
# Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei sistemi industriali

*Tesi Magistrale in Ingegneria Meccatronica*


# Sviluppo di un sistema di visione 3D per il bin picking applicato alla raccolta differenziata


*Relatore*

Chiar.mo Prof. Giovanni Boschetti

*Laureando*

Riccardo De Bortoli

Università degli Studi di Padova

*Numero di matricola*

2015551



*Anno accademico*

2022-2023

A tutti coloro che mi hanno accompagnato in questo percorso.

# Abstract

Lo scopo di questa tesi è quello di definire un sistema di riconoscimento oggetti 3D dedicato al bin picking. In commercio sono presenti numerose proposte di sistemi di visione 3D con software integrato, questo progetto prevede di svincolarsi da queste sviluppando un software dedicato, capace di adattarsi alle diverse implementazioni e applicazioni. Il sistema si compone di una camera 3D per le scansioni tridimensionali dello spazio di lavoro, un software che gestisce il riconoscimento degli oggetti tramite una rete neurale, logiche per la determinazione della posizione e l'orientazione degli oggetti e un robot antropomorfo che sfrutta tali informazioni per il bin picking degli oggetti. L'applicazione considerata è quella della raccolta differenziata: si è ipotizzata una possibile realizzazione di questo sistema nelle sedi universitarie, nelle mense ospedaliere o in altri ambienti nei quali è possibile definire un certo set di prodotti di scarto noti e ben caratterizzabili.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# 1 Introduzione

## 1.1 Problematica della gestione dei rifiuti

Svolgere un'attenta divisione dei rifiuti significa compiere il primo passo per il riutilizzo di una materia prima. Separando correttamente i materiali infatti, si contribuisce in maniera significativa al loro riciclo. Gran parte dei prodotti di consumo possono essere riqualificati per trovare un nuovo utilizzo se vengono opportunamente riciclati.

Svolgendo una corretta raccolta differenziata è possibile riscontrare degli enormi vantaggi a livello di impatto ambientale: in generale è garantita una minore produzione di emissioni inquinanti riducendo in maniera significativa l'inquinamento.

In questa trattazione l'attenzione è rivolta alla gestione dei rifiuti solidi urbani. La definizione ufficiale del termine *rifiuti urbani* è espressa dall'ente statale Lapam, come riportato in [5]. Nell'articolo si evince come a questa categoria appartengano tutti quei rifiuti indifferenziati e differenziati (compresi carta, cartone, vetro, metalli, plastica, rifiuti organici, legno, imballaggi, rifiuti relativi ad apparecchiature elettroniche ed elettrodomestici) originati anche da utenze non domestiche. Rientrano quindi in questa categoria anche materiali derivanti da altre fonti, ad esempio rifiuti provenienti dalle strade o da una qualsiasi area pubblica che risultano simili per composizione e volume ai prodotti di scarto domestici. Non sono invece inclusi i rifiuti relativi all'industria, all'agricoltura e quei materiali di scarto derivanti dall'edilizia e dalla demolizione di edifici.

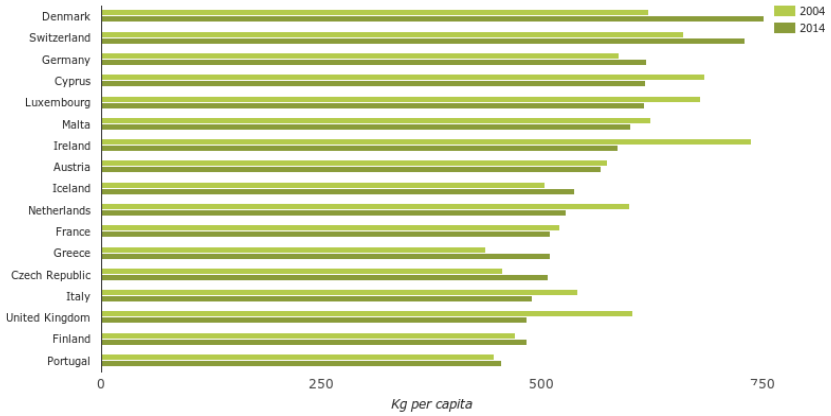



**Figura 1.1:** Produzione di rifiuti pro-capite in alcuni paesi dell'Unione Europea

Riconoscere e classificare correttamente i vari tipi di rifiuti è fondamentale per riuscire a svolgerne una prima divisione, permettendo così un'efficiente raccolta che differenzia i diversi materiali che possono poi essere riutilizzati.

La popolazione globale negli ultimi decenni è in costante crescita, si deve tenere conto che con essa aumentano anche i consumi e la produzione di rifiuti. Nell'anno 2016 globalmente sono stati generati 2.01 bilioni di tonnellate di rifiuti. Si stima che nel 2030 vi sia una produzione di 2.59 bilioni di tonnellate.

Una corretta gestione dei prodotti e dei rifiuti urbani durante il consumo permette di ridurre enormemente l'impatto che questi riportano a livello ambientale. Negli ultimi decenni in Europa, le politiche statali hanno focalizzato sempre più l'attenzione ai metodi di smaltimento e riciclaggio dei rifiuti urbani. Nonostante i rifiuti urbani rappresentino solo circa il 10% del totale dei rifiuti prodotti nell' Unione Europea, si sottolinea come i Paesi che hanno sviluppato sistemi efficienti per la gestione dei materiali di scarto urbano generalmente ottengano risultati migliori nella gestione complessiva dei rifiuti.

### 1.1.1 Gestione dei rifiuti urbani nell'Unione Europea

Vengono di seguito riportati i risultati di uno studio svolto dall'Agenzia Europea per l'Ambiente [6] che contiene un'analisi della gestione dei rifiuti solidi urbani svolta su tutti e 28 gli Stati Membri dell'Unione Europea e su Islanda, Norvegia, Svizzera e Turchia.

