura 1.23c è mostrato un tipico grafo bipartito.

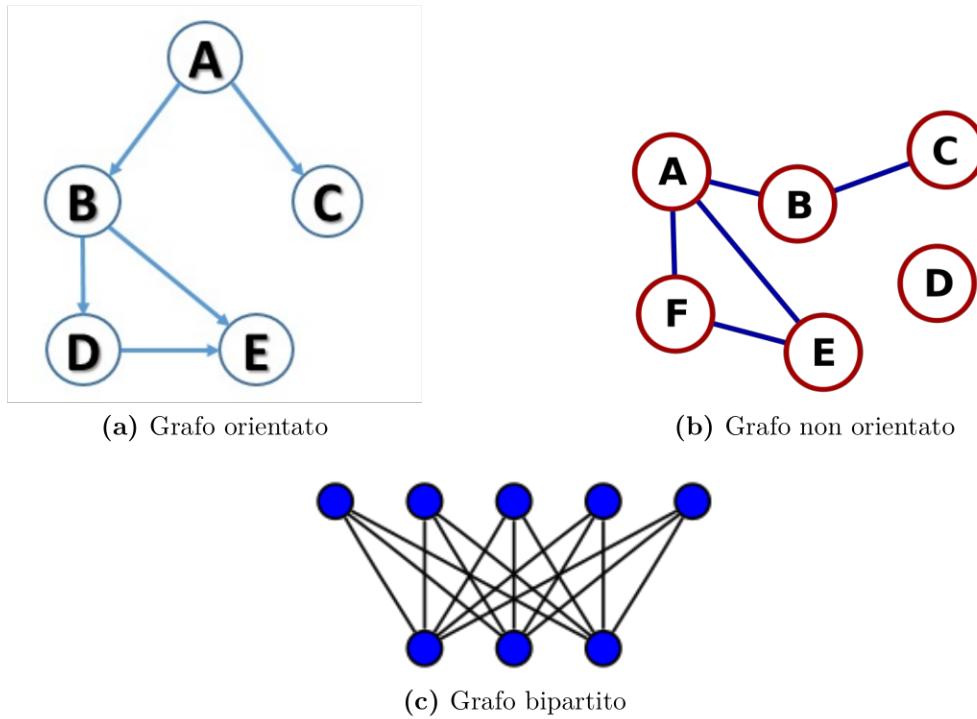


Figura 1.23. Esempi di grafi [84].

Per l'ambito di questa tesi saranno necessarie solo queste e altre due nozioni relative ai grafi: quella di grafo bipartito e quella di DAG. Un grafo

bipartito è un grafo non orientato $G = (V, E)$ in cui V può essere suddiviso in due insiemi V_1 e V_2 tali che $(u, v) \in E$ implica che $u \in V_1$ e $v \in V_2$ oppure che $u \in V_2$ e $v \in V_1$. Ovvero tutti gli archi stanno tra i due insiemi V_1 e V_2 .

È possibile sfruttare una variante della Breadth-first search (BFS) per verificare se un grafo è bipartito o meno. La proprietà del grafo bipartito può semplificare la risoluzione di alcuni problemi oppure essere richiesta come condizione di input del grafo G per un problema P .

Un DAG (Directed Acyclic Graph) è un grafo orientato che non presenta dei cicli al suo interno. In un grafo orientato un ciclo si forma nel momento in cui il primo e l'ultimo vertice, di un cammino considerato, coincidono. Infine, un grafo orientato G è aciclico se e soltanto se una visita in profondità (DFS) di G non genera archi all'indietro [18].

1.4.11 Algoritmi greedy

Gli algoritmi per i problemi di ottimizzazione, tipicamente, eseguono una sequenza di passi, con una serie di scelte a ogni passo. L'obiettivo è quello di arrivare a una soluzione ottima. In alcuni casi, è preferibile utilizzare algoritmi semplici ed efficienti detti algoritmi greedy. Un algoritmo greedy fa sempre la scelta che sembra ottima in un determinato momento, ovvero fa una scelta localmente ottima, nella speranza che tale scelta porterà a una soluzione globalmente ottima. Gli algoritmi greedy non sempre riescono a trovare le soluzioni ottime, ma per molti problemi sono in grado di farlo. Un semplice esempio è il problema del resto in monete: per minimizzare il numero di monete necessarie per formare un determinato resto, è sufficiente selezionare ripetutamente la moneta di taglio più grande, che non supera l'importo ancora dovuto. Ci sono vari problemi come questo, per i quali un algoritmo greedy fornisce una soluzione ottima più rapidamente di quanto sarebbe possibile con un metodo di programmazione dinamica. Tuttavia, questo non sempre è possibile e non sempre è facile dire se una tecnica greedy sarà efficace [18].

Capitolo 2

Progettazione

Questo capitolo illustrerà i passaggi principali che hanno condotto alla realizzazione del progetto AI Watch A1. In particolare, si inizierà prendendo in considerazione le principali sfide e problematiche che si sono presentate, si discuterà delle soluzioni adottate e delle motivazioni alla base di queste scelte. Infine, si illustreranno le seguenti tematiche: il flusso di esecuzione interno al modulo, l'organizzazione architettonica del modulo A1 e i design pattern utilizzati in fase di progettazione.

2.1 Depth Estimation

Il problema della *depth estimation*, anche noto come problema della stima della profondità, è definito come il compito di memorizzare un valore di distanza all'interno di ogni pixel dell'immagine acquisita e dove ogni pixel può far riferimento a un determinato elemento della scena osservata. La distanza memorizzata nel pixel è da intendersi come un'approssimazione della distanza che intercorre tra la telecamera e la parte dell'oggetto presente in quello specifico pixel.

L'obiettivo della stima della profondità è ottenere una rappresentazione della struttura spaziale di una scena, recuperando la forma tridimensionale e l'aspetto degli oggetti nelle immagini bidimensionali. Questo scenario è anche noto come problema inverso, ovvero un problema in cui si cerca di recuperare alcune incognite, una volta date delle informazioni insufficienti per specificare pienamente la soluzione. Questo significa che la mappatura tra la vista 2D e il 3D non è unica e si tratta quindi di un problema *mal posto* (ill-posed inverse problem) [19].

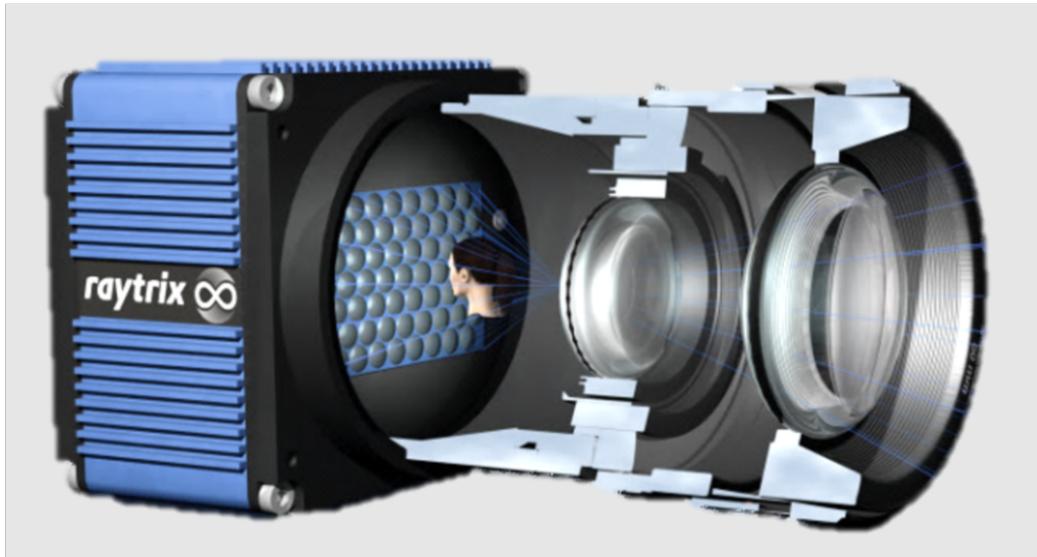


Figura 2.1. Tecnologia Raytrix [77].

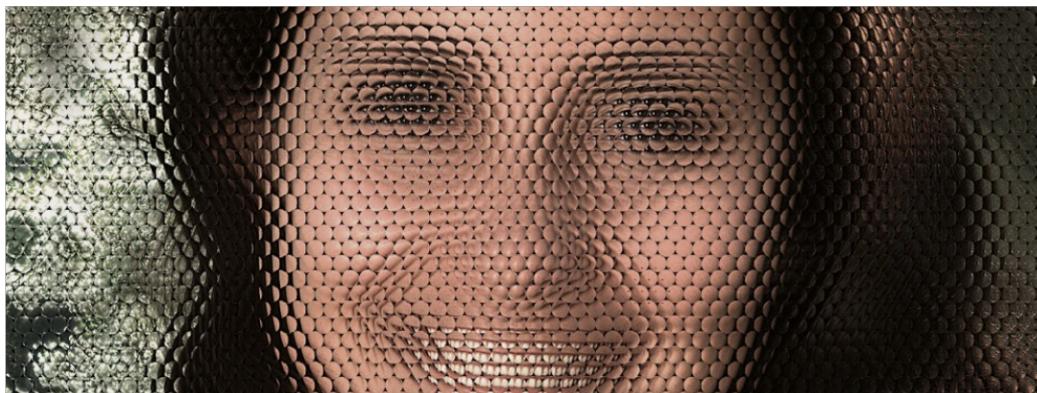


Figura 2.2. Vista della scena mediante microarray lens [77].

Esistono diversi approcci per ricavare la profondità degli oggetti e quindi dei pixel presenti in ogni immagine. Un primo modo è quello di utilizzare una telecamera in grado di determinare la profondità per ogni pixel (tali telecamere sono anche dette nel gergo *telecamere 3D*). A questa categoria appartengono le telecamere con tecnologia Intel RealSense [68, 20]. Vale la pena menzionare la presenza di altre telecamere in grado di adempiere compiti analoghi, come le telecamere Zivid [85], Revopoint [86], Orbbec [87] e Stereolabs [88].

Al fine di ricavare la profondità da un’immagine però, quello di utilizzare una *telecamere 3D* dedicata non è l’unico approccio. Quelle elencate finora sono tutte famiglie di telecamere e in generale tecnologie che fanno uso di una telecamera in cui sono presenti poi due *sotto-telecamere*, ovvero si basano sull’emulazione dell’apparato visivo umano. Alternativamente, è possibile utilizzare una singola telecamera con un sistema complesso e avanzato di lenti esagonali detto *Hexagonal Microarray Lens* che permette di raggiungere lo stesso risultato. Questo tipo di approccio è utilizzato dalle telecamere Raytrix [77]. Questo tipo di telecamere sono anche dette telecamere *lightfield*. Le telecamere *lightfield* sono un nuovo tipo di telecamere 3D che acquisiscono un’immagine standard insieme alle informazioni di profondità di una scena. Questo vuol dire che le informazioni sulla profondità possono essere acquisite anche con una singola telecamera di campo luminoso, attraverso un singolo obiettivo, in un singolo scatto e utilizzando la sola luce disponibile. Raytrix si è specializzata nello sviluppo di telecamere leggere e pratiche per applicazioni industriali. Usando la tecnologia *Hexagonal Microarray Lens*, visibile in figura 2.1, si consente un compromesso ottimale tra alta risoluzione e grande profondità di campo. L’idea è che ogni microlente esagonale dell’array vede l’immagine da un’angolazione leggermente differente, per poi utilizzare lo stesso approccio visto nelle telecamere stereoscopiche: lo stereo-matching mediante geometria epipolare.

Un terzo approccio utile per ricavare la profondità da un’immagine è quello di sfruttare sempre una sola telecamera, ma questa volta la si integra con l’ausilio dell’intelligenza artificiale e del machine learning [89, 21, 22]. Quello che si fa è utilizzare un modello specializzato a determinare la profondità nelle immagini grazie a una corposa fase di addestramento, per poi ottenere la profondità da immagini mai viste in precedenza. Questo approccio è quello che viene utilizzato da PoseNet [90], da MoveNet [91], da OpenPose [72] e dal noto Mediapipe [92].

2.2 Pose Estimation

Quella che in inglese viene definito come *pose estimation*, in italiano viene comunemente espresso come *stima della posa*; dove per *posa* si intende lo *skeleton* (o scheletro), formato da punti di giuntura detti *body key point* o *joint* e segmenti di giuntura chiamati anche *edge*. Tale struttura è stata illustrata in precedenza nella sezione 1.2.

La *pose estimation* è una tecnica di computer vision che permette di rilevare figure umane in immagini e video, in modo da determinare, per esempio, dov'è situato il ginocchio di una persona in un'immagine. Si precisa che questa tecnica non si occupa d'identificare l'identità della persona o delle persone presenti all'interno della scena inquadrata, ma solo di stimare dove sono le loro giunture fondamentali e di collegarle opportunamente per ogni corpo individuato in maniera opportuna, in modo da formare uno skeleton per ogni persona.

Per la *pose estimation* solitamente si fa uso di una rete neurale e la modalità di utilizzo classica consiste nel fornire in input un'immagine RGB. A questo punto nella rete neurale si costruirà una *heatmap image* e una *offset vector image*, dove la prima immagine indica i possibili punti di giuntura e la seconda immagine indica la direzione verso la quale è molto probabile che siano situati i punti di giuntura effettivi. Utilizzando le informazioni presenti in queste due immagini, la rete neurale è in grado di costruire lo skeleton necessario alla risoluzione del problema della *pose estimation*. I punti di giuntura rilevati sono indicizzati da un identificativo del punto di giuntura e viene assegnato loro un punteggio di *confidence* compreso tra 0,0 e 1,0. Un punteggio di *confidence* elevato, indica la probabilità molto alta che esista un punto di giuntura in quella posizione. L'approccio generico prevede quindi l'utilizzo dell'intelligenza artificiale. Esistono attualmente sia approcci di tipo supervisionato, che di tipo non supervisionato per la risoluzione di questo problema.

In figura 2.3 si mostra come questo approccio generico poc'anzi descritto, venga usato anche nel pratico, per esempio da PoseNet [90]. Chiaramente, vi sono degli aspetti molto dettagliati e avanzati che fanno differire PoseNet, così come OpenPose [72] o altro sistema, dagli altri; permettendo così di preferire una tecnologia anziché l'altra, per un determinato dominio applicativo. Tali aspetti non verranno presi in considerazione in questa pubblicazione, in quanto non rientrano nei suoi obiettivi.

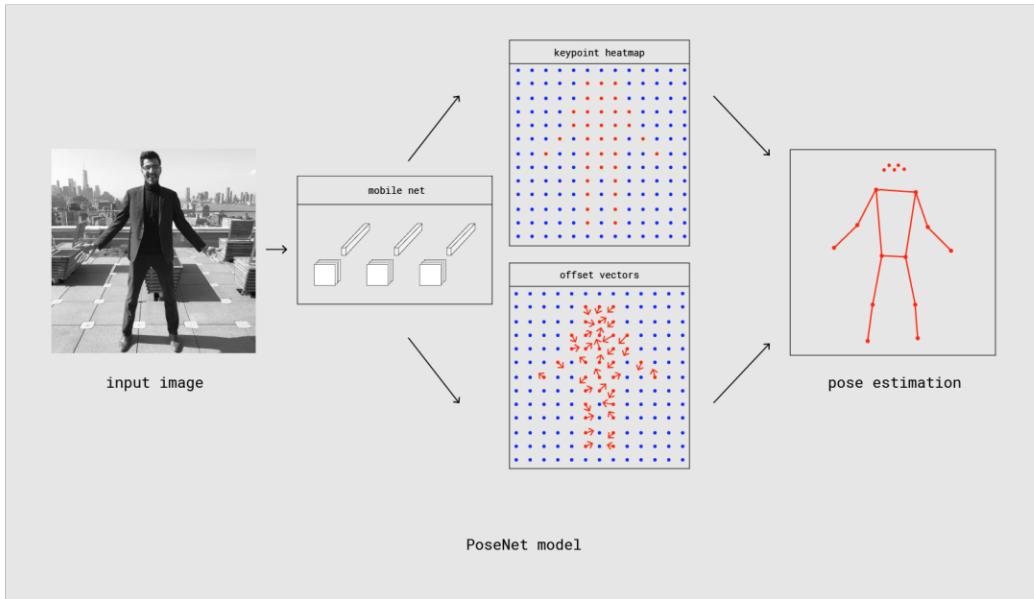


Figura 2.3. Processo di *pose estimation* in PoseNet. Data un’immagine in input, la rete neurale si attiva e ricava la *heatmap image* e l’*offset vector image*, al fine ultimo di determinare lo skeleton mostrato sulla destra [90].

Il riconoscimento delle attività umane, anche noto come HAR (Human Activity Recognition), è un’area di ricerca molto attiva nella comunità computer vision. Il riconoscimento dell’attività umana basato sulla visione si riferisce al compito di etichettare immagini o video che coinvolgono il movimento umano con diverse classi di attività. Enormi quantità di dati visivi vengono generati ogni giorno, e il riconoscimento delle attività li rende semanticamente significativi con l’annotazione automatica. Comprendere le attività è vitale in diversi aspetti della vita quotidiana come la sorveglianza, l’assistenza sanitaria, la navigazione autonoma, il rilevamento attivo, l’interazione uomo-macchina, e così via. Il secondo modulo del progetto è dedicato a quest’attività specifica [1]. Il problema del riconoscimento dell’attività umana è possibile affrontarlo in due modi:

- con metodi basati sulle *feature* (caratteristiche) con approccio manuale;
- con metodi basati sulle *feature* con approccio basato sul deep learning.

Nel primo dei due casi vi sono fasi tipiche di segmentazione, estrazione di caratteristiche fondamentali e una successiva classificazione. Se la telecamera è fissa, si usa e si sfrutta la sottrazione del background e il tracking. Invece, se la telecamera è in movimento si sfrutta il flusso ottico e le differenze

temporali. Analizzati brevemente questi aspetti, è interessante precisare che una delle tecniche usata sia dai metodi della prima categoria, che dai metodi della seconda categoria nell'ambito del riconoscimento dell'attività umana, è quella dello *skeleton-based activity recognition*. Siccome in un flusso video è possibile ritrovare azioni distinte, eterogeneità di scenari e aspetti più complessi da tenere in considerazione, sfruttare l'approccio con *skeleton-based activity recognition*, permette diverse semplificazioni strutturali e pratiche. Per esempio, permette di lavorare con grandi dataset mantenendo velocità elevate di esecuzione. Per ottenere uno skeleton esistono diversi approcci elencati qui di seguito.

- Analisi tridimensionale dei punti di giuntura (joint) del corpo con la loro orientazione. Tale analisi si basano sulla *heatmap image* e sulla *offset vector image* [23].
- Estrarre lo skeleton in maniera diretta a partire da immagini RGB mediante il *pose method* [24].
- Posizionamento e tracciamento dei movimenti a partire dalle giunture rilevate delle persone nella scena inquadrata mediante *marker* (segnaposti) [25].

Un'azione può essere definita come una conseguenza di un singolo o un insieme di cambiamenti, di movimenti compiuti da un agente umano. Di conseguenza, il riconoscimento dell'attività implica la rilevazione di sequenze di azioni. Pertanto, il riconoscimento dell'attività umana (HAR), si riferisce alla rilevazione di azioni eseguite da un soggetto con l'aiuto di osservazioni provenienti da diversi dispositivi sensoriali [26]. A seconda del movimento del sensore, esistono tre tipi di riconoscimento dell'azione: riconoscimento dell'attività di un singolo utente, multiutente e di gruppo [27].

Il sistema di visione basato su skeleton supera le attuali modalità di riconoscimento delle attività preservando la privacy degli utenti: comprende solo i valori di coordinate delle articolazioni specifiche, invece della rappresentazione visiva effettiva di qualsiasi istanza della vita reale. Inoltre, grazie alle semplificazioni che introduce e all'ingente disponibilità di dataset basati su questo sistema, risulta essere quello attualmente più utilizzato. È possibile trovare alcune differenze in merito alla struttura dello skeleton realizzato in sé: numero di punti di giuntura utilizzati, conformazione e così via. Si veda la figura 2.4 per un'idea al riguardo.

Dato un soggetto inquadrato nella scena, le fasi principali di un sistema di riconoscimento delle azioni basato sullo skeleton sono:

- skeleton tracking;
- rappresentazione digitale dello skeleton;
- preprocessing;
- feature extraction: con metodi manuali o con metodi basati sul deep learning;
- riconoscimento: basato sul machine learning o sul deep learning.

Inoltre, c'è da considerare che esistono diversi fattori che rendono complessa la realizzazione di un sistema di riconoscimento delle azioni basato su skeleton. Primo fra tutti, il problema della creazione dei dati, se si sceglie di usare dataset proprietari. Infatti, solitamente si acquisiscono dati a partire da un laboratorio appositamente equipaggiato e non da scenari di vita reale. Questo può portare a una bassa precisione nelle fasi successive, quando il sistema in fase di funzionamento ordinario, deve riconoscere le attività umane a partire da scenari di vita reale. Altre problematiche sono le dimensioni del corpo prese in considerazione e quelle inquadrate nella scena, l'orientazione dei corpi, la loro posizione ed eventuali variazioni di prospettiva dovute alla telecamera, background dinamici, qualità scarsa del materiale adottato, fattori dovuti a illuminazione e riflettanza, latenza elevata, ambiente inappropriato e così via.

Il tracciamento dello skeleton è la base fondamentale del riconoscimento dell'azione umana. Nel campo della ricerca si cerca di estrarre le traiettorie scheletriche o le *default pose* impiegando tecniche di tracciamento diverse. Sensori di profondità, toolbox come OpenPose e marker del corpo, sono alcune delle tecniche più importanti e avanzate nel tracciamento degli skeleton. Al fine di poter effettuare un tracciamento dello skeleton completo, è necessario avere anche l'informazione sulla profondità dei vari punti di giuntura, per ogni persona individuata nella scena osservata. Alcune delle soluzioni in commercio maggiormente utilizzate per le attività di skeleton detection e di skeleton tracking sono le seguenti.

- Sensori di profondità Kinect: tra l'ampia varietà di sensori di profondità disponibili, le fotocamere Kinect di Microsoft hanno guadagnato popolarità tra i ricercatori della HAR grazie alla loro capacità di catturare video RGB e di profondità in tempo reale [28].
- Intel RealSense: con alcuni modelli di telecamere di questa famiglia è possibile fare anche skeleton-tracking in maniera integrata e senza componenti aggiuntive, con il limite di cinque persone per scena e diciotto

punti di giuntura per ogni persona. Questa tecnologia offre prestazioni in tempo reale. Alternativamente, è possibile integrare questa tecnologia con toolbox dedicati per lo skeleton-tracking (e.g. PoseNet, OpenPose).

- Nuitrack SDK: Nuitrack è un middleware in grado di riconoscere i dati dello scheletro e dei gesti tracciando 19 articolazioni contemporaneamente. Ha un SDK multipiattaforma per abilitare l’interfaccia utente naturale (NUI) su Android, Windows e Linux. Questo middleware può essere utilizzato con vari sensori 3D (Kinect V1, Asus Xtion, Orbbec Astra, Orbbec Persee, Intel RealSense).
- Cubemos: è in grado di determinare lo scheletro 2D/3D basandosi sul deep learning. IBM lo ha sviluppato come la tecnologia di tracciamento più all'avanguardia per tracciare un numero illimitato di soggetti in una scena. Può rintracciare 18 punti di giuntura contemporaneamente. È user-friendly e cross-platform (Windows e Linux).
- OpenPose: è stato proposto da alcuni ricercatori della Carnegie Mellon University. OpenPose è il primo sistema multipersona in tempo reale per rilevare contemporaneamente corpo umano, viso, mano, piede (135 punti chiave) da singole immagini o video. OpenPose può stimare 15 (OpenPose MPI), 18 (OpenPose-COCO) o 25 (OpenPose Body-25) punti di giuntura per le articolazioni del corpo umano/piede con un punteggio di *confidence* molto elevato.
- Body Marker: svolge un ruolo indispensabile poiché traccia il movimento durante il riconoscimento dell'azione. Per modellare lo scheletro, ogni specifico marker viene utilizzato per indicare un particolare punto del corpo umano. Errori nel posizionamento o applicazione di uno qualsiasi dei marker sull'agente umano, provocano una forma disorganizzata dello skeleton.

Nel campo del riconoscimento dell'azione umana, si ha che fare con due tipi di rappresentazione: la rappresentazione dello skeleton e la rappresentazione dell'azione. Quest'ultima, non essendo strettamente collegata a questo modulo e avendo già dato dei cenni teorici al riguardo, non verrà considerata. Si prenderà in considerazione solo la rappresentazione di uno skeleton.

Uno skeleton con J punti di giuntura può essere definito come nell'equazione 2.1.

$$X_t = \left\{ X_t^1, X_t^2, \dots, X_t^j, \dots, X_t^J \right\} \in \mathbb{R}^{D \times J} \quad (2.1)$$

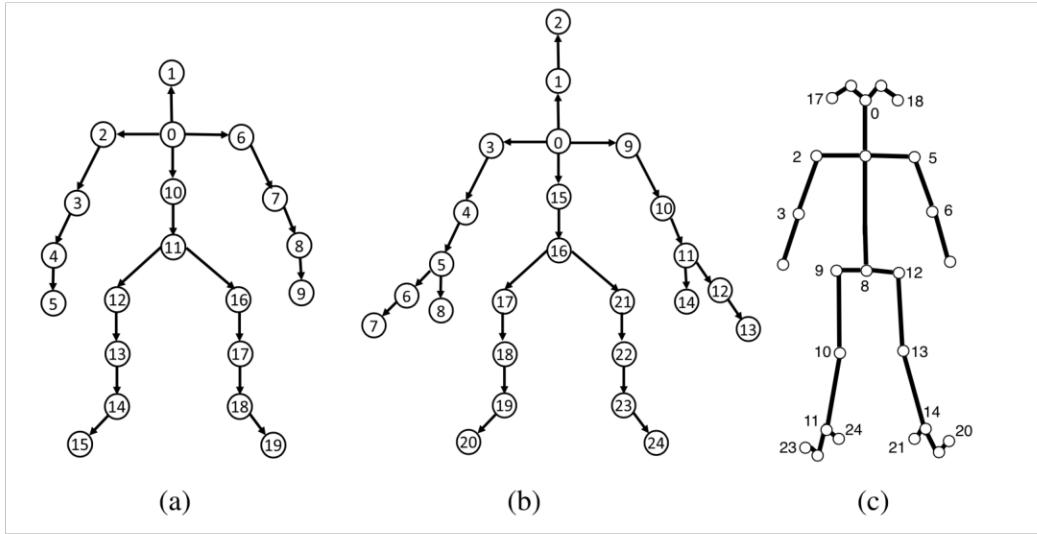


Figura 2.4. Modelli di skeleton ricavati con diversi approcci: (a) Kinect per Xbox 360: 20 joint; (b) Kinect V2: 25 joint; (c) OpenPose (Body-25 Model) 25 joint [29].

Dove X_t^j denota le coordinate di un punto di giuntura j al tempo t , mentre D denota le dimensioni considerate dello skeleton (e.g. per uno skeleton tridimensionale, si avrà $D = 3$). La rappresentazione dello scheletro considera la disposizione della struttura dello scheletro, mentre la rappresentazione dell'azione lavora sulle funzioni legate al movimento del corpo umano. Questa vista poc'anzi, è la definizione matematica di uno skeleton. Per definirlo praticamente in una regione, è possibile usare due approcci che vengono citati di seguito.

- Il primo approccio utilizza un sistema di coordinate. Quest'ultimo è utilizzato per esprimere una posizione geometrica di un particolare punto od oggetto in un sistema a due, tre o multidimensionale. Nelle attività di HAR, sono principalmente impiegati due tipi di sistemi di coordinate, che sono i seguenti.
 - Sistema di coordinate cartesiane: ogni punto viene espresso con una posizione particolare in un piano cartesiano. È possibile esprimere una coordinata cartesiana di un soggetto nel sistema di coordinate bidimensionale come (x, y) o nel sistema di coordinate tridimensionale come (x, y, z) .
 - Sistema di coordinate cilindriche: è un sistema di coordinate tridimensionale che definisce la posizione di un punto sulla base di

tre distanze: quella da un asse di riferimento, quella rispetto alla direzione da un asse relativo e la distanza da un piano di riferimento perpendicolare all'asse scelto.

In ogni caso è possibile passare da un sistema di coordinate all'altro in maniera semplice.

- Il secondo approccio consiste nell'andare a utilizzare una rappresentazione dello skeleton basata su un grafo. Alcune tecniche di rappresentazione dello skeleton basate su grafi sono le seguenti.
 - Skeletal Graph: lo scheletro umano è considerato come un grafo non orientato. I nodi del grafo sono i punti di giuntura delle articolazioni del corpo e una matrice delle adiacenze viene costruita per rappresentare la connessione naturale tra due articolazioni nello scheletro umano [30, 31, 32, 33].
 - Grafo spazio-temporale: è un grafo orientato pesato in cui i nodi e gli archi fungono rispettivamente da punti di giuntura e da ossa di collegamento [34].
 - Grafo diretto aciclico (DAG): i dati dello skeleton possono essere rappresentati dal grafo diretto aciclico (DAG) che si focalizza sulle dipendenze cinematiche [35, 36]. La dipendenza cinematica si basa sulle articolazioni scheletriche (vertici) e sulle ossa (archi) del corpo umano. Se ogni vertice è indicato come j_i , allora l'arco incidente verso il vertice è indicato con o_i^- ; mentre l'arco uscente dal vertice è indicato con o_i^+ .
 - Structural Graph: un grafo non orientato che può essere generato in base alle articolazioni e alle ossa o a parti del corpo del sistema scheletrico umano [37]. È utile quando c'è bisogno di caratteristiche spazio-temporali o informazioni per addestrare il modello. Un grafo diretto è focalizzato sulle dipendenze, mentre il grafo strutturale è utile per trattare le informazioni delle sequenze di più skeleton.
 - Grafo con matrice delle adiacenze pesata: la matrice delle adiacenze è una matrice quadrata, invariante e isomorfa usata per rappresentare un grafo e trovare i vertici di adiacenza. Una matrice delle adiacenze pesata può essere rappresentata come riportato in 2.2 [38].

$$a_{ij} = \begin{cases} 0, & \text{se } i = j \\ \alpha, & \text{se i vertici } i \text{ e } j \text{ sono connessi} \\ \beta, & \text{se i vertici } i \text{ e } j \text{ sono disconnessi} \end{cases} \quad (2.2)$$

- Tree Structure Reference Joints Image (TSRJI): particolare interpretazione dello skeleton come una struttura ad albero [39, 40].

La figura 2.5 mostra le varie rappresentazioni di uno skeleton basate su grafo considerate finora. Per maggiori informazioni in merito alla *Pose Estimation* e all’attività di *Human Activity Recognition*, si consulti [29].

