

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Integrazione Sensoriale di Robot Umanoide Unitree G1

Unitree G1 Humanoid Robot Sensory Integration

Candidato:
Filippo Campenni

Relatore:
Prof. Matteo Claudio Palpacelli

Correlatore:
Dott. Federico Neri

Anno Accademico 2025-2026



UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica

Integrazione Sensoriale di Robot Umanoide Unitree G1

Unitree G1 Humanoid Robot Sensory Integration

Candidato:
Filippo Campenni

Relatore:
Prof. Matteo Claudio Palpacelli

Correlatore:
Dott. Federico Neri

Anno Accademico 2025-2026

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Meccanica
Via Brecce Bianche – 60131 Ancona (AN), Italy

Alla mia famiglia, ai miei amici e a chiunque creda in me.

Sommario

La robotica umanoide rappresenta uno dei campi di ricerca ed applicazione più avanzati della meccatronica, dove l'autonomia operativa in ambienti strutturati nativamente per l'essere umano dipende criticamente dalla percezione e comprensione spaziale e semantica dell'ambiente.

Il presente elaborato tratta l'integrazione sensoriale sulla piattaforma robotica umanoide **Unitree G1 EDU**, focalizzandosi su due ambiti fondamentali: il riconoscimento e classificazione di oggetti in tempo reale (**Object Detection Live**) e la localizzazione e mappatura tridimensionale simultanea (**SLAM**).

Per quanto riguarda l'*Object Detection Live*, il compito di elaborazione visiva è affidato alla telecamera RGB-D Intel RealSense D435i in dotazione al robot ed al modello di rete neurale convoluzionale **Ultralytics YOLO11**. Sfruttando la piattaforma *web-based* **Roboflow**, è stato creato, annotato ed elaborato da zero un dataset dedicato ad oggetti target di varia dimensione, forma e colore. Adottando una metodologia iterativa, lo sviluppo si è articolato in tre sessioni di addestramento della rete neurale, permettendo di ridurre progressivamente le criticità di ambiguità geometrica ed i Falsi Positivi indotti da elementi di disturbo sullo sfondo. Il modello addestrato finale garantisce buone prestazioni di accuratezza e sensibilità unite a minime latenze computazionali durante le prove sperimentali in tempo reale.

Per la parte di SLAM e navigazione autonoma, l'integrazione fa uso del sensore LiDAR Livox Mid-360, anch'esso in dotazione al robot umanoide. Attraverso la ricostruzione dell'ambiente circostante mediante nuvola di punti (*point cloud*), si può redigere uno studio sul suo utilizzo per permettere la localizzazione e la navigazione autonoma del robot umanoide attraverso dei punti di passaggio (*waypoint*) di interesse. Inoltre si sperimenta e discute la fase di *post-processing* della *point cloud* mediante filtraggio statistico ottenendo risultati eccellenti.

Infine, l'architettura software realizzata su sistema operativo **Ubuntu 22.04** e middleware **ROS 2 Humble** dimostra come la sinergia tra la navigazione globale basata sul LiDAR ed il riconoscimento semantico visivo a corto raggio tramite telecamera RGB-D costituisca la base necessaria per sviluppare future operazioni complesse di evitamento dinamico di ostacoli, manipolazione fine e *grasping*.

Indice

Nomenclatura **xv**

1 Introduzione **1**
1.1 Obiettivi . . . 2
1.2 Struttura del documento . . . 2

2 Stato dell'Arte **5**

3 Struttura Hardware **9**
3.1 Unitree G1 EDU . . . 9
3.2 Intel RealSense D435i . . . 12
3.3 LiDAR Livox Mid-360 . . . 13

4 Struttura Software **15**
4.1 Ubuntu . . . 15
4.2 ROS 2 . . . 16
4.3 Ultralytics YOLO . . . 17
4.4 Roboflow . . . 19
4.5 Unitree SDK . . . 20

5 Object Detection Live **23**
5.1 Dataset . . . 23
5.2 Processo di Addestramento . . . 25
5.3 Risultati di Addestramento e loro Valutazione . . . 28
5.3.1 Metriche di Valutazione . . . 28
5.3.2 Descrizione dei Grafici di Output e loro Confronto . . . 30
5.4 Valutazione Sperimentale e Misure di Miglioramento . . . 43
5.4.1 train-1 . . . 46
5.4.2 train-2 . . . 46
5.4.3 train-3 . . . 48

6 SLAM **51**
6.1 Architettura Software dello SLAM . . . 51
6.2 Point Cloud . . . 52

7 Conclusioni **57**
7.1 Sintesi dei Risultati . . . 57

7.2 Sviluppi Futuri 58

Elenco delle figure

2.1 Robot umanoide WABOT-1. 6

3.1 Robot umanoide Unitree G1. 10
3.2 Struttura del robot umanoide Unitree G1. 10
3.3 Moduli RGB-D e LiDAR posizionati sul capo del robot umanoide Unitree G1 EDU. 11
3.4 Campo visivo (FOV) del sensore LiDAR Livox Mid-360 e della telecamera RGB-D Intel RealSense D435i. 12
3.5 FOV del sensore LiDAR Livox Mid-360 unito al FOV della telecamera RGB-D Intel RealSense D435i. 12

4.1 Ecosistema ROS 16

5.1 Oggetti target dell'*Object Detection Live*. 24
5.2 Matrici di Confusione relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 34
5.3 Matrici di Confusione Normalizzate relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 35
5.4 Curve Precision-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 37
5.5 Curve Recall-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 39
5.6 Curve Precision-Recall relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 41
5.7 Curve F1-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c). 43
5.8 Setup delle prove sperimentali. 45
5.9 Riconoscimento oggetti del `train-2`, da notare il Falso Positivo e la confusione inter-classe. 47
5.10 Riconoscimento oggetti del `train-2`, si osserva l'assenza di errori di riconoscimento. 48
5.11 Riconoscimento oggetti del `train-3`, si evidenzia l'accuratezza di riconoscimento. 49
5.12 Riconoscimento oggetti del `train-3`, da notare le due bounding box sovrapposte. 49

6.1 Prima vista prospettica della *point cloud* grezza rilevata dal sensore LiDAR del robot umanoide Unitree G1 EDU. 53
6.2 Seconda vista prospettica della *point cloud* grezza rilevata dal sensore LiDAR del robot umanoide Unitree G1 EDU. 53
6.3 Prima vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico. 54
6.4 Seconda vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico. 55
6.5 Terza vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico. 55

Elenco delle tabelle

3.1 Specifiche del robot umanoide Unitree G1 EDU. 9

5.1 Parametri di Augmentation applicati ai $\mathcal{D}_{\text{train}}$ di train-2 e train-3. . 26
5.2 Tempi computazionali di addestramento. 27

Nomenclatura

Acronimo	Descrizione
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detection and Ranging / Laser Imaging Detection and Ranging
SDK	Software Development Kit
FOV	Field of View (Campo Visivo)
DoF	Degrees of Freedom (Gradi di Libertà)
PMSM	Permanent Magnet Synchronous Motors
IMU	Inertial Measurement Unit

Capitolo 1

Introduzione

La robotica umanoide rappresenta uno dei campi di ricerca e sviluppo più complessi ed interdisciplinari dell'ingegneria meccatronica, dell'automazione e dell'intelligenza artificiale. L'obiettivo primario di questa disciplina è la progettazione e la realizzazione di sistemi meccatronici ad elevato numero di gradi di libertà, antropomorfi nella struttura e nella cinematica, in grado di muoversi, percepire l'ambiente ed operare all'interno di contesti nativamente strutturati per la figura umana. L'autonomia operativa in ambienti di questa tipologia dipende in modo critico dalla capacità di elaborare informazioni spaziali e semantiche in tempo reale. Mentre la navigazione globale ed il posizionamento spaziale del robot umanoide sono affidati a sistemi di mappatura tridimensionale, l'interazione fine e la manipolazione sul piano di lavoro richiedono un'architettura percettiva dedicata, capace di coniugare un'elevata discriminazione visiva con latenze computazionali minime.

Il compito di elaborare informazioni visive, cromatiche e semantiche è affidato all'architettura di **Object Detection Live / Riconoscimento Oggetti in tempo reale**, cioè il processo computazionale attraverso il quale un sistema di visione artificiale elabora in continuo un flusso video in ingresso (acquisito tramite sensoristica RGB-D). Lo scopo è identificare e localizzare istantaneamente oggetti noti all'interno della scena, restituendo per ciascuno di essi un'etichetta categoriale, che definisce la sua classe di appartenenza, con la sua relativa *bounding box*. Gli oggetti noti appena citati sono definiti **oggetti target**, cioè le specifiche entità fisiche di interesse operativo che l'algoritmo di visione deve essere in grado di isolare e classificare, discriminandole sia dal rumore di fondo che da eventuali elementi di disturbo visivamente simili.

La mansione di localizzazione spaziale e navigazione autonoma è affidata agli algoritmi di **SLAM** (*Simultaneous localization and mapping*), mediante l'elaborazione in tempo reale delle nuvole di punti (*point cloud*) fornite dal sensore LiDAR, lo SLAM risolve il problema della navigazione spaziale ed autonoma fino al raggiungimento della postazione di lavoro designata. Solo a posizionamento ultimato subentra il sistema di *Object Detection Live* per l'eventuale manipolazione a corto raggio.

1.1 Obiettivi

L'obiettivo generale del presente elaborato consiste nell'integrazione sensoriale dei sistemi di percezione visiva e mappatura spaziale sulla piattaforma umanoide di riferimento, cioè il robot umanoide Unitree G1 EDU.
Per l'ambito di **Object Detection Live** l'obiettivo consiste nel raggiungere un riconoscimento accurato di tutti gli oggetti target inquadrati dalla telecamera RGB-D in dotazione al robot umanoide, la loro corretta classificazione ed il calcolo di una stima della posizione di ogni oggetto target rispetto alla telecamera stessa. Ciò si può ottenere creando un dataset correttamente strutturato e vario, applicando le corrette trasformazioni alle immagini del dataset, definendo i parametri di addestramento appropriati ed infine implementando adeguatamente l'addestramento al sensore RGB-D.
Per l'integrazione del sensore LiDAR in dotazione al robot umanoide e la sua implementazione per eseguire lo **SLAM**, l'obiettivo consiste nell'eseguire una corretta ed efficace mappatura dell'ambiente circostante, per poi ottenerne una versione pulita dai dati indesiderati nella fase di post-elaborazione mediante filtraggio statistico. Inoltre si vuole definire l'architettura software (insieme al processo per la sua implementazione) per il futuro ottenimento della localizzazione spaziale e navigazione autonoma del robot umanoide attraverso la definizione di punti di passaggio sequenziali prefissati.

1.2 Struttura del documento

Il presente elaborato è organizzato come segue:

- Il Capitolo 2 "Stato dell'Arte" riguarda il settore della robotica umanoide ed i suoi sviluppi recenti, considerando le loro applicazioni reali, per poi entrare nel dettaglio sull'utilizzo dei sistemi di visione sui robot umanoidi.

- Il Capitolo 3 "Struttura Hardware" descrive la struttura hardware adottata. In particolare si discutono le caratteristiche e le specifiche tecniche del robot umanoide Unitree G1 EDU, della telecamera RGB-D e del sensore LiDAR. Entrambi i sensori sono in dotazione al robot umanoide.

- Il Capitolo 4 "Struttura Software" descrive l'architettura e gli strumenti software utilizzati per l'integrazione sensoriale. Verranno descritti il sistema operativo Ubuntu, l'ecosistema ROS 2, la famiglia di reti neurali Ultralytics YOLO, la piattaforma Roboflow e gli SDK dedicati ai robot Unitree.

- Il Capitolo 5 "Object Detection Live" illustra interamente il processo di integrazione sensoriale della telecamera RGB-D in dotazione al robot umanoide Unitree G1 EDU con l'obiettivo di riconoscimento oggetti in tempo reale. Vengono trattati le fasi di creazione ed elaborazione del dataset, il processo di

addestramento e la valutazione sia analitico-grafica che sperimentale delle reti neurali addestrate.

- Il Capitolo 6 "SLAM" tratta l'integrazione del sensore LiDAR in dotazione al robot umanoide con finalità di mappatura tridimensionale ambientale. Si implementa inoltre la fase di *post-processing* della *point cloud*.
- Infine, il Capitolo 7 "Conclusioni" considera le applicazioni di integrazione sensoriale eseguite e valutate in precedenza traendone le conclusioni ed illustrandone i possibili sviluppi futuri.

Capitolo 2

Stato dell'Arte

Il primo robot umanoide in scala reale della storia è il **WABOT-1**, sviluppato nel 1973 presso la *Waseda University* di Tokyo dal gruppo di ricerca guidato dal Professor Ichiro Kato [1]. Disponeva della capacità di deambulazione bipede permettendogli di compiere lentamente dei passi su superfici piane, gli arti superiori articolati e le mani artificiali gli rendevano possibile afferrare e spostare oggetti di forma semplice, inoltre i suoi sistemi visivi ed acustici gli rendevano possibile stimare le distanze e riconoscere dei comandi vocali in lingua giapponese a cui era in grado di dare risposta.

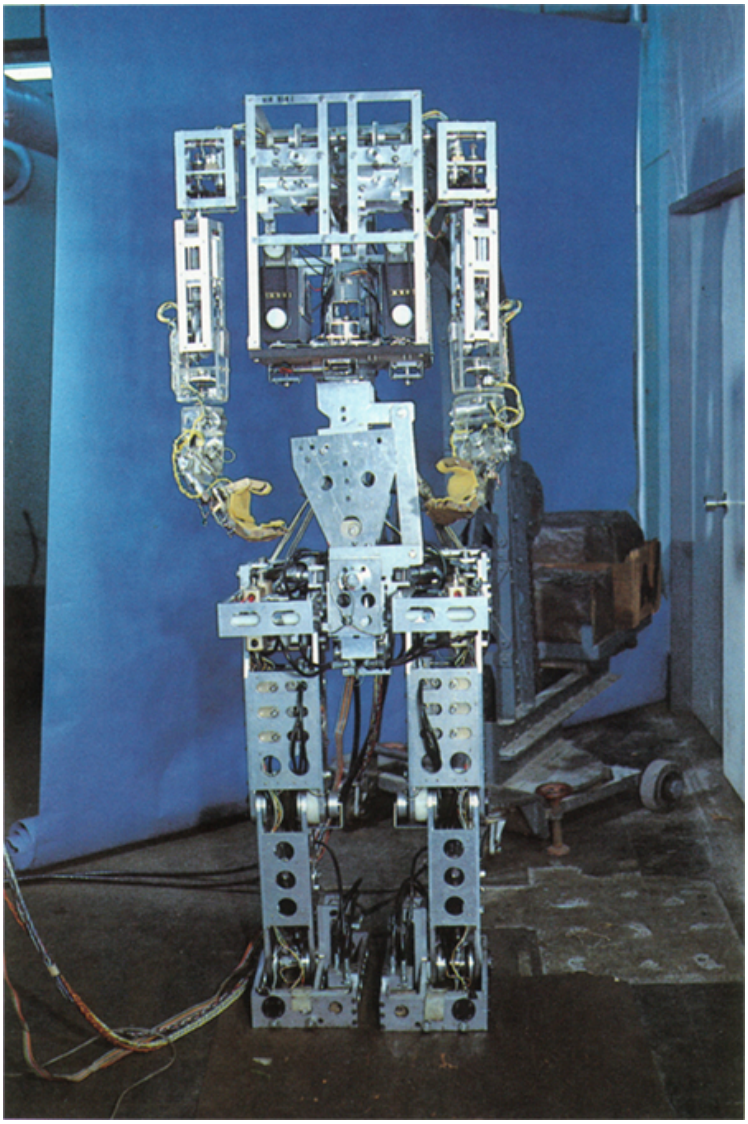


Figura 2.1: Robot umanoide WABOT-1.

Negli ultimi anni la tecnologia ha fatto grandissimi passi in avanti, attualmente il mercato globale della robotica umanoide sta attraversando una fase di grande crescita risultando come una delle industrie più all'avanguardia ed innovative. Il settore è valutato a circa 10.9 miliardi di dollari e si registrano proiezioni di crescita che stimano il raggiungimento dei 190 miliardi di dollari entro il 2035 [2]. A guidare il settore della robotica umanoide e dell'intelligenza artificiale incorporata (*Embodied AI*) è la Cina che con la sua vasta ed avanzata filiera industriale di produzione ed integrazione di componenti meccatronici rende possibile una rapida prototipazione e commercializzazione di robot umanoidi. Di fondamentale importanza sono i massicci investimenti strategici che la Cina sta erogando per garantirsi il primato sul settore, in particolare è stato istituito e mobilitato un fondo di investimento dal valore di 1000 miliardi di Yuan (138 miliardi di dollari) per sostenere start-up ed imprese

attive nei settori di Intelligenza Artificiale, Embodied AI e Robotica Umanoide [3].

Tra i modelli di robot umanoidi più avanzati e diffusi si possono evidenziare:

- *Atlas* di Boston Dynamics [4], si sta implementando negli stabilimenti Hyundai con compiti di movimentazione carichi e nel sequenziamento dei componenti d'assemblaggio grazie alla sua eccellente dinamica, navigazione autonoma e capacità di sostituzione della batteria in completa autonomia.

- *Optimus* di Tesla [5], progettato sia per scopi industriali che collaborativi ed impiegato nei loro stabilimenti industriali per eseguire mansioni fisicamente usuranti e monotone per l'essere umano.

- *Figure 03* di Figure AI [6], appositamente progettato per operare in contesti di produzione ed assemblaggio automobilistico di alta precisione, è attualmente impiegato da BMW in uno dei suoi stabilimenti.

- *Digit* di Agility Robotics [7], progettato appositamente per la logistica ed ottimizzato per mantenere la stabilità durante la movimentazione di carichi verticali è impiegato da Amazon nei propri magazzini.

- *Walker S* di UBTECH Robotics [8], attualmente impiegato nelle *Dark Factories* (stabilimenti industriali privi di illuminazione e climatizzazione) di BYD per la produzione di veicoli elettrici, possibile grazie ai suoi avanzati sistemi di percezione e localizzazione ambientale.

Di maggior rilevanza per il presente elaborato è l'azienda **Unitree Robotics** con sede ad Hangzhou (Cina) alla quale è attribuibile circa il 30% della produzione mondiale di robot umanoidi [9]. Unitree G1 EDU, il suo modello di punta per l'ambito di ricerca accademica, è ampiamente utilizzato per lo studio di cinematica, dinamica, integrazione sensoriale ed esecuzione di operazioni in autonomia quali attività di riconoscimento e presa, movimentazione e manipolazione di oggetti (*grasping*).
Xin Wen et al.[10] hanno ottenuto ottimi risultati nelle simulazioni di movimento bipede del robot Unitree G1 su terreni ripidi ed irregolari e su scalinate con scalini di diversa dimensione. André Schakkal et al.[11] hanno condotto prove sperimentali nelle quali il G1 è riuscito a riconoscere e togliere dalla sua visuale l'oggetto indesiderato per poi afferrare l'oggetto target, la cui attività di grasping era precedentemente intralciata dall'oggetto indesiderato. Caleb Hodgson-Taylor et al.[12] sfruttando le reti neurali di *Object Detection* con riconoscimento di *Oriented Bounding Box* ha ottenuto un ottimo riconoscimento di diversi oggetti. Travis Ryan [13] sfruttando i modelli di *Pose Estimation* è riuscito a far replicare correttamente all'Unitree G1 EDU i movimenti degli arti superiori di un operatore umano ottenuto eseguendo l'analisi cinematica inversa e stima della posa tramite acquisizione di punti chiave sul corpo. Yiyang Mu [14] ha integrato più modelli di reti neurali convoluzionali rendendo l'Unitree G1 EDU capace di eseguire processi di riconoscimento facciale, così da determinare chi è la persona che si trova davanti ed interagire con questa

persona in modo differente sulla base dell'essere umano che ha riconosciuto.
Rimane ovvio comprendere e riconoscere le potenzialità attuali e future della robotica umanoide e della loro integrazione sensoriale.