In certi casi la comparabilità dei dati tra i diversi Stati risulta essere limitata in quanto ci possono essere delle differenze nei tipi di rifiuti urbani comunicati. Questo deriva dalla diversa

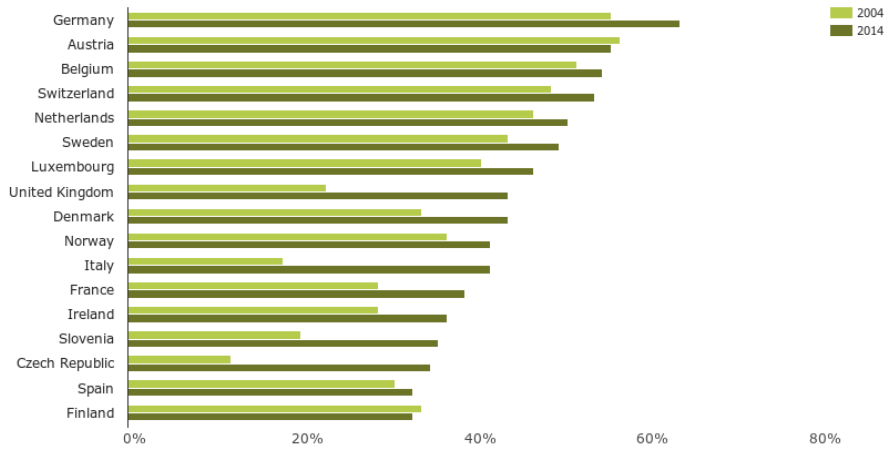



**Figura 1.2:** Percentuale di rifiuti riciclati sul totale dei rifiuti prodotti in alcuni paesi dell'Unione Europea

definizione di rifiuti urbani adottata da Paese a Paese: in alcune situazioni i dati includono rifiuti simili provenienti da attività commerciali e uffici.

I dati relativi allo studio [6] sono raccolti in Figura 1.1 e Figura 1.2 nelle quali si osservano rispettivamente le quantità di rifiuti prodotti annualmente da un singolo cittadino e le percentuali di materiali riciclati in alcuni degli Stati coinvolti nello studio.

Il concetto di riciclaggio e riutilizzo dei materiali è di primaria importanza per diminuire la grave situazione che affligge i sistemi di smaltimento e di raccolta nelle discariche: è fondamentale trovare un'alternativa allo scarico e all'abbandono dei rifiuti in luoghi in cui questi vengono accumulati. È di primaria importanza dare nuova vita ai materiali di scarto. In quest'ottica non solo si sfruttano al meglio le risorse di materia prima utilizzate, ma migliora la condizione ambientale alleggerendo quelli che sono gli sprechi derivati dallo smaltimento dei rifiuti.

L'analisi presentata in [6] studia e confronta la produzione di rifiuti urbani prodotti nell'anno 2004 e 2014. Mediamente nell'arco di 10 anni la produzione totale di materiali di scarto è diminuita del 3% in termini assoluti, mentre la produzione pro capite ha riscontrato una diminuzione media del 7%. Va sottolineato però come non vi sia stata una tendenza uniforme tra i diversi Paesi: negativo è il dato che registra un aumento di produzione in 16 Stati sui 32 considerati.

Analizzando un po' più in dettaglio quanto riportato, in Figura 1.1 vengono indicati i chilogrammi di rifiuti prodotti dal singolo cittadino: si può notare come Danimarca, Svizzera e Germania risultino tra i Paesi che maggiormente incidono nella produzione di materiali di scarto. I paesi caratterizzati da una più bassa produzione di rifiuti invece sono Romania, Polonia e Serbia. Questo trend riflette come i Paesi maggiormente sviluppati e ricchi tendano a generare

maggiori quantità di materiali di scarto.

Avvicinando lo sguardo alla situazione attuale, sempre parlando di quantità di rifiuti prodotti da un singolo abitante, la valutazione fatta dall'Ufficio Europeo di Statistica [7] registra un incremento medio annuo di circa 4 chilogrammi a partire dal 2014 fino ad oggi. Nel 2020 ogni cittadino europeo ha prodotto in media 505 chilogrammi di rifiuti, per un totale di 225,7 milioni di tonnellate di rifiuti urbani prodotti in un anno.

Contestualizzando questi numeri, dal 2005 la produzione di rifiuti urbani a livello europeo è aumentata del 27,7%. Le quantità misurate risultano molto variabili da Stato a Stato. Secondo questo studio [7], inoltre, i maggiori produttori di rifiuti urbani del 2020 sono stati Danimarca e Lussemburgo, rispettivamente con 845 e 790 chilogrammi annui pro capite. Ricercando invece gli Stati che hanno prodotto meno rifiuti, nel decennio precedente troviamo Bulgaria, Ungheria, Belgio, Spagna e Polonia.

Come anticipato in precedenza, anche la situazione attuale conferma come le differenze a livello di reddito, di abitudini di consumo e di stile di vita comportino una diversa gestione della raccolta e del trattamento degli scarti.

Espresse le quantità di rifiuti prodotti nei vari Paesi, è da sottolineare anche il tasso di riciclaggio dei materiali di scarto. Questo parametro è calcolato come la percentuale di rifiuti urbani che vengono riciclati rispetto al totale dei rifiuti generati. Esprime di conseguenza la cura e l'attenzione che viene rivolta alla gestione dei rifiuti, nonchè la qualità del processo di raccolta e trattamenti successivi dopo il consumo: un'attenta politica di raccolta differenziata genera un alto tasso di riciclaggio.