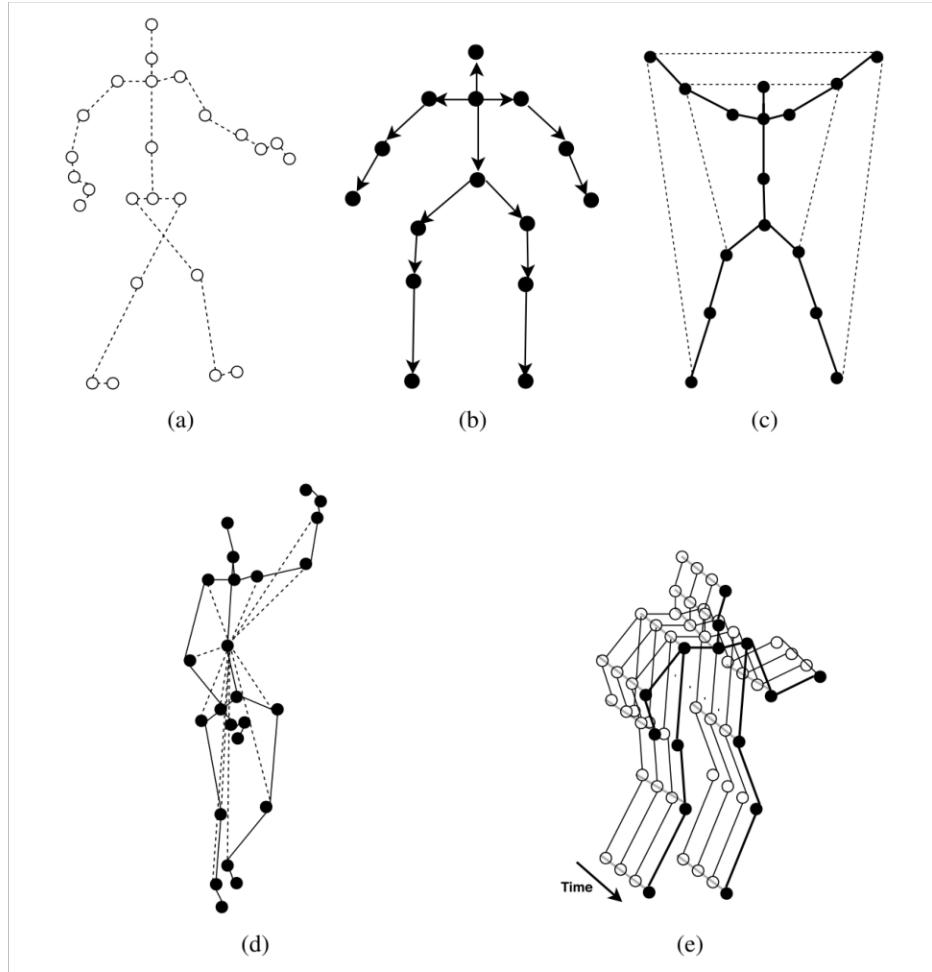


Figura 2.5. Diverse tecniche basate su grafi per la rappresentazione dello skeleton: (a) skeletal graph; (b) DAG; (c) skeleton graph with intrinsic dependencies (solid lines) and extrinsic dependencies (dotted lines); (d) skeleton graph with structural links; (e) spatio-temporal graph [29].

2.3 Trasformazione di sistemi di coordinate

Le tecniche di trasformazione di un sistema di coordinate svolgono un ruolo fondamentale nell'estrazione delle *feature* mantenendo le dimensioni e/o l'orientamento dell'oggetto [41, 42, 43, 44]. I tre metodi più utilizzati di trasformazione di un sistema di coordinate sono: traslazione, rotazione e scaling (ridimensionamento).

La traslazione è la trasformazione geometrica in cui tutti i punti di un oggetto traslato, si sposteranno in un determinato luogo geometrico mantenendo una distanza identica. Per traslare un particolare punto $Z(x, y)$ di m unità verso l'alto o verso il basso, e di n unità verso destra o verso sinistra, il punto traslato sarà: $Z(x \pm n, y \pm m)$.

La rotazione è la trasformazione geometrica in cui l'oggetto ruota di un particolare angolo rispetto a un punto, un asse o una linea. Solitamente la rotazione viene usata quando migliora l'angolazione (e.g. la rotazione delle articolazioni associate alle ossa).

Lo scaling viene applicato quando c'è bisogno di cambiare la dimensione, la forma e l'orientamento dell'oggetto. È un'operazione essenziale al fine di *ingrandire* i dati dello scheletro a disposizione. Se (x, y, z) e $(\bar{x}, \bar{y}, \bar{z})$ definiscono le coordinate di un oggetto, rispettivamente prima e dopo l'applicazione dell'operazione di scaling, allora le coordinate sottoposte a scaling possono essere espresse come nell'equazione 2.3.

$$(\bar{x}, \bar{y}, \bar{z}) = (S_x \times x, S_y \times y, S_z \times z) \quad (2.3)$$

Si precisa che S_x , S_y e S_z sono le matrici di scaling che rappresentano il ridimensionamento di un oggetto rispettivamente attraverso l'asse x , y e z [29].

A questo punto resta da capire come effettuare la trasformazione di coordinate a livello pratico. Si consideri un primo cubo A caratterizzato dall'avere una determinata larghezza w_A , una determinata altezza h_A e una determinata profondità (o lunghezza) d_A . Siccome si tratta di un cubo, queste tre quantità sono identiche, ma è utile definirle in quanto il concetto verrà esteso anche a un parallelepipedo. Si consideri un secondo cubo B caratterizzato dall'avere una determinata larghezza w_B , una determinata altezza h_B e una determinata profondità (o lunghezza) d_B . Anche in questo caso, siccome si tratta di un cubo, queste tre quantità sono identiche. Dato un punto P_A situato nel cubo A o sul cubo A , l'obiettivo è capire come trasformare tale

punto in un *corrispondente* punto P_B situato nel cubo B o sul cubo B ; dove per *corrispondente* si intende un punto che per il secondo cubo è spazialmente analogo, in termini di posizionamento nello spazio, al punto del primo cubo. Un esempio semplificato in 2D consiste nel considerare un rettangolo con base e altezza pari a m (ovvero un quadrato di lato m) e un secondo rettangolo con base e altezza pari a n (ovvero un quadrato di lato n). Si supponga ora che $n = 2 \cdot m$. Se $m = 2$, allora $n = 4$. Inoltre, per semplicità, entrambi i quadrati hanno il vertice in basso a sinistra coincidente con l'origine.

In queste condizioni, si può dedurre che il punto P_A di A di coordinate $(1.5, 1.5)$ è spazialmente analogo al punto P_B di B di coordinate $(3.0, 3.0)$. Una volta chiarito questo aspetto, va precisato che le condizioni indicate sono molto semplificate. Basti considerare che in casi reali e generici, c'è da considerare non un quadrato, non un rettangolo, non un cubo, ma un parallelepipedo retto; ovvero un parallelepipedo avente come facce della superficie laterale dei rettangoli, per cui gli spigoli della superficie laterale sono tutti paralleli tra loro, paralleli all'altezza e perpendicolari alle basi. In modo equivalente, un parallelepipedo retto è un prisma retto avente come basi due parallelogrammi.

Nel caso del progetto di tesi, però, è possibile individuare una prima semplificazione importante al problema: la profondità è uguale per entrambi i parallelepipedi retti. Di conseguenza non è necessario effettuare una trasformazione lungo tale asse. Il problema, così facendo, si tramuta in un problema di trasformazione di spazi di coordinate in 2D per rettangoli. In tal caso, basta applicare delle semplici relazioni di natura geometrica.

Si consideri un rettangolo A e un rettangolo B sul piano cartesiano. Il rettangolo A ha origine in (X_A, Y_A) , mentre il rettangolo B ha origine in (X_B, Y_B) . La base di A è lunga w_A , mentre l'altezza di A è h_A . Invece, la base di B è lunga w_B , mentre l'altezza di B è h_B . Per trasformare un punto $P_A = (x_A, y_A)$ appartenente ad A in un punto spazialmente analogo $P_B = (x_B, y_B)$, è necessario innanzitutto esprimere le coordinate del punto P_A in termini dello spazio del rettangolo A , poi va calcolato il rapporto della posizione del punto del rettangolo A rispetto al lato della coordinata considerata. A questo punto, è necessario scalare il risultato per il lato della coordinata considerata del rettangolo B , per ottenere la posizione relativa al rettangolo B . Infine, si trasla il punto nel rettangolo B , ottenendo così un riferimento assoluto. In base ai passi appena menzionati, si ricavano le equazioni 2.4 e 2.5.

$$x_B = \left(\left(\frac{x_A - X_A}{w_A} \right) \cdot w_B \right) + X_B \quad (2.4)$$

$$y_B = \left(\left(\frac{y_A - Y_A}{h_A} \right) \cdot h_B \right) + Y_B \quad (2.5)$$

Dal momento che, grazie alla telecamera Intel RealSense, il *formato* della coordinata relativa all'altezza, ovvero la y , risulta essere differente rispetto a quello della x , che invece indica la larghezza; è possibile usare una trasformazione semplificata per la y , mentre si lascia invariata la trasformazione per la x vista nell'equazione 2.4. La trasformazione semplificata per la y si è ottenuta in maniera sperimentale tramite l'equazione 2.6.

$$y_B = y_A - y - o \quad (2.6)$$

Nell'equazione 2.6, la o indica un particolare offset relativo all'altezza che, nel caso del *CVPR Lab*, è risultato essere sperimentalmente pari a 0.15 metri. Invece, per quanto riguarda la profondità è importante precisare che va aggiunta, prima di considerare la coordinata prodotta come definitiva; un offset dato dalla distanza che intercorre fra la parete retrostante alla telecamera e la lente della telecamera stessa. Nel caso del posizionamento della telecamera su una delle scrivanie del *CVPR Lab*, tale costante è pari a 0.45 metri. Il posizionamento appena menzionato è quello adottato durante le fasi di test del modulo AI Watch A1 e corrisponde *grosso modo* al posizionamento della telecamera visibile nella figura 1.6.

Inoltre, si precisa che entrambi i valori degli offset sono da intendersi in metri, cioè la o corrisponde a 15 cm, mentre la distanza dalla parete retrostante alla telecamera a 45 cm.

Infine, si precisa che affinché possa essere eseguita una corretta trasformazione delle coordinate da un sistema all'altro, è importantissimo impostare i seguenti parametri relativi all'ambiente nel quale viene installata la telecamera: larghezza scena, altezza scena e distanza della lente della telecamera dalla parete retrostante. E, per un'opportuna conversione nelle coordinate di un mondo virtuale, è importante specificare l'origine dell'asse x e dell'asse z del mondo virtuale; tenendo conto del fatto che, in alcuni mondi virtuali (e.g. Unity), le coordinate vengono invertite tra x e z .

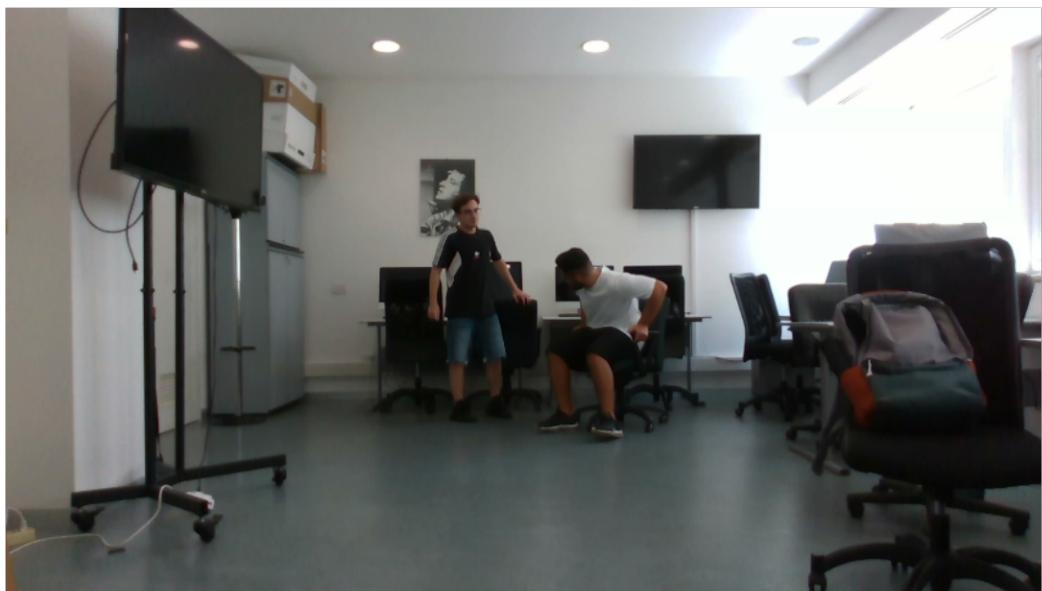


Figura 2.6. Immagine risultato skeleton frame 395 - Test 3.

In figura 2.6 si mostra un'immagine considerata a titolo d'esempio durante uno dei test eseguiti. Si consideri ora la figura 2.7. In quest'ultima viene mostrata l'idea della trasformazione di coordinate commentata in precedenza. L'idea è di ottenere le coordinate dei punti di giuntura nello spazio relativo alla stanza. Per fare ciò e per ottenere delle coordinate che è possibile mostrare anche in un ambiente virtuale Unity, però, è necessario affrontare due problematiche: se si approssima la stanza nello spazio come un parallelepipedo, questa non è uguale al parallelepipedo di riferimento dello spazio di coordinate della telecamera Intel RealSense, né sono uguali gli intervalli utilizzati per individuare i punti nello spazio. Di conseguenza è necessario adoperare una trasformazione tra spazi di coordinate. Dal momento che la profondità resta immutata nei due spazi, allora è possibile semplificare il problema. Infatti, anziché considerare un parallelepipedo, è ora possibile considerare un rettangolo e la trasformazione da fare consiste in una trasformazione di punti 2D appartenenti a rettangoli con conservazione dell'analogia spaziale.

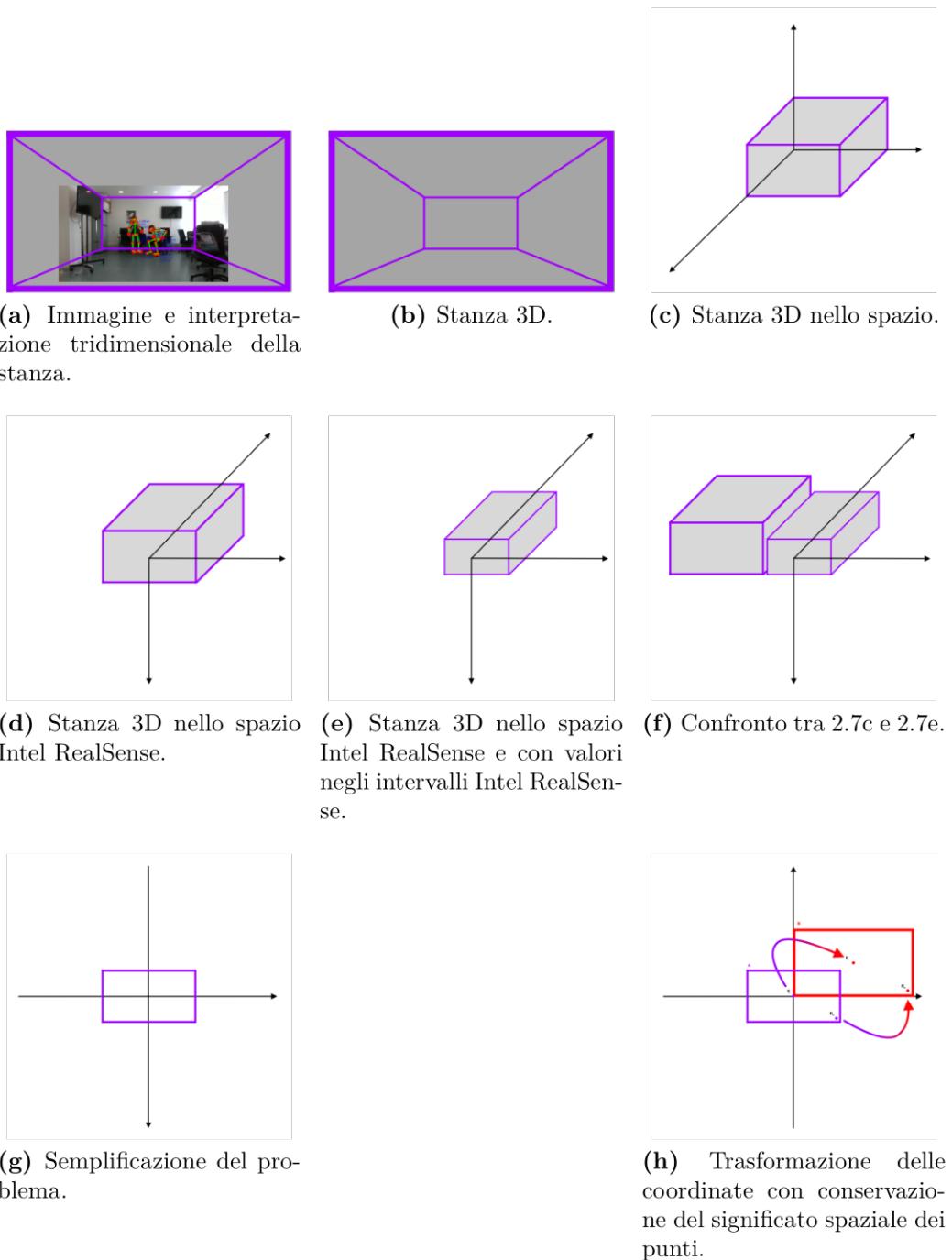


Figura 2.7. Interpretazione del problema relativo alla trasformazione fra sistemi di coordinate.

2.4 Streaming di eventi

Per il mondo digitale, lo streaming di eventi è l'equivalente del sistema nervoso centrale del corpo umano. È la base tecnologica per il mondo “always on” e sempre connesso, dove le aziende sono sempre più basate su un software automatizzato di qualche tipo, e dove vi è l'utente addetto all'utilizzo di tale software.

Tecnicamente parlando, lo streaming di eventi è la pratica di acquisire dati in tempo reale da fonti di eventi come database, sensori, dispositivi mobili, servizi cloud e applicazioni software sotto forma di flussi di eventi; memorizzare questi flussi di eventi in modo duraturo per il recupero successivo; manipolare, elaborare e reagire ai flussi di eventi in tempo reale e instradare i flussi di eventi verso diverse destinazioni (che magari sfruttano diverse tipologie di tecnologie), se necessario. Di conseguenza, lo streaming degli eventi deve garantire un flusso e un'interpretazione continua dei dati, in modo che le informazioni corrette siano al posto giusto e al momento giusto.

Lo streaming di eventi viene applicato a un'ampia varietà di contesti in diversi settori e organizzazioni. Alcune attività per le quali viene impiegato lo streaming di eventi sono le seguenti:

- per elaborare i pagamenti e le transazioni finanziarie in tempo reale, come in borse, banche e assicurazioni;
- per tracciare e monitorare auto, camion, flotte e spedizioni in tempo reale, ad esempio nel settore della logistica e dell'industria automobilistica;
- per acquisire e analizzare continuamente i dati dei sensori da dispositivi IoT o altre apparecchiature, ad esempio in fabbriche e parchi eolici;
- per raccogliere e reagire immediatamente alle interazioni e agli ordini dei clienti, ad esempio nella vendita al dettaglio, nel settore alberghiero e nelle applicazioni su smartphone;
- per monitorare i pazienti in ospedale e prevedere i cambiamenti nelle condizioni di salute, oppure per garantire un trattamento tempestivo in caso di emergenza;
- per collegare, archiviare e rendere disponibili i dati prodotti da diverse divisioni di un'azienda;

- per permettere la trasmissione dei dati in un'architettura software basata su eventi e microservizi;

Al fine di realizzare un sistema che necessita in qualche modo d'impiegare lo streaming di eventi, esistono diversi approcci e tecnologie. Nella tabella 2.1 si riportano le principali tecnologie in merito.

	Descrizione
Apache Kafka [74]	<p>Kafka combina le tre funzionalità chiave che seguono, in modo da permettere l'implementazione dello streaming di eventi end-to-end con una singola soluzione.</p> <ul style="list-style-type: none">• Pubblicare (scrivere) e iscriversi a (leggere) flussi di eventi, tra cui l'import/export continuo dei dati da altri sistemi.• Immagazzinare flussi di eventi in maniera persistente e affidabile per tutto il tempo necessario.• Elaborare flussi di eventi all'occorrenza e in real-time. <p>Tutte queste funzionalità sono fornite in modo distribuito, altamente scalabile, elastico, resistente ai guasti e sicuro. Kafka può essere distribuito su hardware anche di medio e basso livello, macchine virtuali e container, e con le opportune premesse, anche nel cloud. È possibile scegliere tra la gestione autonoma degli ambienti Kafka e l'utilizzo di servizi completamente gestiti da fornitori terzi.</p>

Amazon Kinesis [93]	<p>Amazon Kinesis semplifica la raccolta, l'elaborazione e l'analisi di flussi di dati in tempo reale, per ottenere analisi tempestive e reagire rapidamente rispetto alle nuove informazioni. Offre funzionalità chiave per elaborare i flussi di dati su qualsiasi scala a costi contenuti, nonché la flessibilità di scegliere gli strumenti più adatti ai requisiti di un'applicazione. Con Amazon Kinesis, è possibile acquisire dati in tempo reale, ad esempio video, audio, log di applicazioni, clickstream di siti web e dati telemetrici da dispositivi IoT, per l'apprendimento automatico, per l'analisi e per altre applicazioni. Amazon Kinesis consente di elaborare e analizzare i dati nel momento in cui arrivano, con risposte istantanee e non ritardate dall'attesa che venga completata la ricezione delle informazioni.</p> <p>Inoltre, questa tecnologia può essere impiegata per trasmettere in streaming e in tutta sicurezza, i video provenienti da dispositivi dotati di videocamera presenti in abitazioni, uffici, stabilimenti e luoghi pubblici. Questi flussi video possono essere utilizzati per la riproduzione di video in un secondo momento, per il monitoraggio della sicurezza, per il riconoscimento facciale, per l'apprendimento automatico e per altri tipi di analisi. Inoltre, è facilmente integrabile nell'ambiente AWS di Amazon.</p> <p>Amazon Kinesis può essere usato con applicazioni che operano in tempo reale, ad esempio per il monitoraggio, il rilevamento di attività fraudolente e l'aggiornamento costante di graduatorie. Permette di acquisire i flussi di dati tramite Kinesis Data Streams ed elaborarli con Kinesis Analytics, inoltrando i risultati in un qualsiasi datastore o applicazione mediante Kinesis Data Streams, con latenze nell'ordine di millisecondi. In questo modo, si ha un maggiore controllo su clienti, applicazioni e prodotti, che consente una maggiore reattività agli eventi. È possibile usare questa tecnologia per elaborare flussi di dati da dispositivi IoT quali elettrodomestici e sensori integrati. Si possono usare questi dati per inviare avvisi in tempo reale o attivare operazioni specifiche e programmate quando un sensore supera una determinata soglia operativa. Per usare Amazon Kinesis non è necessario partire da zero, grazie alla documentazione presente e alle diverse implementazioni già presenti anche per dispositivi IoT. Amazon Kinesis però, è a pagamento.</p>
----------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

RabbitMQ [94]	<p>RabbitMQ è un broker di messaggi open source che può essere facilmente distribuito sul cloud. Oltre a supportare servizi di messaggistica asincrona, offre una grande esperienza di sviluppo con Java, Go, Python, Ruby, e altri linguaggi. Kafka può essere utilizzato a partire da più linguaggi di programmazione. In maniera del tutto analoga, anche RabbitMQ dispone di questo supporto. Inoltre, può essere eseguito su diversi sistemi operativi e infrastrutture cloud. Infine, può essere implementato anche in un ambiente distribuito così da garantire un'elevata disponibilità del servizio offerto.</p>
ActiveMQ [95]	<p>ActiveMQ è un broker di messaggistica multiprotocollo, flessibile e open source. Dal momento che fornisce supporto per un'ampia varietà di protocolli, gli sviluppatori possono scegliere di usare diversi di linguaggi di programmazione e piattaforme. Il protocollo AMQP facilita l'integrazione con molte applicazioni basate su diverse piattaforme. ActiveMQ fornisce alcune funzionalità importanti, tra cui il bilanciamento del carico, l'allocazione delle risorse e numerose funzionalità di gestione. Può essere perfettamente integrato in quasi tutte gli applicativi, grazie al fatto che i messaggi presentano una semantica semplice, ma potente. Inoltre, è noto per l'integrazione di servizi tra più applicazioni e per l'allocazione efficiente delle risorse. Apache ActiveMQ è una buona alternativa a Kafka ed è gratuito.</p>

Red Hat AMQ [96]	Così come Apache Kafka e ActiveMQ, anche Red Hat AMQ ha una solida suite di componenti, protocolli e un buon supporto da parte della community. Offre una soluzione sicura e leggera per la consegna dei messaggi ed è una delle migliori alternative a Kafka. Rispetto alla maggior parte degli strumenti di streaming, Red Hat AMQ ha un'esecuzione più veloce e offre uno strumento di messaggistica flessibile che consente la comunicazione istantanea. Di conseguenza, soddisfa in modo efficace le esigenze organizzative. Fornendo informazioni in tempo reale e connettività in maniera continua con i dispositivi IoT, Red Hat AMQ offre informazioni istantanee dalle fonti che trasmettono i dati. Può essere integrato e usato come alternativa ad Apache Kafka. Red Hat AMQ è a pagamento.
-------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tabella 2.1. Approcci e tecnologie per lo streaming di eventi.

Ora che sono state prese in considerazione le principali problematiche affrontate durante il progetto di tesi e le soluzioni che è possibile mettere in atto, si valuteranno quali sono state, nel complesso e in linea generale, le scelte progettuali fondamentali del modulo AI Watch A1.

2.5 Scelte progettuali

Per quanto riguarda il problema della stima della profondità, si è scelto di adoperare la tecnologia fornita da Intel RealSense [68] e in particolare si è utilizzata una telecamera Intel RealSense D435 [97]. Tale scelta è motivata dal fatto che la tecnologia Intel RealSense funziona in maniera ottimale nella maggior parte degli ambienti e condizioni di luce, a differenza di altre tipologie di telecamere menzionate nella sezione 2.1. Inoltre, una telecamera Intel RealSense è di tipo *general purpose*, ovvero può essere sfruttata anche per scopi e impieghi differenti da quelli per la quale è stata utilizzata in questo progetto di tesi. Tra le tante possibili telecamere Intel RealSense è stata scelta la D435, in quanto rappresenta un buon compromesso tra campo visivo, potenzialità tipiche di una telecamera e determinazione accurata della profondità di un punto nello spazio. Maggiori dettagli, approfondimenti tecnici e implementativi riguardanti la telecamera Intel RealSense D435, sono presenti nella sottosezione 3.1.

Invece, per quanto riguarda il rilevamento dello skeleton delle persone presenti all'interno della scena inquadrata dalla telecamera, è stato impiegato *OpenPose* [72]. La motivazione che ha portato a questa scelta è basata prevalentemente sui vantaggi offerti dall'utilizzare OpenPose: livello di precisione elevato, numero di punti per il rilevamento dello skeleton di tipo *body* maggiore rispetto agli altri approcci e tecnologie, capacità di essere riutilizzato, con l'opportuno riaddestramento dei modelli, per il rilevamento non solo di persone, ma anche di determinati veicoli, per esempio. Inoltre, come verrà analizzato e approfondito nella sezione 3.2.1, OpenPose risulta utilizzare un approccio bottom-up per il rilevamento degli skeleton e tali tipi di approcci sono di gran lunga più precisi, efficienti e meno tendenti a errori, rispetto agli approcci top-down di cui, per esempio, AlphaPose [98] fa uso.

Mask R-CNN, rappresenta una buona alternativa a OpenPose. Purtroppo però, richiede di essere esteso per effettuare la skeleton detection, in quanto non nasce prettamente per tale scopo. Di conseguenza utilizzare Openpose, risulta essere una scelta che consente una fase di sviluppo più rapida. Passando al confronto di OpenPose con PoseNet [90], l'alta precisione di OpenPose non è paragonabile a quella fornita da PoseNet. Infine, per quanto riguarda Cubemos e Nuitrack, sono sicuramente alternative valide e avanzate, rispetto a OpenPose, ma si tratta di soluzioni costose da adottare e, tra i tanti obiettivi della tesi di laurea, c'è quello di realizzare una soluzione accurata, allo stato dell'arte ed economicamente conveniente.

Un punto debole di OpenPose è il fattore prestazioni. Infatti, sono sicuramente le prestazioni e la velocità di calcolo, che in alcuni casi e a seconda del calcolatore scelto per la sua esecuzione, può non permettere l'esecuzione del progetto in modalità real-time. Per fare ciò, in ogni caso, servirebbe molta potenza computazionale in termini di GPU. Al momento esistono delle soluzioni alternative che hanno vantaggi sotto questo punto di vista rispetto a OpenPose; per esempio c'è il progetto WrnchAI, che però è stato acquisito da un'azienda e di conseguenza, al momento della realizzazione di questo progetto di tesi, non è stato possibile rintracciare online tutto il necessario per la sua installazione e il successivo sviluppo. D'altro canto, il supporto della community, la documentazione dettagliata e la presenza di tantissimi problemi affrontati e già risolti da altri utenti in giro per il mondo, costituisce nettamente un punto a favore di OpenPose. Ciò, infatti, ha permesso un'accelerazione nel processo di sviluppo del modulo A1. Discorso analogo per la tecnologia Intel RealSense.

Infine, per quanto riguarda lo streaming di dati da effettuare tra i vari moduli del progetto per permettere loro la comunicazione, si sfrutta Apache Kafka. La motivazione è in parte evidente dal confronto condotto nella sezione 2.4. Inoltre, si impiega Confluent al fine di facilitare il setup iniziale per l'avvio di Apache Kafka e la gestione durante tutta la sua esecuzione. Entrambe le tecnologie sono gratuite, e una volta presa la necessaria dimestichezza con Apache Kafka, è possibile, per l'utente finale, fare a meno di Confluent. Maggiori informazioni in merito ad Apache Kafka sono presenti nella sezione 3.4.

2.6 Flusso di esecuzione AI Watch A1

Di seguito verranno illustrati nel dettaglio i passi eseguiti durante l'esecuzione del modulo A1. Per quanto concerne l'installazione delle librerie necessarie alla compilazione e all'esecuzione del modulo A1, si rimanda al manuale utente online [99].

Si tenga presente che il modulo è eseguito per iterazioni, nel senso che vengono acquisiti i frame, si estrae lo skeleton, si effettua la mappatura in uno specifico sistema di coordinate e si inviano i dati al modulo successivo. Dopodiché si riprende con una nuova iterazione, che presenterà gli stessi passaggi poc'anzi menzionati. Una singola iterazione è anche detta *burst*.

L'idea del modulo A1 è quella di avviare innanzitutto l'ambiente di esecuzione della telecamera Intel RealSense D435. Ciò comporta attivare lo stream video RGB e quello che cattura l'informazione sulla profondità. Siccome i frame contenenti l'informazione sulla profondità per ogni pixel e quelli RGB hanno risoluzioni e formati differenti, in quanto vengono acquisiti da dispositivi di imaging differenti all'interno della stessa telecamera, è obbligatorio impostare l'opzione di allineamento dei due frame, così che vi sia risoluzione e formato comune a entrambi. Un altro passaggio fondamentale è calibrare la telecamera. Questo compito può essere delegato alla telecamera oppure eseguito manualmente. Nel primo caso basta far acquisire alla telecamera un certo numero di frame al fine di far stabilizzare i valori di esposizione, contrasto, e così via. A tal punto è necessario tenere presente che le caratteristiche intrinseche ed estrinseche di una telecamera RealSense, sono di fondamentale importanza, come si vedrà nella sottosezione 3.1.1; quindi è necessario memorizzarle in qualche modo.

A questo punto si può procedere con l'acquisizione del frame relativo alla profondità e quello RGB. Una prima considerazione da fare è quella relativa al tempo impiegato da OpenPose per elaborare i frame acquisiti. Infatti, scegliere un quantitativo di frame troppo elevato per il singolo *burst* può essere controproducente in termini di prestazioni. Una seconda considerazione da fare è quella del rumore presente nelle immagini di profondità. Questo fenomeno accade a causa di diversi fenomeni, come le luci all'interno dell'ambiente dov'è situata la telecamera, la riflettanza degli oggetti nella scena, altri fattori ambientali come la differenza tra un ambiente interno e uno esterno, e così via. Per sopperire a tale inconveniente, che nell'immagine di profondità si manifesta solitamente con *buchi* neri, regioni o righe nere, si può applicare un algoritmo di morfologia matematica, detto algoritmo di *hole filling*, come

quello preso in considerazione nella sottosezione 1.4.4. Nell’implementazione della libreria d’Intel, in particolare, si sfruttano le relazioni di adiacenza tra i pixel che fanno parte del *bucco* e quelli circostanti, al fine di valutare quale valore d’intensità assegnare ai pixel facenti parte del *bucco*.

