POLITECNICO DI TORINO

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Programmazione robotica per imitazione: dispositivi aptici e machine learning per riconoscere le azioni dell'utente

Relatori

Prof. Andrea SANNA

Prof. Federico MANURI

Candidato

Luca FILIPPETTI

Anno Accademico 2024/2025

Abstract

Uno degli approcci più moderni per programmare robot collaborativi è la programmazione per imitazione, che consente di far comprendere e replicare i movimenti della mano al robot semplicemente osservando l'operatore umano. Questa metodologia riduce la necessità di programmazione esplicita, rendendo i robot più adattabili, intuitivi e facilmente integrabili in ambienti industriali e produttivi.

Tradizionalmente il riconoscimento delle azioni umane è stato affrontato con tecniche basate su computer vision, che analizzano immagini RGB o RGB-D per estrarre informazioni sulla posizione e il movimento delle mani. Tuttavia, questi metodi presentano limitazioni tra cui occlusioni, sensibilità all'illuminazione e dipendenza dalla posizione della telecamera. Per affrontare e superare queste problematiche, questo studio propone un approccio innovativo basato sulle informazioni dei giunti della mano ricavati da un dispositivo aptico. Questa soluzione permette un'acquisizione più precisa e indipendente dalle condizioni ambientali, migliorando la robustezza del sistema nel riconoscimento delle azioni della mano.

Il sistema proposto riconosce e classifica un insieme di azioni predefinite, selezionate per rappresentare attività semplici e ricorrenti in un tipico ambiente collaborativo con un braccio robotico: *pick and place*, *push*, *pull*, *upside down*, *screw*, *unscrew*, *plug*, e *unplug*. Oltre alla classificazione, il sistema determina la posizione dei giunti della mano nello spazio tramite telecamere a infrarossi HTC Vive Base Station e HTC tracker Vive 3.0 e previa mappatura della posizione degli oggetti nell'area di lavoro, sfrutta i dati di posizione per determinare gli oggetti coinvolti nell'azione.

La pipeline proposta si articola in diverse fasi. Dopo l'acquisizione e preprocessamento dei dati, l'analisi avviene tramite un modello basato sulla Temporal Convolutional Network, adattato dal framework *Domain and View-point Agnostic Hand Action Recognition* [1], sostituendo gli input visivi RGB-D con dati provenienti dal dispositivo aptico. Il processo si conclude con il tracciamento delle coordinate finali dell'oggetto manipolato, essenziali per la replica delle azioni da parte di un sistema robotico.

Un altro contributo di questa tesi è la creazione di un dataset specifico, che contiene i dati scheletrici della posa della mano, rappresentata dalle coordinate

spaziali dei giunti durante l'azione. La registrazione di questi dati è stata condotta con l'utilizzo dei guanti aptici MANUS Prime 3 XR, seguendo una metodologia precisa per garantire variabilità e generalizzazione.

I risultati sperimentali dimostrano che l'utilizzo di dati sensoriali consente di superare i limiti delle tecniche basate esclusivamente su immagini. Inoltre, l'integrazione di questa tipologia di dati porta a un sistema più robusto, capace di classificare correttamente anche azioni non presenti nel dataset di addestramento, dimostrando buone capacità di generalizzazione.

Le future evoluzioni di questa ricerca potranno concentrarsi sull'ampliamento del dataset con nuove azioni, permettendo al sistema di riconoscere un insieme più ampio e diverso di azioni. Parallelamente, un aumento del numero di registrazioni per le singole azioni potrebbe migliorare ulteriormente l'accuratezza e la robustezza del sistema.

Chi non fa niente, non sbaglia.

Indice

Elenco delle tabelle

Elenco delle figure

Abbreviazioni

IA
Intelligenza artificiale

CNN
Convolutional Neural Network

RNN
Recurrent Neural Network

LSTM
Long Short-Term Memory

ViVit
Video Vision Transformer

GNN
Graph Neural Network

GRU
Gated Recurrent Unit

GCN
Graph Convolutional Networks

TCN
Temporal Convolutional Network

KNN
K-Nearest Neighbors

TCP

Transmission Control Protocol

UDP

User Datagram Protocol

t-SNE

t-distributed Stochastic Neighbor Embedding

UMAP

Uniform Manifold Approximation and Projection

Capitolo 1

Introduzione

1.1 Contesto

La combinazione tra *computer vision* e programmazione dei robot ha recentemente rivoluzionato il riconoscimento delle azioni. Questa tesi intende esplorare come l'uso di guanti sensorizzati, capaci di offrire i dati diretti dei movimenti della mano, possa permettere il superamento di alcune limitazioni dei metodi tradizionali basati principalmente su dati visivi, aumentando così la precisione e l'affidabilità dei sistemi robotici.

1.1.1 Computer Vision

La *computer vision* è un'area dell'intelligenza artificiale dedicata all'analisi di immagini e video, che permette ai sistemi artificiali di raccogliere dati rilevanti dall'ambiente circostante. L'unione di metodi sofisticati di apprendimento automatico e di reti neurali ha generato significativi miglioramenti nel riconoscimento di oggetti, nella ricostruzione tridimensionale e nel rilevamento della posizione di elementi in una scena [2].

Un'applicazione fondamentale della *computer vision* riguarda la robotica, nella quale supporta il riconoscimento e la localizzazione di oggetti, la navigazione autonoma e l'interazione uomo-robot[3, 4]. Tecniche di *pose estimation* e *object detection* consentono di riconoscere la disposizione spaziale di un oggetto e prevedere il miglior modo per afferrarlo e maneggiarlo adeguatamente [5]. In aggiunta, grazie ai progressi recenti nell'impiego di modelli di *deep learning*, è fattibile raggiungere una segmentazione accurata della scena e una classificazione attendibile degli oggetti presenti, che portano informazioni utili per i movimenti delle macchine.[6].

L'uso della visione artificiale nella robotica continua a progredire grazie all' avanzamento di algoritmi sempre più performanti e flessibili. L'unione di dati

ottenuti da telecamere con dati tattili e inerziali costituisce un progresso cruciale per una maggiore solidità e adattabilità dei modelli percettivi [7].

1.1.2 Robot Programming

La *computer vision* si è affermata come una tecnologia fondamentale nel campo della robotica, offrendo strumenti avanzati per migliorare la percezione dell'ambiente e semplificare la programmazione dei sistemi robotici. Grazie alla capacità di analizzare e interpretare visivamente lo spazio circostante, è stato possibile sviluppare approcci innovativi che superano i limiti della programmazione esplicita, favorendo soluzioni più flessibili e adattabili [8].

Tradizionalmente, la programmazione dei robot avveniva esclusivamente attraverso istruzioni codificate manualmente da un operatore esperto, che specificava ogni passo da eseguire con precisione. Per quanto riguarda la programmazione robotica essa può essere divisa in tre grandi paradigmi:

- **programmazione esplicita**: ogni azione del robot viene definita manualmente dall'operatore, garantendo un alto grado di affidabilità ma con scarsa adattabilità a nuove situazioni [9].
- **apprendimento per rinforzo**: il robot apprende autonomamente strategie ottimali attraverso l'interazione con l'ambiente, ricevendo ricompense per le azioni corrette e penalizzazioni per quelle errate [10].
- **apprendimento per imitazione**: il robot osserva e imita le azioni di un essere umano, apprendendo dai dati raccolti tramite sensori visivi e dispositivi di tracciamento del movimento [11, 12].

Negli ultimi anni, l'integrazione della visione artificiale con tecniche di apprendimento automatico ha portato alla diffusione di sistemi robotici in grado di interpretare e replicare il comportamento umano con maggiore flessibilità [13, 14]. In particolare, l'uso di sensori visivi avanzati, come telecamere RGB-D e sistemi di stima della posa, consente ai robot di identificare oggetti, riconoscere gesti e prevedere le intenzioni dell'operatore [15]. Questo ha trovato applicazione in numerosi settori, dalla robotica industriale alla realtà aumentata/virtuale, fino alla teleoperazione [16, 17].

L'intelligenza artificiale (IA) ha giocato un ruolo fondamentale in questo processo, permettendo ai robot di apprendere in modo adattivo e migliorare le proprie capacità attraverso il training su grandi dataset di immagini e sequenze di movimento [18]. Grazie alle tecniche di *deep learning*, oggi i robot non si limitano a eseguire istruzioni predefinite, ma possono comprendere il contesto e adattarsi in tempo reale alle variazioni dell'ambiente, garantendo maggiore sicurezza ed efficienza nelle operazioni [19].

1.1.3 Action recognition

L'action recognition è un campo della *computer vision* e del *machine learning* che si occupa di identificare e classificare le azioni compiute da un individuo [20]. Il riconoscimento delle azioni non si limita all'identificazione di un singolo movimento, ma si estende all'interpretazione dell'intenzione dietro di esso. Per esempio, un'azione di sollevamento di un oggetto può indicare diverse intenzioni: spostare l'oggetto, passarlo ad un'altra persona o semplicemente esaminarlo. Per affrontare questa sfida, negli ultimi anni sono stati sviluppati modelli avanzati basati su reti neurali, come *Convolutional Neural Networks* (CNN) [21] e *Recurrent Neural Networks* (RNN) [22], capaci di analizzare sequenze di immagini ed estrapolare il contesto dell'azione. Un altro approccio innovativo è quello basato sui *Transformers* [23, 24], che sfrutta meccanismi di attenzione per catturare la dipendenza spaziale e temporale nelle sequenze video [1].

Hand action recognition

Uno sviluppo più specifico dell'*action recognition* è l'*hand action recognition*, ovvero il riconoscimento delle azioni eseguite dalla mano umana. Questo aspetto è molto importante per applicazioni quali la manipolazione robotica, in cui i robot devono essere in grado di afferare, spostare e utilizzare oggetti proprio come farebbe un essere umano [25]. Tuttavia, il riconoscimento delle azioni della mano presenta diverse sfide:

- **dipendenza dai dati visivi**: la maggior parte dei modelli attuali di *hand action recognition* si basa su sensori visivi (RGB, Depth o Skeleton pose estratti da video) [26]. Tuttavia, questi approcci possono soffrire di problemi legati a condizioni ambientali variabili, occlusioni e cambi di prospettiva che compromettono l'affidabilità del riconoscimento dell'azione [1].

- **occlusioni e variazioni di punto di vista**: quando una mano interagisce con un oggetto, parte delle dita o l'intera mano possono essere nascoste alla telecamera, rendendo difficile il riconoscimento delle azioni. Approcci come HandFormer di Md Salman Shamil et al.[27] cercano di mitigare il problema con reti basate su *Transformers* per estrarre informazioni dettagliate sulla posizione delle mani. Tuttavia, la precisione rimane sensibile all'angolazione della telecamera e alla qualità della segmentazione [28].

- **scarsa generalizzazione tra domini**: i modelli di riconoscimento delle azioni della mano spesso faticano a riconoscere azioni quando vengono applicati a contesti differenti da quelli visti durante l'addestramento [29]

- **dataset limitati**: a differenza dell'*Action recognition* tradizionale, il riconoscimento delle azioni delle mano richiede dataset specializzati e ben annotati [30].

1.2 Motivazione e benefici

Il mio studio si propone di affrontare alcune delle predette limitazioni tramite un approccio basato su guanti sensorizzati. A differenza dei modelli esistenti che utilizzano dati RGB-D o *skeleton* stimati tramite visione artificiale [26],la mia tesi intende utilizzare dati diretti della posizione e dell'orientamento della mano ottenuti grazie all'uso dei guanti. Questa scelta introduce diversi vantaggi:

- **maggiore precisione nella rappresentazione del movimento**: l'uso di guanti con sensori inerziali e di posizione permette di ottenere dati diretti senza il rischio di errori dovuti alla visione artificiale.
- **superamento del problema delle occlusioni**: non essendo basato su immagini o stima visiva, il sistema non è influenzato dalla sovrapposizione di oggetti o dalla posizione della telecamera.
- **robustezza a variazioni ambientali**: l'uso di dati tattili e articolari consente al modello di funzionare anche in condizioni di scarsa illuminazione o in scenari in cui la telecamera non ha una vista chiara delle mani.

Intendo prendere come riferimento il framework proposto da Sabater et al.[1],e adattarlo per l'utilizzo di guanti sensorizzati, con l'obiettivo di valutare l'efficacia del riconoscimento delle azioni delle mani in scenari reali controllati. L'integrazione di questa tecnologia permette di migliorare la generalizzazione del riconoscimento delle azioni, mantenendo un'elevata accuratezza anche in contesti non visti durante l'addestramento.

Capitolo 2

Stato dell'arte

2.1 Approcci al riconoscimento delle azioni

Il riconoscimento delle azioni si basa principalmente sull'analisi di sequenze video piuttosto che su immagini statiche. Questa distinzione rappresenta un elemento chiave nello sviluppo degli algoritmi di analisi, poiché, nonostante la loro natura comune, le tecniche utilizzate per il riconoscimento nelle immagini e nei video hanno alcune caratteristiche differenti.

L'analisi delle immagini ha raggiunto un alto grado di maturazione grazie all'adozione di architetture convoluzionali avanzate, come le reti CNN. Tuttavia, i video introducono una complessità aggiuntiva dovuta alla dimensione temporale, che richiede l'estrazione di informazioni dinamiche oltre a quelle spaziali. Questa caratteristica ha portato allo sviluppo di approcci specifici per sfruttare al meglio la componente temporale.

A livello computazionale, l'analisi di un video implica la gestione di sequenze di fotogrammi, ciascuno trattabile come un'immagine indipendente. Tuttavia, analizzare un video come una mera successione di immagini non permette di catturare le dipendenze temporali tra i frame, rendendo necessario l'utilizzo di architetture capaci di modellare le variazioni nel tempo. Tra le soluzioni più influenti troviamo l'estensione delle reti convoluzionali alle tre dimensioni, come nelle 3D-CNN [31], che permettono di analizzare direttamente blocchi spaziotemporali. Un altro approccio diffuso è quello basato sul flusso ottico, come nelle reti *2-Stream* CNN [32], che sfruttano informazioni derivate dal movimento tra i frame piuttosto che dai singoli fotogrammi.

Tali metodologie, sebbene efficaci nel catturare variazioni locali di movimento, presentano il limite di non modellare in modo esplicito le dipendenze a lungo termine. Per affrontare questo problema, sono state introdotte architetture basate su modelli ricorrenti, come le *Long Short-Term Memory* (LSTM) [33], in grado di

apprendere relazioni a lungo raggio tra gli eventi all'interno della sequenza video. Queste reti sono spesso combinate con CNN per estrarre caratteristiche spaziali e successivamente modellare le dipendenze temporali.

Un ulteriore avanzamento è stato portato dall'introduzione dei *Transformer* [34], che hanno rivoluzionato il campo del riconoscimento delle azioni grazie alla loro capacità di catturare relazioni globali tra i frame senza le limitazioni delle reti ricorrenti. Modelli come *Video Action Transformer Network* [35] e ViViT (*Video Vision Transformer*) [36] hanno dimostrato una superiore capacità di rappresentazione del movimento, permettendo un miglioramento significativo nelle prestazioni del riconoscimento.

L'evoluzione degli approcci ha quindi condotto a una crescente integrazione tra architetture convoluzionali, ricorrenti e basate su attenzione, evidenziando come la rappresentazione efficace della componente temporale rimanga una delle sfide più complesse e affascinanti nel riconoscimento delle azioni nei video.

2.1.1 Approcci basati su visione artificiale

Gli approcci basati su visione artificiale per il riconoscimento delle azioni trattano i video come sequenze di dati RGB, utilizzando tecniche avanzate di *deep learning* per estrarre informazioni spaziali e temporali. L'analisi di sequenze RGB offre un vantaggio significativo in termini di disponibilità dei dati, poiché i video sono facilmente acquisibili attraverso dispositivi standard. Tuttavia, il riconoscimento delle azioni basato esclusivamente su flussi RGB presenta sfide legate alla complessità della modellazione del movimento e alla gestione delle variazioni spaziali e temporali.

I principali approcci proposti nella letteratura possono essere suddivisi in tre grandi categorie [37]:

- reti convoluzionali a due flussi (*Two-Stream* 2D CNN)
- reti neurali ricorrenti (RNN e LSTM)
- reti convoluzionali 3D (3D CNN)

Reti convoluzionali a due flussi (Two-Stream 2D CNN)

Le *Two-Stream* CNN sono un modello basato su due rami di reti convoluzionali 2D, ciascuno dei quali si concentra su caratteristiche diverse del video [32]. Il primo ramo prende in input i frame RGB e cattura informazioni spaziali di apparenza, mentre il secondo utilizza il flusso ottico calcolato tra i frame successivi per estrarre le caratteristiche del movimento. La combinazione delle due reti consente di ottenere una rappresentazione efficace delle azioni, migliorando le prestazioni rispetto a modelli puramente basati su singoli frame.