Capitolo 3

Struttura Hardware

3.1 Unitree G1 EDU

La piattaforma robotica utilizzata per l'integrazione sensoriale e la loro validazione sperimentale è il robot umanoide **Unitree G1** nella sua variante **EDU** progettata per centri di ricerca e laboratori accademici [15]. Il robot Unitree G1 EDU presenta un'architettura antropomorfa guidata da un design strutturale ottimizzato per mantenere una massa contenuta nonostante le sue dimensioni siano compatibili con l'utilizzo in ambienti a contatto con l'essere umano. Si distingue dalla versione *Standard* per la dotazione di una sensoristica più completa ed avanzata, una maggiore potenza di calcolo integrata, un maggior numero di DoF ed un completo accesso all'SDK sia per il controllo *High-Level* che *Low-Level*.

Tabella 3.1: Specifiche del robot umanoide Unitree G1 EDU.

Parametro	**G1 EDU**
DoF totali	43
Ingombro in posizione eretta	$1320 \times 450 \times 200$ mm
Ingombro in configurazione ripiegata da trasporto	$690 \times 450 \times 300$ mm
Massa (batteria inclusa)	35 kg
Massimo momento torcente (al ginocchio)	$120\ \mathrm{N} \cdot \mathrm{m}$
Carico utile massimo per braccio	≈ 3 kg
Massima velocità di movimento	2 m/s
Autonomia della batteria	≈ 2 h
Capacità nominale della batteria	9000 mAh
Sviluppo secondario (SDK)	Abilitato (High-Level / Low-Level)

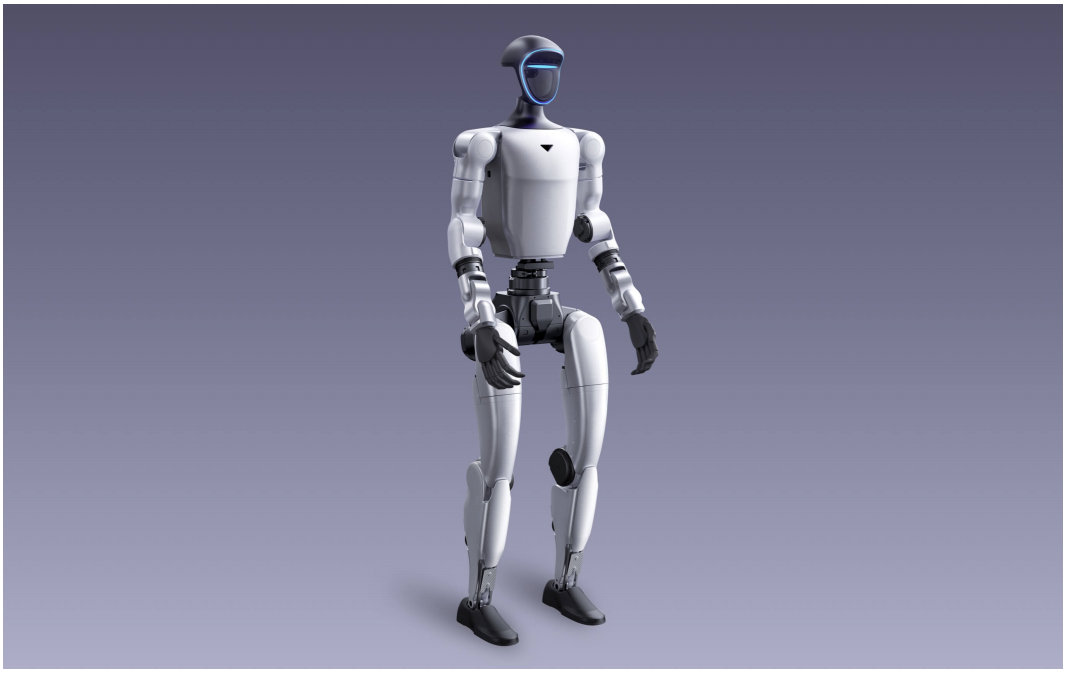


Figura 3.1: Robot umanoide Unitree G1.

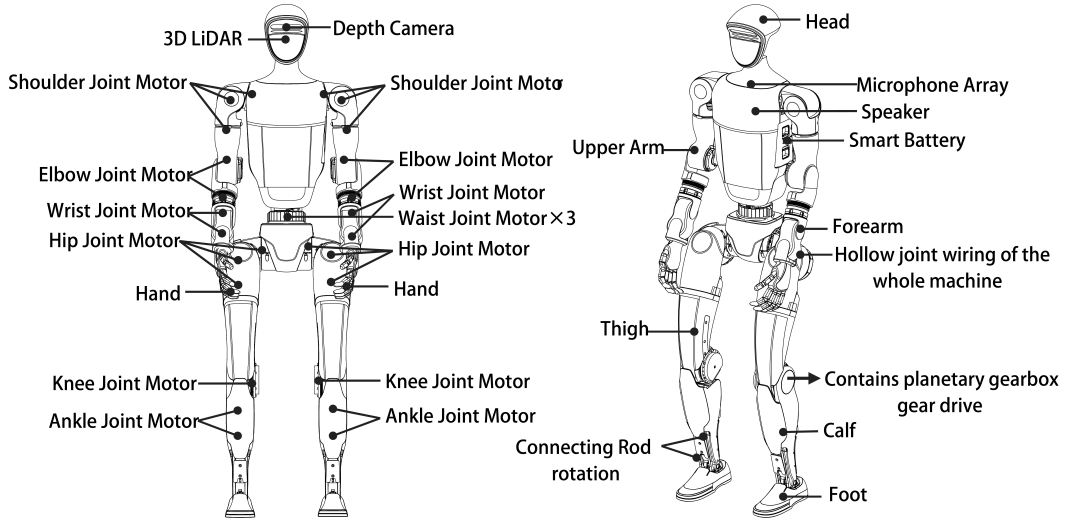


Figura 3.2: Struttura del robot umanoide Unitree G1.

L'attuazione motoria è affidata a servomotori sincroni a magneti permanenti PMSM (*Permanent Magnet Synchronous Motors*) a rotore interno a bassa inerzia. Tale scelta costruttiva assicura elevate velocità di risposta dinamica, ingombri radiali contenuti ed un'efficace dissipazione del calore generato tramite raffreddamento ad aria locale. Nello studio della dinamica e del controllo dei robot umanoidi, gli snodi ad elevate prestazioni (come l'articolazione del ginocchio) costituiscono i punti critici dell'intera catena cinematica. Gli accoppiamenti tra servomotore PMSM ed albero di uscita sono supportati da cuscinetti volventi a rulli incrociati ad elevata capacità

di carico (*Industrial-grade Crossed Roller Bearings*), capaci di sostenere forze assiali, radiali e momenti flettenti tipici del movimento antropomorfo. Inoltre la stima accurata della posa del robot (*Pose Estimation*) è acquisita tramite un sistema a doppio encoder assoluto per giunto. L'elevata accuratezza di misura dell'encoder è garantita dal dimensionamento per interferenza dei cuscinetti volventi a rulli che va ad annullare il gioco meccanico. Da notare in Figura 3.2 come per raggiungere la coppia di picco di 120 $\mathrm{N} \cdot \mathrm{m}$ nei giunti delle ginocchia la Unitree ha deciso di accoppiare i servomotori PMSM all'albero di uscita con riduttori epicicloidali, che permettono di ottenere un elevato rapporto di trasmissione (definito pari a $\omega_{\text{movente}}/\omega_{\text{cedente}}$) pur mantenendo l'ingombro ridotto. I riduttori epicicloidali sono compatibili con il funzionamento dei robot umanoidi anche per la loro ottima resistenza ad urti e carichi impulsivi.
Il punto di forza del G1 EDU risiede nella predisposizione per lo sviluppo secondario, tramite interfaccia Ethernet dedicata è possibile stabilire la comunicazione ad alta frequenza e bassa latenza tra la scheda di bordo del robot e la workstation esterna/PC di sviluppo. Fondiamo la nostra applicazione di integrazione sensoriale sull'utilizzo dei due sensori necessari per il riconoscimento e localizzazione presenti nel capo dell'**Unitree G1 EDU** [16]:

Telecamera di profondità RGB-D (Intel RealSense D435i)

LiDAR 3D (Livox Mid-360)



Figura 3.3: Moduli RGB-D e LiDAR posizionati sul capo del robot umanoide Unitree G1 EDU.

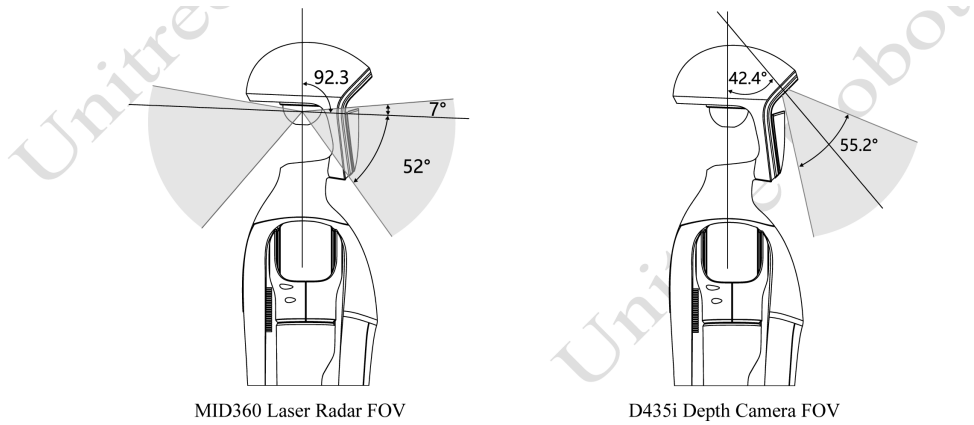


Figura 3.4: Campo visivo (FOV) del sensore LiDAR Livox Mid-360 e della telecamera RGB-D Intel RealSense D435i.

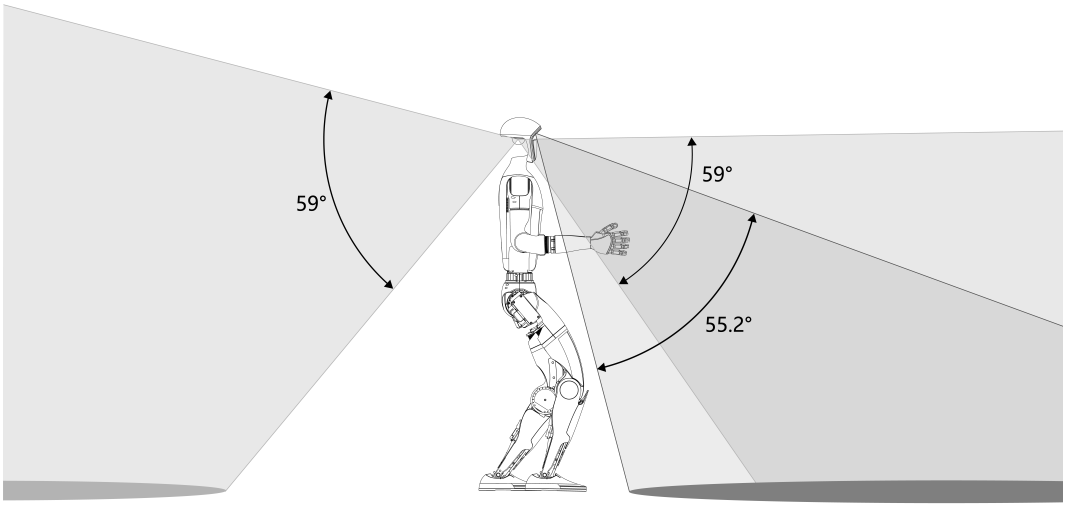


Figura 3.5: FOV del sensore LiDAR Livox Mid-360 unito al FOV della telecamera RGB-D Intel RealSense D435i.

3.2 Intel RealSense D435i

La telecamera a colori e profondità sincronizzata RGB-D (*Depth Camera*) la troviamo in dotazione al robot umanoide Unitree G1 EDU sulla porzione frontale della testa. Si occupa della classificazione e localizzazione fine a corto raggio degli oggetti target presenti sul piano di lavoro. La telecamera sfrutta un'architettura optoelettronica basata sulla visione stereo attiva ad infrarossi (*Active IR Stereo*). Il modulo hardware racchiude quattro componenti principali:

- Coppia di Sensori Infrarossi (*Stereo IR Imagers*), due sensori ad infrarossi posizionati ad una distanza di interasse nota.

- Proiettore di Pattern Infrarosso (*IR Dot Projector*), un emettitore attivo che proietta sulla scena un *pattern* di punti non ripetitivi invisibili all'occhio umano. Tale proiezione è fondamentale in contesti di laboratorio con pareti uniformi, superfici riflettenti o prive di texture visibile, consentendo agli algoritmi di Stereo Matching di trovare la corrispondenza tra i pixel omologhi dei due *Stereo IR Imagers*.
- Sensore Cromatico RGB, un sensore ottico ad alta definizione (1920×1080 pixel a 30 FPS).
- Unità di Misura Inerziale IMU (*Inertial Measurement Unit*), integra un accelerometro ed un giroscopio triassiali necessari per eseguire una stima sufficientemente corretta delle coordinate cartesiane 3D dell'oggetto target anche in situazioni dove la telecamera, e quindi il capo del robot umanoide, è in movimento.

La distanza ideale dalla telecamera per garantirne le giuste prestazioni è compresa tra 0,3 m e 3 m. La minima distanza per il calcolo della profondità dell'oggetto target è di ≈ 28 cm. Dalla Figura 3.4 possiamo notare come l'asse ottico della telecamera RGB-D sia inclinato di 42,4° rispetto all'asse verticale, questa configurazione è preferita rispetto a posizionare l'asse ottico orizzontale in quanto concede al robot umanoide la massima copertura visiva dell'area di lavoro (ad esempio un banco da laboratorio) immediatamente antistante a lui.
Le caratteristiche appena descritte rendono la RealSense D435i adatta ad essere impiegata in situazioni dove la telecamera subisce movimenti ed oscillazioni continue, come nel caso di una piattaforma robotica umanoide [17].

3.3 LiDAR Livox Mid-360

Il sistema di percezione ambientale e navigazione spaziale del robot umanoide Unitree G1 EDU si affida primariamente al sensore **LiDAR Livox Mid-360** installato nella sua testa [18]. Questo dispositivo consente la mappatura tridimensionale dell'ambiente e la sua implementazione è fondamentale per permettere l'evitamento dinamico degli ostacoli.
La tecnologia LiDAR (*Light Detection and Ranging* o *Laser Imaging Detection and Ranging*) si basa sull'emissione di impulsi laser ad alta frequenza e sulla misurazione del tempo impiegato dalla radiazione riflessa da una superficie per tornare al ricevitore. Questo principio, noto come *Time of Flight* (ToF), permette di calcolare la distanza d della superficie rispetto al sensore tramite la seguente relazione cinematica:

$$d = \frac{c \cdot \Delta t}{2} \tag{3.1}$$

dove c rappresenta la velocità della luce nel mezzo considerato e Δt è l'intervallo di tempo intercorso tra l'emissione dell'impulso laser e la ricezione del segnale di

ritorno. Integrando la misurazione della distanza con le coordinate angolari assunte dal meccanismo di deflessione del raggio al momento dell'emissione, il sistema è in grado di calcolare le coordinate spaziali del punto colpito, generando una fitta nuvola di punti (*point cloud*) tridimensionale dell'ambiente circostante. Nel LiDAR Livox Mid-360 la deflessione del raggio (*beam deflection*) avviene tramite un sistema optomeccanico interno composto da prismi rotanti (*non-repetitive scanning technology*), questa tecnologia di deflessione genera un pattern di scansione non ripetitivo che assume una complessa forma a rosone. Il vantaggio di questa soluzione risiede nel fatto che la densità di campionamento (*point density*) all'interno del campo visivo aumenta progressivamente nel tempo. Mantenendo il sensore in posizione, l'area inquadrata viene coperta in modo quasi esaustivo, minimizzando le zone d'ombra (*blind spots*) che tipicamente affliggono i sensori LiDAR dotati di tecnologie di deflessione più obsolete.
Il Livox Mid-360 è stato selezionato per l'Unitree G1 EDU a causa del suo bilanciamento ottimale tra prestazioni, ingombro e massa ($\approx$ 265 g), garantendo un impatto minimo sulle proprietà inerziali del modulo testa e sul baricentro del robot.
Come si può osservare dalla Figura 3.4, il campo visivo del sensore fornisce una copertura omnidirezionale orizzontale pari a 360° ed un'apertura verticale complessiva di 59°. Nello specifico, il FOV verticale si estende da 7° verso l'alto fino a 52° verso il basso rispetto all'orizzonte del sensore. Questa asimmetria è progettata appositamente per le applicazioni robotiche terrestri, massimizzando il campionamento della morfologia del suolo e degli ostacoli a bassa quota, dati indispensabili per la pianificazione dei passi dell'umanoide [19].

Capitolo 4

Struttura Software

4.1 Ubuntu

L'infrastruttura software su cui si poggia l'intera architettura di percezione visiva, navigazione autonoma (SLAM) ed elaborazione neurale (*Object Detection Live*) è il sistema operativo *open-source* **Ubuntu** nella sua distribuzione **22.04 "Jammy Jellyfish"**. La sua scelta come sistema operativo di riferimento è dettata dal suo ruolo di standard *de-facto* nel settore della robotica. Infatti garantisce la massima compatibilità con l'architettura sia hardware che software del robot Unitree G1 EDU, fornendo contemporaneamente il supporto nativo e la stabilità del middleware ROS2 ed un ambiente fortemente ottimizzato per l'inferenza[1] ad alte prestazioni e bassa latenza della rete neurale.
Nelle applicazioni robotiche, l'interazione con il sistema operativo Ubuntu avviene prevalentemente in modalità *headless* (ovvero senza l'impiego continuo di un'interfaccia grafica) al fine di contenere il carico computazionale su CPU e RAM. Di conseguenza tutte le azioni da eseguire sul PC di sviluppo (come l'esecuzione degli script) avvengono primariamente tramite l'interfaccia a linea di comando (**CLI**) fornita dal Terminale di sistema ed interpretata dalla **Shell Bash** (*Bourne-Again SHell*). La Bash è il programma interprete di comandi predefinito sui sistemi operativi Ubuntu. Nello specifico, la Bash funge da interfaccia e ponte di comunicazione tra l'utente ed il sistema operativo.

Per garantire la trasmissione continua di flussi dati ad alta larghezza di banda tra il PC di sviluppo e l'unità computazionale di bordo del robot Unitree G1 EDU, l'interfaccia Ethernet del robot umanoide è collegata al PC di sviluppo tramite adattatore Ethernet-USB. Si configura la comunicazione mediante indirizzamento `IPv4` statico con indirizzo IP `192.168.123.222` e maschera di rete `255.255.255.0`; in tal modo è garantito il canale di comunicazione diretto con la scheda del robot collocata sull'IP `192.168.123.164`. Inoltre per eliminare gli ostacoli di comunicazione tra robot umanoide ed il PC di sviluppo, è stata prevista la disattivazione delle regole del firewall nativo di Ubuntu `ufw` tramite comando da terminale: `sudo ufw disable`.

[1] Con inferenza si intende la fase operativa nella quale una rete neurale già addestrata applica le conoscenze acquisite durante l'addestramento per fare predizioni in tempo reale su nuovi dati mai visti prima (come accade nell'applicazione reale di *Object Detection Live*).