Siccome molti frame acquisiti nello stesso secondo conservano informazioni molto simili, e in alcuni casi addirittura uguali; è possibile pensare di diminuire il numero di frame acquisiti al secondo, in modo da poter estendere il *burst temporale* di OpenPose. Per esempio, un parametro che permette di raggiungere risultati alquanto ottimali e che è pervenuto da test condotti sperimentale, è quello di scartare, di volta in volta, i cinque frame successivi a quello che è stato acquisito. Così facendo, se a ogni *burst* vengono acquisiti x frame e la telecamera acquisisce y frame al secondo, con $y < x$; allora l’ammontare di x frame acquisiti saranno *spalmati* su una fascia temporale più ampia.

Ora che si è in possesso del frame RGB e del frame profondità, si passa ad avviare OpenPose che si occuperà di estrarre gli skeleton dal frame RGB e produrre degli opportuni file di output. All’interno di questi file di output, saranno presenti tutte le coordinate 2D dei punti di giuntura individuati da OpenPose, per ogni persona rilevata all’interno dell’immagine.

Ora il passaggio fondamentale è quello di andare a realizzare una struttura dati che rappresenti uno skeleton per ogni persona rilevata da OpenPose. Per fare ciò si interagisce assiduamente con i file di output prodotti da OpenPose. Una volta memorizzati i dati di output di OpenPose all’interno del programma, tramite una particolare operazione detta operazione di *deprojection* (vedi sottosezione 3.1.1; e tramite l’informazione sulla profondità acquisita dalla telecamera Intel RealSense D435, si vanno a ricavare i punti nello spazio tridimensionale corrispondenti ai pixel ove OpenPose ha determinato un punto di giuntura per un determinato skeleton. Questa operazione è eseguita per tutti i punti di giuntura e per tutti gli skeleton registrati da OpenPose. È necessario precisare che OpenPose potrebbe produrre degli skeleton con un livello di *confidence* più o meno basso. Di conseguenza è opportuno filtrare gli skeleton che, nel loro insieme, presentano un livello di *confidence* basso.

Una volta che le coordinate dei pixel dei punti di giuntura nell’immagine, combinate con le profondità presenti in quei determinati pixel, sono state convertite in coordinate di punti nello spazio tridimensionale; non resta che effettuare una trasformazione tra sistemi di coordinate, al fine di ottenere i punti in un formato adatto al dominio applicativo. In particolare, si farà

riferimento alle grandezze e alle origini degli assi utilizzate nel mondo virtuale, realizzato mediante Unity nel terzo modulo del progetto AI Watch. I dettagli su come effettuare tali operazioni sono forniti nella sottosezione 2.3. Nel momento in cui tutti i punti di giuntura sono stati convertiti nel nuovo sistema di coordinate, si passa a salvarli in un opportuno formato in un file apposito. Un singolo file viene generato per ogni frame analizzato e conterrà le informazioni riguardanti tutti i punti di giuntura di tutti gli skeleton individuati in quel frame specifico.

Come penultimo passaggio, non resta che utilizzare Apache Kafka per inviare le informazioni ricavate da ogni frame al modulo successivo di AI Watch. Infine, siccome è sicuramente doveroso ridurre al minimo i consumi dell'applicativo in termini di memoria e di disco, si effettuano operazioni adeguate di pulizia dei file che sono stati prodotti durante i passaggi precedenti e che non saranno più utilizzati nelle iterazioni successive. È importante precisare che, sebbene nei passaggi appena menzionati si sia fatto riferimento a titolo esemplificativo a un singolo frame, in realtà, si è riscontrato sperimentalmente che un miglior approccio consiste nel far lavorare OpenPose con un numero di frame maggiore a uno; sia per migliorare ulteriormente la precisione, che per ridurre il carico temporale dovuto all'attivazione della rete neurale convoluzionale. Si precisa che non esiste un numero fisso di frame da utilizzare: tutto dipende dalle capacità computazionali del calcolatore sul quale l'applicativo è in esecuzione.

2.7 Complessità computazionale

È importante sapere come aumenta il tempo di esecuzione di un algoritmo al crescere della dimensione dell'input *al limite*, cioè quando la dimensione dell'input cresce senza limitazioni. In tal caso si parla di analisi dell'efficienza asintotica degli algoritmi. Date per note le nozioni di notazione asintotica O , Ω , Θ , funzioni e tempi di esecuzione [18]; si passa all'analisi sullo studio della complessità di tempo e di spazio del modulo AI Watch A1.

Per quanto riguarda la complessità di tempo, vi sono i seguenti contributi.

- $c \cdot k \cdot M \cdot N \cdot u$ operazioni per la fase di acquisizione dei frame; ove u è il numero di frame acquisiti che viene scelto dall'utente nel codice *client*, M e N sono rispettivamente il numero di righe e il numero di colonne del frame acquisito e k indica i passaggi necessari per l'algoritmo di riempimento buchi appartenente al ramo della morfologia matematica. Il valore c è una costante e indica il numero di operazioni impiegate dalla tecnologia Intel RealSense per realizzare l'operazione di acquisizione di un frame.
- $4 \cdot c \cdot M \cdot N \cdot u$ operazioni che comprendono le seguenti attività sui frame acquisiti: allineamento, scaling, determinazione della distanza, applicazione del filtro colore.
- $5 \cdot u$ operazioni per il salvataggio delle immagini.
- $c \cdot t \cdot u$ operazioni per l'attesa e lo scarto dei t frame appartenenti a quelli da ignorare. Qualora si scegliesse di non apportare il miglioramento relativo allo scartare i frame simili e consecutivi, allora tale contributo non sarà presente.
- $O(1)$, per il tempo di esecuzione della CNN in OpenPose [45].
- $O(n^2)$, per la rilevazione e la conversione multipersona degli skeleton, ove n è il numero di persone rilevate all'interno del frame [45].
- $6 \cdot u \cdot n \cdot 25$ operazioni per la costruzione e l'elaborazione degli skeleton all'interno del modulo AI Watch A1, una volta che sono stati prodotti da OpenPose. Il valore 25 indica il fatto che i punti di giuntura individuati da OpenPose sono ben 25, nelle modalità in cui lo si sta utilizzando. Il valore 6 indica, invece, il numero di volte che si fanno ben $u \cdot n \cdot 25$ operazioni, al fine di costruire ed elaborare correttamente i vari skeleton individuati.

- u operazioni per permettere ad Apache Kafka di eseguire l'invio di u file al modulo successivo dell'architettura AI Watch.

Si considerino i primi due contributi. In termini di tempo pratico di esecuzione, tutte queste attività incidono molto poco sul tempo complessivo. Infatti, il valore di k è molto contenuto e le operazioni successive vengono fatte mediante l'ausilio della tecnologia Intel RealSense e del relativo SDK 2.0, che risultano essere ottimizzati. Sostanzialmente questi due primi contributi sono *abbattuti* dalla costante c . Chiaramente, se il numero di frame u da voler acquisire è molto elevato, allora il tempo pratico necessario sarà maggiore, in quanto sarà necessario attenersi ai limiti hardware relativi alla frequenza di acquisizione dei fotogrammi della telecamera (al più 30 FPS).

Se si valutano le operazioni dal punto di vista asintotico, però, questi due contributi vanno considerati entrambi come dei contributi quadratici. Infatti, supponendo che il numero delle righe e il numero delle colonne delle immagini sia pressoché simile, tale contributo risulterebbe quadratico. Quindi è necessario distinguere il tempo di esecuzione operativo, dalla complessità computazionale ottenuta mediante analisi asintotica. Infatti, questi due concetti non sono collegati tra di loro nel modulo A1.

Passando al terzo, al quarto, al settimo e all'ottavo contributo; questi contribuiscono con una quantità lineare o costante alla complessità finale. Infine, vi è il contributo dato da OpenPose. In particolare questo risulta essere il contributo più pesante e infatti, sia in termini di complessità computazionale che di tempo pratico, è la fase che richiede più tempo. Inoltre, è necessario precisare che, benché il contributo relativo all'esecuzione della CNN in OpenPose sembri piccolo e trascurabile dal punto di vista computazionale, in realtà operativamente parlando e in base al documento scientifico che lo presenta [45], richiede un tempo costante abbastanza considerevole. Se si pensa di fondere tutti i contributi all'interno del contributo relativo alla rilevazione e conversione multipersona degli skeleton di OpenPose, e trascurando gli altri contributi lineari o pressoché costanti, la complessità finale risulta essere quadratica.

Una volta valutata la complessità di tempo, di seguito si considereranno i contributi presenti nella valutazione della complessità di spazio.

- un array di e elementi per memorizzare i parametri d'input dell'applicativo.

- $5 \cdot M \cdot N \cdot u$, per le immagini utilizzate man mano per ognuno degli u frame.
- $2 \cdot u \cdot n \cdot 25$ elementi per memorizzare correttamente il file JSON e il corrispettivo array all'interno del quale vengono salvate le relative informazioni. Si ricorda che n è il numero di persone rilevate all'interno della scena.

Quindi, riassumendo e valutando la situazione asintoticamente, la complessità di spazio è quadratica, dal momento che si può supporre che il numero delle righe e il numero delle colonne delle immagini sia pressoché simile.

Formalizzando, la complessità di tempo del modulo AI Watch A1 risulta essere $\Theta(n^2 + N^2)$. In particolare, si precisa che n è il numero di persone rilevate all'interno della scena e N è il numero di righe di un'immagine, dal momento che è possibile inquadrarla come una matrice e, dal momento che si è supposto che il numero delle righe è circa uguale al numero delle colonne. Invece, la complessità di spazio del modulo AI Watch A1 risulta essere $\Theta(N^2)$. Quindi, in entrambi i casi si tratta di una complessità quadratica.

È necessario precisare che l'analisi delle complessità di tempo e di spazio poc'anzi effettuate, fa riferimento alle complessità di una singola iterazione, ovvero di un singolo *burst d'esecuzione*. Nel momento in cui i dati necessari vengono inviati al modulo successivo di AI Watch, si riprende con l'esecuzione ripetendo nuovamente le operazioni.

2.8 Ingegnerizzazione AI Watch A1

Man mano che i sistemi software diventano sempre più complessi, è sempre più importante effettuare delle scelte operative e architetturali che possano permettere al sistema di essere modulare, estensibile, scalabile, resistente ai guasti e facilmente manutenibile. Infatti, le esigenze del contesto o del dominio applicativo, potrebbero cambiare col tempo, potrebbero essere introdotte nuove tecnologie o potrebbero nascere nuovi pericoli. Di conseguenza, in questi casi, vanno apportate delle modifiche, dei cambiamenti, delle estensioni al sistema software già esistente. Per realizzare ciò solitamente si fa riferimento a dei principi progettuali e architetturali ben specifici, ovvero delle soluzioni note, pronte, riusabili e convenienti per problemi tipici. Tali soluzioni sono note come *design pattern*. Utilizzarli permette di ridurre i tempi di sviluppo e di debug, migliora la struttura e rende l'implementazione sicuramente più lineare, pulita e generalmente più leggera. Un design pattern non è un progetto finito che può essere trasformato direttamente in codice. È una descrizione o un modello per come risolvere un problema che può essere utilizzato in molte situazioni diverse ed eventualmente adattato alle circostanze affrontate [46, 47, 48].

Solitamente si è soliti suddividere i design pattern nelle seguenti tre categorie.

- Design pattern creazionali: riguardano il come viene istanziata una determinata classe. Questa categoria può essere ulteriormente suddivisa in design pattern creazionali di classe e design pattern creazionali di oggetti. I pattern creazionali di classe usano l'ereditarietà in modo efficace nel processo di generazione di un'istanza; mentre i pattern creazionali di oggetti usano la delega in modo opportuno per eseguire lo stesso compito.
- Design pattern strutturali: riguardano la composizione di classi e dei corrispettivi oggetti. I pattern strutturali di classe utilizzano l'ereditarietà per realizzare delle interfacce; mentre i pattern strutturali orientati all'oggetto singolo, definiscono diverse strade per comporre degli oggetti tra di loro al fine di ottenere delle nuove funzionalità.
- Design pattern comportamentali: riguardano come gli oggetti comunicano tra di loro, e di conseguenza come le classi interagiscono tra di loro.

Oltre a queste soluzioni si cerca di seguire quanto più possibile i principi SOLID e altri principi dello sviluppo orientato agli oggetti. La parola è un

acronimo che serve a ricordare tali principi (single responsibility, open-closed, Liskov substitution, interface segregation, dependency inversion). I principi SOLID sono intesi come linee guida per lo sviluppo di codice leggibile, estensibile e facilmente manutenibile. Di seguito si riportano i principi SOLID.

- Single responsibility principle: afferma che ogni classe dovrebbe avere una e una sola responsabilità, interamente encapsulata al suo interno.
- Open-closed principle: un'entità software dovrebbe essere aperta alle estensioni, ma chiusa alle modifiche.
- Liskov substitution principle: gli oggetti dovrebbero poter essere sostituiti con dei loro sottotipi, senza alterare il comportamento del programma che li utilizza.
- Interface segregation principle: è preferibile avere più interfacce specifiche, anziché una singola interfaccia generica.
- Dependency inversion principle: una classe dovrebbe dipendere dalle astrazioni, non da classi concrete.

All'interno del modulo AI Watch A1 si è cercato di rendere l'architettura software quanto più snella e chiara possibile. Chiaramente la complessità intrinseca del dominio applicativo rende questo obiettivo abbastanza sfidante. In linea generale, si è favorita la riusabilità del codice e una struttura che faciliti eventuali future manutenzioni ed estensioni.

L'idea di fondo all'interno del modulo AI Watch A1 è stata quella di costruire diversi *manager* che hanno determinate caratteristiche, compiti e funzionalità. Per esempio, è utile considerare il manager che si occuperà della trasmissione dei dati via Kafka e di nient'altro. In questo modo si riesce a minimizzare l'accoppiamento tra manager con ruoli differenti e massimizzare la coesione contestuale interna, rispetto ai compiti svolti dal manager stesso. Analogamente è facile dedurre che vi sia un manager che si occupi della generazione dell'output in un determinato formato, il manager che si occupa della gestione delle telecamere e dell'acquisizione dei frame mediante esse e così via.

Una volta realizzati tutti questi moduli, utilizzando i design pattern illustrati qui di seguito si riesce a mettere a punto un'architettura robusta, estensibile e facilmente manutenibile. In base alla programmazione orientata agli oggetti, alcune classi fondamentali sono: la classe *Skeleton* per la memorizzazione dei dati relativi a uno skeleton (i punti di giuntura, le rispettive

coordinate, e così via) e la classe *Point* per rappresentare il concetto di punto generico e le classi a partire dalle quali è possibile aggiungere dettagli a tale astrazione. Altre classi importanti sono sicuramente quella per la gestione delle immagini, quella per la gestione dei parametri di input del programma, quella per la gestione della *computer vision*, quella per la conversione fra sistemi di coordinate differenti e infine, una classe che permetta di mettere in comunicazione le varie funzionalità e fungere da interfaccia per il client.

Il diagramma UML delle classi completo del modulo AI Watch A1 non è presente in questo documento per motivi pratici di visualizzazione, ma è consultabile nel file

[AI_Watch_A1/uml/class_diagram.pdf](#)

accedendo al repository GitHub online [99]. In figura 2.8, è mostrato invece il diagramma UML delle classi semplificato: sono state riportate soltanto le classi presenti all'interno del modulo, omettendo attributi e metodi.

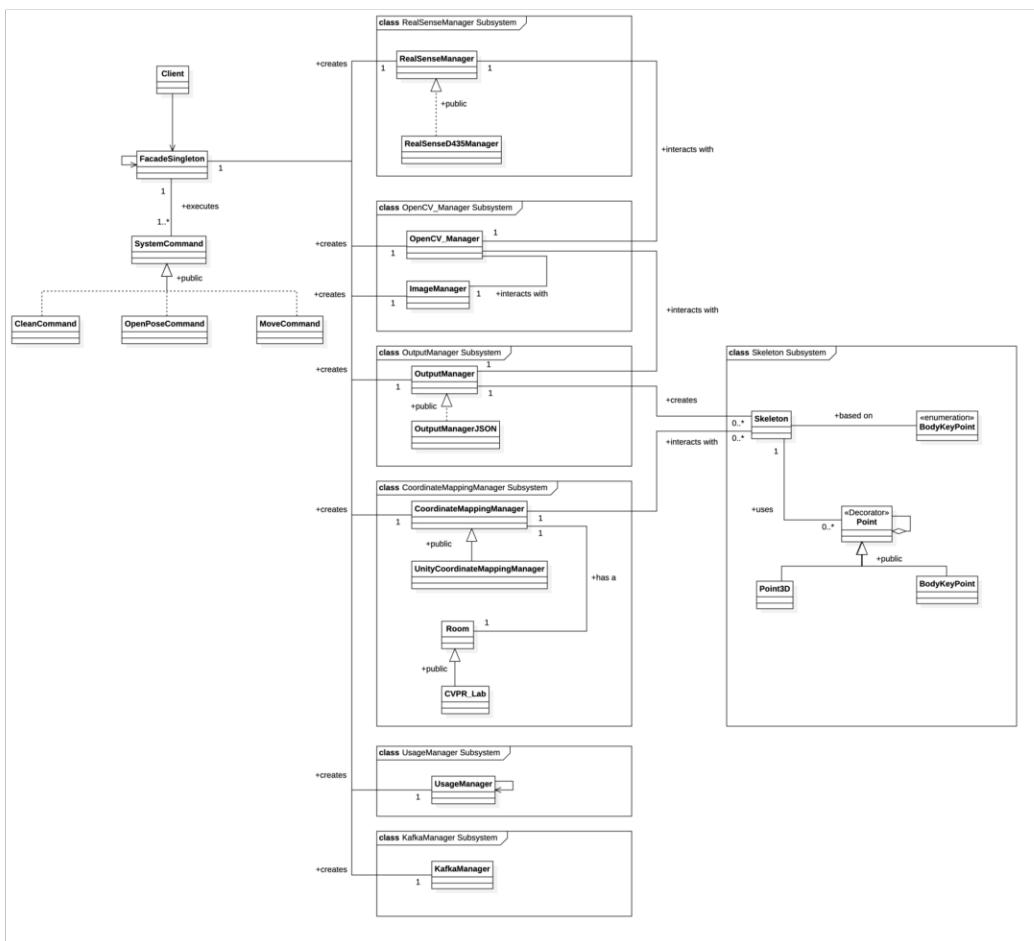


Figura 2.8. Diagramma delle classi AI Watch A1 semplificato.

2.8.1 Design Pattern utilizzati

I design pattern utilizzati nel modulo AI Watch A1 vengono riportati qui di seguito, con una descrizione del contesto in cui sono stati utilizzati e la motivazione che ha portato a utilizzarli.

- Façade: è un pattern strutturale e provvede un’interfaccia unificata che racchiude un insieme d’interfacce in un sottosistema. Façade definisce un’interfaccia di alto livello che rende un intero sottosistema più semplice da usare. Spesso, infatti, viene utilizzato tale design pattern quando si ha un sottosistema che, ai fini della programmazione modulare, coiniosa e riusabile, è composto da tante interfacce piccole e indipendenti. Per realizzare un’interfaccia singola per l’utente e che astragga i meccanismi di funzionamento dell’intero sistema, è utile usare tale pattern. Inoltre, promuove il disaccoppiamento del sistema dai suoi potenziali numerosi utenti. D’altra parte, se la facciata è l’unico punto di accesso per il sottosistema, limiterà le caratteristiche e la flessibilità di cui possono aver bisogno gli “utenti avanzati”. In figura 2.9 è possibile avere un’idea della struttura di questo pattern.

Façade, è stato utilizzato per l’appunto come un’interfaccia di alto livello per il modulo A1 di AI Watch. Infatti, astrae tutto il suo funzionamento mediante delle routine per avviare l’acquisizione dei frame, per eseguire OpenPose, per verificare la correttezza dei parametri in input, per far partire il salvataggio dell’output finale e la trasmissione via Apache Kafka. Se non fosse stato usato Façade, sarebbe stato molto più complesso trovare un modo centralizzato e semplice di raggiungere lo stesso scopo.

- Singleton: è un pattern creazionale e assicura che una classe abbia una e una sola istanza. Inoltre, provvede un unico punto di accesso globale all’istanza di tale classe. Questo pattern, benché sia messo in discussione per alcuni suoi aspetti, è molto utile. Infatti, permette di risolvere due grossi problemi: avere una singola istanza per una determinata classe e avere un solo punto d’accesso per tale istanza. Il primo vantaggio permette di gestire in maniera semplice e ottimale aspetti riguardanti elementi condivisi, connessioni al database, file, e così via. Per il secondo vantaggio, però, a differenza di una variabile globale qualunque, Singleton protegge l’unica e singola istanza della classe dalla sovrascrittura.

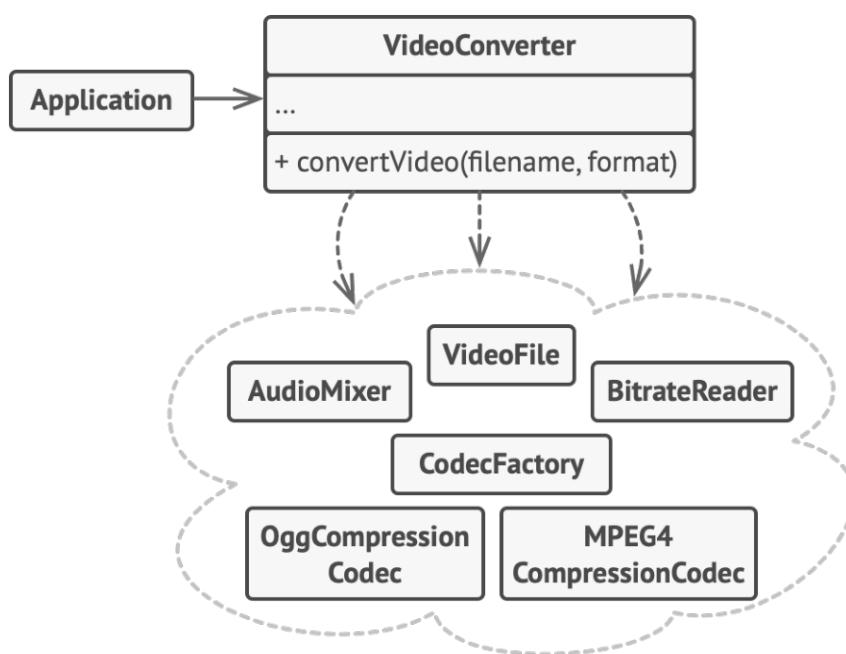


Figura 2.9. Esempio strutturale con Façade per risolvere diverse interdipendenze in un sottomodulo dal funzionamento complesso [49].

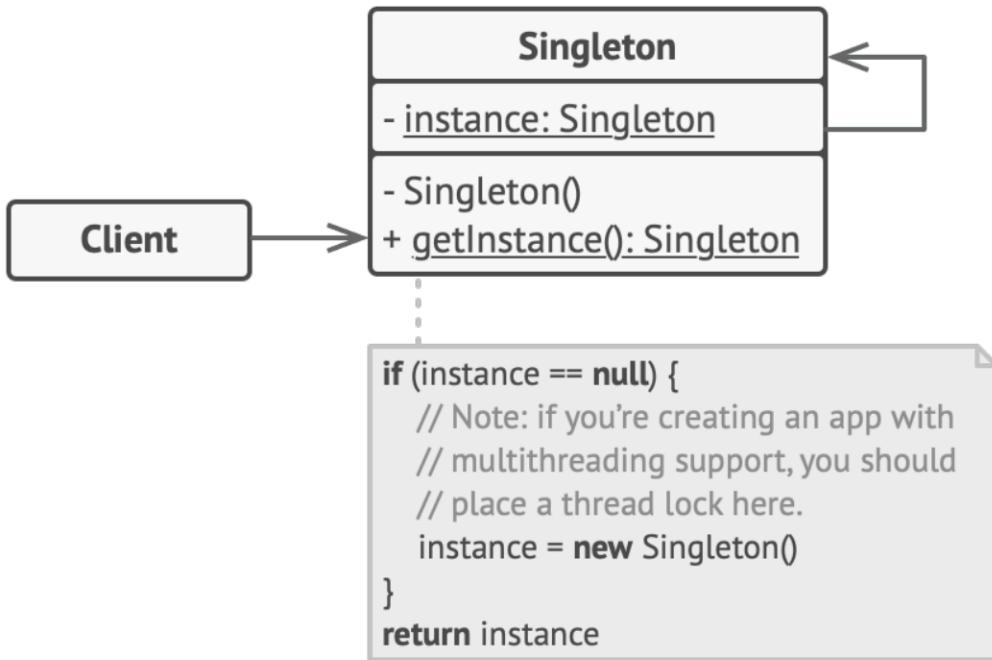


Figura 2.10. Struttura del pattern Singleton [49].

Il pattern Singleton non è stato usato semplicemente da solo, ma è stato abbinato all'utilizzo del pattern Façade. Spesso infatti, un Façade viene utilizzato in combinazione con Singleton, realizzando il noto Façade-Singleton, dal momento che è necessario avere le proprietà di Façade, abbinate alle caratteristiche poc'anzi elencate di un Singleton. In figura 2.10 si illustra la struttura di Singleton.

- Command: è un pattern comportamentale e incapsula una richiesta sotto forma di un oggetto indipendente che contiene tutte le informazioni riguardanti la richiesta fatta. Questa trasformazione da richiesta in oggetto, consente di passare le richieste come argomenti di un metodo, ritardare o accodare l'esecuzione di una richiesta e supportare operazioni annullabili.

Il pattern Command disaccoppia l'oggetto che invoca l'operazione, da quello che sa come eseguirla. Per ottenere questa separazione, il progettista crea una classe di base astratta che mappa un ricevitore (un oggetto) con un'azione (un puntatore a una funzione membro). La classe base contiene solitamente un metodo *execute()* che chiama sem-

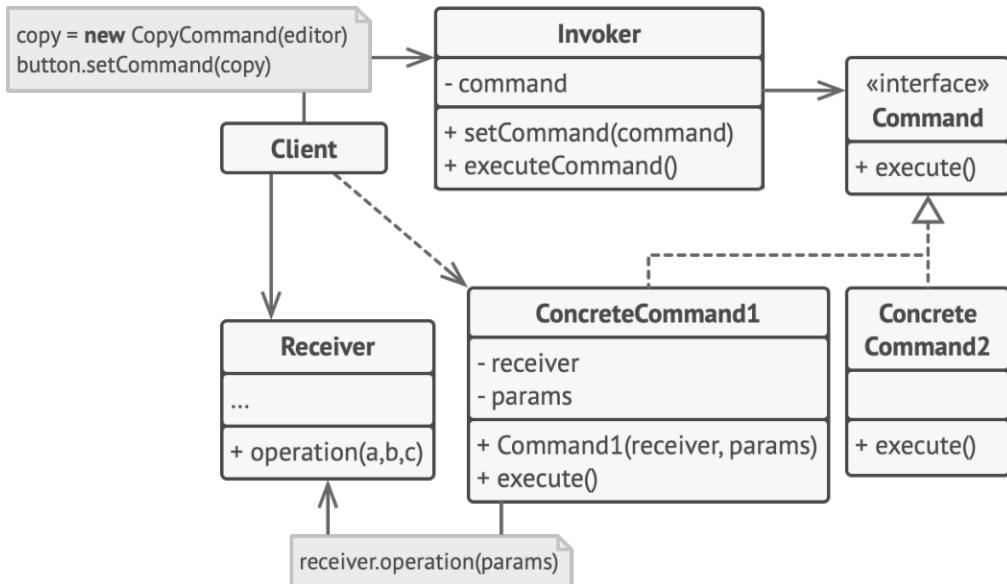


Figura 2.11. Struttura del pattern Command [49].

plicemente l'azione sul ricevitore. Tutti i client degli oggetti Command trattano ogni oggetto come una *scatola nera*, semplicemente invocando il metodo `execute()` dell'oggetto ogni volta che il client richiede la funzionalità. La struttura del pattern Command è mostrata in figura 2.11. Il client che crea un *command* non è lo stesso client che lo esegue. Questa separazione fornisce la flessibilità nella sincronizzazione e nell'ordinamento dei comandi. Materializzare i comandi come oggetti, significa che possono essere passati come parametri, condivisi, caricati in una tabella e strumentalizzati o manipolati all'occorrenza, come qualsiasi altro oggetto.

Il pattern Command è stato utilizzato per astrarre alcune richieste ed esecuzioni di servizi fondamentali per l'esecuzione dell'applicativo. Tra questi è importante ricordare l'esecuzione di OpenPose e il servizio di pulizia per eliminare i file ormai inutilizzati e superflui.

- **Strategy:** è un pattern comportamentale e definisce una famiglia di algoritmi, incapsula ciascuno di essi, e li rende interscambiabili. Il pattern Strategy permette all'algoritmo di essere variato, sostituito e modificato in maniera indipendente dai client che lo utilizzano. Un secondo intento è quello di astrarre un'interfaccia generale, nascondendo i dettagli implementativi nelle classi derivate.

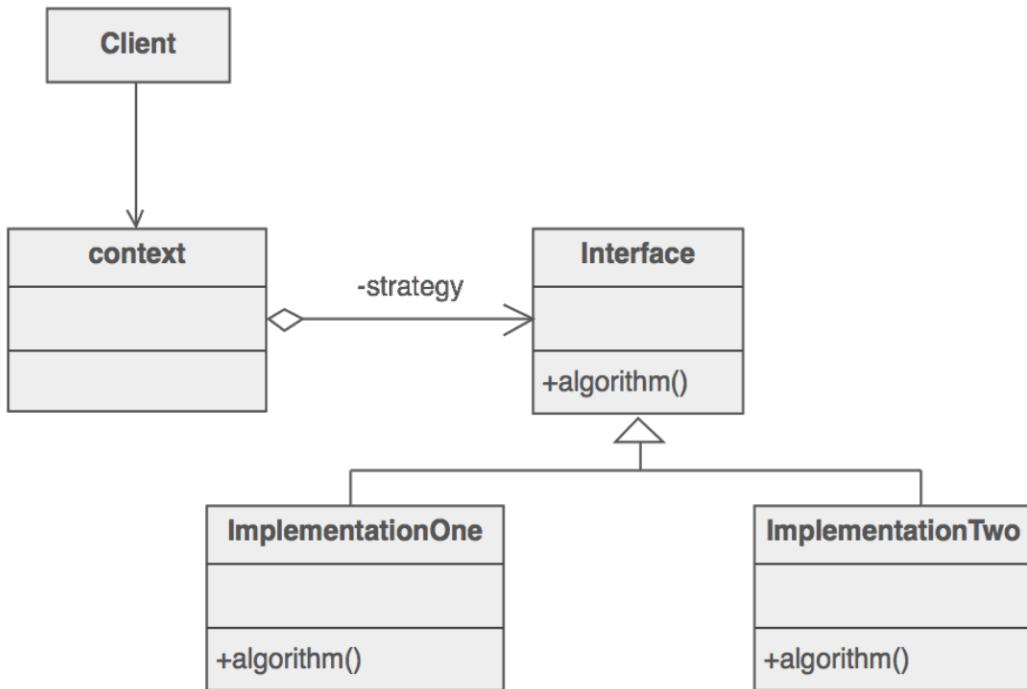


Figura 2.12. Struttura del pattern Strategy [49].