Negli ultimi decenni sono state di grande successo le politiche ambientali adottate in Europa. Se nel 2004 si misurava un tasso di riciclaggio dei rifiuti urbani del 23%, nel 2014 la percentuale media si è alzata al 33%. Questa miglioria deriva dalla maggiore attenzione data al riciclaggio, al compostaggio e allo smaltimento dei rifiuti organici e inorganici. In questo decennio solamente Germania, Austria, Belgio, Svizzera, Paesi Bassi e Svezia hanno riciclato almeno la metà dei rifiuti prodotti. Polonia, Regno Unito e Italia hanno registrato i più alti tassi di riciclaggio tra il 2004 e il 2014. Nella maggior parte degli Stati invece, le percentuali sono migliorate di poco, in altri addirittura risultano leggermente diminuite.

A rigor di logica, ad un aumento dei tassi di riciclaggio corrisponde un calo dei tassi di smaltimento dei rifiuti in discarica. I materiali che nei decenni scorsi venivano convogliati in centri di accumulo vengono ad oggi suddivisi tra impianti di riciclaggio e inceneritori. Alla diminuzione significativa del tasso di utilizzo delle discariche corrisponde quindi una combinazione

| Stato | Rifiuti Riciclati | Rifiuti in Discarica |
|---|---|---|
| Germania | 64% | 1% |
| Austria | 56% | 4% |
| Italia | 42% | 31% |

**Tabella 1.1:** Rifiuti urbani riciclati, [EEA]

di un aumento dei tassi di riciclaggio e di incenerimento. Anche in questo contesto tuttavia, esistono molte differenze da Stato a Stato: in Austria, Danimarca, Germania nessun rifiuto viene conferito in discarica mentre in altri Paesi come Croazia e Grecia più di tre quarti dei rifiuti urbani vengono smaltiti in discarica. In generale, se si vuole estrapolare un valore che rappresenti la media dei 32 Paesi sotto esame, all'anno ogni abitante è responsabile dell'invio di 270 chilogrammi di rifiuti in discariche o inceneritori.

### 1.1.2 Panoramica degli strumenti politici

Da diversi decenni i vari Paesi adottano politiche per convogliare i rifiuti agli impianti di riciclaggio invece che nelle discariche. In ottica generale gli Stati impongono tassazioni maggiori per coloro che vogliono usufruire dello scarico dei materiali nelle discariche. Uno degli strumenti più utilizzati è il *pay-as-you-throw* (paga quanto butti): si impongono tariffe basate sul peso o sul volume dei rifiuti portati in discarica in modo da incentivare economicamente le famiglie a riciclare i propri rifiuti. L'utilizzo di metodi come questo si riflette in modo diretto sul tasso di riciclaggio, permettendone un miglioramento significativo nel corso degli anni.

I fattori che contribuiscono ad ottenere tassi di riciclaggio elevati sono ad esempio il livello di ricchezza e la consapevolezza ambientale del cittadino. In tabella 1.1 sono riportati alcuni dati ricavati delle analisi svolte dalla Commissione Europea per l'Ambiente [6]. Nonostante il trend positivo, le percentuali ottenute risultano essere davvero basse se si contestualizza la problematica. I dati forniti dallo studio riportano come dato principale il tasso di riciclaggio. Avere valori del tasso di riciclaggio sotto il 50% significa che, se un ipotetico consumatore produce due rifiuti, di questi due rifiuti uno e una frazione del secondo non vengono riciclati ma vengono indirizzati verso una discarica dove vengono ammassati e accatastati aumentando prima di tutto il rischio ambientale e diventando puramente un costo per la società che dovrà prevederne lo smaltimento. Ritornando quindi ai dati riportati in tabella le percentuali presentate possono sembrare anche accettabili, ma se contestualizzate non presentano delle situazioni molto rassicuranti.

In Europa nell'arco di un anno la quantità di materiali di scarto che vengono riciclati è attorno ai 67 milioni di tonnellate (corrispondenti a 151 chilogrammi a persona): questo dato è anno per anno in miglioramento in quanto aumenta la percentuale di rifiuti che vengono riciclati sul totale dei rifiuti prodotti.

Nonostante l'aumento della produzione di rifiuti, in Europa la quantità totale di rifiuti urbani indirizzati alle discariche è diminuita. Sul totale dei rifiuti prodotti, a partire dal 1995 si registra una diminuzione della percentuale di rifiuti indirizzati alle discariche con un calo medio annuo del 4%, partendo dagli anni novanta con un 61% fino ad arrivare al 23% nel 2020.

A livello europeo si era fissata una soglia del 50% di riciclaggio dei rifiuti urbani entro il 2020. Questo obiettivo è stato raggiunto solo da pochi Stati membri. È fondamentale che alcuni Paesi intensifichino gli sforzi per poter garantire livelli di riciclaggio coerenti con gli standard vigenti.

È altrettanto importante che in tutti i contesti sociali ci sia un'attenzione sempre più marcata nella gestione dei materiali di consumo, partendo dai produttori, ai quali è richiesto di adottare scelte progettuali il più sostenibili possibile tenendo presente l'impatto che queste hanno sull'ambiente. Per tutti i rifiuti che si generano è necessaria una gestione attenta e curata, così da sfruttare appieno le materie prime ed evitare inutili sprechi favorendo il riciclo.

## 1.2 Computer Vision applicata alla gestione dei rifiuti

I rifiuti solidi generati possono essere classificati in tre diverse categorie. A seconda della loro origine si distinguono in:

- residui residenziali e municipali (RM, Residental and Municipal) derivanti dalle singole abitazioni comprendenti tutti i rifiuti generati dalla routine quotidiana dei cittadini;
- residui industriali e commerciali (ICI, Industrial Commercial and Institutional) prodotti in ambito industriale e in tutte le dinamiche economiche-produttive come ad esempio scarti di lavorazione;
- residui del settore edile (C&D, Construction and De- molition) generati dalla demolizione di fabbricati e dalle attività di costruzione di infrastrutture.

La ricerca scientifica si è concentrata maggiormente nello studio di metodi e algoritmi per la raccolta differenziata in ambito domestico, mettendo in secondo piano gli altri due settori. Sono stati sviluppati dei sistemi per la raccolta differenziata capaci di isolare i materiali inorganici

dal resto dei rifiuti tramite l'utilizzo di sistemi di visione e bracci robotici. Molta attenzione è stata rivolta alla separazione della plastica tramite servizi di computer vision.