4.2 ROS 2

L'architettura software di base per lo sviluppo, la distribuzione, l'esecuzione e la manutenzione dell'applicazione robotica si fonda sull'ecosistema *open-source* **ROS** (*Robot Operating System*) [20]. Nonostante la denominazione, ROS non costituisce un sistema operativo tradizionale, bensì un middleware strutturato che fornisce un insieme di librerie, strumenti ed interfacce standardizzate per consentire lo sviluppo modulare e multipiattaforma. ROS è lo standard di riferimento per un'ampia varietà di sistemi robotici quali i robot industriali, i robot collaborativi (*Cobot*), i robot mobili, i droni ed i robot umanoidi avanzati. Un middleware è uno strato software intermediario situato tra il sistema operativo di base (come Ubuntu) e le singole applicazioni ed algoritmi. Il suo scopo fondamentale è fungere da colonna portante di comunicazione distribuita, consentendo a programmi differenti, processi separati o dispositivi fisici diversi (come la sensoristica di bordo del robot umanoide) di scambiarsi dati e coordinarsi. Per il PC di sviluppo ci si è affidati alla distribuzione **ROS2 Humble Hawksbill** nativa per la versione di Ubuntu 22.04 Jammy Jellyfish [21].
L'ecosistema ROS si articola su quattro pilastri fondamentali: **plumbing**, **tools**, **capabilities** e **community**.



Figura 4.1: Ecosistema ROS

Plumbing / Framework (Infrastruttura) fornisce un sistema di passaggio messaggi che funge da vera e propria "tubatura" dell'architettura. La comunicazione rappresenta una delle prime e più critiche necessità quando si implementa un'applicazione robotica distribuita destinata ad interagire con l'hardware. Il framework ROS gestisce lo scambio dati tra nodi (**Nodes**) attraverso un modello anonimo di *Publish/Subscribe* (Pubblicazione/Sottoscrizione). I nodi sono unità computazionali indipendenti responsabili dell'esecuzione di un singolo compito logico. Le interfacce ROS (*Interfaces*) definiscono le modalità con cui i nodi scambiano messaggi, la tipologia di interfaccia utilizzata per le nostre applicazioni è il **Topic**, adatto per flussi continui di dati, come ad esempio un flusso di dati in tempo reale proveniente da un sensore a bordo del robot umanoide. I topic implementano il modello *Publish/Subscribe*, un nodo pubblica i dati ad un topic ed un altro nodo si sottoscrive al topic per ricevere quei dati.

Tools (Strumenti) messi a disposizione dall'ecosistema ROS per la fase di sviluppo, diagnostica e collaudo dell'applicazione robotica. Gli strumenti non in-

troducono nuovi comportamenti al robot ma accelerano la progettazione e la validazione sperimentale.

Capabilities (Funzionalità) costituiscono una vasta raccolta di pacchetti software pronti all'uso che l'ecosistema ROS 2 mette a disposizione. Queste funzionalità offrono i blocchi costruttivi fondamentali che consentono allo sviluppatore di concentrarsi sulla specifica applicazione senza dover riprogettare gli algoritmi di base da zero.

Community di ROS è una rete globale distribuita, composta da studenti, ricercatori accademici, aziende ed enti che mantengono attivo ed in costante evoluzione il progetto *open-source*. Questo costituisce un grande vantaggio in quanto l'esperienza collettiva della community ha portato alla standardizzazione dei formati di messaggio di ROS per l'interazione con qualsiasi periferica (come LiDAR o telecamere RGB-D), rendendo semplice la condivisione del codice, lo scambio di idee e la riproducibilità scientifica dei risultati.

L'utilizzo di ROS 2 ci ha garantito l'interoperabilità tra diversi linguaggi di programmazione quali C++ e Python, ci ha permesso di avviare ed interagire con i nodi driver dei sensori in dotazione al robot Unitree G1 EDU i quali pubblicano su topic dedicati e sfruttando l'implementazione di **CycloneDDS** (*Data Distribution Service*) ci ha ridotto la latenza durante il flusso dati in tempo reale. Per quest'ultima funzione abbiamo configurato la *Quality of Service* QoS (Qualità del Servizio) sul profilo *Best Effort*, questa modalità permette di scartare automaticamente i dati obsoleti (i fotogrammi nel caso del riconoscimento oggetti) in caso di rallentamenti di comunicazione sopprimendo i ritardi accumulati (*lag*) e garantendo un flusso dati realmente in tempo reale.

4.3 Ultralytics YOLO

All'interno dell'architettura software sviluppata per la percezione visiva del robot umanoide Unitree G1 EDU, il compito di riconoscimento visivo è affidato ad **Ultralytics YOLOv11** [22]. Storicamente, i primi sistemi di visione artificiale analizzavano le immagini in più fasi sequenziali: ricercavano dapprima le regioni di interesse per poi classificarne il contenuto. La famiglia di reti neurali convoluzionali CNN (*convolutional neural network*) YOLO (acronimo di *You Only Look Once*) ha rivoluzionato il settore diventando l'attuale standard di riferimento industriale ed accademico per l'operazione di *detection* nell'ambito della *Computer Vision*. Il punto di forza di questa rete è la progettazione volta ad ottimizzare il *trade-off* prestazionale tra l'accuratezza di rilevazione e l'efficienza computazionale. Tale bilanciamento garantisce elevate velocità di inferenza, rendendo la rete idonea all'esecuzione in tempo reale su piattaforme robotiche.

Il principio fondante delle reti neurali YOLO risiede nell'elaborazione del segnale visivo attraverso un approccio unificato, caratterizzato da quattro fasi principali:

- Divisione in griglia: nei primi modelli YOLO l'immagine in input viene geometricamente suddivisa in una griglia spaziale di dimensione $S \times S$ così da localizzare le caratteristiche fondamentali della scena. In YOLOv11 l'architettura non utilizza più un'unica griglia rigida, ma adotta una struttura piramidale multi-scala (*Feature Pyramid Network*) estraendo le caratteristiche dell'immagine a tre livelli di risoluzione denominati *strides* pari a 8, 16 e 32 pixel. Quindi considerando un'immagine di input 640×640 pixel vengono generate tre griglie di dimensioni 80×80 pixel per rilevare oggetti piccoli, 40×40 pixel per rilevare oggetti di medie dimensioni e 20×20 pixel per rilevare oggetti grandi. L'estensione della griglia su più scale spaziali garantisce una rilevazione accurata di oggetti di diverse dimensioni.

- Analisi unica: la rete neurale elabora le matrici d'immagine in un singolo passaggio computazionale (concetto alla base del nome del modello).

- Predizione simultanea: ciascuna cella di ogni griglia computa contemporaneamente le coordinate geometriche dei *bounding box 2D*, lo score di confidenza probabilistica (un indice di certezza della rilevazione) e la classe categoriale assegnata all'oggetto.

- Filtraggio delle predizioni: nella fase di post-elaborazione viene impiegata la tecnica di *Non-Maximum Suppression* per eliminare i riquadri di rilevamento sovrapposti, mantenendo unicamente le predizioni dotate del punteggio di affidabilità massimo.

L'utilizzo dei modelli YOLO in questo documento risiede nella loro implementazione per l'*Object Detection Live* (in tempo reale). A livello formale, tale operazione di intelligenza artificiale unisce la localizzazione geometrica e spaziale di un elemento all'interno di un'immagine alla sua caratterizzazione semantica. Con caratterizzazione semantica si intende il processo per cui la rete neurale associa all'oggetto rilevato un significato concettuale, una categoria o una classe.
L'output primario di tale algoritmo è il ***bounding box 2D***: un riquadro rettangolare bidimensionale definito dalle coordinate normalizzate (x, y, w, h) che circoscrive i contorni dell'oggetto identificato. Definendo W ed H come rispettivamente larghezza ed altezza dell'immagine in pixel possiamo calcolare le coordinate normalizzate:

x - coordinata orizzontale normalizzata del centro della bounding box

$$x = \frac{u_{\text{center}}}{W} \tag{4.1}$$

dove u_{center} è la coordinata orizzontale in pixel del centro della bounding box.

y - coordinata verticale normalizzata del centro della bounding box

$$y = \frac{v_{\text{center}}}{H} \tag{4.2}$$

dove v_{center} è la coordinata verticale in pixel del centro della bounding box.

w - larghezza normalizzata della bounding box

$$w = \frac{W_{\text{box}}}{W} \tag{4.3}$$

dove W_{box} è la larghezza in pixel della bounding box.

h - altezza normalizzata della bounding box

$$h = \frac{H_{\text{box}}}{H} \tag{4.4}$$

dove H_{box} è l'altezza in pixel della bounding box.

A ciascun bounding box viene univocamente associata un'etichetta testuale ed uno score di confidenza probabilistica, con funzione di filtro contro i falsi positivi.
YOLO si configura come una piattaforma di elaborazione visiva capace di svolgere un'ampia gamma di attività di visione artificiale. Oltre all'*Object Detection* bidimensionale [23], l'architettura supporta nativamente task di segmentazione delle istanze (*Instance Segmentation*) [24], stima della posa (*Pose Estimation*) [25], rilevamento con riquadri orientati OBB (*Oriented Bounding Boxes*) [26] e classificazione globale delle immagini (*Image Classification*) [27]. Per via della sua efficienza, la rete è ampiamente impiegata in contesti *live* come l'automazione industriale e la robotica autonoma.
La disponibilità di differenti modelli per ogni versione di YOLO, come le varianti YOLOv11 Nano e YOLOv11 Small, permette di scalare il modello in base alle risorse hardware disponibili.

4.4 Roboflow

Roboflow [28] è una piattaforma *cloud-based* progettata per standardizzare e velocizzare il ciclo di preparazione dei dati (*Computer Vision Data Pipeline*).
In ambito informatico e nell'apprendimento automatico (*Machine Learning/Deep Learning*), un **dataset** (o insieme di dati) è una collezione strutturata, organizzata e formalmente definita di dati, informazioni o campioni omogenei raccolti per poi essere analizzati, elaborati ed impiegati nell'addestramento di modelli predittivi. Nel contesto specifico della *Computer Vision* e dell'addestramento di reti neurali per il riconoscimento oggetti, come le architetture YOLO, il dataset è costituito dall'insieme di immagini affiancate dalle *bounding box* e dalle etichette relative alle classi di oggetti target.

La piattaforma Roboflow è stata scelta in quanto accentra in un unico ambiente software tutte quelle funzioni che rendono il processo di creazione ed implementazione del dataset facilitato e guidato. Partendo dall'acquisizione delle immagini grezze ed il successivo *upload* su Roboflow, la piattaforma è stata impiegata nell'ottica dell'*Object Detection Live* per le seguenti funzioni:

- Definizione delle classi di oggetti target.
- Annotazione o *Labelling*, con la funzione di annotazione semi-automatica *Smart Select* la definizione delle bounding box degli oggetti target in ogni immagine viene resa nettamente più rapida.
- Partizionamento delle immagini nei tre sottoinsiemi nei quali viene diviso il dataset: *Training Set*, *Validation Set*, *Test Set*.
- Applicazione di trasformazioni deterministiche alle immagini grezze (*Pre-processing*).
- Generazione di nuove varianti sintetiche di addestramento sulla base delle immagini grezze tramite la funzione di *Data Augmentation*.
- Esportazione nel formato nativo per l'utilizzo con Ultralytics YOLO e generazione del file di configurazione del dataset `data.yaml` da fornire in input alla rete neurale YOLO scelta per avviare il processo di addestramento.

Risulta importante sottolineare come uno dei punti di forza di Roboflow è la *community open-source* **Roboflow Universe** [29] che rende la piattaforma un hub di condivisione, apprendimento e sviluppo collaborativo tra studenti, ricercatori ed aziende. Roboflow Universe contiene oltre un milione di dataset ed oltre un miliardo di immagini annotate create da utenti in tutto il mondo. Questi dataset pubblici possono essere scaricati o uniti ad un dataset proprio per poi essere impiegati nell'ambito della *Computer Vision*.
Nella nostra applicazione di *Object Detection Live* si è scelto di creare un dataset da zero, cioè senza affidarci ad immagini o dataset già esistenti nella community di Roboflow Universe.

4.5 Unitree SDK

La gestione cinematica e dinamica del robot umanoide Unitree G1 EDU si fonda su un'infrastruttura software di sviluppo integrata messa a disposizione dal costruttore, denominata **Unitree SDK** (*Software Development Kit*). A differenza dei sistemi robotici industriali ad architettura "chiusa", nei quali i comportamenti motori sono preprogrammati in fabbrica e non modificabili dall'utente, la configurazione EDU mette a disposizione gli SDK per lo sviluppo secondario da parte degli utenti.
L'intera architettura software degli SDK di Unitree è mantenuta e distribuita pubblicamente come codice *open-source* tramite i *repository* ufficiali dell'azienda sulla

piattaforma **GitHub** [30]. Questa apertura consente a ricercatori e sviluppatori di ispezionare il codice sorgente ed integrare direttamente gli algoritmi di controllo all'interno di nodi ROS 2. L'ecosistema software di Unitree su GitHub si articola in diversi pacchetti scritti in C++, Python e Rust, progettati per garantire l'interoperabilità tra la simulazione ed il robot reale, di particolare rilevanza sono:

- `unitree_sdk2` rappresenta la libreria nativa in linguaggio C++ per il controllo in tempo reale dei robot di nuova generazione di Unitree (incluso il robot umanoide G1). Gestisce la serializzazione, a frequenze pari o superiori ad 1 kHz, dei pacchetti dati e la comunicazione tramite protocollo CycloneDDS con le schede elettroniche di bordo del robot.

- `unitree_sdk2_python` costituisce l'interfaccia di alto livello in linguaggio Python per unitree_sdk2. Tale modulo è fondamentale per le applicazioni di *Computer Vision*.

Inoltre tra i *repository* ufficiali di Unitree su GitHub è possibile trovare, per ogni modello di robot da loro prodotto, i file di descrizione cinematica, dinamica e strutturale **URDF** (*Unified Robot Description Format*). I file URDF descrivono le proprietà geometriche ed inerziali (quali masse e matrici d'inerzia) e permettono l'integrazione del robot all'interno dell'ambiente di visualizzazione grafica **RViz2** e l'esecuzione di simulazioni dinamiche in motori di simulazione fisica come **MuJoCo**.
Per gestire la complessità strutturale del robot umanoide (caratterizzato da un numero massimo di DoF pari a 43), l'SDK di Unitree suddivide la gestione cinematica in due livelli di controllo gerarchici e disaccoppiati: **Alto Livello** (*High-Level Control*) e **Basso Livello** (*Low-Level Control*).
Il controllo ad alto livello si occupa della gestione macroscopica, invece la dinamica dei singoli motori ed il mantenimento dell'equilibrio bipede vengono gestiti in modo del tutto autonomo dagli algoritmi integrati di bordo. L'utente non deve calcolare il movimento di ogni singola articolazione né preoccuparsi di evitare la caduta del robot. Il robot viene pilotato impartendo comandi semantici globali, come la direzione di marcia o la velocità lineare di avanzamento. Il computer di bordo riceve questi comandi ed elabora automaticamente la coordinazione degli arti, risolvendo in tempo reale la cinematica inversa ed il calcolo del centro di massa del robot così da regolarne la stabilità dinamica. In presenza di urti o terreni sconnessi, il controllore ad alto livello impartisce autonomamente i comandi ai servomotori per mantenere l'equilibrio.
Il controllo a basso livello è concepito per le operazioni in cui è richiesto un dominio assoluto e diretto sulla meccanica del robot. L'utente può assumere il controllo individuale e simultaneo di ciascun singolo servomotore articolare, invece di inviare velocità di marcia globali, gli script eseguiti nel PC di sviluppo inviano ad ogni singolo giunto i riferimenti di posizione angolare, la velocità di rotazione desiderata, la coppia che ogni motore deve esercitare ed i guadagni di rigidezza e smorzamento.

Questi ultimi due parametri risultano particolarmente interessanti in quanto la loro accurata regolazione conferisce la cedevolezza meccanica desiderata agli arti, assorbendo gli urti accidentali preservando l'integrità dei componenti interni e garantendo un'interazione fluida e sicura durante la manipolazione di precisione.

Capitolo 5

Object Detection Live

Il presente capitolo tratta la prima delle due integrazioni sensoriali sviluppate, descrivendo dettagliatamente la costruzione del dataset, il processo di addestramento, l'interpretazione dei risultati e la successiva validazione sperimentale dell'architettura di *Object Detection Live* (**Riconoscimento Oggetti in tempo reale**) basata sulla rete neurale **Ultralytics YOLO11**.

Per garantire elevati standard di affidabilità, robustezza visiva ed assenza di errori di classificazione nell'applicazione robotica reale, lo sviluppo del modello non si è limitato ad una singola sessione di addestramento su un dataset statico, ma si è intrapreso l'approccio del *Batch Learning*. Si è partiti dalla prima versione del dataset e dopo il primo addestramento è stato testato il modello identificando le difficoltà ed i limiti nell'applicazione reale per poi adottare misure di miglioramento quali arricchire maggiormente il dataset e scegliere diversi parametri o modelli di addestramento. Successivamente all'implementazione di migliorie si è addestrato il modello da zero combinando sia i dati iniziali che i nuovi dati aggiunti. Si è arrivati così ad avere tre versioni del dataset e del modello addestrato denominate **train-1**, **train-2**, **train-3**.

5.1 Dataset

Per la fase di creazione del dataset ci si è appoggiati alla piattaforma web-based Roboflow che funge da ponte tra l'acquisizione delle immagini e la generazione del dataset da fornire in input a YOLO per l'addestramento. Indispensabile per avere buoni risultati di addestramento è creare un dataset correttamente strutturato e vario. Sono stati scelti **oggetti target** di diversa forma, dimensione e colore e si è abbinato ad ogni oggetto target la corrispettiva `class` (classe):

- Pallone verde denominato `ball`;
- Forbici dal manico rosso denominate `forbici`;
- Tre mattoncini di colore rosso, verde e blu denominati rispettivamente `R-brick`, `G-brick` e `B-brick`;

- Quattro chiavi Brugola di dimensioni crescenti (2.5mm, 4mm, 6mm, 10mm) denominate rispettivamente `brugola2.5`, `brugola4`, `brugola6`, `brugola10`.

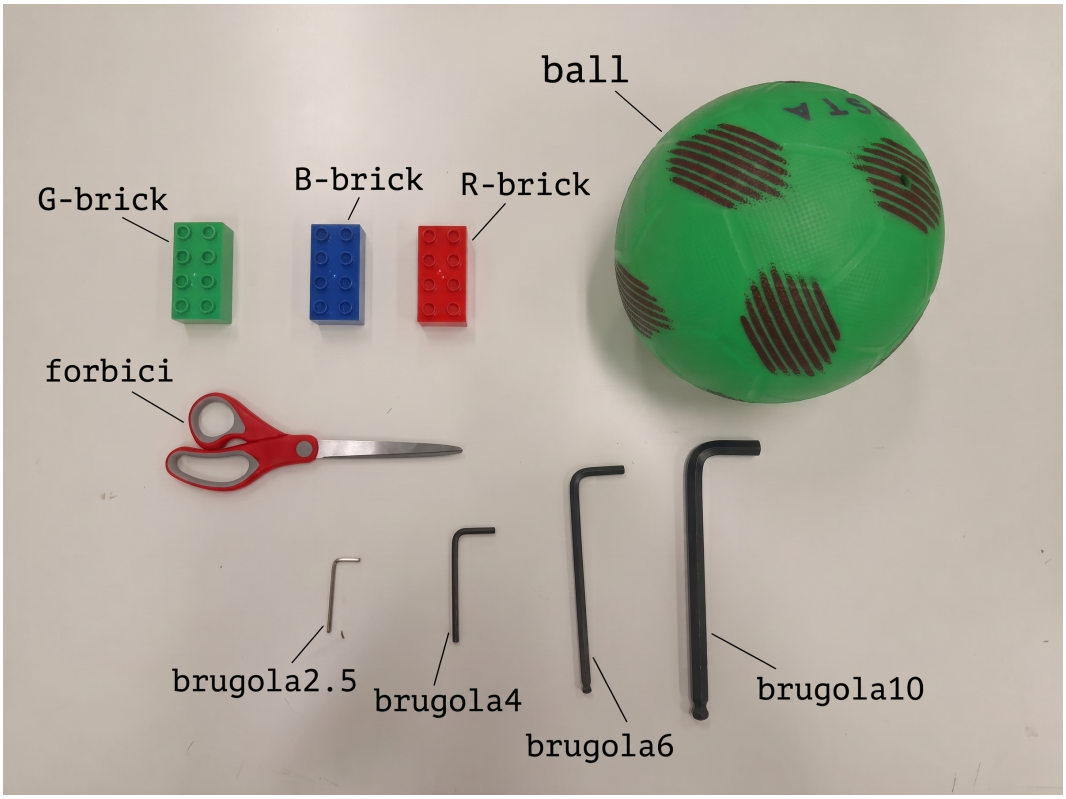


Figura 5.1: Oggetti target dell'*Object Detection Live*.

