POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Human-Robot Interaction, Learning from Demonstration

Analisi video per la replica di azioni in ambienti di collaborazione uomo-macchina

Politecnico di Torino

Relatore
Prof. SANNA Andrea

Correlatore:
Prof. MANURI Federico

Laureando
ABATIANNI Luca

Dicembre 2024

Indice

Elenco delle figure

Capitolo 1

Introduzione

Nella storia recente della Computer Vision, i lavori che mirano alla creazione di metodologie efficaci per permettere a un agente robotico di replicare un'azione umana solo numerosi, e variano ampiamente in termini di approcci e risultati ottenuti: le pratiche utilizzate, al contempo, sono spesso complesse e tendono ad accumulare numerosi livelli di astrazione, pur dimostrando diffusamente la loro efficacia.

Il seguente lavoro di Tesi si inserisce nell'ampio e crescente ambito della Computer Vision al servizio della robotica, in particolare della manipolazione di oggetti e scene di vita reale a seguito di dimostrazioni eseguite da un operatore umano: gli agenti robotici, in forte crescita come varietà di modelli e capacità lavorative, possono rappresentare un grande passo in avanti verso la collaborazione, quotidiana e attiva, con utenti comuni e lavoratori specializzati, mettendosi a disposizione per l'esecuzione puntuale di mansioni di vario tipo.

L'idea fondamentale che ha caratterizzato tale lavoro di ricerca è la necessità di fornire all'agente robotico, che approccia la scena davanti a sé, una quanto più ampia consapevolezza di ciò che andrà a eseguire: molti approcci che affrontano questa problematica ignorano, o mettono da parte, l'analisi dell'azione eseguita dall'operatore umano, o parimenti il comando testuale impartito all'agente. Le metodologie più diffuse, che andremo ad approfondire nella sezione dedicata allo stato dell'arte, s'impongono di generalizzare quanto più possibile l'approccio alla scena, proponendosi come valide soluzioni "in the wild", ma mancando di caratterizzare a dovere l'input fornito all'agente: ciò si traduce in un buon compromesso per l'esecuzione pratica, ma in una scarsa comprensione di ciò che il manipolatore robotico andrà a eseguire, anche nell'ottica di future implementazioni che si possono affidare a tali informazioni.

1.1 Obiettivi

Questo lavoro di ricerca e implementazione si pone come obiettivo principale il raggiungimento della massima comprensione dell'azione eseguita dall'agente umano in una singola dimostrazione della stessa, per poi tradurre sul campo questa pletora di informazioni in una efficace manipolazione della scena. La necessità principale, che ha accompagnato il lavoro in tutte le sue fasi, non è stata quella di predisporre il sistema ad affrontare sfide nuove e mai viste, o a diversificare con massima precisione le scene affrontabili dall'agente robotico, bensì si è concentrata su un'esaustiva analisi dell'informazione giunta in input al sistema, assieme a quella dell'ambiente di lavoro del robot, per ottenere un ampio spettro di informazioni utili allo stesso per eseguire il compito desiderato.

In tal senso, e come verrà approfondito più avanti, si è cercato di minimizzare le difficoltà introdotte da ciò che circonda l'agente lavorando con un ambiente ben definito e circoscritto, con oggetti altresì semplici e con un set di azioni in numero limitato, al fine di concentrarsi sull'efficace estrapolazione e comprensione delle informazioni fondamentali, piuttosto che sull'adattabilità del sistema.

Come verrà analizzato nel capitolo dello stato dell'arte, questo lavoro tenta di diversificare l'approccio classico alla manipolazione robotica basata sulla ripetizione di azioni, che si è spesso concentrato sull'apprendimento da parte del robot stesso utilizzando complessi algoritmi di Machine Learning, per focalizzarsi sull'idea di analizzare a fondo le dimostrazioni ricevute in input, raccogliere da esse la maggior quantità di informazioni possibili, e infine comunicare tali dati all'agente per permettergli di eseguire l'azione in maniera lineare e precisa.

1.2 Organizzazione del documento

Nel capitolo successivo (Cap. 2) affronteremo alcuni aspetti introduttivi propri dell'ambito di ricerca, assieme all'analisi dello stato dell'arte, al fine di comprendere al meglio il punto di partenza del lavoro. Seguiranno i capitoli dedicati alla descrizione dell'architettura di lavoro (Cap. 3) e dei risultati ottenuti (Cap. 4), che valuteranno l'efficacia dell'approccio seguito. Infine, verrà analizzato il lavoro svolto e gli spunti futuri che suggerisce (Cap. 5).

Capitolo 2

Stato dell'arte

2.1 Aspetti introduttivi

Analizziamo ora alcuni concetti e aspetti introduttivi dell'ambito di ricerca, la cui comprensione servirà a fronteggiare con maggiore consapevolezza le soluzioni che rappresentano lo stato dell'arte. Per affrontare un certo numero di dettagli di questo capitolo sono di grande aiuto alcuni studi specifici, noti come *survey*, che hanno approcciato il problema da punti di vista differenti, fornendo ognuno una loro visione e categorizzazione del problema e dei suoi aspetti. Si citano tra i principali contributori alle ricerche di questa sezione gli studi [11] [14] [37].

2.1.1 Learning from Demonstration e ambiti di applicazione

I lavori di ricerca e le applicazioni pratiche facenti parte dell'ambito affrontato da questo lavoro sono accomunati dal concetto di *Learning from Demonstration*, che descriviamo brevemente: con esso s'intende la capacità di un agente robotico di imparare a eseguire una certa tipologia di azioni attraverso l'apprendimento proprio del Machine Learning, che permea questo ambito di ricerca in maniera decisa: la sua l'implementazione pratica varia ampiamente a seconda dell'approccio scelto, ma è innegabile la sua centralità nell'esecuzione degli obiettivi previsti.

Il learning from demonstration, dunque, suggerisce inevitabilmente un ampio spettro di applicazioni pratiche dove è molto importante far eseguire a un agente robotico una o più azioni, possibilmente impartendo il comando in

forma testuale o, come nel caso di questo lavoro, dimostrando effettivamente l'azione, al fine di ottenere una maggiore precisione e generalizzazione.

Gli ambiti maggiormente predisposti a trarre vantaggio da metodologie di learning by demonstration sono quelli propri dell'industria 4.0/5.0 e della robotica domestica: da una parte, la tecnologia può essere sfruttata per permettere a un'agente di eseguire azioni ripetitive, composite o addirittura rischiose, mantenendo un'accuratezza potenzialmente maggiore di quella umana nel lungo termine; dall'altra, l'agente può coadiuvare un dimostratore umano nell'esecuzione di compiti domestici, fatti da manipolazione di oggetti e strumenti di lavoro, al fine di facilitare e automatizzare l'esecuzione di compiti di varia natura.

Qualunque sia l'ambito di applicazione, la possibilità di avere un aiutante automatizzato, in grado di comprendere un compito in pochi passaggi e attraverso dimostrazioni pratiche, risulta avere notevoli prospettive future.

2.1.2 Destinatari dei lavori

Nello sviluppo di sistemi che possano rappresentare soluzioni anche commerciali è molto importante comprendere chi siano i potenziali destinatari degli stessi, e che tipo di utente rappresentino ai fini dello sviluppo. Proseguendo il discorso precedente, possiamo identificare un'utenza appartenente principalmente agli ambiti domestici e industriali, dunque perlopiù persone comuni, lavoratori e operai di vario genere che si avvalgono dell'aiuto di un agente robotico per varie mansioni.

Ciò evidenzia che questi utilizzatori siano, con buona probabilità, non esperti: dal punto di vista degli sviluppatori questa constatazione rappresenta una direzione ben precisa da intraprendere, ma anche una forte limitazione nel dover sviluppare un sistema pronto e ben calibrato, ma al contempo di facile utilizzo e competitivo sotto il profilo commerciale.

L'idea di avere un utilizzatore non esperto suggerisce che non possa essere chiesto, praticamente in nessun caso, di eseguire operazioni tecniche troppo complesse, di apprendere comandi poco intuitivi, o ancora di doversi interfacciare con calibrazioni e istruzioni poco maneggevoli. Il sistema deve poter offrire all'utente un metodo d'uso che sia contemporaneamente lineare, quanto più veloce possibile e comunque accurato.

Tutto ciò obbliga gli sviluppatori a progettare un flusso di lavoro che trovi il giusto equilibrio tra rapidità d'uso ed efficacia: non è totalmente vietato richiedere all'utente di eseguire alcune operazioni al fine di massimizzare l'accuratezza, come calibrazioni preliminari o sistemazioni in corso d'opera,

ma ciò non deve rendere l'esperienza di utilizzo troppo macchinosa. Diverse delle scelte di design di cui si parlerà più avanti dipendono anche da questi aspetti, una su tutte il numero di dimostrazioni dell'azione che l'utente deve fare prima che il robot la esegui correttamente.

2.1.3 Il ruolo del Machine Learning

In un campo come quello della Computer Vision moderna, la collaborazione con il mondo del Machine Learning non può che essere di immensa importanza. I numerosi progressi fatti in tutti gli step delle varie fasi di questi lavori vanno di pari passo con quelli compiuti in questo ampio e crescente ambito. Per inquadrare e riassumere una così vasta scelta di tecnologie e soluzioni, correlandole al loro utilizzo effettivo nei lavori che stiamo analizzando, è utile suddividere le stesse in due macro-categorie, a seconda della fase del progetto a cui appartengono: parleremo di machine learning sia al servizio dell'analisi video che al servizio dell'apprendimento sul campo da parte del robot.

Nel primo caso, le implementazioni delle reti neurali sono utilizzate al fine di eseguire un analisi esaustiva delle dimostrazioni delle azioni eseguite, dunque dei video delle stesse: si ricollegano ampiamente ai classici casi d'uso del machine learning nell'ambito della computer vision, risultando dei mattoncini fondamentali nello sviluppo del sistema. Elenchiamo di seguito alcuni dei casi d'uso principali, dove quelli effettivamente utilizzati saranno approfonditi nei capitoli dell'implementazione:

- Rilevamento di oggetti: è un campo che ha raggiunto livelli di accuratezza e velocità di esecuzione estremamente alti, e la sua utilità è cruciale. È utilizzato al fine di identificare il *bounding box* degli oggetti utilizzati sul campo, e lascia spazio a un grandissimo insieme di utilizzi successivi. La rete neurale più famosa in quest'ambito è senza dubbio YOLO (*You Only Look Once*) [30], e le sue evoluzioni principali sono nei campi di segmentazione, tracciamento degli oggetti, stima della posa umana e stima del bounding box 3D. La più importante di tutte è quella che riguarda la *open vocabulary object detection*, ovvero la capacità di un modello di riconoscimento di individuare in un'immagine istanze di classi con le quali non è stato precedentemente allenato, fornendo semplicemente una descrizione testuale di ciò che si sta cercando. I modelli principali in quest'ambito sono OWLv2 [26], GroundingDINO [23] e YOLO-World [10]

- Riconoscimento di azioni: un ambito che si concentra sull'analisi di video al fine di riconoscere ciò che avviene nello stesso, nel lungo o breve periodo. Affronta, dunque, la sfida di analizzare e correlare più immagini contemporaneamente, per produrre delle etichette che specifichino le azioni in atto. Le architetture utilizzate sono varie, tra cui citiamo le reti ricorrenti (es. LSTM), le reti convolutive 3D (es. I3D e C3D), quelle basate su *transformers* (es. ViViT) e quelle basate su grafi (es. ST-GCN). Di grande interesse sono le reti a più *stream*, che combinano i risultati provenienti da vari flussi predisposti ad analizzare informazioni diverse, come le immagini RGB, i flussi ottici o le pose dei giunti dei dimostratori. Il riconoscimento dell'azione finale sarà composto dalle informazioni ottenute da tutti gli stream

- Stima della posa 6D: è un campo ancora nel pieno del suo sviluppo, che non possiede modelli totalmente consolidati o tecniche considerate più importanti di altre. Si propone di stimare la posizione nello spazio e la rotazione attorno ai 3 assi di uno o più oggetti in scena. Le tecniche variano a seconda dei dati di cui hanno bisogno e della capacità di generalizzazione. Possiamo distinguere le tecnologie basate su modelli CAD degli oggetti da quelle basate su immagini degli stessi da varie angolazioni, oppure tecniche che necessitano di dati di profondità rispetto a altre che utilizzano solo camere monoculari, o ancora soluzioni in grado di generalizzare più facilmente e altre che si concentrano su dataset specifici. A seconda del risultato cercato si possono anche trovare metodi che non utilizzano affatto il machine learning, ma piuttosto algoritmi ben definiti di stima della posa che richiedono altri tipi di dati visivi

Nel secondo caso, le reti neurali sono utilizzate all'atto pratico per addestrare un robot a muoversi correttamente per portare a termine il compito affidatogli. Le tecniche sono numerose, e vi è grande discussione sulla loro efficacia e ciò che è necessario per usarle: c'è distinzione sulle tempistiche più o meno lunghe di raccolta dei dataset ad hoc, o ancora sulla necessità di addestramento con la supervisione di un operatore. I loro dettagli implementativi, sebbene non banali e assolutamente importanti per l'esecuzione sul campo, esulano dagli ambiti su cui si concentra questo lavoro, ma vanno riassunte al fine di inquadrare completamente il flusso di lavoro. Alcune soluzioni consolidate e ampiamente utilizzate sono:

- Behavior cloning (BC): anche detto Imitation Learning (IL), è il metodo più semplice che permette di ottenere una policy, ovvero una politica di

esecuzione di un'azione, a partire dai dati di allenamento. Esso utilizza dati accoppiati noti come stato-azione, che permettono di correlare ogni frame osservato in scena a uno specifico stato del dimostratore robotico. Il problema può essere sia di classificazione che di regressione, a seconda che il dominio considerato sia discreto o continuo. L'obiettivo è minimizzare la differenza tra l'azione prevista per un certo stato dell'ambiente e quella reale (*ground truth*). Ha come lato negativo quello di dipendere fortemente dalla qualità e varietà del dataset di addestramento, poiché si allena tentando di copiare le azioni in esso contenute

- Reinforcement learning (RL): rappresenta una famiglia di algoritmi di allenamento che seguono un metodo comune; l'agente osserva l'ambiente in un certo istante, calcola una policy sulla base degli input ricevuti e riceve un "premio", basato su una funzione di *reward* propria dell'ambiente, a seconda della correttezza dell'azione intrapresa. L'obiettivo è massimizzare il "premio" accumulato durante le predizioni. E' efficace nell'adattarsi a situazioni nuove e ambienti non visti in precedenza, ma può risultare lento poiché necessita di molte interazioni per sviluppare delle buone capacità. Un metodo per migliorare le prestazione è fare un pre-allenamento con behavior closning, per apprendere le feature principali del dataset e raffinare le predizioni con il RL

- Inverse reinforcement learning (IRL): se con l'uso del RL vi è la necessità di calcolare delle funzioni di reward dalle osservazioni dei task e dell'ambiente, che possono diventare anche molto complesse in caso di compiti e ambientazioni compositi, l'IRL propone un metodo diverso. Mentre il BC sfrutta i dataset video per calcolare una policy di azione, l'IRL li usa per il calcolo della funzione di reward. Questa potrà poi essere utilizzata dall'agente per trovare la policy d'azione corretta con metodi quali il RL classico. Sebbene possa apparire come una soluzione completa, porta con sé la necessità di ottimizzare il calcolo della funzione di reward e di adattarsi correttamente ad azioni e traiettorie non viste nel dataset di addestramento

- Generative Adversarial Imitation Learning (GAIL): prendendo forte ispirazione dai modelli di reti generative GAN, propone di imparare una reward function dal dataset d'ingresso, come nell'IRL, e di calcolare le policy con l'uso del RL. Il modulo che si occupa delle policy funge da generatore, mentre un altro modulo discriminatore ha il compito di indovinare se questa è stata prodotta dal generatore o pescata a caso

dal dataset d'ingresso. Come nelle GAN, i due contendenti provano a migliorarsi a vicenda, l'uno tentando di creare policy sempre più realistiche e l'altro affinando le sue capacità di riconoscimento

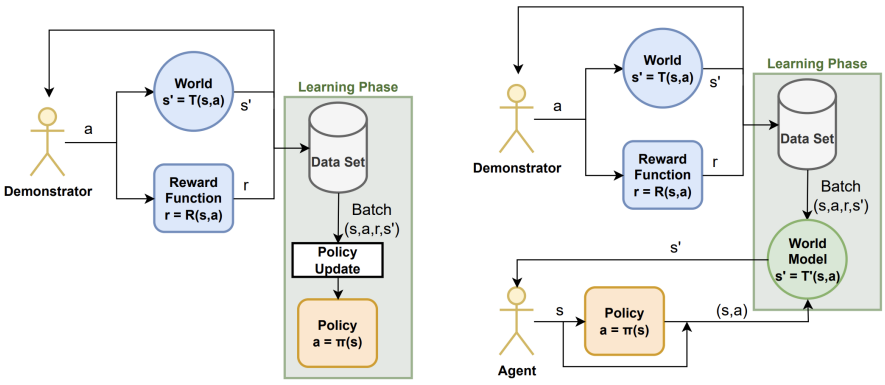


Figura 2.1: Differenze di approccio tra BC e IRL

2.1.4 Approccio "in the wild" o circoscritto

Una delle principali sfide che permeano la manipolazione robotica riguarda la scelta dell'ambiente di lavoro, o meglio, la capacità di adattarsi ad ambienti differenti e/o nuovi. In tal senso, molti lavori propri di questo ambito tentano degli approcci cosiddetti *in the wild*, letteralmente "in natura", parafrasando l'idea di permettere all'agente di eseguire un task di manipolazione qualunque sia l'ambiente che lo circondi, e dunque qualunque sia il set di oggetti con il quale avrà a che fare. L'idea di far affrontare a un robot un task di manipolazione che sia indipendente dalla scena che si trova davanti aggiunge numerosi livelli di sfida per ottenere un'implementazione efficace e puntuale.

Al contrario, un ambiente circoscritto e ben definito aiuta il robot a concentrarsi su altri aspetti determinanti, a discapito della capacità di generalizzazione e della varietà di compiti in grado di affrontare. Al contempo, l'agente può utilizzare dei riferimenti fissi e delle pratiche standard per velocizzare alcuni aspetti dell'azione sul campo, come, in un esempio banale, la determinazione della propria posizione e la circoscrizione della posizione degli oggetti in un area ben determinata, come può essere un certo tavolo da lavoro in posizione fissa, che permette di evitare la ricerca della posizione degli oggetti lungo l'asse dell'altezza.

Nel caso dell'ambientazione di questo lavoro, la scelta è ricaduta sulla seconda strategia, per i fini di analisi già descritti in precedenza.

2.1.5 Approcci "task-agnostic" e "task-oriented"

Al pari del dualismo affrontato nel paragrafo precedente, un'altra scelta che è necessario intraprendere per sviluppare un sistema che dimostri una buona precisione è quella che riguarda l'insieme di azioni che l'agente è in grado di eseguire. In particolare, si dividono gli approcci in due categorie note come *task-agnostic* e *task-oriented*: il primo, al pari dell'approccio "in the wild" appena analizzato, vuole permettere all'agente di eseguire grossomodo qualsiasi tipo di azione, entro i confini imposti dai limiti meccanici e strutturali del robot stesso; il secondo, al contrario, vuole limitare il sistema a un insieme ben definito di azioni, per le quali l'agente è ben addestrato e può garantire, sulla carta, una maggiore accuratezza.

Il vantaggio del primo approccio è, senza dubbio, la capacità di generalizzazione e l'adattabilità a compiti di vario tipo, altresì esponendosi al rischio di imprecisione e inefficacia. Nel secondo caso, invece, è ovvia la necessità di circoscrivere i compiti affrontabili al solo gruppo di azioni conosciute, ma con il vantaggio di ottenere un'esecuzione potenzialmente più precisa. Inoltre, addestrando l'agente all'esecuzione di un gruppo sufficientemente ampio di azioni base, esso può affrontare compiti complessi con una strategia *divide et impera*, ovvero scomponendoli in sotto-azioni riconducibili a quelle che conosce già.

Come si vedrà nei paragrafi successivi, sebbene la prima strategia suggerisca una maggiore propensione a sfide sempre più complesse, la seconda è più indirizzata verso implementazioni di alto livello, ad esempio con i modelli di linguaggio più interessanti del panorama del machine learning, proprio per garantire un certo livello di consapevolezza delle azioni che ci si propone di affrontare.

2.1.6 Sfide sulle dimostrazioni "one-shot"

Come già citato in precedenza, un utente che si interfaccia con queste soluzioni deve poter essere messo nelle condizioni migliori per l'utilizzo delle stesse. Ciò significa anche permettergli di ottenere il risultato voluto con il minimo sforzo, ovvero, nel caso che si sta analizzando, con il minimo numero necessario di dimostrazioni dell'azione che si vuole replicare. Questa ricerca della semplicità d'uso ha spinto gli sviluppatori a concentrarsi sulle metodologie cosiddette

one-shot, ovvero dove al dimostratore è richiesto di eseguire l'azione una volta sola. Tale scelta genera un compromesso necessario, dove è richiesto che le reti neurali a supporto dell'architettura ricevano un addestramento completo, preciso ed efficace, e che in fase di inferenza permetta loro di focalizzarsi sulla singola dimostrazione, rendendola sufficiente a estrapolare tutti le informazioni necessarie.

2.1.7 Sfide sulla disponibilità di dati

Il modo in cui una macchina possa apprendere una metodologia di azione attraverso l'analisi di immagini è una sfida che affligge da sempre l'ambito di ricerca congiunto tra computer vision e robotica, ma che lo espone, contemporaneamente, a una forte limitazione: la quantità di dati che è necessario raccogliere al fine di eseguire un apprendimento efficace è considerevole per qualunque metodo prescelto, ma la loro disponibilità sulla rete è fortemente limitata.

I dataset a disposizione dei ricercatori che rappresentino, in maniera esaustiva e accurata, azioni e attività di vita quotidiana, o addirittura compiti propri di determinati ambienti industriali, sono pochi e spesso troppo generici. Per l'addestramento di un agente robotico in vari task di manipolazione vi è innanzitutto la necessità di reperire fonti video, che non siano dunque solo immagini statiche ma che dimostrino un vero flusso di esecuzione, e che sappiamo esistere in numero indubbiamente non sufficiente. Vi si aggiunge la necessità, di certo non secondaria, che i video delle azioni dimostrate siano ben concentrati sugli oggetti manipolati, perciò con un buon punto di vista sull'ambiente e un buon focus sulla manipolazione.