Tuttavia, il calcolo del flusso ottico introduce un costo computazionale significativo, rendendo l'addestramento e l'inferenza particolarmente onerosi [32]. Inoltre, la separazione esplicita della componente temporale e spaziale può limitare la capacità del modello di apprendere rappresentazioni più profonde e integrate delle azioni.

Reti neurali ricorrenti (RNN e LSTM)

Per affrontare le limitazioni delle reti a due flussi, si sono sviluppati approcci basati su reti neurali ricorrenti (RNN), che modellano direttamente le dipendenze temporali tra i frame [33]. Le RNN standard, tuttavia, soffrono del problema della scomparsa del gradiente, limitando la loro capacità di apprendere relazioni a lungo termine.

Una soluzione efficace a questo problema è rappresentata dalle LSTM, che introducono meccanismi di gating per preservare le informazioni rilevanti lungo la sequenza [33]. Le LSTM possono essere combinate con reti convoluzionali (CNN-LSTM) per estrarre caratteristiche spaziali con le CNN e modellare la dinamica temporale con le LSTM. Questo approccio ha dimostrato miglioramenti significativi nel riconoscimento delle azioni rispetto ai modelli esclusivamente convoluzionali o ricorrenti.

Un ulteriore miglioramento è stato introdotto attraverso le *bidirectional* LSTM (Bi-LSTM) [38], che processano la sequenza in entrambe le direzioni per catturare dipendenze temporali più complesse. Nonostante questi vantaggi, le LSTM possono risultare computazionalmente costose su sequenze lunghe e soffrono di difficoltà nell'apprendimento di rappresentazioni gerarchiche del movimento.

Reti convoluzionali 3D (3D CNN)

Le 3D CNN rappresentano un'estensione naturale delle reti convoluzionali 2D, in cui il terzo asse convoluzionale opera sulla dimensione temporale, permettendo di apprendere simultaneamente caratteristiche spaziali e dinamiche [31]. A differenza delle *Two-Stream* CNN, le 3D CNN elaborano direttamente sequenze di frame, senza necessità di calcolare il flusso ottico separatamente.

L'architettura 3D CNN più nota è il modello C3D [31], che utilizza convoluzioni 3D per catturare il movimento nei video. Tuttavia, l'aumento del numero di parametri rispetto alle reti 2D richiede dataset di grandi dimensioni per un addestramento efficace, oltre a una maggiore capacità computazionale. Modelli più recenti, come I3D [20], sfruttano una pre-formazione su dataset di immagini per migliorare l'efficienza dell'addestramento.

Approcci basati su Transformer

Negli ultimi anni, i modelli basati su *Transformer* hanno guadagnato popolarità nel riconoscimento delle azioni, grazie alla loro capacità di catturare dipendenze a lungo raggio all'interno delle sequenze video [34]. A differenza delle RNN, i *Transformer* utilizzano meccanismi di attenzione per pesare dinamicamente l'importanza di ogni frame rispetto agli altri, superando i limiti della modellazione sequenziale delle LSTM.

Modelli come il *Video Action Transformer Network* [35] e ViVit [36] hanno dimostrato prestazioni superiori sfruttando l'attenzione per apprendere rappresentazioni spaziali e temporali gerarchiche. Questi approcci eliminano la dipendenza dal flusso ottico e dalle strutture ricorrenti, garantendo maggiore parallelismo durante l'addestramento e una più efficace modellazione delle azioni.

L'evoluzione dei modelli ha quindi evidenziato un progressivo spostamento dagli approcci basati su feature ingegnerizzate e reti ricorrenti verso architetture più avanzate, come le 3D CNN e i *Transformer*. Il futuro del riconoscimento delle azioni sembra orientato verso modelli che combinano efficacemente informazioni spaziali e temporali, ottimizzando le risorse computazionali senza sacrificare la qualità delle predizioni.

2.1.2 Approcci basati su dati scheletrici

Gli approcci basati su dati scheletrici per modellare il movimento umano presentano diversi vantaggi rispetto alle tecniche precedentemente descritte basate su immagini RGB, in quanto i dati sono invarianti rispetto a fattori come illuminazione, colore, texture e sfondo. Questi approcci sfruttano quindi rappresentazioni strutturate per del corpo sotto forma di punti articolari, che riducono le dipendenze visive [39].

Modalità di acquisizione dei dati scheletrici

Le informazioni scheletriche possono essere acquisite attraverso diverse tecnologie, ciascuna con vantaggi e limitazioni:

- **algoritmi di stima della posa su video RGB:** sistemi come OpenPose e MediaPipe elaborano le immagini per estrarre *keypoints* delle articolazioni principali, fornendo una stima accurata della postura [40]. Questi metodi *markerless* rappresentano un'opzione accessibile e flessibile, ma possono soffrire di imprecisioni dovute a occlusioni o pose complesse,

- **sensori di profondità:** dispositivi come il Microsoft Kinect combinano informazioni RGB e mappe di profondità per stimare la posizione 3D delle articolazioni [41]. Sebbene offrano una maggiore robustezza rispetto ai metodi

esclusivamente basati su RGB, la loro accuratezza può essere influenzata da condizioni ambientali e dalla qualità del sensore,

- **sistemi di motion capture (MoCap):** questi sistemi rappresentano la soluzione più precisa per il tracciamento del movimento umano, suddividendosi in tre categorie principali:
  - **Sistemi ottici:** dispositivi come Vicon e OptiTrack utilizzano telecamere ad alta velocità e *marker* riflettenti per acquisire dati con elevata precisione [42]. Tuttavia, il loro utilizzo è limitato ad ambienti strutturati e richiede attrezzature costose.
  - **Sistemi inerziali:** basati su sensori IMU *(Inertial Measurement Unit)* e magnetometri, questi sistemi offrono una maggiore portabilità e possono essere utilizzati in scenari non strutturati. Strumenti come i guanti MANUS Prime 3 XR permettono di tracciare con precisione il movimento delle mani senza necessità di *marker* visibili [43]. Sebbene più flessibili, soffrono di problemi di *drift* accumulato nel tempo.
  - **Sistemi markerless basati su visione:** OpenPose, Kinect e altri sistemi sfruttano algoritmi di *deep learning* per stimare la posa direttamente dai video [41]. Questa soluzione è meno invasiva e più economica rispetto ai sistemi ottici o inerziali, ma presenta limitazioni in termini di accuratezza, soprattutto in condizioni di scarsa visibilità o con soggetti parzialmente occlusi.

La scelta del sistema di acquisizione dipende dall'applicazione specifica: mentre i sistemi ottici sono ideali per applicazioni ad alta precisione come l'animazione o la biomeccanica, quelli inerziali sono più adatti a contesti dinamici come la robotica o la realtà virtuale. I metodi *markerless*, invece, rappresentano una soluzione intermedia per scenari meno controllati, come l'analisi del comportamento umano in ambienti naturali.

Modelli e architetture per il riconoscimento delle azioni scheletriche

Il riconoscimento delle azioni basato su dati scheletrici ha subito una notevole evoluzione nel corso degli anni, passando dall'uso di feature ingegnerizzate a modelli di *deep learning* capaci di apprendere automaticamente le caratteristiche spaziali e temporali del movimento umano. Le architetture più utilizzate in questo ambito includono le reti neurali ricorrenti (RNN), le reti convoluzionali (CNN) e le *Graph Neural Networks* (GNN) [44]. Ciascuna di queste architetture offre vantaggi specifici per la modellazione dei dati scheletrici e ha contribuito significativamente ai progressi nel riconoscimento delle azioni.

Le **RNN**, e in particolare le loro varianti avanzate come LSTM e GRU *(Gated Recurrent Unit)*, sono state tra le prime architetture utilizzate per il riconoscimento delle azioni basato su dati scheletrici. Un esempio significativo è rappresentato dall'Hierarchical RNN (H-RNN) di Du et al. [45], che suddivide il corpo umano in cinque parti, elaborando separatamente ciascun sottoinsieme di articolazioni prima di aggregare le informazioni per una comprensione più strutturata dell'azione.

Un altro modello rilevante è il Differential RNN (dRNN) [46], che introduce una funzione per quantificare il cambiamento di stato tra i frame, migliorando la capacità di catturare le variazioni di movimento e risultando particolarmente efficace per riconoscere transizioni dinamiche. Tuttavia, le RNN tradizionali soffrono del problema della scomparsa del gradiente *(vanishing gradient)*, che limita la loro capacità di modellare dipendenze temporali a lungo raggio. Per superare questa limitazione, il Global Context Aware Attention LSTM (GCA-LSTM) [47] integra un meccanismo di attenzione [34], permettendo di focalizzarsi sulle articolazioni più informative e migliorando la robustezza del modello rispetto a occlusioni e rumore nei dati.

Parallelamente, le **reti convoluzionali** sono state adattate per il riconoscimento delle azioni scheletriche trasformando le sequenze di pose in rappresentazioni pseudo-immagine [48, 49]. Questo approccio consente di sfruttare la capacità delle CNN di apprendere pattern spaziali complessi, riducendo la necessità di progettare manualmente feature ingegnerizzate. Tuttavia, le CNN tradizionali non sono ottimali per modellare le relazioni temporali tra frame consecutivi, motivo per cui si è passati a soluzioni alternative come le *Graph Convolutional Networks* (GCN).

Le **GCN** vengono utilizzate nel riconoscimento delle azioni scheletriche per la loro capacità di modellare le relazioni tra articolazioni attraverso una struttura a grafo. Il modello Spatial-Temporal Graph Convolutional Network (ST-GCN) [39] introduce una rappresentazione dei dati scheletrici in cui i nodi rappresentano le articolazioni e i bordi descrivono le connessioni fisiche e temporali tra di esse. Questo approccio consente di modellare le interazioni tra le articolazioni in modo più efficace rispetto alle RNN e alle CNN tradizionali [37]. Tuttavia, l'implementazione delle GCN può essere computazionalmente onerosa, specialmente per grafi di grandi dimensioni o in applicazioni in tempo reale.

Un'ulteriore architettura che ha dimostrato notevoli prestazioni nel riconoscimento delle azioni scheletriche è rappresentata dalle **Temporal Convolutional Networks (TCN)** [50]. Le TCN rappresentano un approccio innovativo per il modellamento di sequenze temporali. A differenza delle reti ricorrenti (RNN) e delle LSTM, le TCN sono strutture interamente convoluzionali che operano su sequenze di lunghezza arbitraria, mantenendo il principio della causalità: per ogni istante temporale t, l'output dipende unicamente dagli input fino a tale istante [51].

Un aspetto fondamentale delle TCN è l'impiego di convoluzioni dilatate, le

quali permettono di espandere il campo recettivo senza un incremento esponenziale del numero di parametri. Ad esempio, aumentando progressivamente il fattore di dilatazione (es. 1, 2, 4, ...) lungo i vari layer, la rete è in grado di catturare sia dipendenze a breve termine che informazioni su periodi più lunghi della sequenza [50].

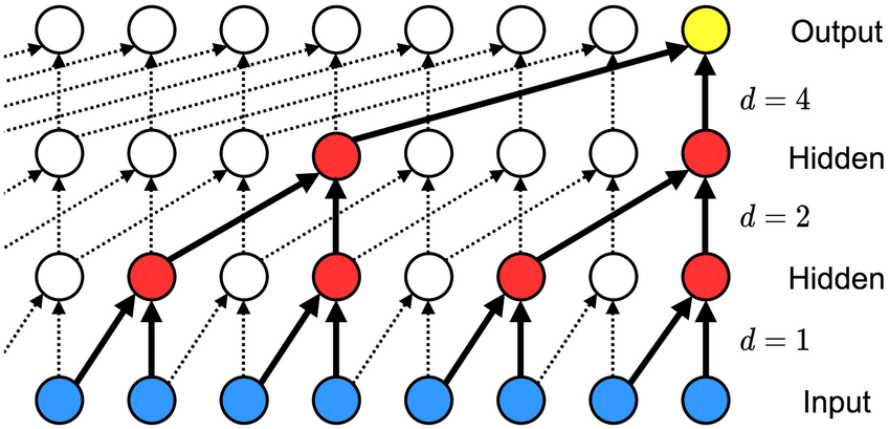

Figura 2.1: Rappresentazione dell'architettura di una TCN con convoluzioni dilatate. I nodi blu rappresentano gli input della sequenza temporale, quelli rossi corrispondono agli stati nascosti nei vari livelli (hidden layers), e il nodo giallo rappresenta l'output finale. Ogni livello applica convoluzioni causali con fattore di dilatazione crescente ($d = 1, 2, 4$), permettendo di ampliare progressivamente il campo recettivo della rete senza aumentare il numero di parametri. Le frecce indicano le connessioni convoluzionali, con la causalità garantita dall'assenza di collegamenti verso il futuro [52].

I principali vantaggi delle TCN rispetto ai modelli basati su RNN/LSTM sono [53]:

- **Causalità**: ogni output è calcolato utilizzando solo i dati passati, garantendo così coerenza nel modello temporale;
- **Parallelismo**: le operazioni convoluzionali possono essere eseguite in parallelo su tutti i dati della sequenza, accelerando l'addestramento e l'inferenza;
- **Stabilità dei gradienti**: la struttura feed-forward elimina i problemi di vanishing/exploding gradient tipici dei modelli ricorrenti.

Questi aspetti hanno portato all'impiego delle TCN in numerose applicazioni di riconoscimento delle azioni, sia in ambito di *human action recognition* sia in contesti specifici come il *hand action recognition*, dove è fondamentale catturare in modo efficiente la dinamica temporale dei movimenti [54].

2.2 Differenze tra il riconoscimento delle azioni del corpo e della mano

Il riconoscimento delle azioni del corpo e della mano presenta differenze significative, dovute a diversi fattori strutturali e metodologici. Mentre esistono numerose soluzioni avanzate per stimare la posa del corpo umano e dataset consolidati per il loro addestramento, il riconoscimento delle azioni delle mani richiede approcci specifici, data la loro complessità articolare e le sfide intrinseche legate alla loro individuazione. Nonostante gli stimatori di posa corporea come OpenPose [40] siano altamente efficaci quando il corpo è completamente visibile, la loro precisione nel riconoscere le mani risulta spesso insufficiente. Studi come 100DOH [6] dimostrano che la stima delle mani basata esclusivamente sulla posa corporea porta a imprecisioni, rendendo necessario l'uso di rilevatori o dataset dedicati.

2.2.1 Motivazioni delle differenze tra il riconoscimento del corpo e delle mani

Le mani presentano una struttura altamente articolata, con una gamma di movimenti complessi e direzioni arbitrarie. A differenza delle articolazioni del corpo, che si muovono secondo schemi relativamente prevedibili, le mani possono assumere posizioni molto varie senza cambiamenti significativi nelle loro articolazioni [27]. Inoltre, il rilevamento delle mani è complicato dalla loro dimensione ridotta rispetto al corpo, dal rischio di occlusioni e dalla loro elevata variabilità in orientazione e forma [55].

Gli approcci tradizionali al riconoscimento delle mani si dividono principalmente in due strategie:

- **trattare le mani come una classe generica di oggetti**, addestrando rilevatori basati sull'aspetto visivo, utilizzando framework come DPM [56] e Mask R-CNN [57]. Questo approccio, sebbene efficace, soffre delle limitazioni comuni ai metodi puramente visivi, come la sensibilità alle condizioni di illuminazione e l'occlusione parziale.

- **considerare le mani come parte del corpo umano**, determinandone la posizione in base alla stima della posa corporea. Sebbene questa strategia sfrutti modelli e dataset già consolidati per la posa del corpo, presenta limiti in scenari in cui le mani non sono chiaramente visibili, come nelle riprese in prima persona [40].

2.2.2 Dataset per il riconoscimento delle mani

Un'ulteriore sfida nel riconoscimento delle azioni delle mani è la scarsità di dataset dedicati (Sezione 1.1.3). Mentre esistono ampi dataset per il riconoscimento della posa corporea, i dataset specifici per il riconoscimento delle mani sono limitati, sia in numero che in varietà delle azioni incluse. L'assenza di dataset standardizzati e di larga scala per il riconoscimento delle mani ha reso difficile lo sviluppo di modelli robusti in grado di generalizzare su diversi scenari [40, 15].