Le immagini per il dataset sono state scattate nei locali e nei corridoi del DIISM del Polo Montedago dell'Università Politecnica delle Marche in condizioni variabili di luce, di angolazione, di distanza dall'oggetto e di sfondo. Successivamente sono state caricate le immagini scattate sulla piattaforma Roboflow per effettuare la fase di **annotazione** (*labelling*), per ogni immagine sono state create sia manualmente sia tramite la funzione di annotazione semi-automatica *Smart Select* le bounding box degli oggetti target presenti in esse abbinando la rispettiva classe. Sono state scattate ed inserite nel dataset anche numerose immagini di solo sfondo, comunemente dette `"null"` (cioè senza la presenza di oggetti target), dove non è stata ovviamente definita nessuna bounding box. Si è arrivati ad avere per il train-3 un numero pari a 891 immagini nel dataset (indicandolo come $\mathcal{D}_{\text{totale}}$). Un dataset di immagini non può essere somministrato direttamente per intero ad un modello Deep Learning come YOLO, il modello imparerebbe "a memoria" le immagini senza generalizzare incorrendo nel fenomeno dell'*overfitting*. Si preferisce partizionare il dataset tramite le funzioni di Roboflow, il dataset totale viene così suddiviso in tre sottoinsiemi disgiunti:

$$\begin{cases} \mathcal{D}_{\text{totale}} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{test}} \\ \mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{val}} \cap \mathcal{D}_{\text{test}} = \emptyset \end{cases} \tag{5.1}$$

II Training/Train Set ($\mathcal{D}_{\text{train}}$): costituisce la porzione effettiva di apprendimento su cui il modello impara a generalizzare gli oggetti target. Rappresenta il 70% $\sim$ 80% del dataset totale.

II Validation/Valid Set ($\mathcal{D}_{\text{val}}$): viene applicato dopo l'apprendimento effettuato con il Train Set per stimare le prestazioni del modello e guidare la selezione del checkpoint ottimale. Costituisce il 10% $\sim$ 15% del dataset totale.

II Test Set ($\mathcal{D}_{\text{test}}$): rimane completamente isolato dal modello fino alla fine dell'addestramento, a questo punto si possono testare le sue prestazioni mettendolo alla prova su questa porzione di dataset che il modello non ha mai elaborato. Corrisponde al 5% $\sim$ 10% del dataset totale.

Con ***Pre-Processing*** (pre-elaborazione dei dati) si intende quell'insieme di trasformazioni da applicare all'insieme di dati grezzi prima che essi vengano somministrati alla rete neurale, risulta un passaggio necessario per rendere il dataset omogeneo, standardizzato e compatibile con l'architettura della rete neurale. Per il *Pre-Processing* del dataset di immagini ci si è affidati alle funzioni incorporate in Roboflow. Si è applicata la funzione di *Resize: stretch to* 640×640 *pixel* ridimensionando tutte le immagini alla risoluzione quadrata 640×640 nativa di tutti i modelli YOLO permettendo un riconoscimento di oggetti di medie e piccole dimensioni pur mantenendo una latenza bassa durante l'*Object Detection Live*. Per il train-2 e train-3 è stata inoltre applicata la tecnica della ***Data Augmentation***, anch'essa funzione presente in Roboflow [31]. La *Data Augmentation* è una tecnica di pre-elaborazione delle immagini del Train Set che consiste nell'applicare lievi trasformazioni geometriche, cromatiche e fotometriche alle immagini reali così da ottenerne delle varianti. Serve a simulare le condizioni reali con cui interagisce la telecamera durante l'applicazione reale del riconoscimento oggetti e per ampliare il dataset senza scattare nuove immagini.

A questo punto possiamo ottenere tramite download diretto dal *terminal* di Ubuntu il file `data.yaml` (*Dataset Configuration File*), cioè il file in formato YAML che funge da "mappa" del dataset definendo la sua struttura, le sue classi (con le relative etichette/*labels*) ed i percorsi relativi ai Set di immagini (Train, Valid, Test). `data.yaml` è il file di input da fornire a YOLO per avviare l'addestramento.

5.2 Processo di Addestramento

Disponendo ora del file necessario per l'addestramento `data.yaml` è possibile dare il comando da terminale di Ubuntu tramite `Bash` andando a specificare i parametri

Parametro	**Valore**
Augmentation factor	3
Flip	Horizontal
Crop	0% Minimum Zoom, 20% Maximum Zoom
Rotation	Between -15° and +15°
Hue	Between -10° and +10°
Brightness	Between -15% and +15%
Exposure	Between -10% and +10%
Blur	Up to 1.5px
Noise	Up to 1.05% of pixels

Tabella 5.1: Parametri di Augmentation applicati ai $\mathcal{D}_{\text{train}}$ di train-2 e train-3.

di addestramento. Considerando il comando di avvio addestramento del train-1:

```
yolo detect train \
model=yolo11s.pt \
data=/home/filippo/Desktop/training_0.yolov11/data.yaml \
epochs=100 \
imgsz=640 \
batch=4 \
device=cpu \
patience=20
```

- `yolo` richiama lo strumento della libreria Ultralytics, il sotto-comando `detect` indica l'obiettivo finale dell'addestramento (quindi l'Object Detection), `train` dichiara la procedura di avvio addestramento;

- la seconda riga è dedicata alla scelta del modello YOLO di partenza da addestrare. Per il train-1 e train-2 è stato scelto il modello **YOLOv11s** (*Small*), invece per il train-3 si è optato per **YOLOv11n** (*Nano*). YOLOv11s è vantaggioso in termini di accuratezza, essendo dotato di 9.4 milioni di parametri è ottimo per il riconoscimento di oggetti di piccole dimensioni e simili tra loro, però a causa dell'elevato carico computazionale i tempi di addestramento risultano dilatati. YOLOv11n avendo 2.6 milioni di parametri permette di avere tempi di addestramento non eccessivi, avendo però il compromesso di un'accuratezza leggermente inferiore. Il vantaggio fondamentale del modello *Nano* rispetto a quello *Small* è la massima velocità di riconoscimento, usando un modello *Nano* si può garantire la minima latenza durante l'*Object Detection Live*;

- la terza riga indica il percorso assoluto del file di configurazione in formato YAML;

- alla quarta riga si definisce il numero di epoche (`epochs`) di addestramento, un'epoca corrisponde al ciclo completo durante il quale l'intero Training Set viene presentato alla rete neurale ed elaborato esattamente una volta. Al termine di ogni epoca viene presentato il Valid Set alla rete neurale per valutarne le prestazioni e calcolarne le metriche di interesse;

- alla quinta riga si definisce la `image size`, scelta pari a 640×640 pixel in quanto è la risoluzione scelta per il *Resize* nel pre-processing del dataset e la risoluzione nativa di YOLOv11;

- alla sesta riga viene definita la dimensione del `batch` (*Batch Size*), ci indica quante immagini del Train Set vengono elaborate in parallelo. La sua scelta deriva dal compromesso di avere tempi computazionali ragionevoli ed al contempo evitare il surriscaldamento e la saturazione della RAM della risorsa hardware utilizzata per l'addestramento (il PC di sviluppo). `batch=4` è un numero contenuto ed adatto per macchine dotate di risorse contenute;

- alla settima riga viene specificata la risorsa hardware assegnata ai calcoli di addestramento, nel presente caso la CPU. Da notare che avere a disposizione una GPU dedicata permetterebbe di avere tempi di calcolo ridotti;

- all'ottava riga si definisce il parametro dedicato all'*Early Stopping*, ponendo `patience=20` quando definiamo i parametri di addestramento stiamo comandando di fermare e concludere l'addestramento dopo 20 epoche che il file dei pesi `best.pt` (generato come output dall'addestramento) non viene sovrascritto, così da evitare tempo computazionale che non comporterebbe miglioramenti prestazionali della rete neurale.

I tempi di addestramento sono dipendenti dalle risorse hardware disponibili, dalla dimensione del dataset utilizzato, dal modello di partenza scelto e dai parametri di addestramento quali numero di epoche scelto e dimensione dei `batch`. Nel nostro caso sono stati riscontrati i seguenti tempi computazionali:

Addestramento	**Modello YOLO**	**Tempo (ore)**
train-1	YOLOv11s	16
train-2	YOLOv11s	58
train-3	YOLOv11n	28

Tabella 5.2: Tempi computazionali di addestramento.

È possibile osservare il notevole incremento di tempo tra train-1 e train-2, imputabile all'aggiunta di numerose immagini e all'utilizzo della *Data Augmentation* per il train-2 (da ricordare che l'*Augmentation factor* è un moltiplicatore delle immagini del Train Set). Il tempo computazionale del train-3 è risultato nettamente inferiore

rispetto a quello del train-2 a causa della diversa scelta del modello YOLO di partenza, infatti il modello YOLOv11n ha permesso di avere tempi di calcolo ridotti nonostante i dataset di train-2 e train-3 fossero quasi identici ed i parametri di *Data Augmentation* fossero gli stessi.

Al termine delle epoche di addestramento la libreria Ultralytics YOLO genera nella directory di output, oltre ai file fondamentali per l'impiego dell'addestramento nell'applicazione reale, i risultati e le metriche dello stesso analizzabili tramite un set di grafici [32].

5.3 Risultati di Addestramento e loro Valutazione

La valutazione quantitativa dell'architettura di rilevamento oggetti in tempo reale (come la famiglia Ultralytics YOLO) si basa su un insieme di metriche standard per la misura della precisione di localizzazione spaziale delle bounding box e di correttezza di classificazione.

5.3.1 Metriche di Valutazione

La valutazione quantitativa dell'architettura per il rilevamento di oggetti in tempo reale si basa su un insieme di metriche standard formalizzate nell'ambito della Computer Vision.

L'indice IoU (*Intersection over Union*) quantifica il grado di sovrapposizione geometrica sul piano dell'immagine tra il *bounding box* predetto dalla rete neurale (Box_{pred}) e la relativa annotazione reale o *Ground Truth* (Box_{gt}):

$$IoU = \frac{\text{Area}(\text{Box}_{\text{pred}} \cap \text{Box}_{\text{gt}})}{\text{Area}(\text{Box}_{\text{pred}} \cup \text{Box}_{\text{gt}})} \tag{5.2}$$

L' IoU assume valori continui nell'intervallo $[0, 1]$, dove 0 indica l'assenza di sovrapposizione e 1 indica una perfetta coincidenza dei contorni.

Fissata una soglia di sovrapposizione geometrica α (convenzionalmente $\alpha = 0.50$) ed una soglia di confidenza probabilistica (*confidence*) $\tau \in [0, 1]$, ogni predizione viene classificata in:

- **Vero Positivo (TP):** la rete predice la classe corretta e la sovrapposizione rispetta la condizione $IoU \geq \alpha$.
- **Falso Positivo (FP):** la rete rileva un oggetto in corrispondenza dello sfondo, assegna una classe errata o genera un riquadro con $IoU < \alpha$.
- **Falso Negativo (FN):** un oggetto reale presente nell'immagine non viene individuato dal modello.

A partire dal conteggio cumulativo di TP, FP e FN, si definiscono le metriche puntuali in funzione della soglia di confidenza τ:

$$\text{Precision}(\tau) = \frac{TP(\tau)}{TP(\tau) + FP(\tau)} \tag{5.3}$$

$$\text{Recall}(\tau) = \frac{TP(\tau)}{TP(\tau) + FN(\tau)} \tag{5.4}$$

$$\text{F1-Score}(\tau) = 2 \cdot \frac{\text{Precision}(\tau) \cdot \text{Recall}(\tau)}{\text{Precision}(\tau) + \text{Recall}(\tau)} \tag{5.5}$$

- La **Precision** (precisione/accuratezza delle rilevazioni) misura la percentuale di rilevazioni corrette rispetto al totale delle predizioni effettuate dal modello, rappresenta la capacità del modello di evitare i falsi positivi.
- La **Recall** (sensibilità) misura la percentuale di oggetti reali che la rete è riuscita ad individuare rispetto al totale degli oggetti effettivamente presenti nella scena, rappresenta la sensibilità e completezza del modello. Una Recall alta assicura che il modello non ignori oggetti target effettivamente presenti.
- L'**F1-Score** (media armonica bilanciata) è la media armonica tra Precision e Recall, calcolata per fornire un indicatore sintetico unico del bilanciamento del modello. Si è scelta la media armonica invece di quella aritmetica in quanto penalizza fortemente i casi in cui una delle due metriche è estremamente bassa.

L'***Average Precision*** (AP) misura l'area sottesa alla curva Precision-Recall per una specifica classe i al variare della soglia di confidenza $\tau \in [0, 1]$. La ***Mean Average Precision*** (mAP) misura principalmente la capacità del modello di trovare ed identificare l'oggetto target nell'immagine, estende il calcolo estraendo la media aritmetica dell'AP su tutte le N classi del dataset:

$$\text{mAP@0.5} = \frac{1}{N} \sum_{i=1}^{N} AP_i \quad \text{con } IoU \geq 0.50 \tag{5.6}$$

La metrica estesa mAP@0.5:0.95 (o mAP50-95) indica la capacità del modello di circoscrivere con estrema precisione i bordi dell'oggetto target, rappresenta la media aritmetica delle mAP calcolate su 10 soglie di IoU crescenti (da 0.50 a 0.95 con passo 0.05):

$$\text{mAP@0.5:0.95} = \frac{1}{10} \sum_{k=0}^{9} \text{mAP@}(0.50 + 0.05k) \tag{5.7}$$

La ***confidence*** τ (o *score* di confidenza probabilistica) indica la misura di certezza stimata della rete neurale su di un rilevamento. Nel modello YOLO, per ogni bounding box 2D ipotizzato sul piano dell'immagine, la *confidence* è definita come il prodotto tra due probabilità distinte:

$$\text{Confidence} = P(\text{Object}) \times P(\text{Class}_i \mid \text{Object}) \tag{5.8}$$

$P($**Object**$)$ **(Objectness Score):** la probabilità stimata che all'interno di quel riquadro sia effettivamente presente un oggetto target (e non lo sfondo vuoto).

$P($**Class**$_i$ | **Object**$)$ **(Class Probability):** la probabilità condizionata che l'oggetto rilevato appartenga nello specifico alla classe i-esima.

La soglia di confidenza viene impostata dall'utente nell'ambiente di esecuzione tramite il parametro `conf` ed agisce come un filtro di severità delle predizioni del modello. Una confidence elevata massimizza la Precision ed elimina i Falsi Positivi, ma rischia di ridurre eccessivamente la Recall, ignorando oggetti target presenti che risultano parzialmente occlusi o in cattive condizioni di illuminazione. Una *confidence* bassa permette al modello di accettare anche ipotesi dubbie aumentando la Recall (quindi rilevando tutti gli oggetti target presenti), ma riducendo la Precision, generando diversi Falsi Positivi e rilevazioni errate.

5.3.2 Descrizione dei Grafici di Output e loro Confronto

Al termine del ciclo di apprendimento e validazione, il framework genera i due file necessari per l'applicazione reale (`best.pt` e `last.pt`) ed un insieme di report grafici e matriciali salvati automaticamente nella directory di output. Questi ultimi tramite la loro lettura ed interpretazione ci permettono di valutare, a priori, le prestazioni del modello addestrato, cioè senza testarlo sperimentalmente.

`best.pt` (*Best Weights File*) è il file in formato *PyTorch* contenente i pesi ottimizzati della rete neurale che hanno registrato le migliori prestazioni in assoluto durante l'addestramento. Il framework ad ogni epoca di addestramento valuta il modello sul Validation Set e sovrascrive il file `best.pt` con le prestazioni dell'epoca corrente solo quando si ottiene un miglioramento assoluto della metrica chiamata *Fitness* calcolata considerando sia mAP@0.5 che mAP@0.5:0.95:

$$\text{Fitness} = 0.1 \times \text{mAP@0.5} + 0.9 \times \text{mAP@0.5:0.95} \tag{5.9}$$

Attribuendo il 90% del peso a mAP@0.5:0.95 viene privilegiata nettamente la precisione geometrica e l'accuratezza nella localizzazione delle *bounding box*.

`best.pt` è il file di maggiore importanza ottenuto dall'addestramento in quanto è il file da utilizzare nell'applicazione reale di riconoscimento oggetti, ci garantisce la massima accuratezza e la minore percentuale di errore nelle predizioni sul campo in quanto rappresenta il modello ottimale con i pesi che hanno registrato il massimo valore di *Fitness*.

`last.pt` (*Last Checkpoint File*) è il file in formato *PyTorch* contenente i pesi corrispondenti all'ultima epoca di addestramento. Viene usato come punto di ripristino (*checkpoint*) per riprendere l'addestramento nel caso di una sua interruzione accidentale o volontaria.

Spesso le prestazioni di `last.pt` risultano inferiori rispetto a quelle di `best.pt` in quanto la rete neurale dopo numerose epoche potrebbe subire decadimento prestazionale riconducibile all'***overfitting*** [33], cioè il fenomeno per cui la rete neurale comincia ad imparare "a memoria" le immagini del Train Set senza generalizzare, ottenendo durante la fase di validazione e la corrispondente interazione col Valid Set risultati peggiori di quelli previsti. Il fenomeno è causato dall'uso di un dataset troppo piccolo e con poca varietà, dall'uso di un modello YOLO troppo complesso rispetto al dataset utilizzato e da un numero eccessivo di epoche di addestramento. Preveniamo l'*overfitting* includendo nel dataset numerose immagini di sfondo e prive di oggetti target `"null"`, applicando il *Data Augmentation* al dataset ed utilizzando il parametro `patience` dedicato all'*Early Stopping*, definibile nel comando di avvio addestramento.