Molti dataset video di ampia diffusione, *Kinetics* per citarne uno su tutti, sono perlopiù raccolte di azioni generiche dimostrate da punti di vista molto diversi, e che spesso non si adattano correttamente al tipo di informazione di cui il sistema avrebbe bisogno per apprendere correttamente come eseguire i movimenti. Da qui nasce, appunto, la sfida di reperire un numero sufficiente di dati, sia raccolti ad hoc che presi dalla rete, per addestrare correttamente l'agente nei compiti prestabiliti.

2.2 Strumenti e dati

Analizziamo ora i principali strumenti utilizzati nei numerosi lavori di ricerca in questo ambito, sia in termini di hardware che di software.

2.2.1 Robot e simulatori

Appare naturale che, al netto della necessità di apprendere efficacemente dalle dimostrazioni fornite al sistema, l'agente robotico in sé rappresenti un punto importante dell'architettura di lavoro, essendo il responsabile dell'esecuzione sul campo di quanto analizzato e appreso in precedenza. Sebbene la sua rilevanza dal punto di vista delle fasi di apprendimento e analisi possa risultare secondaria, l'efficacia della dimostrazione pratica si fonda anche e soprattutto sulle capacità degli agenti che la espletano.

I robot che vengono utilizzati maggiormente sul campo per l'implementazione pratica delle ricerche sono della famiglia dei manipolatori, ovvero degli agenti costruiti con una forma simile a un braccio umano e specializzati in compiti di interazione semplice con oggetti di vario tipo.

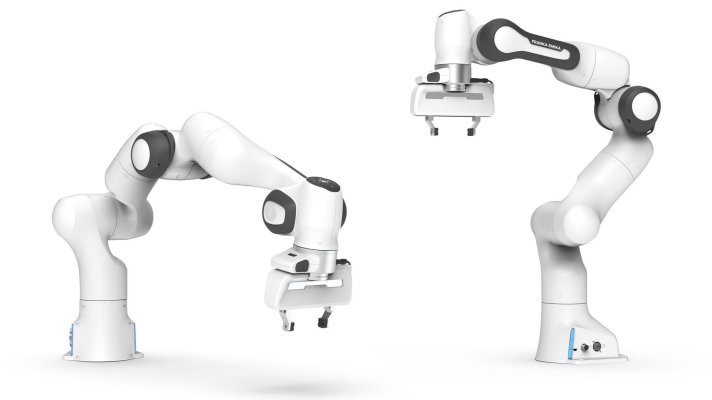


Figura 2.2: Il robot Franka Emika Panda, uno dei più diffusi in questo campo

I manipolatori si distinguono per il numero di gradi di libertà (anche detti *DOF*, dall'inglese *Degree Of Freedom*) da cui sono caratterizzati, dove con essi intendiamo il numero di giunti in grado di fornire un movimento rotatorio o traslatorio in una certa direzione. È evidente pensare che un maggior numero di gradi di libertà rappresenti anche una maggiore libertà di movimento e una più ampia manovrabilità dell'agente, ma essi possono costituire anche una sfida nel corretto controllo del robot stesso. La maggior parte dei manipolatori utilizzati sono costituiti da sei o sette gradi di libertà, che rappresentano un ottimo compromesso per un numero esteso di utilizzi.

Tali manipolatori sono forniti di vari tipi di *gripper*, ovvero le pinze poste al termine del robot stesso e responsabili dell'effettiva esecuzione dei compiti: nella maggior parte dei casi, e per la maggior parte dei compiti, sono sufficienti dei gripper costituiti da due "dita" robotiche, lineari e senza ulteriori giunti, che riprendono il movimento proprio del contatto umano tra pollice e indice e sono in grado di manipolare, con forze e aperture differenti, gli oggetti che hanno davanti. Talvolta, per ottenere una gamma di azioni quanto più vicina a quella umana possibile, i gripper sono costituiti da vere e proprie "mani" robotiche, composte da numerose dita estremamente somiglianti a quelle umane: è necessario sottolineare che queste soluzioni, per quanto innovative e interessanti, sono riservate a progetti specifici che richiedono un alto livello di somiglianza con una manipolazione reale, e rappresentano una minoranza rispetto ai ben più semplici, ma più diffusi, gripper a due dita. Inoltre, l'uso di "mani" robotiche aumenta considerevolmente il numero di gradi di libertà del sistema, spesso aggiungendo dei livelli di difficoltà non necessari in questo specifico ambito.

Alcuni tra i più utilizzati sono robot pensati per ambienti industriali, e in alcuni casi possono essere dotati di un sistema di movimento per la circolazione a terra. Citiamo, tra i modelli commerciali e non più diffusi, Emika Panda prodotto da Franka, Kinova Gen. 3, UR5 e UR10 di Universal Robots, LBR iiwa di Kuka, Stretch di Open Robotics e i modelli di manipolatore mobile creati da Google e Everyday Robots per la serie di lavori nota come RT.

Una menzione è necessaria anche per gli ambienti software che simulano tali manipolatori robotici, e che sono di grande aiuto quando le possibilità di ottenere robot reali sono scarse, anche per motivazioni economiche. Alcuni degli ambienti più utilizzati sono Gazebo, sviluppato da Open Robotics e uno dei pochi gratuiti e open-source, CoppeliaSim, RoboDK e IsaacSIM di Nvidia. A questi si aggiungono anche delle piattaforme di benchmarking, pre-confezionate per simulare determinati compiti con alcuni specifici robot, in ambienti ben definiti e con meno possibilità di personalizzazione, tra i quali citiamo RLBench e FrankaKitchen.

2.2.2 Videocamere e sensori

In un ambito che coinvolga la Computer Vision la tipologia di videocamere e sensori, utilizzati per cogliere le informazioni provenienti dall'ambiente di lavoro, è fondamentale. L'analisi delle azioni e delle loro conseguenze passa anche da una corretta acquisizione di ciò che è avvenuto, con l'eventuale necessità di condire le semplici immagini di una videocamera con dati aggiuntivi

che aiutino il sistema per tale scopo.

Le videocamere classiche, in grado di raccogliere solo immagini RGB, sono il punto di partenza fondamentale: costituiscono la scelta più leggera e scalabile, oltre che quella più appetibile sia in termini di costi che di quantità di informazioni da processare. Sono reperibili ovunque, e spesso suggeriscono l'implementazione di soluzioni anche su dispositivi mobili o poco potenti computazionalmente.

Allo stesso tempo, muovendosi e manipolando oggetti in un ambiente tridimensionale, sono spesso utilizzate delle videocamere che raccolgono anche informazioni sulla profondità, meglio note come RGBD, esistenti con varie tecnologie volte a questo scopo. L'informazione di profondità può rivelarsi molto comoda in compiti come la stima della posizione di un oggetto o la ricostruzione della point map tridimensionale della scena, e molto spesso queste videocamere spesso offrono ottime prestazioni a un prezzo relativamente contenuto. Qui una breve panoramica delle tecnologie utilizzate, con relativo esempio d'uso commerciale:

- Time of Flight (ToF): tecnologia basata sull'invio di impulsi di luce infrarossa e sulla misurazione del tempo impiegato a ritornare al sensore. Sono influenzate negativamente da forti fonti di luce. Tra le più usate, la Azure Kinect DK e la Intel RealSense L515
- LiDAR (Light Detection And Ranging): evoluzione delle ToF dove viene usato più di un impulso di luce a infrarossi, che confrontati punto-punto producono una scansione di maggiore precisione. Sono ancora abbastanza costosi, ma si trovano integrati in dispositivi mobili. Un esempio su tutti, l'Apple LiDAR Scanner presente a bordo dei più recenti iPhone e iPad
- A luce strutturata: la telecamera invia un pattern di luce infrarossa e calcola come questi viene deformato dal contatto con gli oggetti. Anche queste sensibili a forti luci. Tra le più note, il Kinect di prima generazione e Intel RealSense D415
- Stereo Vision: riprendendo il funzionamento degli occhi umani, usano due obiettivi a distanza nota per raccogliere due immagini distinte, e il loro confronto produrrà l'informazione di profondità. La precisione è ridotta con oggetti con texture lisce o riflettenti. Tra gli esempi, Stereolabs ZED 2 e Intel RealSense D435

Per ottenere ulteriori informazioni riguardo l'ambiente circostante e l'azione compiuta ci si può anche avvalere di altre tecnologie, con scopi diversi e

livelli di complessità aggiuntivi. Una delle possibilità è quella di utilizzare sensori di vario tipo indossati dal dimostratore, che aiutano nella ricezione e comprensione di informazioni molto utili: un esempio è l'uso di guanti sensorizzati, come i noti MANUS, che permettono di cogliere con ampia precisione la posa delle mani e dei loro giunti.

Figura 2.3: Esempi di camere di profondità: 1) ToF; 2) LiDAR; 3) A luce strutturata; 4) Stereo vision

Ancora, è di grande interesse in questo campo l'integrazione con sistemi di Augmented Reality (AR) e Virtual Reality (VR), per scopi abbastanza diversi che citiamo brevemente e che verranno approfonditi nei paragrafi successivi: ad esempio, possono essere utilizzati per muovere direttamente il braccio robotico in fase di raccolta di dataset, per visualizzare sul campo l'andamento e le traiettorie dello stesso, o ancora per simulare, con piani di lavoro e oggetti reali, le azioni che potrebbe intraprendere l'agente.

Ricollegandosi ad alcune affermazioni precedenti, è facile intendere che questi sistemi rendano i lavori a essi correlati meno avvezzi a utenti non esperti, andando dunque in contrasto con la propensione a sviluppare soluzioni utilizzabili da chiunque, senza la necessità di avere con sé un utente esperto o di dover investire troppo tempo nell'imparare a usare il sistema. Tuttavia, continuano a rappresentare delle idee interessanti per fasi intermedie dei progetti o soluzioni finali avveniristiche.

2.2.3 Dataset

Come citato nell'introduzione del capitolo, la ricerca, o la raccolta, di dataset da utilizzare per l'addestramento del sistema rappresenta una sfida importante in questo campo. La quantità di materiale è sicuramente limitata, ma non per questo assente: negli anni diversi lavori hanno avuto come obiettivo quello di creare e organizzare raccolte di dati, principalmente in formato video, utili per varie tipologie di sistemi e approcci, spesso raccolti per una specifica soluzione e divenuti utili per lavori simili. Distinguiamo due tipologie di dataset utilizzate in questo ambito: quelli che raccolgono video "egocentrici", ovvero dal punto di vista umano, e quelli che raccolgono video di dimostrazioni eseguite dal robot.

Figura 2.4: Esempi di video egocentrici presi dal dataset Ego4D

Nel primo caso, si tratta di video registrati da un dimostratore umano, che esegue un azione e la registra da un punto di vista estremamente simile a quello dei suoi occhi, dando l'impressione di impersonare l'operatore stesso. Sono estremamente utili nei numerosi casi in cui la telecamera principale del sistema è montata a bordo del robot stesso, fornendo la stessa prospettiva sull'ambiente che avrebbe il dimostratore. Alcuni esempi importanti di questa categoria sono:

- Ego4D, con oltre 3600 ore di video raccolti in varie parti del mondo usando dispositivi indossabili, principalmente focalizzato su azione a lungo termine

- Something-Something-V2, con oltre 200.000 video e 174 classi e maggiormente focalizzato su azioni semplici, brevi e ben definite
- EpicKitchens, con 55 ore di video di azioni correlate al mondo della cucina, dunque della manipolazione di semplici oggetti domestici
- 100DOH (Days Of Hands), ovvero 100 giorni di video focalizzati su azioni compiute con le mani, molto utili per la stima della posa dei giunti delle stesse

Figura 2.5: Esempi di video robot-centrici presi dal dataset Open-X Embodiment

Nel secondo caso, si tratta di dataset composti da video che mostrano direttamente il robot mentre esegue delle azioni sul campo, accompagnati da dati aggiuntivi come la posa dei giunti, lo stato del sistema e il comando testuale impartito. Spesso questi dataset sono raccolti con metodi quali la *teleoperation*, ovvero il controllo diretto del braccio robotico attraverso degli opportuni controller, eventualmente coadiuvati da sistemi VR, oppure attraverso la meno diffusa *kinesthetics*, ovvero la guida del robot da parte di un operatore che ne muove fisicamente i giunti. Qui, gli esempi principali sono:

- Open-X Embodiment, con milioni di esempi che promuovono il *cross-embodiment*, ovvero la capacità di un robot di imparare a eseguire azioni osservando altri robot
- BC-Z Dataset, raccolto per l'omonimo lavoro di ricerca, con un focus su azioni più simili a quelle che eseguirebbe un umano

- RT-1 Dataset, anch'esso raccolto per l'omonimo lavoro, incentrato sull'usabilità con reti neurali basate su Transformer, dunque sulla correlazione tra azione e comando testuale

Nonostante questi esempi, rimane di centrale importanza scegliere, o raccogliere, un dataset che sia conforme al caso d'uso specificato, e utilizzarlo al meglio per l'addestramento corretto delle librerie di machine learning utilizzate.

2.3 Strategia di lavoro

Una volta identificati i vari mattoncini che compongono i lavori principali in quest'ambito, è bene inquadrare i vari punti che costituiscono il flusso di lavoro comune a tutti loro. Distinguiamo, in maniera riassuntiva, quattro fasi che rappresentano al meglio tale flusso.

In prima battuta vi è la necessità di scegliere la configurazione della strategia e dell'ambiente di lavoro, assieme a chi eseguirà le azioni e come verranno raccolti i dati necessari. Come già citato in precedenza, la scelta dipende essenzialmente da quali dati richiede il sistema e quali siano i più convenienti in termini di velocità di raccolta e di analisi. Il dimostratore prescelto può essere, sostanzialmente, sia un operatore che agisce direttamente sulla scena che il robot stesso: nel primo caso raccoglieremo semplicemente dati visivi e/o di profondità, eventualmente coadiuvati da sensoristica aggiuntiva, mentre nel secondo potremmo raccogliere anche dati sulla posa dei giunti del robot, che garantiscono maggiore precisione.

Segue, come seconda fase, la raccolta pratica sul campo dei dati. La tecnica prescelta, allora, dipenderà direttamente dal dimostratore selezionato nella fase precedente:

- con un operatore umano si parla di raccolta indiretta, eseguita essenzialmente con la registrazione video o di sensori indossabili
- con l'uso del robot si parla di raccolta diretta, operando la macchina attraverso tecniche differenti. Come già citato, alcune di queste sono la teleoperation, dove si usa strumentazione VR e telecomandi specifici per operare l'agente, o la kinestetics, dove i giunti del robot sono mossi manualmente dall'operatore

Il terzo step è costituito dalla fase di configurazione e addestramento del sistema selezionato. I dati raccolti, dopo le opportune fasi di sistemazione

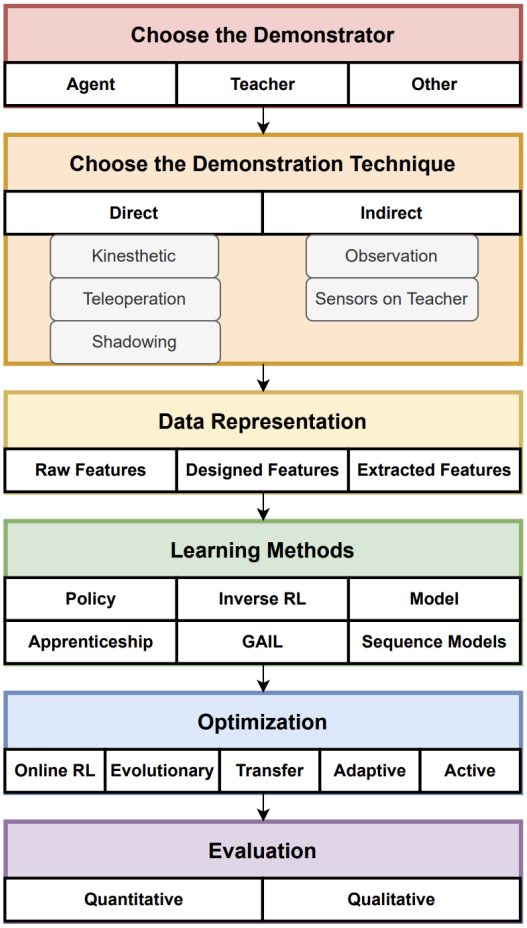


Figura 2.6: Flusso di lavoro proposto da [11]

e organizzazione, sono utilizzati per addestrare il sistema sia ad analizzare gli input in fase di inferenza che a eseguire correttamente le azioni scelte. L'addestramento riguarda, perciò, tutte le reti neurali che si occupano di compiti come il riconoscimento di oggetti e di azioni, o ancora l'analisi della posa 6D o dei bounding box 3D, ma anche delle reti che si occupano dell'esecuzione pratica dell'azione, e che implementano gli algoritmi elencati nella sezione dedicata.

Nell'ultima fase troviamo il test effettivo sul campo, l'ottimizzazione della metodologia e la valutazione dei risultati trovati. Le prove pratiche con l'agente suggeriscono quali siano i punti più critici del sistema assieme a quelli più accurati, e permettono di esaminare i risultati da un punto di vista sia qualitativo che quantitativo.

2.4 Lavori principali

Affrontiamo ora l'analisi di alcuni dei lavori più importanti facenti parte dello stato dell'arte del *learning from demonstration*. In questa sezione si vedranno soluzioni che, pur utilizzando strategie diverse, hanno in comune il focus sugli aspetti di analisi delle dimostrazioni pratiche e comprensione dell'ambiente e delle azioni svolte: mentre i lavori che verranno elencati nella sezione successiva alzano il livello di astrazione del lavoro, implementando soluzioni più avanzate e dando per scontate altre scelte (o anche seguendo filosofie d'approccio diverse), in queste ricerche ritroviamo modi diversi di affrontare lo stesso flusso di lavoro, raccogliere i dati, analizzarli ed eseguire l'azione sul campo. Si noteranno implementazioni che coinvolgono l'uso di modelli di linguaggio e input testuali, che esulano dallo scopo di questa ricerca e di quest'analisi, ma le cui ricerche affrontano la parte di acquisizione e analisi video in maniera innovativa e interessante.

Uno dei paper di ricerca che ha dato gli spunti più interessanti per affrontare questo lavoro è stato quello di WHIRL [2], il cui sistema si propone di imparare a imitare gli esseri umani osservandoli mentre compiono qualunque tipo di azione. Riprendendo la categorizzazione che abbiamo affrontato prima, si tratta di un approccio *in the wild*, ovvero aperto a qualunque scenario e ambiente, task agnostic, ovvero indipendente dalle azioni da imitare, e *one-shot*, cioè avente bisogno di una singola dimostrazione dell'azione. La pretesa è particolarmente alta, ma i risultati dichiarati sembrano rispettare le premesse. All'atto pratico, si analizza il video dell'azione eseguita, e una rete di hand-object interaction [32] identifica gli istanti in cui è avvenuto il contatto tra mano e oggetto, riconoscendoli e calcolando la distanza tra i due bounding box. Ciò avviene per tutta la sequenza dell'interazione con l'oggetto, permettendo di stimare i frame di inizio e fine contatto assieme a un numero variabile di frame intermedi, che permetteranno di stimare la traiettoria del movimento. Queste informazioni, assieme ad altre aggiuntive sull'apertura della mano e sulla sua rotazione, costituiscono un insieme noto come *human priors*. Questi priors, una volta tradotti nel dominio del robot, non possono rappresentare una guida unica per eseguire l'azione, per via delle ovvie differenze morfologiche tra agente robotico e umano. Si decide, dunque, di utilizzare due politiche di addestramento in collaborazione: la prima è chiamata *task policy*, e sfrutta il concetto di Residual Learning [21] coadiuvato dai Variational Auto-Encoders (VAE) per calcolare le azioni da eseguire basandosi sui prior precedenti; la seconda è chiamata *exploration policy*, e si occupa, come suggerisce il nome, di "esplorare attorno" alla policy

precedente per compensare il divario tra umano e robot, e permettergli di provare numerose volte l'azione fino al raggiungimento del risultato desiderato. Per comprendere se tale risultato è stato effettivamente ottenuto, dapprima si rimuovono sia il robot che l'umano dai rispettivi video dell'azione con una procedura di *video impainting*, e successivamente si confrontano tali video con una specifica rete di action recognition [27] che determina se le azioni eseguite producono lo stesso risultato, o meglio lo stesso effetto visivo finale.

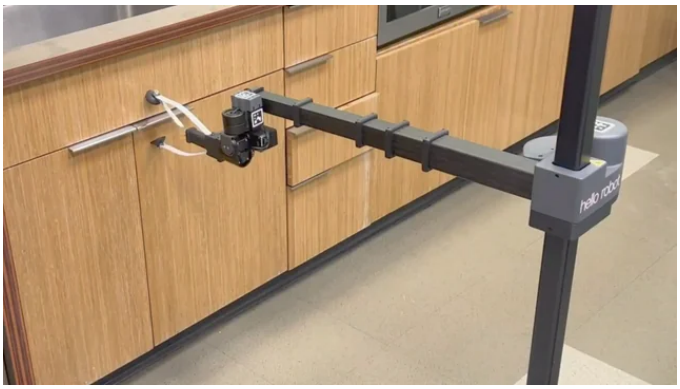


Figura 2.7: Esecuzione di task in-the-wild in WHIRL

Questo tipo di approccio appare piuttosto efficace, ma non produce una comprensione totale dell'azione che è stata eseguita: il sistema usa diverse metodologie per riprodurre i movimenti e i risultati ottenuti dall'umano, ma non ha consapevolezza né dell'oggetto con cui si è interagito né del tipo di azione che è stata eseguita, mirando a una semplice, sebbene efficace, imitazione del dimostratore.

Un paper che mostra un approccio lievemente diverso è quello di Vision Robotics Bridge (VRB) [3], che introduce al lettore il concetto di *affordances*, meglio traducibile in italiano come "invito all'uso": con esso s'intende la capacità di un oggetto, o più in generale di un qualsiasi punto nello spazio, di suggerire le azioni appropriate per manipolarlo, o comunque di indicare la possibilità di interagirvi. Un oggetto possiede una certa affordance solo in relazione con qualche attore che lo manipoli, poiché, ad esempio, una sedia suggerisce l'azione del "sedersi" solo se qualcuno in precedenza l'ha effettivamente eseguita. In tale senso, il progetto è diviso in due parti: nella prima, si utilizzano dei dataset di video egocentrici, assieme alla rete di hand-object detection già citata [32], per addestrare un visual encoder basato su ResNet al fine di comprendere, data una scena, quali siano gli oggetti interagibili, i vari modi con cui questi possano essere manipolati e

le traiettorie post-contatto che è più probabile che vengano eseguite; nella seconda parte, si testano delle tecniche diverse per mettere in campo quanto raccolto dall'addestramento precedente, allenando a loro volta dei modelli specifici. In quest'ultima fase, le tecniche preposte sono l'imitation learning basato su immagini obiettivo, la reward-free exploration, l'addestramento goal-conditioned e le affordances usate per modellare uno spazio d'azione.