Capitolo 3

Materiali e Metodi

3.1 Descrizione della pipeline

Come illustrato in precedenza, l'obiettivo principale di questa tesi è sviluppare un sistema in grado di riconoscere azioni compiute da un operatore umano e acquisire tutte le informazioni necessarie affinché un robot possa replicarle accuratamente. In particolare, le informazioni considerate fondamentali sono:

- **azione**
- **oggetto coinvolto**
- **coordinate spaziali iniziali e finali**

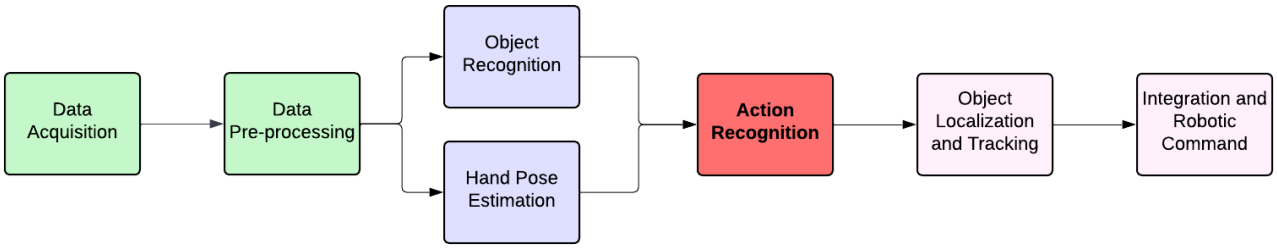

Figura 3.1: Schema della pipeline progettata per il riconoscimento delle azioni umane e la successiva replica robotica. Il processo si articola in sei moduli principali: acquisizione dei dati sensorizzati, preprocessing, stima della posa della mano e riconoscimento dell'oggetto, riconoscimento dell'azione, localizzazione e tracciamento degli oggetti, ed infine comunicazione con il robot.

Per raggiungere questo obiettivo è stata definita una pipeline sequenziale, composta da diversi moduli integrati tra loro. La figura 3.1 riassume schematicamente il flusso operativo. Nei paragrafi successivi si fornisce una breve panoramica di ciascun modulo della pipeline, mentre una trattazione più dettagliata verrà presentata nelle sezioni successive. I primi moduli riguardano l'acquisizione e il preprocessing dei dati. Una delle prime scelte progettuali è stata quella relativa alla tipologia di dati da acquisire. Come evidenziato nell'introduzione (si veda Sezione 1.1.3), i metodi tradizionali basati sulla visione artificiale tramite immagini RGB o RGB-D presentano diversi limiti, quali occlusioni e dipendenza dalle condizioni ambientali. Per superare queste problematiche si è optato per l'utilizzo dei guanti aptici MANUS Prime 3 XR, che, integrati con il sistema HTC Vive (Base Station e tracker Vive 3.0), consentono di acquisire con precisione le coordinate spaziali globali delle articolazioni della mano, garantendo così informazioni affidabili sulla posa e sui movimenti compiuti durante l'esecuzione delle azioni.

I moduli successivi della pipeline si focalizzano principalmente sulle informazioni relative alla mano e all'oggetto manipolato, che rappresentano gli elementi centrali del sistema proposto.

Per quanto riguarda la stima della posa della mano, i dati scheletrici acquisiti tramite i guanti sensorizzati costituiscono direttamente l'input per il modulo di riconoscimento delle azioni. Quest'ultimo è basato su un modello derivato da un framework che utilizza esclusivamente i dati scheletrici della mano per classificare correttamente le azioni eseguite dall'operatore.

Avendo concentrato l'attenzione sulle informazioni dettagliate della mano, il riconoscimento dell'oggetto manipolato e la sua localizzazione risultano più semplici e diretti. Infatti, poiché le coordinate degli oggetti sono note a priori e mappate nell'ambiente operativo, il sistema identifica l'oggetto coinvolto attraverso un algoritmo basato sulla distanza spaziale minima tra la mano dell'operatore e gli oggetti disponibili nell'area di lavoro.

Infine, la pipeline termina con il modulo di comunicazione verso il robot collaborativo. Questo modulo gestisce la formattazione e l'invio dei dati elaborati (tipo di azione, oggetto coinvolto, e coordinate spaziali di inizio e di fine) al braccio robotico, fornendo tutte le informazioni necessarie per una precisa e affidabile replicazione dell'azione compiuta dall'operatore umano.

3.2 Riconoscimento dell'azione

La fase di riconoscimento dell'azione costituisce il nucleo centrale della pipeline proposta. La scelta del metodo è ricaduta su una soluzione basata esclusivamente su dati scheletrici, al fine di garantire una facile adattabilità al nuovo tipo di dati forniti dai guanti sensorizzati. Dopo un'analisi e una valutazione approfondita di

diverse soluzioni (si veda Sezione 2.1.2), la scelta finale è ricaduta sul framework *Domain and View-point Agnostic Hand Action Recognition* proposto da Sabater et al. [1], che introduce un innovativo modello di rappresentazione dei movimenti della mano basato su dati scheletrici, indipendente dal dominio applicativo e dalla prospettiva delle telecamere.

3.2.1 Modello

Il sistema proposto si suddivide in tre fasi principali:

- modellazione della posa della mano
- rappresentazione del movimento
- classificazione dell'azione

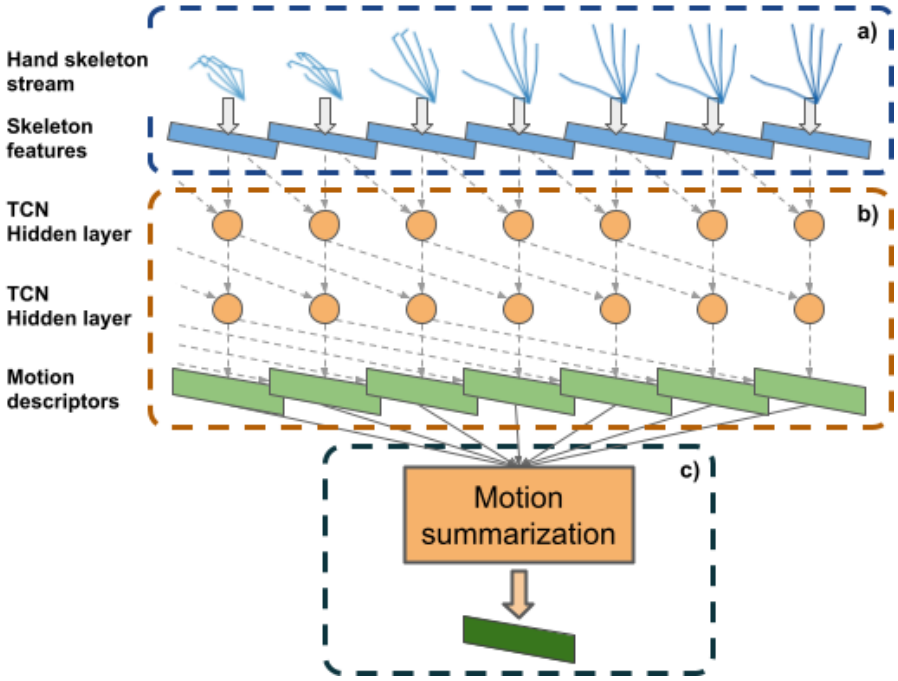

Figura 3.2: Rappresentazione del sistema di Sabater et al.[1], articolato in tre fasi: estrazione delle feature scheletriche, elaborazione temporale tramite TCN e sintesi del movimento attraverso un modulo di motion summarization.

Le sequenze di movimento della mano umana vengono descritte tramite una successione di pose scheletriche. Ogni scheletro è definito dalle coordinate spaziali dei suoi giunti nello spazio euclideo, e ciascun frame temporale presenta una nuova

configurazione dello scheletro della mano. Nella prima fase, vengono applicate alcune tecniche di *preprocessing* volte a migliorare la robustezza e la precisione della rappresentazione, che saranno approfondite nelle sezioni successive.

La seconda fase costituisce il nucleo centrale del modello proposto e utilizza una TCN per generare descrittori di movimento per ciascun frame. La rete TCN elabora le pose in sequenza considerando esclusivamente le informazioni passate e non quelle future, garantendo quindi la causalità temporale nella rappresentazione del movimento [51]. Di conseguenza, ogni descrittore risultante integra informazioni provenienti da tutti i frame precedenti fino a quel punto della sequenza.

Generalmente, l'ultimo descrittore prodotto può essere considerato il più rappresentativo, poiché incorpora tutte le informazioni disponibili sino a quel momento [58]. Tuttavia, nel contesto specifico di questo studio, non sempre i frame finali risultano i più rilevanti per la classificazione. Per ovviare a tale limite, è stato introdotto un modulo di sintesi del movimento (*motion summarization module*), che combina i descrittori generati dalla TCN tramite una media ponderata. In particolare, questo modulo assegna maggiore importanza ai descrittori ritenuti più informativi attraverso una rete neurale composta da un layer convoluzionale monodimensionale seguito da un singolo layer completamente connesso con attivazione sigmoidale. La rete viene addestrata end-to-end insieme alla TCN. Una volta appresi i pesi, il modulo calcola una media ponderata dei descrittori, generando così un unico descrittore finale di movimento.

Quest'ultimo descrittore viene utilizzato nella terza e ultima fase della pipeline, ossia la classificazione dell'azione, che può avvenire secondo due differenti approcci:

- **intra-domain**: il modello viene addestrato e valutato nello stesso dominio. In questo scenario, si utilizza un classificatore lineare addestrato end-to-end con una funzione di perdita basata sulla cross entropy categorica (*categorical cross entropy loss*) [59]. L'ottimizzazione di tale funzione permette di ottenere previsioni prossime alle etichette corrette.

- **cross-domain**: in questo approccio, la classificazione si avvale di tecniche di apprendimento contrastivo per creare rappresentazioni invariate rispetto al dominio applicativo. Viene utilizzata la *normalized temperature-scaled cross entropy loss* (NT-Xent) [60], che ottimizza il modello affinché rappresentazioni della stessa categoria risultino vicine nello spazio delle caratteristiche. La classificazione finale avviene tramite un algoritmo K-Nearest Neighbors (KNN) [61]. Quest'ultimo metodo è particolarmente efficace in scenari cross-domain, come evidenziato anche in SimpleShot di Wang et al. [62]. Questa metodologia permette al sistema di riconoscere anche nuove azioni o variazioni che non erano state direttamente osservate durante la fase di addestramento.

Nel nostro sistema il modulo basato su TCN è utilizzato per modellare la dinamica temporale delle sequenze di feature estratte dai dati acquisiti dai guanti

MANUS. Come descritto nella Sezione 2.1.2, le TCN sfruttano convoluzioni 1D causali e dilatate per ottenere un ampio campo recettivo, integrando in modo efficace informazioni a breve e lungo termine.

In particolare, le sequenze di dati, rappresentate come matrici di dimensione $T \times d$, dove T indica il numero di istanti temporali (ad esempio, il numero di frame) e d rappresenta la dimensionalità del vettore di caratteristiche associato a ciascun istante, vengono elaborate da una TCN configurata con blocchi convoluzionali a dilatazione crescente. L'output della TCN, consistente in un *embedding* temporale, viene successivamente utilizzato dal classificatore per determinare la tipologia di azione eseguita. Questa configurazione consente di ottenere un modello efficiente sia in termini di parallelismo sia di stabilità durante l'addestramento, aspetti critici per l'operatività in tempo reale del sistema.

3.2.2 Dataset

La soluzione proposta da Sabater et al. [1] si avvale principalmente di due dataset distinti per l'addestramento e la valutazione del modello, selezionati per le caratteristiche specifiche che facilitano l'applicazione e la generalizzazione del metodo sviluppato.

F-PHAB

Il primo dataset considerato è il *First-Person Hand Action Benchmark* (F-PHAB), introdotto da Garcia-Hernando et al. nel loro lavoro [26]. F-PHAB è stato concepito specificatamente per analizzare azioni svolte in prima persona, acquisendo sequenze video RGB-D arricchite con annotazioni accurate della posa tridimensionale della mano.

Questo dataset comprende complessivamente 1175 video suddivisi in 45 classi di azioni differenti. Le sequenze sono state registrate da sei soggetti distinti in tre scenari diversi, garantendo una significativa variabilità intra-classe. I circa 26 video per ciascuna classe di azione presentano variazioni in termini di velocità, ampiezza dei movimenti e configurazione della mano, contribuendo così alla robustezza e alla capacità di generalizzazione dei modelli addestrati su questi dati.

Le acquisizioni sono state effettuate mediante camere RGB-D Intel RealSense SR300, posizionate sulla spalla degli operatori, con una frequenza di 30 fps e risoluzione pari a 1920×1080 pixel per il flusso RGB e 640×480 pixel per il flusso di profondità.

Le pose scheletriche della mano sono state rilevate tramite sensori magnetici applicati direttamente sulla mano dei soggetti, generando un modello scheletrico composto da 21 articolazioni spaziali. La durata media delle sequenze varia tra i 2 e i 5 secondi e riguarda azioni di vita quotidiana strettamente connesse a 26

oggetti specifici presenti nell'ambiente. Ad esempio, tra le azioni troviamo: *spargere, raccogliere, mescolare*, tutte associate all'oggetto *cucchiaio*. Infine, tutte le sequenze del dataset sono state etichettate manualmente per assicurare un'elevata qualità delle annotazioni.

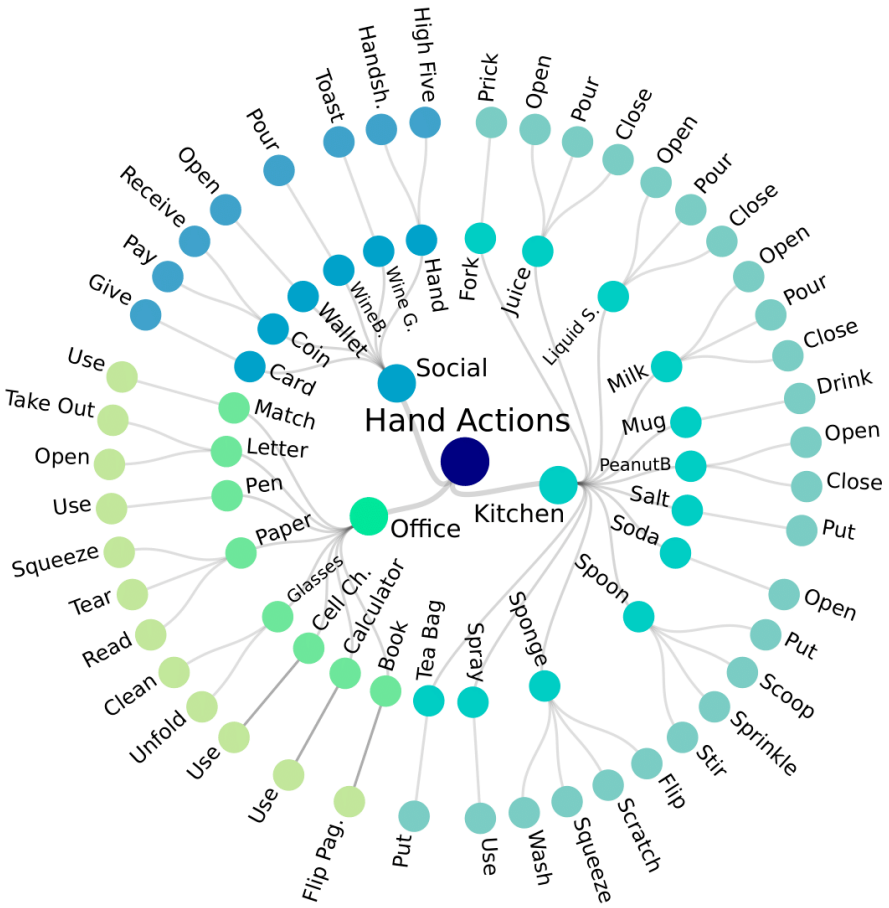

Figura 3.3: Rappresentazione delle azioni e degli oggetti presenti nel dataset F-PHAB [26]. Le azioni manuali sono suddivise in categorie semantiche (es. *Kitchen*, *Office*, *Social*) e collegate agli oggetti coinvolti, riflettendo scenari di interazione quotidiana.

SHREC-17

Il secondo dataset utilizzato è SHREC-17, introdotto da De Smedt et al. nel lavoro [63]. A differenza di F-PHAB, SHREC-17 non si concentra su azioni quotidiane in relazione a specifici oggetti, bensì su gesti elementari eseguiti con la mano. Esso include in totale 2800 sequenze, con una durata media di circa 1 o 2 secondi,

registrate da 28 partecipanti, tutti destrorsi, che eseguono due categorie principali di gesti: 14 realizzati esclusivamente con un singolo dito e 28 coinvolgendo l'intera mano.

Class	Name of the gesture	Type of the gesture
1	Grab	Fine
2	Tap	Coarse
3	Expand	Fine
4	Pinch	Fine
5	Rotation CW	Fine
6	Rotation CCW	Fine
7	Swipe right	Coarse
8	Swipe left	Coarse
9	Swipe up	Coarse
10	Swipe down	Coarse
11	Swipe X	Coarse
12	Swipe V	Coarse
13	Swipe +	Coarse
14	Shake	Coarse

Figura 3.4: Elenco delle 14 azioni del dataset SHREC-17 [63], suddivise in base al tipo di gesto. I gesti sono classificati come *fine* (movimenti precisi e localizzati) o *coarse* (movimenti ampi).