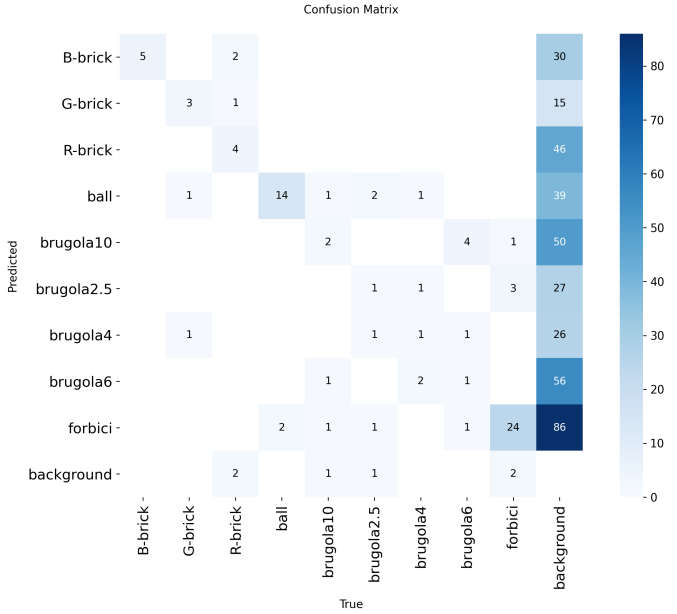
`Confusion Matrix, Normalized Confusion Matrix` (Matrice di Confusione e Matrice Normalizzata di Confusione) viene elaborata sul Validation Set al termine dell'addestramento applicando i pesi ottimali `best.pt`. Il sistema esegue l'inferenza impostando una *confidence* di default pari a 0.25 ed una IoU ≥ 0.50. Sulle righe troviamo le istanze reali (*True*) e sulle colonne le predizioni generate dal modello (*Predicted*) andando a formare una matrice di dimensione $(N+1) \times (N+1)$ dove N è il numero di classi, la colonna aggiuntiva è dedicata allo sfondo (*background*). La versione normalizzata della Matrice di Confusione converte i valori assoluti in tassi percentuali compresi nell'intervallo da 0.0 a 1.0, dividendo ogni cella per la somma totale degli elementi reali della corrispettiva riga rendendola più facilmente interpretabile.

La diagonale principale (Veri Positivi TP) misura il tasso di correttezza della classificazione. Oggetti visivamente e geometricamente caratteristici registrano prestazioni ottimali (cioè vicini ad 1.0), mentre la confusione tra le classi di oggetti target si trova rappresentata dagli elementi fuori diagonale.

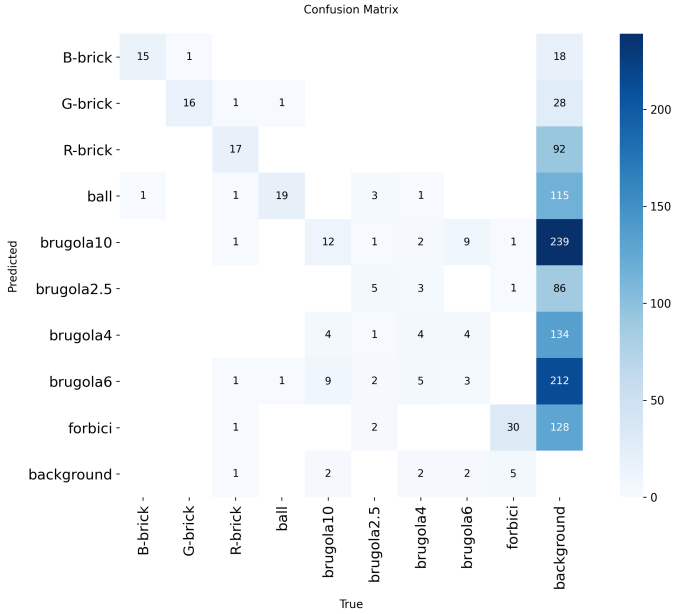
Dall'osservazione delle tre matrici normalizzate di Figura 5.3 possiamo renderci conto dell'evoluzione nelle prestazioni dei tre modelli addestrati ancor prima delle prove sperimentali. L'arricchimento del dataset del `train-2` rispetto a quello del `train-1` è dimostrato da una crescita generale dei valori della diagonale principale e dalla diminuzione della confusione inter-classe tra le tre classi di mattoncini.

Nonostante le misure di miglioramento applicate al `train-2`, isolando la sottomatrice 4×4 relativa alle quattro classi di chiavi Brugola, la loro confusione inter-classe rimane elevata. Questo è ovviamente dovuto all'elevata somiglianza tra i quattro oggetti target. Passando alla matrice normalizzata del `train-3` è immediato osservare come i valori sulla diagonale siano mediamente inferiori rispetto a quelli del `train-2`, giustificabile dall'utilizzo del modello *Nano* rispetto al modello *Small* dei primi due addestramenti. Avendo accorpato le quattro classi di chiavi Brugola in un'unica classe denominata `brugola` si è

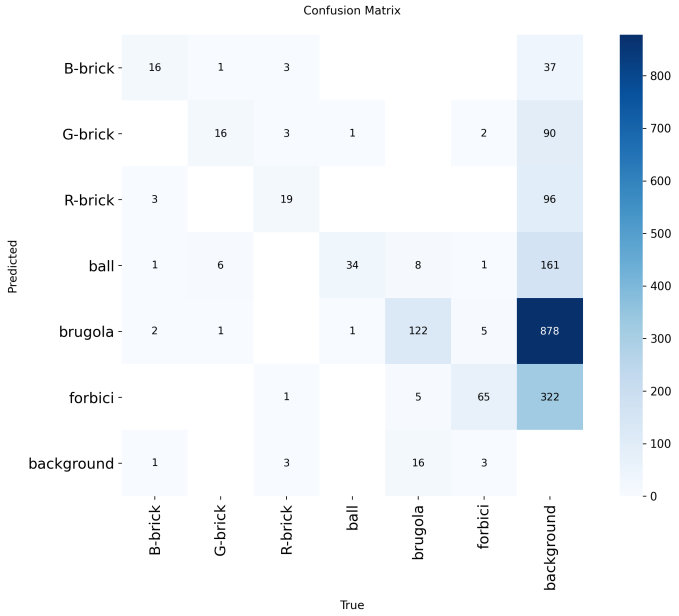
avuto un effetto positivo sul loro riconoscimento registrando un corrispettivo valore elevato sulla diagonale. In prima lettura ci aspettiamo un riconoscimento di Falsi Positivi sul `background` associati alla classe `brugola`, ma c'è da ricordare che le matrici di confusione sono generate ponendo `conf` $= 0.25$. Riprendendo la Figura 5.7 si evidenzia come ponendo nell'applicazione reale una *confidence* maggiore, nell'intorno del suo valore ottimale, si ottiene la rimozione della maggior parte dei Falsi Positivi.
Ponendoci sulle classi `ball` e `forbici` osserviamo che nelle matrici normalizzate di tutti e tre gli addestramenti le due classi ottengono ottimi risultati, troviamo il motivo di questo nelle loro proprietà geometriche e cromatiche distintive e singolari che ne facilitano il riconoscimento anche in situazioni di luce e sfondo sfavorevoli.
Nella matrice normalizzata del `train-1` il valore 1.00 raggiunto dalla classe `B-brick` sembrerebbe garantire un riconoscimento quasi perfetto senza un'interpretazione approfondita. Infatti osservando il `train-2` ed il `train-3` notiamo come la stessa classe raggiunga ottimi punteggi seppur minori del valore ideale 1.00, nonostante il dataset sia nettamente più ampio e vario negli ultimi due addestramenti. La conclusione è ovvia, nel `train-1` l'utilizzo del modello *Small* su un dataset piccolo e poco vario ha portato la classe `B-brick` a soffrire di *overfitting*.



(a) `train-1`

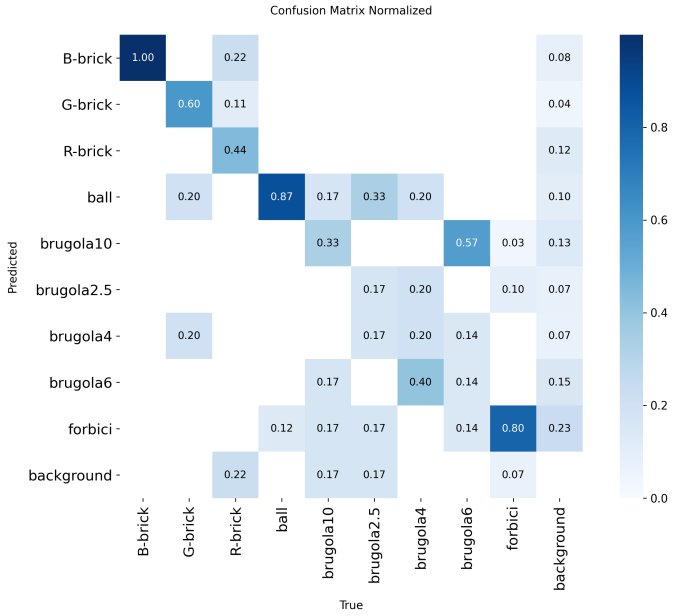


(b) `train-2`

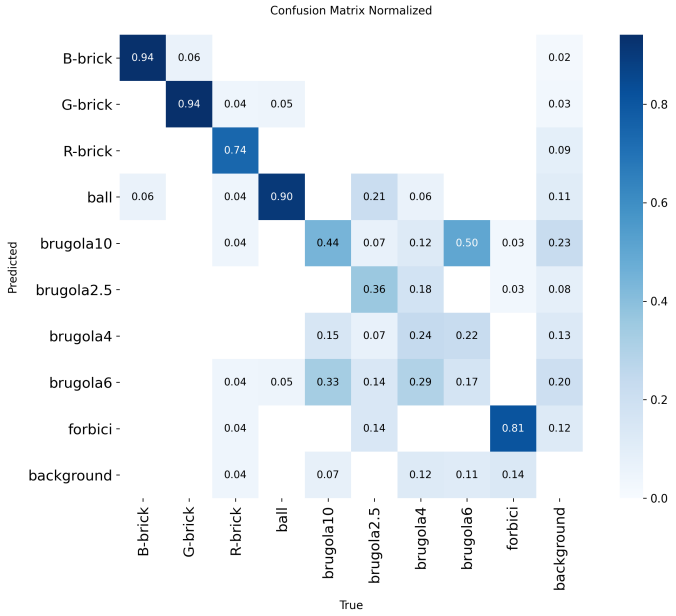


(c) `train-3`

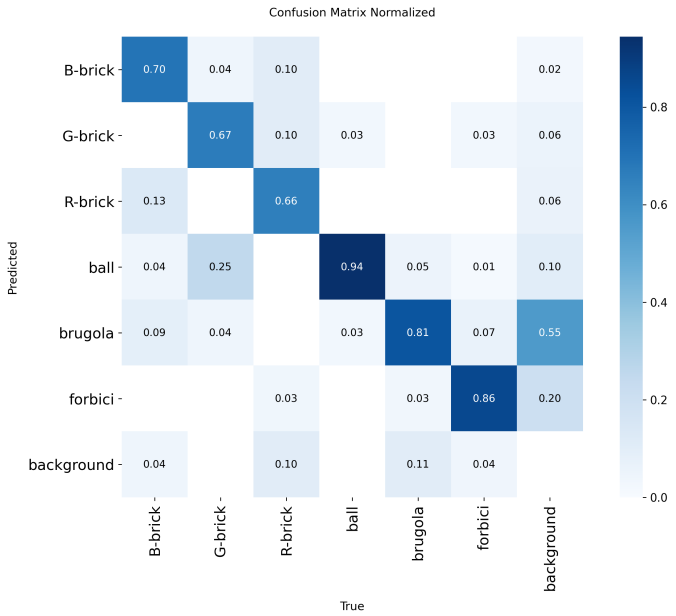
Figura 5.2: Matrici di Confusione relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).



(a) `train-1` (Normalizzata)



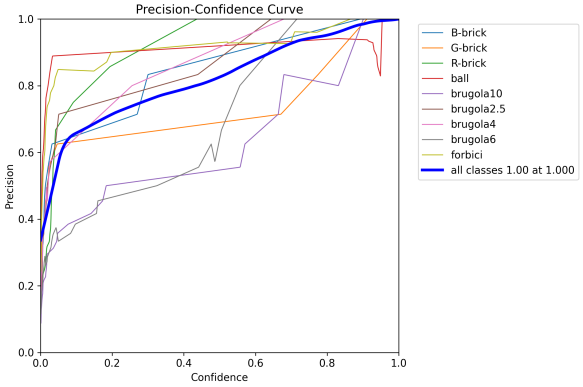
(b) `train-2` (Normalizzata)



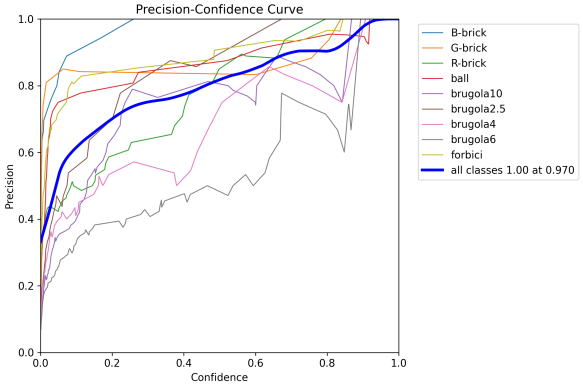
(c) `train-3` (Normalizzata)

Figura 5.3: Matrici di Confusione Normalizzate relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).

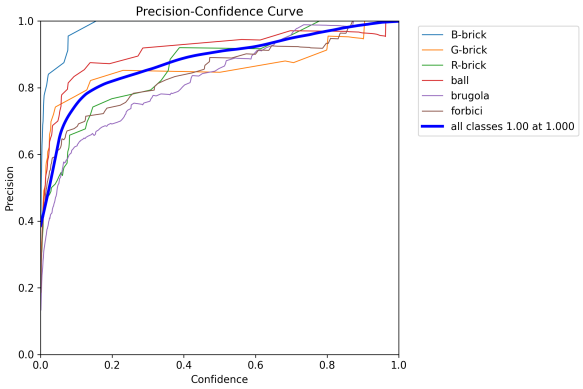
`Curva Precision-Confidence` rappresenta l'andamento della precisione in funzione della *confidence* τ, è calcolata al termine dell'addestramento sul Valid Set applicando i pesi ottimizzati `best.pt`. Osservando la Figura 5.4 e considerando le curve relative alle classi di chiavi Brugola e mattoncini, i grafici di `train-1` e `train-2` risultano molto frastagliati, possiamo prevedere incertezza nelle loro classificazioni anche nell'applicazione reale. Analizzando sia la curva blu media `"all classes"` che le curve singole si può notare come il `train-3` abbia raggiunto un'elevata stabilità di accuratezza grazie all'accorpamento delle quattro classi di chiavi Brugola. Nel tratto con $\tau \geq 0.2$ mostra un'accuratezza superiore a 0.8 ed un andamento quasi lineare, ci aspettiamo un'ottima precisione durante l'*Object Detection Live*.



(a) `train-1`



(b) `train-2`

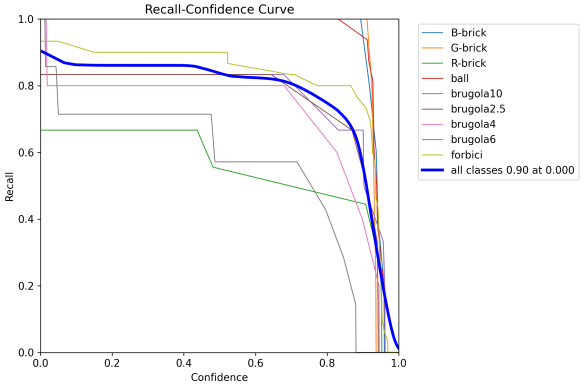


(c) `train-3`

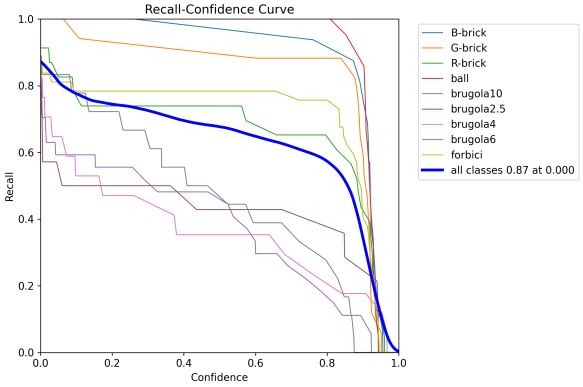
Figura 5.4: Curve Precision-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).

`Curva Recall-Confidence` rappresenta l'andamento della sensibilità in funzione della *confidence* τ, è calcolata al termine dell'addestramento sul Valid Set applicando i pesi ottimizzati `best.pt`. Considerando i grafici dei tre addestramenti di Figura 5.5 riscontriamo una situazione analoga al precedente confron-

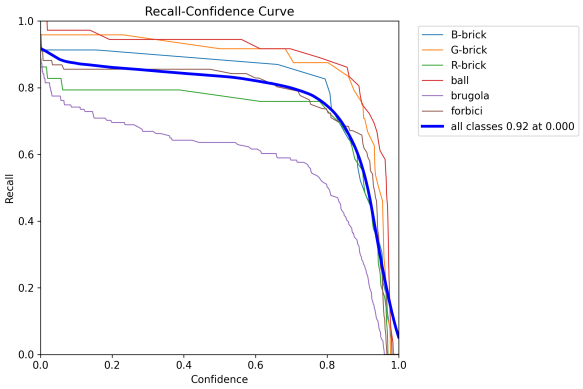
to. Il grafico del `train-1` mostra un comportamento apparentemente buono, in realtà dovuto ad un dataset ridotto. Questa interpretazione è confermata dall'analisi del grafico del `train-2` che ci mostra una netta distinzione tra le classi relative alle brugole e tutte le altre classi. Ci si aspetta un'elevata difficoltà da parte del secondo modello addestrato di rilevare le chiavi Brugola a causa di valori bassi di sensibilità per tutto il dominio della *confidence*. La pessima prestazione di sensibilità delle quattro classi di chiavi Brugola è tale da abbassare la curva media `"all classes"`, nonostante i buoni risultati delle altre classi nel `train-2`. Passando ora al grafico del `train-3` notiamo immediatamente un salto di qualità nella sensibilità. Le chiavi Brugola essendo accorpate in un'unica classe raggiungono un risultato migliore rispetto all'addestramento precedente, ma notiamo come la differenza di sensibilità tra la classe `brugola` e le altre classi a parità di *confidence* sia importante (specialmente nell'intervallo $0.1 \leq \tau \leq 0.9$). Quest'ultima considerazione è imputabile alla geometria ed al colore delle chiavi Brugola che rendono la loro rilevazione più difficoltosa rispetto a quella degli altri oggetti target.



(a) `train-1`



(b) `train-2`



(c) `train-3`

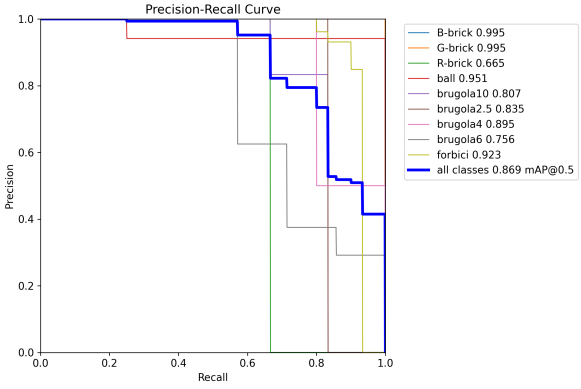
Figura 5.5: Curve Recall-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).