Durante lo sviluppo software si cerca sempre di rispettare il seguente principio: “massimizzare la coesione e ridurre al minimo l’accoppiamento”. L’approccio progettuale orientato agli oggetti è incentrato sulla minimizzazione dell’accoppiamento. Dal momento che il client è accoppiato solo a un’astrazione, e non a una particolare realizzazione di tale astrazione, si può affermare che c’è un “accoppiamento astratto”. Una caratterizzazione più comune di questo principio di “accoppiamento astratto” è: “programma un’interfaccia, non un’implementazione”. I client dovrebbero preferire il *livello aggiuntivo di astrazione* che un’interfaccia o una classe astratta offrono. L’interfaccia presenta l’astrazione di cui il client ha bisogno, e le implementazioni di tale interfaccia sono effettivamente nascoste.

Il pattern Strategy suggerisce di prendere una classe che fa qualcosa di specifico in molti modi diversi ed estrarre tutti questi algoritmi in classi separate chiamate *strategy*. La classe originale, chiamata *context*, deve avere un campo per memorizzare un riferimento a una delle strategie. *Context* delega il lavoro a un oggetto *strategy* collegato, invece di eseguirlo da solo. Il *context* non è responsabile della selezione di un

algoritmo appropriato per il lavoro. Invece, il client passa la strategia desiderata al *context*. Infatti, il *context* non conosce molto circa le *strategy* presenti. Funziona con tutte le *strategy* attraverso la stessa interfaccia generica, che espone solo un singolo metodo per attivare l'algoritmo encapsulato all'interno della *strategy* selezionata. In questo modo il *context* diventa indipendente dalle *strategy* concrete, in modo da poter aggiungere nuovi algoritmi o modificare quelli esistenti senza modificare il codice del contesto o altre *strategy*.

Il pattern Strategy è stato impiegato sia per astrarre a disaccoppiare l'algoritmo di conversione delle coordinate e quindi la trasformazione fra spazi di coordinate tridimensionali, sia per la selezione di un particolare modello di telecamera, sia per un particolare metodo di generazione dell'output da inviare successivamente al modulo successivo di AI Watch e in generale in tutti i casi in cui è possibile scegliere fra più algoritmi o approcci a disposizione per risolvere un determinato problema. La struttura del pattern Strategy è mostrata in figura 2.12.

- Decorator: è un pattern strutturale e consente di collegare nuovi comportamenti agli oggetti, posizionando tali oggetti all'interno di oggetti wrapper speciali, che contengono i comportamenti. L'intento è quindi quello di assegnare dinamicamente ulteriori responsabilità a un oggetto. I *decorator* forniscono un'alternativa flessibile all'uso delle sottoclassi per estendere le funzionalità della classe base. Per fare ciò, si *avvolge* ricorsivamente un oggetto all'interno di uno o più wrapper: una sorta di scatola.

Estendere una classe è la prima cosa che viene in mente quando è necessario modificare il comportamento di un oggetto. Tuttavia, l'ereditarietà ha diversi punti a sfavore di cui si deve essere a conoscenza. Tali punti vengono elencati di seguito.

- L'ereditarietà è statica. Non è possibile modificare il comportamento di un oggetto esistente durante l'esecuzione. Si può solo sostituire l'intero oggetto con un altro creato da una sottoclasse diversa.
- Le sottoclassi possono avere una sola classe genitore. Nella maggior parte dei linguaggi di programmazione, l'ereditarietà non permette a una classe di ereditare i comportamenti di più classi contemporaneamente.

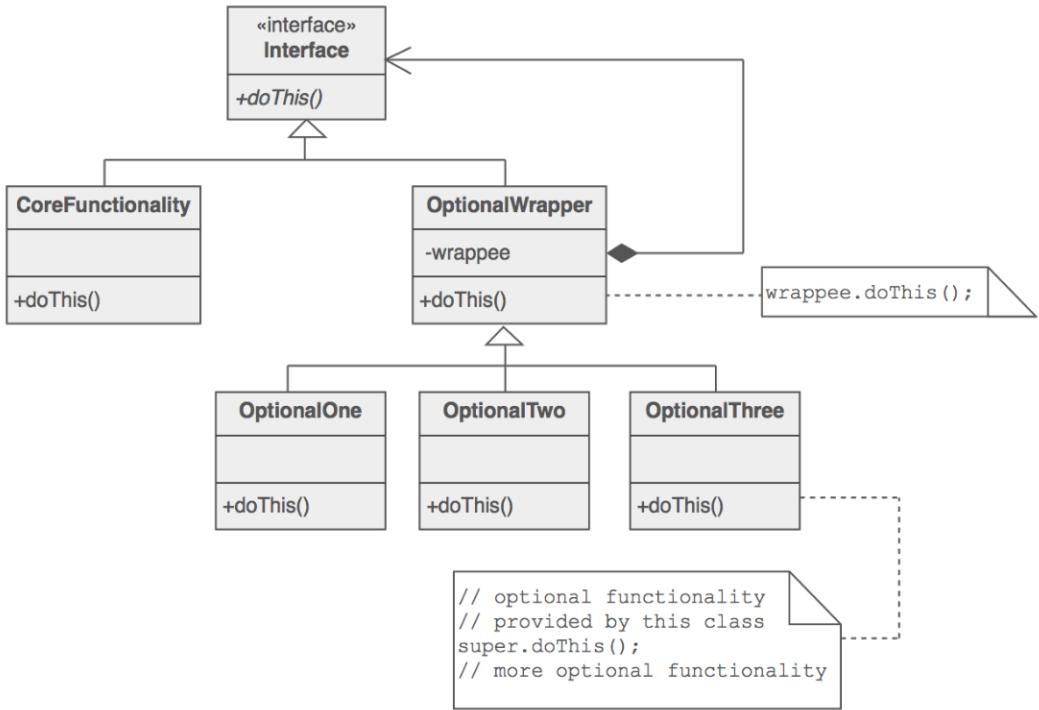


Figura 2.13. Struttura del pattern Decorator [49].

Uno dei modi per superare queste problematiche è usare l’aggregazione o la composizione, invece dell’ereditarietà. Entrambe le alternative funzionano quasi allo stesso modo: un oggetto ha un riferimento a un altro oggetto e delega a quest’ultimo un lavoro; mentre con l’ereditarietà, l’oggetto stesso è in grado di fare quel lavoro, ereditando il comportamento dalla sua classe genitore. Inoltre, sfruttando il pattern Decorator, sarà possibile cambiare il comportamento dell’oggetto durante l’esecuzione.

Un oggetto può utilizzare il comportamento di varie classi, avendo riferimenti a più oggetti e delegando loro tutti i tipi di lavoro. Aggregazione/composizione è il principio chiave dietro molti modelli di progettazione, tra cui Decorator. Dal momento che è possibile *decorare* un oggetto più volte e in maniera differente, l’ultimo Decorator nella pila delle decorazioni effettuate, è l’oggetto con cui il client lavora. Dal momento che tutti i Decorator implementano la stessa interfaccia della classe di base, al resto del codice client non importa se funziona con l’oggetto *puro* della classe, ovvero senza *decorazioni* o quello decorato. La struttura del pattern Decorator è mostrata in figura 2.13.

Il pattern Decorator è stato utilizzato per evitare l'esplosione delle sottoclassi nel caso della definizione e descrizione della classe *Punto* e delle sue derivate. Infatti, sarebbe possibile avere diversi punti come un punto 2D, un punto 3D e un punto che rappresenti un punto di giuntura del corpo. Siccome è necessario anche realizzare le combinazioni delle classi e siccome in futuro potrebbero essere aggiunte altre rappresentazioni di un punto, è opportuno e molto conveniente utilizzare il pattern Decorator.

Capitolo 3

Implementazione e test

Questo capitolo ha lo scopo di esplicitare com'è avvenuta l'implementazione effettiva del modulo A1 in termini pratici. Si chiariranno le principali tecnologie utilizzate, il loro funzionamento e come sono state integrate tra di loro all'interno del modulo A1. Infine, saranno considerati i risultati ottenuti da alcuni test di esecuzione.

3.1 Intel RealSense

La telecamera Intel RealSense D435 è una telecamera della famiglia D400 RealSense e offre uno dei campi visivi più ampi di tutte le fotocamere Intel RealSense. Questa telecamera dispone di un otturatore globale sul sensore di profondità che la rende ideale anche per applicazioni in rapido movimento e rappresenta una soluzione che offre misure della profondità qualitativamente ottimali per diversi campi applicativi. Infatti, il suo ampio campo visivo la rende perfetta per applicazioni afferenti la robotica, la realtà aumentata e virtuale, dove vedere quanto più possibile della scena è di vitale importanza. Con una distanza di visuale fino a dieci metri e con la sua forma compatta, questa telecamera può essere anche integrata in soluzioni già esistenti. Inoltre, grazie al supporto fornito con Intel RealSense SDK 2.0 e al supporto multipiattaforma, lo sviluppo viene semplificato e velocizzato.

Un altro vantaggio di questa telecamera è dato dal fatto che rappresenta una soluzione a basso costo, leggera e potente. Consente lo sviluppo di soluzioni innovative in grado di comprendere l'ambiente circostante e interagire con esso. I sensori dell'otturatore globale forniscono una grande sensibilità alla luce bassa. Questo aspetto consente ai robot, sui quali potenzialmente

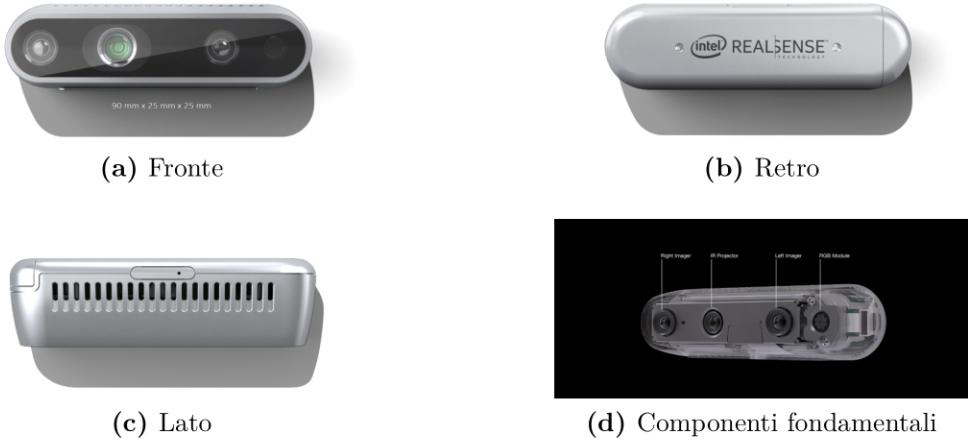


Figura 3.1. Viste differenti della telecamera Intel RealSense D435 [68].

può essere installata la telecamera, di navigare negli spazi che li circondano con le luci spente.

In figura 3.1 si mostrano diverse viste della telecamera D435, mentre in figura 3.2 si mostra com’è strutturata internamente la telecamera D430. Come si nota nelle figure poc’anzi menzionate, la telecamera Intel RealSense D430 e D435 risultano molto simili tra di loro e sono formate da diverse componenti. Quelle principali, sulle quali è importante soffermarsi, sono il *left imager*, il *right imager*, il modulo RGB e l’emettitore di raggi infrarossi.

Ora si prenderanno in considerazione le principali specifiche tecniche della telecamera Intel RealSense D435, così da chiarire campi e contesti applicativi. La telecamera, come riportato dal sito ufficiale d’Intel [68], fornisce validi risultati sia in ambienti interni che esterni; il range di utilizzo ideale è compreso fra i 0.3 metri e i 3 metri. La tecnologia adottata per la profondità è di tipo stereoscopico, ovvero una tecnica di realizzazione e visione d’immagini, disegni, fotografie e filmati, atta a trasmettere un’illusione di tridimensionalità, analoga a quella generata dalla visione binoculare del sistema visivo umano. Per quanto concerne il canale in grado di rilevare la profondità, il campo visivo è di $87^\circ \times 58^\circ$, la minima distanza rilevabile è fissata a 0.28 metri, la risoluzione massima raggiungibile è 1280x720, quella consigliata per questo modello è 848x480, il massimo frame rate è al più 90 FPS, mentre l’errore commesso sulla profondità è inferiore al 2% a una distanza di due metri.

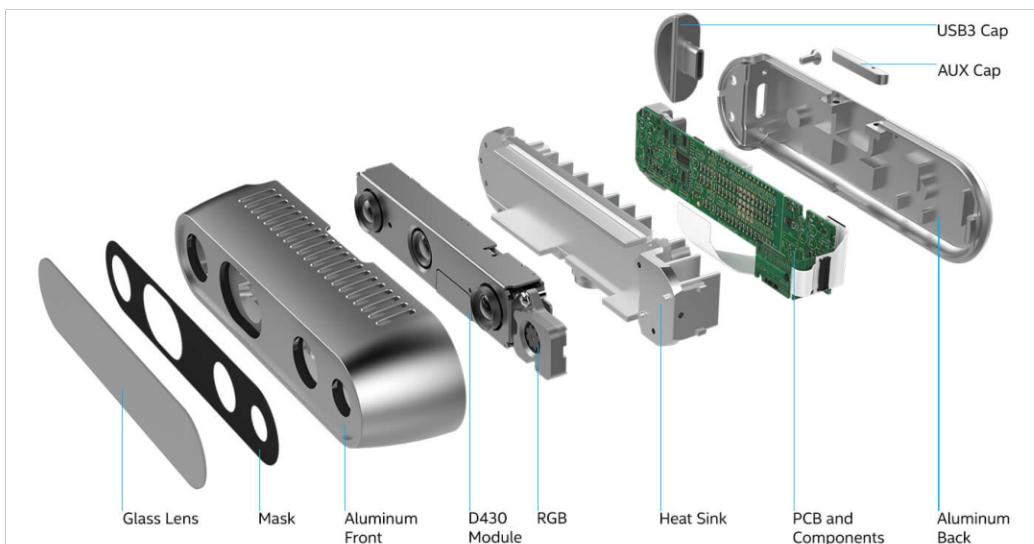


Figura 3.2. Architettura interna della telecamera Intel RealSense D430 [68].

Per quanto riguarda il canale RGB, la risoluzione massima del frame è 1920x1080, quella consigliata per questo modello è 848x480, il campo visivo è di 69°x42° (H x V), il frame rate è al più 30 FPS, la risoluzione del sensore RGB è di 2 MP, mentre l'otturatore è di tipo “rolling shutter”. Purtroppo con un otturatore di questo tipo, e non di tipo globale, si ha l'effetto rolling shutter: un oggetto che si muove rapidamente apparirà con leggere distorsioni nell'immagine acquisita. Questo fenomeno si verifica quando si scatta una foto dal finestrino di un treno: gli oggetti con forti forme verticali appariranno piegati nella parte superiore o inferiore dell'immagine. Questo può capitare anche se i soggetti sono fermi e si esegue un rapido movimento della fotocamera. La distorsione del rolling shutter influisce anche sulla registrazione video.

In figura 3.3 si riporta un esempio dell'effetto rolling shutter. Ulteriori dettagli e specifiche tecniche riguardanti la telecamera Intel RealSense D435 sono consultabili al seguente riferimento [97].

Le telecamere Intel RealSense sono dispositivi *plug-and-play*; ovvero una volta collegate al calcolatore è già possibile iniziare a utilizzarle, sviluppando i primi programmi con l'ausilio del SDK 2.0 fornito da Intel [101] o testando il funzionamento delle telecamere con programmi d'esempio come Intel RealSense Viewer [102] e simili.



Figura 3.3. Effetto Rolling Shutter [100].

	Tracking	Depth Quality
Field of View	Wide	Narrow
Light Spectrum	Narrow	Wide

Tabella 3.1. Differenze tra telecamere orientate al “field of view” oppure al “light spectrum”.

Quando si parla d’immagini contenenti informazioni sulla profondità degli oggetti nella scena inquadrata, non si può non menzionare il dislivello esistente tra *qualità della profondità rilevata e tracciamento di persone e/o oggetti* che è possibile realizzare. Si consideri la tabella 3.1. Una telecamera orientata al *field of view* (FOV), ha vantaggi che una telecamera orientata al *light spectrum* non ha e viceversa. Una soluzione ottimale è quella di adottare una telecamera con un ampio FOV, ma dotata di filtro infrarossi. Le telecamere Intel RealSense adottano esattamente questo approccio.

La tecnologia delle telecamere Intel RealSense D435 è integrabile con l’unità di elaborazione *Movidius*. Quest’ultima permette di adempiere task avanzati di computer vision, evitando l’utilizzo eccessivo delle risorse del sistema. In ultima analisi, si precisa che la telecamera Intel RealSense D435 è in grado di utilizzare sia la tecnologia laser che quella a infrarossi, al fine di ottenere l’informazione sulla profondità. Le differenze tra i due approcci, saranno trattate successivamente in questa sezione. Inoltre, al fine di ottenere una precisione molto maggiore, le telecamere Intel RealSense fanno utilizzo dei concetti di geometria epipolare introdotti nella sottosezione 1.4.3.

La telecamera Intel RealSense T265 è visibile in figura 3.4 e si tratta di un’altra telecamera Intel, appositamente progettata per attività di tracking e detection. Infatti, ha a disposizione un field of view molto esteso e ciò rende le operazioni di tracking e detection molto precise. Viceversa, la telecamera Intel RealSense D435 ha un field of view più ristretto e, di conseguenza, è maggiormente utilizzata per domini applicativi ove è necessaria molta più precisione sull’informazione relativa alla profondità. Di contro, il modello T265 non è una telecamera propriamente pensata per ricavare informazioni sulla profondità e non fornisce dati a essa relativi. Una possibile applicazione allo stato dell’arte è quella di abbinare una telecamera Intel RealSense D435 (anche modelli precedenti o successivi), con una telecamera Intel RealSense T265. Così facendo, è possibile rilevare la profondità dalla scena con molta precisione e, al tempo stesso, assolvere in maniera molto dettagliata e molto precisa i compiti di tracking e di detection.



Figura 3.4. Telecamera Intel RealSense T265 [68].

In diversi modelli di telecamere Intel RealSense è presente un sensore IMU che permette di ricavare informazioni anche in merito all'orientazione della telecamera all'interno dell'ambiente circostante, così come di ciò che viene osservato nella scena. Il modello di telecamera Intel RealSense D435 non dispone di questo ulteriore modulo, a differenza di altri modelli come D435i e il modello T265. Il sensore IMU (Inertial Measurements Units) serve a fornire misurazioni accurate anche in presenza di rotazioni e/o grandi velocità. Si tratta di un dispositivo elettronico utilizzato per calcolare la forza espressa dal corpo, la velocità angolare e la sua direzione. Tutto ciò può essere ottenuto utilizzando una miscela di due sensori: giroscopio e accelerometro. Alcuni dispositivi IMU sono dotati anche di un magnetometro. Questi sensori sono normalmente utilizzati per pianificare aerei UAV (veicoli aerei senza pilota) e veicoli spaziali, compresi lander e satelliti. Inoltre, il sensore IMU è alla base dell'risoluzione del problema computazionale *Simultaneous localization and mapping* (SLAM).

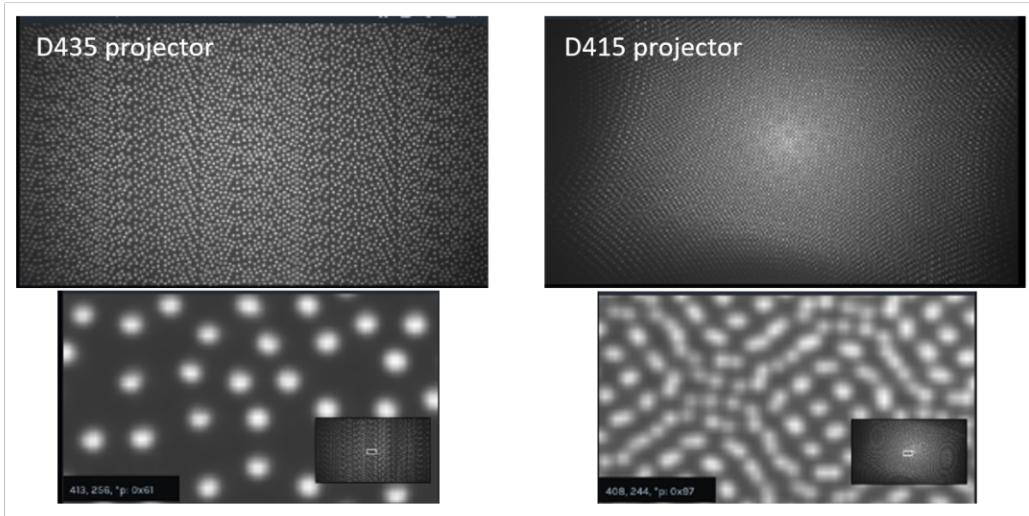


Figura 3.5. Pattern circolari emessi da Intel RealSense D415 e D435 [103].

La procedura di determinazione dei punti di equivalenza, al fine di ricavare la profondità per ogni pixel dell’immagine acquisita della scena inquadrata, diventa più complicata quando la scena è a “tinta unita”, per esempio una parete monocromatica. In tal caso, si sfruttano le potenzialità dei raggi infrarossi: la telecamera trasmette dei pattern solitamente circolari in maniera pseudo-casuale sulla scena e, in base alla deformazione subita dai pattern circolari nella scena, se ne ricava la profondità. Queste deformazioni fanno sì che a un pattern circolare più piccolo, in termini di raggio, sarà associata una profondità maggiore; mentre a un pattern circolare più grande, in termini di raggio, sarà associata una profondità inferiore. Si precisa che tutti i pattern circolari proiettati sono dello stesso raggio. Questo approccio è usato dalla famiglia di telecamere Intel RealSense e permette di ottenere risultati migliori, anche in situazioni particolari dal punto di vista della luce, della cromaticità e quant’altro. In generale, i raggi infrarossi vengono sfruttati per avere informazioni sulla tessitura (o maglia) degli oggetti. In figura 3.5 si mostrano i pattern circolari emessi dalle telecamere Intel RealSense D415 e D435.

Si supponga di coprire una delle due telecamere di imaging: *left imager* o *right imager*. In base a quanto detto nella sottosezione 1.4.3, ciò che si otterrà è la perdita dell’informazione sulla profondità. Se invece si copre il sensore infrarossi, allora l’immagine continua a essere prodotta e visibile, ma la qualità dell’immagine cala in maniera drastica. Questo perché si perdono alcune informazioni fondamentali sulla tessitura degli oggetti; per esempio

le pareti monocromatiche risulteranno molto rumorose. Inoltre, si precisa che il dispositivo delegato ai raggi infrarossi non è soltanto un emettitore di raggi infrarossi, ma anche un sensore di raggi infrarossi. Così facendo è in grado di emettere i pattern circolari di cui prima, e successivamente dedurne le deformazioni quando questi vengono riflessi. Ecco il motivo per cui la riflettanza è così importante.

In ambienti esterni, il contributo dei raggi infrarossi, come già accennato in precedenza, è abbastanza limitato, se non del tutto assente se la telecamera inquadra il Sole. Quindi, in ambienti esterni, solitamente la telecamera fa molto più affidamento sulla visione stereoscopica. Infatti, l'idea è quella di migliorare e integrare il rilevamento della profondità sia mediante i concetti di geometria epipolare, che mediante un insieme di sensori di alto livello.

La riflessione dei pattern circolari, acquisita dal sensore infrarossi di tipo CMOS [50] della telecamera Intel RealSense, è usata per calcolare la profondità in ogni punto della scena e assegnarla, di volta in volta, a uno specifico pixel. Per fare ciò, si confronta ciò che è stato acquisito dalla riflessione con ciò che è stato proiettato dall'emettitore di raggi infrarossi. A tal punto si sfruttano i metodi trigonometrici e di triangolarizzazione visti in precedenza. Questo è anche il principio di funzionamento del Kinect nella sua versione *v1*. Invece, nella versione *v2* si sfrutta il *ToF*, ovvero il *time of flight*. Il concetto è simile a quello delle telecamere Intel RealSense.

Tutta la computazione relativa allo stereo-matching, ovvero attività come la triangolarizzazione per corrispondenza tra le due immagini e conseguente ottenimento della profondità, l'emissione e il rilevamento delle deformazioni relative ai pattern circolari basati su raggi infrarossi, il miglioramento delle tessiture, l'abbinamento del frame RGB con quello della profondità, il pre-processing e il post-processing; sono tutte effettuate interamente sul dispositivo Intel RealSense, permettendo così al calcolatore al quale viene collegato, di realizzare solo la fase di rendering del video. Questo aspetto è molto importante! Infatti, qualora dovessimo usare parte delle risorse del calcolatore per effettuare tutte le operazioni poc'anzi descritte o solo una frazione di esse, l'intero processo sarebbe di gran lunga rallentato e questo soprattutto se il calcolatore, nel mentre, è utilizzato anche per altre attività. Inoltre, questo aspetto è utile anche per applicativi in campo IoT ed embedded. Approcci alternativi all'utilizzo della tecnologia a infrarossi sono il laser e il lidar [104]. Si tratta di tecnologie che permettono di raggiungere un grado di precisione maggiore, ma sono anche di gran lunga più costose.

Una telecamera che fa affidamento sul ToF, è composta da alcune componenti principali: lenti, sorgente luminosa integrata, sensore che cattura tutte le informazioni dell'immagine e un'interfaccia (e.g. USB) per il collegamento esterno. Questo approccio e con queste componenti, permette di ottenere informazioni sia sulla profondità degli oggetti nella scena, che sul livello d'intensità da memorizzare per ogni pixel dell'immagine. L'idea di funzionamento è ancora una volta quella di emettere un fascio luminoso sulla scena, per poi osservare e determinare la luce riflessa dagli oggetti. Siccome la velocità della luce è nota, allora la distanza degli oggetti nella scena è facilmente ricavabile per ogni punto della scena. L'uso del ToF comporta alcuni vantaggi, ma anche svantaggi; per esempio la sensibilità molto maggiore alla luce del Sole rispetto alla tecnologia infrarossi o laser, riflessioni multiple e conseguenti artefatti nell'immagine risultato.

Infine, per quanto riguarda la tecnologia laser, questa permette di raggiungere livelli di precisione molto maggiori rispetto alla tecnologia infrarossi. Alcune tecnologie laser possono far uso a loro volta di tecnologia infrarossi, oppure essere basate sull'ultravioletto o sulla luce visibile. Invece, per quanto riguarda la tecnologia lidar (light detection and ranging), questa identifica la tecnologia che misura la distanza da un oggetto illuminandolo con una luce laser e che, al contempo, è in grado di restituire informazioni tridimensionali e ad alta risoluzione sull'ambiente circostante. Un lidar sfrutta tipicamente diversi componenti: laser, fotorilevatori e circuiti integrati di lettura (ROIC) con capacità di tempo di volo (TOF), in modo da misurare la distanza illuminando un bersaglio e analizzando la luce da esso riflessa. Di base il lidar è una tecnica simile a un radar: è basato sul principio dell'eco. Un'immagine ottenuta con la tecnologia lidar è visibile in figura 3.6.

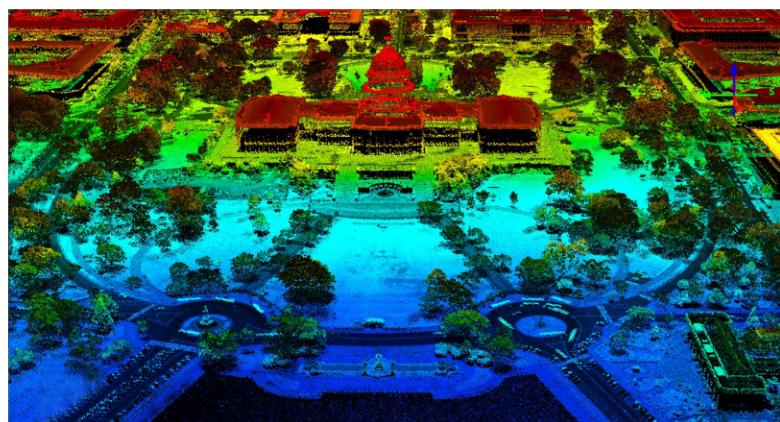


Figura 3.6. Immagine lidar [103].

3.1.1 Sistema di coordinate tridimensionale Intel RealSense

Una telecamera Intel RealSense in grado di rilevare informazioni sulla profondità, permette di svolgere alcune operazioni molto interessanti, come la determinazione delle coordinate (x, y, z) di un punto specifico all'interno della scena osservata. Si consideri che si tratta di punti nello spazio, quindi caratterizzati da tre coordinate. In questa sottosezione si analizzeranno le proiezioni matematiche che mettono in relazione le immagini fornite dalle telecamere RealSense, al loro sistema di coordinate 3D associato.

Ogni flusso d'immagini fornito dal SDK d'Intel RealSense, è associato con un sistema di coordinate bidimensionale separato e specificato in pixel. La coordinata $(0, 0)$ si riferisce al centro del pixel in alto a sinistra (o alternativamente al pixel in alto a sinistra in quanto tale); mentre la coordinata $(w - 1, h - 1)$, si riferisce al pixel in basso a destra dell'immagine. Di conseguenza l'immagine contiene w colonne e h righe. Quindi l'asse x e l'asse y sono strutturati nel seguente modo: l'asse x va da sinistra verso destra, mentre l'asse y va dall'alto verso il basso. Queste coordinate sono chiamate, all'interno di questo sistema, coordinate dei pixel: servono per indicizzare l'immagine e trovare il contenuto di determinati pixel in essa.

Ogni flusso d'immagini fornito dal SDK d'Intel RealSense, è associato anche con un sistema di coordinate tridimensionale separato. Tale spazio è specificato in metri con la coordinata $(0, 0, 0)$ che si riferisce idealmente al centro del dispositivo di acquisizione. All'interno di questo spazio, l'asse x positivo punta a destra, l'asse y positivo punta verso il basso, mentre l'asse z positivo punta in avanti. Le coordinate in questo spazio sono indicate come punti e sono usate per descrivere le posizioni all'interno di uno spazio tridimensionale ricavato dalle informazioni presenti all'interno di una particolare immagine.

I sistemi di coordinate (SC) 2D e 3D di una telecamera Intel RealSense, sono entrambi basati su alcuni parametri noti: i parametri intrinseci e i parametri estrinseci della telecamera stessa. La correlazione tra uno stream 2D e uno stream 3D rispetto al sistema di coordinate adoperato, è descritta dai parametri intrinseci della telecamera. Ogni modello di telecamera Intel RealSense è differente in qualcosa dagli altri modelli. In generale però, vengono fatte le seguenti assunzioni di base in merito ai parametri intrinseci.

- Le immagini acquisite possono essere di dimensioni differenti.

- IL FOV con cui è stata acquisita un’immagine può essere differente da quello con cui è stata acquisita un’altra immagine.
- I pixel di un’immagine non sono necessariamente quadrati.
- Il centro di proiezione non è necessariamente il centro dell’immagine.
- L’immagine può presentare delle distorsioni.

Si precisa che la telecamera, per ognuna di queste assunzioni, fornisce delle informazioni al riguardo. Per esempio, le dimensioni effettive dell’immagine acquisita, la lunghezza focale in fase di acquisizione, le dimensioni del singolo pixel considerato, le coordinate del centro di proiezione e il modello di distorsione eventualmente presente nell’immagine e quale modello di distorsione è stato usato per calibrare la telecamera. Una volta che i parametri intrinseci della telecamera sono noti, è possibile effettuare operazioni particolari e fondamentali, dette *operazioni di mapping*, quali la *projection* e la *deprojection*. La tabella 3.2 mostra in cosa consistono queste due operazioni appena menzionate.

Operazione	Descrizione
Projection	Si considera un punto 3D e lo si associa (<i>3D to 2D coordinate mapping</i>) a un pixel (coordinate 2D) sul flusso immagine bidimensionale.
Deprojection	Si considera un pixel sul flusso immagine bidimensionale con un profondità specificata in metri e si associano queste informazioni a un determinato punto 3D nello spazio delle coordinate tridimensionale.