Le pose scheletriche della mano nel dataset SHREC-17 sono composte da 22 articolazioni e sono state ottenute utilizzando una telecamera di profondità Intel RealSense, acquisendo le sequenze a 30 fps con una risoluzione di 640×480 pixel.

I gesti selezionati seguono gli standard riconosciuti nello stato dell'arte e comprendono, ad esempio, movimenti quali *grab*, *swipe* e *rotation*, scelti per la loro rilevanza e diffusione nei sistemi di riconoscimento gestuale [64].

3.3 Setup sperimentale

Le azioni selezionate per il riconoscimento rappresentano operazioni essenziali frequentemente eseguite in scenari collaborativi tra esseri umani e robot, caratterizzati dalla necessità di precisione e affidabilità. Le 8 azioni collaborative raccolte sono: *pick and place* (presa e rilascio di un oggetto), *push* (spinta di un oggetto), *pull* (trascinamento/tiraggio), *upside-down* (capovolgimento di un oggetto),

screw/unscrew (avvitare e svitare, ad es. un tappo), *plug* e *unplug* (inserire e rimuovere un connettore). Tali azioni sono state scelte in quanto rappresentative di compiti tipici in operazioni uomo-robot.

Rispetto ai dataset già utilizzati da Sabater et al.[1], le azioni specifiche richieste dalla nostra applicazione non risultano presenti. Inoltre, tali dataset esistenti presentano limitazioni intrinseche legate all'utilizzo esclusivo di dati visivi RGB o RGB-D, i quali soffrono di problematiche note, quali occlusioni, variabilità nelle condizioni di illuminazione e accuratezza limitata nelle stime della posa (sezione 1.2). Pertanto, è emersa la necessità di creare un nuovo dataset specifico, acquisito direttamente tramite guanti sensorizzati, in grado di fornire dati più precisi e robusti.

3.3.1 Strumentazione

Per la realizzazione del dataset sono stati impiegati i guanti MANUS Prime 3 Haptic XR [1], integrati con tracker HTC Vive 3.0 e telecamere a infrarossi HTC Vive (base station).

Tutta l'attrezzatura è stata configurata e gestita tramite software specifici installati sul sistema di acquisizione: Manus Core 2.2.1 per l'acquisizione e il salvataggio dei dati provenienti dai guanti, e Steam VR 2.8.8 per l'integrazione e il monitoraggio delle telecamere e tracker HTC.

I guanti MANUS registrano le coordinate locali di 25 giunti della mano, come riportato in Tabella 3.1. Le abbreviazioni nella tabella indicano le articolazioni carpometacarpali (CMC), metacarpofalangee (MCP), interfalangee prossimali (PIP), interfalangee distali (DIP) e la punta (TIP). Le lettere iniziali rappresentano il dito corrispondente: T (thumb, pollice), I (indice), M (medio), R (ring, anulare), P (pinky, mignolo).

La Tabella 3.2 riporta le specifiche tecniche principali dei guanti MANUS utilizzati.

Grazie ai tracker HTC applicati ai guanti, queste coordinate locali vengono trasformate e mappate nello spazio globale, consentendo un monitoraggio accurato della posizione e dei movimenti delle mani nell'ambiente.

Le telecamere a infrarossi HTC sono state posizionate strategicamente agli angoli superiori dell'area sperimentale, in modo da garantire un tracciamento ottimale dell'interazione. L'area di lavoro è costituita da una scrivania centrale sulla quale vengono posizionati gli oggetti coinvolti nelle azioni.

[1]MANUS Prime 3 Haptic XR

Tabella 3.1: Divisione dei giunti delle mani registrati dai guanti MANUS.

Mano	**Giunti registrati**
Polso	1 giunto
Pollice	4 giunti: TCMC, TMCP, TDIP, TTIP
Indice	5 giunti: ICMC, IMCP, IPIP, IDIP, ITIP
Medio	5 giunti: MCMC, MMCP, MPIP, MDIP, MTIP
Anulare	5 giunti: RCMC, RMCP, RPIP, RDIP, RTIP
Mignolo	5 giunti: PCMC, PMCP, PPIP, PDIP, PTIP

Tabella 3.2: Specifiche tecniche dei guanti MANUS Prime 3 Haptic XR.

Parametro	**Valore**
Signal latency	$\leq$ 7.5 ms
Sensor sample rate	90 Hz
Battery duration	10 ore (batterie intercambiabili)
Charging time	3 ore (USB Type-C)
Weight	134 grammi
Wired communication	Bluetooth Low Energy 5
Wireless communication	Windows 10/11
Supported operating system	Windows 10/11
Sensors	5 sensori flessibili (2DoF), 6 IMU (9DoF)
Finger flexible sensor repeatability	$>$ 1.000.000 cicli
Haptic feedback	Vibrotattile integrato
Textile information	77% poliestere, 23% spandex

3.3.2 Metodologia

Per garantire la validità e la robustezza del nuovo dataset, è stato definito un protocollo sperimentale ispirato ai criteri adottati dai creatori dei dataset F-PHAB e SHREC-17 [26, 63].

Per ciascuna delle 8 azioni identificate, sono state effettuate circa 80 registrazioni,da un unico operatore, variando in modo sistematico velocità, tipo di presa e oggetti coinvolti (ad es. bottiglia, caricatore, penna, mouse, ...). L'obiettivo di tale diversificazione era rendere il dataset sufficientemente realistico e rappresentativo delle condizioni operative tipiche. Tutte le azioni sono state eseguite utilizzando esclusivamente la mano destra, con sequenze di durata media compresa tra 2 e 4 secondi.

Le acquisizioni sono state effettuate tramite il software Manus Core, che consente l'esportazione immediata dei dati in formato CSV con un framerate di 30 fps. Il sistema di coordinate utilizzato dal software segue lo standard tipico di Blender, caratterizzato dall'asse Z verticale, l'asse X orizzontale e l'asse Y profondità. Ogni file CSV generato contiene tre colonne iniziali relative al numero del frame e al timestamp, seguite da 75 colonne corrispondenti alle coordinate spaziali (x, y, z) dei 25 giunti descritti in precedenza.

Per semplificare la gestione dei dati raccolti, è stato sviluppato uno script Python dedicato all'organizzazione automatica delle registrazioni in una struttura gerarchica di cartelle basata sulle azioni. Parallelamente, lo script genera un file di testo contenente il percorso relativo di ogni file CSV, accompagnato dalla corrispondente

etichetta numerica, fondamentale per le successive fasi di analisi e addestramento del modello.

Infine, per prevenire eventuali problemi legati ai sensori magnetici dei guanti e garantire la qualità dei dati, ogni 10 registrazioni è stata effettuata una nuova calibrazione dei guanti e dei tracker attraverso l'utilizzo del software Manus Core.

3.4 Implementazione

La componente implementativa della pipeline proposta si basa sul progetto originale di Sabater et al. [1]. Partendo dal loro framework disponibile su GitHub[2], ho realizzato del codice aggiuntivo necessario per adattare il nostro dataset specifico all'input richiesto dalla rete TCN.

La soluzione originale includeva già uno script denominato *data generator*, che permette la selezione dei giunti rilevanti e, in base a una specifica configurazione, genera i vettori di feature da passare al modello TCN. Questa parte di adattamento è stata principalmente sviluppata in linguaggio Python utilizzando l'ambiente Visual Studio Code. Parallelamente, per la gestione e integrazione dei dati in tempo reale, è stato sviluppato del codice in linguaggio C# all'interno dell'ambiente Unity, come approfondito successivamente.

Abbiamo optato per l'approccio cross-domain, quindi per la rete backbone TCN abbiamo utilizzato il modello pre-trainato su un dataset di gesti semplici, SHREC-17 [63], già descritto nella Sezione 3.2.2. In particolare, la rete TCN adottata è strutturata come un *encoder* che processa sequenze di *feature* di dimensione 54, che verrà spiegato in seguito. *L'encoder* è composto da *layer* convolutivi 1D con 256 filtri, *kernel* di dimensione 4, organizzati in 2 stack con *skip connections* abilitate e *padding* impostato in modalità *causal*; quest'ultimo garantisce che, per ogni timestep, l'output dipenda solo dai frame correnti e passati, evitando il *look-ahead bias* e risultando essenziale in applicazioni real-time. Il parametro di dilatazione, inizialmente impostato a `[4]`, viene interpretato come la lista `[1,2,4]`, in modo da espandere il campo ricettivo senza un aumento eccessivo dei parametri. Successivamente, la rete integra un modulo di attenzione che, partendo dagli *embedding* per frame, applica una convoluzione 1D con 64 filtri (*kernel* size pari a 1) seguita da un *layer fully-connected* con 32 unità e attivazione `sigmoid` per generare dei pesi di rilevanza per ciascun frame. Questi pesi sono ulteriormente modulati da un fattore lineare (che dà maggiore importanza ai frame finali) e normalizzati (L1), per poi essere usati nella media pesata degli *embedding*; il vettore risultante viene infine normalizzato in L2 per ottenere un descrittore globale robusto della sequenza. La scelta di questi parametri, come l'uso di convoluzioni dilatate e di una struttura

[2]Domain and view point agnostic hand action recognition

causale, è motivata dalla necessità di catturare efficacemente le dipendenze a lungo termine e di garantire operatività in tempo reale, soprattutto in contesti dove il sistema non può fare affidamento su dati futuri. Inoltre, il modulo di attenzione consente di enfatizzare le parti più informative della sequenza, migliorando la discriminatività dell'*embedding*, mentre il *contrastive learning* (con loss NT-Xent [60] e temperatura 0.07) assicura che la rappresentazione appresa sia robusta anche in scenari cross-domain.

Le prossime sezioni saranno dedicate ai dettagli riguardanti le fasi di preprocessing dei dati, la localizzazione degli oggetti coinvolti nelle azioni e l'integrazione del sistema sviluppato con il robot, inclusa la gestione real-time della pipeline.

3.4.1 Preprocessing

La fase di *preprocessing* rappresenta un passaggio essenziale per uniformare e ottimizzare i dati acquisiti prima della loro elaborazione tramite il modello. Considerando che i dati acquisiti dai guanti presentano una configurazione di 25 giunti, diversa rispetto a quella dei dataset precedentemente utilizzati (si veda Sezione 3.2.2), è stata necessaria una fase preliminare di allineamento. In particolare, i dati originali sono stati adattati a una configurazione standard costituita da 20 giunti selezionati, riportati in Tabella 3.3.

Tabella 3.3: Configurazione standard dei 20 giunti selezionati.

Mano	**Giunti selezionati**
Polso	1 giunto
Pollice	TMCP, TDIP, TTIP
Indice	IMCP, IPIP, IDIP, ITIP
Medio	MMCP, MPIP, MDIP, MTIP
Anulare	RMCP, RPIP, RDIP, RTIP
Mignolo	PMCP, PPIP, PDIP, PTIP

Successivamente, come suggerito da Sabater et al.[1], i dati così ottenuti sono stati ulteriormente ridotti per rappresentare ogni posa della mano tramite un sottogruppo di soli 7 giunti significativi: polso, cima del palmo e le punte delle cinque dita (pollice, indice, medio, anulare e mignolo). Questi punti sono collegati tramite sei ossa virtuali: una per il palmo e una per ciascun dito. Tale rappresentazione più compatta permette di semplificare il modello e migliorare la robustezza rispetto a fenomeni di rumore e variazioni intra-soggetto.

In seguito, il dataset è stato convertito in un *array numpy* e sottoposto ad ulteriori operazioni di normalizzazione. Nello specifico, le coordinate spaziali della

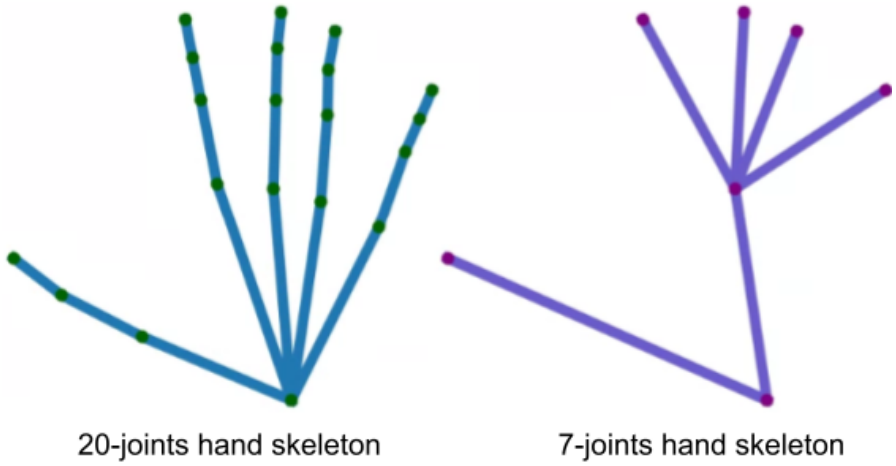

Figura 3.5: Confronto tra la configurazione standard dello scheletro della mano a 20 giunti (a sinistra) e la versione semplificata a 7 giunti (a destra), proposta da Sabater et al.[1]. La rappresentazione ridotta conserva i punti chiave per la descrizione della posa, riducendo la complessità e aumentando la robustezza del modello.

mano sono state normalizzate in base alla distanza tra il polso e la base del dito medio, garantendo così coerenza tra le sequenze e una rappresentazione invariata rispetto alla scala dei dati acquisiti.

Ciascuna sequenza è stata standardizzata a una lunghezza fissa di 32 frame, corrispondenti inizialmente a circa un secondo di registrazione a 30 fps. Tali sequenze sono poi ulteriormente estese a 96 frame reali mediante una funzione implementata da Sabater[1], che prende un frame ogni tre.

Per mitigare il problema del numero limitato di frame nel modello pre-trainato, abbiamo implementato una funzione che, prima della generazione dei vettori, estrae un segmento di 96 frame contenente la parte in cui l'azione è maggiormente evidente, scartando quindi gli inizi e le code del movimento. In pratica, la funzione calcola la differenza tra frame consecutivi e ne computa la norma, ottenendo così una misura dell'intensità del movimento per ciascun frame. Successivamente, si scorre l'intera sequenza con una finestra mobile di 96 frame e si individua quella con la somma cumulativa del movimento più elevata, partendo da quell'indice vengono selezionati i 96 frame successivi.

Il risultato finale del preprocessing è un vettore di caratteristiche con dimensione 54 per ciascun frame, composto da:

- **Coordinate relative della mano** (7 giunti $\times$ 3 assi).
- **Differenza delle coordinate spaziali rispetto al frame precedente** (7 giunti $\times$ 3 assi).

- **Differenze degli angoli delle ossa della mano rispetto al frame precedente** (6 ossa × 2 angoli).

Queste caratteristiche descrivono, rispettivamente, la posa della mano, la direzione e velocità del movimento di ciascun giunto, e la variazione angolare tra le ossa della mano, consentendo una rappresentazione dinamica completa e robusta del movimento [1].

3.4.2 Localizzazione dell'oggetto

Dopo aver trattato la gestione della posa della mano e il riconoscimento delle azioni, questa sezione descrive la metodologia utilizzata per determinare l'oggetto coinvolto e le relative coordinate spaziali, informazioni indispensabili per consentire la replica dell'azione da parte del robot (vedi Sezione 3.1).

Per semplificare questa fase, abbiamo assunto che gli oggetti fossero già noti e localizzati a priori nell'ambiente di lavoro. Le coordinate spaziali di ciascun oggetto sono state acquisite utilizzando uno script Python interfacciato con i tracker HTC Vive tramite l'applicazione SteamVR e salvate in un file JSON organizzato a dizionario.

La determinazione dell'oggetto coinvolto nell'azione è stata realizzata mediante un algoritmo basato sulla minima distanza spaziale. Nello specifico, le coordinate medie della mano calcolate sui primi 10 frame dell'azione vengono confrontate con tutte le coordinate degli oggetti presenti nel dizionario JSON. L'oggetto con distanza minima rispetto alla posizione media della mano risulta essere quello selezionato come oggetto coinvolto.

Per identificare con precisione il punto di interazione tra mano e oggetto, abbiamo integrato ulteriori informazioni nel file JSON relative al tipo di contatto (polso, polpastrello dell'indice, o centro del palmo). In base a tale indicazione, vengono selezionate le coordinate corrispondenti sulla mano per identificare con precisione il punto di inizio e fine dell'interazione.

In particolare, le coordinate rilevanti vengono convertite dal sistema di coordinate utilizzato da SteamVR al sistema Blender, e viceversa, applicando opportuni fattori di conversione e rotazioni.