`Curva Precision-Recall` mette in relazione diretta l'accuratezza delle predizioni effettuate con la sensibilità e la capacità di individuazione del modello. Viene calcolata partendo dal valore di *confidence* $\tau = 1$ e mettendo sul grafico i corrispettivi valori di *Precision* e *Recall*, successivamente si prosegue per valori

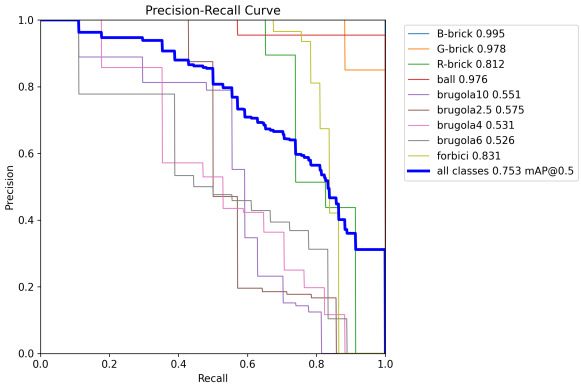
di τ decrescenti e ponendo nel grafico da sinistra a destra i corrispettivi valori di *Precision* e *Recall* per ogni valore di τ fino ad arrivare a $\tau = 0$. L'integrazione dell'area sottesa alla curva *Precision-Recall* per la classe i-esima definisce la metrica dell'*Average Precision* (AP_i):

$$AP_i = \int_0^1 \text{Precision}(\text{Recall})\, d(\text{Recall}) \tag{5.10}$$

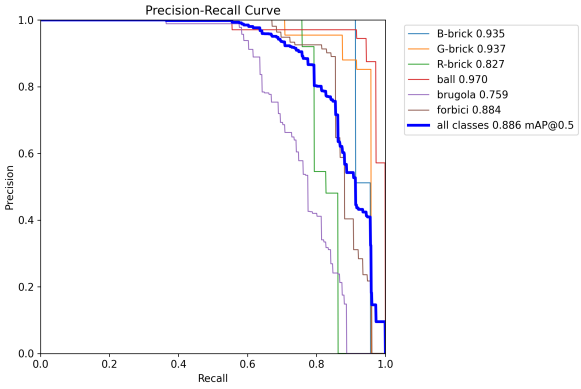
La situazione ideale è una retta costante orizzontale, cioè *Precision* pari ad 1.0 per ogni valore di *Recall*.
Osservando la Figura 5.6, notiamo come il grafico corrispondente al `train-1` ha un andamento "a gradini", simbolo di un dataset ridotto e poco vario. Per giudicare meglio le prestazioni ci spostiamo al grafico corrispondente al `train-2`, le curve relative alle classi di chiavi Brugola abbassano la curva media, dimostrando la difficoltà per il modello di distinguere le diverse dimensioni di chiavi Brugola. Osservando l'andamento decrescente della curva media `"all classes"` si arriva alla conclusione che il modello deve scegliere se essere molto accurato ma poco sensibile (zona in alto a sinistra) oppure essere molto sensibile ma molto poco accurato (zona in basso a destra). Il grafico relativo al `train-3` è quello che più si avvicina alla situazione ideale raggiungendo il valore massimo di *Mean Average Precision* tra i tre addestramenti: mAP@0.5 = 0.886 . Inoltre il `train-3` non soffre del compromesso tra accuratezza e sensibilità, si può evidenziare come ponendo Recall ≤ 0.8 la *Precision* mantenga valori molto elevati, superiori a 0.8.



(a) `train-1`



(b) `train-2`

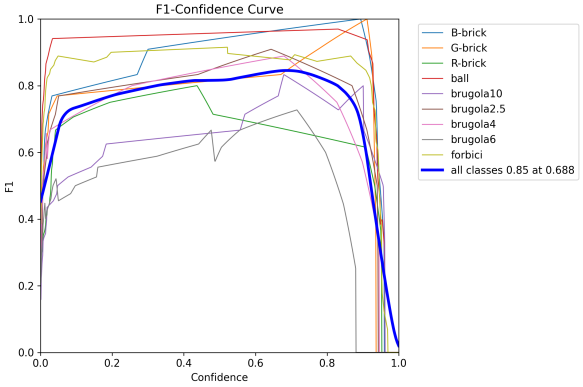


(c) `train-3`

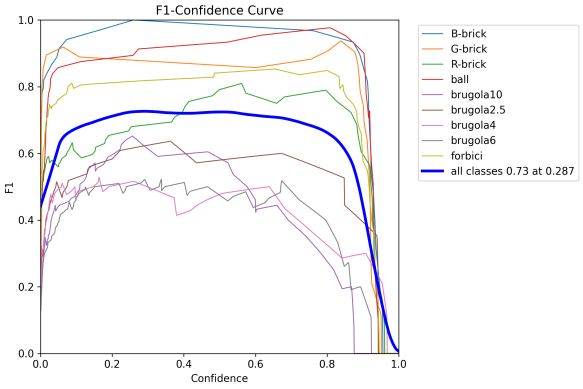
Figura 5.6: Curve Precision-Recall relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).

`Curva F1-Confidence` mostra l'andamento della metrica *F1-Score* al variare della *confidence*. L'analisi di questa curva è di fondamentale importanza per l'implementazione del modello addestrato. La *confidence* da configurare per l'applicazione reale di *Object Detection Live* non viene scelta arbitrariamente,

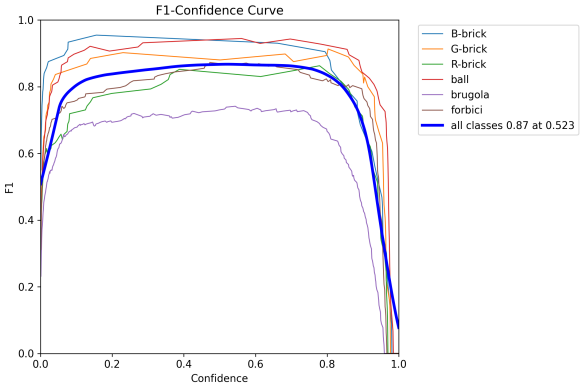
ma si determina matematicamente il valore di *confidence* ottimale, definito come l'ascissa τ corrispondente al punto di massimo della curva *F1-Confidence*. La *confidence* ottimale è tale da bilanciare nella maniera migliore possibile sia l'accuratezza di classificazione che la sensibilità di rilevamento.
Analizzando i grafici dei tre addestramenti di Figura 5.7 si possono giudicare le prestazioni generali sia in termini di *Precision* che di *Recall*. Considerando il `train-1` notiamo un elevato F1 (≈ 0.85), anche in questo caso giustificato da un dataset ridotto e poco vario. Confrontandolo col grafico relativo al `train-2` è evidente come in questo secondo caso ci sia un abbassamento generale dell'F1-Score della curva media `"all classes"` causato dai bassi risultati ottenuti nel riconoscimento delle quattro classi di chiavi Brugola per via della loro confusione inter-classe. Nel `train-3` andando a leggere la curva media osserviamo un andamento promettente, caratterizzato da un elevato F1 massimo (≈ 0.87) e da un ampio intervallo di *confidence* $0.1 \leq \tau \leq 0.85$ dove la curva si attesta quasi costante a valori elevati di F1-Score ≥ 0.8 .



(a) `train-1`



(b) `train-2`



(c) `train-3`

Figura 5.7: Curve F1-Confidence relative alle tre sessioni di addestramento: `train-1` (a), `train-2` (b) e `train-3` (c).

5.4 Valutazione Sperimentale e Misure di Miglioramento

Successivamente all'aver valutato le prestazioni dei tre addestramenti `train-1`, `train-2`, `train-3` tramite l'analisi dei grafici e delle metriche di output di ogni

addestramento, si può passare alla loro implementazione sul robot umanoide Unitree G1 EDU per poi valutarne sperimentalmente le prestazioni durante l'applicazione reale di *Object Detection Live*. Si affida ad uno script Python il compito di ricevere in input il file dei pesi ottimizzati `best.pt` della rete neurale addestrata, sulla base dei pesi ottimizzati tramite lo script sarà possibile per il modello YOLO elaborare le predizioni e definire le bounding box. Inoltre lo script conoscendo la posizione in pixel dei centri delle bounding box, tramite l'utilizzo del modulo di profondità della telecamera, fornirà una stima della posizione del centro della bouding box in coordinate cartesiane aventi come origine il centro ottico della telecamera (il sensore cromatico RGB della Intel RealSense D435i). Nel preambolo dello script possiamo evidenziare due comandi:

- `import rclpy`: importa la libreria client ufficiale di Python per ROS 2 (*ROS Client Library for Python*). Costituisce il modulo fondamentale per inizializzare i nodi ROS.

- `from ultralytics import YOLO`: importa il framework Ultralytics per il caricamento del modello di intelligenza artificiale YOLOv11 e l'esecuzione dell'inferenza.

Il setup delle prove sperimentali, come si deduce dalla Figura 5.8, è così composto:

- il robot umanoide Unitree G1 EDU è la piattaforma robotica di riferimento, per queste prove sperimentali si è deciso di tenerlo in una condizione statica, posizionandolo davanti al banco da laboratorio ed abbassandolo leggermente (tenendolo appeso alla gru a carrello pieghevole in dotazione al robot) per avere una buona inquadratura degli oggetti target;

- la telecamera RGB-D Intel RealSense D435i in dotazione all'Unitree G1 EDU è il sensore usato per l'*Object Detection Live*, è posizionata sul capo del robot;

- gli oggetti target sono appoggiati sul banco da laboratorio il quale è di colore bianco ed uniforme, inoltre su di esso vengono posti anche oggetti casuali, in particolare cablaggi di colore nero, per controllare la presenza di Falsi Positivi;

- sul PC di sviluppo, monitorato dall'operatore, viene lanciato lo script Python di *Object Detection Live*. Sulla base del file contenente i pesi ottimizzati `best.pt` del relativo addestramento, la rete neurale YOLO addestrata elabora le sue predizioni per ogni fotogramma che riceve dalla telecamera;

- l'operatore si occupa di monitorare il PC di sviluppo e di spostare gli oggetti target in modo da farli acquisire dalla telecamera nelle varie situazioni ed angolazioni possibili (completamente visibili, parzialmente coperti dalla mano o da un altro oggetto, parzialmente fuori dall'inquadratura).

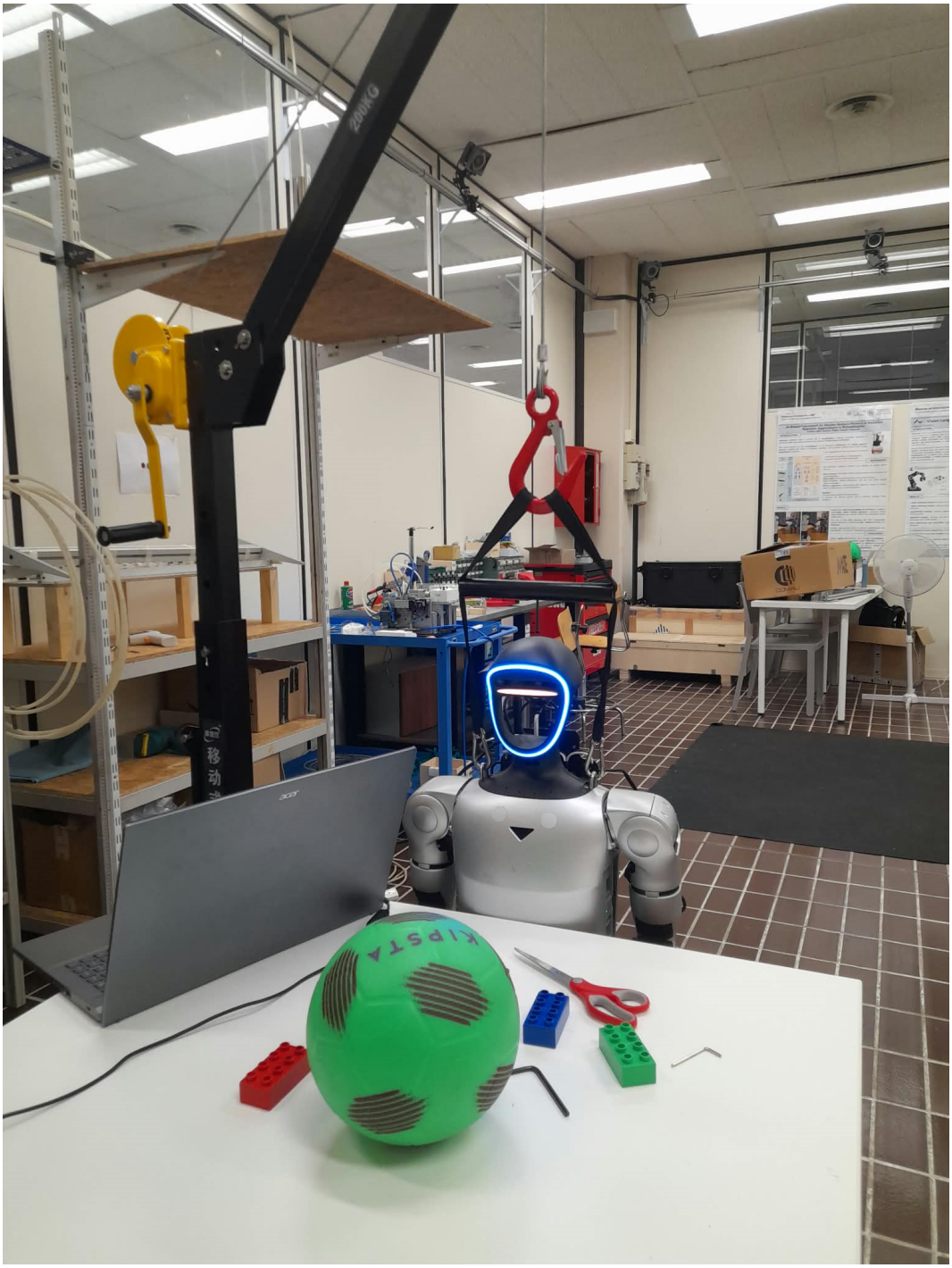


Figura 5.8: Setup delle prove sperimentali.

La valutazione sperimentale verrà redatta analizzando singolarmente i tre addestramenti `train-1`, `train-2`, `train-3` così da osservare l'evoluzione nelle prestazioni durante l'applicazione reale di *Object Detection Live*.

5.4.1 train-1

Dalle prove sperimentali del primo addestramento, nonostante la valutazione delle metriche di valutazione e dei grafici di addestramento fosse apparentemente buona, si notano le grandi limitazioni del modello.
Dalla Figura 5.7 leggiamo che la *confidence* ottimale per il `train-1` è pari a 0.688, di conseguenza si è impostato il parametro `conf` nelle varie prove sperimentali a 0.7 ed in altre situazioni a 0.6 per provare ad aumentare la sensibilità del modello. Il `train-1` è dotato di una buona capacità di riconoscimento della classe `ball`, invece per tutte le altre classi si è osservato che in situazioni di sfondo uniforme e bianco il riconoscimento è dotato di una buona sensibilità, altrimenti su sfondo casuale (oppure sfondo bianco ma con numerosi oggetti casuali inquadrati dalla telecamera) si riscontrano numerosi Falsi Negativi e Falsi Positivi. In tutte le situazioni si è notato gravi problemi di confusione inter-classe tra le quattro classi di chiavi Brugola ed una bassa sensibilità per quanto riguarda la classe `forbici`, se posizionate ad angolazioni sfavorevoli il modello non è in grado di riconoscerle. In linea generale il problema del primo addestramento è la bassa sensibilità, in quanto solo la classe `ball` risulta riconoscibile in tutte le situazioni, mentre tutte le altre classi in condizioni di parziale copertura (ad esempio una mano appoggiata sopra al'oggetto) o scarsa illuminazione risultano impossibili da riconoscere da parte del modello addestrato.
Si può facilmente concludere che il `train-1` sia completamente inadatto alla mansione di riconoscimento oggetti, sono necessari numerosi miglioramenti.

5.4.2 train-2

Per superare i problemi del primo addestramento si è deciso di agire sul dataset, è stato scattato ed annotato un ingente numero di immagini dove sono presenti numerosi oggetti target di classi diverse, oggetti target parzialmente coperti o in ombra ed oggetti target in condizioni di sfondo variabili. Siamo passati dalle 594 immagini del dataset del `train-1` alle 833 immagini per il `train-2`. Per ottenere un ulteriore miglioramento è stata applicata la funzione di pre-elaborazione *Resize* per adeguare le immagini alla risoluzione nativa di YOLO ed è stata applicata la tecnica della *Data Augmentation* con i parametri di Tabella 5.1. Così facendo abbiamo ottenuto un dataset nettamente più vario ed esteso, per questo motivo è stato riscontrato un tempo computazionale di addestramento della rete neurale particolarmente elevato, come testimonia la Tabella 5.2.
Dalla Figura 5.7 si legge che la *confidence* ottimale per il `train-2` è pari a 0.287, nonostante il valore ottimale sia questo è stato deciso di utilizzare valori di *confidence* tra 0.6 e 0.7 così da abbassare la sensibilità ed eliminare la maggior parte dei Falsi Positivi. Durante le prove sperimentali è stato riscontrato un miglioramento di sensibilità per le classi che precedentemente ne soffrivano. La confusione inter-classe tra le quattro classi di chiavi Brugola si è mantenuta in egual misura. Di grande ri-

levanza è l'osservazione di un elevato numero di Falsi Positivi attribuiti ad una delle classi di chiavi Brugola nel caso di sfondo misto e variabile, quando la telecamera inquadra zone dove sono presenti dei cablaggi di colore nero, accade che qualche cavo (a causa della similitudine in colore e forma) venga rilevato come una chiave Brugola. Dalla Figura 5.9 si osserva la confusione inter-classe ed un Falso Positivo dovuto ad un cavo nero inquadrato. Nel caso di sfondo bianco ed uniforme le prestazioni generali (escludendo le chiavi Brugola) sono accettabili, come si appare in Figura 5.10 dove i problemi appena evidenziati non sono presenti.

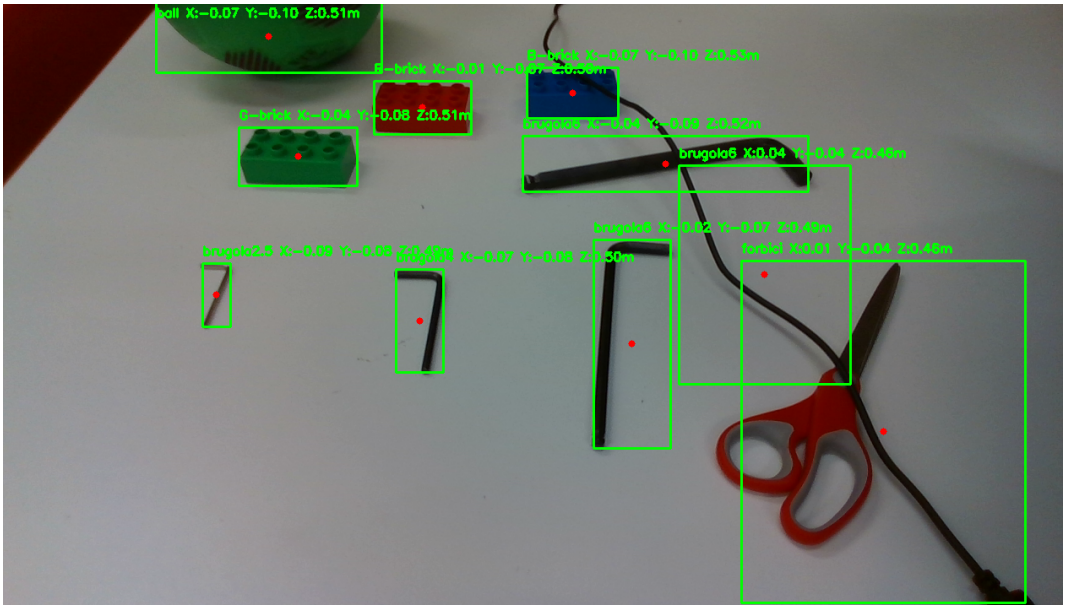


Figura 5.9: Riconoscimento oggetti del `train-2`, da notare il Falso Positivo e la confusione inter-classe.