Tabella 3.2. Operazioni di *projection* e *deprojection* a confronto.

Invece, per quanto riguarda i modelli di distorsione citati in precedenza e con cui viene acquisita un’immagine, ne sono disponibili molteplici. I principali modelli di distorsione adoperati nelle telecamere Intel RealSense sono il modello di distorsione modificato di Brown-Conrady e il modello di distorsione inversa di Brown-Conrady. Inoltre, è possibile che l’immagine non presenti distorsioni in fase di acquisizione. Tale tipo d’immagine viene prodotta da una telecamera *pin-hole* ideale ed è il risultato di un algoritmo hardware e software che si occupa della rimozione delle distorsioni da un’immagine; oppure, è ottenuta dal fatto che l’immagine considerata è derivata da un’immagine o più, già senza distorsioni. Conoscere se e con quale modello di distorsione è stata acquisita l’immagine, è di fondamentale importanza, perché determina la possibilità o l’impossibilità, in alcune circostanze, di eseguire

le operazioni di *projection* e di *deprojection*. Sebbene vi sia questa problematica però, le telecamere Intel RealSense supportano sempre l'operazione di *deprojection* da un'immagine contenente informazioni sulla profondità e l'operazione di *projection* verso immagini a colori. Di conseguenza, è sempre possibile associare un'immagine contenente informazioni sulla profondità a un insieme di punti 3D, anche detto *point cloud*; ed è sempre possibile scoprire dove un oggetto 3D appare nella corrispettiva immagine a colori.

Oltre alla variazione dei parametri intrinseci, il sistema di coordinate 3D di ogni flusso video può essere differente. Per esempio, è comune per la profondità, essere valutata da uno o più sensori a infrarossi; mentre il flusso video del colore è fornito da un dispositivo di acquisizione dell'immagine differente. La relazione fra i diversi sistemi di coordinate tridimensionali è descritta dai parametri estrinseci. In generale, vengono fatte le seguenti assunzioni di base in merito ai parametri estrinseci.

- Diversi dispositivi di imaging possono essere situati in posizioni differenti e distanti, ma sono *montati* sullo (o alternativamente fanno parte dello) stesso dispositivo fisico. La telecamera conosce la posizione esatta di entrambi i dispositivi grazie a specifici parametri impostati in fase di costruzione.
- Analogamente, l'orientazione dei dispositivi di imaging può essere differente, ma sono pur sempre *montati* sullo (o alternativamente fanno parte dello) stesso dispositivo fisico.
- Tutti i sistemi di coordinate 3D sono specificati in metri: non c'è necessità di scalare i valori nella trasformazione tra sistemi di coordinate mediante l'uso delle operazioni di *projection* e *deprojection*.
- Tutti i sistemi di coordinate sono destrorsi e con base ortogonale.

Conoscere i parametri estrinseci esistenti tra due flussi video, dà la possibilità di trasformare punti da uno spazio di coordinate tridimensionale all'altro. I parametri estrinseci e intrinseci, così come altri dettagli hardware, possono essere ricavati mediante opportune routine fornite dall'Intel RealSense SDK 2.0. Inoltre, si precisa che i parametri estrinseci sono indipendenti dal contenuto della scena inquadrata e quindi costanti per l'intero tempo di esecuzione del programma che sfrutta le telecamere.

Il *mapping* da un pixel 2D a un punto 3D tramite l'operazione di *deprojection*, richiede la conoscenza della profondità di quel pixel in metri. Alcuni formati d'immagine, grazie a Intel RealSense SDK 2.0, contengono informazioni sulla profondità per ogni pixel e possono essere immediatamente sfruttate con questa operazione. Altre immagini, però, non contengono l'informazione sulla profondità per ogni pixel e quindi va eseguita prima l'operazione di *projection*, anziché eseguire subito l'operazione di *deprojection*. Se quando si ricava la profondità da un'immagine per un determinato pixel si ottiene il valore *zero*, vuol dire che molto probabilmente c'è stato un errore durante la rilevazione, dal momento che tutti i pixel con una profondità pari a zero corrispondono alla stessa posizione fisica del centro del dispositivo di imaging stesso.

All'interno dell'Intel RealSense SDK 2.0 è possibile reperire anche dei blocchi predefiniti e pronti all'uso per estrarre la cosiddetta nuvola di punti (point cloud), visibile nel programma *Intel RealSense Viewer*. Analogamente, sono presenti gli strumenti specifici per realizzare un allineamento tra le varie tipologie di frame che quel determinato modello di telecamera è in grado di acquisire. Così facendo, è molto più semplice associare un pixel di un'immagine profondità a un pixel di un'immagine RGB.

Infine, è doveroso specificare che le immagini prodotte dai sensori infrarossi a sinistra e a destra, nelle telecamere della famiglia Intel RealSense D400, sono già rettificate e ripulite dalle distorsioni della telecamera. Quindi i due sensori e i due flussi hanno caratteristiche intrinseche uguali, non vi sono distorsioni, non vi sono rotazioni; mentre la traslazione è presente solo lungo uno dei tre assi del sistema di coordinate tridimensionale: l'asse *x*.

3.1.2 Hole-filling spaziale

Quando si visualizzano delle immagini di profondità, è possibile notare diverse zone di colore nero. Tecnicamente queste zone vengono dette *buchi* o *fori*. Vi sono alcuni metodi di riempimento di fori molto semplici all'interno d'Intel RealSense SDK 2.0. Un metodo molto semplice consiste nell'utilizzare pixel validi vicini a destra o a sinistra entro un raggio specificato, per riempire il foro. Questo approccio è implementato nella libreria *librealsense* nella fase di filtraggio spaziale per considerazioni di efficienza. Un altro semplice metodo di riempimento dei fori è quello di utilizzare pixel validi a sinistra per riempire il foro, poiché l'algoritmo di ricostruzione dell'immagine fa riferimento all'imager sinistro. Questo algoritmo è implementato nella libreria *librealsense* come un blocco di elaborazione separato, chiamato *hole filling filter*. In totale vi sono tre modalità per il riempimento di un foro all'interno del SDK d'Intel RealSense e che vengono elencate qui di seguito.

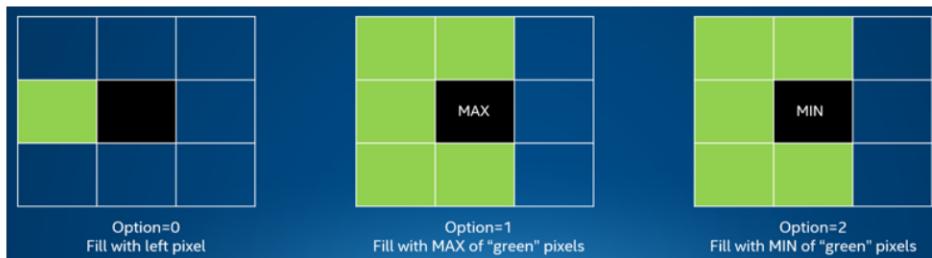


Figura 3.7. Tre approcci differenti per effettuare il riempimento dei fori nelle immagini acquisite con telecamera Intel RealSense [68].

- Sostituire il foro con un valore valido d'intensità di un pixel a sinistra.
- Sostituire il foro con il valore d'intensità più grande tra quelli presenti in un intorno 3×3 del pixel e che si trovi all'interno di un pixel al di sopra, al di sotto o a sinistra di quello di partenza. Questo metodo è applicato solitamente con le immagini di profondità.
- Sostituire il foro con il valore d'intensità più piccolo tra quelli presenti in un intorno 3×3 del pixel e che si trovi all'interno di un pixel al di sopra, al di sotto o a sinistra di quello di partenza. Questo metodo è applicato solitamente con le mappa di disparità.

Le tecniche di *hole-filling spaziale* vengono usate in AI Watch A1 al fine di ridurre i fori di colore nero all'interno delle immagini di profondità elaborate. In particolare viene utilizzata la seconda modalità, fra quelle poc'anzi menzionate, in quanto maggiormente adatta per le immagini di profondità. In figura 3.7, vengono ricapitolati i tre metodi analizzati poc'anzi.

3.2 Convolutional Pose Machines

Le *pose machines* provvedono a un framework di predizione sequenziale per l'apprendimento di modelli spaziali implicitamente complessi. L'idea è quella di sfruttare le reti convoluzionali ai fini dell'apprendimento delle *feature* nelle immagini, e di usare i modelli spaziali dipendenti dalle immagini per il conseguimento della *pose estimation*. In particolare, si può raggiungere tale scopo progettando un'architettura sequenziale composta da reti convoluzionali che operano direttamente su mappe di affidabilità, dette *beliefs map* e immagini delle feature ottenute dai passaggi precedenti, producendo man mano delle stime sempre più raffinate per la localizzazione delle varie parti del corpo di un umano. In figura 3.8 si mostra come agisce una tipica *convolutional pose machine*, che è detta tale nel momento in cui una *pose machine* viene integrata con delle reti convoluzionali [51].

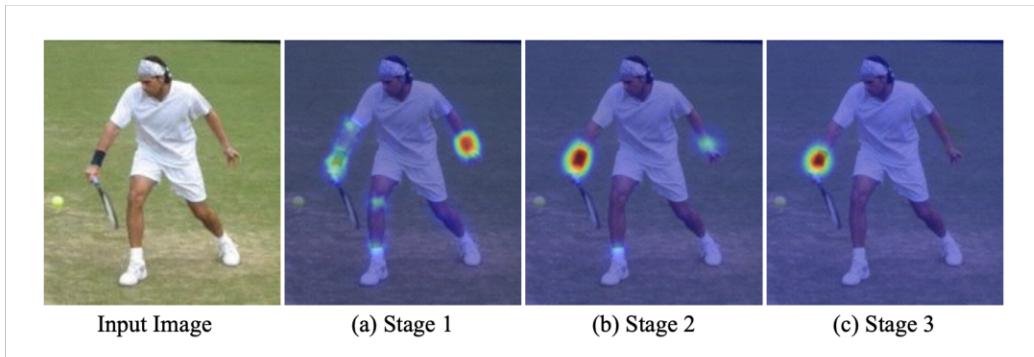


Figura 3.8. Una *convolutional pose machine* consiste di una sequenza di *predictors* allenati in modo da fare predizioni accurate per ogni immagine data in input. Si mostra come passo dopo passo, si ottenga un raffinamento sempre maggiore per la posizione del polso destro: (a) passo 1 in cui si sfruttano i dati locali che causano qualche errore; (b) le ambiguità vengono risolte grazie alla contestualizzazione per zone dell'immagine; (c) le iterazioni successive e la contestualizzazione globale permettono di avere un rilevamento molto accurato del gomito destro [52].

L'approccio classico teorico per la *pose estimation* è quello di sfruttare il modello basato sulle *pictorial structure* [53, 54, 55, 56, 57, 58, 59, 60]. In questo modello, le correlazioni spaziali tra le parti del corpo sono espresse graficamente con una struttura ad albero che presenta una priorità per il collegamento dei punti che hanno una maggiore probabilità di essere effettivamente parte dello skeleton nell'immagine data in input. Ovvero, si cerca di dedurre dall'immagine in input, dove si trovano gli arti, i collegamenti tra

le possibili giunture e si connettono opportunamente tali punti in modo da formare una struttura ad albero. Questi approcci sono ottimali in immagini in cui tutta la persona è ben visibile, ma sono sensibili all'errore in caso contrario. Per capire quali dei possibili collegamenti è quello che ha maggiore probabilità di esistere effettivamente, si considerano le relazioni tra i punti di giuntura e tra i vari possibili link tra cui è possibile scegliere e rispetto a quelli già scelti.

Si denoti la posizione di un pixel del punto di giuntura p -esimo (noto anche come parte), $Y_p \in Z \subset \mathbb{R}^2$, dove Z è l'insieme di tutte le (u, v) posizioni in un'immagine. L'obiettivo di una *convolutional pose machine* [52], è quello di predire le posizioni all'interno dell'immagine, denotate dall'insieme Y , dove $Y = (Y_1, \dots, Y_P)$. A ogni iterazione $t \in (1, \dots, T)$ si predice la più probabile assegnazione per ogni giuntura, basandosi sulle *feature*, su delle *mappe* particolari, dette *belief map* e sulle informazioni del contesto attualmente analizzato. Infine, si collegano le giunture che con maggior probabilità rispecchiano lo skeleton del soggetto nell'immagine data in input.

Sebbene una *convolutional pose machine* sia un approccio molto avanzato e preciso per il rilevamento dello skeleton, alcune problematiche sorgono in occasioni dove molti soggetti nella stessa immagine o video sono molto vicini tra di loro.

Per maggiori informazioni in merito alle *pose machine* e alle *convolutional pose machine*, si consultino rispettivamente i seguenti riferimenti [52, 53].

3.2.1 OpenPose

OpenPose è il primo sistema multipersona in grado di rilevare contemporaneamente il corpo umano, la mano, il viso, e i piedi in singole immagini o video; per un totale di 135 punti di giuntura. Si tratta di un approccio molto avanzato tecnologicamente parlando. Infatti, rilevamenti parziali o completi vengono effettuati anche quando la persona presente nella scena inquadrata si trova di profilo. Inoltre, è presente un plug-in per essere integrato con Unity. Di seguito si elencano le diverse funzionalità messe a disposizione da OpenPose.

- Rilevamento (detection) dei punti di giuntura in modalità bidimensionale, real-time e multipersona. La stima di ogni skeleton è realizzabile con 15, 18 oppure 25 punti di giuntura. Di questi punti di giuntura, ben sei in tutto sono riservati per i piedi. Questa tipologia di rilevamento fa riferimento al corpo intero della persona, ed è detta *body detection* o *skeleton detection*.
- Rilevamento (detection) di 2×21 punti di giuntura per le mani. Ancora una volta l'approccio è in modalità bidimensionale, real-time e multipersona. Di conseguenza ben 21 punti di giuntura fanno riferimento a una mano e altri 21 punti di giuntura fanno riferimento all'altra mano di ogni coppia di mani individuata nella scena. Si precisa che, le attività di rilevamento dei punti di giuntura e dei collegamenti, nel caso delle mani, possono essere realizzate anche con modalità tridimensionali. Inoltre, sebbene questo tipo di rilevamento specifico permetta d'individuare più mani contemporaneamente e sebbene riesca a produrre buoni risultati anche con mani non interamente o non perfettamente visibili, vi sono problemi quando sono presenti molte mani vicine nella stessa scena considerata. Per maggiori informazioni in merito a questa specifica tipologia di rilevamento si consulti il seguente riferimento [61].
- Rilevamento (detection) di 70 punti di giuntura per il volto. Ancora una volta l'approccio è in modalità bidimensionale, real-time e multipersona.
- Rilevamento (detection) real-time e per persona singola, ma in modalità tridimensionale. Per fare ciò è possibile integrare OpenPose con telecamere di tipo flir (telecamere termiche) in maniera semplice, dal momento che il supporto di base è già presente in un toolbox targato OpenPose. Alternativamente, è possibile sincronizzare manualmente OpenPose con altre tipologie di telecamere al fine di ottenere una vista tridimensionale grazie alla triangolarizzazione di più viste singole. Per

fare ciò si sfruttano le nozioni di triangolarizzazione e geometria epipolare analizzate nella sottosezione 1.4.3.

All'interno del toolbox di OpenPose sono anche presenti dei supporti per telecamere IP, webcam e telecamere a scala di grigi. Non è presente il supporto per telecamere di profondità come quelle prese in considerazione nella sezione 3.1.

- Kit di strumenti per la calibrazione: all'interno di questo kit è presente uno strumento per la stima della distorsione, per la valutazione dei parametri intrinseci ed estrinseci della telecamera considerata. Nozioni in merito ai parametri intrinseci ed estrinseci della telecamera sono fornite nella sottosezione 3.1.1.
- Tracking per persona singola con successiva opportuna preparazione dei dati di output per eventuali future elaborazioni.
- È presente una versione *lightweight* di OpenPose, che permette così l'esecuzione anche su hardware di fascia media a bassa, dispositivi *embedded* e al tempo stesso garantire prestazioni *real-time* o prossime al *real-time*.

L'input richiesto da OpenPose consiste di una o più immagini, video, flussi immagine o video acquisiti da telecamere di vario tipo, come webcam, telecamere IP, di profondità, termiche, a scala di grigi e così via. L'output fornito di default da OpenPose in caso di richiesta di rilevamento di skeleton completi, consiste di una o più immagini mostrate a video e sulle quali viene *disegnato* ogni skeleton individuato. È possibile salvare le singole immagini mostrate (PNG, JPG, AVI) e richiedere, tramite opportuni parametri d'impostazione di OpenPose, che le informazioni relative ai punti di giuntura di ogni skeleton individuato e per ogni immagine mostrata, siano salvate in un apposito file in formato JSON, XML, YML, e altri. Così facendo, un applicativo ulteriore può leggere da tale file ed eseguire le elaborazioni di sua competenza. Infine, OpenPose fornisce un API con funzioni invocabili di alto livello sia per il linguaggio C++, che Python.

OpenPose è basato sulla *convolutional pose machine* introdotta nella sezione 3.2. Il suo funzionamento completo è descritto ai seguenti riferimenti [52, 61]. Di conseguenza, in maniera sequenziale all'interno della rete, si migliora l'individuazione dello skeleton per poi fornire il risultato completo e finale. La *pose estimation* che permette di realizzare OpenPose, è molto utile in problemi di *action recognition* considerati nella sezione 2.2. I collegamenti

fondamentali tra le giuntura da individuare a tal proposito, sono quelli degli arti delle persone presenti all'interno della scena considerata.

OpenPose funziona nel seguente modo. Per semplificare, si consideri un'immagine con un solo individuo umano all'interno e ben visibile nella sua completezza. Inizialmente si estraggono le *feature* dall'immagine ritenute fondamentali da OpenPose, sfruttando i primi livelli della rete neurale convoluzionale (CNN). Le *feature* estratte sono poi date in input a due categorie di livelli della CNN: la prima categoria (o gruppo), si occuperà di predire un insieme di 18 *confidence map*, dove ognuna di queste mappe denota una specifica parte dello skeleton umano. L'altra categoria (o gruppo), predice un insieme di 38 *part affinity field* (P.A.F.), che invece denotano il livello di associazioni tra le differenti parti individuate dello skeleton. La figura 3.9 mostra una rappresentazione grafica dei *part affinity field*.

I passaggi successivi sono utilizzati per ripulire le predizioni fatte dai passaggi precedenti. Con l'aiuto delle *confidence map*, vengono creati dei grafi bipartiti tra le coppie di parti (anche dette collegamenti) realizzate. Attraverso i valori dati dai *part affinity field*, i collegamenti deboli vengono eliminati dal grafo bipartito. A questo punto, lo skeleton può essere stimato in maniera corretta e abbinato a una determinata persona all'interno dell'immagine di input. Nel caso di un'immagine con più persone all'interno, OpenPose riesce a elaborare contemporaneamente tutti gli skeleton presenti, impiegando però maggiore tempo di esecuzione all'atto pratico.

Grazie ai *part affinity field*, è possibile capire l'orientazione e la posizione delle parti del corpo individuate. Un vettore 2D è assegnato a ogni pixel di un *part affinity field*. Usare i *part affinity field* è come assegnare un punteggio che viene man mano modificato (o raffinato) lungo gli strati della rete neurale convoluzionale, così da massimizzare la precisione. I *part affinity field* sono una rappresentazione che consiste di un insieme di flussi di campi vettoriali che codificano relazioni non strutturate tra coppie di parti del corpo di un numero variabile di persone.

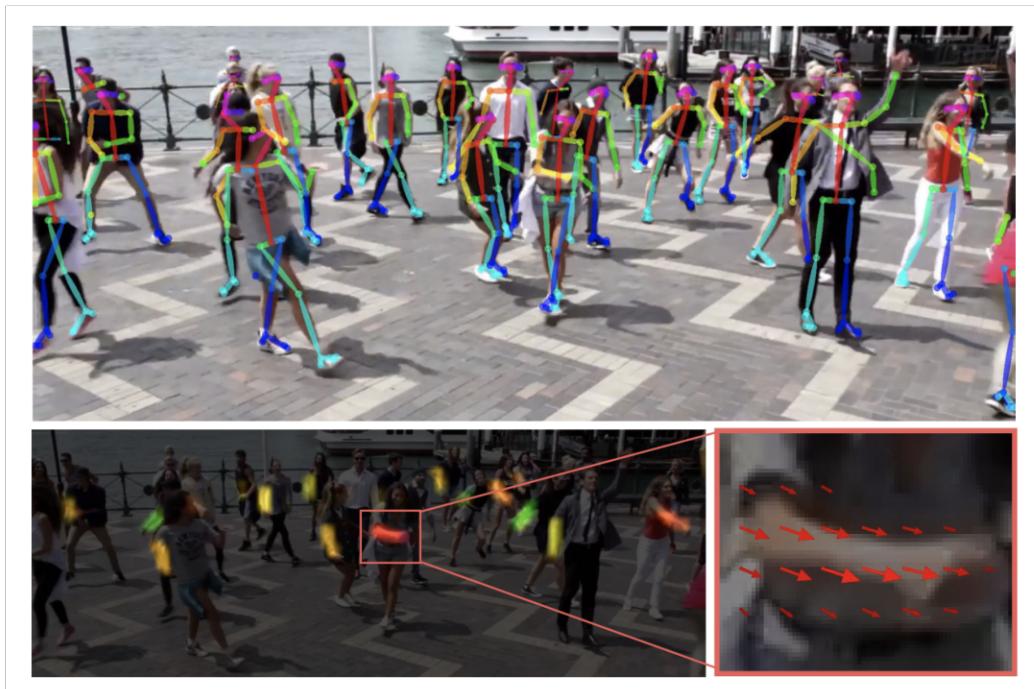


Figura 3.9. Immagine in alto: stima della posa multipersona. Le parti del corpo appartenenti alla stessa persona vengono connesse tra di loro. Immagine in basso a sinistra: *part affinity field* (PAFs) corrispondenti all'arto che collega il gomito destro al polso destro. Il colore codifica l'orientazione. Immagine in basso a destra: un vettore 2D codifica, per ogni pixel di ogni PAF, la posizione e l'orientazione dell'arto [45].

In sostanza l'immagine I di dimensioni $w \times h$ viene data in input alla CNN. All'interno di quest'ultima, tramite l'utilizzo delle *confidence map* relative alle parti del corpo individuate e ai *part affinity field*, si effettua una predizione raffinata strato dopo strato, rendendo il risultato più accurato. Infine, si applica la tecnica del *bipartite matching*, per poi convertire i risultati in un output per l'utente. L'algoritmo usato da OpenPose per la parte finale che include l'utilizzo del grafo e del *bipartite matching* è un algoritmo di tipo *greedy*.

Durante l'attraversamento della CNN, si applica una *non maxima suppression* 1.4.9 andando a considerare così solo i punti di giuntura più plausibili e le parti che, con maggiore probabilità (maggiore grado di certezza), sono effettivamente collegate. Per unire i punti di giuntura che vengono individuati inizialmente, si sfruttano ancora una volta i *part affinity field*, poiché questi conservano informazioni sia sulla posizione, sia sull'orientamento. Infatti, per ogni pixel nell'area appartenente a una particolare parte del corpo, un vettore 2D codifica la direzione che punta da una parte del corpo all'altra. I *part affinity field* vengono impiegati anche come fattore di ridondanza: cioè si usano dei *part affinity field* con connessioni ridondanti al fine di migliorare la precisione in immagini con un elevato numero di persone. L'algoritmo ordina tutte le possibili coppie che formano una determinata parte del corpo di una persona in base al punteggio del *part affinity field* associato. Se si cerca di effettuare una connessione che connette due punti di giuntura che erano stati già assegnati a una persona differente in quanto parte di una *body part*, si ricava che eseguire nuovamente il collegamento porterebbe a una contraddizione e la proposta di connessione tra i punti di giuntura viene ignorata. A questo punto si passa alla proposta di connessione successiva, caratterizzata dall'avere un punteggio associato al PAF inferiore o uguale rispetto alla proposta di collegamento precedente.

La figura 3.10 riassume brevemente e graficamente il funzionamento di OpenPose.

Nel caso del *multi-person skeleton detection*, utilizzare la NMS aiuta, ma non è sufficiente a contrastare errori, falsi positivi e falsi negativi. Il problema di trovare la conversione ottimale corrispondente a un matching k -dimensionale è un problema NP-hard. Si sfrutta quindi un'approssimazione che permette di risolvere in maniera sub-ottima il problema mediante i grafi bipartiti e il relativo *matching*, oltre a ulteriori semplificazioni. Il matching in un grafo bipartito è dato da un insieme di archi, scelti in maniera tale per cui non vi siano due archi che condividono uno stesso vertice.

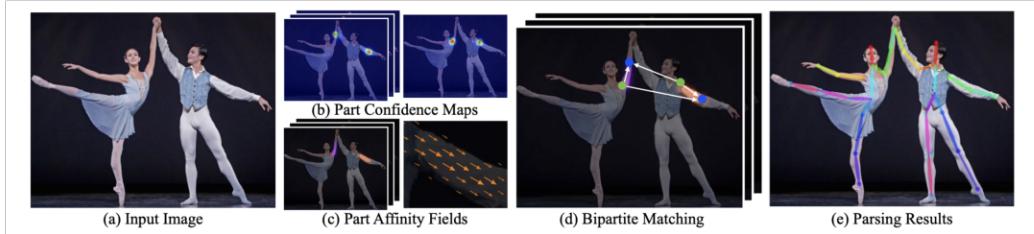


Figura 3.10. Schematizzazione del funzionamento di OpenPose: (a) l’intera immagine viene data in input alla rete convoluzionale neurale, al fine di predire le (b) *confidence map* per il rilevamento delle parti del corpo e i (c) *part affinity field* per l’associazione tra le parti del corpo individuate. In (d) si mostra come il passaggio di conversione esegua un insieme di *bipartite matching* al fine di associare le parti del corpo candidate con maggiore probabilità di connessione. In (e) si mostra invece come tutte le informazioni prodotte vengono assemblate in unico output finale [45].

Si precisa che OpenPose risolve il problema della costruzione dello skeleton in maniera *bottom-up*, ed è l’approccio più noto. Questo approccio prevede di rilevare inizialmente i punti di giuntura e associarli a un individuo specifico all’interno dell’immagine. Un approccio alternativo è quello *top-down*, adottato da *Alpha Pose* [98]. In un approccio *top-down*, si ha maggiore accuratezza nel trovare le parti appartenenti effettivamente a una persona specifica. Infatti, questi tipi di approcci si basano su aree fissate all’interno dell’immagine dove viene avviata la *pose estimation*. Di contro però, un approccio *top-down* porta a errori nella localizzazione di parti più o meno distanti dal resto del corpo. Questo fa sì che l’algoritmo, in diverse occasioni, non riesca a produrre degli output ottimali. In un approccio *top-down* l’idea è di adoperare un *person detector* e compiere la stima della posa delle singole persone per ogni persona rilevata. Se però il *person detector* fallisce (e.g. fotografia con molte persone ravvicinate al suo interno), non c’è modo di riprendersi da tale situazione. AlphaPose, d’altra parte, riesce a correggere e sopperire ad alcune di queste e altre problematiche (non citate in questo documento) degli approcci *top-down*, in maniera più o meno completa.

Vale la pena menzionare che, in alcuni approcci *bottom-up*, si sfruttano reti neurali convoluzionali di tipo *region-based*.

Il problema del determinare la posizione delle componenti anatomiche del corpo umano in un'immagine o in un video presenta alcune problematiche fondamentali riportate di seguito.

- Ogni immagine analizzata può contenere un numero non noto a priori di persone, che possono comparire in qualsiasi posizione dell'immagine e in qualsiasi fattore di scala (dimensione).
- Le interazioni tra persone nelle immagini inducono a complesse interferenze spaziali dovute a contatti, occlusioni e sovrapposizioni. Tali aspetti rendono più complessa e articolata la determinazione dei punti di giuntura del corpo e, di conseguenza, anche dei collegamenti fra di essi.
- I tempi di esecuzione tendono a crescere con l'aumentare del numero di persone presenti nell'immagine. Ciò rende l'obiettivo del real-time una vera sfida. Per far fronte a questa problematica, è possibile realizzare ed eseguire OpenPose in parallelo su GPU avanzate. Così facendo, i tempi di esecuzione sono di gran lunga abbreviati.

Si consideri il seguente scenario. Si supponga di valutare immagini o video in input, al cui interno vi sono *pose* atipiche delle persone (e.g. persone capovolte o con occlusioni importanti), oppure sono presenti molte persone o animali e statue dalle sembianze umane. In tali circostante OpenPose ha diverse difficoltà, portando a *falsi positivi* e *falsi negativi*. Al fine di risolvere questo problema, è possibile addestrare nuovamente i modelli di OpenPose con esempi al riguardo, così da avere maggiore precisione su una più vasta gamma di casistiche.

Infine, vale la pena menzionare il fatto che OpenPose si avvale di un approccio che potenzialmente non è limitato solo alla stima della posa del corpo umano, dei piedi, delle mani e del volto. Infatti, con le opportune modifiche al/ai modelli usati, è possibile generalizzarlo per task simili, ma diversi. Per esempio, è possibile generalizzare OpenPose per individuare le giunture e i relativi collegamenti per una determinata gamma o più di veicoli.

La figura 3.11 mostra delle esecuzioni di OpenPose in alcuni casi limite. Per maggiori informazioni in merito a OpenPose e al suo funzionamento si consulti il seguente riferimento [45].

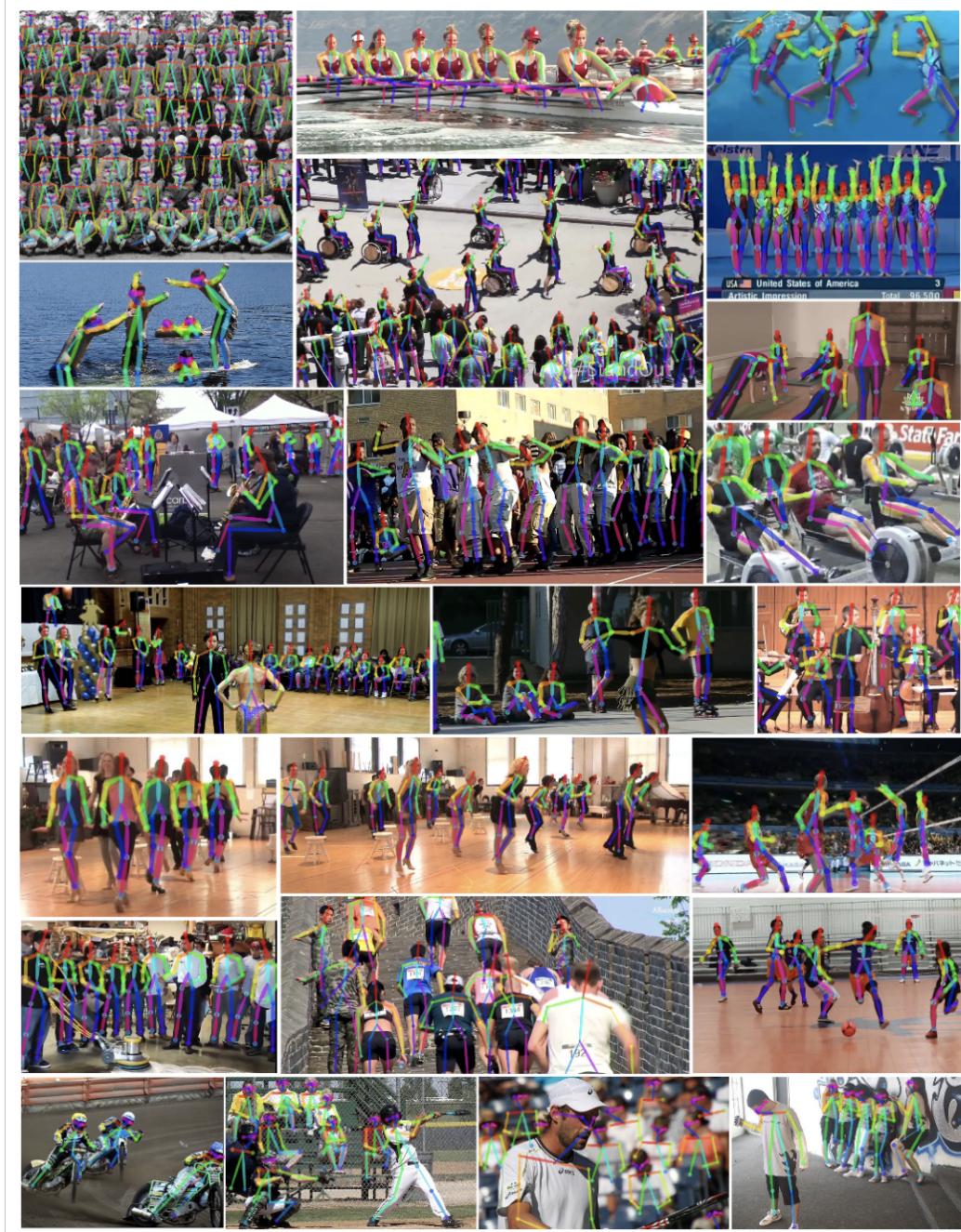


Figura 3.11. Risultati raggiunti da OpenPose con punti di vista differenti, casi limite, occlusioni, folle, contatti, sovrapposizioni e scenari simili [45].