Segue un esempio di pseudocodice per chiarire il processo:

Algorithm 1 Algoritmo di localizzazione oggetto e coordinate di interazione

1: Caricare il file JSON contenente le coordinate degli oggetti.
2: Calcolare le coordinate medie della mano sui primi 10 frame dell'azione.
3: Selezionare l'oggetto più vicino tramite minima distanza euclidea.
4: Identificare il tipo di contatto associato all'oggetto selezionato.
5: Calcolare le coordinate iniziali e finali del punto di contatto corrispondente.
6: Applicare conversione di coordinate tra sistemi SteamVR e Blender.
7: Salvare l'oggetto e le coordinate trasformate in un file di output.

Questa strategia permette una rapida e affidabile determinazione dell'oggetto e delle coordinate di interazione, minimizzando l'impiego di complessi algoritmi computazionali.

3.4.3 Integrazione con il robot

In questa sezione viene descritta la fase conclusiva della pipeline, focalizzandosi sulla gestione e comunicazione dei dati elaborati verso il sistema robotico.

Per la comunicazione tra il sistema di elaborazione e il robot è stata adottata una soluzione basata su socket TCP (*Transmission Control Protocol*), preferita rispetto al protocollo UDP (*User Datagram Protocol*) per garantire maggiore affidabilità e un controllo degli errori. Questa scelta deriva dal fatto che il sistema non richiede la trasmissione continua e ad alta velocità dei dati, bensì necessita soprattutto che ogni messaggio giunga a destinazione correttamente e senza perdite.

Dopo la classificazione dell'azione, il sistema genera un messaggio nel seguente formato:

```
azione|oggetto|coordinate_inizio|coordinate_fine|
```

Una volta inviato, il sistema resta in attesa di ricevere una conferma di ricezione dal robot.

La gestione **real-time** del sistema rappresenta un'evoluzione importante rispetto all'approccio iniziale basato esclusivamente sui dati CSV esportati dal software Manus Core. In questo scenario avanzato, si utilizza il plugin MANUS integrato direttamente nell'ambiente Unity, il quale permette l'acquisizione in tempo reale delle coordinate spaziali provenienti dai guanti. Tali dati vengono trasmessi tramite una connessione TCP ad uno script Python responsabile di gestire ciclicamente tutte le operazioni della pipeline, ovvero:

1. Preprocessing dei dati.
2. Localizzazione dell'oggetto.

3. Classificazione dell'azione.

4. Comunicazione finale con il robot.

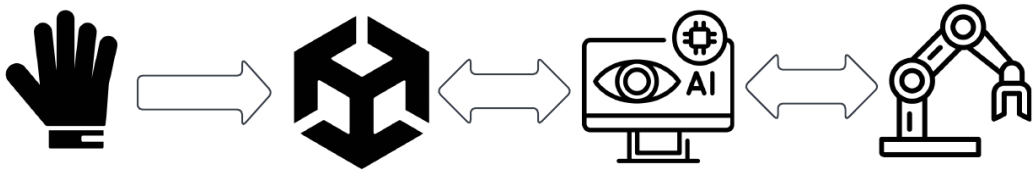

Figura 3.6: Architettura della pipeline real-time. I dati acquisiti dal guanto sensorizzato vengono elaborati in tempo reale tramite il plugin MANUS integrato in Unity, che comunica con uno script Python responsabile delle fasi di preprocessing, riconoscimento dell'azione, localizzazione dell'oggetto e invio del comando al robot. Ogni componente è connessa tramite una comunicazione bidirezionale basata su socket TCP.

Il sistema è composto da due componenti principali: il *Client Unity* e il *Server Python*, che comunicano tramite una connessione TCP. Il client raccoglie i dati della mano utilizzando il guanto sensorizzato e invia i dati al server per l'elaborazione. Il server gestisce la pipeline di riconoscimento delle azioni e trasmette il risultato al robot.

Il client è implementato in C# all'interno di Unity e si occupa di acquisire i dati della mano e inviarli al server. Quando l'utente preme il tasto 'S':

- Se la registrazione non è attiva, viene avviata e viene inviato un comando "START" al server.

- Se la registrazione è già in corso, questa viene interrotta e i dati raccolti vengono inviati per l'elaborazione.

Algorithm 2 Client Unity: flusso semplificato

```
ConnettiAlServer()
while app_in_esecuzione do
    if tasto 'S' premuto then
        InviaComando("START"/"STOP")
        Avvia/InterrompiRegistrazione()
        if comando == "STOP" then
            AvviaThreadRisposta()
        end if
    end if
    if registrazione attiva then
        InviaDatiFrame()
    end if
end while
ChiudiConnessione()
```

Per i dettagli implementativi, si veda l'Appendice A.1.

Algorithm 3 Server Python: flusso semplificato

```
AvviaServer()
while server attivo do
    connessione ← AttendiConnessione()
    while connessione attiva do
        comando ← RiceviComando()
        if comando == "START" then
            CreaFileTemporaneo()
            while ricezione attiva do
                RiceviEDiscrivi()
                if comando == "STOP" then
                    EseguiPipelineML()
                    InviaRisultato()
                    break
                end if
            end while
        else
            GestisciErrore()
        end if
    end while
end while
ChiudiServer()
```

Per i dettagli del codice, si veda l'Appendice A.2.

Capitolo 4

Esperimenti e Risultati

In questo capitolo vengono presentati gli esperimenti condotti per valutare le prestazioni del sistema di riconoscimento delle azioni, insieme ai risultati ottenuti. Vengono illustrate le metriche utilizzate per l'analisi delle performance e il confronto con i risultati raggiunti dal modello sui dataset descritti nella Sezione 3.2.2.

Gli esperimenti si sono articolati su più livelli, partendo dalla modifica di singoli moduli del modello fino all'applicazione di tecniche di data augmentation, con l'obiettivo di migliorare la robustezza e la capacità di generalizzazione del sistema. In aggiunta, è stata svolta un'analisi qualitativa sullo spazio degli *embeddings* per interpretare meglio il comportamento del modello e la separabilità tra classi.

4.1 Metriche e Valutazioni

Per la valutazione delle prestazioni del modello nella classificazione delle azioni, sono state adottate le seguenti metriche standard:

- **Accuratezza (Accuracy)**: rapporto tra il numero di predizioni corrette e il numero totale di campioni.
- **Precisione (Precision)**: rapporto tra le predizioni corrette di una classe e il totale delle predizioni effettuate per quella classe.
- **Recall**: rapporto tra le predizioni corrette di una classe e il numero effettivo di istanze appartenenti a quella classe.
- **F1-score**: media armonica tra precisione e recall, utile in presenza di classi sbilanciate.

Poiché il problema trattato è di tipo multi-classe e potenzialmente soggetto a squilibri tra le classi, l'utilizzo della sola accuratezza risulterebbe limitativo.

Per questo motivo, le metriche sopra elencate sono state affiancate dalla rappresentazione mediante una matrice di confusione, che consente di visualizzare in maniera dettagliata la distribuzione delle predizioni rispetto alle etichette reali. Nel nostro caso, la classificazione finale viene eseguita attraverso un classificatore KNN (si veda tabella 4.1) applicato nello spazio degli *embeddings*, come approfondito nella Sezione 3.2.1. Tale classificazione si basa principalmente sulle distanze tra rappresentazioni, e la matrice di confusione si è dimostrata uno strumento efficace per identificare classi particolarmente simili tra loro e più soggette a errore.

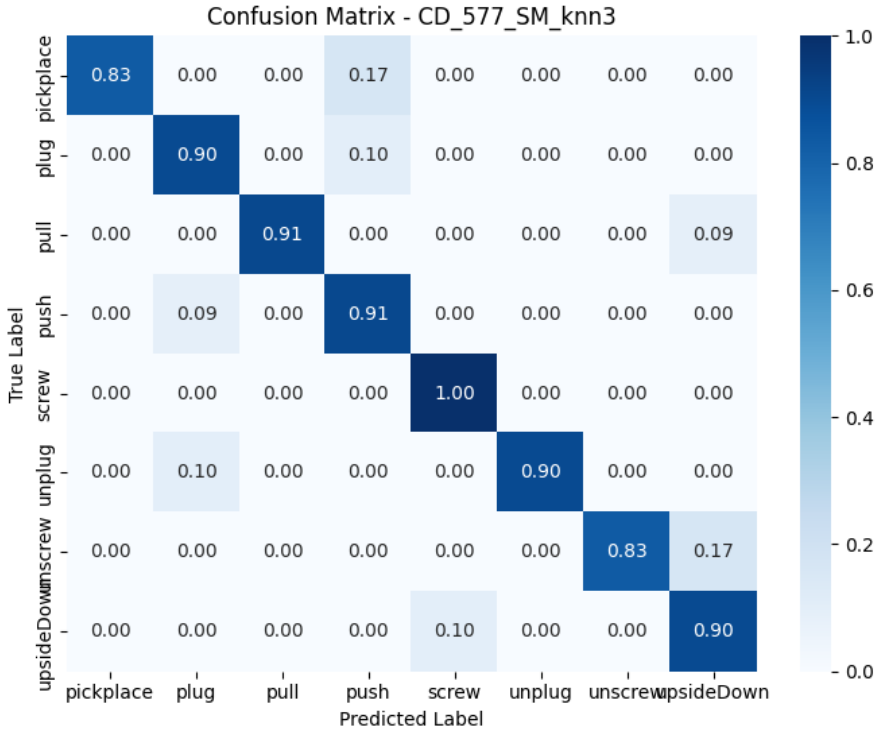

Figura 4.1: Matrice di confusione ottenuta con classificatore KNN ($k = 3$) applicato nello spazio degli *embeddings*. Ogni riga rappresenta le etichette reali, mentre ogni colonna le predizioni del modello.

4.1.1 Data Augmentation

Un secondo aspetto approfondito durante la fase sperimentale è stato l'impatto delle tecniche di *data augmentation* sulle prestazioni del modello. Come descritto nel lavoro originale di Sabater et al.[1], il framework mette a disposizione una

Tabella 4.1: Metriche di valutazione ottenute variando il parametro k del classificatore KNN. Si osserva che i valori migliori in termini di accuratezza, precisione, recall e F1-score sono ottenuti per $k = 1$ e $k = 3$, con un progressivo decadimento delle prestazioni per valori più alti.

KNN Value	Accuracy	Precision	Recall	F1-Score
1	0.89	0.89	0.89	0.89
3	0.88	0.89	0.88	0.88
5	0.85	0.87	0.85	0.85
9	0.82	0.84	0.82	0.81
11	0.80	0.83	0.80	0.79

serie di trasformazioni applicabili ai dati di input per aumentare la variabilità e migliorare la capacità di generalizzazione del modello.

Il *data augmentation* consiste nel generare nuove istanze di dati a partire da quelli esistenti, applicando trasformazioni che riproducono condizioni realistiche. Le tecniche adottate in questo lavoro includono:

- **Aggiunta di rumore sui dati dei sensori**: viene introdotto rumore gaussiano di bassa intensità sui valori delle articolazioni e sulle traiettorie della mano, per simulare imprecisioni nella rilevazione e aumentare la tolleranza del modello al rumore sensoriale.

- **Variazioni temporali**: le sequenze vengono modificate tramite operazioni di stretching o compressione temporale, generando così versioni accelerate o rallentate delle azioni. Questa trasformazione consente di esporre il modello a una maggiore diversità nei tempi di esecuzione.

- **Rotazioni e perturbazioni spaziali**: vengono applicate piccole rotazioni casuali e trasformazioni geometriche all'intera traiettoria della mano nello spazio tridimensionale. In questo modo si riduce la dipendenza del modello dall'orientamento assoluto dell'azione.

Queste trasformazioni hanno lo scopo di testare la robustezza del sistema in condizioni non ideali, simulando variazioni naturali dovute a diverse modalità di esecuzione da parte dell'utente, infatti notiamo un leggero calo delle prestazioni (Figura 4.2).

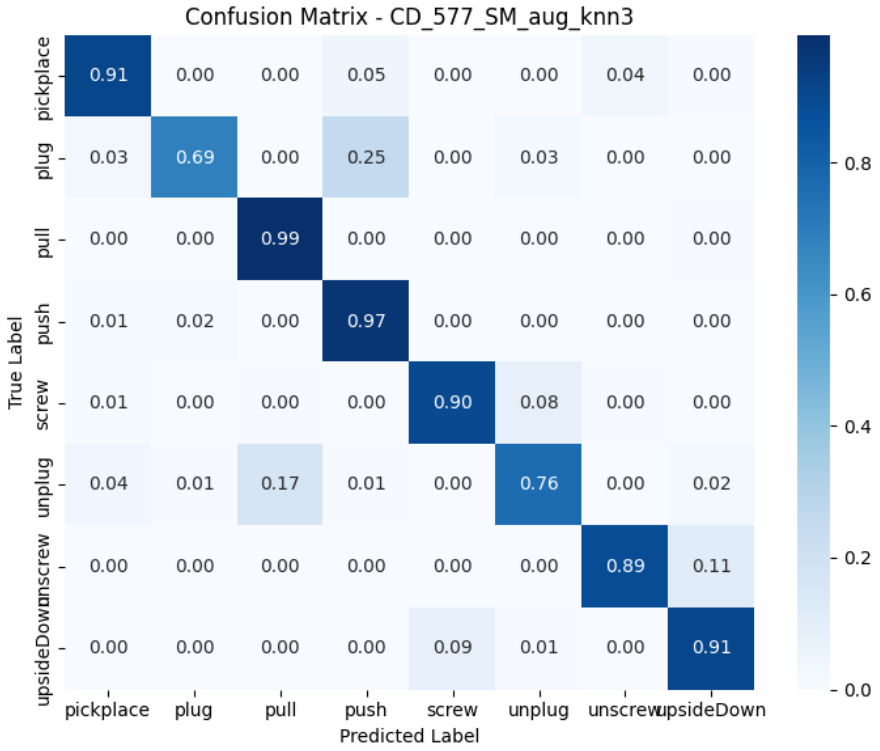

Figura 4.2: Matrice di confusione ottenuta con classificatore KNN ($k = 3$) su dati aumentati. L'aumento dei dati introduce variazioni nel movimento per simulare condizioni reali.

4.1.2 Motion Summarization

Un ulteriore insieme di esperimenti è stato condotto per valutare l'efficacia delle modifiche introdotte nel modulo di sintesi del movimento (*motion summarization module*). Come illustrato nella Sezione 3.2.1, la versione originale del modulo utilizza una media pesata dei descrittori temporali, dove i pesi sono appresi tramite un layer denso di attenzione. Questo meccanismo consente di dare maggiore rilevanza ai frame più informativi, ma trattava la sequenza in modo uniforme dal punto di vista temporale. Nel nostro studio, è stato introdotto un nuovo schema di ponderazione che combina l'attenzione appresa con un peso temporale crescente, il cui scopo è valorizzare maggiormente la parte finale della sequenza. Tale scelta è motivata dalla struttura delle sequenze del nostro dataset, le quali terminano esattamente con la conclusione dell'azione. Di conseguenza, i frame finali risultano particolarmente informativi e rappresentativi (Figure 4.3, 4.4). Dal punto di vista implementativo,

la modifica è stata integrata nella funzione `compute_weighted_average`. Dopo aver ottenuto il vettore dei pesi di attenzione $\mathbf{w}$ tramite il layer denso, è stato generato un vettore di pesi temporali $\mathbf{t}$, crescente linearmente da 0,5 a 1,5 lungo la lunghezza della sequenza. Il vettore risultante $\mathbf{w}'$ viene infine normalizzato affinché la somma totale sia pari a 1, e utilizzato per calcolare la media pesata dei descrittori temporali.

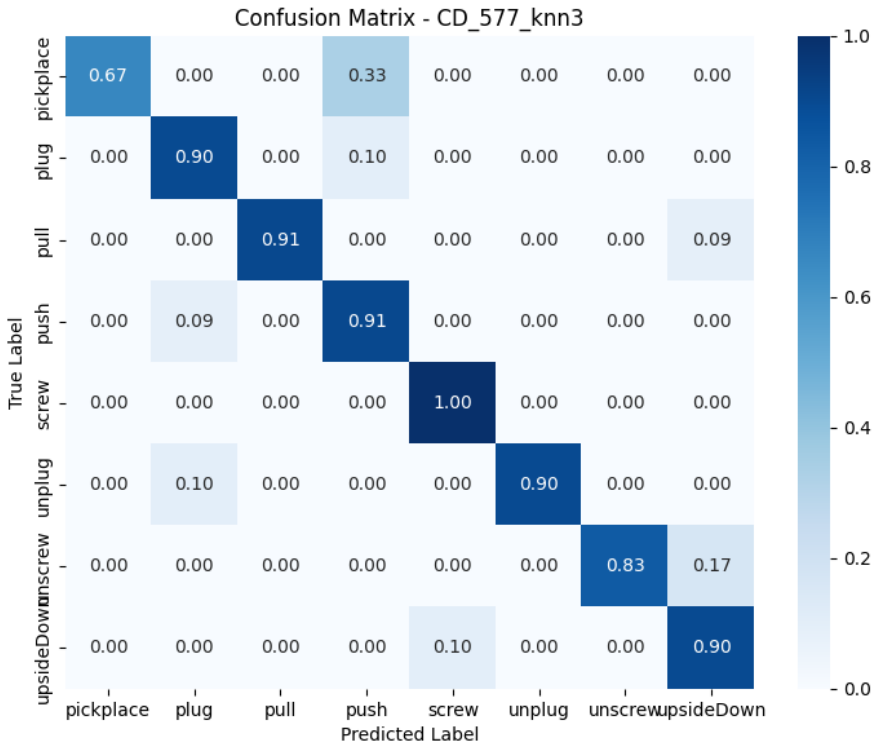

Figura 4.3: Matrice di confusione con classificatore KNN ($k = 3$) utilizzando la versione modificata del modulo di *motion summarization*.