Alcuni ricercatori si sono concentrati sull'uso della computer vision per distinguere tra rifiuti domestici quali vetro, alluminio, carta, plastica e cartone ([8], [9], [10]).

La computer vision comprende tutti i metodi computazionali che permettono ai sistemi di ricavare informazioni da immagini digitali, video e altri input visivi. Si cerca di applicarla nella raccolta differenziata dei rifiuti tra i diversi settori dell'economia nazionale quali l'industria nucleare, l'industria automobilistica, l'agricoltura e la produzione alimentare. Il riconoscimento degli oggetti che sfrutta i sistemi di visione è considerato un aspetto fondamentale e indispensabile per tali modelli robotici. Nelle realtà industriali è economicamente vantaggioso effettuare il riciclaggio delle materie prime. In questi termini si progettano dei sistemi di telecamere capaci di misurare i parametri di forma dei pezzi di scarto della produzione o dei prodotti smessi per poterli separare e classificare per un efficiente riciclo e riutilizzo. La stessa dinamica avviene in ambito agricolo e nel settore alimentare: vengono utilizzati sistemi di visione per la gestione e l'ottimizzazione degli scarti di materiale.

Prendendo poi in considerazione i rifiuti derivanti dal settore edile, questi occupano una componente importante dei materiali di scarto prodotti: in alcune città rappresentano il 40% del totale di tutti i rifiuti. Per questo sono stati progettati sistemi robotici in grado di trattare diversi materiali quali legno, mattoni, gomma, roccia e cemento. I modelli addestrati hanno raggiunto prestazioni elevate: sono state utilizzate reti neurali convoluzionali (definite nei paragrafi successivi) capaci di determinare la composizione dei materiali di riciclo con una velocità quasi in real-time. Nonostante gli studi di ricerca, nella pratica le tecnologie di raccolta intelligente in ambito edilizio sono ancora molto limitate. È da tenere in considerazione che un ipotetico efficientamento del riciclaggio di questi scarti risulterebbe di notevole importanza poichè in alcune città e regioni i rifiuti derivanti da questo settore hanno dei volumi percentuali davvero rilevanti.

In generale, per lo smistamento dei rifiuti nei vari settori, sempre più frequentemente si utilizzano tecnologie intelligenti per seguire quella che è la volontà di automatizzare il processo di suddivisione e di manipolazione dei rifiuti. Ove possibile, le proposte vengono applicate tramite implementazioni pratiche, ma, per incompatibilità economica o inefficienze, in alcuni contesti ci si ferma ad un'idea e ad un concetto puramente teorico.

Il progetto presentato in questo elaborato cerca di trattare il tema della raccolta differenziata cercando di favorire la loro precisa suddivisione in un'applicazione molto specifica: si prevede di

affiancare un sistema di visione artificiale per l'acquisizione delle informazioni del campo di lavoro ad un braccio robotico a sette gradi di libertà capace di raggiungere gli oggetti e suddividerli in base alle logiche imposte dalle normative ambientali.

Per contestualizzare tale progetto, è bene ripercorrere il percorso evolutivo della computer vision e comprendere come questa si sia introdotta e insediata nel settore della gestione e della suddivisione dei rifiuti solidi urbani. Viene proposta una breve revisione delle proposte del passato, si analizza lo stato dell'arte sottolineando quelli che sono i limiti delle prestazioni ad oggi ottenibili e si propongono dei possibili spunti di miglioramento.

### 1.2.1 Creazione di un modello standardizzato per la progettazione dei sistemi

La composizione dei rifiuti solidi varia molto in base al grado di sviluppo e alle diverse dinamiche sociali che si prendono in considerazione. Ai fini di creare un modello standardizzato per la progettazione di sistemi, è importante definire un ciclo caratteristico di gestione dei rifiuti: partendo dalla loro generazione e raccolta, si passa poi alla fase di classificazione grazie alla quale si possono suddividere gli scarti in base alla loro composizione, definendo i metodi di trattamento e smaltimento caratteristici per ciascuna tipologia di rifiuto.

Dato che non esiste una definizione universale ed esplicita, il termine *raccolta differenziata* può fare riferimento a diversi processi di raggruppamento dei rifiuti. I materiali di scarto possono essere raccolti e suddivisi quando sono generati alla fonte (ad esempio nei cantieri o nelle case) o in alternativa possono essere accumulati tutti negli stessi contenitori e indirizzati a dei centri specializzati per la suddivisione.

Quando i rifiuti sono separati alla fonte è l'utente stesso ad essere responsabile della loro suddivisione in categorie in base al tipo. Tuttavia, con la crescente complessità nel riuscire a svolgere una corretta classificazione dei materiali, diventa sempre più difficile suddividere i rifiuti in maniera corretta. Questa problematica affligge sia i cittadini che le autorità di gestione e regolamentazione.

### 1.2.2 Prime implementazioni della computer vision nella raccolta differenziata

L'utilizzo della tecnologia e in particolare della CV può potenzialmente aiutare lo smistamento dei rifiuti: con la selezione di opportuni dati è possibile addestrare un sistema di visione in grado di identificare i vari materiali raccolti. Una possibile applicazione pratica può essere quella di implementare il software su piattaforme robotiche per consentire la raccolta automatica dei rifiuti su ambienti controllati. Il focus di questo progetto si rifà proprio a questa specifica situazione e applicazione.

#### Inquadramento generale: machine learning e concetti di computer vision

Nel contesto in cui lo smaltimento dei rifiuti non sia svolto direttamente dal consumatore, ma affidato ai sistemi centralizzati di smistamento, è previsto l'impiego di strumenti quali nastro trasportatore, macchinari e robot per permetterne la selezione e distinzione ai fini di riciclaggio. Possono essere adottate due tipologie di approccio, ovvero l'ordinamento diretto e quello indiretto.