3.3 Algoritmi per la trasformazione di SC

In base a quanto descritto nella sezione 2.3 e nella sottosezione 3.1.1, si propongono di seguito due brevi algoritmi (Algoritmo 1 e Algoritmo 2) per la trasformazione di coordinate di un punto 2D $P_A = (x_A, y_A)$ appartenente a un determinato rettangolo A in un punto 2D $P_B = (x_B, y_B)$ appartenente a un determinato rettangolo B , in modo che i due punti siano spazialmente analoghi tra di loro, rispetto al proprio rettangolo di appartenenza.

Si precisa che la trasformazione fra sistemi di coordinate analizzata qui di seguito, è solo una delle due trasformazioni (o alternativamente operazioni di *mapping*) di coordinate che avviene in AI Watch A1. Infatti, in questo caso si tratta del passaggio dal sistema di coordinate tridimensionale Intel RealSense a un sistema di coordinate tridimensionale opportuno, ovvero quello utilizzato nei moduli successivi per la rappresentazione dell'ambiente virtuale. La prima trasformazione fra sistemi di coordinate è espressa dall'operazione di *deprojection* considerata nella sottosezione 3.1.1.

Si consideri l'algoritmo 1. Quello che si fa è esprimere l'ascissa del punto P_A in termini dello spazio del rettangolo A . Poi si calcola il rapporto fra l'ascissa così espressa e l'ampiezza di A . A questo punto, è necessario scalare il risultato per la base del rettangolo B , per ottenere la posizione relativa rispetto al rettangolo B . Infine, si trasla l'ascissa nel rettangolo B , ottenendo così un riferimento assoluto.

Di seguito si commentano brevemente i parametri e le variabili presenti in questo algoritmo:

- x_A è l'ascissa del punto in A da convertire nel nuovo intervallo in B ;
- X_A è la più piccola ascissa che fa parte del rettangolo A ;
- w_A è l'ampiezza (o base) del rettangolo A ;
- X_B è la più piccola ascissa che fa parte del rettangolo B ;
- w_B è l'ampiezza (o base) del rettangolo B ;
- x_B è l'ascissa del punto in B ;

Algoritmo 1. Trasformazione ascissa di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale.

Input : x_A, X_A, w_A, X_B, w_B

Output: x_B

$$1 \ s \leftarrow \frac{(x_A - X_A)}{w_A};$$

$$2 \ t \leftarrow s \cdot w_B;$$

$$3 \ x_B \leftarrow t + X_B;$$

Algoritmo 2. Trasformazione ordinata di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale.

Input : y_A, Y_A, h_A, Y_B, h_B

Output: y_B

$$1 \ s \leftarrow \frac{(y_A - Y_A)}{h_A};$$

$$2 \ t \leftarrow s \cdot h_B;$$

$$3 \ y_B \leftarrow t + Y_B;$$

Si consideri l'algoritmo 2. Quello che si fa è esprimere l'ordinata del punto P_A in termini dello spazio del rettangolo A . Poi si calcola il rapporto fra l'ordinata così espressa e l'altezza di A . A questo punto, è necessario scalare il risultato per l'altezza del rettangolo B , per ottenere la posizione relativa rispetto al rettangolo B . Infine, si trasla l'ordinata nel rettangolo B , ottenendo così un riferimento assoluto.

Di seguito si commentano brevemente i parametri e le variabili presenti in questo algoritmo:

- y_A è l'ordinata del punto in A da convertire nel nuovo intervallo in B ;

- Y_A è la più piccola ordinata che fa parte del rettangolo A ;
- h_A è l'altezza del rettangolo A ;
- Y_B è la più piccola ordinata che fa parte del rettangolo B ;
- h_B è l'altezza del rettangolo B ;
- y_B è l'ordinata del punto in B ;

Siccome la tecnologia Intel RealSense, come già discusso nella sezione 2.3, fornisce la coordinata relativa all'ordinata in un *formato* semplificato rispetto a quello dell'ascissa, è possibile usare la seguente trasformazione semplificata (Algoritmo 3).

Di seguito si commentano brevemente i parametri e le variabili presenti in questo algoritmo:

- y_A è l'ordinata del punto in A da convertire nel nuovo intervallo in B ;
- Y_A è la più piccola ordinata che fa parte del rettangolo A ;
- h_A è l'altezza del rettangolo A ;
- o è un particolare offset relativo all'altezza che, nel caso del *CVPR Lab*, è risultato essere sperimentalmente pari a 0.15 metri;
- y_B è l'ordinata del punto in B ;

Algoritmo 3. Trasformazione ordinata semplificata di un punto P_A in A in un punto P_B in B con conservazione di analogia spaziale (valido per formato ordinata Intel RealSense D435).

Input : y_A, Y_A, o

Output: y_B

1 $s \leftarrow Y_A + h_A - y_A$;

2 $y_B \leftarrow s + o$;

3.4 Apache Kafka

Sebbene ci siano grandi potenzialità anche nelle alternative ad Apache Kafka, com'è possibile notare nella sezione 2.4; quest'ultimo resta indiscutibilmente la scelta migliore in diversi ambiti. Di conseguenza, di seguito viene analizzato maggiormente nel dettaglio il funzionamento di Apache Kafka. Apache Kafka è la risposta al problema della realizzazione di un sistema di messaggistica scalabile e distribuito. Si supponga di voler realizzare un sistema che si mette in ascolto degli aggiornamenti di una partita di calcio da varie fonti (e.g. punteggio attuale, tempo, marcatori, e così via). È necessario poi mostrare queste informazioni su tipologie di dispositivi differenti (PC, telefono, tablet). Nell'architettura che si sta supponendo, si ha a disposizione un processo che si occupa di leggere gli aggiornamenti e di scriverli in una coda. Questo processo è chiamato *produttore*, perché *produce* gli aggiornamenti che poi inserisce nella coda. Un produttore Kafka può essere implementato in diversi modi, ma per sua natura risulta essere asincrono. Dall'altro lato della coda, l'operazione è fatta da processi detti *consumatori*, perché *consumano* questi aggiornamenti. Questi processi estraggono le informazioni dalla coda e le mostrano sul canale di riferimento (PC, telefono, tablet). Kafka è da intendersi semplicemente come un meccanismo di trasporto d'informazioni.

Ora si supponga che, col tempo, il sistema realizzato non segua più una sola partita di calcio, ma più partite di calcio. Il problema è che i server non riuscirebbero a gestire un tale carico. Questo accade principalmente perché la coda è ospitata su un solo server e se si pongono più code su un solo server, automaticamente ciò comporta che tutte le code siano più piccole in termini di capacità e di conseguenza cala il numero di aggiornamenti che sono in grado di memorizzare, mentre aumenta la probabilità di avere errori di memoria. L'idea iniziale è quella di aggiungere potenza computazionale e distribuire l'architettura. Com'è possibile distribuire una struttura dati come la coda? Quest'ultima, con i suoi elementi, segue un ordine specifico. Si potrebbe provare a distribuire in maniera casuale il contenuto della coda in più code. In tal modo, però, i consumatori potrebbero trovarsi a consumare gli *eventi* contenuti nelle code in un ordine non corretto, ottenendo delle inconsistenze (e.g. punteggi differenti su dispositivi differenti).

Un approccio è quello di permettere all'applicazione di specificare come ripartire i dati all'interno delle code, magari *assegnando* un identificativo a ogni aggiornamento (o evento). Così facendo, aggiornamenti relativi a uno stesso match saranno ordinati e si troveranno anche all'interno della stessa coda. Questa è l'idea di base di Apache Kafka. Invece, per inviare e ricevere

è necessario stabilire una strategia di distribuzione dei dati. Un approccio avanzato a tal proposito è usare una funzione di hash sull'identificativo dell'evento generato.

In Kafka, ogni coda è detta *partizione*. Il numero totale delle partizioni è anche detto *contatore delle partizioni*. Ogni server mantiene una o più partizioni. Tale server è detto *broker*. Ogni elemento di una partizione è detto *record*. L'identificativo che stabilisce quali eventi vanno in una determinata partizione è detto *partition key*. È compito dell'applicativo stabilire quale valore usare come partition key. Se non ne viene specificato uno, allora Kafka sfrutta identificativi pseudo-casuali.

Un gruppo di partizioni che gestiscono lo stesso tipo di dati è detto *topic*. Un topic è possibile considerarlo come una sorta di cartella in un filesystem e gli eventi sono i file all'interno di questa cartella. Al fine d'identificare ogni record univocamente, Kafka fornisce un numero sequenziale identificativo per ogni record. Tale valore è detto *offset*. Quindi, un record in un topic è identificato univocamente dal *partition key* e dall'*offset*. Nell'applicativo finale, dal momento che sono stati distribuiti i dati nel topic usando un particolare valore come *partition key*, è possibile parallelizzare i consumatori. Avendo un solo consumatore per ogni partizione, si riescono a leggere gli aggiornamenti in maniera corretta e ordinata. In figura 3.13 si mostra un tipico caso d'uso di Kafka. In figura 3.13 e 3.14, vengono illustrati i concetti detti finora.

L'idea classica per la memorizzazione dei dati è quella di usare un database e modellare secondo entità e relazioni il contesto analizzato. Questo approccio si concentra sulla modellazione di *cose*. Un altro approccio è quello di modellare gli stati. Siccome un evento è caratterizzato principalmente da un *timestamp*, non è semplice memorizzarlo in un database. Quello che si fa è usare un *log di dati*. Apache Kafka è un sistema per gestire questi log che vengono chiamati topic.

È possibile avere anche più consumatori per ogni partizione. Un applicativo consumatore è molto leggero, quindi è possibile crearne diversi senza andare a incidere sulle prestazioni. Infatti, Kafka ha soltanto bisogno di ricordare l'ultimo offset letto da ogni consumatore in ascolto su quella particolare partizione. Con Kafka si lascia scegliere al consumatore come *leggere* i record: tipicamente si sfrutta una lettura FIFO (First In First Out). Per raggruppare i consumatori che leggono da partizioni differenti, ma che sono consumatori dello stesso tipo (e.g. tutti telefoni) si parla di *consumer group*.

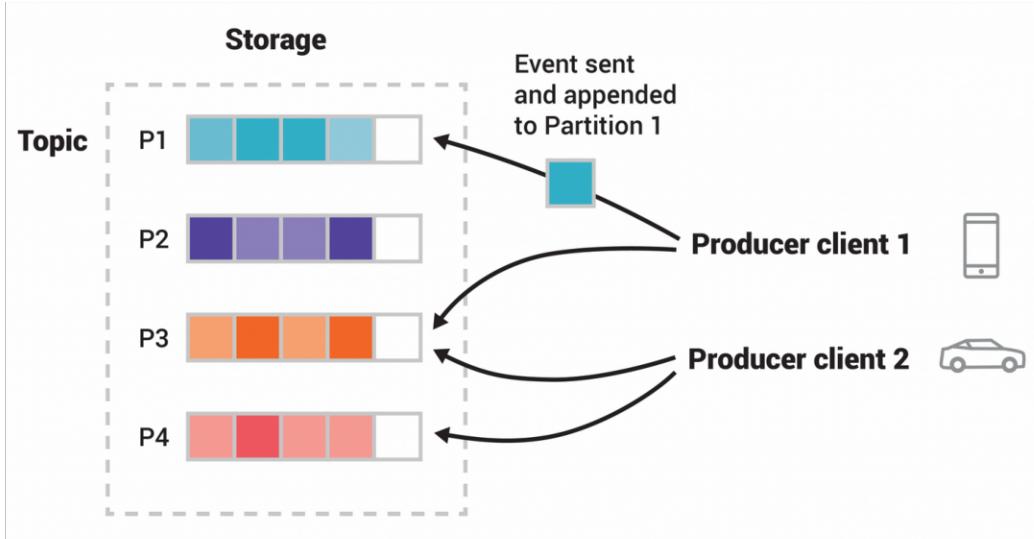


Figura 3.12. Un esempio di utilizzo di Apache Kafka con 4 partizioni P1-P4. Due diversi produttori pubblicano, indipendentemente l’uno dall’altro, nuovi eventi nel topic scrivendoli nelle opportune partizioni del topic. Gli eventi con la stessa chiave di partizione (indicata dal loro colore in figura), sono scritti nella stessa partizione. Si noti che entrambi i produttori possono scrivere nella stessa partizione, se appropriato [74].

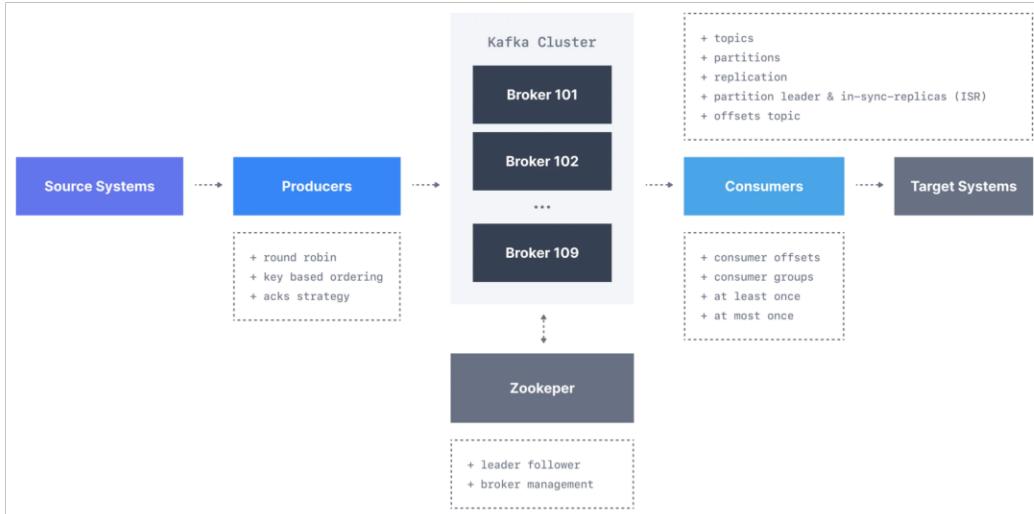


Figura 3.13. Componenti di base in Kafka [105].

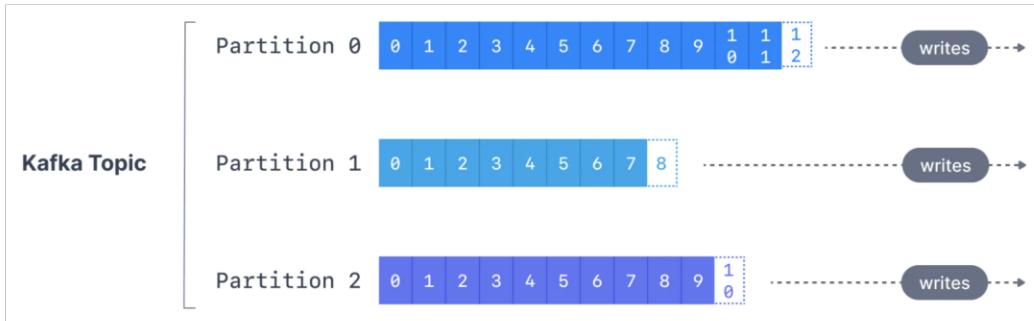


Figura 3.14. Topic e partizioni in Kafka [105].

identifier. Ogni consumatore del gruppo avrà *offset pointer* separati per tenere traccia dell'ultimo record letto in quella particolare partizione.

In figura 3.15, è mostrata la tipica struttura di un messaggio inviato da un produttore mediante Apache Kafka.

Si supponga che un applicativo consumatore scelga di leggere i record di una partizione usando un ordinamento personalizzato e non di tipo FIFO. Come fa Kafka a determinare che il record è stato già consumato e può quindi cancellarlo in maniera sicura? Il problema di questo approccio è l'accumulo di record non letti. Per risolvere questo problema è possibile far usare a Kafka politiche differenti (e.g. cancellare tutti i record che non sono stati consumati entro 24 ore. Di contro, ciò potrebbe portare alla perdita di aggiornamenti se l'applicativo smette di funzionare per più di 24 ore). Quindi, gli eventi una volta letti dal topic non vengono cancellati. Inoltre, più produttori possono produrre nella stessa partizione del topic. Alternativamente, gli eventi vengono cancellati a partire dal più vecchio, per esempio quando la coda è piena o si supera una certa soglia (tecniche di log cleaning).

Un'altra capacità di Kafka è quella di memorizzare i record in maniera tale che sia resistente ai guasti e duraturo. Ogni record è infatti memorizzato su un dispositivo di memorizzazione permanente, così che se un broker smette di funzionare, si può ripristinare lo stato a prima che smettesse di funzionare. Inoltre, Kafka replica le partizioni, così che se un broker smette di funzionare, una partizione di backup da un altro broker prende il posto da sostituto. Questo è possibile farlo grazie alla replicazione e alla ridondanza dei dati. Il livello di ridondanza è detto *scaling factor* (e.g. 3 indica una partizione *leader* e due partizioni di backup. In tal caso il sistema tollera che al più due broker smettano di funzionare contemporaneamente).

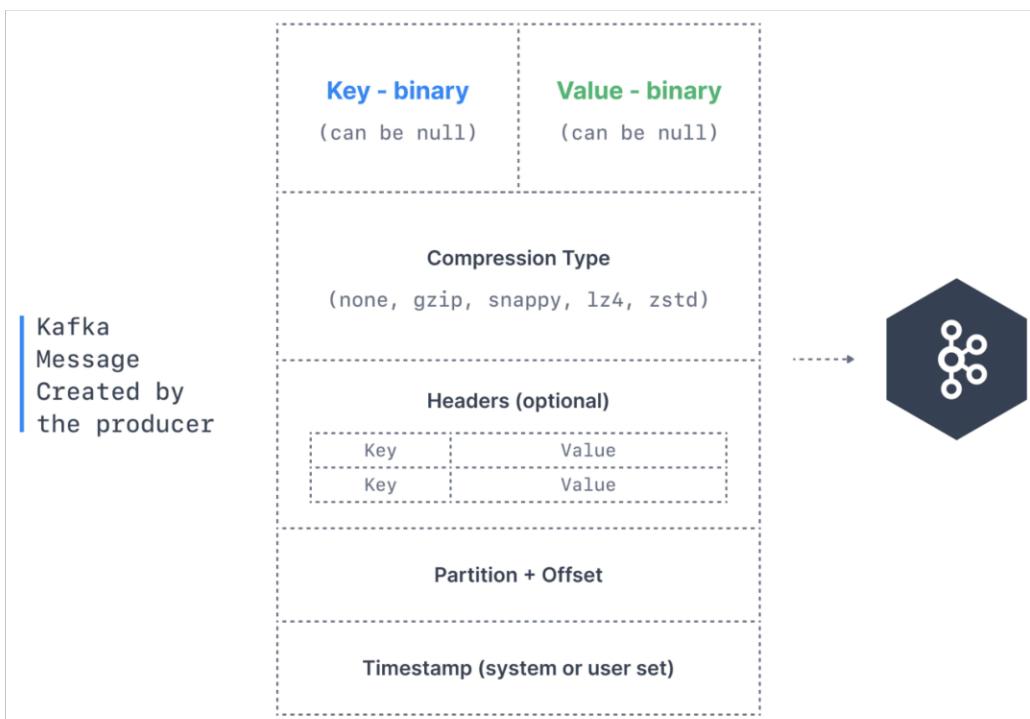


Figura 3.15. Struttura di un messaggio inviato da un produttore in Kafka [105].

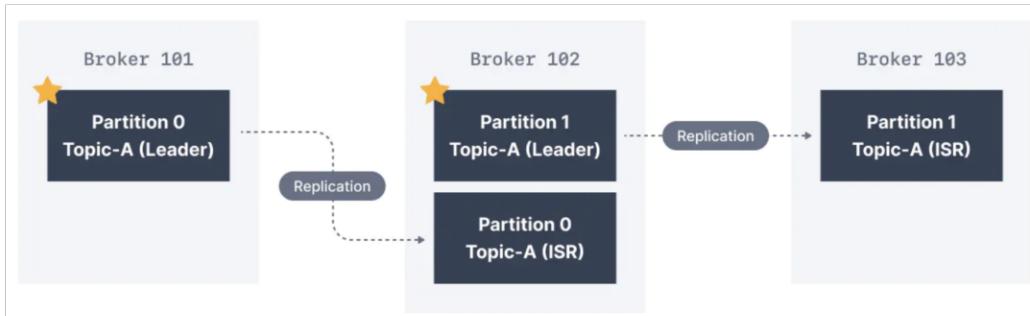


Figura 3.16. Replicazione delle partizioni in Kafka [105].

Un'illustrazione in merito alla replicazione delle partizioni, è visibile in figura 3.16.

Al giorno d'oggi una delle pratiche più adoperate è quella di realizzare architetture software composte da moduli brevi, compatti e completi, cioè da tante piccole entità software e non più da un unico grande programma che interagisce con il database). Questi tanti e piccoli programmi cambiano e si evolvono nel tempo in maniera veloce e indipendente dagli altri. Inoltre, hanno solitamente la necessità di comunicare tra di loro, al fine di produrre comportamenti e funzionalità più avanzate. Per adempiere ciò, è possibile realizzare dei topic in Kafka e assegnare loro dei ruoli da produttore e/o da consumatore in maniera opportuna. Così facendo questo tipo di architettura, detta *architettura orientata ai microservizi*, risulta essere facilmente scalabile.

Inoltre, si precisa che è possibile integrare database esistenti con un'architettura basata su Kafka, grazie a un tool chiamato *Kafka Connect* che permette quindi l'integrazione anche dei cosiddetti sistemi *legacy*. Invece, grazie a un altro tool chiamato *Kafka Streams*, è possibile anche collegare dati fra più topic emulando in qualche modo l'operazione di *join* di un database e ottenendo un risultato *simile*. Chiaramente, Kafka non è da intendersi come un database, ma come un mezzo per collegare sistemi, servizi e microservizi fra di loro, in maniera efficiente ed efficace. Inoltre, Kafka non sostituisce un database, in quanto non è in grado di consentire interrogazioni tanto complesse come in un database, non garantisce le proprietà ACID e non ha lo stesso supporto per il rollback in caso di guasti. Dal punto di vista tecnico, Apache Kafka fa uso di un particolare protocollo di rete basato su TCP ad alte prestazioni, che permette di far comunicare produttori e consumatori.

Kafka può essere gestito manualmente oppure in maniera semiautomatica o automatica, con sistemi come Confluent [106] o Conduktor [105]. Confluent non fa altro che facilitare la gestione nella fase d'inizializzazione e durante la fase di esecuzione di Kafka. Il *deployment* dell'architettura integrata Kafka può avvenire su hardware dedicato, macchine virtuali, contenitori appositi oppure via cloud. Per maggiori dettagli relativi ad Apache Kafka, consultare il seguente riferimento [74].

3.5 Aspetti implementativi

Il linguaggio di programmazione scelto è stato C++ [107]. Alternativamente, era possibile scegliere di realizzare il modulo A1 in Python. La scelta finale è orientata al C++, dal momento che il supporto di alcune librerie, che poi verranno prese in considerazione successivamente, è nettamente migliore per il linguaggio C++, rispetto al Python. Inoltre, sebbene C++ presenti una curva di apprendimento più ripida e risulti più complesso in alcuni aspetti o gestioni; è più veloce in termini di tempo di esecuzione rispetto al codice sorgente corrispettivo in Python.

Per quanto riguarda le librerie utilizzate, vengono elencate qui di seguito con i corrispettivi collegamenti sul web.

- Intel RealSense [108].
- OpenPose [72].
- OpenCV [109].
- Apache Kafka [74].
- Kafka for C/C++ [110].
- Confluent [106].
- JSON CPP [111].

Sono messi a disposizione sia un file *CMake* [112] che un file *Make* [113], che possono essere impiegati come mostrato nel manuale utente online [99]. Per quanto riguarda la documentazione del codice, è stata interamente realizzata con l'ausilio di Doxygen [114]. Tale documentazione è reperibile online al riferimento che segue [115].

È importante precisare la particolare struttura data alle cartelle e ai file del modulo AI Watch A1. Supponendo di non considerare altro che la cartella contenente i sorgenti, l'eseguibile, i file di configurazione e le sottocartelle contenenti i file prodotti dall'applicativo; la struttura ad albero è riportata in figura 3.17.

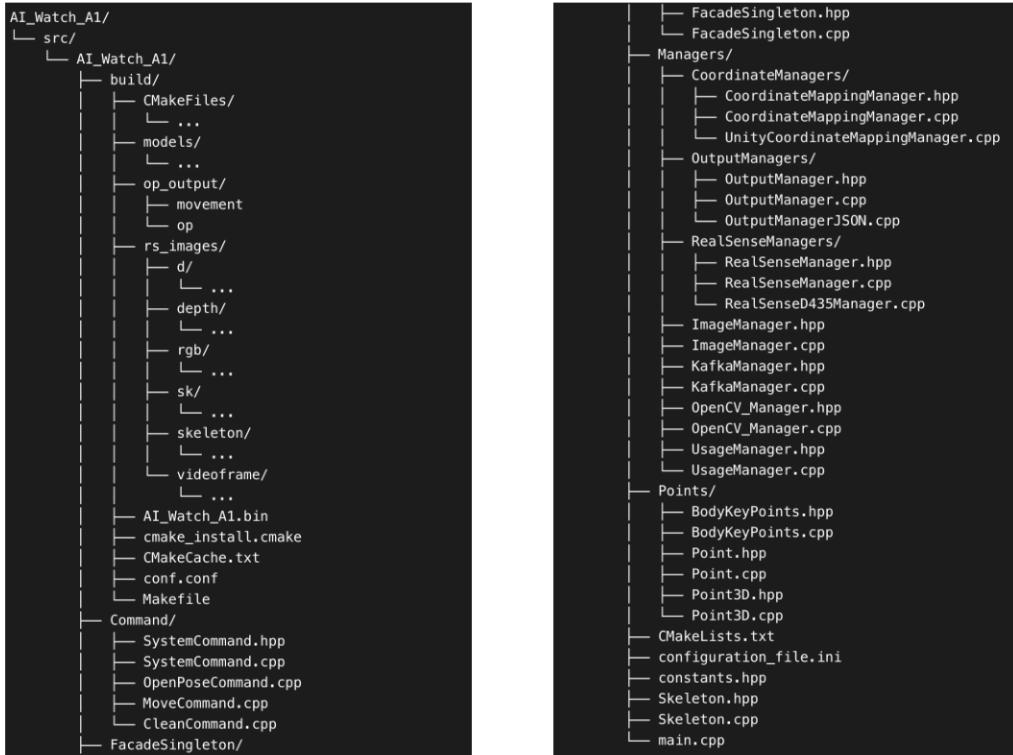


Figura 3.17. Struttura ad albero di file e cartelle AI Watch A1.

In ultima analisi, si prendono in considerazione quali sono i file di configurazione, cosa contengono e qual è il significato di particolari cartelle nella struttura ad albero poc'anzi illustrata. Innanzitutto, il file

`configuration_file.ini`

contiene i parametri di configurazione relativi ad Apache Kafka, tra cui l'indirizzo e la porta con cui mettersi in collegamento col *broker*, il *group ID* e la modalità in base alla quale leggere i messaggi. Altri parametri di configurazione si trovano all'interno del file seguente.

`build/conf.conf`

In particolare, sulla prima riga vi è il nome dell'eseguibile finale; sulla seconda riga la cartella all'interno del calcolatore ove è installato OpenPose; sulla terza riga il comando d'avvio di OpenPose con percorso completo; sulla quarta riga il percorso completo dove si trova la cartella

`rs_images`

del modulo A1; sulla quinta riga il percorso completo dove si trova la cartella `op_output`

del modulo A1. Ulteriori parametri di configurazione, che però si consiglia altamente di non variare, sono contenuti nelle ultime righe del file seguente.

`constants.hpp`

Infine, ulteriori parametri di configurazione relativi alla scena, all'interno della quale si posiziona la telecamera, sono quelli presenti nel file seguente.

`Room/Room.hpp`

I valori importanti da impostare sono i seguenti:

- `minWidth`: indica il valore dell'origine relativa alla coordinata *x*;
- `maxWidth`: indica la larghezza massima della scena inquadrata rispetto alla coordinata *x*;
- `minHeight`: indica il valore dell'origine relativa alla coordinata *y*;
- `maxHeight`: indica l'altezza massima della scena inquadrata rispetto alla coordinata *y*;
- `minWidthRS`: valore da dedurre sperimentalmente e che indica il minimo valore individuato dalla telecamera Intel RealSense per la coordinata *x*;
- `maxWidthRS`: valore da dedurre sperimentalmente e che indica il massimo valore individuato dalla telecamera Intel RealSense per la coordinata *x*;
- `minHeightRS`: valore da dedurre sperimentalmente e che indica il minimo valore individuato dalla telecamera Intel RealSense per la coordinata *y*;
- `maxHeightRS`: valore da dedurre sperimentalmente e che indica il massimo valore individuato dalla telecamera Intel RealSense per la coordinata *y*;
- `xOriginUnity`: valore *x* d'origine del sistema di coordinate in Unity;
- `zOriginUnity`: valore *z* d'origine del sistema di coordinate in Unity;

- `distanceCameraFromBackWall`: valore di distanza dal centro della telecamera fino alla parete retrostante alla telecamera stessa, supponendo che quest'ultima sia disposta in maniera perpendicolare rispetto a tale parete;
- `heightOffset`: valore dedotto sperimentalmente per migliorare la precisione sul valore relativo all'altezza, ovvero relativo alla coordinata y .

Nel caso del *CVPR Lab* i valori per tali parametri, in base a delle misurazioni realizzate in maniera accurata, sono risultati essere i seguenti:

- `minWidth`: 0.0;
- `maxWidth`: 6.03;
- `minHeight`: 0.0;
- `maxHeight`: 2.89;
- `minWidthRS`: -3.5;
- `maxWidthRS`: 3.5;
- `minHeightRS`: -1.0;
- `maxHeightRS`: 1.0;
- `xOriginUnity`: -26.89;
- `zOriginUnity`: -3.842;
- `distanceCameraFromBackWall`: 0.45;
- `heightOffset`: 0.15.

Tuttavia, siccome il modello Unity è stato realizzato con valori leggermente differenti da questi prima che il modulo AI Watch A1 fosse progettato e realizzato, e al fine di garantire la massima compatibilità possibile con gli altri moduli, i valori adottati a livello pratico che differiscono sono i seguenti:

- `maxWidth`: 5.271;
- `maxHeight`: 3.0.