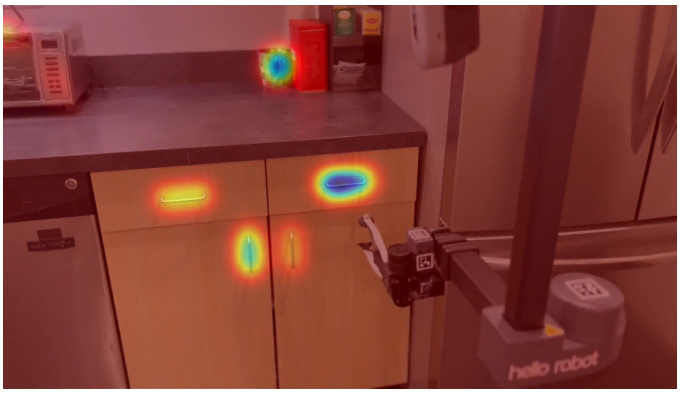


Figura 2.8: Visualizzazione delle affordances con VRB

Due altri lavori prodotti dal team di Meta AI propongono un approccio più orientato al machine learning e alla ricerca di feature all'interno dei video. I paper denominati R3M (Robot-centric Representation for Manipulation) [28] e VIP (Value Implicit Pre-training) [25] sono incentrati sull'addestramento di determinati modelli al fine di estrarre informazioni implicite dai dataset di riferimento. Il paper R3M fa affidamento sull'apprendimento supervisionato per carpire delle informazioni spaziali e temporali dai video, basandosi sul predire quando due immagini sono consecutive o sull'associare a un certo video una descrizione testuale dello stesso. Una volta apprese queste feature visive e riadattate al dominio del robot, utilizza il reinforcement learning per generare una funzione di ricompensa specifica per il compito desiderato, che guida l'addestramento della policy usata per manipolare il robot. In maniera molto simile, il paper VIP propone un sistema che apprende delle feature dal video e usa il reinforcement learning per calcolare le policy corrette, aggiungendo la capacità di identificare scene e configurazioni che propongano una funzione di reward di maggiore appetibilità.

Il paper denominato VoxPoser [18] propone una maniera innovativa per comprendere la scena che si trova davanti al robot. Parte dalla combinazione di due tecnologie basate sul linguaggio fortemente utilizzate di recente: i Large Language Models (LLM) sono utilizzati per analizzare un comando testuale

e scomporlo in sotto-comandi più piccoli e più semplici, mentre i Vision Language Models (VLM) aiutano a collegare le descrizioni testuali di azione e oggetti alla scena effettivamente visionata. Quest'ultimo è una combinazione di reti che effettuano la segmentazione della *voxel map* della scena e sono in grado di ricollegare determinati suoi punti a specifiche combinazioni di parole. Da questa analisi sono generate una serie di altre voxel map che rappresentano graficamente, per ogni punto nello spazio, uno tra i cinque seguenti parametri del movimento: affordance (quanto vogliamo manipolare quel punto), avoidance (quanto lo vogliamo evitare, ad esempio per non interagire con oggetti non necessari), velocità dell'end-effector, rotazione dell'end-effector e apertura del gripper. Queste voxel maps guideranno il motion planner nella decisione della strada migliore per eseguire l'azione considerata.

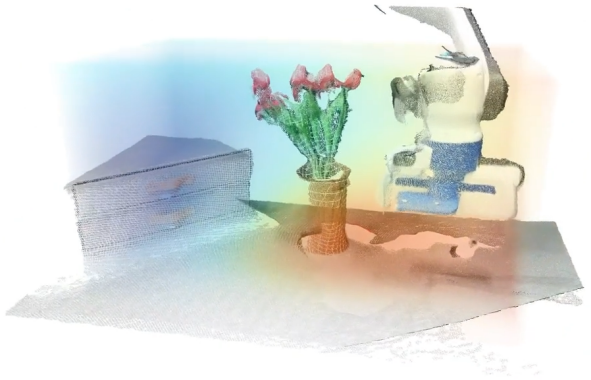


Figura 2.9: Esempio di voxel map prodotta da VoxPoser

Il lavoro noto come BC-Z [19], di Google Robotics, mostra la raccolta di un grandissimo dataset omonimo, eseguita con il metodo della teleoperation, di interazioni tra il robot e una scena con degli oggetti. Tale dataset viene utilizzato per addestrare una rete ResNet a cui viene dato in pasto, oltre al video eseguito dal robot, uno tra un video della stessa azione fatta da un umano o il corrispettivo comando testuale. Da questa rete, con le opportune letture degli output, per ogni frame siamo in grado di ottenere la posizione nello spazio, la rotazione e l'apertura del gripper del robot. Un punto negativo del sistema è che ha bisogno di diverse dimostrazioni dell'azione per generalizzare correttamente.

Due lavori interessanti provengono dal settore ricerche di Nvidia. La soluzione proposta da CLIPort [33] si ispira alla teoria dei due flussi, definita in campo psicologico sulla formulazione delle azioni nel cervello umano, e propone un sistema che include due reti neurali differenti per la definizione dei movimenti, di cui una identificata come "semantica" (CLIP) e caratterizzata dall'analisi del comando testuale, e una considerata "spaziale" (Transporter ResNet) che invece analizza la scena sotto il profilo delle immagini RGB-D), al fine di produrre il movimento corretto da far replicare al robot. Il lavoro di PerAct [34], allo stesso modo, evolve il concetto di CLIPort sfruttando le capacità dei moduli transformer: si ricostruiscono delle voxel map a partire dalle osservazioni del movimento del robot (o meglio, dalle pose dei suoi giunti), e si forniscono in ingresso alla rete basata su transformer assieme a un comando testuale (codificato, come prima, con l'uso di CLIP). La rete è addestrata usando il reinforcement learning a predire la corretta posa del robot a partire da comando testuale e voxel map, ed è in grado di produrre in uscita il corretto movimento che deve effettuare il robot per quel frame, in termini di posa dei giunti, rotazione e apertura del gripper.

Infine, una serie di lavori più avanzati che forniscono ancora spunti interessanti sulle metodologie di learning from demonstration è quella di Robotic Transformers, sviluppati da Google Robotics. Il sistema alla base di RT-1 [5] sfrutta anch'esso la potenza messa a disposizione dai transformer per combinare un input visivo assieme a uno testuale: l'encoder testuale USE [8] produce un embedding del comando, mentre la combinazione delle reti FiLM [29] e EfficientNet [38] tokenizza tale embedding assieme a sei immagini della scena che ha di fronte. Questi token sono utilizzati dalla rete basata su trasformatori per ottenere in uscita le azioni che servono al corretto movimento del robot, e si propone di farlo a una velocità molto simile a quella dell'umano. Se la frequenza di generazione delle azioni del sistema "base" è stimata intorno ai 2-4 Hz, si possono migliorare le prestazioni riducendo il numero di token o anche sfruttando quelli già calcolati. Evoluzione di questo paper è il suo successore, RT-2 [4], che fa uso di alcuni visual-language models pre-allenati, come PaLI-X [9] o PaLM-E [12], per fare il fine-tuning di un modello basato su transformer, attraverso tecniche come il visual-question answering ("Cosa succede in questa immagine?") o la predizione su coppie immagini-azioni ("Data questa scena, cosa dovrebbe fare il robot per eseguire questa azione?"). La rete sarà quindi in grado di identificare l'azione corretta e di produrla in output come token, che andranno opportunamente interpretati come traslazione, rotazione e apertura del gripper. La grandezza del modello, fatto da un elevato numero di parametri, obbliga il sistema a mettere questi

sul "cloud" e interrogarlo man mano che serve. RT-2 dimostra risultati simili al suo predecessore sui task già noti, ma una capacità di generalizzazione a task nuovi e mai visti decisamente migliorata.

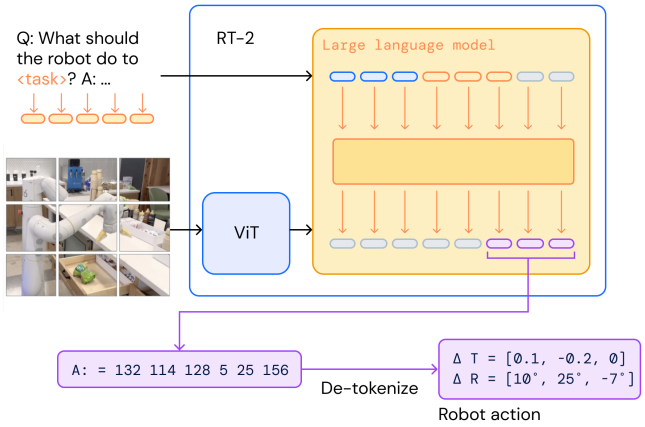


Figura 2.10: Architettura di RT-2 e analisi degli output prodotti

2.5 Metodi alternativi

Analizziamo ora alcuni paper di ricerca che presentano delle soluzioni con approcci e filosofie lievemente differenti rispetto a quelle "classiche. Come preannunciato, in questi casi alcune implementazioni pratiche riguardanti l'analisi dei dati video possono essere date per scontato, focalizzandosi maggiormente su altre novità tecnologiche, o ancora presentano degli approcci piuttosto differenti per ottenere risultati simili, come nel caso dei lavori che coinvolgono tecnologie di AR e VR.

L'approccio di SayCan [1], di Google Robotics, risulta particolarmente interessante. Il paper parte dall'idea di addestrare un robot a eseguire una pletora di azioni-base semplici e ben definite, note come *skills*, di modo da poter affrontare qualunque tipo di compito scomponendo la mansione in una serie di sotto-compiti più elementari. Tali skills vengono insegnate al robot seguendo il metodo di behavior cloning definito da BC-Z [19], oppure attraverso il reinforcement learning, utilizzando sia dataset video registrati sul campo che descrizioni testuali della scena. La scelta del metodo migliore, o i suoi dettagli implementativi, non costituiscono comunque il punto fondamentale della soluzione. Una volta terminato l'addestramento del robot, la ricerca afferma di voler combinare la decisione sulle azioni da intraprendere in una

determinata ambientazione a partire da quanto suggerito da due "opinioni" differenti: quella proveniente da un modello di linguaggio e quella proveniente da un modello di affordance. Il primo è in grado di suggerire quale possa essere l'azione-base migliore da fare sulla base del comando impartito, mentre la seconda è in grado di stabilire se è possibile eseguire quell'azione con la scena che ci si trova davanti. A titolo di esempio, se il comando è "portami una mela" e il robot non si trova davanti a essa, il modello di linguaggio suggerirà come azione "afferra la mela", mentre il modello delle affordance suggerirà come azione migliore "ricerca una mela nella stanza", poiché non è possibile eseguire l'azione nel contesto corrente. Il modello sembra adattarsi bene a diversi tipi di comandi e situazioni, mentre si ritiene più macchinoso il processo di insegnamento di nuove skill all'agente.

Ulteriore evolizione della serie di lavori su Robotic Transformers di Google Robotics è RT-Trajectory [17], che propone di sfruttare le informazioni sulla traiettoria del movimento e utilizzarle in maniera innovativa in fase di inferenza. Esso mostra la raccolta in teleoperation di un dataset ad-hoc di azioni eseguite da un robot, dai cui video si estraggono le informazioni grafiche sui movimenti eseguiti: viene utilizzata una linea per la traiettoria dell'end effector, un pallino per il contatto (o la fine del contatto) con un oggetto, e colorazioni diverse per descrivere la velocità del gripper e la coordinata di profondità rispetto alla camera. Con queste informazioni viene allenata una rete specifica in grado di predire la policy corretta per un certo tipo di movimento. In fase di inferenza l'utente può disegnare la traiettoria desiderata, su un'immagine della scena che si trova di fronte e con un opportuna interfaccia, assieme ai punti di contatto con gli oggetti e a colori diversi per le informazioni aggiuntive già citate. In tal modo, si offre un'interfaccia unica e di facile utilizzo per permettere a chiunque di istruire correttamente il robot nella ripetizione di una certa azione.

Il lavoro di Robotic Telekinesis [35] vuole istruire il sistema a replicare in tempo reale l'azione dell'operatore umano, attraverso la ricerca di una vera e propria corrispondenza morfologica con il braccio robotico. L'operatore è posto davanti a una telecamera che inquadra i movimenti della mano e del braccio umano, e contemporaneamente l'agente robotico replica quanto visto ai fini di manipolare alcuni oggetti. Il sistema si compone di due flussi paralleli, uno dedicato alla mano e l'altro all'operatore nella sua interezza, composti sostanzialmente dagli stessi punti chiave: all'inizio OpenPose [6] si concentra sulla mano o sul corpo intero dell'operatore per estrarre i giunti principali dei due; segue la rete FrankMocap [31] che esegue in tempo reale una stima della posa e la ricostruzione del modello 3D di mano e corpo; infine

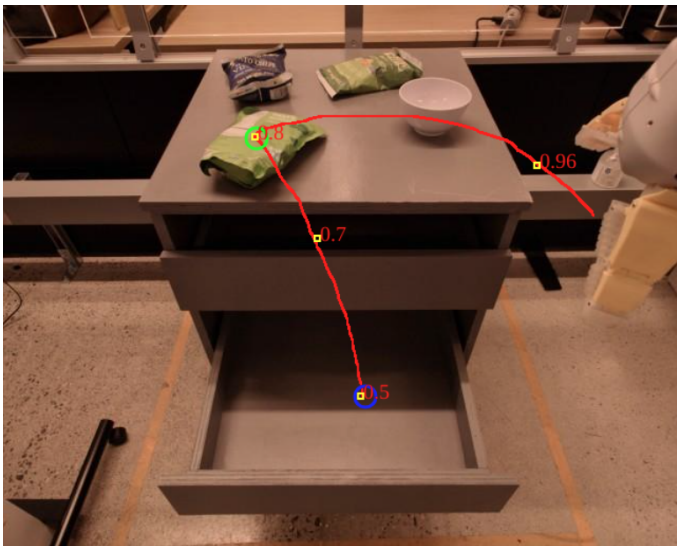


Figura 2.11: Rappresentazione della traiettoria e dei contatti in RT-Trajectory

due *retargeter* utilizzano queste informazioni per tradurle nella stessa posa ma nel dominio del robot, e posizionare correttamente sia i giunti dell'agente che l'end effector, che invece del classico gripper a due dita è fatto da un modello realistico di una mano robotica.

Il paper di Soares et al. [36] propone un pratico metodo basato su un sistema di VR per indirizzare la traiettoria di movimento di braccio robotico. Utilizzando il sistema Hololens 2 visualizza l'area entro cui si può muovere l'agente, mentre con la mano destra disegna la traiettoria desiderata nello spazio e con la mano sinistra impartisce i comandi di inizio e fine annotazione. Il lavoro è fortemente limitato dalla mancanza di interazione con oggetti dell'ambiente circostante.

La soluzione nota come ARC-LfD [24] propone l'utilizzo di un'interfaccia in AR per selezionare e salvare il movimento desiderato del braccio robotico: questi viene manipolato con la Kinesthetics, e quando si raggiunge un keypoint desiderato per il movimento si attuano gli opportuni comandi e si salva la posa raggiunta sia in termini di traslazione che di rotazione e azione del gripper. Il processo è interessante ma macchinoso dal punto di vista della raccolta dell'azione desiderata.

Infine, il lavoro di Nvidia noto come AR2-D2 [13] permette di facilitare la raccolta di dataset di dimostrazioni robotiche attraverso l'uso di una pratica applicazione per dispositivi mobili che integra un sistema di realtà aumentata. L'app proietta su una superficie piana un modello virtuale di un

robot realmente esistente, e rileva, in fase di registrazione, la posa nello spazio delle mani dell'operatore. Questi può utilizzare i comandi forniti per spostare l'end effector del robot virtuale e simulare la manipolazione dello stesso con oggetti realmente presenti in scena, con i quali nel frattempo l'operatore può effettivamente interagire. L'app è anche in grado di raccogliere una voxel map della scena corrente per il successivo utilizzo: difatti il sistema è stato testato per addestrare una versione del già citato PerAct [34], che necessità proprio delle voxel map della scena.

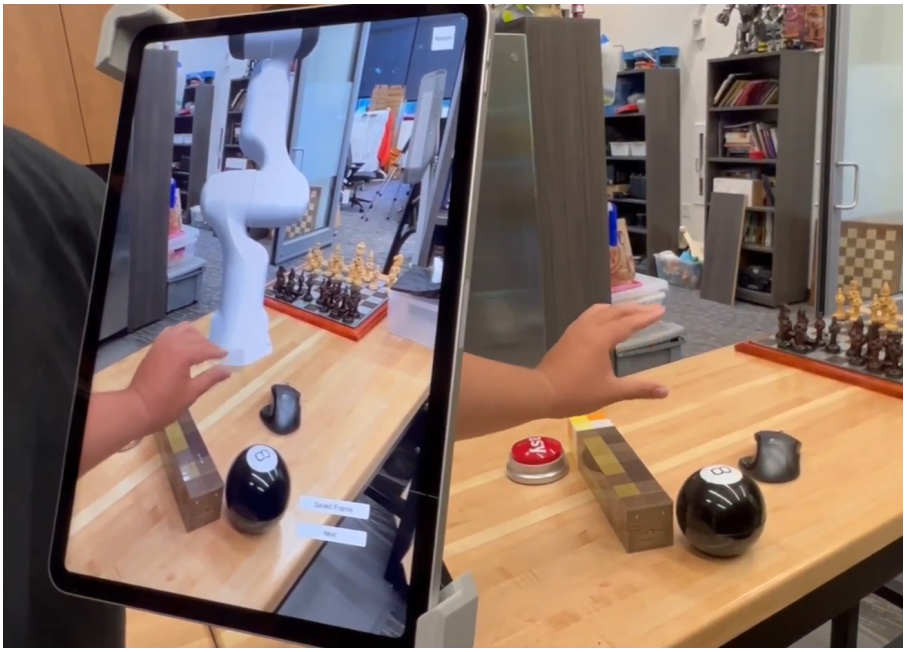


Figura 2.12: Applicazione mobile di AR2-D2 in funzione su uno scenario reale

Capitolo 3

Architettura e implementazione

Questo capitolo si occuperà di descrivere gli aspetti dettagliati dell'architettura sviluppata, al fine di comprendere in toto il flusso di lavoro che ha caratterizzato la soluzione messa a punto. Si partirà dalla definizione degli obiettivi prefissati, dell'ambiente di lavoro e della strumentazione utilizzata, e successivamente ci si addentrerà negli aspetti tecnici che costituiscono i vari punti del lavoro.

3.1 Obiettivi tecnici

L'architettura del sistema sviluppato mira a riprendere alcuni tratti delle filosofie implementative visti sinora nell'analisi dello stato dell'arte, e ad approfondire alcune delle tematiche più tecniche che riguardano la computer vision e il machine learning. Il sistema sviluppato vuole concentrarsi sulla fase di analisi degli input al fine di ottenere una panoramica quanto più approfondita e accurata possibile delle azioni che l'agente robotico tenterà di replicare, assieme a una valida comprensione dell'ambiente circostante e degli oggetti con cui questi andrà a interagire. Sebbene venga dato poco spazio all'aspetto puramente controllistico di questo task di manipolazione robotica, si ritiene che un focus maggiore sui dati e sugli input a disposizione del sistema possa rivelarsi un vantaggio interessante anche nell'implementazione pratica della soluzione.

3.2 Strumenti e setup

A partire dalle numerose scelte possibili in fatto di strumenti hardware e software, tra quelli maggiormente utilizzati in quest'ambito che sono stati analizzati nel capitolo precedente, è opportuno elencare quali di essi sono stati selezionati per l'architettura di questo lavoro.

3.2.1 Videocamera e librerie

Sebbene le potenzialità delle tecnologie moderne siano tali da permettere di scegliere delle semplici videocamere monoculari, la scelta è ricaduta su una videocamera che aggiungesse l'informazione di profondità, nello specifico la Azure Kinect DK sviluppata da Microsoft, una delle ultime della nota serie Kinect. Il dispositivo implementa una camera di profondita a 1MP con tecnologia "Time of Flight" (ToF), una camera RGB da 12MP, accelerometro e giroscopio digitale e una serie circolare di 7 microfoni.

Figura 3.1: Videocamera Azure Kinect DK scelta per il progetto

Il corrispettivo SDK sviluppato da Microsoft offre una lunga serie di strumenti software *out-of-the-box*, tra cui la generazione di point cloud e il tracciamento dei giunti per la stima della posa umana, oltre alle ovvie funzioni riguardanti la stima della profondità. Entrando nello specifico, per questo progetto è stata ampiamente utilizzata la libreria PyKinectAzure [16], ottima implementazione indipendente del SDK in Python, il quale è invece scritto prevalentemente in C++ e C. Tale libreria offre tantissime soluzioni ed

esempi pronti per replicare le esatte funzioni messe a disposizione in origine da Microsoft, scritte in maniera esaustiva e facile da implementare.

3.2.2 Framework principali

Per una descrizione completa del sistema, senza dettagliare ogni scelta, si riportano alcuni dei framework software principali utilizzati nel progetto: il codice delle implementazioni, gli script e le librerie più importanti utilizzano Python come linguaggio di programmazione, e per l'organizzazione del codice e la sua modularità si fa anche affidamento sui *notebook Jupyter*. Il framework di machine learning prescelto è stato PyTorch, molto diffuso e tutt'ora in crescita come numero di utenti, oltre che scelta base di tante soluzioni utilizzate. Per l'organizzazione delle librerie e delle dipendenze è stato utilizzato ampiamente Miniconda, versione leggera del genitore Anaconda. Come ambiente per l'addestramento delle reti, oltre che per l'uso e il test delle soluzioni software, è stato utilizzato il servizio online Google Colab. Per la parte di stima della posa dell'oggetto ci si è inizialmente serviti di Unity e della libreria Vuforia, passando per la libreria PyTorch3D e concludendo con l'ampio uso del pacchetto Open3D.

3.2.3 Area di lavoro, setup e oggetti

Come suggerito dagli obiettivi del progetto, questi si concentra sull'esecuzione di azioni che siano circoscritte a un ambiente di lavoro ben definito, di modo da minimizzare le sfide implementative proprie di circostanze mai viste prima dal sistema. Si è ritenuto opportuno, dunque, mantenere una certa costanza negli ambienti utilizzati nelle varie fasi del progetto.

Come punto di partenza, si opera su un tavolo classico, preso come riferimento per permettere al sistema di operare su una qualunque superficie pianeggiante. La telecamera di profondità è montata a bordo di un apposito sostegno, a un'altezza variabile per diversificare le prove: per dare un riferimento realistico, nella maggior parte delle configurazioni l'obiettivo della videocamera è posizionato tra i 20 e 40cm di altezza rispetto al piano del tavolo, a una distanza orizzontale da esso di circa 5cm e in posizione centrale rispetto al lato lungo del tavolo. Si noti che queste scelte non sono obbligatorie per il corretto funzionamento del sistema, ma rappresentano una base di partenza per comprendere il setup.