Lo smistamento diretto separa i materiali di scarto direttamente applicando forze come la gravità, la forza magnetica o il prelievo manuale. Lo smistamento indiretto, invece, utilizza dei sensori per la rilevazione e l'individuazione degli oggetti: vengono utilizzate ad esempio termocamere, sensori ottici o sensori induttivi per il riconoscimento delle specifiche tipologie di materiali di scarto e successivamente viene eseguita la selezione e il prelievo dei materiali tramite robot.

Le grandi potenzialità della computer vision per la raccolta differenziata indiretta sono ampiamente sfruttate da molti anni: nuove tecnologie e logiche permettono uno sviluppo continuo dei metodi di riconoscimento e selezione. Nel 1997 è stato presentato un sistema robotico con visione stereoscopica per rilevare e separare oggetti di carta ([11]). Successivamente nel 2000 è stata proposta una soluzione di smistamento dei rifiuti in diverse classi, quali plastica, vetro, cartone, prevedendo l'utilizzo di un nastro trasportatore e di opportuni dispositivi ottici ([12]).

Rispetto ad altre tecnologie di rilevamento visivo come i raggi X o immagini iperspettrali, i sensori ottici risultano essere di facile manutenzione e versatili per un'ampia gamma di rifiuti. La CV inoltre richiede solamente la semplice installazione di telecamere RGB e sfrutta la potenza degli algoritmi per il rilevamento, quindi, anche in termini di investimento per l'acquisto dei dispositivi hardware è preferibile ad altre soluzioni.

Nonostante le ottime prospettive, il ruolo della CV nello smistamento dei rifiuti è stato limitato e marginale per lungo tempo. Lo sviluppo lento è attribuibile agli enormi sforzi manuali richiesti per la realizzazione, alla robustezza del sistema relativamente bassa nella fase iniziale, alla necessità di avere set di dati completi e alla complessità computazionale che ne deriva dalla loro elaborazione.

Un sistema di riconoscimento oggetti basato sul machine learning funziona utilizzando algoritmi di apprendimento automatico per identificare gli oggetti in un'immagine. In generale, il processo di individuazione avviene in tre fasi principali: pre-elaborazione, estrazione delle caratteristiche e classificazione.

All'inizio del processo, prima ancora che l'immagine possa essere analizzata, è necessario effettuare una serie di operazioni preliminari per migliorare la qualità dell'immagine. Queste operazioni possono includere la riduzione o la rimozione del rumore di disturbo, la correzione del colore, la segmentazione dell'immagine e altre tecniche di elaborazione delle immagini.

Succede poi la fase di estrazione delle caratteristiche: il sistema di riconoscimento degli oggetti ricava le caratteristiche rilevanti per l'identificazione degli oggetti. Queste caratteristiche possono includere la forma, la texture, il colore e altre informazioni.

Il processo di classificazione, infine, permette l'effettivo riconoscimento degli oggetti. Tramite l'uso di un algoritmo di apprendimento automatico. Per classificare l'oggetto presente nell'immagine in base alle caratteristiche estratte, il sistema di riconoscimento viene addestrato su un set di immagini di esempio con etichette conosciute (*Training set*), in modo da poter poi, una volta allenato, generalizzare e classificare oggetti non visti in precedenza.

Un importante sviluppo di questa tecnologia basata sull'intelligenza artificiale si è potuto verificare con l'avvento del deep learning: un primo progetto innovativo è stato presentato nel 2012 con il sistema basato su rete neurale ImageNet ([13]).

Il deep learning è un sottoinsieme dell'apprendimento automatico machine learning che utilizza reti neurali artificiali (*deep neural networks*) per l'elaborazione di informazioni complesse. Nel contesto della computer vision, il deep learning consente di analizzare immagini e video in modo efficiente e accurato con una velocità e una precisione migliore rispetto a semplici sistemi sprovvisti di reti neurali.

Le reti neurali sono composte da molteplici strati di nodi che elaborano le informazioni in modo gerarchico.

Un nodo, definito anche come neurone artificiale, è un'unità computazionale che elabora le informazioni in ingresso e produce un'uscita. I nodi sono organizzati in modo gerarchico a strati e le informazioni che questi producono o acquisiscono sono accoppiate a dei pesi che determinano l'importanza delle informazioni trasmesse. L'output di un nodo è determinato dalla somma ponderata degli input pesati secondo i riferimenti gerarchici dei dati che elabora. I nodi possono essere di diversi tipi, a seconda della funzione che svolgono all'interno della rete. Si distinguono ad esempio nodi di input che ricevono i dati in ingresso e li trasmettono ai nodi successivi, nodi di output che producono i risultati dell'elaborazione della rete e nodi nascosti che sono posizionati tra i nodi di input e quelli di output e svolgono una funzione di elaborazione intermedia.

Ogni strato riceve input dai nodi del livello precedente e passa le informazioni elaborate ai nodi del livello successivo, fino a raggiungere lo strato di output. Il processo di apprendimento avviene mediante l'aggiustamento dei pesi delle connessioni tra i nodi, in modo da minimizzare l'errore di predizione durante la fase di addestramento. Una volta trovati i pesi ottimi, o comunque considerati buoni per una data applicazione, il sistema è capace di analizzare e decifrare anche immagini mai viste prima, riconoscendo quegli oggetti contenuti nell'elenco delle etichette durante la fase di training.

Nella computer vision, le reti neurali possono essere addestrate su grandi dataset di immagini per riconoscere e classificare gli oggetti presenti nelle immagini stesse. In particolare, le reti neurali convoluzionali (*CNN, Convolutional Neural Network*) sono di fondamentale rilevanza poiché in grado di catturare le caratteristiche delle immagini in modo automatico.

Il deep learning nella computer vision ha permesso di raggiungere prestazioni di riconoscimento oggetti molto elevate, superando le prestazioni dei metodi tradizionali di elaborazione delle immagini.