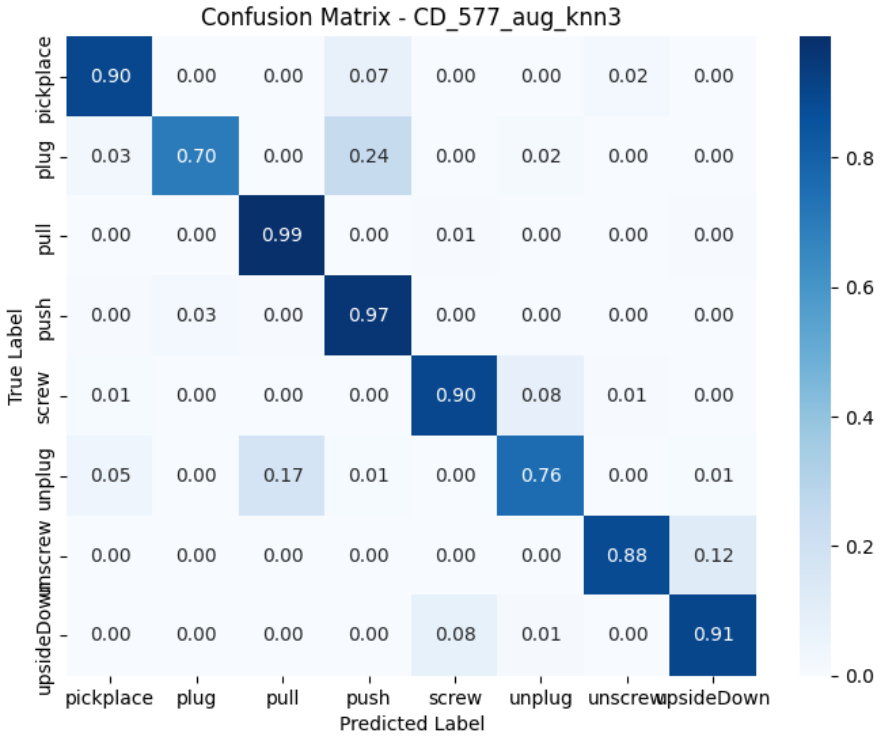

Figura 4.4: Matrice di confusione con classificatore KNN ($k = 3$), modulo di *motion summarization* modificato e dati aumentati.

4.2 Analisi dello spazio degli embeddings

Come introdotto nelle sezioni precedenti, questa parte finale raccoglie le analisi condotte sugli *embeddings* generati dal modello, ovvero le rappresentazioni numeriche compatte che sintetizzano l'informazione contenuta nei dati grezzi. Il modello, infatti, non opera direttamente sulla sequenza originale dei giunti della mano, ma trasforma tali sequenze in uno spazio latente di dimensione più contenuta e semanticamente più significativo. Gli *embeddings* possono essere interpretati come una sorta di "mappa concettuale" interna al modello, nella quale i campioni con caratteristiche simili vengono proiettati in posizioni vicine. Questi vettori vengono generati in spazi ad alta dimensionalità, ma per poterli analizzare e visualizzare è necessario ricorrere a tecniche di riduzione dimensionale che permettano di rappresentarli in due o tre dimensioni. In letteratura sono comunemente utilizzati metodi come t-SNE (*t-distributed Stochastic Neighbor Embedding*) e UMAP (*Uniform Manifold Approximation and Projection*), entrambi impiegati in questa analisi per

esplorare la struttura interna dello spazio latente [65].

L'obiettivo è duplice: da un lato, verificare se gli esempi appartenenti alla stessa classe d'azione risultano raggruppati correttamente, e dall'altro, individuare eventuali strutture inattese o sovrapposizioni tra classi diverse. In particolare, in un contesto di *learning* contrastivo (come nel caso del nostro approccio *cross-domain*), ci si attende che gli *embeddings* si organizzino in base alla categoria dell'azione, indipendentemente dalla provenienza dei dati. Tale comportamento suggerisce che il modello è riuscito ad apprendere rappresentazioni invarianti rispetto al dominio, confermando la generalizzabilità dell'approccio.

Tabella 4.2: Legenda delle Etichette delle azioni

- pickplace
- plug
- pull
- push
- screw
- unplug
- unscrew
- upsideDown

Figura 4.5: Visualizzazione bidimensionale degli *embeddings* generati dal modello tramite la tecnica t-SNE. I punti colorati rappresentano le diverse classi d'azione secondo la legenda in Tabella 4.2.

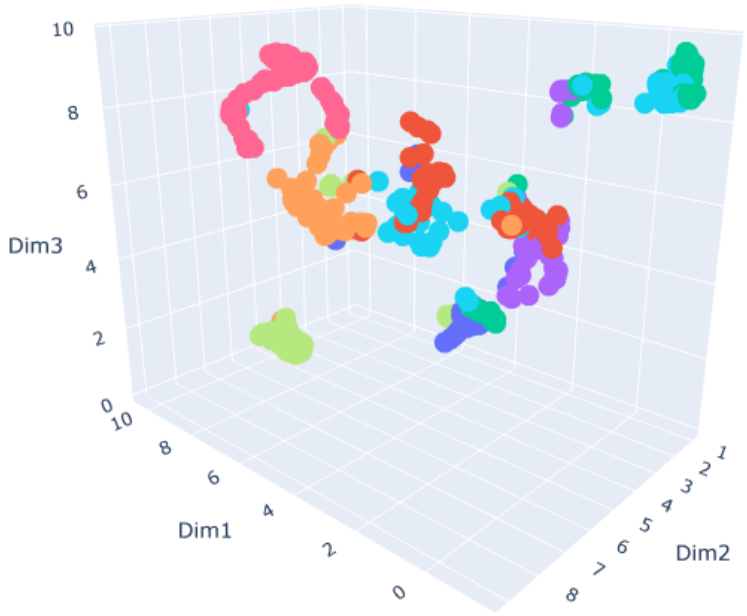

Figura 4.6: Visualizzazione degli *embeddings* tramite riduzione dimensionale con UMAP. I gruppi compatti e ben separati confermano che il modello è in grado di proiettare le azioni in uno spazio semantico coerente, utile per la classificazione.

Capitolo 5

Conclusione

In questa tesi è stato progettato e implementato un sistema per il riconoscimento delle azioni della mano umana, basato sull'elaborazione di dati scheletrici acquisiti tramite dispositivi aptici. Come descritto nel Capitolo 4, il sistema è stato validato su un dataset appositamente realizzato, comprendente otto classi di azioni rappresentative di interazioni mano-oggetto in un contesto collaborativo. I risultati ottenuti sono complessivamente positivi: il modello ha raggiunto un'accuratezza media di circa l'0.848% nella classificazione delle azioni.

Tuttavia, un'analisi più dettagliata delle prestazioni mette in evidenza alcune criticità. In particolare, l'esame delle matrici di confusione e delle metriche per classe ha rivelato che il modello tende a focalizzarsi sulle tipologie di presa utilizzate durante l'esecuzione dell'azione. Questo porta a difficoltà nella distinzione tra azioni che condividono configurazioni della mano o traiettorie simili, ma che differiscono per finalità. È il caso, ad esempio, delle coppie di azioni *plug* vs *push* e *unplug* vs *pull*, che risultano frequentemente confuse dal classificatore.

Questa tendenza è stata ulteriormente confermata dall'analisi qualitativa dello spazio latente generato dal modello. Le visualizzazioni ottenute tramite tecniche di riduzione dimensionale come t-SNE e UMAP hanno mostrato che i cluster associati alle diverse classi non sono completamente separati, indicando che le rappresentazioni interne non sono pienamente discriminative. Ciò evidenzia il ruolo predominante del tipo di presa e del movimento nelle decisioni del sistema.

Nonostante queste limitazioni, i risultati ottenuti sono in linea con le prestazioni riportate in letteratura su dataset standard. Ad esempio, sul dataset SHREC-17, orientato al riconoscimento di gesti statici e dinamici della mano, i migliori approcci hanno ottenuto accuratezze di circa 88.24% per 14 gesti semplici, e 81.9% quando il numero di classi sale a 28 [63]. A confronto, il nostro sistema opera su un numero inferiore di classi, ma con azioni più complesse e semanticamente ricche, spesso legate all'interazione con oggetti. Inoltre, il nostro dataset presenta una

dimensione più contenuta e coinvolge un numero limitato di soggetti, fattori che possono influenzare le percentuali di accuratezza.

Analogamente, confrontando i risultati con quelli ottenuti sul dataset F-PHAB, che include 45 azioni di manipolazione in prima persona, si osserva che modelli sequenziali, come le reti LSTM, raggiungono valori di accuratezza attorno all'80% [26]. Anche in questo caso, va considerata la maggiore complessità del task e la variabilità introdotta da una più ampia popolazione di soggetti e scenari.

Tabella 5.1: Confronto delle accuracy tra metodi cross-domain allenati su SHREC-17.

Metodo su F-PHAB	**1:3**	**1:1**	**3:1**	**Cross-person**
Sabater [1]	70.60	75.50	77.70	58.40
Sabater* (augmented) [1]	76.20	79.70	82.00	62.70
Nostro dataset con guanti aptici				
KNN ($k = 1$)	–	–	–	**89.00**
KNN ($k = 3$)	–	–	–	**88.00**
KNN ($k = 5$)	–	–	–	**85.00**

5.1 Sviluppi Futuri

I risultati presentati confermano la validità dell'approccio adottato, ma evidenziano anche alcune limitazioni che aprono a possibili sviluppi e miglioramenti futuri. In questa sezione vengono delineate alcune direzioni promettenti per proseguire e ampliare il lavoro svolto.

5.1.1 Estensione del dataset

Una delle priorità principali riguarda l'espansione del dataset. Coinvolgere un numero maggiore di soggetti nelle sessioni di registrazione permetterebbe di aumentare la variabilità interpersonale nei movimenti e nelle modalità di esecuzione delle azioni, migliorando così la capacità di generalizzazione del modello.

Parallelamente, un'estensione del repertorio di azioni permetterebbe di aumentare la complessità semantica e motoria del dataset, spingendo il modello ad apprendere rappresentazioni ancora più robuste e discriminative. Un dataset più ampio e bilanciato costituirebbe una base più solida per l'addestramento, contribuendo sia a migliorare l'accuratezza generale che a ridurre l'ambiguità tra classi simili.

5.1.2 Strategie per migliorare la separabilità delle classi

L'ampliamento del dataset potrebbe anche abilitare un addestramento completo del modello a partire da zero, utilizzando esclusivamente dati raccolti tramite i dispositivi aptici, anziché fare affidamento su modelli preaddestrati su dataset di dominio differente.

Questo approccio permetterebbe alla rete di apprendere, fin dalle prime fasi, le caratteristiche specifiche del contesto applicativo: le dinamiche tipiche delle azioni da riconoscere, le variazioni gestuali proprie del task, e le peculiarità dei sensori utilizzati. Un addestramento end-to-end mirato al dominio reale del sistema aumenterebbe potenzialmente la separabilità tra le classi e la precisione complessiva.

5.1.3 Integrazione di dati multimodali

Un'evoluzione interessante consiste nell'integrare ai dati scheletrici della mano altre fonti informative, in particolare dati visivi. L'inclusione di flussi RGB-D (immagini a colori e mappe di profondità) potrebbe fornire un contesto più ricco, aumentando la capacità del sistema di comprendere l'ambiente e gli oggetti coinvolti nell'azione.

Ad esempio, l'identificazione visiva dell'oggetto manipolato (una presa elettrica vs un pulsante) potrebbe aiutare a distinguere azioni altrimenti simili dal punto di vista della sola cinematica della mano (es. *plug* vs *push*). Le immagini RGB offrirebbero informazioni semantiche, mentre le mappe di profondità consentirebbero una stima più precisa delle interazioni mano-oggetto, complementando i dati scheletrici. Questa integrazione multimodale inizia ad essere supportata in letteratura e ha dimostrato di migliorare le prestazioni nei task di *human action recognition* [27].

5.1.4 Nuovi modelli di deep learning e riconoscimento bimanuale

Un'altra possibile evoluzione riguarda l'utilizzo di architetture di deep learning più sofisticate. In particolare, le *Graph Neural Networks* (GNN) si sono dimostrate molto efficaci nella modellazione delle relazioni spaziali e temporali tra giunti in numerosi lavori recenti. Applicare queste architetture al nostro scenario potrebbe portare benefici significativi nella capacità del modello di apprendere strutture di movimento più complesse.

Inoltre, il sistema potrebbe essere esteso al riconoscimento di azioni bimanuali. Molte attività di manipolazione reale richiedono l'uso coordinato di entrambe le mani, e l'acquisizione simultanea dei movimenti di mano destra e sinistra rappresenterebbe un arricchimento importante per la classificazione. Le informazioni aggiuntive derivate dalla simmetria, opposizione o cooperazione tra le due mani potrebbero costituire indizi forti per distinguere azioni complesse [30].

Appendice A

Codice sorgente

Di seguito si riporta il codice completo dei componenti del sistema.

A.1 Client Unity

Il seguente codice, scritto in C#, implementa il client Unity che raccoglie i dati dai guanti e li invia al server.

Codice A.1: Script `HandDataLogger.cs` del client Unity

```
using System;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using UnityEngine;

public class HandDataLogger : MonoBehaviour
{
    public Transform wrist; // Polso
    public Transform[] fingerJoints; // Giunti delle dita
    [SerializeField] private string host = "127.0.0.1";
    [SerializeField] private int port = 12345;

    private TcpClient client;
    private NetworkStream stream;

    private int frameCount = 0;
    private float elapsedTime = 0f;
    private bool isRecording = false;
    private bool isProcessing = false;

    private readonly object syncLock = new object();