Per un task di manipolazione è ovviamente necessario includere degli oggetti che siano protagonisti delle azioni. Poiché quello considerato per

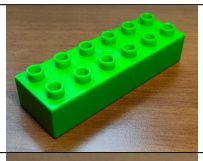
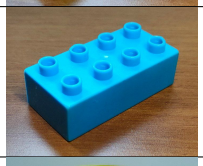
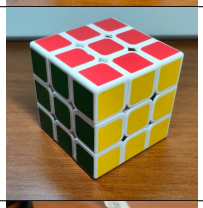
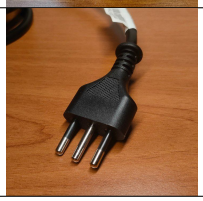
Immagine	**Oggetto**	**Dimensioni** (bounding box 3D in cm)
	Mattoncino verde	9.5 x 3 x 2
	Mattoncino blu	6.5 x 3 x 2
	Bicchiere di plastica	9 x 7.5 x 7.5
	Martello giocattolo	15.5 x 9 x 3.5
	Barattolo	6.5 x 6.5 x 5
	Cubo di Rubik	5.5 x 5.5 x 5.5
	Spina elettrica	7 x 4 x 1.5

Tabella 3.1: Oggetti utilizzati per le prove pratiche

questo lavoro non è un'approccio "in the wild", il set di oggetti utilizzati è stato scelto in anticipo, di modo da aggirare le necessità tecniche che questa filosofia avrebbe inevitabilmente portato con se, e che emergeranno con la descrizione del sistema. Il gruppo di oggetti è stato selezionato di modo che

fossero di facile reperibilità, sufficientemente leggeri per poter essere sollevati dal robot e con forme variabilmente semplici da afferrare, di modo da mettere alla prova il sistema in alcune occasioni e semplificargli il lavoro in altre.

La selezione di questi oggetti è anche figlia della scelta delle azioni che è possibile eseguire e riconoscere da parte del sistema: ogni oggetto è più o meno utilizzabile per semplici movimenti di *pick and place*, ma altre delle categorie che verranno elencate e spiegate nella sezione apposita avranno bisogno di oggetti specifici.

Il senso di questo set di oggetti, dunque, sarà più chiaro una volta definito anche il set di azioni possibili.

3.3 Flusso di lavoro

Si descrive adesso il flusso di lavoro che costituisce il cuore del progetto, di modo da ottenere una panoramica più ampia e comprendere al meglio gli approfondimenti dei paragrafi successivi. Con l'obiettivo di permettere a un robot di replicare un'azione umana nel setup precedentemente descritto, il fulcro del lavoro consiste nel registrare e analizzare a fondo tale azione, fornendo al robot stesso una quantità di informazioni sufficienti per operare correttamente. Elenchiamone i punti principali:

1. Registrazione del video: una volta preparato l'ambiente e decisa l'azione, questa viene eseguita dal dimostratore e registrata con l'uso della videocamera di profondità.

2. Analisi del contatto tra mano e oggetto: usando di un'apposita rete neurale, con codice opportunamente modificato, si rilevano gli istanti il cui la mano umana entra in contatto con l'oggetto d'interesse, assieme ai bounding box degli stessi

3. Rilevamento degli oggetti: partendo dall'informazione sulla posizione degli oggetti nel frame bidimensionale, si utilizza una rete di riconoscimento per identificare correttamente gli stessi e ridurre le possibilità di azioni che si potranno rilevare

4. Riconoscimento dell'azione: una rete neurale appositamente addestrata si occupa di identificare a quale classe appartiene l'azione effettuata, selezionando tra le otto possibili scelte e tenendo conto che alcuni oggetti non possono essere usati per alcune classi di azione

5. Stima di posa e posizione degli oggetti: con l'utilizzo di Open3D si analizza la nuvola di punti registrata durante la fase iniziale per stimare la posa e la posizione esatta dell'oggetto protagonista del movimento, attraverso opportune manipolazioni della point cloud

6. Esecuzione dell'azione: si comunicano al robot tutte le informazioni raccolte durante la fase di analisi, al fine di permettergli di ri-eseguire correttamente l'azione dimostrata. Quest'ultimo step, nel contesto del lavoro che si sta analizzando, è avvenuto solo in fase di test

3.4 Registrazione dell'azione

Come appena citato, dopo che l'ambiente è stato opportunamente preparato si inizia con la registrazione dell'azione prescelta. Da questa fase si ottengono tre file: il semplice video RGB, un "video di profondità", ovvero una sequenza di frame che rappresentano, per ogni pixel, la coordinata di profondità rispetto al sensore, e infine una serie di nuvole di punti della scena, che sono state raccolte per ogni fotogramma considerato. Queste ultime due informazioni sono salvate sotto forma di file `.npy`, ovvero di array della libreria Numpy, e sono riutilizzabili per le analisi successive.

Il video registrato vuole dimostrare l'esecuzione dell'azione per la sua successiva ripetizione, quindi suggerisce di dover riconfigurare l'ambiente com'era prima della stessa per permetter al robot di rieseguirla. Tale approccio può essere modificato per ottenere un flusso di lavoro diverso, ad esempio suggerendo un comando testuale, ma ai fini dell'analisi di un video dimostrativo è necessario riportare gli oggetti e l'ambiente in generale alle posizioni e alle configurazioni di partenza.

3.5 Interazione mano-oggetto

La prima rete neurale a entrare in gioco è quella impiegata per riconoscere l'interazione tra la mano del dimostratore e l'oggetto di interesse, i cui risultati in uscita sono di fondamentale importanza per il resto dell'analisi.

Il punto di partenza è una soluzione già citata nell'analisi dello stato dell'arte, ovvero il lavoro di Shan et al. del 2020 [32]. In questo paper i ricercatori hanno sviluppato due importantissimi strumenti per l'analisi di video di interazioni umane con oggetti: il primo è l'ampio e completo dataset di 100DOH (100 Days Of Hands), una raccolta di oltre 27.000 video

appartenenti a 11 categorie differenti, e il secondo è una rete in grado di identificare quando vi è contatto tra una mano umana e un oggetto in video.

La rete costituisce ovviamente la parte più interessante di questo progetto. Essa è costruita a partire dal modello di una Faster-Recurrent Convolutional Neural Network (F-RCNN) ed è stata addestrata con il dataset precedentemente citato. Il compito della rete è quello di rilevare, all'interno di un video di input, i bounding box sia delle mani che dei vari oggetti in scena. La presenza o meno di contatto è affidata a una stima della distanza dei due bounding box, che suggerisce la possibilità di errori nel momento in cui una mano si sovrappone a un altro oggetto in un frame del video, pur trovandosi a debita distanza da lui o mancando di interagirvi. Questi possibili errori hanno creato la necessità di modificare il codice per evitare fraintendimenti nella scena.

Nell'uso effettivo, la rete effettua le sue predizioni sul video RGB: questi viene suddiviso nei singoli frame con l'uso della libreria `ffmpeg`, e la cartella generata funge da percorso di input. Una volta caricati i pesi del modello pre-allenato, la rete effettua i suoi calcoli analizzando le immagini una per una e producendo in output le corrispettive immagini annotate, segnalando i bounding box precedentemente citati e lo stato delle mani. Per maggiore precisione, la rete annota se le mani identificate siano destre o sinistre, se stiano effettuando o meno contatto e, in caso positivo, se questi sia con la persona stessa, con un oggetto mobile o con uno fermo (ad esempio con del mobilio).

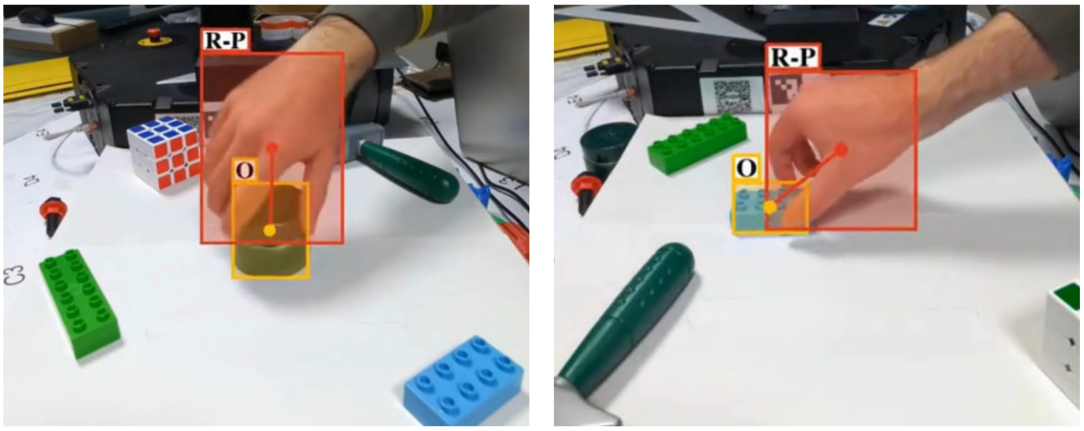


Figura 3.2: Alcuni esempi di rilevamento di contatto fatti dalla rete

Per permettere di effettuare un analisi successiva di quanto rilevato dalla

rete, e dunque non rallentare la stessa nel processo, il codice è stato modificato quanto segue: viene definita una nuova classe python `Detection` che raccoglie: l'identificativo numerico di un certo frame, i bounding box della mano e dell'oggetto in scena, i pixel centrali degli stessi, la distanza tra gli stessi e la semi-diagonale del bounding box della mano.

Un primo controllo vede se la rete abbia rilevato la presenza nel frame sia di una mano che di un oggetto qualsiasi. Stabilita una certa soglia di attendibilità, si controlla se sia stato rilevato una probabilità di contatto tra i due maggiore della soglia. In caso positivo, il frame corrente è salvato in una nuova istanza della classe `Detection` e unito a un vettore contenente tutti i fotogrammi d'interesse, che chiamiamo `detection_list` e che verrà salvato infine come file `.pkl`, ovvero in formato python pickle. Tale vettore costituisce l'insieme dei fotogrammi dove è stato rilevato un contatto non futile tra una mano e un oggetto qualsiasi della scena, ma si vedrà che non tutti i frame considerati sono davvero importanti nell'azione: alcuni includono falsi contatti con oggetti non facenti parte dell'azione, altri inc

Proprio per via della necessità di filtrare i dati irrilevanti, in un secondo momento tale vettore `detection_list` viene analizzato, e per prima cosa si ricerca quale sia il primo frame di contatto tra quelli d'interesse. Per stabilire se il contatto sia stato di nostro interesse (ovvero che non sia un contatto falso dato dalla sovrapposizione tra la mano e un oggetto d'ambiente) si utilizzano alcune euristiche:

1. è considerato valido un contatto per cui la distanza tra i centri dei due oggetti è inferiore alla semi-diagonale del bounding box della mano stessa, ovvero c'è una sufficiente sovrapposizione dei bounding box

2. è considerato il contatto che cerchiamo se, dal momento in cui c'è un buon candidato, nei successivi dieci frame almeno per la metà di questi il contatto esiste ancora. Ciò significa che, in questo intervalli di fotogrammi, verrà analizzata la dimensione dei bounding box dell'oggetto in ognuno di essi, e calcolato il valore di *Intersection over Union* (IoU) rispetto al bounding box candidato, definito come:

$$IoU = \frac{Intersezione\_delle\_aree}{Unione\_delle\_aree}$$

   Questo valore indica la "somiglianza" tra i due bounding box considerati, che dev'essere maggiore di `0.5`, permettendoci di capire se il focus è rimasto sullo stesso oggetto o se si è spostato su un altro in scena.

Qualora non ci fossero abbastanza frame con un valore di IoU sufficiente, molto probabilmente il candidato era un oggetto non coinvolto in un vero contatto.

Quest'ultimo controllo è necessario al fine di sopperire ad alcune imprecisioni della rete, per cui in determinati frame non viene rilevata correttamente alcuna interazione, sebbene sia effettivamente in atto.

Si noti che tali euristiche non costituiscono una verità assoluta, ma sono frutto di una certa serie di prove che ha fatto notare le imprecisioni appena elencate. Possono non rilevarsi ottimali in caso di azioni e configurazioni specifiche, motivo per cui devono essere facili da modificare a seconda del caso. La ricerca di un metodo più puntuale è sicuramente importante per permettere al sistema di lavorare con quante meno modifiche possibile.

Se il controllo fornisce esito positivo, il frame analizzato è considerato il primo del contatto centrale del nostro video. In maniera simile si valuta l'ultimo frame di contatto, eseguendo la stessa ricerca utilizzando la lista di frame candidati invertita: in tal modo si può evitare di includere contatti fasulli che sono stati rilevati mentre la mano usciva dall'inquadratura, considerando solo il frame dove la mano ha effettivamente lasciato l'oggetto protagonista del movimento.

Per stimare, invece, la traiettoria che segue l'oggetto si possono identificare un certo numero di frame intermedi nei quali analizzare la posizione dello stesso, utile nei casi in cui il percorso che ha compiuto non sia stato "dritto" (ovvero quando la traiettoria proiettata sul piano del tavolo non disegni una retta). In tal caso, si può specificare un intero chiamato `numero_frame_intermedi`, che permetterà di selezionare tra i fotogrammi compresi tra quello iniziale e quello finale tanti quanti specificati da tale variabile, e distanziati tra di loro allo stesso modo. A titolo d'esempio, se il primo frame di contatto è alla posizione `20` mentre l'ultimo alla posizione `80`, indicando `numero_frame_intermedi = 3` verranno selezionati anche i frame `35, 50, 65`, ovvero tre fotogrammi intermedi a eguale distanza reciproca. Analogamente, con `numero_frame_intermedi = 4`, verrebbero selezionati i frame `32, 44, 56, 68`, e così via con l'aumentare o il diminuire della variabile. In diversi casi, per ottenere una descrizione sufficientemente precisa dell'azione, è sufficiente anche un singolo frame intermedio.

Alla fine del processo di analisi otteniamo, dunque, un vettore composto da tuple: in ognuna di essa è presente il numero del frame identificato come d'interesse e il corrispettivo bounding box dell'oggetto al suo interno, tutto

ordinato temporalmente. Questi dati saranno utilizzati nella fase successiva di rilevamento della posizione degli oggetti.

3.6 Rilevamento di posizione e classe degli oggetti

Dalle informazioni ottenute in precedenza possiamo ricavare dove si trovi approssimativamente l'oggetto protagonista dell'azione all'interno dei singoli fotogrammi d'interesse, ovvero quelli di inizio e fine azione e quelli intermedi che determinano la traiettoria della stessa. Questi dati sono dunque utilizzate per acquisire due informazioni importanti: la posizione nello spazio dell'oggetto durante tutto il movimento e quale sia questo oggetto, cioè la sua classe di appartenenza.

Per estrapolare questo tipo di nozione torna utile il "video di profondità" che abbiamo registrato in precedenza e salvato come file `.npy`. Questo vettore contiene sostanzialmente dei frame di profondità, costituiti esattamente come se fossero un fotogramma RGB, ma con la differenza che il valore numerico associato a ogni pixel non rappresenta un'informazione di colore, bensì la distanza del pixel stesso rispetto al relativo sensore sulla camera. È ovvio che tale informazione di profondità possa essere, se necessario, ri-mappata su una scala di colori, per estrarne un'immagine RGB che visualizzi graficamente i punti più vicini e più lontani dalla camera stessa.

Il flusso di lavoro sarà, dunque, il seguente: si usano i bounding box approssimativi rilevati dalla rete di hand-object interaction per circoscrivere, sia sul frame di profondità che sul relativo frame RGB, l'area dov'è presente l'oggetto. Tale area viene ampliata manualmente di un fattore pari al 50% della lunghezza di ogni lato, di modo da ovviare a eventuali imprecisioni di rilevamento della rete precedente, e permettere alla successiva rete di lavorare su un area sufficientemente grande per identificare appieno l'oggetto.

3.6.1 Riconoscimento dell'oggetto

Una volta identificata l'area di interesse in un certo fotogramma si può procedere con l'identificazione accurata della classe dell'oggetto stesso, assieme al suo bounding box definitivo e preciso. Tale compito spetta quasi d'obbligo a una delle reti più utilizzate e di maggior precisione in questo campo, ovvero la già citata YOLO (*You Only Look Once*) [30], impareggiabile dal punto di vista delle prestazioni e dell'accuratezza. Nello specifico, in una prima fase è

stata utlizzata la versione nota come YOLOv8 [20] e sviluppata da Ultralytics, una delle più nuove e considerata come vero e proprio stato dell'arte, ma è risultata non sufficiente a identificare tutti gli oggetti del progetto. Sebbene tale framework fornisca una gran serie di modelli pre-addestrati su diversi dataset più o meno ampi, nessuno di essi include tutte le classi che sono parte di questo lavoro: a titolo di esempio, la versione base di YOLOv8 è allenata sul dataset COCO (*Common Objects in Context*) [22], che include 80 classi di oggetti differenti e tra cui sono presenti alcune di interesse, ma purtroppo costituiscono una minoranza.

Si è reso necessario ricercare un metodo per identificare qualunque oggetto tra quelli elencati in precedenza. Tale necessità suggeriva l'addestramento (o il fine-tuning) ad-hoc di una rete YOLO, o di un altro modello con pari capacità, che avrebbe previsto un processo di raccolta dati e annotazione dalle tempistiche non indifferenti. La ricerca di una soluzione valida ha portato, contemporaneamente, alla scoperta dei modelli di riconoscimento cosiddetti *zero-shot*: la loro idea di base è quella di permettere di identificare la classe e il bounding box di un oggetto senza essere stati allenati con un dataset che li includesse, bensì riconoscendoli attraverso una descrizione testuale degli stessi.

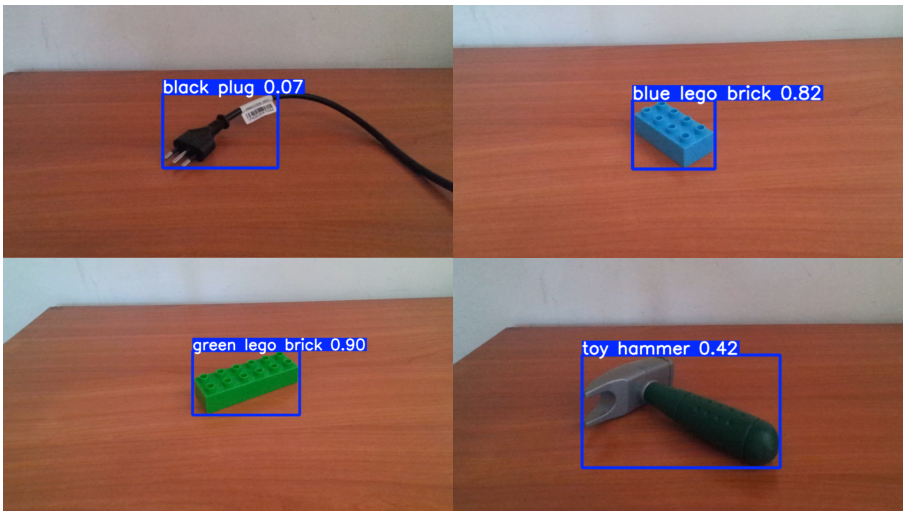


Figura 3.3: Esempi di rilevamenti di YOLO-World sugli oggetti prescelti

Sono state testate alcune soluzioni appartenenti a questo campo e considerate lo stato dell'arte, tra cui i già citati modelli noti come OWLv2 e GroundingDINO. La scelta è però ricaduta su YOLO- World, basato sulla versione 8 del suo semi-omonimo predecessore e selezionato in quanto uno

dei migliori in termini di velocità di calcolo, mentre le altre reti citate non offrivano prestazioni confrontabili in termini di rapidità. Il principio di funzionamento dei modelli di open vocabulary detection è abbastanza ricorrente, con poche differenze nella struttura del modello alla base: si basa sull'uso di reti di encoding tra testo e informazione visiva, come la già citata CLIP, per trasformare le descrizioni testuali degli oggetti in embedding in un determinato spazio vettoriale. In contemporanea, la rete identifica le aree che possono contenere un'oggetto interessante, trasformando anche quest'informazione visiva in un embedding nello stesso spazio e confrontando i due dati: nel caso in cui ci sia una corrispondenza, viene evidenziato l'oggetto rilevato assieme alla percentuale di sicurezza con cui la rete lo ha identificato. Ciò significa che nella fase pratica di questo lavoro viene fornita al modello la descrizione testuale di tutti gli oggetti selezionati per la manipolazione, e YOLO-World si occupa di determinarne il bounding box corretto da passare alla fase successiva del flusso di lavoro.

3.6.2 Da bounding box a posizione 3D

Una volta ottenuto il bounding box preciso dell'oggetto protagonista dell'azione è necessario identificarne la posizione nello spazio di lavoro. Senza dubbio, l'uso della telecamera di profondità semplifica tale processo, e la libreria PyKinectAzure offre gli strumenti adatti. In linea di massima, in questa fase è opportuno stimare una posizione generica dell'oggetto, al meglio identificabile in quella del centro del suo bounding box, poiché la maggior parte degli oggetti considerati non ha forme complesse o per le quali tali centro non corrisponda a un proprio punto. Come vedremo più avanti, tale soluzione non è definitiva e l'accuratezza della posizione sarà migliorata durante la stima della posa dell'oggetto.

A rigore, ciò non è ancora sufficiente per comprendere la posizione dell'oggetto nell'ambiente, poiché è necessario trovare la coordinata del suo centro non rispetto alla telecamera (la cui posizione può cambiare facilmente), bensì all'ambiente in sé. Nasce il bisogno di stabilire un sistema di riferimento dell'area di lavoro, e dunque innanzitutto la sua origine fisica.

Viene utilizzata una soluzione molto diffusa per identificare dei riferimenti unici all'interno di una scena, i cosiddetti ArUco markers (*Agumented Reality University of Cordoba markers*) [15]: si tratta di figure quadrate, assimilabili a codici a barre o QR code, composte da uno spesso bordo nero e da una griglia centrale, di dimensioni variabili, fatta da un'alternanza di quadrati bianchi e neri. Il pattern generato da questa alternanza costituisce l'identificativo

numerico del singolo marker, che non è universale: per ogni applicazione per cui vengono utilizzati viene generato un "dizionario", che associa a ogni pattern il suo identificativo, rendendo questi accoppiamenti validi per il solo caso corrente.