Le tecniche di deep learning possono superare i limiti degli algoritmi di visione tradizionali attraverso due principali innovazioni. In primo luogo, la potenza del deep learning nell'estrazione delle informazioni permette di evitare la creazione manuale dei dataset, poiché i dati e le caratteristiche dei diversi rifiuti possono essere apprese automaticamente dall'infrastruttura della rete neurale. In secondo luogo, la robustezza degli algoritmi di classificazione può essere migliorata alimentando il modello con un'enorme quantità di dati visivi acquisiti in ambienti diversi e in constante aggiornamento durante il funzionamento del sistema.

Grazie all'avvento del deep learning, dal 2016 in poi, è stato possibile portare a termine numerosi studi nei quali sono state definite diverse applicazioni gestite tramite intelligenza artificiale per lo smaltimento dei rifiuti.

Si confrontano ora diversi algoritmi utilizzati in questo ambito utilizzando come principale metrica di valutazione l'accuratezza del riconoscimento e considerando solo in modo marginale il concetto di tempo computazionale. Questo è motivato dal fatto che il tempo ciclo di pick-and-place che si andrà a comandare risulta essere di gran lunga più ampio rispetto al tempo necessario per lo svolgimento dei calcoli previsti dall'elaborazione dei dati relativi alle misurazioni ottiche.

### Algoritmi tradizionali

I primi algoritmi utilizzati per il riconoscimento oggetti nel settore del riciclaggio di rifiuti erano basati su modelli di machine learning: si definiscono delle strutture semplici caratterizzate da una prima fase di estrapolazione delle informazioni e dei dettagli necessari al sistema per il successivo riconoscimento degli oggetti di scarto puramente manuale. In queste infrastrutture l'utente era responsabile della creazione dei set di immagini dei rifiuti da fornire al sistema nella fase di training e questo costituiva un aspetto gravemente oneroso.

I metodi tradizionali utilizzati sono basati sulla ricerca di schemi comuni e pattern all'interno delle immagini.

Un primo metodo, chiamato *Linear Discriminant Analysis*, prevede di ricercare combinazioni lineari di caratteristiche relative a una o a un'altra classe di oggetti da identificare. Una volta che si riconoscono delle macro-combinazioni di informazioni, si restringe la dimensione dell'immagine concentrandosi sulla regione in cui si prevede di trovare il soggetto ricercato. Questo fino ad arrivare a sottospazi così piccoli da raggiungere e riconoscere l'oggetto in esame.

Un altro esempio di algoritmo non parametrico tradizionale è il *Nearest Neighbor*. Dal nome si evince la logica: dato un insieme di punti campione $S$ appartenenti a classi diverse ma ben definite, dato un punto di classe sconosciuta $q$, l'algoritmo calcola iterativamente la distanza tra il punto e tutti i punti appartenenti ai diversi campioni per cui il punto appartenente ad $S$ più vicino al punto $q$ definisce la classe di appartenenza incognita del punto.

Altre logiche utilizzate fanno riferimento a strutture ad albero (*Decision Tree*), oppure a modelli che prendono in considerazione la forma, il colore e altre proprietà degli oggetti che si vuole riconoscere (*Baysian Network*).

Un grande passo nello sviluppo dei modelli è stato compiuto con l'introduzione delle reti neurali artificiali. Queste permettono di effettuare il riconoscimento degli oggetti tramite una struttura che imita le reti neurali biologiche (*Artificial Neural Network*: [14], [15]). Se

inizialmente gli algoritmi prevedevano l'utilizzo di strutture semplici in grado di apprendere solamente schemi lineari, i modelli nuovi portano con se l'innovazione di reti multi-layer capaci di distinguere dati più complessi. Questi algoritmi sono stati utilizzati per la classificazione di vari materiali di scarto, dalla plastica ai rifiuti metallici. L'accuratezza che si riesce a garantire con questa tecnologia è superiore al 95%.

### Algoritmi Deep Learning

Il deep learning si distingue dai metodi tradizionali perchè sfrutta la potenza dei big data durante la fase di training della rete neurale utilizzata. Più precisamente non sono necessarie fasi di pre-elaborazione o di estrazione delle informazioni, perchè l'addestramento del modello avviene in una singola elaborazione. In una tipica architettura deep learning le immagini originali vengono inviate direttamente alla rete neurale costituita da nodi disposti in più strati convoluzionali di classificazione completamente connessi tra loro. Attraverso questi nodi pertanto le caratteristiche nascoste delle immagini possono essere automaticamente apprese ed estratte per poi prevedere la classe di appartenenza.

Questo caratteristico meccanismo di allenamento, basato appunto su un'unica fase, è definito processo *end-to-end*. Sfruttando questo concetto è possibile evitare il lento processo di creazione manuale dei dataset che la rete richiede come input. Una conseguenza diretta è quella di permettere il notevole ampliamento della gamma di prodotti riconoscibili aumentando così la precisione e l'accuratezza del sistema.

Con un set di dati che comprende un'ampia collezione di campioni di rifiuti, i modelli risultanti tendono ad essere più robusti di quelli addestrati con algoritmi tradizionali. La rete neurale convoluzionale (CNN) è un algoritmo di deep learning all'avanguardia che è stato applicato con successo in un ampio range di attività correlate alla computer vision [16].

A partire dal 2015 vengono progettate reti neurali come la *ResNet* o la *AlexNet* basate su algoritmi che sfruttano l' elaborazione del contenuto dell'immagine utilizzando il concetto di finestra scorrevole: si evidenziano sotto sezioni del frame e si filtra il contenuto di queste porzioni analizzandone il contenuto. La finestra viene fatta scorrere per tutto il file di input e grazie alla creazione di un vettore è possibile determinare la probabilità che nell'immagine sorgente sia presente o meno un determinato materiale di scarto. Non si può quindi parlare di un vero e proprio algoritmo di riconoscimento oggetti in quanto con questi modelli non è possibile determinare posizioni, orientazioni o altri dati di alto livello. Si tratta piuttosto di un codice

che fornisce in output informazioni puramente riguardanti la presenza di oggetti all'interno dei frame, spoglie da ogni altro tipo di dettaglio.