Si precisa che tutti questi valori dei parametri presenti nel file `Room/Room.hpp` sono da intendersi in metri.

3.6 Test e risultati

L'intento di questa sezione è quello di andare a discutere dei test che sono stati condotti e dei risultati ottenuti. Si precisa che in merito ai risultati ottenuti, sarà possibile solo valutare le prestazioni dal punto di vista quantitativo e non qualitativo, in quanto non si è effettuato il confronto con altre tecnologie alternative a quella di OpenPose o a quella d'Intel RealSense, e così via. Però, verranno proposte alcune informazioni per avere un'idea dei tempi e delle prestazioni di OpenPose rispetto ad altre tecnologie affini su uno stesso insieme di dati.

Innanzitutto, nei video riportati al seguente riferimento [71], è possibile notare visivamente l'accuratezza abbastanza elevata della ricostruzione degli skeleton sulle varie persone rilevate. Nel secondo video in particolare, è possibile notare anche la precisione della determinazione per i valori delle coordinate per tre punti di giuntura fondamentali per ogni skeleton individuato (estremità del naso, punto a sinistra del bacino e caviglia sinistra). Il primo valore corrisponde alla distanza dalla parete sinistra, il secondo all'altezza dal pavimento e il terzo alla profondità dalla parete dietro alla telecamera.

Il calcolatore sul quale sono stati condotti i test non è un calcolatore di fascia alta in quanto a *calcolo computazionale*. Le sue specifiche sono le seguenti.

- Sistema operativo: MacOS Monterey 12.6.
- Modello: MacBook Pro (16-inch 2019)
- Processore: Intel Core i9-9980HK, 8 core, 16 thread per core, frequenza base a 2.4 Ghz, frequenza massima a 5.0 Ghz.
- Cache CPU: 16 MB, Intel Smart Cache.
- Memoria: 32 GB 2667 Mhz DDR4.
- GPU: AMD Radeon Pro 5300M 4 GB (dedicata) + Intel UHD Graphics 630 1536 MB (integrazione).

Diversi studi e ricerche condotti nel campo della computer vision in merito a OpenPose [62, 63, 64, 65], mettono in luce sia la precisione che l'accuratezza di OpenPose in merito alla rilevazione degli skeleton di persone all'interno d'immagini e video, ma anche la necessità di un'elevata potenza di calcolo al fine di poter avvicinarsi o raggiungere quelle che sono delle prestazioni

real-time. Conclusioni simili sono tratte dai documenti originali relativi a OpenPose [52, 61, 45]. Di conseguenza, dal momento che i test non sono stati condotti su un calcolatore con grosse potenzialità di calcolo, i risultati che seguiranno vanno comunque rapportati a cosa accadrebbe se tali test fossero condotti su calcolatori più potenti.

La fase di acquisizione dei frame è molto rapida, grazie all'ausilio della tecnologia Intel RealSense sulla telecamera D435; quindi il tempo necessario per questa fase può essere trascurabile. Analogamente per la fase di comunicazione con il modulo successivo. Ciò che richiede molto tempo e molta potenza computazionale è la fase di attivazione di OpenPose, della sua rete neurale convoluzionale e della conseguente rilevazione degli skeleton. Al fine di ridurre la problematica del tempo di esecuzione elevato, è possibile adottare due approcci elencati di seguito.

- Ridurre le dimensioni della rete convoluzionale neurale tramite parametri di avvio OpenPose. Questa opzione non è consigliata in quanto porterebbe a una precisione molto più bassa in termini d'individuazione dei punti di giuntura degli skeleton e di conseguenza degli skeleton stessi.
- Ridurre il numero di frame acquisiti al secondo. La telecamera Intel RealSense D435 riesce a catturare fino a 30 frame al secondo. Però, dal momento che molti frame in successione tendono a contenere informazioni molto simili tra di loro, e dal momento che il terzo modulo che si interfaccia con Unity e che replica il comportamento mediante i digital twin, riesce a interpolare e ricostruire movimenti fluidi e lineari anche con frame non esattamente consecutivi; è possibile adottare l'approccio d'ignorare (o scartare) un certo numero di frame durante la fase di acquisizione. Questo è un approccio altamente consigliato, in quanto permette di migliorare le prestazioni.

Un valore dedotto sperimentalmente dal secondo approccio elencato poc'anzi, è 5. Ovvero, si scartano ben 5 frame ogni frame acquisito. Tale valore permette di avere dei frame che, in successione, risultano tutto sommato fluidi, e che contengono informazioni abbastanza distinte dal momento che non sono stati acquisiti in modo troppo ravvicinato.

Perché questo approccio è importante? Considerando che sul calcolatore menzionato in precedenza, al fine di elaborare 60 frame con OpenPose, sono necessari in media 50 secondi. Dal momento che potenzialmente potrebbero

essere acquisiti al più 30 frame al secondo, allora per elaborare solo 2 secondi di video, ci vorrebbero ben 50 secondi. Usando l'approccio menzionato poc'anzi, però, è possibile *distribuire* i frame lungo un periodo temporale più ampio, semplicemente ignorando i successivi 5 frame a ogni frame acquisito. Ciò significa, che con l'acquisizione massima di 30 frame al secondo, il risultato consiste nell'acquisire solo 5 di questi frame, ignorando tutti gli altri. Di conseguenza, in 50 secondi circa verranno elaborati circa 12 secondi di video (5 FPS). Chiaramente il risultato non è ancora ottimale, ma in ogni caso è possibile modificare il numero di frame da ignorare di volta in volta, tenendo conto che ciò inficia poi sulla fluidità dei movimenti. Inoltre, non sempre la telecamera Intel RealSense D435 è in grado di acquisire 30 frame al secondo. Da test sperimentali condotti mentre era collegata al calcolatore considerato precedentemente, acquisisce circa 15 frame al secondo. Di conseguenza, in tal caso si ottengono risultati migliori, riuscendo in 50 secondi circa a elaborare ben 24 secondi di video. In quest'ultimo caso, però, è come se venissero acquisiti meno frame per secondo (2-3.5 FPS), ma non così pochi da rendere i movimenti poco fluidi. Ovviamente, con la possibilità di adottare un calcolatore con più potenza di calcolo a disposizione per OpenPose, si stima che è possibile ridurre più del 90% le tempistiche complete, arrivando a elaborare 12 secondi di video in circa 3 secondi nel caso con 5 FPS, oppure 24 secondi di video nel caso con 2-3.5 FPS.

Per quanto riguarda l'occupazione in memoria, l'applicativo risulta occupare in media, durante l'esecuzione, circa 120 MB. Mentre, per quanto riguarda l'occupazione sul disco, questa risulta essere molto contenuta. Infatti, una volta inviate le informazioni al secondo modulo di AI Watch A1 mediante Apache Kafka; tutti i frame acquisiti, i file prodotti e quant'altro, vengono immediatamente cancellati, così da liberare spazio utile al salvataggio dei file del *burst* successivo di esecuzione.

Un aspetto interessante è che, mentre nella documentazione OpenPose si asserisce che all'aumentare del numero di persone, i tempi richiesti siano gli stessi; in realtà sperimentalmente si è notato che, eseguendo OpenPose su un certo numero di frame, tutti contenenti più persone, i tempi risultano essere maggiori rispetto allo stesso insieme di frame con una sola persona all'interno o nessuna.

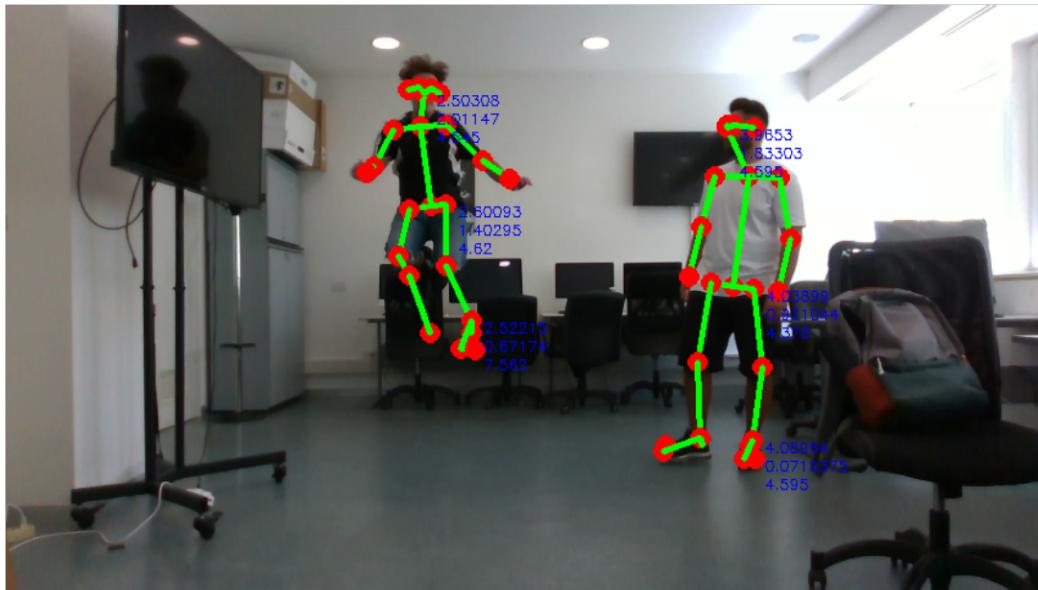


Figura 3.18. Immagine risultato skeleton frame 155 - Test 2.

Complessivamente, sono stati condotti tre test del singolo modulo. Dopo il primo test sono state apportate grandi modifiche al fine di migliorare la precisione e la stima delle coordinate; mentre dopo il secondo test si è migliorata soltanto la sua ingegnerizzazione. L'ultimo test conferma un funzionamento stabile del modulo. Un aspetto sicuramente da considerare perché importante, è la precisione dei risultati ottenuti. Sebbene, non si possano confrontare con altri risultati ottenuti da altre tecnologie, è possibile confrontare idealmente le misurazioni prodotte relative alle coordinante dei punti degli skeleton, con le dimensioni della persona nella realtà.

In figura 3.18, 3.19 e 3.20, si mostrano alcuni esempi che riportano tre valori in colonna, in tre punti differenti, per ogni skeleton individuato: distanza dalla parete ove giace la porta del laboratorio a sinistra, altezza rispetto al pavimento e distanza dalla parete retrostante alla telecamera Intel RealSense. Sono specificati tali valori per ogni skeleton e per tre punti di ognuno di esso a scopo esemplificativo. Tali punti sono la punta del naso dello skeleton, il punto sinistro del bacino e la caviglia sinistra. Quindi, nel caso delle immagini seguenti si tratta di avere le coordinante 3D dei punti all'interno della stanza. Si precisa che questo test è stato condotto in metri reali, così da dare un senso ed effettuare analisi valutative.

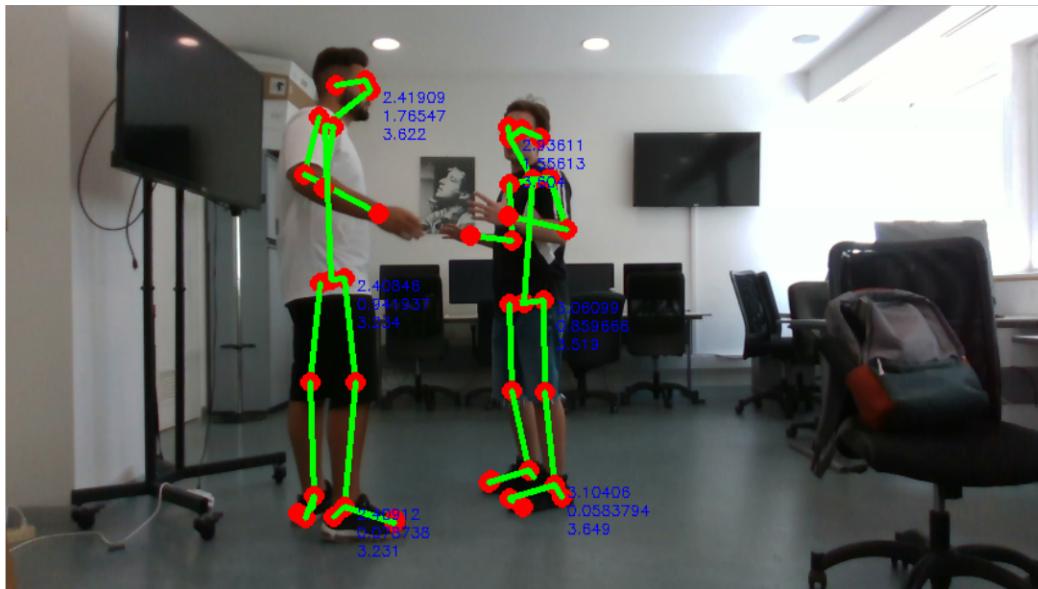


Figura 3.19. Immagine risultato skeleton frame 198 - Test 2.

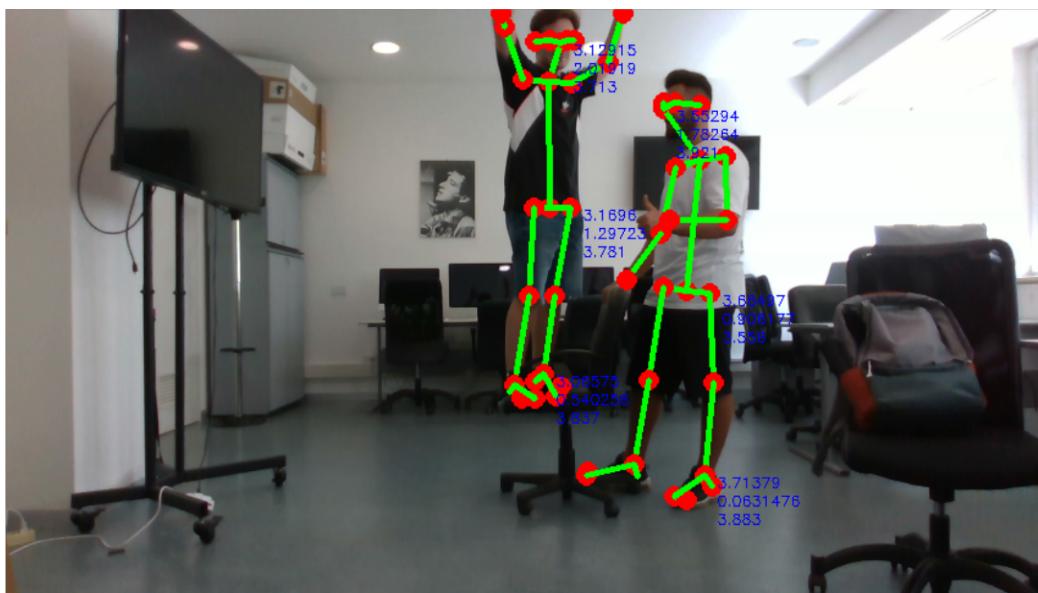


Figura 3.20. Immagine risultato skeleton frame 341 - Test 2.

Misura altezza reale	Renato Esposito	Denny Caruso
Punta del naso	1.74m	1.55m
Punto sinistro del bacino	1.00m	0.90m
Caviglia sinistra	0.10m	0.07m

Tabella 3.3. Misure reali di Renato Esposito e Denny Caruso

Misura altezza rilevata	Renato Esposito	Denny Caruso
Punta del naso	1.76-1.83 m	1.55 m
Punto sinistro del bacino	0.90-0.94 m	0.85 m
Caviglia sinistra	0.078 m	0.058 m

Tabella 3.4. Misure rilevate di Renato Esposito e Denny Caruso

In tabella 3.3, si riportano le misure reali dei tre punti menzionati per entrambi i soggetti. Tali misurazioni sono espresse in metri. Analogamente si fa lo stesso per le misure rilevate dal modulo AI Watch A1 in tabella 3.4

Si notino le differenze minime tra le misure rilevate e quelle effettive. Dal momento che si tratta di una misurazione approssimativa, dal momento che all'aumentare della distanza dalla telecamera, cala la precisione, dal momento che la stessa telecamera Intel RealSense ha una accuratezza elevata, ma comunque non perfetta al millimetro e dal momento che i frame catturati illustrano fasi di movimenti rapidi e veloci, si può stabilire che i risultati ottenuti sono ottimi. Deduzioni simili, è possibile trarrele considerando le misure reali della stanza e le coordinate assegnate ai punti riportati in figura. L'intervallo di variazione dell'errore medio commesso è di circa 10 cm. Ovviamente, quanto più precise sono le misurazioni della scena reale dove si installa la telecamera e quanto più preciso è il posizionamento della telecamera, quanto meno errore si accumula nelle fasi successive.

È importante precisare che, una volta ottenuti gli skeleton da OpenPose, il modulo AI Watch A1 effettua un ulteriore controllo per prevenire la visualizzazione di skeleton parziali, incompleti, inesistenti o poco accurati. Per fare ciò, si effettua una media del punteggio assegnato da OpenPose per ogni punto di ogni skeleton individuato e se il punteggio finale dello skeleton supera una certa soglia, allora viene considerato nelle fasi successive. Nonostante questo approccio, è possibile vedere degli errori in scene con figure di umani, fotografie, poster, statue e animali con sembianze umane. Ciò è causato dai modelli utilizzati nella fase di addestramento di OpenPose. La soluzione a tale problema è stata già discussa nella sezione 3.2.1. Un esempio

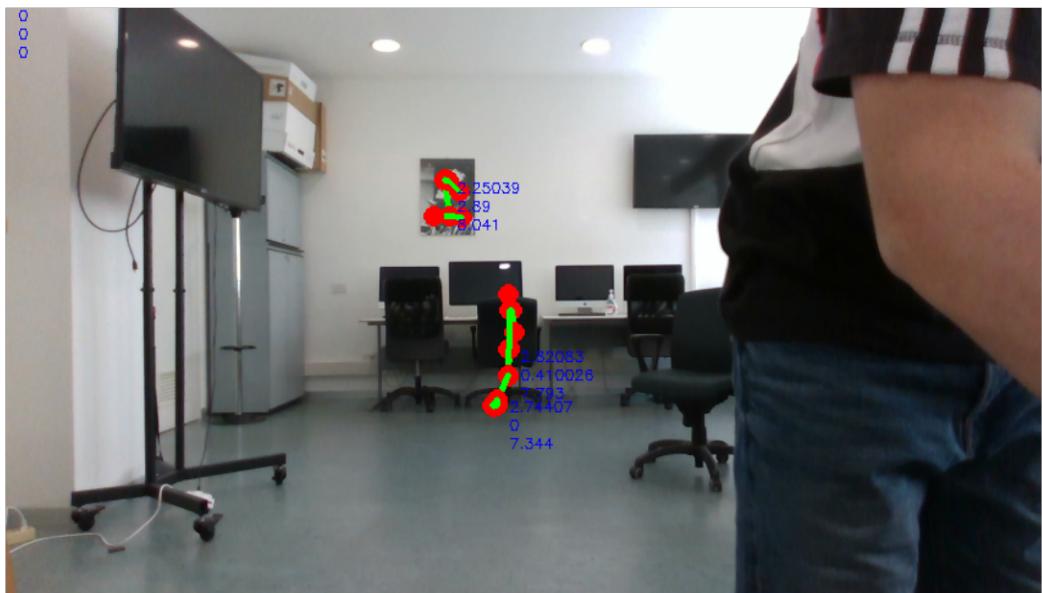


Figura 3.21. Immagine risultato skeleton frame 576 - Test 2.

di tale situazione è visibile in figura 3.21. Inoltre, si noti come nella stessa figura, non viene mostrato anche lo skeleton parziale che potrebbe contenere idealmente un gomito, parte del bacino e parte della gamba del candidato. Questo tipo di filtraggio, che avviene in questo modulo, è molto importante. Infatti, così facendo, è possibile evitare per i moduli successivi carichi di lavoro per skeleton con pochi o pochissimi punti, rappresentazioni mediante digital twin molto approssimate, incomplete o imprecise.

Invece, in figura 3.22 e 3.23 è mostrato il comportamento del modulo in presenza di persone di spalle o non completamente visibili. Si precisa che, in questo caso, il formato delle coordinate è quello che fa riferimento al mondo virtuale in Unity.

In figura 3.24a e 3.24b, si mostra come viene effettuata la scissione del frame RGB da quello relativo alla profondità, successivamente alla fase di acquisizione da parte della telecamera nel formato RGBD. In figura 3.24c, si mostrano gli skeleton estratti, dopo l'opportuna fase di rilevamento degli stessi. Invece, in figura 3.24d, si applica lo skeleton appena estratto sull'immagine 3.24a e si aggiunge l'informazione, per ogni punto di giuntura individuato, relativa alle coordinate spaziali facenti riferimento alla stanza ove è installata la telecamera.

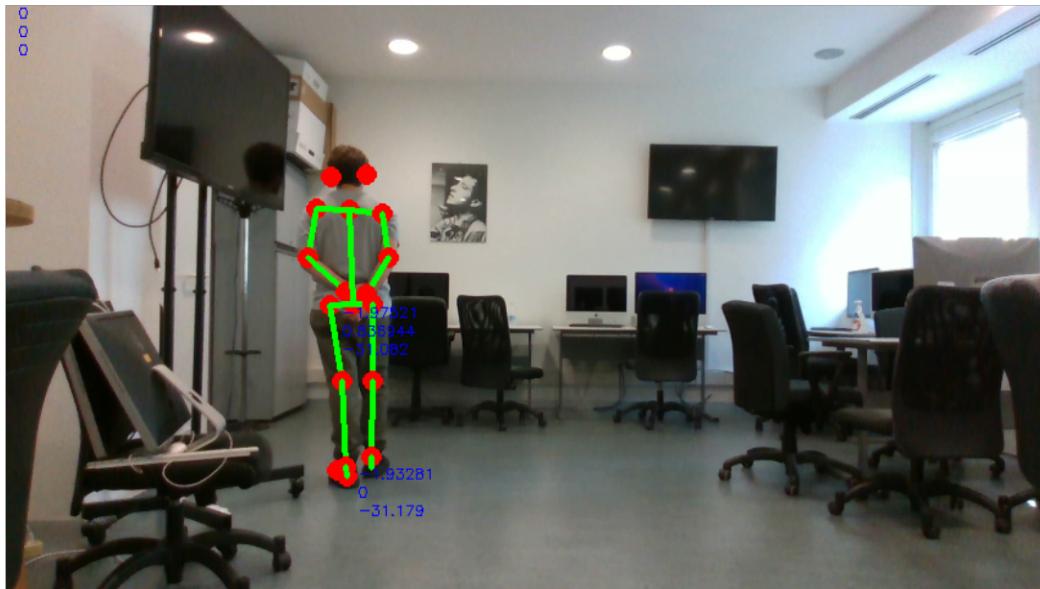


Figura 3.22. Immagine risultato skeleton frame 278 - Test 3.

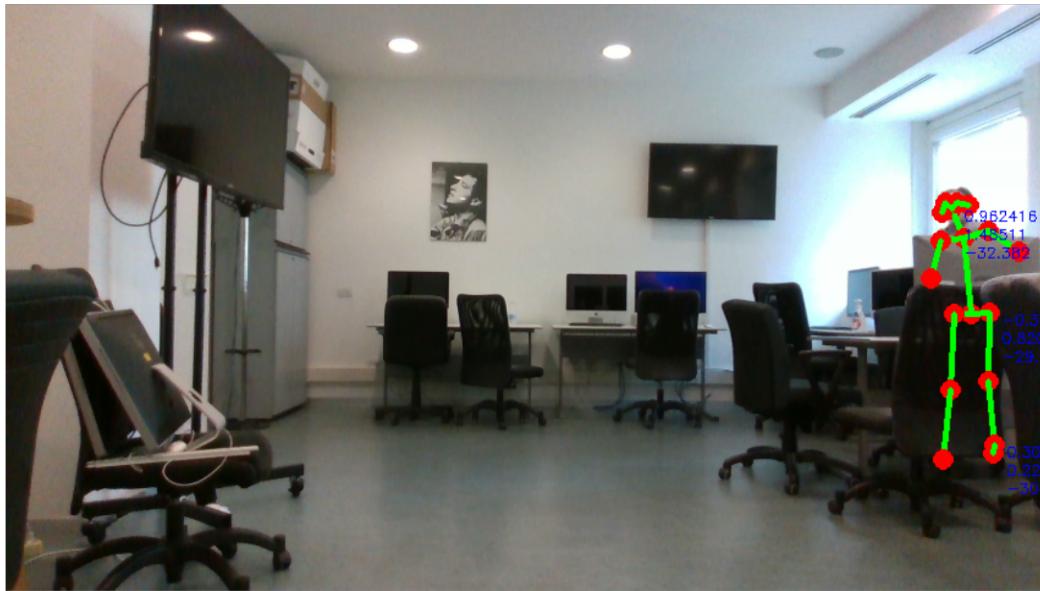
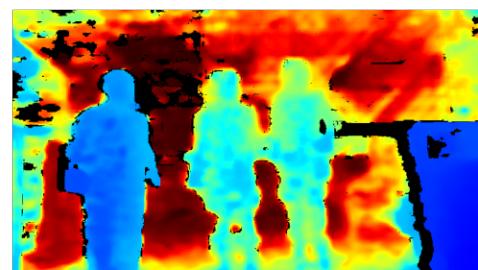


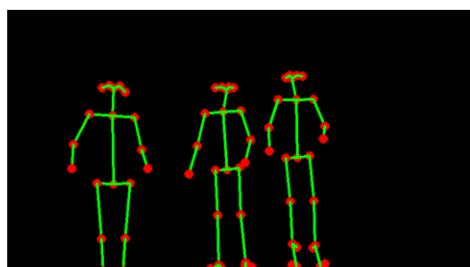
Figura 3.23. Immagine risultato skeleton frame 163 - Test 3.



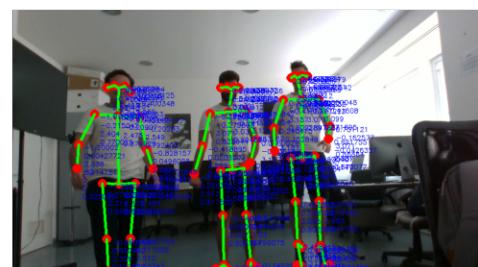
(a) Frame RGB



(b) Frame Profondità



(c) Frame con estrazione skeleton



(d) Frame con skeleton in overlay e coordinate

Figura 3.24. Acquisizione e scissione frame video, rilevazione ed estrazione skeleton, conversione e rendering delle coordinate.

3.7 Architettura complessiva A1

Dopo aver discusso ampiamente sia degli aspetti progettuali, che implementativi, si riporta in figura 3.25 l’architettura completa realizzata durante il progetto di tesi. Nel primo capitolo sono state introdotte alcune delle problematiche che il contesto applicativo, considerato durante il progetto di tesi, portava con sé.

Come accennato sia durante questo capitolo, che durante il capitolo della progettazione, nel modulo A1 vi sono alcune componenti fondamentali: quella per l’acquisizione dei frame, quella per la loro conversione, quella per il rilevamento e costruzione degli skeleton, quella per la conversione dei sistemi di coordinate, quella per l’invio delle informazioni al modulo successivo e tante altre. Schematizzando e associando a ogni componente un relativo approccio o una relativa tecnologia si ottiene lo schema mostrato di seguito.

Si precisa che l’illustrazione semplifica di molto l’organizzazione architettonica dell’intero progetto e che tiene conto di una sola telecamera, così come tiene conto di un solo topic in Apache Kafka. Per quanto riguarda l’estensibilità, questa è garantita dal fatto che vengono già generati da OpenPose tutti gli skeleton di tutte le persone individuate all’interno della scena. Così facendo basta modificare opportunamente la componente che si interfaccia con Apache Kafka per andare a produrre le informazioni di skeleton differenti in topic differenti. Analogamente è possibile modificare opportunamente e con le dovute considerazioni di carattere geometrico, la componente relativa all’acquisizione dei frame, in maniera tale per cui si consideri l’informazione proveniente da più telecamere di profondità.

Inoltre, si precisa che nel seguente schema la dicitura “RW”, indica operazioni di *lettura* e *scrittura* che, a seconda della componente che le implementa, avvengono in una determinata maniera, anziché un’altra. Per esempio il significato delle operazioni di *lettura* e *scrittura* per la componente relativa a *OpenCV*, indica la lettura, la scrittura e il caricamento d’immagini da e sul disco. Invece, il significato delle operazioni di *lettura* e *scrittura* per la componente relativa a *json cpp*, indica la lettura, la scrittura e il caricamento di file contenenti strutture di tipo JSON.

Infine, nell’algoritmo 4 si riassumono brevemente le operazioni principali di AI Watch A1 mediante pseudocodice.

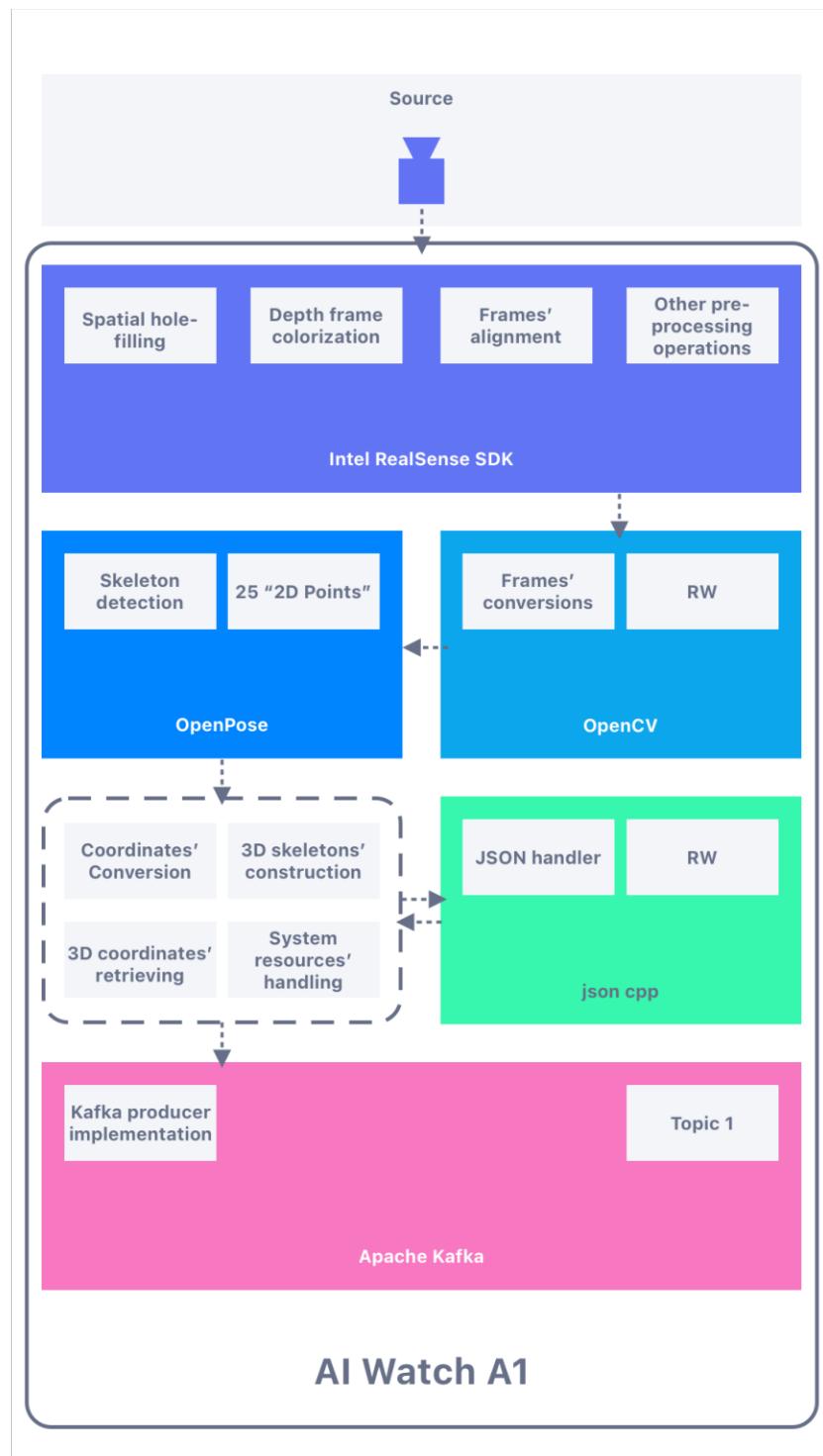


Figura 3.25. AI Watch A1 nel dettaglio.