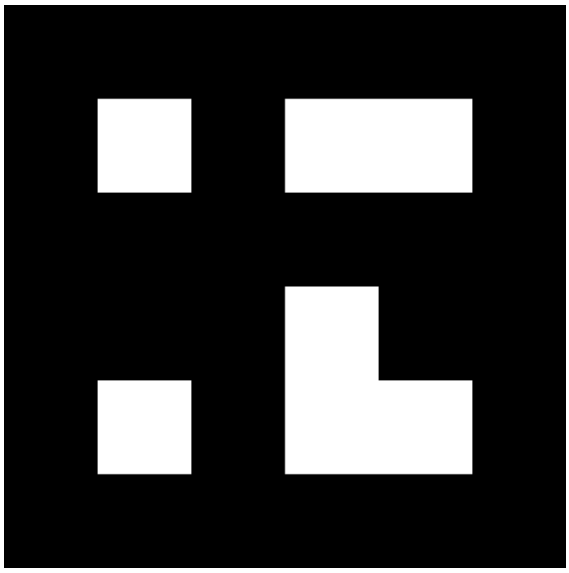


Figura 3.4: Esempio di marker ArUco

Una volta generato il dizionario, si procede alla stampa dei marker su dei fogli di carta e al loro posizionamento nella scena: per il caso d'uso corrente è sufficiente il piazzamento di un solo marker in un punto nel quale possa essere facilmente inquadrato dalla telecamera (sebbene, a rigore, il sistema sia in grado di individuarlo anche a numerose angolazioni), e in modo tale che possa semplificare il calcolo delle coordinate nello spazio di lavoro del robot. Difatti, come vedremo nel caso delle prove pratiche eseguite sul campo, tale identificatore verrà posizionato lungo l'asse x del sistema di riferimento del robot, a una distanza nota dallo stesso e parallelamente all'asse z (indirizzato verso l'alto).

L'origine del nuovo sistema di riferimento coinciderà con l'angolo in alto a sinistra del marker, e avrà la seguente struttura: la direzione dell'asse `x` sarà perpendicolare al foglio del marker e da esso uscente, quella dell'asse `y` andrà alla sua sinistra parallelamente al suo foglio, mentre quella dell'asse `z` sarà perpendicolare e uscente dal piano di lavoro. La FIGURA fornisce un esempio pratico.

Elenchiamo, dunque, i punti principali del processo di stima della posizione:

- All'inizio dell'analisi viene individuata la posizione del marker ArUco rispetto alla telecamera, che si suppone fisso (di modo da non doverla ricercare anche nei frame successivi) e che verrà usato come punto di riferimento

- A partire dal bounding box dell'oggetto, si seleziona facilmente il pixel corrispondente al suo centro, di cui si estrapola il valore di profondità rilevato in fase di registrazione. Esso viene fornito alla funzione `convert_2D_to_3D` della libreria apposita (assieme alla coordinata del pixel centrale e ad altri valori di calibrazione della camera) per trasformare il semplice valore di profondità in una più completa coordinata tridimensionale, che indica la posizione del punto rispetto al sensore della telecamera

- Conoscendo la posizione del marker e del centro dell'oggetto, entrambi rispetto al sistema di riferimento della telecamera, si può calcolare la posizione dell'oggetto rispetto al marker con una semplice sottrazione delle loro coordinate. Il risultato sarà relativo a un nuovo sistema di coordinate centrato nel marker (come desiderato), ma gli assi di tale sistema saranno ruotati con le stesse angolazioni di quelli della telecamera, dunque non in linea con quanto cercato per l'ambiente di lavoro.

- Si trova manualmente l'angolazione della telecamera rispetto al terreno, e successivamente si calcolano le matrici di rotazione necessarie ad allineare gli assi del momentaneo sistema di riferimento del marker con quelli del robot. Se la camera è ruotata di un angolo `- alpha` attorno a `x`, si calcolerà prima una matrice di rotazione di `alpha + 90°` attorno a esso per allineare gli assi `z`, e poi una matrice di rotazione di `180°` attorno a `z` per allineare i restanti, come descritto nella figura 3.5

La principale problematica rilevata nel calcolo della posizione dell'oggetto consiste nell'accuratezza del metodo appena descritto. Il calcolo attuale propone di considerare come posizione dell'oggetto quella relativa al centro del suo bounding box, che non costituisce un'euristica valida in tutti i casi possibili: da un lato è possibile incontrare oggetti con forme, angolazioni o semplici posizionamenti inusuali, per i quali il centro del bounding box non ricada affatto sull'oggetto stesso (più probabilmente sul piano di lavoro); dall'altro, anche per oggetti che occupino a sufficienza il loro bounding box, il più delle volte il relativo centro ricade su una parte della superficie frontale dello stesso, o nel caso di oggetti concavi anche su una parte della superficie

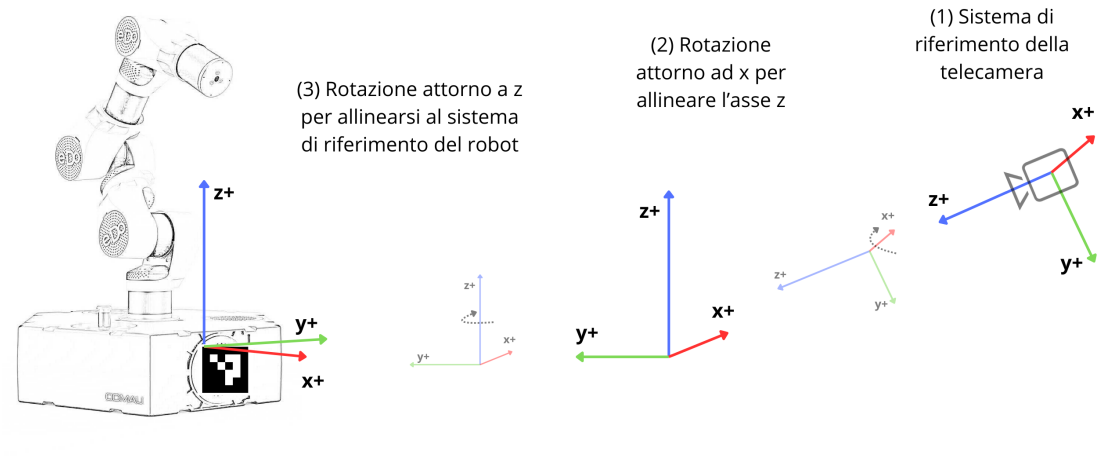


Figura 3.5: Rappresentazione delle rotazioni calcolate per allineare i sistemi di riferimento

interna, e non dunque su quello che dovrebbe essere il loro punto centrale in un senso tridimensionale. Conoscendo l'oggetto che stiamo analizzando possiamo proporre delle euristiche personalizzate, tentando di "arretrare" manualmente il punto rilevato sulla base dello stesso (ad esempio, di 1cm nel caso del mattoncino o di 3cm per un vasetto) ma tutto ciò non generalizza a sufficienza: potremmo incorrere in pose inusuali dell'oggetto o angolazioni diverse della camera per cui questa modifica "manuale" non risulti sufficientemente efficace.

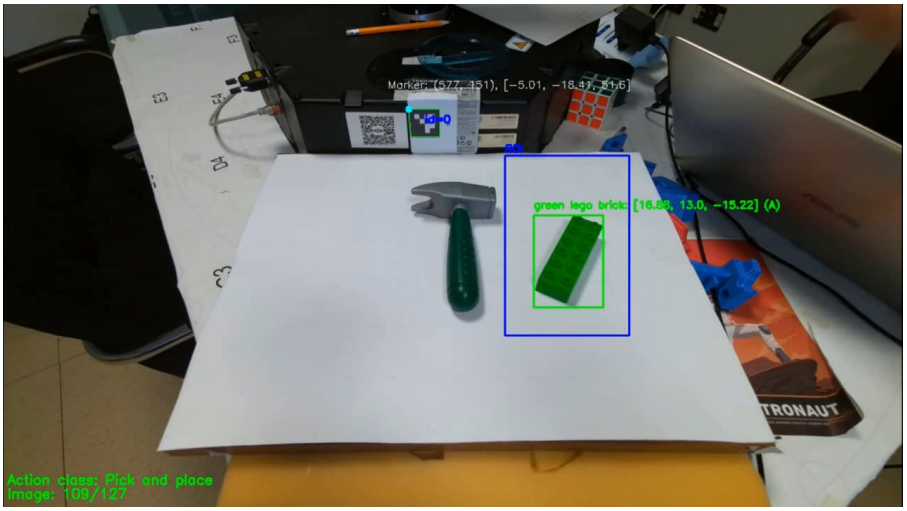


Figura 3.6: Esempio di frame che dimostra il riconoscimento dell'oggetto e la prima stima della posizione

Tuttavia, la posizione individuata nello spazio con questo metodo fornisce una prima indicazione importante: come si vedrà nella sezione dedicata alla stima della posa dell'oggetto, essa verrà integrata con un metodo più efficace, basato sulle nuvole di punti, che fornirà una posizione dell'oggetto più precisa, mentre questa prima stima rimarrà a disposizione nel caso in cui il secondo metodo risulti impreciso.

3.7 Riconoscimento dell'azione

Come discusso nei capitoli precedenti, la piena comprensione di ciò che ha eseguito il dimostratore passa anche dal riconoscimento dell'azione compiuta, o meglio dall'identificazione nella stessa in una delle classi prescelte.

Al fine di lavorare con un set di azioni esaustivo sono state selezionate otto tipologie diverse delle stesse, che rappresentassero movimenti e interazioni piuttosto semplici e basilari, ma che, se unite insieme, possono generare gesti compositi e descrivere anche azioni piuttosto complesse: il riconoscimento e la replica di dimostrazioni lunghe e varie necessita della piena comprensione dei movimenti-base di cui sono costituite. Dunque, le classi selezionate sono le seguenti:

- *Push*: rappresenta la spinta di un oggetto sul tavolo allontanandolo dal dimostratore, senza necessità di afferrarlo e, dunque, di sollevarlo dal piano
- *Pull*: parimenti, rappresenta l'azione di tirare un oggetto verso il dimostratore. Anche qui può essere necessario o meno afferrare l'oggetto, ma comunque non verrà sollevato
- *Pick-and-place*: la classica azione, ben nota in robotica, che prevede di afferrare un oggetto, sollevarlo e spostarlo in un altra posizione
- *Rotate*: il capovolgimento di un oggetto sul posto, invertendone il verso del proprio asse `z`
- *Plug*: verbo correlato all'azione di inserire una spina in una presa elettrica, oppure similmente un cavo nell'apposita porta d'ingresso. È più in generale il gesto di inserimento di un'oggetto in un altro
- *Unplug*: opposto di plug, è lo scollegamento di una spina elettrica o di un cavo, o il generico disinserimento di un oggetto da un altro

- *Screw*: rappresenta il gesto dell'avvitamento, inteso come rotazione in senso orario, proprio di oggetti come una vite, un tappo di bottiglia o una valvola

- *Unscrew*: opposto di screw, è il gesto dello svitamento, inteso come rotazione in senso antiorario, della stessa tipologia di oggetti

Allo stato dell'arte non esistono soluzioni di *action recognition* nel mondo del deep learning che siano in grado di riconoscere tutte queste categorie, o comunque non nei termini e nei modi desiderati: i principali dataset utilizzati per gli addestramenti possono contenere classi come *push* o *pull*, ma queste non si concentrano su video di manipolazioni di oggetti su un piano di lavoro, com'è importante che sia nel caso corrente. Si è ritenuto necessario, dunque, raccogliere un dataset ad-hoc di video contenenti le tipologie di dimostrazioni cercate, al fine di eseguire il fine-tuning di una rete di riconoscimento di azioni che le potesse identificare correttamente.

La rete che è stata selezionata per tale compito è denominata I3D (*Inflated 3D ConvNet*) [7], un modello specializzato nell'action recognition considerato di altissimo livello allo stato dell'arte. Esso si compone di due *stream* differenti, ovvero due flussi separati che analizzano i dati in input e forniscono due risultati distinti, che verranno combinati per generare il risultato definitivo. Ogni flusso è costituito dalla combinazione di una rete convolutiva "tridimensionale" e di una serie di moduli noti come *inception*: la rete risulta in grado di espandere le capacità dei modelli tradizionali di ConvNet (che operano sulla singola immagine 2D) al fine di includere anche la dimensione temporale, e dunque analizzare i cambiamenti che intercorrono nello scorrere dei fotogrammi.

I due flussi distinti analizzano dati differenti in `batch` di 64 immagini, dove il primo si occupa dei classici frame RGB mentre il secondo del relativo flusso ottico: con esso intendiamo una tipologia di dato grafico che descrive il movimento relativo dei pixel al passaggio tra due fotogrammi successivi, in termini di vettore di probabile spostamento. Ciò che otterremo, visivamente, sono delle immagini in scala di grigi che evidenziano le aree soggette a movimento durante lo scorrere del video. Questa informazione risulta essere di grandissima importanza nella distinzione delle classi da riconoscere, soprattutto se esse possono essere molto simili tra di loro: in questo progetto, è il caso della coppia di classi *screw-unscrew*, dove a cambiare è principalmente il modo in cui ruotano le dita delle mani, ma anche di quelle *plug-unplug* e *push-pull*, sebbene abbiano direzioni di movimento diverse. Dal punto di vista pratico,

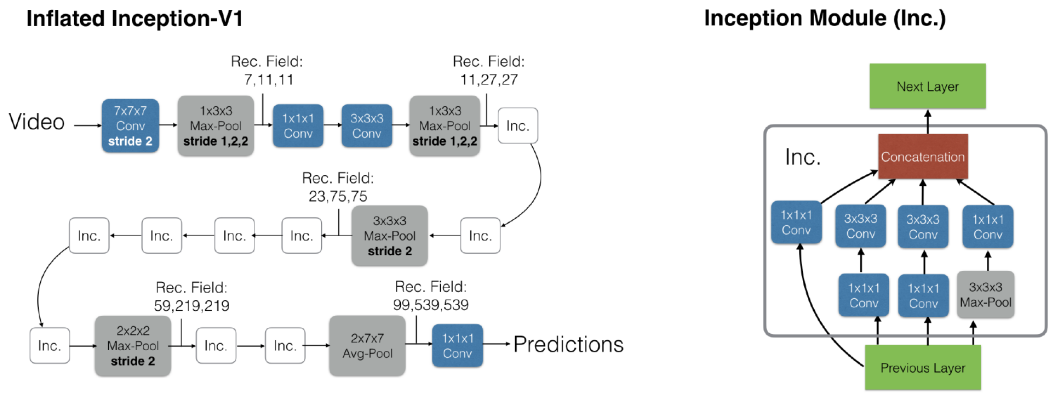


Figura 3.7: Architettura del singolo flusso di I3D

l'algoritmo utilizzato per il calcolo del flusso ottico è quello di Farneback, che risulta al contempo efficiente e accurato.

3.7.1 Raccolta del dataset

Le classi di azioni che abbiamo elencato in precedenza possono essere eseguite in video brevi, generalmente lunghi dai 2 ai 5 secondi e con durata variabile a seconda dell'entità del video stesso. In assenza di una regola generale sulla quantità di video necessari per eseguire un fine tuning sufficiente, si è stimato un valore di circa 100 video per classe (anche lievemente maggiore considerando la suddivisione del dataset per allenamento e validation) che ha portato a un totale di oltre 800 video di durata complessiva superiore a un'ora di contenuto.

I video sono stati registrati usando un setup fisso, posizionando un tavolo contro una parete bianca per diminuire le interferenze circostanti: la raccolta è avvenuta seguendo alcuni euristiche comuni e altre diverse a seconda della classe, al fine di massimizzare la varietà di video raccolti. Questi sono stati ripresi in condizione di luce differente, sia diurna che artificiale, e includono la presenza delle sole mani del dimostratore e non del resto del corpo.

In generale, la telecamera è stata posizionata per avere almeno tre punti di vista differenti, ovvero dall'alto (circa 40cm sopra il livello del tavolo), dal basso (circa 20cm) e lateralmente (40cm sopra al tavolo e un angolazione di circa 45º verso sinistra). Per ogni classe sono stati utilizzati almeno sette oggetti differenti, inclusi quelli selezionati per gli esperimenti e altri non precedentemente citati ma simili, e i video includevano sia la presenza

Classe	Numero di video	Durata totale (s)
Push	109	493
Pull	105	422
Pick and place	106	416
Rotate	106	289
Plug	110	423
Unplug	111	428
Screw	105	775
Unscrew	105	746
Totale	**857**	**3992 (1h 6m 32s circa)**

Tabella 3.2: Riassunto dei video registrati

di un solo oggetto in scena che quella di altri, utile in casi come quello di pick and place per evidenziare lo scavalcamento di altri oggetti nel movimento.

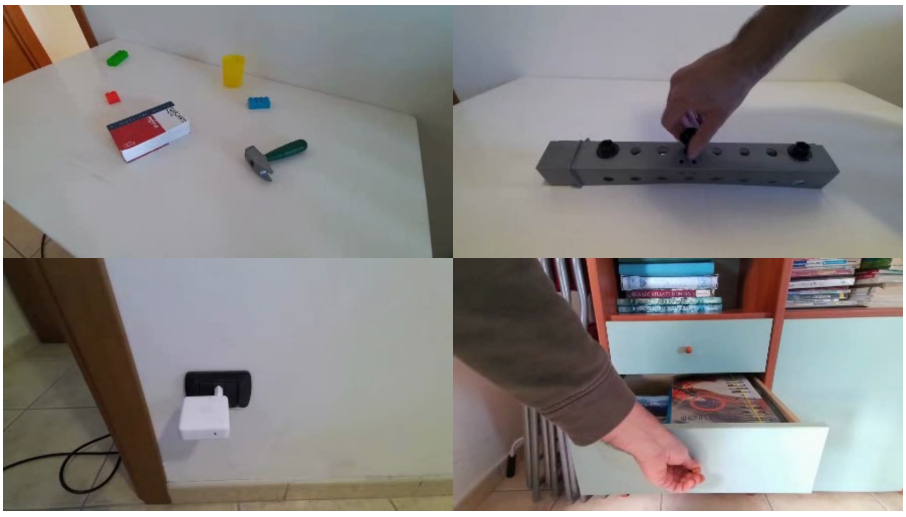


Figura 3.8: Frame tratti dal dataset raccolto, che evidenziano varie ambientazioni e angolazioni di registrazione

Per alcune classi sono stati raccolti dei video anche al di fuori del setup del tavolo, di modo da permettere alla rete di concentrasi sul movimento delle

mani e dell'oggetto: in casi come *push* e *pull* vi sono video dell'apertura e chiusura di cassetti e porte, sempre con angolazioni diverse, mentre per plug e *unplug* c'è l'esecuzione del gesto utilizzando sia prese elettriche a muro che chiavette usb e cavi collegati a computer non situati sul tavolo principale.

I video sono stati raccolti facendo particolare attenzione a includere solo il gesto considerato, escludendo il più possibile i frame iniziali e finali senza movimento che avrebbero potuto apportare del rumore aggiuntivo dannoso per l'addestramento. La loro risoluzione di registrazione è 1280x720, poi successivamente ridotta con un apposito script fino a 480x360. E' seguita la loro trasformazione da video interi a cartelle di singoli fotogrammi, e sempre con del codice apposito in questa fase preliminare è stato calcolato il flusso ottico, anch'esso tramutato in singoli frame e salvato nello stesso modo.

3.7.2 Fine-tuning

Come accennato, è stato scelto di effettuare il fine-tuning di una rete pre-addestrata: questa decisione nasce dal fatto che l'allenamento da zero di un modello di tale portata abbia bisogno di una mole di dati non indifferente, mentre con tale approccio si può sfruttare ciò che la rete ha imparato da un dataset molto ampio e diversificato, lasciando il compito del riconoscimento delle otto classi scelte ai soli layer finali della stessa.

Per eseguire il fine tuning, si parte con l'elaborazione del dataset: questi è suddiviso nelle due parti di allenamento e validation con un rapporto di circa 80:20, creando dei textttdataloader specifici che processano ogni frame portandolo a una dimensione di 224x224, richiesta dalla rete per una corretta elaborazione. Contemporaneamente, tali dataloader applicano delle tecniche di *data augmentation* per ampliare lo spettro di dati con cui lavora la rete: nello specifico, si tratta di ritagli casuali di aree dell'immagine, ribaltamenti dei frame lungo l'asse orizzontale e cambianti nei colori degli stessi.

A questo punto si può definire il modello della rete, utilizzando l'implementazione Pytorch di I3D proposta da citazione alla repo, che fornisce contemporaneamente i pesi del suo allenamento eseguito sull'esteso dataset video di Kinetics 400. Tale modello viene dunque duplicato per includere il flusso RGB e quello del flusso ottico, su di essi vengono modificati gli strati finali completamente connessi per indicare le 8 classi riconoscibili, e vengono caricati i pesi dell'addestramento facendo attenzione a "congelare" la maggior parte degli strati della rete, meno gli ultimi tre livelli noti come `logits`, `Mixed_5b` e `Mixed_5c`: tale scelta è stata frutto di diversi tentativi,

che l'hanno identificata come la migliore dal punto di vista del rapporto tra velocità di addestramento e accuratezza del risultato ottenuto.

Per la fase di fine tuning vera e propria, è stato scelta la funzione di perdita *cross entropy*, la migliore per task di classificazione con numero di classi ben definito, e l'ottimizzatore `Adam`. Le epoche di allenamento sono state 12, poiché questi è stato manualmente interrotto dopo aver raggiunto dei valori di accuratezza e perdita stabili a partire dalla decima epoca in poi. Al termine di ognuna di esse, il sistema eseguiva la fase di validazione con l'apposito dataset per stimare l'andamento e la bontà dell'allenamento.

In fase di utilizzo vero e proprio sul campo, si fornisce alla rete il semplice video RGB dell'azione registrata, e lo script apposito creerà il corrispettivo flusso ottico in tempo reale, fornendo entrambe le informazioni in ingresso al modello e producendo un'etichetta di riconoscimento della dimostrazione considerata.

I risultati dell'allenamento sono riportati nell'apposita sezione del capitolo successivo.

3.8 Posa degli oggetti e posizione più accurata

Un buon sistema per l'analisi video ai fini della manipolazione robotica non può prescindere da una fase dedicata alla stima della posa dell'oggetto considerato. Pur avendo definito, nel lavoro che si sta descrivendo, un sistema circoscritto e un setup ben definito (sia in termini di area di lavoro che di oggetti considerati), è improbabile pensare di operare all'interno dell'ambiente senza conoscere la posa degli oggetti protagonisti del movimento: a partire dalla posizione nello spazio, che fornisce un'indicazione già importante, bisogna individuare la rotazione tridimensionale dell'oggetto perlomeno negli istanti di inizio e fine azione, di modo da fornire le indicazioni corrette al robot per approcciare lo stesso e per riposizionarlo in modo coerente. Questa fase del progetto appartiene a un ambito di ricerca ben definito, denominato "6 DOF Object Pose Estimation" e ritenuto un campo ancora in attiva ricerca di soluzioni considerabili come stato dell'arte: una parte di esse sono state elencate nell'apposito capitolo, mentre qui si analizzeranno le implementazioni sperimentate per il progetto corrente.