In generale, tutti i modelli per il riconoscimento oggetti basati sul deep learning richiedono notevoli potenze di calcolo. L'addestramento dei modelli, inoltre, si basa su un'enorme quantità di dati difficili da ottenere e raccogliere sul campo. Per questo motivo è possibile che algoritmi di deep learning vengano scartati per implementazioni in ambito industriale.

Nell'applicazione relativa alla raccolta differenziata è necessaria una vera e propria rilevazione degli oggetti e delle loro posizioni nello spazio.

L'algoritmo alla base del modello, una volta riconosciuta la classe di appartenenza dell'oggetto, deve essere in grado di definire l'area entro cui l'oggetto si trova: è quindi possibile l'individuazione di oggetti su appositi riquadri di delimitazione, definiti *bounding box* ([17]). Logiche come la segmentazione ad istanza (*instance segmentation*) vanno ancora oltre, permettono infatti di estrarre zone di pixel corrispondenti agli oggetti.

Una particolare classe di reti neurali convoluzionali è la R-CNN (*region-based convolutional neural network*). Questa categoria utilizza un algoritmo chiamato *selective search* che permette di definire delle regioni candidate, ovvero dei settori dell'immagine caratterizzati da una probabilità maggiore di ritrovare oggetti al loro interno. Dato che l'estrapolazione delle informazioni viene ripetuta molte volte per tutte le varie porzioni dell'immagine, la rete R-CNN non ha grandi prestazioni in termini di efficienza. Sono quindi stati sviluppati una serie di algoritmi basati sulla logica R-CNN che garantiscono processi di elaborazione accelerati (*Fast R-CNN*) così da ridurre i tempi di elaborazione. Questa versione *fast* trova applicazione in diversi settori della raccolta differenziata.

Esiste un'altra classe di reti chiamate rilevatori a singolo stadio, o *single-stage detectors*. Questi algoritmi prendono in carico il problema della rilevazione degli oggetti nelle immagini come un unico, singolo, problema di regressione. Tra gli esempi si citano YOLO (You Only Look Once, rete neurale scelta per il progetto), SSD (Single Shot Multibox Detector) e RetinaNet. Sono stati ideati dei sistemi basati sulle reti neurali convoluzionali con delle specifiche caratteristiche in base al dominio di rifiuti che si prevede di fornire ai diversi sistemi. Si sono sviluppate, ad esempio, delle architetture di apprendimento multi-task basate sulla simultanea classificazione e localizzazione dei rifiuti, arrivando a delle ottime e robuste accuratezze.

Nella pratica industriale, i rifiuti in ingresso agli impianti di smistamento si trovano in uno stato molto disordinato in cui diversi materiali sono sovrapposti l'uno sull'altro. Inoltre, es-

sendo prodotti di scarto, è possibile che questi arrivino nei centri di raccolta schiacciati, accartocciati, in pezzi o degradati. Una buona classificazione dei rifiuti deve tenere conto di queste variabili per l'individuazione delle posizioni specifiche dei vari oggetti registrati dai sistemi di visione.

In generale, gli algoritmi che compongono le reti di riconoscimento devono essere in grado non solo di individuare i diversi rifiuti classificandoli nelle varie categorie predeterminate (*waste recognition*), ma anche di localizzare e definire le posizioni dei prodotti riconosciuti all'interno delle immagini con bounding box ed etichette dedicate.

Per la fase di riconoscimento dei materiali, con lo sviluppo del deep learning, è possibile un' elaborazione diretta delle immagini senza che queste vengano precedentemente manipolate o modificate. Un'importante lavoro in questi termini è stato svolto nel progetto TrashNet ([16]): gli autori hanno semplificato la raccolta differenziata come problema di classificazione delle immagini tramite la ricerca di un singolo oggetto relativo ad una specifica classe di rifiuti. Il sistema è stato poi implementato con una rete neurale (WasNet) che incorpora un algoritmo che permette, una volta individuata l'area nella quale si riconosce l'oggetto, di prestare maggiore attenzione a quest'area sensibile così da massimizzare l'efficienza dei dati che si chiede di elaborare. Negli anni successivi sono state svolte ulteriori migliorie per quanto riguarda l'ottimizzazione dei parametri elaborati, fino ad arrivare con il lavoro di *Mao et al* ([18]) a raggiungere un' accuratezza del 99,60%.

Nonostante i notevoli progressi, le tecnologie del riconoscimento dei rifiuti ad oggi trovano delle applicazioni limitate in ambito industriale. In primo luogo questo è dovuto al fatto che il *waste recognition* può solo classificare gli oggetti riconosciuti in una delle categorie predefinite, non è adatto alla cernita automatizzata dei rifiuti con la robotica: geometrie e posizioni dei materiali di scarto devono essere molto precise per poter permettere il recupero tramite robot. In secondo luogo, il riconoscimento dei rifiuti richiede che i singoli rifiuti siano organizzati su uno sfondo relativamente semplice, aspetto che si allontana molto dalla maggior parte degli scenari di vita reale in cui i materiali di scarto si disperdono o addirittura si sovrappongono l'uno sull'altro.

Detto questo, le tecniche di separazione dei rifiuti introdotte dagli studi presentati dovrebbero essere prese in considerazione principalmente per la separazione alla fonte nella fase di raccolta dei rifiuti, dove ad esempio un utente all'interno della sua abitazione possa essere aiutato, tramite anche l'utilizzo di strumentazione robotica, a distinguere i diversi tipi di rifiuti.