Algoritmo 4. Operazioni principali di AI Watch A1.

Input : nFrame, kafkaTopicName

```

1 camera ← turnOnCamera();
2 camera.turnOnStreams(camera.RGB, camera.D);
3 camera.D.alignTo(camera.RGB);
4 camera.calibrate();
5 camera.saveParameters();
6 while true
7   for i ← 0 to nFrame, i += 1 do
8     rgbFrame ←
9       camera.capture(camera.RGB, withSkipOffset : 5);
10    depthFrame ←
11      camera.capture(camera.D, withSkipOffset :
12        5, withFilter : holeFillingFilter);
13      saveFrames(rgbFrame, depthFrame, withFrameID : i);
14    end
15    executeOpenPose(frameAmount : nFrame);
16    for i ← 0 to nFrame, i += 1 do
17      peopleAmount ←
18        getPeopleAmountFromFrame(withFrameID : i);
19      for j ← 0 to peopleAmount, j += 1 do
20        singleSkeleton ← buildSkeleton(withFrameID :
21          i, forPerson : j);
22        if singleSkeleton.getOverallConfidence() <= 0.4
23          | continue;
24        depthFrame ← readFrame(type :
25          camera.D, withFrameID : i);
26        singleSkeleton3D ←
27          createSkeleton3D(singleSkeleton, depthFrame);
28        singleSkeletonUnity3D ←
29          convertSkeletonToFormat(singleSkeleton3D, Unity);
30          skeletonsFrame.addSkeleton(singleSkeletonUnity3D);
31      end
32      saveFrames(skeletonFrame);
33      currentJSON ← generateJSON(skeletonFrame, i);
34      sendOutputViaKafka(currentJSON, kafkaTopicName);
35      clean();
36    end
37  end
38 end

```

Capitolo 4

Conclusione e sviluppi futuri

Questo capitolo ha lo scopo di riassumere il percorso affrontato durante questo elaborato di tesi e valutare il raggiungimento degli obiettivi iniziali proposti. Inoltre, si provvederà a proporre alcuni miglioramenti e sviluppi futuri per il modulo A1 e per l'intero sistema AI Watch.

Si è giunti al termine di questo elaborato di tesi. Inizialmente sono stati introdotti gli aspetti teorici fondamentali. Per esempio sono stati presi in considerazione i seguenti argomenti: com’è strutturata e rappresentata un’immagine, quali sono le differenze tra un’immagine in scala di grigi e una a colori, cos’è RGBD, come l’occhio percepisce e valuta la profondità nella scena che osserva, come quest’ultima attività viene eseguita da una telecamera di profondità, come viene determinata la profondità degli oggetti presenti all’interno delle immagini acquisite, quali sono i concetti fondamentali di geometria epipolare e morfologia matematica da tenere presenti, dei cenni relativi all’intelligenza artificiale, cos’è una rete neurale artificiale, cos’è una rete neurale convoluzionale e come funziona, il concetto di “Non Maximum Supresion”, cos’è un grafo e cos’è un algoritmo greedy.

Dati questi fondamentali ingredienti, si è passati ad analizzare le problematiche, le soluzioni e gli approcci risolutori esistenti. Si è presentata l’architettura del sistema software progettato, si è analizzata la complessità computazionale e il flusso di esecuzione. A tal punto sono state presentate le scelte implementative, poi sono state motivate e integrate con il resto del sistema. Così facendo, è stato possibile avere una panoramica completa e dettagliata del modulo AI Watch A1 e del suo funzionamento. Tra le varie scelte implementative sono state considerate quella relativa alla telecamera Intel RealSense D435, quella relativa all’algoritmo per la trasformazione di sistemi di coordinate, quella relativa a OpenPose e ad Apache Kafka. Infine,

Conclusione e sviluppi futuri

sono stati presentati i test condotti e i risultati ottenuti da diverse esecuzioni del modulo A1.

Considerando gli obiettivi iniziali di questo elaborato di tesi, si può affermare che sono stati raggiunti tutti. Infatti, è stato possibile realizzare con le diverse tecnologie prese in considerazione, con i diversi approcci e le differenti tecniche di progettazione del software, realizzare il modulo A1 di AI Watch. Ricapitolando, AI Watch rappresenta, nel suo insieme, un sistema di videosorveglianza 3D che:

- tiene conto dell'informazione sulla profondità grazie a una telecamera Intel RealSense D435;
- è in grado di rilevare più persone all'interno della scena osservata a partire dai loro skeleton mediante OpenPose;
- è in grado di comunicare tra i suoi vari moduli interni mediante Apache Kafka;
- è capace di rilevare anomalie e comportamenti anomali;
- è di facile implementazione ed economicamente conveniente;
- riproduce la scena osservata in ambiente virtuale Unity con digital twin;
- memorizza in maniera permanente e sicura le informazioni relative alle anomalie su una blockchain dedicata.

Il modulo AI Watch A1 in termini di risultati, riesce a ottenere dei dati molto accurati se si tiene conto del fattore movimento, del fattore distanza e del fattore relativo all'imprecisione, seppur minima, della telecamera. Inoltre, sono stati commentati ampiamente gli aspetti implementativi, le scelte progettuali, l'organizzazione del progetto e l'architettura complessiva.

Chiaramente è sempre possibile apportare miglioramenti, introdurre novità e nuove caratteristiche al modulo A1. Ciò è possibile grazie all'estensibilità e alla facilità di manutenzione che lo caratterizzano. In particolare, di seguito, sono proposti alcuni miglioramenti e sviluppi futuri che è possibile apportare al modulo A1 e più in generale all'intero sistema di videosorveglianza AI Watch.

Conclusione e sviluppi futuri

- Aggiungere il supporto per più telecamere Intel RealSense D435. Questo miglioramento è significativo e complesso. Infatti, è necessario realizzare un meccanismo per il matching delle coordinate e degli skeleton rilevati da più telecamere Intel RealSense D435. Inoltre, è necessario riuscire a sincronizzare dal punto di vista temporale i frame, e di conseguenza le informazioni, acquisite da telecamere differenti.
- Ottenere informazioni sulla rotazione lungo i tre assi del sistema di coordinate fissato, così da determinare anche l'angolazione degli skeleton nella scena e rispetto a essa. Un approccio potrebbe essere mediante telecamere D435i, ovvero il modello che prevede anche il sensore IMU.
- Attuare un approccio che permetta di determinare l'orientazione dello skeleton individuato, in base ai punti rilevati. Per esempio in base alla posizione dei punti del viso è possibile stabilire dove è rivolto il volto. Processo analogo per le gambe, il busto e così via. Questo permette di realizzare un gemello digitale molto accurato nelle fasi successive.
- Permettere la visione della scena, visualizzata momentaneamente in Unity, anche con Oculus o tecnologie simili al fine di realizzare un applicativo che comprenda anche la realtà virtuale. Potenzialmente in questo scenario, l'operatore che indossa il visore, vede ciò che il gemello digitale vede ed eventualmente si può fare in modo che possa anche interagire con la scena.
- Miglioramenti nelle dimensioni dello spazio di coordinate e relativo ambiente Unity rispetto alle dimensioni effettive del laboratorio.
- Realizzare un'implementazione su una o più macchine ad alte prestazioni in modo da velocizzare i tempi di esecuzione e ottenere un'architettura dell'intero sistema in cui vi sono più telecamere che inquadrano più scene, che verranno poi elaborate successivamente dai corrispettivi moduli seguenti.
- Considerare eventuali alternative all'attuale rilevamento di anomalie ed eventualmente potenziarlo in base a comportamenti anomali inquadратi nella scena mediante convolutional LSTM autoencoder deep neural network come descritto al seguente riferimento [66].
- Permettere la distinzione di diversi comportamenti anomali, nonché il relativo rilevamento.
- Permettere la distinzione e il rilevamento di oggetti pericolosi.

Conclusione e sviluppi futuri

- Realizzare un applicativo che permetta di far replicare a un robot i movimenti dell'intero corpo o soltanto di un arto della persona inquadrata nella scena. Questo può essere utile, per esempio, al fine di compiere azioni in ambienti estremamente pericolosi per l'essere umano, ma non per un robot; oppure in ambienti fatali per la vita dell'operatore umano.
- Introduzione di un meccanismo ottimale di approssimazione e trasformazione delle coordinate, in caso d'installazione in una scena la cui stanza non è precisamente quadrata o rettangolare o approssimabile a tale forme, ma per esempio trapezoidale. In tal caso è possibile, per esempio, usare una costante moltiplicativa che è funzione della distanza e, in base a tale costante, ottenere una distanza dalla parete a sinistra, considerata obliqua in questa supposizione. Chiaramente però, operativamente parlando, stanze di forma diversa da un quadrato e da un rettangolo non sono molto comuni.

Riferimenti

Bibliografia

- [1] Luca Rubino
“Sviluppo di un’architettura di comunicazione distribuita con digital twins per la videosorveglianza”
Tesi di laurea triennale,
Informatica,
Università degli Studi di Napoli Parthenope. 2022.
- [2] Renato Esposito
“Sviluppo di un ambiente virtuale con digital twins per la videosorveglianza”
Tesi di laurea triennale,
Informatica,
Università degli Studi di Napoli Parthenope. 2022.
- [3] Rafael C. Gonzalez e Richard E. Woods
“Elaborazione delle immagini digitali”
pp. 43-70; 88-94; 389-408; 433-449; 631-650; 669-674. 2008.
- [4] Frank L. Kooi e Alexander Toet
Visual Comfort of Binocular and 3-D Displays
In Proceedings of SPIE — The International Society for Optical Engineering
pp. 99–108. 2004.
- [5] Barbara Sweet e Mary Kaiser
“Depth perception, cueing, and control”
In: AIAA Modeling and Simulation Technologies Conference.
p. 6424. 2011.
- [6] Luisa Sangregorio
Estimating Depth Images from Monocular Camera with Deep Learning for Service Robotics Applications

Riferimenti

- Tesi di laurea magistrale,
Ingegneria meccatronica,
Politecnico di Torino
pp. 3-6. 2022.
- [7] R. I. Hartley e A. Zisserman
“Multiple View Geometry in Computer Vision”
Cambridge University Press, ISBN: 0521540518. 2004.
 - [8] Gang Xu e Zhengyou Zhang
“Epipolar Geometry in Stereo, Motion and Object Recognition: A Unified Approach”
1996.
 - [9] WS. McCulloch e W. Pitts
“A logical calculus of the ideas immanent in nervous activity - The bulletin of mathematical biophysics”
1943.
 - [10] D. O. Hebb
“The organization of behavior: a neuropsychological theory”
1949.
 - [11] F. Rosenblatt
“The perceptron: a probabilistic model for information storage and organization in the brain”
Psychological review. 1958.
 - [12] Marvin Minsky e Seymour Papert
“An introduction to computational geometry”
Cambridge tiass. and HIT. 1968.
 - [13] P. J. Werbos
“An overview of neural networks for control”
IEEE Control Systems Magazine. 1991.
 - [14] Zhenzhu Meng, Yating Hu e Christophe Ancey
“Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows”
2020.
 - [15] Y. LeCun e Y. Bengio
“Convolutional networks for images speech and time series of brain theory and neural networks”
1995.

Riferimenti

- [16] Pasquale Auriemma
“*Classificazione di segnali EEG mediante tecniche di deep learning*”
Tesi di laurea magistrale,
Informatica,
Università degli Studi di Napoli Parthenope. 2019.
- [17] Ian Goodfellow, Yoshua Bengio e Aaron Courville
“*Deep Learning*”
<http://www.deeplearningbook.org>,
MIT Press. 2016.
- [18] Thomas H. Cormen (Autore), Charles E. Leiserson (Autore), Ronald L. Rivest (Autore), Clifford Stein (Autore) e L. Colussi (a cura di)
“*Introduzione agli algoritmi e strutture dati*”
pp. 344; 970-980. 2010.
- [19] Richard Szeliski
Computer Vision: Algorithms and Applications
2010.
- [20] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen e Achintya Bohwmik
“*Intel RealSense Stereoscopic Depth Cameras*”
CCD 2017, CVPR 2017. 2017.
- [21] Clément Godard, Oisin Mac Aodha e Gabriel J. Brostow
“*Unsupervised monocular depth estimation with left-right consistency*”
In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 270– 279. 2017.
- [22] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman e Vivienne Sze
“*Fastdepth: Fast monocular depth estimation on embedded systems*”
International Conference on Robotics and Automation (ICRA). IEEE, pp. 6101–6108. 2019.
- [23] Baek S., Kwang I.K. e Kim T.-K.
“*Augmented skeleton space transfer for depth-based hand pose estimation*”
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,
pp. 8330-8339. 2018.

Riferimenti

- [24] Mehta D., Sridhar S., Sotnychenko O., Rhodin H., Shafiei M., Seidel H.P., Weipeng X., Casas D. e Theobalt C.
“*Real-time 3D human pose estimation with a single RGB camera*”
ACM Trans. Graph. (TOG). 2017.
- [25] Ling J., Tian L., Li C. e Theobalt C.
“*3D human activity recognition using skeletal data from rgbd sensors*”
Conference paper, pp. 133–142. 2016.
- [26] Wang J., Liu Z., Ying W. e Yuan J.
“*Learning actionlet ensemble for 3D human action recognition*”
IEEE Trans. Pattern Analy. Mach. Intell.,
914–927. 2013.
- [27] Wang J., Liu Z., Ying W. e Yuan J.
“*Sensor-based human activity recognition in a multi-user scenario*”
European Conference on Ambient Intelligence,
pp. 78–87. 2009.
- [28] Wang L., Huynh D.Q. e Koniusz D.Q.
“*A comparative review of recent kinect-based action recognition algorithms*”
IEEE Trans. Image Process. and
29, 15–28. 2019.
- [29] Md Atiqur Rahman Ahad, Upal Mahbub e Tauhidur Rahman
“*Contactless Human Activity Analysis*”
chpt. 1-2. 2021.
- [30] Huang J., Xiang X., Gong X., Zhang B. e et al.
“*Long-short graph memory network for skeleton-based action recognition*”
The IEEE Winter Conference on Applications of Computer Vision,
pp. 645–652. 2020.
- [31] Si C., Chen W., Wang W., Wang L. e Tan T.
“*An attention enhanced graph convolutional lstm network for skeleton-based action recognition*”
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,
pp. 1227–1236. 2019.
- [32] Yan S., Li Z., Xiong Y., Yan H. e Lin D.
“*Convolutional sequence generation for skeleton-based action synthesis*”

Riferimenti

- Proceedings of the IEEE International Conference on Computer Vision,
pp. 4394–4402. 2019.
- [33] Zhao R., Wang K., Su K. e Ji Q.
“Bayesian graph convolution lstm for skeleton based action recognition”
Proceedings of the IEEE International Conference on Computer Vision,
pp. 6882–6892. 2019.
- [34] Shi L., Zhang Y., Cheng J. e Lu H.
“Two stream adaptive graph convolutional networks for skeleton-based action recognition”
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,
pp. 12026–12035. 2019.
- [35] Yang H., Yan D., Zhang L., Li D., Sun Y.D., You S.D. e Maybank S.J.
“Feedback graph convolutional network for skeleton-based action recognition”.
- [36] Zhu G., Zhang L., Li H., Shen P., Afaq Ali Shah S. e M. Bennamoun
“Topology-learnable graph convolution for skeletoBased action recognition”
Pattern Recogn. Lett. 2020.
- [37] Chen Y., Ma G., Yuan C., Li B., Zhang H., Wang F. e Hu W.
“Graph convolutional network with structure pooling and joint-wise channel attention for action recognition”
Pattern Recogn. and
p. 107321. 2020.
- [38] Tang Y., Tian Y., Lu J., Li P. e Zhou J.
“Deep progressive reinforcement learning for skeleton-based action recognition”
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,
pp. 5323–5332. 2018.
- [39] Caetano C., Brémont F. e Schwartz W. R.
“Skeleton image representation for 3D action recognition based on tree structure and reference joints”
32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI),
pp. 16–23. IEEE. 2019.

Riferimenti

- [40] Ke Q., Bennamoun M., An A., Sohel F. e Boussaid F.
“*A new representation of skeleton sequences for 3D action recognition*”
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,
pp. 3288–3297. 2017.
- [41] de Boissiere A.M. e Noumeir R.
“*Infrared and 3D skeleton feature fusion for RGBD action*”
2020.
- [42] Lee I., Kim D., Kang S. e Lee S.
“*Ensemble deep learning for skeleton-based action recognition using temporal sliding LSTM networks*”
Proceedings of the IEEE International Conference on Computer Vision,
pp. 1012–1020. 2017.
- [43] Rahmani H. e Bennamoun M.
“*Learning action recognition model from depth and skeleton videos*”
Proceedings of the IEEE International Conference on Computer Vision,
pp. 5832–5841. 2017.
- [44] Zhang P., Lan C., Xing J., Zeng W., Xue J. e Zheng N.
“*View adaptive recurrent neural networks for high performance human action recognition from skeleton data*”
Proceedings of the IEEE International Conference on Computer Vision,
pp. 2117–2126. 2017.
- [45] Zhe Chao, Gines Hidalgo, Tomas Simon e Yaser Sheikh
“*OpenPose - Realtime multi-person 2D pose estimation using Part Affinity Fields*”
IEEE Transactions on Pattern Analysis and Machine Intelligence,
Carnegie Mellon University. 2019.
- [46] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides
“*Design Patterns: Elements of Reusable Object-Oriented Software*”
pp. 127-134; 175-184; 185-193; 233-242; 315-323. 1995.
- [47] Eric Freeman, Elisabeth Robson, Kathy Sierra e Bert Bates
“*Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software*”
2020.

Riferimenti

- [48] Robert C. Martin
“*Clean Code: A Handbook of Agile Software Craftsmanship*”
2008.
- [49] Alexander Shvets
“*Dive into Design Patterns*”
2021.
- [50] Jun Ohta
“*Smart CMOS Image Sensors and Applications*”
2020.
- [51] A. Toshev e C. Szegedy
“*DeepPose - Human pose estimation via deep neural networks*”
CVPR. 2013.
- [52] Varun Ramakrishna, Daniel Munoz, Martial Hebert, J. Andrew Bagnell e Yaser Sheikh
“*Pose Machines - Articulated Pose Estimation via Inference Machines*”
Carnegie Mellon University. 2014.
- [53] M. Andriluka, S. Roth e B. Schiele
“*Pictorial structures revisited - People detection and articulated pose estimation*”
CVPR. 2009.
- [54] M. Andriluka, S. Roth e B. Schiele
“*Monocular 3D pose estimation and tracking by detection*”
CVPR. 2010.
- [55] P. Felzenszwalb e D. Huttenlocher
“*Pictorial structures for object recognition*”
IJCV. 2005.
- [56] S. Johnson e M. Everingham
“*Clustered pose and nonlinear appearance models for human pose estimation*”
BMVC. 2010.
- [57] L. Pishchulin, M. Andriluka, P. Gehler e B. Schiele
“*Poselet conditioned pictorial structures*”
CVPR. 2013.
- [58] L. Pishchulin, M. Andriluka, P. Gehler e B. Schiele
“*Strong appearance and expressive spatial models for human pose estimation*”
ICCV. 2013.

Riferimenti

- [59] D. Ramanan, D. A. Forsyth e A. Zisserman
“*Strike a Pose - Tracking people by finding stylized poses*”
CVPR. 2005.
- [60] Y. Yang e D. Ramanan
“*Articulated pose estimation with flexible mixtures-of-parts*”
CVPR. 2011.
- [61] Tomas Simon, Hanbyul Joo, Iain Matthews e Yaser Sheikh
“*Hand Keypoint Detection in Single Images using Multiview Bootstrapping*”
CVPR,
Carnegie Mellon University. 2017.
- [62] Chuan-Bi Lin, Ziqian Dong, Wei-Kai Kuan e Yung-Fa Huang
“*A Framework for Fall Detection Based on OpenPose Skeleton and LSTM/GRU Models*”
2020.
- [63] BeomJun Jo e SeongKi Kim
“*Comparative Analysis of OpenPose, PoseNet, and MoveNet Models for Pose Estimation in Mobile Devices*”
2021.
- [64] Sarah Mroz, Natalie Baddour, Connor JC McGuirk, Pascale Juneau, Albert Tu, Kevin Cheung e Edward Donat Lemaire
“*Comparing the Quality of Human Pose Estimation with BlazePose or OpenPose*”
4th International Conference on Bio-Engineering for Smart Technologies (BioSMART). 2021.
- [65] Pablo Vicente Moñivar
“*Computer Vision For Body Tracking In Professional Environments*”
Universidad Autónoma De Madrid, Master Thesis. 2019.
- [66] Giovanni Hauber
“*Analisi spazio-temporale per il rilevamento di anomalie in videosorveglianza mediante convolutional lstm autoencoder deep neural network*”
Tesi di laurea triennale,
Informatica,
Università degli Studi di Napoli Parthenope. 2019.

Sitografia

- [67] Treccani - Enciclopedia Online
Definizione “Sicurezza”
[https://www.treccani.it/enciclopedia/sicurezza/.](https://www.treccani.it/enciclopedia/sicurezza/)
- [68] Intel Real Sense
Tecnologia Intel RealSense
www.intelrealsense.com,
www.intelrealsense.com/use-cases/,
[https://dev.intelrealsense.com/docs/projectors](http://dev.intelrealsense.com/docs/projectors),
[https://www.intelrealsense.com/tracking-camera-t265/](http://www.intelrealsense.com/tracking-camera-t265),
[https://www.intelrealsense.com/depth-camera-d435/](http://www.intelrealsense.com/depth-camera-d435),
[https://www.intelrealsense.com/depth-camera-d435i/](http://www.intelrealsense.com/depth-camera-d435i),
<https://www.youtube.com/watch?v=tcJHnHpwCXk&t=1s>,
https://www.youtube.com/watch?v=62vm0_RZ1nU,
[https://dev.intelrealsense.com/docs/depth-post-processing](http://dev.intelrealsense.com/docs/depth-post-processing).
- [69] www.width.ai/services/object-recognition-software-services,
www.biometricupdate.com/202004/morocco-extends-facial-recognition-moratorium-to-year-end-proposes-biometric-authentication-service,
www.therobotreport.com/righthands-yaro-tenzer-on-robotic-piece-picking-waymos-sf-expansion/,
[https://maelfabien.github.io/tutorials/open-pose/](http://maelfabien.github.io/tutorials/open-pose/).
- [70] <https://www.uniparthenope.it/>,
<http://cvprlab.uniparthenope.it/>.
- [71] Denny Caruso
Demo YouTube: OpenPose + RealSense D435 - 3D Body Tracking - CVPRLAB
www.youtube.com/watch?v=Ac0V8Dj0FbI,
<https://www.youtube.com/watch?v=pq3m9U3hRrQ>.
- [72] *OpenPose*
<https://github.com/CMU-Perceptual-Computing-Lab/openpose>.
- [73] Unity Technologies
Unity
www.unity.com.
- [74] *Apache Kafka*
<https://kafka.apache.org/>,
<https://github.com/apache/kafka>.

Riferimenti

- [75] [https://www.radio2space.com/it/le-componenti-dello-spettro-elettromagnetico/.](https://www.radio2space.com/it/le-componenti-dello-spettro-elettromagnetico/)
- [76] Microsoft
Microsoft Kinect
[https://azure.microsoft.com/it-it/services/kinect-dk/.](https://azure.microsoft.com/it-it/services/kinect-dk/)
- [77] Raytrix
Telecamere Raytrix
[https://raytrix.de/.](https://raytrix.de/)
- [78] Marko Teittinen
“*Depth Cues in the Human Visual System*”
http://www.hitl.washington.edu/projects/knowledge_base/virtual-worlds/EVE/III.A.1.c.DepthCues.html.
- [79] [https://lenti-contatto.1000lenti.it/accomodazione-oculare-di-cosa-si-tratta/.](https://lenti-contatto.1000lenti.it/accomodazione-oculare-di-cosa-si-tratta/)
- [80] <https://www.burlo.trieste.it/newsletter-febbraio-2022/stereopsi-presente-o-assente-cosa-comporta>.
- [81] https://it.wikipedia.org/wiki/Geometria_epipolare,
<https://medium.com/minidistill/pps-efficient-deep-learning-for-stereo-matching-de253fc411d4>.
- [82] [https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/.](https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/)
- [83] *Non Maximum Suppression - Theory and Implementation in PyTorch*
<https://learnopencv.com/wp-content/uploads/2021/06/nms-intro.png>.
- [84] <https://it.wikipedia.org/wiki/Grafo>,
https://it.wikipedia.org/wiki/Grafo_bipartito,
[https://www.mecchanismocomplesso.org/programming-graphs-in-python-part-1-grafi/.](https://www.mecchanismocomplesso.org/programming-graphs-in-python-part-1-grafi/)
- [85] Zivid
Telecamere Zivid
[https://www.zivid.com/.](https://www.zivid.com/)
- [86] Revopoint
Telecamere Revopoint
[https://3dcamera.revopoint3d.com/.](https://3dcamera.revopoint3d.com/)
- [87] Orbbec
Telecamere Orbbec
[https://orbbc3d.com/.](https://orbbc3d.com/)

Riferimenti

- [88] Stereolabs
Telecamere Stereolabs
[https://www.stereolabs.com/.](https://www.stereolabs.com/)
- [89] Sanja Fidler
“*Intro to Image Understanding*”
[https://www.cs.utoronto.ca/~fidler/teaching/2021/CSC420.html.](https://www.cs.utoronto.ca/~fidler/teaching/2021/CSC420.html)
- [90] *PoseNet*
[https://github.com/tensorflow/tfjs-models/tree/master/pose-detection.](https://github.com/tensorflow/tfjs-models/tree/master/pose-detection)
- [91] *MoveNet*
[https://www.tensorflow.org/hub/tutorials/movenet.](https://www.tensorflow.org/hub/tutorials/movenet)
- [92] *Mediapipe*
[https://mediapipe.dev/.](https://mediapipe.dev/)
- [93] *Amazon Kinesis*
[https://aws.amazon.com/it/kinesis/.](https://aws.amazon.com/it/kinesis/)
- [94] *RabbitMQ*
[https://www.rabbitmq.com/.](https://www.rabbitmq.com/)
- [95] *ActiveMQ*
[https://activemq.apache.org/.](https://activemq.apache.org/)
- [96] *RedHatAMQ*
[https://www.redhat.com/it/technologies/jboss-middleware/amq.](https://www.redhat.com/it/technologies/jboss-middleware/amq)
- [97] Intel Real Sense
Specifiche tecniche Intel RealSense D435
[https://www.intelrealsense.com/wp-content/uploads/2022/05/Intel-RealSense-D400-Series-Datasheet-April-2022.pdf/.](https://www.intelrealsense.com/wp-content/uploads/2022/05/Intel-RealSense-D400-Series-Datasheet-April-2022.pdf/)
- [98] *AlphaPose*
[https://github.com/MVIG-SJTU/AlphaPose.](https://github.com/MVIG-SJTU/AlphaPose)
- [99] [https://github.com/dennewbie/AI_Watch_A1.](https://github.com/dennewbie/AI_Watch_A1)
- [100] [https://en.wikipedia.org/wiki/Rolling_shutter.](https://en.wikipedia.org/wiki/Rolling_shutter)
- [101] [https://www.intelrealsense.com/sdk-2/.](https://www.intelrealsense.com/sdk-2/)
- [102] Intel
Intel RealSense Viewer
[https://www.intelrealsense.com/download/7144/.](https://www.intelrealsense.com/download/7144/)

Riferimenti

- [103] <https://gisgeography.com/lidar-light-detection-and-ranging/>,
<https://dev.intelrealsense.com/docs/projectors>.
- [104] *Tecnologia Lidar*
<https://consystem.it/faq/tecnologia-lidar-che-cosa-e-come-funziona/>.
- [105] *Conduktor*
<https://www.conduktor.io/>,
<https://www.conduktor.io/kafka/kafka-fundamentals>,
<https://www.conduktor.io/kafka/kafka-topics>,
<https://www.conduktor.io/kafka/kafka-producers>,
<https://www.conduktor.io/kafka/kafka-topic-replication>.
- [106] *Confluent*
<https://www.confluent.io/>.
- [107] <https://docs.microsoft.com/it-it/cpp/cpp/?view=msvc-170>.
- [108] <https://github.com/IntelRealSense/librealsense>.
- [109] <https://github.com/opencv/opencv>.
- [110] <https://github.com/edenhill/librdkafka>.
- [111] <https://github.com/open-source-parsers/jsoncpp>.
- [112] <https://cmake.org/>.
- [113] <https://opensource.com/article/18/8/what-how-makefile>.
- [114] <https://doxygen.nl/>.
- [115] https://dennewbie.github.io/AI_Watch_A1/.

Ringraziamenti

Alla fine di questo elaborato di tesi di laurea e di questo lungo percorso, credo sia giusto e corretto dare un riconoscimento anche a chi, a vario titolo e ruolo, ha reso possibile tutto ciò. Innanzitutto, voglio ringraziare vivamente l'intero corpo docente del corso di laurea triennale in *Informatica* per la loro disponibilità, per l'impegno e la dedizione alla loro attività. Tutte queste caratteristiche insieme alla preparazione poliedrica che forniscono, fanno sì che l'esperienza di uno studente universitario di questo corso di laurea sia veramente eccezionale. In particolare, ci tengo a ringraziare il mio relatore: il prof. *Ferone Alessio*, per la sua affabilità e per i preziosi suggerimenti forniti. Un ringraziamento specifico va anche all'Università *Parthenope* che ha messo a disposizione laboratori avanzati e apparecchiature come la telecamera, utilizzata durante tutto il progetto di tesi.

Inoltre, un ringraziamento particolare va ai miei genitori, *Angela* e *Salvatore Caruso*, che mi hanno sostenuto e che hanno fatto in modo che lo *studio* potesse essere la mia unica preoccupazione durante questi tre anni.

Infine, una menzione speciale anche per i miei colleghi con i quali ho avuto modo d'interagire durante questi tre anni. Quindi ringrazio *Alfredo Mungari*, *Massimiliano Giordano Orsini*, *Dominick Ferraro* e *Francesco Calcopietro*, con i quali sono avvenuti confronti e collaborazioni anche su base quotidiana; i rappresentanti degli studenti *Aniello Genovese* e *Ilaria Scuotto*: sempre pronti a fornire importanti informazioni relative al percorso di laurea; i colleghi con i quali ho lavorato al progetto *AI Watch A1*: *Luca Rubino* e *Renato Esposito* e in generale tutti i colleghi universitari che hanno reso questa esperienza molto più piacevole, tra cui *Mariano Aponte*, *Gianfranco Terrazzano*, *Gianluca Ilardo*, *Luigi Malvone*, *Simone Cioffi*, *Nicola Improta*, *Mario Giordano*, *Francesco Zibalbo*, *Zeno Dall'Acqua*, *Giuseppe Margherita*, *Nicola Esposito*, *Igor Bosco*, *Giuseppe Cuccurullo*, *Fabio Esposito*, *Michele Fiorentino*, *Eugenio Militerno*, *Veronica Filosa*, *Carlo Matarrelli*, *Stefano Leva*.