Per affrontare un compito di questo tipo bisogna innanzitutto comprendere a che punto del lavoro si è arrivati e quali informazioni siamo in grado di ottenere dal sistema: si conoscono con buona certezza sia gli oggetti con cui possiamo interagire sia l'oggetto effettivamente protagonista del movimento,

sappiamo riconoscere l'azione dimostrata e conosciamo la posizione dell'oggetto con una precisione sufficiente ma non ottimale. Abbiamo a disposizione sia le informazioni RGB che quelle di profondità, e, poiché il set di oggetti è ben definito, possiamo ottenerne il modello tridimensionale con relativa facilità.

A ciò vanno aggiunte delle considerazioni dal punto di vista prestazionale: per raccogliere queste informazioni, i dati in input devono passare attraverso l'analisi di tre reti neurali costituite da una grande quantità di parametri, e che necessitano di un certo tempo per eseguire correttamente il loro compito. Sebbene tali calcoli possano essere parallelizzati con dell'hardware apposito, non risulta una scelta saggia appesantire la pipeline di lavoro con ulteriori step di machine learning e calcoli complessi. Inoltre, la conoscenza delle classi di oggetti manipolabili dal sistema, la stima approssimativa della posizione dell'oggetto e l'informazione di profondità che siamo in grado di ricavare suggeriscono di ricercare un approccio di "computer vision classica", evitando il mondo delle reti neurali apposite che non hanno, tra l'altro, uno stato dell'arte ben definito. Analizzeremo dapprima due soluzioni che non sono risultate totalmente efficaci, ma le cui implementazioni e i cui errori hanno indirizzato la scelta e l'ideazione delle due soluzioni ritenute, invece, valide e ben funzionanti.

3.8.1 Primi approcci di object tracking

Il primo approccio tentato include l'utilizzo della libreria Vuforia, integrabile nel game engine Unity e specializzata nel tracciamento di oggetti virtuali nel mondo reale: a partire dal modello CAD dello stesso è in grado di individuarlo nella scena e far combaciare costantemente la mesh digitale con l'oggetto reale, restituendo dunque la sua rotazione rispetto alla camera (o meglio, la rotazione della camera rispetto a esso, facilmente traducibile in quella dell'oggetto). Avendo definito un gruppo di oggetti fisso e limitato, si è proceduto a modellarli virtualmente con l'utilizzo del noto strumento di modellazione Blender, partendo dalle fotografie prospettiche degli stessi, e successivamente a integrarli all'interno della libreria sopra citata: utilizzando un apposito software facente parte della libreria, il *Model Target Generator*, si è fornito un modello virtuale alla volta al sistema, che ha effettuato un apposito allenamento per essere in grado di riconoscere lo stesso da tutte le angolazioni possibili, restituendo un pacchetto importabile in Unity. Una volta che questi è stato aggiunto correttamente nel software, viene utilizzato in tempo reale per tracciare la rotazione tridimensionale dell'oggetto nella scena corrente.

Al momento delle prime prove pratiche di questa soluzione sono emerse alcune problematiche che l'hanno resa difficile da utilizzare e inadatta al task corrente. Si elencano alcune delle lacune più importanti:

- Il sistema sa riconoscere l'oggetto da tutte le angolazioni, ma per essere correttamente inizializzato prima della dimostrazione dell'azione deve inquadrare l'oggetto da un'angolazione fissa, nota come *guide view* ("vista guida"), e poiché nel setup dell'ambiente di lavoro la telecamera è considerata immobile ciò si traduce nel dover avvicinare l'oggetto alla stessa prima di ogni azione e ruotarlo correttamente per farlo combaciare con la silhouette di inizializzazione, ovvero una pratica che rallenta fortemente il flusso di lavoro e non è particolarmente avvezza alla generalizzazione ad ambienti nuovi

- Il modello tridimensionale di ogni oggetto, essendo stato creato "manualmente", non combacia con una perfezione millimetrica con l'oggetto reale, soprattutto nei casi di oggetti curvilinei e con forme inusuali. Inoltre, i due sono piuttosto differenti sul piano della colorazione e della texture, che difficilmente combacia con quella reale anche per la possibilità di incontrare condizioni di illuminazione differenti. Tutto ciò risulta in una minore precisione del sistema, che spesso "perde traccia" dell'oggetto e costringe a ricominciare da capo il processo

- L'oggetto d'interesse, quando si trova sul piano di lavoro all'inizio e fine di un'azione, è spesso molto distante dalla telecamera, dunque il suo bounding box descrive un'area dell'immagine a risoluzione molto bassa. Poiché Vuforia si basa su di essa per tracciare l'oggetto in tempo reale, spesso ciò si traduce in una stima della posa imprecisa e inefficace, non essendo in grado di rilevare correttamente delle rotazioni anche di diversi gradi, impercettibili nelle immagini ma importantissime per il robot

I diversi punti negativi hanno portato all'esclusione dell'uso di Vuforia come metodo per la stima della posa dell'oggetto, ma fornendo una buona base di partenza per le ricerche successive

3.8.2 Differentiable rendering

Nonostante il metodo di tracciamento appena descritto sia stato considerato inadeguato, esso ha lasciato in dote una serie di mesh tridimensionali degli oggetti di questo progetto e un primo approccio approfondito alla manipolazione

di modelli virtuali per la stima della loro posa. Questi dati sono stati accolti come punti di partenza per provare alcuni approcci differenti, producendo dei risultati sicuramente di migliore fattura rispetto ai precedenti.

Il metodo noto come *differentiable rendering* che ci si propone di descrivere in questa sezione ha come obiettivo quello di far allineare il rendering di un oggetto virtuale con un'immagine-obiettivo dello stesso, in questo caso proveniente dal mondo reale, tentando di far combaciare le "silhouette" delle due. La libreria PyTorch3D possiede gli strumenti necessari a eseguire questi calcoli.

Il processo è abbastanza lineare: si ritaglia il bounding box identificato in fase di riconoscimento dell'oggetto e ne si ricava una "silhouette", eseguendo una segmentazione piuttosto semplice basata sul *thresholding*, ovvero sulla trasformazione dell'immagine in scala di grigi e successiva eliminazione dei pixel sotto una certa soglia. Dopodiché la libreria usa la mesh dell'oggetto in formato `.obj` assieme a un `renderer` (formato da un rasterizzatore e uno shader) per inizializzare una telecamera virtuale che inquadra il modello fornito.

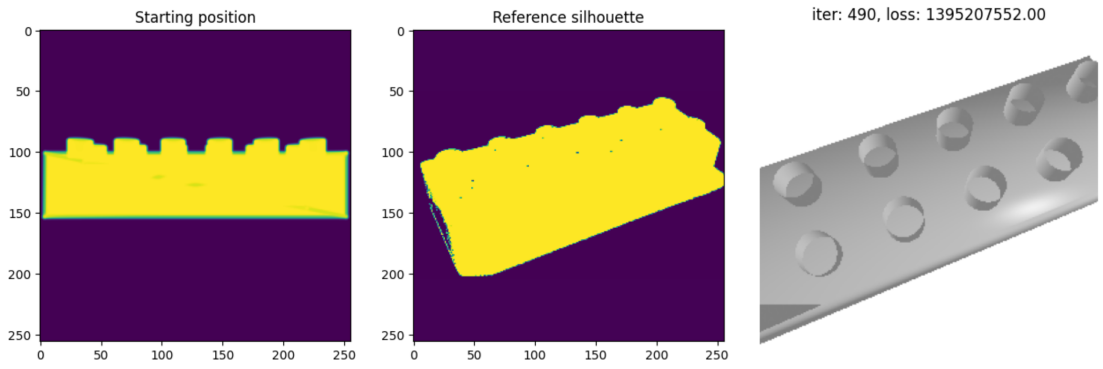


Figura 3.9: Esempio di ottimizzazione della posa della camera con il differentiable rendering

Infine, si avvia un ciclo di ottimizzazione dalla durata variabile che, con un metodo piuttosto simile all'addestramento di una rete neurale, con tanto di funzione di loss e passo di ottimizzazione, esegue il rendering dell'oggetto da un punto di vista sempre più simile a quello della silhouette di riferimento, tentando di diminuire il valore di loss e avvicinando le due prospettive.

Sebbene il differentiable rendering costituisca una soluzione efficace per la stima della rotazione dell'oggetto, persistono alcuni problemi: il loop di ottimizzazione ha bisogno di una certa quantità di di tempo per produrre un risultato valido, e contemporaneamente il risultato dipende dalla qualità della silhouette di riferimento che si è calcolata in prima istanza, spesso condizionata dalla presenza di ombre o oggetti vicini.

Dopo aver fatto queste considerazioni, la ricerca di un metodo di stima della posa più rapido ed efficace ha spostato il proprio focus su approcci diversi-

3.8.3 Segmentazione e analisi delle nuvole di punti

Il fatto che la telecamera Azure Kinect DK utilizzata in questo lavoro sia in grado di acquisire una nuvola di punti della scena che ha davanti suggerisce di poter utilizzare questa informazione in maniera vantaggiosa, sviluppando una soluzione che sia coerente con gli obiettivi prefissati.

In questa fase del flusso di lavoro si sa di conoscere il bounding box (e dunque la posizione approssimativa) dell'oggetto negli istanti di inizio e fine azione, nonché in quelli intermedi. Ipotizzando di selezionare il primo di questi frame, possiamo usare l'informazione del bounding box per ritagliare coerentemente la point cloud intorno all'oggetto nel momento desiderato. Provenendo da una telecamera posta a un'angolazione variabile, e non perfettamente perpendicolare al piano di lavoro, la nuvola che otterremo includerà una parte dell'ambiente circostante all'oggetto, soprattutto nel suo lato posteriore, e dunque non sarà perfettamente circoscritta a esso: potrebbe contenere parti di altri oggetti vicini a quello considerato, parti dell'ambiente di lavoro che ricadono nella proiezione della nuvola e, in generale, punti che costituiscono del "rumore". In ogni caso, anche se l'oggetto d'interesse si trova in un punto libero da disturbi limitrofi, la point cloud attorno a esso contiene inevitabilmente una parte del piano su cui poggia.

Viene naturale pensare di segmentare l'oggetto fuori dalla point cloud, isolandolo da tutti i punti che non gli appartengono per ottenerne una rappresentazione completa nello spazio: questo è un compito che spesso, per funzionare efficacemente, richiede soluzioni di machine learning, che si vogliono evitare sia per una questione prestazionale che per il fatto di aver ricavato una nuvola di punti molto vicina a quella del solo oggetto, suggerendo la possibilità di trovare strade alternative per raggiungere velocemente il risultato sperato. Entra in gioco il noto algoritmo RANSAC che può essere adattato al fine di individuare il piano dominante all'interno di una point

Figura 3.10: Esempio di ritaglio della point cloud attorno all'oggetto e segmentazione del piano

cloud, risultando particolarmente efficace: esso è descritto, in termini più generici, come un metodo per la stima dei parametri di un sistema, e si basa sulla selezione casuale di dati di input al fine di trovare i loro "vicini" e individuare il sotto-sistema di dimensione maggiore. Nel caso in oggetto, viene usato per trovare il piano più grande descritto dai punti della nuvola.

Per la manipolazione delle point cloud si fa largo uso della libreria Open3D, che fornisce numerose funzioni volte a calcolare varie proprietà delle stesse. Elenchiamo le fasi principali di questa soluzione, volta a isolare l'oggetto nella sua nuvola di punti:

- Il primo passo è l'eliminazione degli *outliers* in scena utilizzando la funzione `remove_statistical_outlier` della libreria sopra citata, che si basa sulla numero di vicini che possiede ogni punto per scartare quelli considerati "anomali"

- Segue la rimozione del piano d'appoggio, che risulta essere il piano dominante della scena e che viene individuato utilizzando la funzione `segment_plane` basata sull'algoritmo RANSAC appena descritto: una volta trovato, può essere "filtrato via" dalla nuvola in analisi, che conterrà solo l'oggetto su di esso posato assieme a eventuale "rumore", costituito principalmente da punti dell'oggetto non correttamente rilevati dalla telecamera (laterali o retrostanti) e sezioni di oggetti circostanti a quello principale.

- Successivamente si riesegue la rimozione degli outliers che possono essere nati dalla segmentazione del piano, e si passa a selezionare l'oggetto di interesse tra i cluster di punti che sono rimasti in scena. Per fare ciò si fa ricorso a DBSCAN, noto algoritmo di *clustering* che basa i propri calcoli

sul concetto di densità e che viene implementato con una relativa funzione in Open3D: essa individua tutti i cluster di punti della scena e seleziona quello a dimensione maggiore, che andrà a rappresentare la nuvola di punti dell'oggetto interessato. Si è ragionevolmente sicuri di questo poiché la prima nuvola di punti registrata è incentrata attorno all'oggetto di interesse, dunque gli altri cluster individuati possono rappresentare solo sezioni di oggetti secondari e di dimensione minore

- Ottenuta la point cloud dell'oggetto, se ne può calcolare il baricentro ricavando la media delle coordinate di tutti i punti. Poiché ognuno di essi contiene le sue coordinate rispetto alla telecamera, la posizione del baricentro sarà quella che si cercava sin dall'inizio, ovvero quella che potremo definire propria dell'oggetto e che può essere utilizzata dal robot per la manipolazione.

- Open3D permette inoltre di calcolare il bounding box 3D orientato dell'oggetto con la funzione `get_minimal_oriented_bounding_box`, che esegue automaticamente il calcolo del bounding box orientato di dimensione più piccola possibile e contenente tutti i punti. Da esso, possiamo ricavarne la matrice di rotazione rispetto agli assi, che viene facilmente convertita negli angoli di Eulero necessari al robot per approcciare efficacemente la scena.

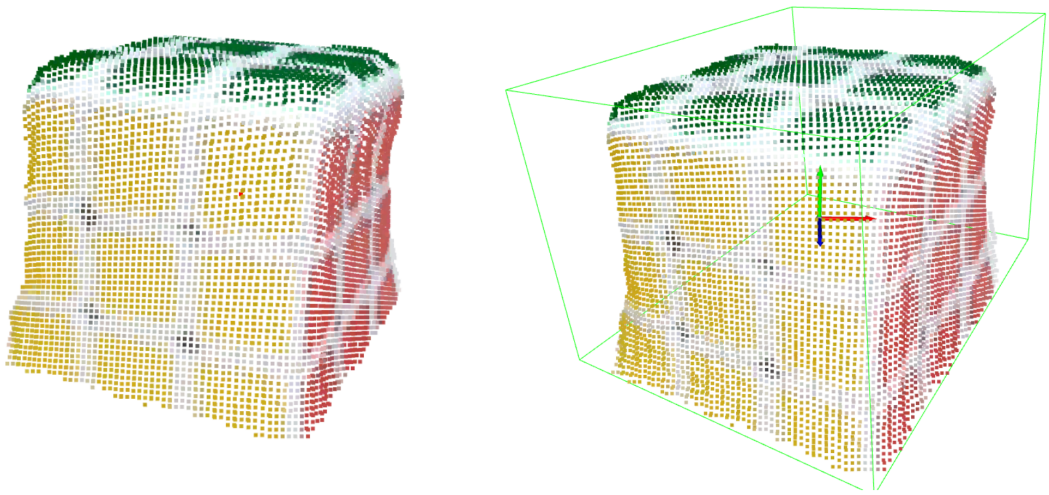


Figura 3.11: Nuvola di punti ripulita dagli outliers e con bounding box e centro evidenziati

Se il calcolo della rotazione dell'oggetto dà un risultato di qualità confrontabile con quello del rendering differenziale, questo metodo presenta numerosi vantaggi, tra cui l'evidente maggiore velocità di calcolo e la possibilità di ottenere anche la posizione dell'oggetto con maggiore precisione rispetto alla fase precedente.

Senza dubbio è un metodo che porta con sé anche diversi punti negativi: innanzitutto la posa viene stimata senza avere una reale percezione della texture dell'oggetto, dunque senza conoscere un vero e proprio riferimento rispetto alla quale eseguire i calcoli. Le rotazioni che si otterranno saranno, dunque, quelle minime rispetto a un bounding box 3D non ruotato.

Inoltre, la stima della posa è condizionata dalla forma finale della nuvola, che, anche dopo numerosi passaggi di manipolazione, può contenere del rumore e dei punti che ostacolo il calcolo preciso delle rotazioni: come immaginabile, la point cloud è raccolta da un solo punto di vista, e nella maggior parte dei casi non contiene la parte posteriore dell'oggetto considerato, poiché la telecamera non lo può ovviamente inquadrare da tutte le angolazioni.

Capitolo 4

Test e risultati

In questo capitolo verranno analizzati i test eseguiti al fine di comprendere l'accuratezza del progetto in tutte le sue parti, il quale è stato messo alla prova sia come sistema a sé stante che in relazione a un robot vero e proprio per le prove sul campo. Le due fasi sono approfondite in sezioni distinte del capitolo.

4.1 Setup dell'ambiente di lavoro

Al fine di ottenere un riferimento unico nell'analisi dei risultati prodotti è stato scelto un ambiente di lavoro "standard" e ben definito, che potesse rappresentare una verosimile area di manipolazione e contemporaneamente essere pronto per le prove pratiche dell'agente robotico vero e proprio. Sebbene per la maggior parte della durata del suo sviluppo il sistema sia stato testato in ambientazioni domestiche, su piani di lavoro di dimensioni e forme arbitrarie, si è reso necessario definire un luogo unico e un setup ben preciso al fine di avere un riferimento standard entro cui muoversi e calibrare il sistema per i test, oltre che per gli aggiustamenti che si è reso necessario eseguire dopo le numerose prove.

4.1.1 Robot

Visto il prestabilito intento di testare il sistema con l'uso di un vero robot, è stato sin da subito chiaro che l'area di esecuzione dei test sarebbe dovuta coincidere con una raggiungibile dal macchinario a disposizione. Il setup di test è, dunque, fortemente legato all'agente protagonista delle prove pratiche: si tratta di *e.DO* di Comau, un braccio robot a 6 gradi di libertà pensato a

scopo educativo e in grado di manipolare oggetti fino a un chilogrammo di peso. Possiede un gripper classico "a pinza" ed esegue i comandi impartiti con una velocità sufficiente a emulare l'azione eseguita da un umano. Esso è stato posizionato su un tavolo per visualizzare al meglio le azioni nell'area di lavoro, oltre che per facilitare il posizionamento della telecamera e l'interazione del dimostratore umano.

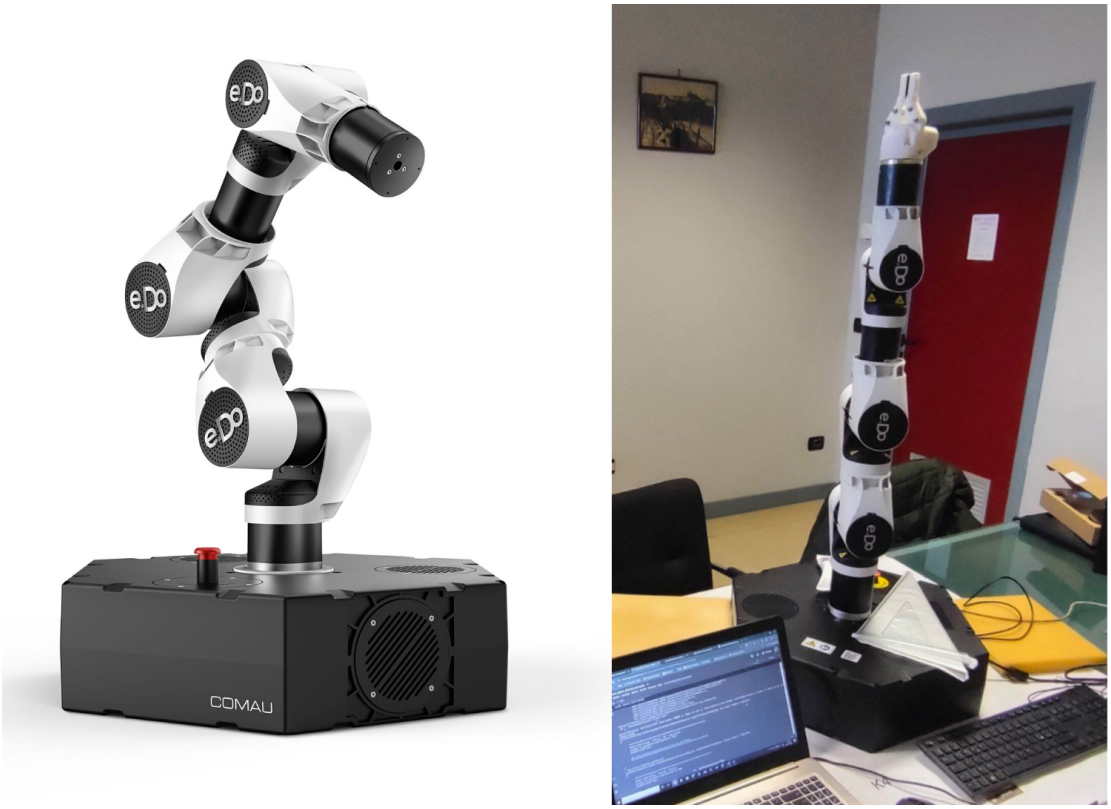


Figura 4.1: Robot e.DO di Comau, visibile anche in posizione di partenza

4.1.2 Area di lavoro

Una volta definito il robot da utilizzare in fase di test, è possibile descrivere il setup di interazione in relazione a esso. L'agente e.DO possiede un proprio sistema di riferimento, utilizzato al fine di orientarsi nel suo spazio di manovra e mostrato in 4.2: l'asse di altezza `z` è perpendicolare al terreno (ovvero al tavolo sui cui poggia) e da esso uscente, mentre gli assi `x` e `y` sono paralleli a esso e puntano, rispettivamente, in avanti e a sinistra del robot. L'origine del sistema è posizionata in un punto interno all'asse principale del robot, grossomodo al

centro della sua base, ed è stata rilevata con precisione posizionando il gripper del robot in un punto noto e calcolando manualmente le distanze dall'origine.

Come area di lavoro è stata scelta una lastra rigida ricoperta da un foglio bianco, al fine sia di appuntare informazioni che di facilitare il lavoro della rete di hand-object interaction recognition (sensibile agli oggetti di sfondo). Essa è stata posizionata sul tavolo immediatamente davanti alla base rigida del robot, di modo che corrispondesse, con un errore trascurabile, a un piano parallelo a quello individuato dagli assi `x` e `y` del robot (il quale coincide, idealmente, con il tavolo stesso). Inoltre, è stata sollevata a sufficienza per facilitare il suo raggiungimento da parte del gripper del robot, nello specifico con la sua superficie superiore a 6.5 cm di altezza dal tavolo.