```

```
24      private const float TARGET_FRAME_TIME = 1f / 30f;
25      private float timer = 0f;
26
27      private void Start()
28      {
29          ConnectToServer();
30      }
31
32      private void Update()
33      {
34          if (Input.GetKeyDown(KeyCode.S))
35          {
36              lock (syncLock)
37              {
38                  if (isProcessing)
39                  {
40                      Debug.LogWarning("Pipeline già in esecuzione,
    attendere...");
41                      return;
42                  }
43
44                  if (!isRecording)
45                  {
46                      Debug.Log("Registrazione avviata.");
47                      frameCount = 0;
48                      elapsedTime = 0f;
49                      timer = 0f;
50                      isRecording = true;
51                      SendStartStopCommand("START");
52                      SendHeader();
53                  }
54                  else
55                  {
56                      Debug.Log("Registrazione interrotta.");
57                      isRecording = false;
58                      isProcessing = true;
59                      SendStartStopCommand("STOP");
60                      Thread receiveThread = new Thread(ReceiveResponse
    );
61                      receiveThread.Start();
62                  }
63              }
64          }
65
66          if (isRecording)
67          {
68              timer += Time.deltaTime;
69
70              if (timer >= TARGET_FRAME_TIME)
```

```
71              {
72                  timer -= TARGET_FRAME_TIME;
73                  SendHandData();
74              }
75          }
76      }
77
78      private bool IsConnected()
79      {
80          try
81          {
82              return client != null && client.Connected && !(client.
    Client.Poll(1, SelectMode.SelectRead) && client.Client.Available
    == 0);
83          }
84          catch
85          {
86              return false;
87          }
88      }
89
90      private void SendStartStopCommand(string command)
91      {
92          if (!IsConnected())
93          {
94              Debug.LogError("Connessione TCP persa! Impossibile
    inviare comando.");
95              return;
96          }
97
98          byte[] data = Encoding.UTF8.GetBytes(command + "\n");
99          try
100         {
101             stream.Write(data, 0, data.Length);
102             Debug.Log($"Comando {command} inviato al server.");
103         }
104         catch (Exception e)
105         {
106             Debug.LogError($"Errore invio comando {command}: {e.
    Message}");
107         }
108     }
109
110
111     private void SendHeader()
112     {
113         string header = "Frame;Elapsed_Time_In_Milliseconds;Time;
    Hand_X;Hand_Y;Hand_Z;" +
```

```
114                         "Thumb_CMC_X;Thumb_CMC_Y;Thumb_CMC_Z;
    Thumb_MCP_X;Thumb_MCP_Y;Thumb_MCP_Z;" +
115                         "Thumb_DIP_X;Thumb_DIP_Y;Thumb_DIP_Z;
    Thumb_TIP_X;Thumb_TIP_Y;Thumb_TIP_Z;" +
116                         "Index_CMC_X;Index_CMC_Y;Index_CMC_Z;
    Index_MCP_X;Index_MCP_Y;Index_MCP_Z;" +
117                         "Index_PIP_X;Index_PIP_Y;Index_PIP_Z;
    Index_DIP_X;Index_DIP_Y;Index_DIP_Z;" +
118                         "Index_TIP_X;Index_TIP_Y;Index_TIP_Z;
    Middle_CMC_X;Middle_CMC_Y;Middle_CMC_Z;" +
119                         "Middle_MCP_X;Middle_MCP_Y;Middle_MCP_Z;
    Middle_PIP_X;Middle_PIP_Y;Middle_PIP_Z;" +
120                         "Middle_DIP_X;Middle_DIP_Y;Middle_DIP_Z;
    Middle_TIP_X;Middle_TIP_Y;Middle_TIP_Z;" +
121                         "Ring_CMC_X;Ring_CMC_Y;Ring_CMC_Z;Ring_MCP_X;
    Ring_MCP_Y;Ring_MCP_Z;" +
122                         "Ring_PIP_X;Ring_PIP_Y;Ring_PIP_Z;Ring_DIP_X;
    Ring_DIP_Y;Ring_DIP_Z;" +
123                         "Ring_TIP_X;Ring_TIP_Y;Ring_TIP_Z;Pinky_CMC_X
    ;Pinky_CMC_Y;Pinky_CMC_Z;" +
124                         "Pinky_MCP_X;Pinky_MCP_Y;Pinky_MCP_Z;
    Pinky_PIP_X;Pinky_PIP_Y;Pinky_PIP_Z;" +
125                         "Pinky_DIP_X;Pinky_DIP_Y;Pinky_DIP_Z;
    Pinky_TIP_X;Pinky_TIP_Y;Pinky_TIP_Z\n";
126
127             byte[] headerData = Encoding.UTF8.GetBytes(header);
128             try
129             {
130                 stream.Write(headerData, 0, headerData.Length);
131                 Debug.Log("Header inviato al server.");
132             }
133             catch (Exception e)
134             {
135                 Debug.LogError($"Errore invio header: {e.Message}");
136             }
137         }
138
139         private void SendHandData()
140         {
141             if (wrist == null || fingerJoints == null || fingerJoints.
    Length == 0)
142             {
143                 Debug.LogError("Wrist o Finger Joints non impostati!");
144                 return;
145             }
146
147             frameCount++;
148             elapsedTime += TARGET_FRAME_TIME * 1000f;
149
```

```
150         StringBuilder dataBuilder = new StringBuilder();
151         dataBuilder.Append($"{frameCount};{(int)elapsedTime};");
152         TimeSpan timeSpan = TimeSpan.FromMilliseconds(elapsedTime);
153         dataBuilder.Append($"{timeSpan:hh\\:mm\\:ss\\:fff};");
154 
155         Vector3 wristPosition = ConvertCoordinates(wrist.position);
156         dataBuilder.Append($"{-wristPosition.x};{-wristPosition.z};{
    wristPosition.y};");
157 
158         foreach (Transform joint in fingerJoints)
159         {
160             Vector3 jointPosition = ConvertCoordinates(joint.position
    );
161             dataBuilder.Append($"{-jointPosition.x};{-jointPosition.z
    };{jointPosition.y};");
162         }
163 
164         string rowData = dataBuilder.ToString().TrimEnd(';');
165         if (rowData.Split(';').Length != 78) return;
166 
167         byte[] data = Encoding.UTF8.GetBytes(rowData + "\n");
168         try
169         {
170             stream.Write(data, 0, data.Length);
171             Debug.Log($"Frame {frameCount} inviato. Dati: {rowData}");
172         }
173         catch (Exception e)
174         {
175             Debug.LogError($"Errore invio dati: {e.Message}");
176         }
177     }
178 
179     private Vector3 ConvertCoordinates(Vector3 unityCoordinates)
180     {
181         return new Vector3(unityCoordinates.x * 1000f,
182                            unityCoordinates.y * 1000f,
183                            unityCoordinates.z * 1000f);
184     }
185 
186     private void ConnectToServer()
187     {
188         client = new TcpClient(host, port);
189         stream = client.GetStream();
190         Debug.Log("Connesso al server TCP!");
191     }
192 
193     private void ReceiveResponse()
194     {
195         byte[] buffer = new byte[1024];
```

```
196         string completeResponse = "";
197 
198         try
199         {
200             while (true)
201             {
202                 int bytesRead = stream.Read(buffer, 0, buffer.Length)
    ;
203                 if (bytesRead == 0)
204                 {
205                     Debug.LogWarning("Connessione chiusa dal server."
    );
206                     break;
207                 }
208 
209                 string response = Encoding.UTF8.GetString(buffer, 0,
    bytesRead);
210                 completeResponse += response;
211 
212                 // Controlla se hai ricevuto il messaggio finale
213                 if (completeResponse.Contains("Processing complete"))
214                 {
215                     Debug.Log($"Risposta completa ricevuta dal server
    : {completeResponse}");
216                     break;
217                 }
218             }
219         }
220         catch (Exception e)
221         {
222             Debug.LogError($"Errore ricezione risposta: {e.Message}")
    ;
223         }
224         finally //serve a garantire l'esecuzione di un pezzo di
    codice indipendentemente dal fatto che si verifichi o meno un'
    eccezione nel blocco try
225         {
226             lock (syncLock)
227             {
228                 isProcessing = false;
229                 isRecording = false;
230                 Debug.Log("Pipeline pronta per nuova registrazione.")
    ;
231             }
232         }
233     }
234 
235 
236 
```

```
237     private void OnApplicationQuit()
238     {
239         stream.Close();
240         client.Close();
241         Debug.Log("Connessione TCP chiusa.");
242     }
243 }
```

A.2 Server Python

Il seguente codice Python implementa il server che riceve i dati, esegue la pipeline di riconoscimento e trasmette il risultato al robot.

Codice A.2: Script `run_pipeline.py` del server Python

```
1  import socket
2  import subprocess
3  import threading
4  import sys
5  import time
6  import csv
7  import os
8
9  # Configurazione IP e porta del server TCP
10 SERVER_IP = "127.0.0.1"
11 SERVER_PORT = 12345
12 BUFFER_SIZE = 1024
13
14 #testServer
15 ROBOT_IP = "127.0.0.1"
16 ROBOT_PORT = 5000
17
18 # robot e.Do
19 #ROBOT_IP = "192.168.1.58"
20 #ROBOT_PORT = 8585
21
22 # Assicurarsi che la cartella di output esista
23 output_dir = "./results"
24 os.makedirs(output_dir, exist_ok=True)
25
26 # Percorsi assoluti agli script
27 script_dir = os.path.abspath("./dataset_scripts/common_pose")
28 script_myAction = os.path.join(script_dir, "myAction_to_common_pose.
       py")
29 script_det_coordinates = os.path.abspath("./det_coordinates/
       det_coordinates.py")
30 script_action_recognition = os.path.abspath("./
       action_recognition_evaluation.py")
```

```
31
32 # Percorsi aggiuntivi richiesti dall'ultimo step
33 path_model = "./pretrained_models/xdom_summarization"
34 loss_name = "mixknn_best"
35 reference_actions = os.path.abspath("./dataset_scripts/myDataset/
       ref_ComPose_annotations.txt")
36
37 # Flag per controllare il ciclo del server
38 running = True
39
40 # Global variable for robot socket
41 robot_socket = None
42
43 def validate_data_row(row):
44     return len(row.split(";")) == 78
45
46 def handle_client(conn, addr):
47     global running
48     print(f"Connessione accettata da: {addr}")
49
50     while running:
51         data = conn.recv(BUFFER_SIZE).decode("utf-8").strip()
52
53         if not data:
54             print("Connessione chiusa dal client Unity.")
55             break
56
57         if data == "START":
58             print("Inizio registrazione dati")
59             csv_dir = os.path.normpath("./temp_data")
60             os.makedirs(csv_dir, exist_ok=True)
61             csv_filename = os.path.join(csv_dir, f"
       temp_realtime_data_{int(time.time())}.csv")
62             with open(csv_filename, "w", newline="") as f:
63                 csv_writer = csv.writer(f, delimiter=';', quotechar='
       "', quoting=csv.QUOTE_MINIMAL)
64
65                 while True:
66                     data = conn.recv(BUFFER_SIZE)
67                     if not data:
68                         break
69                     data_decoded = data.decode("utf-8").strip()
70                     if data_decoded == "STOP":
71                         print("Fine ricezione dati, avvio pipeline ML
       ")
72                         break
73                     if validate_data_row(data_decoded):
74                         csv_writer.writerow(data_decoded.split(";"))
75                     else:
```

```
76                      print(f"Riga malformata: {data_decoded}")
77
78          # Step 1: Conversione dati
79          print("Eseguendo la conversione in common_pose...")
80          try:
81              subprocess.run(["python", script_myAction, "--
    single_csv", csv_filename], check=True)
82              common_pose_file = os.path.join(
83                  "./datasets/common_pose/testSingleAction",
84                  os.path.basename(csv_filename).replace(".csv", "
    _combined_skeleton.txt")
85              )
86              if not os.path.exists(common_pose_file):
87                  raise FileNotFoundError(f"Errore: il file {
    common_pose_file} non è stato creato.")
88          except Exception as e:
89              print(f"Errore durante la conversione dati: {e}",
    file=sys.stderr)
90              continue
91
92          # Step 2: Calcolo delle coordinate
93          print("Calcolando le coordinate di inizio e fine azione
    ...")
94          try:
95              subprocess.run(["python", script_det_coordinates, "--
    csv_file", csv_filename], check=True)
96              coordinates_file = os.path.join(output_dir, "
    coordinates_obj", "action_coordinates.txt")
97          except Exception as e:
98              print(f"Errore durante il calcolo delle coordinate: {
    e}", file=sys.stderr)
99              continue
100
101         # Step 3: Creazione annotazioni per classificazione
102         annotation_file = os.path.join(output_dir, "
    single_action_annotation.txt")
103         try:
104             with open(annotation_file, "w") as f:
105                 f.write(f"{common_pose_file} 9\n")
106         except Exception as e:
107             print(f"Errore durante la creazione del file di
    annotazione: {e}", file=sys.stderr)
108             continue
109
110         # Step 4: Classificazione azione
111         print("Classificando l'azione...")
112         try:
113             subprocess.run([
114                 "python", script_action_recognition,
```

```
115                     "--path_model", path_model,
116                     "--loss_name", loss_name,
117                     "--single_action", annotation_file,
118                     "--reference_actions", reference_actions,
119                 ], check=True)
120             except Exception as e:
121                 print(f"Errore durante la classificazione dell'azione: {e}", file=sys.stderr)
122                 continue
123 
124             # Step 5: Lettura dei dati e invio al robot
125             print("Preparazione dei dati per l'invio al robot...")
126             action_name = "UNKNOWN"
127             recognized_action_file = os.path.join(output_dir, "recognized_action.txt")
128             while not os.path.exists(recognized_action_file):
129                 time.sleep(0.1)
130             with open(recognized_action_file, "r") as f:
131                 action_name = f.read().strip()
132 
133             with open(coordinates_file, "r") as f:
134                 lines = f.readlines()
135                 if len(lines) < 6:
136                     raise ValueError("Errore: il file action_coordinates.txt non contiene abbastanza righe.")
137                 start_coords = lines[1].strip().split(": ")[1]
138                 end_coords = lines[3].strip().split(": ")[1]
139                 object_name = lines[5].strip()
140 
141             message = f"{action_name}|{object_name}|{start_coords}|{end_coords}|"
142             print(f"Inviando dati al robot: {message}")
143 
144             # Invia messaggio al robot (thread per non chiudere la connessione)
145             robot_thread = threading.Thread(target=send_to_robot, args=(message,conn))
146             robot_thread.start()
147 
148             # # Invia il messaggio al client Unity
149             # conn.sendall((message +"\n").encode('utf-8'))
150             # # Invia una risposta di completamento al client Unity
151             # time.sleep(0.1)
152             # conn.sendall(b"Processing complete")
153             # print("Pipeline completata. Pronta per la prossima azione.")
154         else:
155             print(f"Comando sconosciuto ricevuto: {data}")
156 
```

```
157     #conn.shutdown(socket.SHUT_WR)
158     #conn.close()
159 
160 def listen_for_keypress():
161     global running
162     while running:
163         key = input()
164         if key.lower() == 'k':
165             print('Tasto "k" premuto, chiusura del server...')
166             running = False
167             break
168 
169 def send_to_robot(message, conn):
170     global robot_socket
171     try:
172         robot_socket.sendall(message.encode("utf-8"))
173         response = robot_socket.recv(1024).decode('utf-8')
174         print(f"Risposta robot: {response}")  # response -> messaggio
    ricevuto
175 
176         #FORSE HA SENSO CHE MANDO QUI MESSAGGIO A UNITY ?
177         # Invia il messaggio al client Unity
178         conn.sendall((message +"\n").encode('utf-8'))
179         # Invia una risposta di completamento al client Unity
180         time.sleep(0.1)
181         conn.sendall(b"Processing complete")
182         print("Pipeline completata. Pronta per la prossima azione.")
183 
184     except Exception as e:
185         print(f"Errore durante l'invio dei dati al robot: {e}", file=
    sys.stderr)
186 
187 def handle_robot_communication():
188     global robot_socket
189     robot_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
190     robot_socket.connect((ROBOT_IP, ROBOT_PORT))
191     print("Connessione con il robot stabilita.")
192 
193 def main_server():
194     global running
195     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
196     server_socket.bind((SERVER_IP, SERVER_PORT))
197     server_socket.listen(5) #5 è la lunghezza massima della coda di
    connesioni in attesa
198     print(f"Server TCP in ascolto su {SERVER_IP}:{SERVER_PORT}...")
199 
200     # Avvia un thread per ascoltare l'input della tastiera
201     thread = threading.Thread(target=listen_for_keypress)
202     thread.start()
```

```
203
204     # Avvia un thread per la comunicazione con il robot
205     robot_thread = threading.Thread(target=handle_robot_communication
        )
206     robot_thread.start()
207
208     while running:
209         try:
210             server_socket.settimeout(1.0)
211             conn, addr = server_socket.accept()
212             thread = threading.Thread(target=handle_client, args=(
        conn, addr))
213             thread.start()
214         except socket.timeout:
215             continue
216         except socket.error:
217             if not running:
218                 break
219
220     server_socket.close()
221     robot_socket.close()
222     print("Server chiuso.")
223
224 if __name__ == '__main__':
225     main_server()
```

Bibliografia

[1] Alberto Sabater, Iñigo Alonso, Luis Montesano e Ana C. Murillo. *Domain and View-point Agnostic Hand Action Recognition*. 2021. arXiv: 2103.02303 [cs.CV]. URL: https://arxiv.org/abs/2103.02303 (cit. alle pp. ii, 3, 4, 16, 18, 21, 24–27, 33, 42).

[2] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis e Eftychios Protopapadakis. «Deep Learning for Computer Vision: A Brief Review». In: *Computational Intelligence and Neuroscience* 2018.1 (2018), p. 7068349. DOI: https://doi.org/10.1155/2018/7068349. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2018/7068349. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/7068349 (cit. a p. 1).

[3] Javad Ghofrani, Robert Kirschne, Daniel Rossburg, Dirk Reichelt e Tom Dimter. «Machine Vision in the Context of Robotics: A Systematic Literature Review». In: *CoRR* abs/1905.03708 (2019). arXiv: 1905.03708. URL: http://arxiv.org/abs/1905.03708 (cit. a p. 1).

[4] Nicole Robinson, Brendan Tidd, Dylan Campbell, Dana Kulić e Peter Corke. «Robotic Vision for Human-Robot Interaction and Collaboration: A Survey and Systematic Review». In: *ACM Transactions on Human-Robot Interaction* 12.1 (feb. 2023), pp. 1–66. ISSN: 2573-9522. DOI: 10.1145/3570731. URL: http://dx.doi.org/10.1145/3570731 (cit. a p. 1).