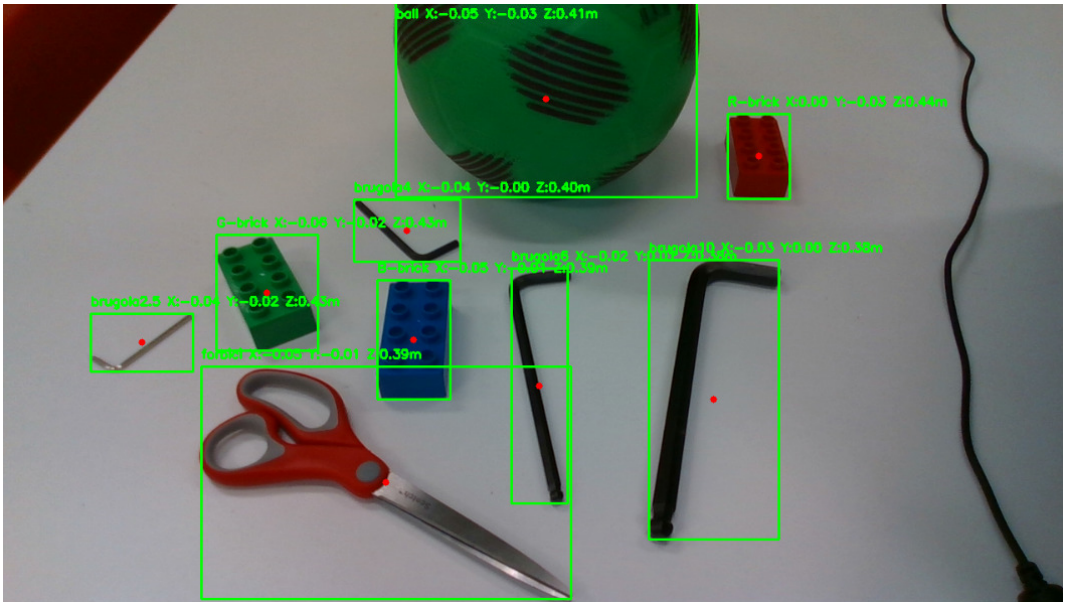


Figura 5.10: Riconoscimento oggetti del `train-2`, si osserva l'assenza di errori di riconoscimento.

A questo punto si può affermare che le prestazioni del `train-2` sono peggiori di quanto immaginassimo, il suo comportamento è accettabile solo su sfondo bianco ed uniforme. Le prestazioni del `train-2` sono ancora insufficienti per l'utilizzo in una possibile applicazione reale, sono necessari ulteriori miglioramenti.

5.4.3 train-3

Per il `train-3` si è deciso di inserire nel dataset delle immagini `"null"` dove sono presenti dei cablaggi di colore nero, come tentativo di eliminare il problema osservato nel modello addestrato precedente, arrivando ad 891 immagini nel dataset totale. È stato deciso di accorpare le quattro classi di chiavi Brugola in un'unica classe denominata `"brugola"`, eliminando il problema della loro confusione inter-classe così da focalizzarci sul loro accurato riconoscimento e sulla risoluzione del problema dei Falsi Positivi dei cablaggi. Analogamente al `train-2`, anche al dataset del `train-3` abbiamo applicato le funzioni di pre-elaborazione di *Resize* e *Data Augmentation* con i parametri di Tabella 5.2. La vera differenza rispetto al `train-2` è nel modello di addestramento scelto, infatti per il `train-3` è stato scelto il modello YOLOv11n (*Nano*), riducendo i tempi computazionali di addestramento.
Dalla Figura 5.7 leggiamo che la *confidence* ottimale per il `train-3` è pari a 0.523, ma anche in questo caso la *confidence* scelta è del valore di 0.6 o 0.7 per mantenere una sensibilità non eccessiva. Dalle prove sperimentali, come acquisito in Figura 5.11, emerge un'ottima sensibilità ed accuratezza di classificazione per tutte le classi, inoltre a causa dell'utilizzo del modello YOLO *Nano* è stata osservata una latenza estremamente ridotta. Come si evince dalla Figura 5.12, il problema dei Falsi Positivi della classe `"brugola"` attribuiti a cavi neri è ancora presente in qualche occasione.

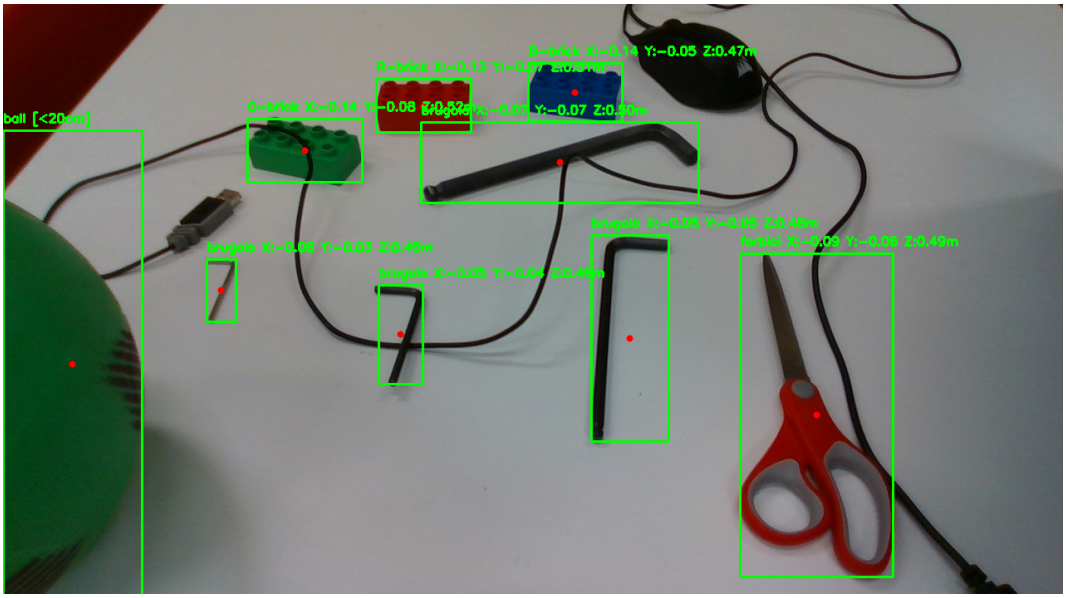


Figura 5.11: Riconoscimento oggetti del `train-3`, si evidenzia l'accuratezza di riconoscimento.

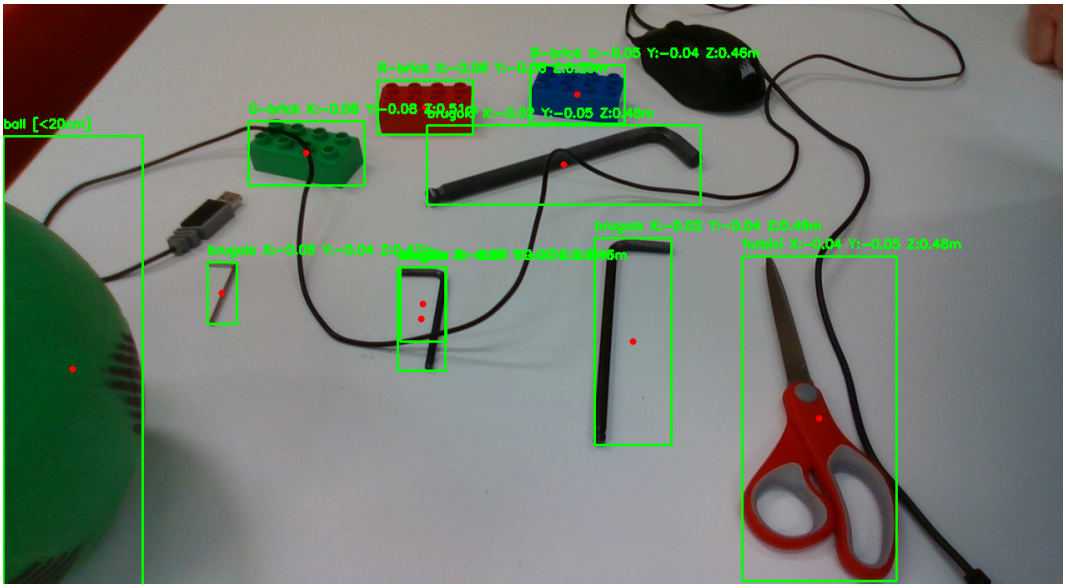


Figura 5.12: Riconoscimento oggetti del `train-3`, da notare le due bounding box sovrapposte.

Come appena descritto, possiamo valutare il `train-3` come un buon modello addestrato capace di svolgere, in condizioni favorevoli di sfondo ed illuminazione, il suo compito.

Capitolo 6

SLAM

Il presente capitolo tratta la seconda delle due integrazioni sensoriali sviluppate, andando a descrivere dal punto di vista teorico il processo di sviluppo dell'architettura di **SLAM** attraverso la quale permettiamo al robot umanoide di eseguire la mappatura ambientale tridimensionale, la sua localizzazione spaziale ed una navigazione autonoma attraverso dei punti di passaggio denominati *waypoint*. Inoltre si valuterà sperimentalmente la fase di mappatura tridimensionale mediante ricostruzione della nuvola di punti (*point cloud*) rappresentante l'ambiente circostante al robot umanoide. L'esecuzione dello SLAM è resa possibile dal sensore LiDAR Livox Mid-360 in dotazione al robot umanoide Unitree G1 EDU e posizionato sul suo capo, come si evince dalla Figura 3.3.

6.1 Architettura Software dello SLAM

L'infrastruttura software per eseguire i compiti prefissati è costituita da due script in linguaggio C++ denominati `slam_mapping_saver.cpp` e `slam_navigation_actions.cpp` da eseguire sul PC di sviluppo connesso tramite interfaccia Ethernet all'Unitree G1 EDU. Le condizioni di collegamento tra robot umanoide e PC di sviluppo sono le stesse descritte alla Sezione 4.1 dedicata al sistema operativo Ubuntu.

I due script si fondano su due librerie fondamentali. La prima è `unitree_sdk2`, descritta alla Sezione 4.5, funge da ponte di comunicazione tra il robot umanoide ed il PC di sviluppo. La seconda è `nlohmann/json` [34] la quale si occupa di interpretare ed organizzare le informazioni scambiate. In particolare, consente la gestione dei file in formato JSON sui quali sono salvate le coordinate spaziali dei punti di passaggio (*waypoint*), inoltre la libreria viene integrata per estrarre facilmente con continuità la posizione in tempo reale dell'umanoide e gli stati di avanzamento della navigazione. Il processo di mappatura, localizzazione e navigazione autonoma è così eseguito:

`slam_mapping_saver.cpp` alla sua esecuzione da terminale del PC di sviluppo avviene l'attivazione del sensore LiDAR del robot umanoide con conseguente avvio della mappatura SLAM. All'aumentare del tempo di mappatura, maggiore sarà il numero di punti rilevati dal sensore, così facendo si otterrà una mappatura più completa e con minori punti ciechi. Successivamente, con lo script ancora in esecuzione, l'operatore movimenterà il robot attraverso i punti

di passaggio di interesse mediante controllo manuale. Ogni volta che si raggiunge uno dei punti di interesse tramite controllo manuale, l'operatore avrà il compito di premere il pulsante del PC di sviluppo designato dallo script alla definizione del *waypoint*, in questo modo la scheda di bordo del robot memorizzerà le coordinate (insieme alla sua posa in quel preciso istante) del *waypoint*. Quando l'esecuzione dello script viene terminata, sul PC di sviluppo avverrà il salvataggio della *point cloud* in formato `.pcd` e del file `.json` relativo agli *waypoint* definiti durante la navigazione manuale. Quest'ultimo file contiene sia le coordinate che la posa del robot in ogni *waypoint* e la relativa sequenza di memorizzazione di essi.

`slam_navigation_actions.cpp` alla sua esecuzione da terminale sul PC di sviluppo riceve in input sia la *point cloud* che il file di coordinate, posa e sequenza degli *waypoint*. Al termine del caricamento da parte del robot della mappa tridimensionale dell'ambiente si può avviare la sua relocalizzazione, il robot riuscirà a riconoscere la sua posizione attuale all'interno della *point cloud*, ciò avviene in quanto il robot sovrappone la *point cloud* fornita in input con la scansione LiDAR in tempo reale. La relocalizzazione efficace avviene nel caso in cui l'ambiente di lavoro del robot umanoide sia ricco di elementi geometrici statici ed eterogenei nello spazio quali pareti, angoli, pilastri ed arredi. Il robot potrebbe non riuscire a relocalizzarsi correttamente nei casi di modifiche strutturali o modifica della posizione degli oggetti di grandi dimensioni presenti nell'ambiente. Un caso particolare è la *geometric degeneracy*, in ambienti caratterizzati da forte simmetria o da superfici piane estese e prive di discontinuità e vincoli geometrici (quali lunghi corridoi rettilinei), la scansione LiDAR soffre dell'assenza di vincoli lungo quella determinata direzione impedendo la stima univoca della posizione del robot. A relocalizzazione completata è possibile dare il comando di avvio della navigazione autonoma sequenziale, il robot inizierà a muoversi in completa autonomia attraversando ognuno degli *waypoint* nell'ordine in cui sono stati definiti in precedenza. Al raggiungimento dell'ultimo *waypoint* della sequenza definita nel file `.json`, il robot umanoide ripeterà il ciclo di navigazione autonoma sequenziale ripartendo dal primo *waypoint*, la navigazione si interrompe nel momento in cui l'operatore termina l'esecuzione dello script dal PC di sviluppo. Inoltre, mediante rapide modifiche a questo script, è possibile abbinare ad ogni *waypoint* un'azione da eseguire al suo raggiungimento. Una volta terminata l'azione prevista il robot proseguirà autonomamente con la sequenza di navigazione.

6.2 Point Cloud

La mappatura tridimensionale, mediante *point cloud*, dell'ambiente circostante al robot umanoide funge da fondamenta per tutte le operazioni di localizzazione spazia-

le e navigazione autonoma. Per l'approfondimento sulla mappatura tridimensionale si considererà una *point cloud* eseguita dal LiDAR Livox Mid-360 in dotazione al robot umanoide Unitree G1 EDU nei locali di i-Labs Industry a Jesi (AN).

La visualizzazione delle *point cloud* in formato `.pcd` è reso possibile da uno script in linguaggio Python denominato `slam_view.py` nel quale emergono le due librerie fondamentali per ottenere la corretta elaborazione grafica. La libreria `Open3D` permette la lettura del file in formato `.pcd` mentre `PyVista` ha garantito un rendering grafico di alta qualità.

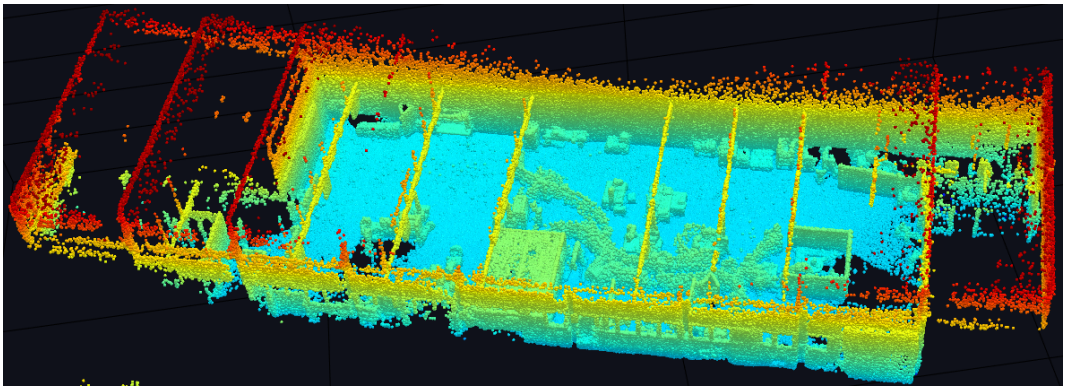


Figura 6.1: Prima vista prospettica della *point cloud* grezza rilevata dal sensore LiDAR del robot umanoide Unitree G1 EDU.

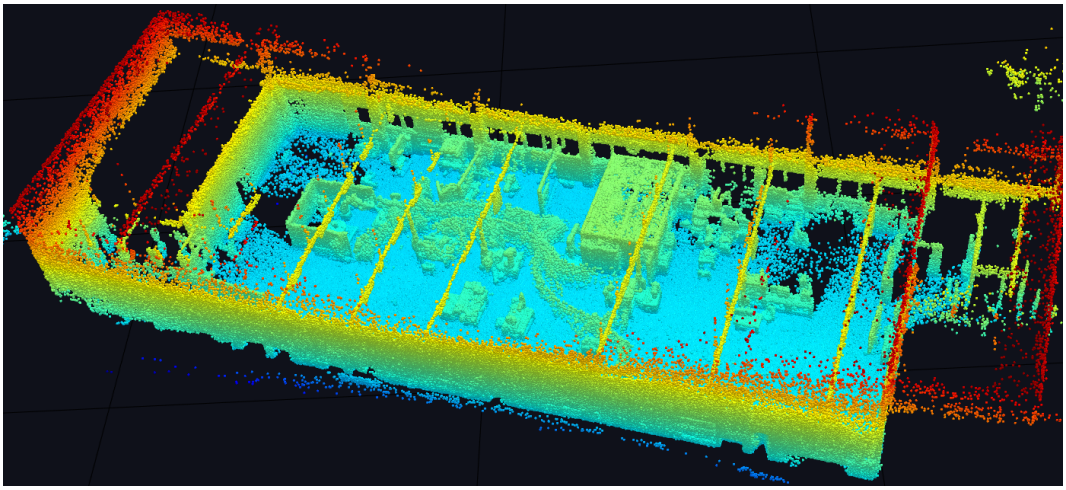


Figura 6.2: Seconda vista prospettica della *point cloud* grezza rilevata dal sensore LiDAR del robot umanoide Unitree G1 EDU.

La visualizzazione grafica della nuvola di punti grezza (cioè priva di elaborazioni successive alla rilevazione) è quella che appare dalla Figura 6.1 e dalla Figura 6.2 secondo due viste prospettiche diverse. Si può immediatamente notare come il soffitto non sia stato rilevato dal sensore, ciò è dovuto al FOV del sensore LiDAR visibile in Figura 3.4.

Nelle zone centrali della mappa tridimensionale si possono notare delle scie continue di punti che attraversano buona parte dell'ambiente. Queste scie si trovano ad un'altezza comparabile all'essere umano, si può quindi interpretare che queste scie sono formate dalla rilevazione da parte del sensore LiDAR di un operatore umano in movimento durante l'esecuzione della mappatura. Questi agglomerati di punti dovuti all'operatore in movimento sono caratterizzati da una densità spaziale locale di punti nettamente inferiore a quella delle superfici fisse quali elementi strutturali ed arredi. Per questo motivo è possibile ridurre o eliminare questi agglomerati di punti indesiderati nella fase di *post-processing* della *point cloud* mediante un filtraggio statistico.

Si è scritto uno script in linguaggio Python addetto all'integrazione dell'algoritmo *Statistical Outlier Removal*. La funzione fondamentale di questo script è la seguente:

```
pcd.remove_statistical_outlier(nb_neighbors=30, std_ratio=1.5)
```

Questa funzione applica un filtro statistico basato sulla distanza spaziale dei punti fra loro, così da riconoscere le zone a minor densità di punti per poi sopprimerle. Per ciascun punto della nuvola $\mathbf{p}_i$, l'algoritmo calcola la distanza media d_i rispetto ai k punti vicini più prossimi (impostato a `nb_neighbors=30`). Un punto viene classificato come *outlier*, e quindi rimosso dalla mappa tridimensionale, se la sua distanza media d_i supera la soglia di tolleranza impostata dal parametro `std_ratio=1.5`. Applicando il filtraggio mediante l'algoritmo *Statistical Outlier Removal* in fase di *post-processing* alla *point cloud* grezza è stato reso possibile ottenerne una versione molto più pulita dal rumore di misura grazie alla rimozione di tutte quelle zone a bassa densità di punti, in particolar modo l'estesa scia rappresentante l'operatore umano in movimento. La versione della *point cloud* successiva al *post-processing* di filtraggio statistico è quella visibile in Figura 6.3, Figura 6.4 e Figura 6.5.