4.1.3 Posizione del marker ArUco

In linea con quanto definito nella parte di architettura del sistema, la posizione dell'oggetto in scena è calcolata in relazione al marker ArUco in essa presente, che rappresenterà il sistema di riferimento rispetto al quale calcolare la nuova posizione. Al fine di eseguire i test pratici con il robot si è rivelato necessario posizionare il marker nella maniera migliore possibile per facilitare la trasformazione della posizione nel sistema di riferimento corretto: poiché si prende in considerazione l'angolo in alto a sinistra del marker come origine, questi è stato posizionato lungo l'asse `x` del sistema di riferimento del robot, sulla parete frontale della sua base rigida per essere ben visibile alla telecamera e di modo che avesse coordinata `y` pari a zero.

Calcolando manualmente la posizione dell'angolo in alto a sinistra del marker, rispetto all'origine del sistema del riferimento del robot, otteniamo le seguenti coordinate: 24cm lungo l'asse `x`, 0cm lungo l'asse `y` e 12.5cm lungo l'asse `z`.

Guardando il sistema nel complesso, il processo di trasformazione della posizione diventa, dunque, triplice: si parte dal sistema di riferimento della camera, si passa a quello del marker ArUco ed, infine, a quello del robot stesso.

4.1.4 Posa della videocamera

Il modo in cui la videocamera è posizionata all'interno della scena ha una grande importanza al fine di garantire dei test validi e privi di interferenze. Sono state testate diverse configurazioni: dapprima è stata posizionata con un angolazione di -25° attorno all'asse x e un altezza minore, per dare una

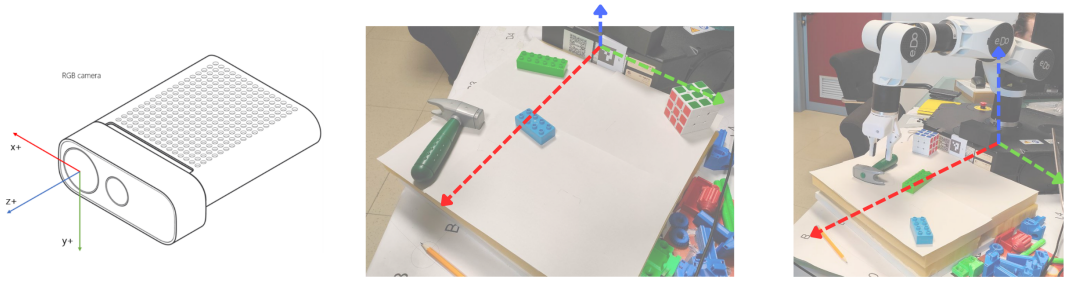


Figura 4.2: Sistemi di riferimento a confronto

prospettiva più realistica degli oggetti in scena e inquadrare correttamente il marker, ma dopo numerose prove ci si è resi conto che la posa migliore, al fine sia garantire una buona visibilità di oggetti e marker che di inquadrare quanto meno possibile l'ambiente circostante, fosse a 45cm di altezza dalla superficie di lavoro, a 15cm di profondità dal suo bordo anteriore, centrata quanto più possibile rispetto al marker e ruotata fino a raggiungere -55° attorno all'asse `x`. In tal modo si crea un'inquadratura abbastanza verticale da concentrarsi solo sul piano di lavoro, ma mantenendo una buona prospettiva sugli oggetti e sul marker di scena, semplificando il lavoro della rete di hand-object interaction detection ed evitando di interferire con la sfera d'azione del braccio robotico.

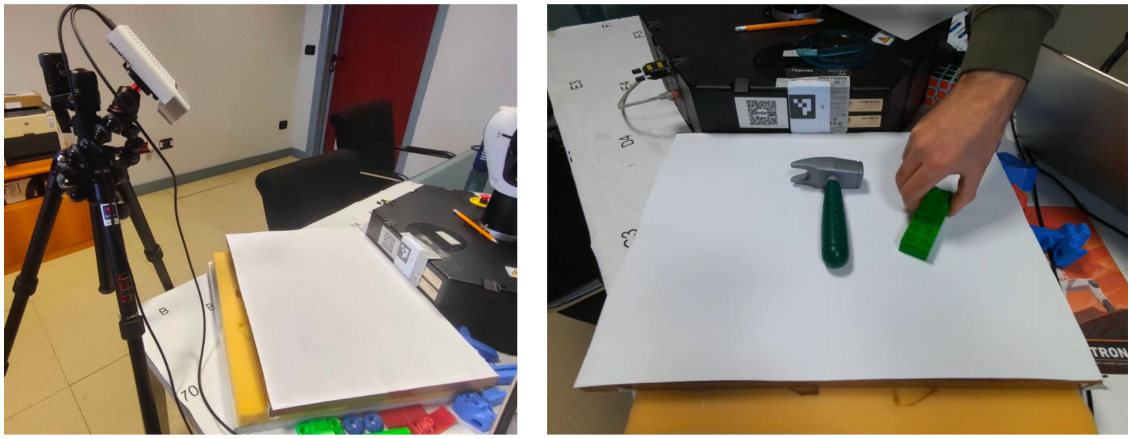


Figura 4.3: Posa della videocamera in scena e relativo punto di vista

4.2 Risultati ottenuti

In questa sezione verranno presentati i risultati conseguiti in tutte le fasi dello sviluppo di progetto e durante i test: in quest'ultimo caso si parlerà delle prove eseguite con il sistema al completo e con l'ambiente di lavoro appena descritto, ma senza l'uso del robot vero e proprio.

4.2.1 Test eseguiti

Al fine di valutare correttamente il sistema si è deciso di eseguire un certo numero di azioni utilizzando il setup di test, per poi calcolare le seguenti metriche d'interesse e comprendere le capacità del progetto nell'uso pratico:

- accuratezza della rete di hand-object interaction detection
- accuratezza della rete di riconoscimento dell'azione
- accuratezza e precisione nel calcolo della posizione dell'oggetto
- accuratezza e precisione nel calcolo della posa dell'oggetti

Al fine di valutare le metriche di accuratezza sono state eseguite 50 dimostrazioni di tutte le azioni: per ognuna di esse è stata misurata manualmente la posizione del centro dell'oggetto e della sua rotazione all'inizio e alla fine dell'azione, mentre la pipeline del sistema si occupava dell'analisi e della produzione dei risultati.

Successivamente, per valutare la precisione del sistema nella stima di posa e posizione, è stata registrata una singola dimostrazione di ognuna delle otto classi di azione, per poi passare la stessa cinque volte al sistema e confrontare la vicinanza dei risultati.

4.2.2 Rete di action recognition

L'analisi dei risultati parte da quella della rete di riconoscimento dell'azione svolta, per la quale siamo in grado di fornire sia gli esiti riguardanti il dataset di validazione che quelli relativi ai test appena descritti.

I risultati ottenuti durante il fine-tuning sono stati molto soddisfacenti, producendo delle metriche di valutazione di buon livello osservabili nei grafici relativi: il valore di loss di allenamento si è attestato intorno allo 0.25, mentre lo stesso per la validation ha raggiunto lo 0.27; contemporaneamente, l'accuratezza di classificazione in allenamento ha superato il 90% all'ultima

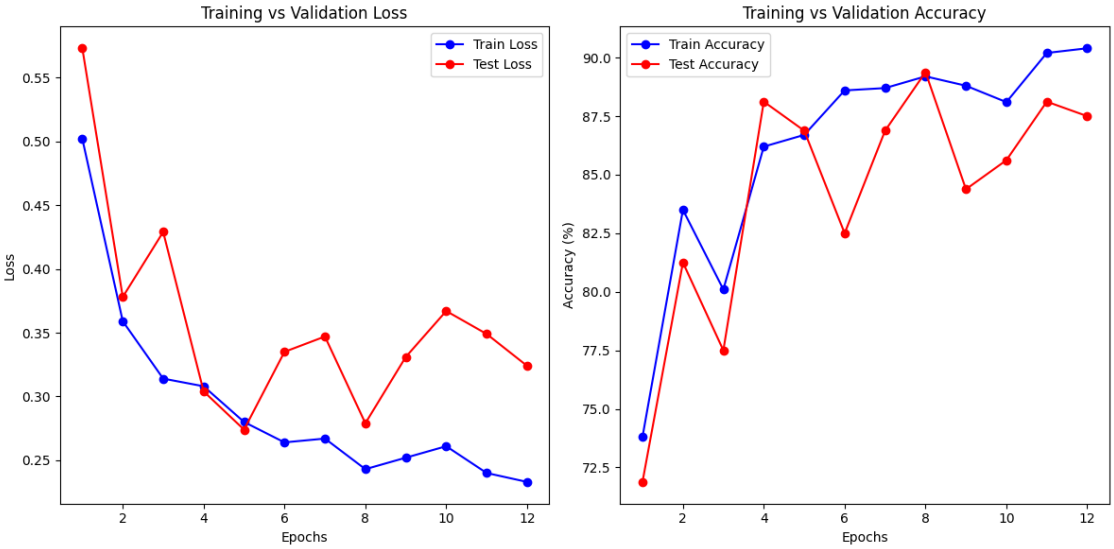


Figura 4.4: Grafici di perdita e accuratezza lungo le epoche di allenamento

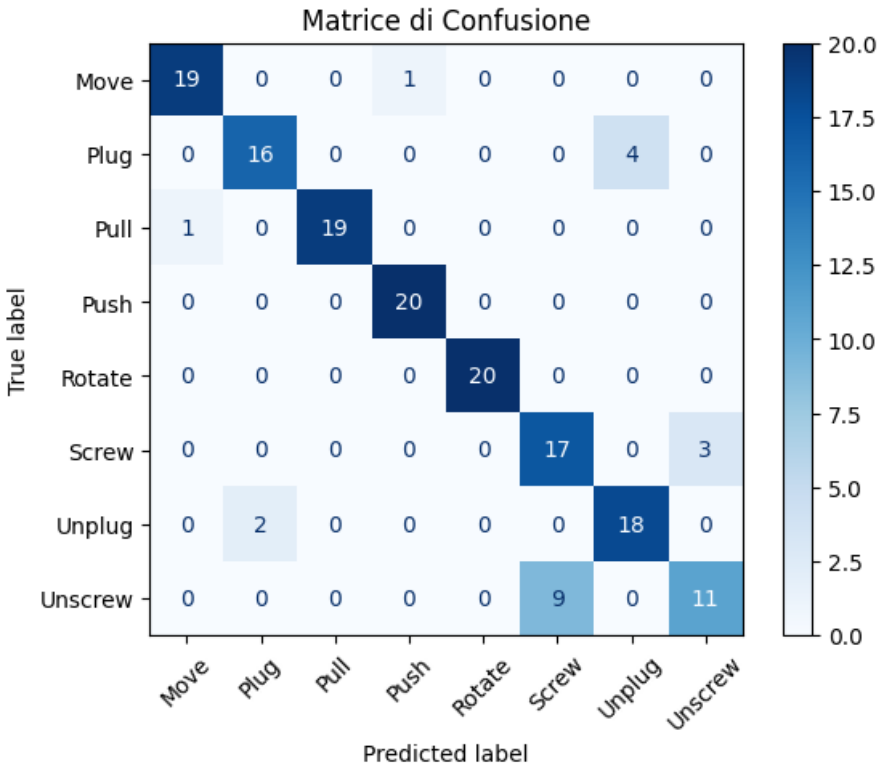


Figura 4.5: Matrice di confusione della fase di validazione

epoca, oscillando tra l'85% e il 90% nel caso della fase di validazione già a partire dalla quarta epoca. La matrice di confusione prodotta proprio all'ultimo step di addestramento evidenzia le classi maggiormente propense a essere confuse e quelle più solide nel riconoscimento. Mentre la maggior parte di esse non dimostra particolari anomalie, si nota che la rete fa una leggera confusione nel riconoscere le classi "Plug" e "Plug", mentre è più evidente tra le classi "Screw" e "Unscrew": quest'ultimo caso è spiegabile con il fatto che la distinzione tra le due non è data da movimenti evidenti, differendo nella sola direzione di rotazione delle dita, ovvero un'informazione difficile da rappresentare efficacemente con il flusso ottico e il semplice video RGB (anche considerando che i video sono scalati in dimensione, compromettendo la chiara rappresentazione di movimenti così locali).

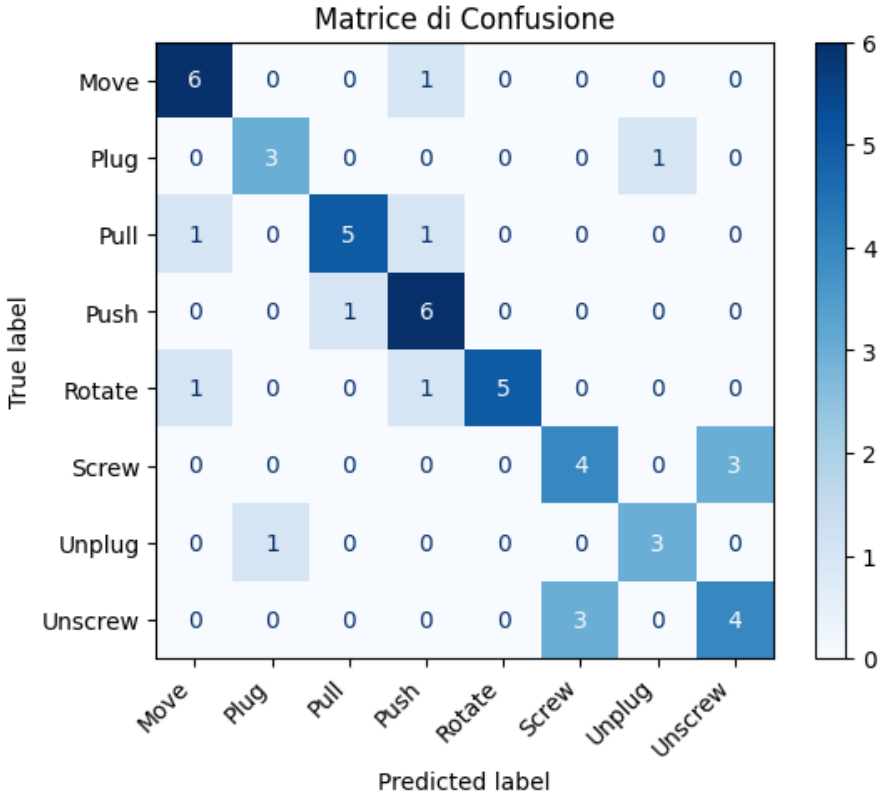


Figura 4.6: Matrice di confusione della fase di test

Le statistiche di riconoscimento raccolte durante i test finali vedono le prestazioni peggiorare leggermente: la rete riconosce l'azione corretta nel 72% dei casi, riportando un esito errato nel restante 28%. La matrice di confusione rappresentata in 4.6 mostra quali classi sono state maggiormente protagoniste

di predizioni erronee: la confusione principale persiste tra le classi "Plug" e "Unplug", che per quasi la metà dei tentativi non hanno riportato il risultato corretto, mentre, in qualche occasione, la classe "Rotate" è stata invertita con quella "Move".

Sicuramente la discrepanza tra l'ambiente nel quale sono stati registrati i video del dataset di fine-tuning e l'ambiente di test ha contribuito al peggioramento delle prestazioni, a cui si aggiunge doverosamente che, per tale dataset, le azioni sono state registrate da angolazioni piuttosto differenti, mentre durante le prove pratiche la telecamera è rimasta fissa in una posa anche leggermente più "verticale" di quelle viste dalla rete durante l'allenamento.

4.2.3 Rete di hand-object interaction

Sia durante lo sviluppo del sistema che durante i test è stato notato che la rete che identifica il contatto tra la mano e gli oggetti in scena non è costante nei rilevamenti, rendendo necessaria la stima della sua accuratezza. Nel corso delle numerose prove si è avuto modo di distinguere quattro diverse casistiche di rilevamento, la cui connotazione è sia qualitativa che riferita alle conseguenze che hanno sul risultato finale, ma che sono comunque in grado di spiegare il tipo di risultati che produce la rete:

- *Rilevamento corretto*: la rete riconosce correttamente il contatto tra la mano e l'oggetto nel momento esatto, e la stima del bounding box porta al risultato giusto
- *Rilevamento parziale*: la rete riconosce il contatto al momento giusto, ma il bounding box rilevato contiene solo una parte dell'oggetto, portando comunque a un risultato di posizione e posa sufficiente
- *Frame di contatto errato*: la rete riconosce alcuni contatti, sia quello corretto che altri falsi con oggetti di sfondo, ma la ricerca del giusto frame produce un risultato inesatto
- *Rilevamento mancato*: la rete non riconosce il contatto con l'oggetto, il più delle volte perché è occluso dalla mano o perché identifica un contatto con il piano di lavoro, come se fosse un oggetto a sé stante

Durante la registrazione e il passaggio dei 50 video di test attraverso il sistema, dunque, sono stati segnalati tutti i casi in cui il rilevamento ricadeva in una delle casistiche appena citate, producendo i risultati riportati nella tabella 4.1

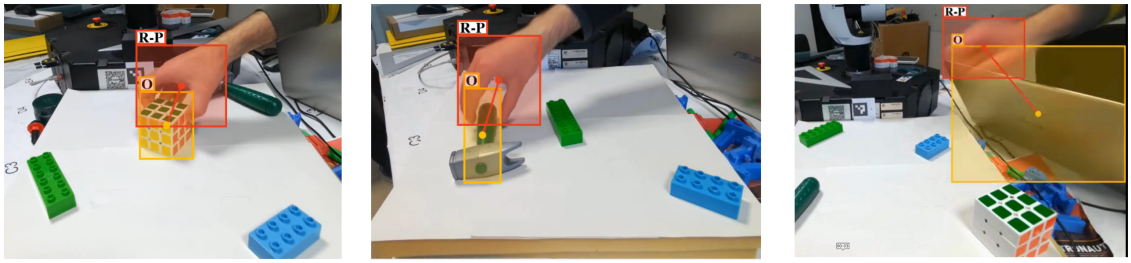


Figura 4.7: Da sinistra, esempi di rilevamento corretto, parziale ed errato

Rilev. / Istante	Corretto	Parziale	Frame errato	Oggetto mancato
Inizio azione	42	2	2	4
Fine azione	43	2	3	2
Totale	**85**	**4**	**5**	**6**

Tabella 4.1: Rilevamenti della rete di hand-object interaction detection sui 50 video di test (100 posizioni)

Si nota che nel 85% dei casi possiamo apprezzare un rilevamento valido, mentre nel 4% un rilevamento parziale ha comunque portato a un risultato sufficiente. Questo secondo caso ha riguardato principalmente l'oggetto "martello giocattolo", la cui forma inusuale ha reso difficile la sua identificazione, specialmente quando si trovava in posa perpendicolare alla telecamera.

Il 5% di casi riguardanti i frame errati coincidono prevalentemente con momenti in cui la mano è apparsa in scena troppo a lungo senza eseguire l'azione, evidenziando un contatto con un oggetto di sfondo, mentre nel 6% degli istanti l'oggetto protagonista è stato confuso con il piano di lavoro, il cui bounding box è di dimensione maggiore e contiene quello dell'oggetto cercato.

4.2.4 Riconoscimento degli oggetti

Anche la rete YOLO-World, utilizzata per il riconoscimento degli oggetti presenti in scena, svolge un ruolo importantissimo nella pipeline del progetto, poiché l'individuazione della posizione dell'oggetto passa dalla corretta

identificazione dello stesso, altrimenti impedendo il prosieguo dei calcoli da effettuare.

YOLO-World ha la necessità di ricevere in partenza una serie di descrizioni testuali degli oggetti che ricercherà nella regione d'interesse individuata dalla rete di hand-object interaction detection: durante lo sviluppo del sistema sono stati individuate le migliori per ottenere i risultati più adatti, e sono state riportate anche in fase di test.

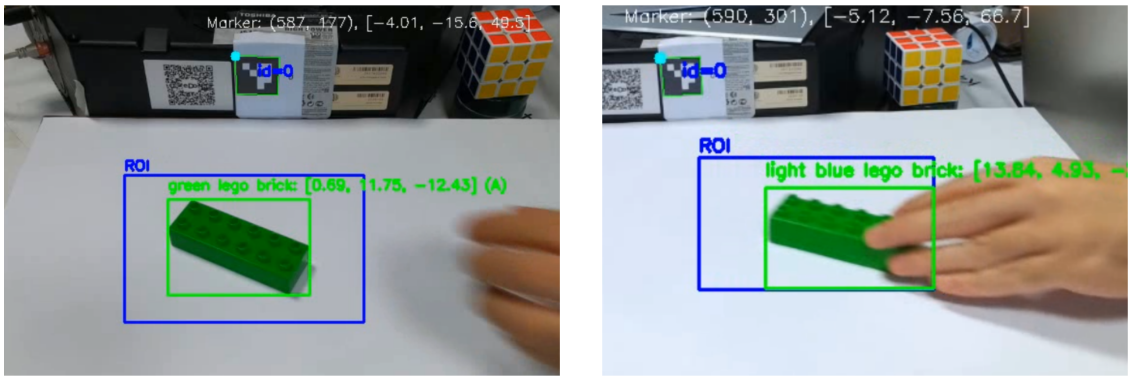


Figura 4.8: Esempi di riconoscimento corretto e impreciso

Per taluni oggetti è stato necessario fornire più di una descrizione dello stesso, poiché a seconda di alcune condizioni ambientali, di illuminazione, di posa o di occlusione dell'oggetto potrebbe dare priorità a una rappresentazione testuale piuttosto che ad altre. A titolo d'esempio, per il cubo di Rubik sono state utilizzate le diciture "Rubik's cube" e "Cube", mentre per il barattolo le parole "Jar", "Dark green jar" e "Dark green can".

Per ogni istante di inizio e fine azione delle 50 dimostrazioni effettuate sono stati segnalati i rilevamenti erronei o imprecisi, secondo le seguenti casistiche:

- *Identificazione corretta*: la rete riconosce correttamente l'oggetto e la sua classe di appartenenza
- *Identificazione imprecisa*: riguarda le casistiche dove, nonostante le imprecisioni, si giunge comunque a un risultato valido. Si tratta principalmente di confusioni di classi (ad esempio, il mattoncino verde identificato come "Cube") ma con un bounding box giusto, oppure di identificazioni parziali propagatesi dalla rete di hand-object interaction detection (ad esempio, il martello giocatolo identificato solo sul manico).

- *Identificazione errata*: la rete non riconosce l'oggetto in scena, impedendo di proseguire nell'analisi

Anche in questa occasione si forniscono le statistiche riguardo i casi appena citati, dettagliati nella tabella 4.2.