[5] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan e Dieter Fox. *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. 2018. arXiv: 1711.00199 [cs.CV]. URL: https://arxiv.org/abs/1711.00199 (cit. a p. 1).

[6] Dandan Shan, Jiaqi Geng, Michelle Shu e David F. Fouhey. *Understanding Human Hands in Contact at Internet Scale*. 2020. arXiv: 2006.06669 [cs.CV]. URL: https://arxiv.org/abs/2006.06669 (cit. alle pp. 1, 12).

[7] Bruce Yu, Yan Liu, Xiang Zhang, Sheng-hua Zhong e Keith Chan. «MMNet: A Model-based Multimodal Network for Human Action Recognition in RGB-D Videos». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (mag. 2022), pp. 1–1. DOI: 10.1109/TPAMI.2022.3177813 (cit. a p. 2).

[8] Geoffrey Biggs e Bruce Macdonald. «A Survey of Robot Programming Systems». In: (gen. 2004) (cit. a p. 2).

[9] Brenna Argall, Sonia Chernova, Manuela Veloso e Brett Browning. «A survey of robot learning from demonstration». In: *Robotics and Autonomous Systems* 57 (mag. 2009), pp. 469–483. DOI: 10.1016/j.robot.2008.10.024 (cit. a p. 2).

[10] Jens Kober, J. Bagnell e Jan Peters. «Reinforcement Learning in Robotics: A Survey». In: *The International Journal of Robotics Research* 32 (set. 2013), pp. 1238–1274. DOI: 10.1177/0278364913495721 (cit. a p. 2).

[11] Harish chaandar Ravichandar, Athanasios S. Polydoros, Sonia Chernova e Aude G Billard. «Recent Advances in Robot Learning from Demonstration». In: *Annu. Rev. Control. Robotics Auton. Syst.* 3 (2020), pp. 297–330. URL: https://api.semanticscholar.org/CorpusID:208958394 (cit. a p. 2).

[12] Nathan Ratliff, J. Andrew Bagnell e Siddhartha S. Srinivasa. «Imitation learning for locomotion and manipulation». In: *2007 7th IEEE-RAS International Conference on Humanoid Robots*. 2007, pp. 392–397. DOI: 10.1109/ICHR.2007.4813899 (cit. a p. 2).

[13] Josua Spisak, Matthias Kerzel e Stefan Wermter. *Robotic Imitation of Human Actions*. 2024. arXiv: 2401.08381 [cs.RO]. URL: https://arxiv.org/abs/2401.08381 (cit. a p. 2).

[14] Mehrdad Tavassoli, Sunny Katyara, Maria Pozzi, Nikhil Deshpande, Darwin G. Caldwell e Domenico Prattichizzo. *Learning Skills from Demonstrations: A Trend from Motion Primitives to Experience Abstraction*. 2022. arXiv: 2210.08060 [cs.RO]. URL: https://arxiv.org/abs/2210.08060 (cit. a p. 2).

[15] Philip Ezigbo, Onyebuchi Nosiri, Ekene Mbonu, Victor Ofor e Jude Obichere. «Hand Gesture Recognition and Control for Human-Robot Interaction Using Deep Learning». In: *International Journal of Electrical and Electronic Engineering & Telecommunications* 12 (nov. 2023), pp. 433–441. DOI: 10.18178/ijeetc.12.6.433-441 (cit. alle pp. 2, 13).

[16] Ajay Tanwani e Sylvain Calinon. «A generative model for intention recognition and manipulation assistance in teleoperation». In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Set. 2017, pp. 43–50. DOI: 10.1109/IROS.2017.8202136 (cit. a p. 2).

[17] Sonya Allin e Deva Ramanan. «Assessment of Post-Stroke Functioning Using Machine Vision.» In: gen. 2007, pp. 299–302 (cit. a p. 2).

[18] M. Raj e R. Seamans. «Primer on artificial intelligence and robotics». In: *Journal of Organization Design* 8.11 (2019). DOI: 10.1186/s41469-019-0050-0. URL: https://doi.org/10.1186/s41469-019-0050-0 (cit. a p. 2).

[19] Shikhar Bahl, Abhinav Gupta e Deepak Pathak. *Human-to-Robot Imitation in the Wild*. 2022. arXiv: 2207.09450 [cs.RO]. URL: https://arxiv.org/abs/2207.09450 (cit. a p. 2).

[20] Joao Carreira e Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2018. arXiv: 1705.07750 [cs.CV]. URL: https://arxiv.org/abs/1705.07750 (cit. alle pp. 3, 7).

[21] Shaoqing Ren, Kaiming He, Ross Girshick e Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497 (cit. a p. 3).

[22] Yong Du, Wei Wang e Liang Wang. «Hierarchical recurrent neural network for skeleton based action recognition». In: giu. 2015, pp. 1110–1118. DOI: 10.1109/CVPR.2015.7298714 (cit. a p. 3).

[23] Chuhan Zhang, Ankush Gputa e Andrew Zisserman. «Helping Hands: An Object-Aware Ego-Centric Video Recognition Model». In: *International Conference on Computer Vision (ICCV)*. 2023 (cit. a p. 3).

[24] Roei Herzig, Elad Ben-Avraham, Karttikeya Mangalam, Amir Bar, Gal Chechik, Anna Rohrbach, Trevor Darrell e Amir Globerson. *Object-Region Video Transformers*. 2022. arXiv: 2110.06915 [cs.CV]. URL: https://arxiv.org/abs/2110.06915 (cit. a p. 3).

[25] Yuzhe Qin, Yueh-Hua Wu, Shaowei Liu, Hanwen Jiang, Ruihan Yang, Yang Fu e Xiaolong Wang. *DexMV: Imitation Learning for Dexterous Manipulation from Human Videos*. 2022. arXiv: 2108.05877 [cs.LG]. URL: https://arxiv.org/abs/2108.05877 (cit. a p. 3).

[26] Guillermo Garcia-Hernando, Shanxin Yuan, Seungryul Baek e Tae-Kyun Kim. *First-Person Hand Action Benchmark with RGB-D Videos and 3D Hand Pose Annotations*. 2018. arXiv: 1704.02463 [cs.CV]. URL: https://arxiv.org/abs/1704.02463 (cit. alle pp. 3, 4, 18, 19, 23, 42).

[27] Md Salman Shamil, Dibyadip Chatterjee, Fadime Sener, Shugao Ma e Angela Yao. *On the Utility of 3D Hand Poses for Action Recognition*. 2024. arXiv: 2403.09805 [cs.CV]. URL: https://arxiv.org/abs/2403.09805 (cit. alle pp. 3, 12, 43).

[28] Suriya Singh, Chetan Arora e C. V. Jawahar. «First Person Action Recognition Using Deep Learned Descriptors». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2620–2628. DOI: 10.1109/CVPR.2016.287 (cit. a p. 3).

[29] Wa Lu, Yiqiang Chen, Jindong Wang e Xin Qin. *Cross-domain Activity Recognition via Substructural Optimal Transport*. Gen. 2021. DOI: 10.48550/arXiv.2102.03353 (cit. a p. 3).

[30] Taein Kwon, Bugra Tekin, Jan Stuhmer, Federica Bogo e Marc Pollefeys. *H2O: Two Hands Manipulating Objects for First Person Interaction Recognition*. 2021. arXiv: 2104.11181 [cs.CV]. URL: https://arxiv.org/abs/2104.11181 (cit. alle pp. 4, 43).

[31] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani e Manohar Paluri. «Learning Spatiotemporal Features with 3D Convolutional Networks». In: dic. 2015, pp. 4489–4497. DOI: 10.1109/ICCV.2015.510 (cit. alle pp. 5, 7).

[32] Karen Simonyan e Andrew Zisserman. *Two-Stream Convolutional Networks for Action Recognition in Videos*. 2014. arXiv: 1406.2199 [cs.CV]. URL: https://arxiv.org/abs/1406.2199 (cit. alle pp. 5–7).

[33] Lin Sun, Kui Jia, Kevin Chen, Dit Yan Yeung, Bertram E. Shi e Silvio Savarese. *Lattice Long Short-Term Memory for Human Action Recognition*. 2017. arXiv: 1708.03958 [cs.CV]. URL: https://arxiv.org/abs/1708.03958 (cit. alle pp. 5, 7).

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762 (cit. alle pp. 6, 8, 10).

[35] Rohit Girdhar, João Carreira, Carl Doersch e Andrew Zisserman. *Video Action Transformer Network*. 2019. arXiv: 1812.02707 [cs.CV]. URL: https://arxiv.org/abs/1812.02707 (cit. alle pp. 6, 8).

[36] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić e Cordelia Schmid. *ViViT: A Video Vision Transformer*. 2021. arXiv: 2103.15691 [cs.CV]. URL: https://arxiv.org/abs/2103.15691 (cit. alle pp. 6, 8).

[37] Zehua Sun, Qiuhong Ke, Hossein Rahmani, Mohammed Bennamoun, Gang Wang e Jun Liu. «Human Action Recognition From Various Data Modalities: A Review». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), pp. 1–20. ISSN: 1939-3539. DOI: 10.1109/tpami.2022.3183112. URL: http://dx.doi.org/10.1109/TPAMI.2022.3183112 (cit. alle pp. 6, 10).

[38] Zhiheng Huang, Wei Xu e Kai Yu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. 2015. arXiv: 1508.01991 [cs.CL]. URL: https://arxiv.org/abs/1508.01991 (cit. a p. 7).

[39] Sijie Yan, Yuanjun Xiong e Dahua Lin. *Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition*. 2018. arXiv: 1801.07455 [cs.CV]. URL: https://arxiv.org/abs/1801.07455 (cit. alle pp. 8, 10).

[40] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei e Yaser Sheikh. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2019. arXiv: 1812.08008 [cs.CV]. URL: https://arxiv.org/abs/1812.08008 (cit. alle pp. 8, 12, 13).

[41] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman e Andrew Blake. «Real-time human pose recognition in parts from single depth images». In: *CVPR 2011*. 2011, pp. 1297–1304. DOI: 10.1109/CVPR.2011.5995316 (cit. alle pp. 8, 9).

[42] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur e Xavier Savatier. «A Study of Vicon System Positioning Performance». In: *Sensors* 17 (lug. 2017), p. 1591. DOI: 10.3390/s17071591 (cit. a p. 9).

[43] Manuel Caeiro Rodriguez, Iván González, Fernando Mikic Fonte e Martín Llamas Nistal. «A Systematic Review of Commercial Smart Gloves: Current Status and Applications». In: *Sensors* 21 (apr. 2021), p. 2667. DOI: 10.3390/s21082667 (cit. a p. 9).

[44] Lei Shi, Yifan Zhang, Jian Cheng e Hanqing Lu. «Skeleton-Based Action Recognition With Directed Graph Neural Networks». In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 7904–7913. DOI: 10.1109/CVPR.2019.00810 (cit. a p. 9).

[45] Yong Du, Wei Wang e Liang Wang. «Hierarchical recurrent neural network for skeleton based action recognition». In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1110–1118. DOI: 10.1109/CVPR.2015.7298714 (cit. a p. 10).

[46] Vivek Veeriah, Naifan Zhuang e Guo-Jun Qi. *Differential Recurrent Neural Networks for Action Recognition*. 2015. arXiv: 1504.06678 [cs.CV]. URL: https://arxiv.org/abs/1504.06678 (cit. a p. 10).

[47] Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan e Alex C. Kot. «Global Context-Aware Attention LSTM Networks for 3D Action Recognition». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3671–3680. DOI: 10.1109/CVPR.2017.391 (cit. a p. 10).

[48] li Chuankun, Yonghong Hou, Pichao Wang e Wanqing Li. «Joint Distance Maps Based Action Recognition with Convolutional Neural Networks». In: *IEEE Signal Processing Letters* (mar. 2017) (cit. a p. 10).

[49] Carlos Caetano, François Brémond e William Robson Schwartz. *Skeleton Image Representation for 3D Action Recognition based on Tree Structure and Reference Joints*. 2019. arXiv: 1909.05704 [cs.CV]. URL: https://arxiv.org/abs/1909.05704 (cit. a p. 10).

[50] Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter e Gregory D. Hager. «Temporal Convolutional Networks for Action Segmentation and Detection». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 1003–1012. DOI: 10.1109/CVPR.2017.113 (cit. alle pp. 10, 11).

[51] Shaojie Bai, J. Zico Kolter e Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2018. arXiv: 1803.01271 [cs.LG]. URL: https://arxiv.org/abs/1803.01271 (cit. alle pp. 10, 17).

[52] Kookjin Lee, Jaideep Ray e Cosmin Safta. «The predictive skill of convolutional neural networks models for disease forecasting». In: *PLOS ONE* 16 (lug. 2021), e0254319. DOI: 10.1371/journal.pone.0254319 (cit. a p. 11).

[53] Mihai Nan, Mihai Trăscău, Adina Magda Florea e Cezar Cătălin Iacob. «Comparison between Recurrent Networks and Temporal Convolutional Networks Approaches for Skeleton-Based Action Recognition». In: *Sensors* 21.6 (2021). ISSN: 1424-8220. DOI: 10.3390/s21062051. URL: https://www.mdpi.com/1424-8220/21/6/2051 (cit. a p. 11).

[54] Jingxuan Hou, Guijin Wang, Xinghao Chen, Jing-Hao Xue, Rui Zhu e Huazhong Yang. «Spatial-Temporal Attention Res-TCN for Skeleton-Based Dynamic Hand Gesture Recognition». In: *Computer Vision – ECCV 2018 Workshops*. A cura di Laura Leal-Taixé e Stefan Roth. Cham: Springer International Publishing, 2019, pp. 273–286. ISBN: 978-3-030-11024-6 (cit. a p. 11).

[55] Supreeth Narasimhaswamy, Zhengwei Wei, Yang Wang, Justin Zhang e Minh Hoai. *Contextual Attention for Hand Detection in the Wild*. 2019. arXiv: 1904.04882 [cs.CV]. URL: https://arxiv.org/abs/1904.04882 (cit. a p. 12).

[56] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester e Deva Ramanan. «Object Detection with Discriminatively Trained Part-Based Models». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167 (cit. a p. 12).

[57] Kaiming He, Georgia Gkioxari, Piotr Dollár e Ross Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870 [cs.CV]. URL: https://arxiv.org/abs/1703.06870 (cit. a p. 12).

[58] Alberto Sabater, Laura Santos, Jose Santos-Victor, Alexandre Bernardino, Luis Montesano e Ana C. Murillo. *One-shot action recognition in challenging therapy scenarios*. 2021. arXiv: 2102.08997 [cs.CV]. URL: https://arxiv.org/abs/2102.08997 (cit. a p. 17).

[59] Zhilu Zhang e Mert R. Sabuncu. *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels*. 2018. arXiv: 1805.07836 [cs.LG]. URL: https://arxiv.org/abs/1805.07836 (cit. a p. 17).

[60] Ting Chen, Simon Kornblith, Mohammad Norouzi e Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG]. URL: https://arxiv.org/abs/2002.05709 (cit. alle pp. 17, 25).

[61] Pádraig Cunningham e Sarah Jane Delany. «k-Nearest Neighbour Classifiers - A Tutorial». In: *ACM Computing Surveys* 54.6 (lug. 2021), pp. 1–25. ISSN: 1557-7341. DOI: 10.1145/3459665. URL: http://dx.doi.org/10.1145/3459665 (cit. a p. 17).

[62] Yan Wang, Wei-Lun Chao, Kilian Q. Weinberger e Laurens van der Maaten. *SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning*. 2019. arXiv: 1911.04623 [cs.CV]. URL: https://arxiv.org/abs/1911.04623 (cit. a p. 17).

[63] Quentin de Smedt, Hazem Wannous, Jean-Philippe Vandeborre, Joris Guerry, Bertrand Le Saux e David Filliat. «SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset». In: *3DOR - 10th Eurographics Workshop on 3D Object Retrieval*. A cura di I. Pratikakis, F. Dupont e M. Ovsjanikov. Lyon, France, apr. 2017, pp. 1–6. DOI: 10.2312/3dor.20171049. URL: https://hal.science/hal-01563505 (cit. alle pp. 19, 20, 23, 24, 41).

[64] Eshed Ohn-Bar e Mohan Manubhai Trivedi. «Hand Gesture Recognition in Real Time for Automotive Interfaces: A Multimodal Vision-Based Approach and Evaluations». In: *IEEE Transactions on Intelligent Transportation Systems* 15.6 (2014), pp. 2368–2377. DOI: 10.1109/TITS.2014.2337331 (cit. a p. 20).

[65] Sebastian Damrich, Jan Niklas Böhm, Fred A. Hamprecht e Dmitry Kobak. *From t-SNE to UMAP with contrastive learning*. 2023. arXiv: 2206.01816 [cs.LG]. URL: https://arxiv.org/abs/2206.01816 (cit. a p. 38).