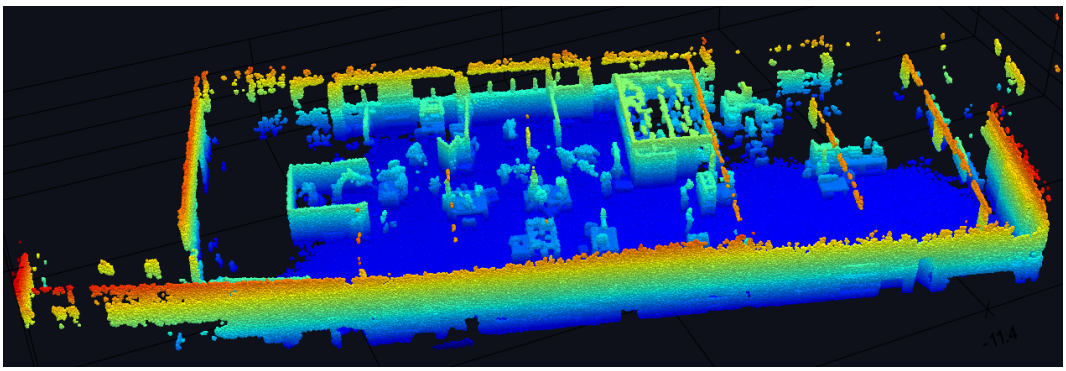


Figura 6.3: Prima vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico.

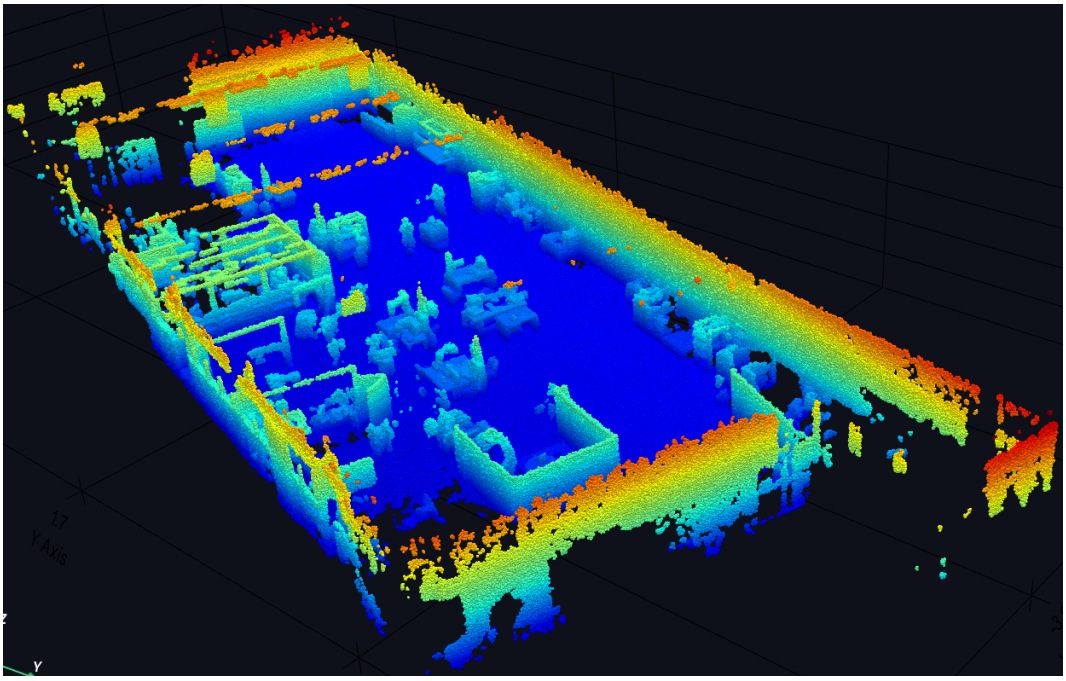


Figura 6.4: Seconda vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico.

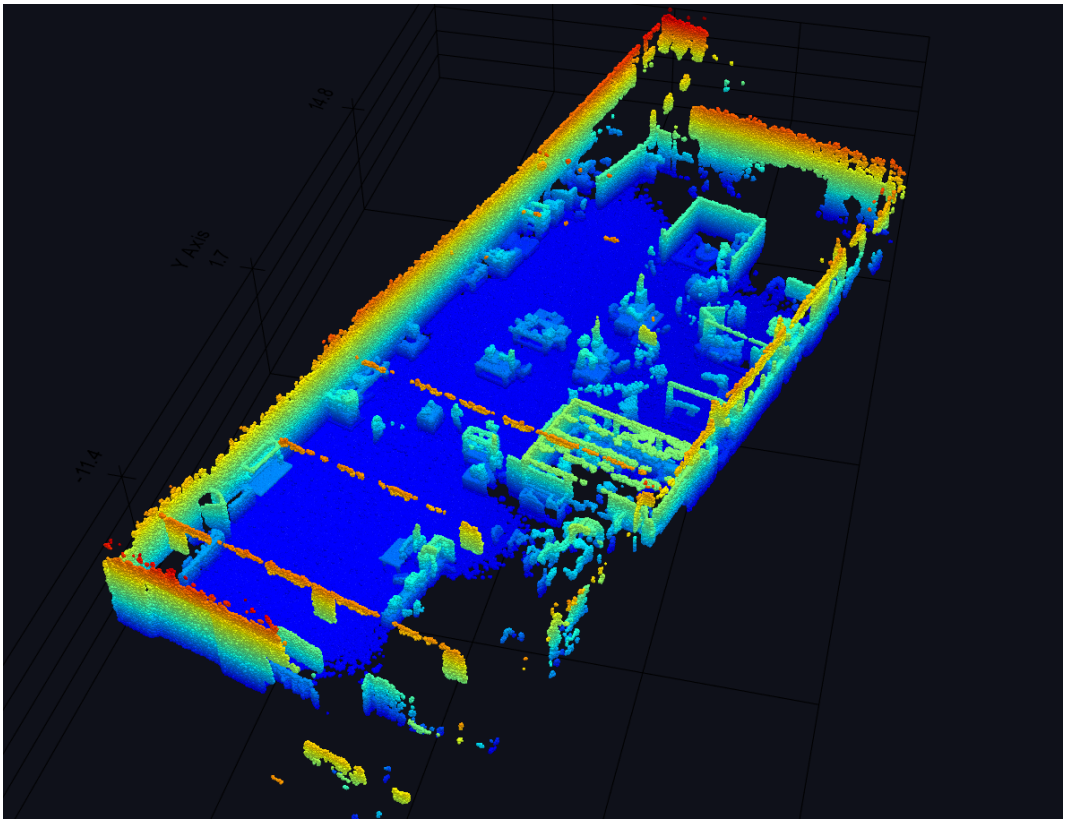


Figura 6.5: Terza vista prospettica della *point cloud* successiva al *post-processing* mediante filtraggio statistico.

Risulta immediato osservare la completa rimozione della scia di agglomerati di punti a bassa densità spaziale causata dall'operatore in movimento durante l'esecuzione della mappatura. Si può notare che alcuni elementi presenti in prossimità del soffitto nella versione grezza sono stati eliminati dal filtraggio statistico. Questo è dovuto al ridotto FOV verticale del sensore LiDAR sul robot umanoide Unitree G1 EDU che permette la rilevazione di pochi punti sulle superfici ad altezza elevata. Perciò in presenza di questi elementi ad altezza elevata, gli agglomerati di punti hanno bassa densità spaziale locale portando alla loro rimozione dalla *point cloud* grezza. Ciò è irrilevante per le operazioni successive alla mappatura come la navigazione autonoma, in quanto non siamo interessati a quegli elementi posti ad altezze di gran lunga superiori a quella del robot umanoide.

Complessivamente è possibile affermare gli ottimi risultati dell'applicazione dell'algoritmo *Statistical Outlier Removal* sulla versione grezza della *point cloud*. La *point cloud* risultante dal *post-processing* è nettamente più pulita e leggibile, inoltre è sicuramente in grado di garantire una relocalizzazione efficace al robot umanoide rappresentando in maniera più veritiera la morfologia dell'ambiente.

Capitolo 7

Conclusioni

7.1 Sintesi dei Risultati

Nell'ambito di *Object Detection Live*, i risultati sono soddisfacenti ed adeguati. Si può concludere che per garantire un buon riconoscimento è fondamentale strutturare correttamente il dataset in ogni sua fase di creazione (sia acquisizione delle immagini che pre-elaborazione) e scegliere il giusto modello ed i giusti parametri di addestramento. Valutando sperimentalmente l'ultimo modello addestrato è stata raggiunta una situazione in cui il riconoscimento degli oggetti target è sia sensibile che accurato, garantendo inoltre una bassa latenza dovuta all'utilizzo del modello di addestramento YOLOv11 *Nano*. Accorpando le chiavi Brugola in un'unica classe semantica è stato reso possibile evitare il problema di riconoscimento della dimensione di ogni chiave Brugola presente, per di più l'aggiunta di immagini di sfondo con presenza di cablaggi ha portato come risultato l'elevata riduzione della probabilità di incorrere in Falsi Positivi.

Nell'ambito dello SLAM la descrizione dell'architettura software di mappatura tridimensionale, localizzazione spaziale e navigazione autonoma funge da base solida per le future prove sperimentali sul robot umanoide Unitree G1 EDU. Si è inoltre arrivati ad una corretta comprensione del funzionamento delle *point cloud* e del processo di sviluppo della loro corretta visualizzazione. Infine si è raggiunto il risultato atteso di *post-processing* della *point cloud* mediante filtraggio statistico tramite l'applicazione dell'algoritmo *Statistical Outlier Removal*, riuscendo così ad eliminare gli agglomerati di punti a bassa densità spaziale indesiderati. La versione della *point cloud* successiva al *post-processing* risulta rappresentare in modo veritiero l'ambiente di lavoro del robot umanoide.

In conclusione, il presente elaborato ha raggiunto gli obiettivi prefissati nei suoi vari aspetti, dimostrando come il robot umanoide Unitree G1 EDU, tramite futuri sviluppi del lavoro appena descritto, possegga ampie potenzialità di integrazione sensoriale avanzata.

7.2 Sviluppi Futuri

La precedente applicazione di *Object Detection Live* può essere utilizzata come base per modelli addestrati più elaborati e per mansioni di riconoscimento avanzate come la *Instance Segmentation*. Integrando la struttura di riconoscimento oggetti in tempo reale con la cinematica degli arti superiori del robot umanoide si può presupporre futuri compiti di *grasping* e manipolazione fine.

Sul fronte dello SLAM, della mappatura ambientale e della localizzazione spaziale si può intuire come passo successivo l'integrazione del sensore LiDAR per scopi di navigazione autonoma elaborata come l'*Obstacle Avoidance*, cioè il riconoscimento in tempo reale da parte del robot umanoide di ostacoli statici e dinamici sulla sua traiettoria di movimento prefissata, con successiva correzione dinamica del percorso di camminata.

Lo sviluppo naturale dell'integrazione dei sensori RGB-D e LiDAR sul robot umanoide Unitree G1 EDU sono quelle mansioni che necessitano il funzionamento in contemporanea di entrambi i sensori, cioè la localizzazione e navigazione spaziale dovrà lavorare in sinergia con il riconoscimento oggetti ed il calcolo della loro posizione a corto raggio per la manipolazione fine. Il framework di navigazione spaziale (SLAM) dovrà condurre l'umanoide ad avvicinarsi autonomamente alla postazione di lavoro desiderata. Solo una volta che lo SLAM avrà garantito il posizionamento corretto e la stabilizzazione del corpo del robot di fronte alla postazione, l'architettura di *Object Detection Live* subentrerà per elaborare le coordinate metriche locali degli oggetti target e demandare l'esecuzione della presa alle catene cinematiche degli arti superiori (*grasping*). La riduzione dei Falsi Positivi ottenuta mediante l'arricchimento del dataset e la configurazione della soglia di *confidence* rappresenta un primo passo verso applicazioni di manipolazione robotica. Tuttavia, la presenza di errori residui richiede ulteriori attività di validazione e l'integrazione di adeguate strategie di sicurezza prima di poter impiegare il sistema in operazioni autonome di presa.

Bibliografia

[1] Waseda University. Humanoid robotics institute: Wabot-1. https://www.humanoid.waseda.ac.jp/booklet/kato_2.html. Accesso: Ottobre 2026.

[2] Global Market Insights. Humanoid robot market size, share & industry analysis report. https://www.gminsights.com/industry-analysis/humanoid-robot-market, 2024. Accesso: Ottobre 2026.

[3] Emerald Insight. Humanoid robots: from the laboratory to the industry. https://www.emerald.com/ir/article/52/4/458/1269760/Humanoid-robots-from-the-laboratory-to-the, 2024. Accesso: Ottobre 2026.

[4] Boston Dynamics. Atlas: The world's most dynamic humanoid robot. https://bostondynamics.com/products/atlas/, 2024. Accesso: Ottobre 2026.

[5] HDMotori. Robot umanoidi nelle fabbriche d'auto: il futuro della produzione automobilistica. https://www.hdmotori.it/robot-umanoidi-fabbriche-auto/, 2024. Accesso: Ottobre 2026.

[6] BMW Group. Successful test of humanoid robots at bmw group plant spartanburg. https://www.press.bmwgroup.com/global/article/detail/T0444265EN/successful-test-of-humanoid-robots-at-bmw-group-plant-spartanburg, 2024. Accesso: Ottobre 2026.

[7] Amazon. Amazon introduces new robotics solutions. https://www.aboutamazon.com/news/operations/amazon-introduces-new-robotics-solutions, 2024. Accesso: Ottobre 2026.

[8] UBTECH Robotics. Ubtech industrial humanoid robot solutions. https://www.ubtrobot.com/en/humanoid/solutions/industry, 2024. Accesso: Ottobre 2026.

[9] Smart Analytics Global. Global humanoid robot shipments 2026: Agibot, unitree and market trends. https://smartanalyticsglobal.com/global-humanoid-robot-shipments-2026-agibot-unitree/, 2026. Accesso: Ottobre 2026.

[10] MDPI. Applied sciences: Volume 16, issue 5, article 2371. https://www.mdpi.com/2076-3417/16/5/2371, 2026. Accesso: Ottobre 2026.

[11] arXiv. Preprint arxiv:2506.22827. https://arxiv.org/abs/2506.22827, 2025. Accesso: Ottobre 2026.

[12] Taylor Hodgson. Humanoid manipulation. https://roblab.org/theses/2026-Humanoid-Manipulation-Hodgson-Taylor.pdf, 2026. Master's thesis, RobLab. Accesso: Ottobre 2026.

[13] A. Ryan. Humanoid robotics. https://roblab.org/theses/2025-Humanoid-Ryan.pdf, 2025. Master's thesis, RobLab. Accesso: Ottobre 2026.

[14] IEEE. Document 11168008. https://ieeexplore.ieee.org/abstract/document/11168008, 2026. Accesso: Ottobre 2026.

[15] Unitree Robotics. Unitree g1 humanoid agent. https://www.unitree.com/g1, 2024. Accesso: Ottobre 2026.

[16] Unitree Robotics. Unitree g1 user manual. https://marketing.unitree.com/article/en/G1/User_Manual.html, 2024. Accesso: Ottobre 2026.

[17] Intel RealSense. Intel realsense depth camera d435i. https://www.realsenseai.com/products/depth-camera-d435i/, 2024. Accesso: Ottobre 2026.

[18] Livox Tech. Livox mid-360 3d lidar sensor. https://www.livoxtech.com/mid-360, 2024. Accesso: Ottobre 2026.

[19] Livox Tech. Livox mid-360 lidar sensor specifications. https://www.livoxtech.com/mid-360/specs, 2024. Accesso: Ottobre 2026.

[20] Open Robotics. Ros: Robot operating system. https://www.ros.org/, 2024. Accesso: Ottobre 2026.

[21] Open Robotics. About ros 2 humble hawksbill documentation. https://docs.ros.org/en/humble/About-ROS.html, 2024. Accesso: Ottobre 2026.

[22] Ultralytics. Ultralytics yolo11 documentation. https://docs.ultralytics.com/it/models/yolo11, 2024. Accesso: Ottobre 2026.

[23] Ultralytics. Object detection with ultralytics yolo. https://docs.ultralytics.com/it/tasks/detect, 2024. Accesso: Ottobre 2026.

[24] Ultralytics. Instance segmentation with ultralytics yolo. https://docs.ultralytics.com/it/tasks/segment, 2024. Accesso: Ottobre 2026.

[25] Ultralytics. Pose estimation with ultralytics yolo. `https://docs.ultralytics.com/it/tasks/pose`, 2024. Accesso: Ottobre 2026.

[26] Ultralytics. Oriented bounding boxes (obb) object detection with ultralytics yolo. `https://docs.ultralytics.com/it/tasks/obb`, 2024. Accesso: Ottobre 2026.

[27] Ultralytics. Image classification with ultralytics yolo. `https://docs.ultralytics.com/it/tasks/classify`, 2024. Accesso: Ottobre 2026.

[28] Roboflow. Roboflow: Computer vision datasets and computer vision tools. `https://roboflow.com/`, 2024. Accesso: Ottobre 2026.

[29] Roboflow. Roboflow universe: Open source computer vision datasets & pretrained models. `https://universe.roboflow.com/`, 2024. Accesso: Ottobre 2026.

[30] Unitree Robotics. Unitree robotics open source github repository. `https://github.com/unitreerobotics`, 2024. Accesso: Ottobre 2026.

[31] Roboflow. Data augmentation in computer vision. `https://roboflow.com/blog/data-augmentation`, 2024. Accesso: Ottobre 2026.

[32] Ultralytics. Yolo performance metrics. `https://docs.ultralytics.com/guides/yolo-performance-metrics`, 2024. Accesso: Ottobre 2026.

[33] Ultralytics. What is overfitting in computer vision and how to prevent it? `https://www.ultralytics.com/blog/what-is-overfitting-in-computer-vision-how-to-prevent-it`, 2024. Accesso: Ottobre 2026.

[34] Niels Lohmann. JSON for Modern C++. `https://json.nlohmann.me/`, 2026. Accesso: Ottobre 2026.

Ringraziamenti

Desidero ringraziare il mio relatore, il professor Matteo Claudio Palpacelli, il mio correlatore, il Dott. Federico Neri e l'Ing. Federico Gentilucci per la possibilità di svolgere il progetto di tirocinio e tesi e per tutto il supporto dato durante le attività svolte al MIR Laboratory e per la successiva scrittura della tesi. Ringrazio inoltre i-Labs Industry per l'aiuto ricevuto sulla fase finale dell'elaborato.

Ringrazio tutti i miei amici e tutti i compagni di studio che mi hanno accompagnato lungo il percorso di studi con cui anche le giornate più noiose e pesanti diventavano occasioni per una risata. Un ringraziamento particolare va ad Alessio, amico e collega di tirocinio. A tutti loro auguro il meglio ed il raggiungimento dei loro traguardi.

Ringrazio Emma per il supporto di ogni giorno e per essermi stata sempre accanto nel raggiungere qualsiasi obiettivo.

Ringrazio mio padre Alfredo, mia madre Eleonora, mia sorella Stella, i miei nonni Katia, Laura e Renato e tutti i miei familiari per i sacrifici fatti e per aver sempre creduto in me.

Ancona, Ottobre 2026

Filippo Campenni