Ident. / **Istante**	**Corretta**	**Imprecisa**	**Errata**
Inizio azione	46	3	1
Fine azione	46	2	2
Totale	**92**	**5**	**3**

Tabella 4.2: Identificazioni degli oggetti della rete YOLO-World durante i test

Se nel 92% dei casi l'identificazione è corretta, nel 5% dei casi l'oggetto viene rilevato parzialmente, principalmente per l'imprecisione di rilevamento propagata dalla fase precedente. Solo nel 3% dei casi non abbiamo alcuna identificazione, riguardando esclusivamente situazioni in cui la mano occlude una buona parte dell'oggetto.

4.2.5 Posizione e posa degli oggetti

Le metriche più importanti per analizzare la bontà del sistema sono sicuramente quelle che riguardano la posizione e la posa rilevata nelle fasi finali, poiché condizionano l'efficace esecuzione dell'azione stessa.

Per ottenere la metrica di accuratezza di questa parte del progetto sono stati misurati manualmente tutti i valori di coordinate e rotazioni nello spazio per gli oggetti manipolati durante le 50 esecuzioni di test.

Come possiamo notare dai risultati, l'accuratezza si attesta su valori soddisfacenti e che non impediscono, a rigora, una corretta esecuzione dell'azione considerata. Vi è anche da considerare che essa è calcolata confrontandola con le misurazioni eseguite a mano sulla posa e posizione degli oggetti coinvolti, che potrebbe introdurre, quasi certamente, un errore umano di misurazione, senza il quale il risultato potrebbe sia migliorare che peggiorare.

Allo stesso tempo, per ottenere la precisione del sistema, come già citato sono stati confrontati i risultati ottenuti passando per cinque volte lo stesso

Errore / Istante	**Coord. X (cm)**	**Coord. Y (cm)**	**Coord. Z (cm)**	**Media (cm)**
Inizio azione	0.55	0.60	0.46	**0.53**
Fine azione	0.52	0.57	0.45	**0.51**
Media	**0.53**	**0.58**	**0.45**	**0.52**

Tabella 4.3: Errore di accuratezza nella stima della posizione degli oggetti, per tutte le azioni

Errore / Istante	**Coord. X (gradi)**	**Coord. Y (gradi)**	**Coord. Z (gradi)**	**Media (gradi)**
Inizio azione	5.8	5.8	6.9	**6.2**
Fine azione	5.7	5.5	6.7	**6.0**
Media	**5.7**	**5.6**	**6.8**	**6.1**

Tabella 4.4: Errore di accuratezza nella stima della posa degli oggetti, per tutte le azioni

input lungo l'intera pipeline, ripetuto ovviamente per ognuna delle azioni scelte. Le metriche selezionate sono la deviazione standard, che indica di quanto i risultati varino rispetto alla loro media, e l'errore di ribetibilità medio, ovvero lo scostamento massimo che otteniamo nelle varie misurazioni.

Analizzando questi dati possiamo notare che il sistema risulta particolarmente costante nel rilevamento della posa e della posizione dell'oggetto, evitando di variare eccessivamente nei risultati prodotti. Ovviamente questa metrica ha un'importanza relativa se confrontata con quella dell'accuratezza, che descrive più efficacemente la capacità del sistema di ottenere dei risultati veritieri, ma suggerisce anche che dei futuri miglioramenti possono generare un sistema piuttosto costante e puntuale nel calcolo di posa e posizione.

Misura / Azione	Deviazione standard (cm)	Errore di ripetizione (cm)
Push	0.02	0.4
Pull	0.04	0.3
Pick and place	0.03	0.2
Rotate	0.03	0.4
Screw	0.02	0.2
Unscrew	0.02	0.2
Plug	0.04	0.5
Unplug	0.05	0.4
Media	**0.03**	**0.32**

Tabella 4.5: Misure di precisione nella stima della posizione degli oggetti, per tutte le coordinate e gli istanti

Misura / Azione	Deviazione standard (gradi)	Errore di ripetizione (gradi)
Push	0.95	4.2
Pull	0.92	4.4
Pick and place	1.15	6.5
Rotate	1.02	8.1
Screw	0.90	3.3
Unscrew	0.92	2.0
Plug	1.52	8.1
Unplug	1.32	8.4
Media	**1.08**	**5.62**

Tabella 4.6: Misure di precisione nella stima della posa degli oggetti, per tutte le coordinate e gli istanti

4.3 Test con il robot

Per concludere la sezione dedicata ai test del sistema si citano una serie di prove effettuate sul campo con l'utilizzo vero e proprio del robot e.DO, che è stato citato nella fase di descrizione dell'ambiente di prova. Le esecuzioni sono state effettuate al fine di validare il funzionamento del sistema al completo, dall'inizio alla fine del flusso di lavoro, sebbene non forniscano effettivamente alcuna metrica oggettiva di valutazione della bontà del progetto, ma solo alcune considerazioni qualitative.

Trattandosi di un progetto fortemente incentrato sulla parte di analisi video, e dunque focalizzato sul raccogliere e comunicare al robot la maggior quantità possibile di informazioni utili per replicare un certo set di azioni, non è stato interessato da sezioni legate alla programmazione robotica o all'apprendimento incentrato sull'agente: i test eseguiti con il robot vero e proprio servono a "chiudere il cerchio" e fornire una visione d'insieme sul funzionamento della pipeline dal momento della registrazione della dimostrazione a quello della sua ripetizione.

Il robot e.DO, dal canto suo, non esprime assolute capacità di manovra nello spazio di lavoro, possedendo alcune limitazioni che devono essere prese in considerazione per ottenere delle esecuzioni lineari e senza intoppi: alcune di queste includono la limitata capacità di carico o l'apertura massima del gripper, che sono state considerata in fase di scelta degli oggetti del progetto, oppure le limitazioni in termini di forza e accuratezza millimetrica di posizionamento.

Si aggiunge che le azioni testate in questa fase non includono le due di "Plug" e "Unplug", proprio per via delle modeste capacità dell'agente e.DO: esso avrebbe dovuto alternativamente tirare verso di sé un oggetto come una spina elettrica, infilata e ben agganciata nel propria presa, o tentare di spingere la stessa (o un oggetto simile, come una unità USB) in una porta tanto piccola da richiedere un'accuratezza millimetrica che non possiede.

4.3.1 Comunicazione con il robot

Al termine della pipeline di lavoro otteniamo il gruppo definitivo di informazioni che descrivono la dimostrazione eseguita:

- Classe dell'azione eseguita
- Classe dell'oggetto interessato
- Posizione e posa iniziale dell'oggetto

- Posizioni e posa finale dell'oggetto
- Eventuali posizioni e pose intermedie, in numero variabile

Tali informazioni devono essere trasferite al sistema del robot, e per fare ciò si è scelto di comunicarle attraverso la libreria `socket` di Python, che attraverso un setup molto semplice è in grado di mettere in comunicazione due host collegati alla stessa rete: l'ambiente Python relativo all'agente robotico si mette in ascolto su una porta predefinita, mentre quello relativo al sistema di analisi video apre una connessione con il primo scegliendo l'indirizzo IP e la porta adeguata. I due sono in grado di scambiarsi una stringa alla volta, che sarà composta in questo modo:

```
ACTION; x0; y0; z0; a0; b0; z0; x1; y1; z1; a1; b1; g1;
                object_class; action_class;
```

La stringa sarà passata senza spazi, e conterrà la parola fissa `ACTION` per indicare l'arrivo di una nuova azione da eseguire. Ognuna delle variabili successive sarà sostituita dal corrispettivo valore corretto: i termini `x0`, `y0` e `z0` rappresentano le coordinate di posizione dell'oggetto all'inizio dell'azione e nel sistema di riferimento del robot; i termini `a0`, `b0` e `g0` gli angoli di rotazione (in gradi) attorno ai tre assi dello stesso sistema; tutti i termini con suffisso `1` sono i corrispettivi dei precedenti ma riferiti all'istante di fine azione; infine `object_class` conterrà la descrizione dell'oggetto identificato, mentre `action_class` la classe dell'azione tra le otto predefinite.

L'ambiente Python alla base dell'agente robotico sarà responsabile dell'interpretazione della stringa, della suddivisione dei suoi pezzi e dell'opportuna trasformazione delle variabili ricevute, che verranno passate al sistema di controllo del robot per l'esecuzione. Esso ha anche il compito di rapportare le informazioni ricevute alle proprie caratteristiche intrinseche: ad esempio, il gripper del robot in posizione di partenza si trova ruotato di circa 45° attorno all'asse `z`, valore che dev'essere opportunamente sommato a quello ricevuti dal primo sistema. Inoltre, le pinze del gripper si considerano a rotazione 0° quando si trovano perpendicolari rispetto all'asse `x` del robot, fatto che dev'essere considerato nel riportare correttamente la posa stimata dell'oggetto.

4.3.2 Esecuzione dell'azione

Prima di descrivere l'esecuzione pratica delle azioni, è opportuno precisare alcune scelte che sono state intraprese al fine di garantire un test valido, ma

che non rispettano del tutto la filosofia di sviluppo del sistema. È facile notare che non sono state fornite informazioni riguardanti i frame intermedi, poiché il percorso che dovrà intraprendere il braccio verrà indicato manualmente e rispetterà le limitazioni di movimento dell'agente. Esso, inoltre, calcola delle euristiche personalizzate sulla base dell'azione e dell'oggetto considerato, di cui si conoscono le dimensioni: a titolo di esempio sull'azione di "Push" il gripper sarà posizionato al lato dell'oggetto per eseguire la spinta, in un punto dipendente dalla direzione da intraprendere e dalle sue dimensioni.

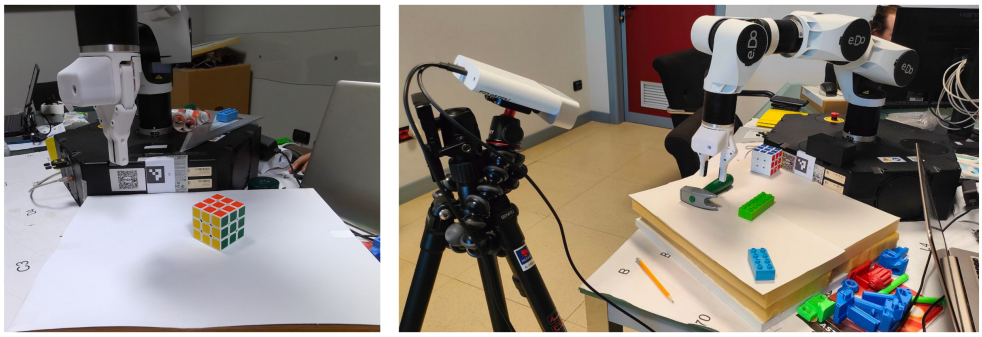


Figura 4.9: Robot e.DO in azione sulla scena di lavoro

Una volta definite queste scelte, si è potuto procedere ai test veri e propri, che hanno fornito dei risultati positivi e delle dimostrazioni di buon livello delle azioni interessate. In una prima fase sono state eseguite delle calibrazioni qualitative al fine di evitare la collisione con oggetti vicini, anche da parte del braccio robotico stesso, e successivamente si sono eseguite le azioni in maniera lineare e senza intoppi.

Alcune altre accortezze sono state messe in pratica per garantire un'esecuzione senza ostacoli: nell'esecuzione di azioni di "Screw" e "Unscrew" il barattolo coinvolto è stato selezionato di modo da essere abbastanza pesante da rimanere fermo sul posto, e il suo tappo è stato preventivamente e leggermente svitato per pareggiare le limitazioni di forza di e.DO. Inoltre, per eseguire al meglio le azioni, esse sono state dimostrate in una parte abbastanza centrale del piano di lavoro, indicativamente tra i 15cm e i 25cm di distanza dal marker ArUco, poiché è stata individuata come una zona dove il gripper del robot potesse arrivare in posa sufficientemente verticale per garantire un corretto *grasp* dell'oggetto coinvolto.

Capitolo 5

Conclusioni

5.1 Considerazioni finali

Il lavoro che è stato presentato nei precedenti capitoli ha avuto modo di dimostrare che l'approccio all'imitazione di azioni umane da parte di un agente robotico può concentrarsi sull'analisi della dimostrazione e di tutti i dati in input a essa correlati, evitando gli approcci classici che riguardano esclusivamente il robot e l'apprendimento da esso condizionato.

Le moderne tecnologie di Machine Learning, sebbene abbiano offerto delle soluzioni interessanti in termini di apprendimento legato a dati robotici, hanno visto un forte crescita negli ultimi anni nel campo della Computer Vision, mettendo a disposizione tecnologie di analisi video e manipolazione di dati grafici sempre più efficaci. Tutto ciò rende più semplice far eseguire delle azioni a un robot partendo dalla profonda comprensione del movimento che dovrà portare a termine, avendo maggiore consapevolezza di ciò che deve fare. Il progetto sin qui analizzato ha un focus più forte sul "cosa", piuttosto che sul "come", relativamente all'azione protagonista dell'analisi.

Il sistema sviluppato non è certo privo di difetti, tantomeno si può considerare pronto per l'applicazione in casi reali. Alcune delle imperfezioni che è necessario denotare riguardano la velocità della pipeline del progetto, l'accuratezza del riconoscimento dell'azione e la qualità della stima della posa dell'oggetto, tutti candidati ideali per dei miglioramenti mirati. Inoltre, la filosofia di approccio selezionata all'inizio lega lo sviluppo del sistema a un set di azioni e oggetti ben definiti, che non è concorde con la prospettiva di adattare il sistema a situazioni e ambienti nuovi.

Allo stesso tempo si possono fare alcune considerazioni sui risultati calcolati: l'inaccuratezza rilevata sui test precedenti, che risulta nell'ordine di alcuni

millimetri, non è stata particolarmente d'ostacolo a una corretta esecuzione dell'azione, soprattutto nei test che hanno coinvolto direttamente il robot. È opportuno sottolineare che questo tipo di errore, nel contesto delle azioni che sono state scelte, non rappresenta necessariamente una problematica da risolvere a tutti i costi: la filosofia di applicazione di sistemi come questo appartiene sia ad ambienti propri dell'industria 4.0 moderna che a situazioni di collaborazione domestica tra uomo e robot, per le quali è importante replicare correttamente l'azione entro un certo grado di accuratezza, ma non necessariamente al netto di un'esattezza millimetrica.

Certamente questa considerazione non è più valida parlando di ambienti e azioni dov'è necessario eseguire il movimento sia con cura che in modo accurato, ad esempio per manipolare oggetti che vanno posizionati in cavità e spazi ben precisi, ma ad ogni modo un approccio come quello descritto in questo progetto si mette nella condizione di essere utile per una grande quantità di casistiche e applicazioni, proponendo una base di partenza forte per lo sviluppo di soluzioni che siano ben focalizzate sull'analisi sia dei dati in input che dell'ambiente che circonda il robot coinvolto.

5.2 Sviluppi futuri

Appare piuttosto evidente che il sistema sviluppato presenti un grande potenziale in termini di sviluppi futuri, poiché esso coinvolge numerosi ambiti di ricerca propri del mondo della Computer Vision e del Machine Learning, i quali possono essere ulteriormente approfonditi per ricercare le più corrette soluzioni e ottimizzare il processo di analisi del sistema.

Innanzitutto, come già citato, un primo progresso potrebbe arrivare dal miglioramento della velocità generale del sistema, che soffre dei passaggi riguardanti soprattutto le reti di riconoscimento di azione e interazione tra mano e oggetto. Molto probabilmente questo genere di sviluppo passerebbe dalla ricerca di modelli che abbiano un buon rapporto tra qualità dell'output e rapidità di calcolo, anche se potrebbe essere risolto dall'utilizzo di una maggiore potenza di calcolo.

In seconda istanza, la presenza di un set di azioni e oggetti limitato suggerisce l'ampliamento di questi due gruppi, in favore di una maggiore generalizzazione a casi d'uso differenti. Un'idea molto interessante sarebbe quella di addestrare un modello a riconoscere un set di azioni "base", similmente a quanto fatto in questo lavoro, di modo da poter analizzare azioni più lunghe e complesse e scomporle in sotto-azioni riconducibili a quelle di allenamento:

un esempio semplice è quello di un'azione dove il dimostratore raccoglie un tappo in scena e lo avvita sull'apposito barattolo, che significherebbe riconoscere nel video in ingresso le azioni di "Pick and place" e "Screw", una di seguito all'altra.

Al contempo, un'idea sicuramente valida è quella di migliorare e ampliare il fine-tuning della rete di riconoscimento dell'azione, raccogliendo un numero maggiore di video che includano una varietà sempre più ampia di oggetti, magari aggiungendo all'allenamento, oltre alle già usate immagini di colore e profondità, anche un'informazione sulla posa della mano del dimostratore in scena, di modo da facilitare il processo di riconoscimento della classe dell'azione.

Mantenendo il focus sul movimento effettuato nella dimostrazione, si potrebbe pensare di raccogliere e aggiungere informazioni sull'ambiente di lavoro riguardanti le cosiddette *affordances* e *avoidance*, idee che sono state già introdotte in altri progetti di maggiore livello: la prima riguarda la capacità di alcuni punti della scena di suggerire la loro "manipolabilità", ovvero quanto sono avvezzi a prendere parte all'azione, mentre la seconda suggerisce quali possono essere punti dell'ambiente che devono essere evitati, in quanto zone di ostacolo allo svolgimento dell'azione o contenenti altri oggetti più fragili.

Un punto del progetto che potrebbe un grande potenziale di miglioramento è quello riguardante la stima della posa, che in questo lavoro è stata eseguita seguendo delle euristiche "locali" e metodi di Computer Vision più classici: come già citato, la pose estimation è un campo di ricerca attivo e che sta crescendo molto negli ultimi anni, soprattuto sui modelli in grado di valutare con grande affidabilità la rotazione di un oggetto nello spazio. Nel contesto di questo progetto si è evitato di sovraccaricare la pipeline con un ulteriore rete neurale, scelta che avrebbe rallentato ulteriormente il flusso di produzione dei risultati, ma con soluzioni sufficientemente veloci potrebbe risultare una decisione molto valida.

La presenza in scena di una sola telecamera in posizione fissa è stata sicuramente una limitazione, ma ciò potrebbe essere ovviato dall'uso di dispositivi più moderni o addirittura in numero maggiore, scelta che potrebbe permettere di ottenere una mappatura più completa dell'ambiente di lavoro e facilitare anche alcune fasi del progetto attuale. Inoltre, da una visione a più ampio spettro dell'area di movimento si potrebbe aggiungere l'informazione riguardante la presa che la mano effettua sull'oggetto e tradurla in una relativa presa del gripper del robot, al fine di raffinare l'imitazione dell'azione in tutti i suoi dettagli.

Infine, il mondo del Machine Learning moderno è fortemente guidato dallo

sviluppo dei cosiddetti modelli di linguaggio (NLM/LLM), i quali costituiscono una delle tecnologie di maggiore potenziale diffusesi nell'ultimo periodo: visto il livello di analisi semantica dell'azione che è stato introdotto in questo lavoro, potrebbe risultare di grande interesse l'idea di integrare il linguaggio naturale al fine di raffinare o spiegare il movimento desiderato al robot, aprendo la strada a numerose implementazioni che, in futuro, potrebbero ampliare l'usabilità e l'accessibilità del sistema stesso.

Bibliografia

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[2] Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. Human-to-robot imitation in the wild. 2022.

[3] Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics. 2023.

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2:

Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*, 2023.

[5] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*, 2022.

[6] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[7] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018.

[8] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[9] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, Siamak Shakeri, Mostafa Dehghani, Daniel Salz, Mario Lucic, Michael Tschannen, Arsha Nagrani, Hexiang Hu, Mandar Joshi, Bo Pang, Ceslee Montgomery, Paulina Pietrzyk, Marvin Ritter, AJ Piergiovanni, Matthias Minderer, Filip Pavetic, Austin Waters, Gang Li, Ibrahim Alabdulmohsin, Lucas Beyer, Julien Amelot, Kenton Lee, Andreas Peter Steiner, Yang Li, Daniel Keysers, Anurag Arnab, Yuanzhong Xu, Keran Rong, Alexander Kolesnikov, Mojtaba Seyedhosseini, Anelia Angelova, Xiaohua Zhai, Neil Houlsby, and Radu Soricut. Pali-x: On scaling up a multilingual vision and language model, 2023.

[10] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*,

2024.

[11] André Correia and Luís A. Alexandre. A survey of demonstration learning, 2023.

[12] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

[13] Jiafei Duan, Yi Ru Wang, Mohit Shridhar, Dieter Fox, and Ranjay Krishna. Ar2-d2: Training a robot without a robot. 2023.

[14] Chrisantus Eze and Christopher Crick. Learning by watching: A review of video-based learning approaches for robot manipulation, 2024.

[15] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Manuel Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47:2280–2292, 06 2014.

[16] Ibai Gorordo Fernandez. pyKinectAzure. https://github.com/ibaiGorordo/pyKinectAzure.

[17] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, Priya Sundaresan, Peng Xu, Hao Su, Karol Hausman, Chelsea Finn, Quan Vuong, and Ted Xiao. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches, 2023.

[18] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[19] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-z: Zero-shot task generalization with robotic imitation learning. In *5th Annual Conference on Robot Learning*, 2021.

[20] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023.

[21] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control, 2018.

[22] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr

Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[23] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[24] Matthew B. Luebbers, Connor Brooks, Carl L. Mueller, Daniel Szafir, and Bradley Hayes. Arc-lfd: Using augmented reality for interactive long-term robot skill maintenance via constrained learning from demonstration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3794–3800, 2021.

[25] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training, 2022.

[26] Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Scaling open-vocabulary object detection, 2024.

[27] Mathew Monfort, Bowen Pan, Kandan Ramakrishnan, Alex Andonian, Barry A McNamara, Alex Lascelles, Quanfu Fan, Dan Gutfreund, Rogerio Feris, and Aude Oliva. Multi-moments in time: Learning and interpreting models for multi-action video understanding, 2019.

[28] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation, 2022.

[29] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[31] Yu Rong, Takaaki Shiratori, and Hanbyul Joo. Frankmocap: A monocular 3d whole-body pose estimation system via regression and integration. In *IEEE International Conference on Computer Vision Workshops*, 2021.

[32] Dandan Shan, Jiaqi Geng, Michelle Shu, and David Fouhey. Understanding human hands in contact at internet scale. 2020.

[33] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Proceedings of the 5th Conference on Robot Learning (CoRL)*, 2021.

[34] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[35] Aravind Sivakumar, Kenneth Shaw, and Deepak Pathak. Robotic telekinesis: Learning a robotic hand imitator by watching humans on youtube, 2022.

[36] Inês Soares, Marcelo Petry, and António Paulo Moreira. Programming robots by demonstration using augmented reality. *Sensors*, 21(17), 2021.

[37] Arturo Daniel Sosa-Ceron, Hugo Gustavo Gonzalez-Hernandez, and Jorge Antonio Reyes-Avendaño. Learning from demonstrations in human–robot collaborative scenarios: A survey. *Robotics*, 11(6), 2022.

[38] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.