Nelle applicazioni ingegneristiche pratiche dove queste tecnologie trovano impiego è abba-

stanza comune che più oggetti di scarto appaiano sulla stessa immagine. Pertanto, le operazioni di selezione (*waste detection*) si basano non solo sulle categorie identificate, ma anche sull'esatta geometria dei rifiuti per garantire informazioni precise per la determinazione delle effettive posizioni tridimensionali per la successiva presa o cernita tramite bracci robotici. Nel processo di selezione dei rifiuti, la maggior parte dei materiali di scarto viene gettata in modo casuale sul nastro trasportatore. La problematica è da superare prima ancora di iniziare il processo di individuazione dei rifiuti.

Le attuali tecniche consentono di addestrare i modelli per l'identificazione dei rifiuti direttamente in ambienti complessi dove è possibile trovare situazioni di rifiuti raggruppati. I precedenti sistemi erano capaci di riconoscere solamente un singolo oggetto, con le nuove tecnologie deep learning si sottolinea come si riesca a raggiungere delle ottime accuratezze e prestazioni nel riconoscimento degli oggetti appartenenti anche ad ammassi di spazzatura. Per fare questo si caratterizzano i rifiuti studiandone le geometrie e cercando di riconoscere, oltre che l'oggetto vero e proprio, anche le sue caratteristiche geometriche, come ad esempio la forma concava tipica del fondo di una bottiglia.

Recenti studi ([19]) hanno sottolineato come nei set di immagini forniti nella fase di training sia fondamentale riportare anche il contesto e le situazioni al contorno del rifiuto in modo da riconoscerlo anche all'interno di gruppi disordinati di materiali. Con questa logica, in base all'applicazione e al settore in cui si vanno ad implementare i sistemi, si creeranno diversi dataset di dati per allenare le reti neurali relegate al riconoscimento dei rifiuti.

### 1.2.3 Scenari Applicativi

Vi sono principalmente due scenari applicativi su cui si è concentrato lo studio dello smistamento dei rifiuti secondo le tecniche viste in precedenza. Il primo è la raccolta di rifiuti alla fonte, in prossimità quindi dell'abitazione o del servizio ultimo all'utente e il secondo è quello dislocato negli impianti di smaltimento.

#### Raccolta differenziata alla sorgente

Svolgere una corretta differenziazione dei rifiuti alla fonte permette di definire piani di raccolta efficienti per tutti gli step successivi del sistema di smaltimento e riciclo. In questi termini recentemente sono stati sviluppati dei sistemi robotici ed intelligenti capaci di aiutare il cittadino nella classificazione dei rifiuti prodotti in casa. Con questa logica, oltre che migliorare la qualità

e l'efficienza del riciclo, è possibile facilitare lo scambio di informazioni tra l'utente finale e le società di raccolta.

Un altro notevole braccio di ricerca riguarda lo sviluppo di bidoni intelligenti ovvero cassonetti capaci di utilizzare la computer vision per rilevare il livello di spazzatura contenuta al loro interno. Si ricavano quindi informazioni che possono essere utili ai dipartimenti e ai servizi di raccolta dei rifiuti. Alcuni progetti si spingono oltre, andando a separare automaticamente i rifiuti nei cassonetti stessi grazie all'integrazione della computer vision con la robotica. La maggior parte di questi sistemi si basa su tecnologie di machine learning tradizionali senza l'implementazione del deep learning.

#### Raccolta differenziata agli impianti di smaltimento

Confrontando le varie tecniche di individuazione e riconoscimento automatizzate presenti negli impianti di raccolta si può affermare che i sensori di visione ottici riescono a garantire una certa applicabilità in quanto possono raggiungere una precisione accettabile con un consumo di tempo minimo nel riconoscimento di una vasta gamma di prodotti e materiali. Lo sviluppo dell'intelligenza artificiale e del deep learning migliora significativamente la robustezza degli algoritmi ed espande l'applicabilità ad un'ampia gamma di materiali di scarto. Il concetto di rilevamento dei rifiuti rimane un problema per la raccolta differenziata automatizzata e richiede sforzi di ricerca e progettazione aggiuntivi per poter integrare la robotica nei sistemi di riconoscimento applicati all'industria.

### 1.2.4 Accessibilità dei dataset

Un altro aspetto fondamentale per i sistemi ottici per il riconoscimento degli oggetti è il grado di accessibilità delle informazioni e dei dataset per l'allenamento delle reti. I set di dati sui rifiuti possono essere classificati in pubblici, ai quali si può accedere liberamente, o privati, quando possono essere utilizzati solo da chi li crea. Negli studi di ricerca si può in generale affermare che i dataset utilizzati il più delle volte risultano essere un'esclusiva dei proprietari del progetto ai quali fanno riferimento. Con questa volontà di mantenere private le informazioni non è possibile stabilire una progressiva evoluzione in termini di accuratezza, in quanto l'uso di dataset privati rende difficile fornire marcatori unificati per un confronto significativo delle prestazioni.

Intraprendendo una condivisione dei dati raccolti invece è possibile avere a disposizione piattaforme dedicate e numerosi set già utilizzabili per il riconoscimento degli oggetti. Con l'aumen-

to delle pubblicazioni e dei lavori di ricerca il trend di precisione del set di immagini raccolte è capace di definire una crescita continua.

Nonostante le prestazioni significative raggiunte, il set di dati raccolti ad esempio in TrashNet ([16]) risulta essere troppo idealistico per consentire applicazioni pratiche: per l' allenamento della rete vengono sfruttate immagini caratterizzate da sfondi bianchi o monocromatici. Questa caratteristica si distacca dalla realtà delle possibili situazioni applicative. Si sottolinea infatti come sia necessario l'utilizzo di immagini raccolte in contesti di vita reale. In questi termini TACO (Trash Annotations in COntext, [19]) risulta essere un'iniziativa notevole. TACO è un dataset pubblico che comprende svariate immagini di rifiuti estrapolate da contesti naturali. Queste immagini sono elaborate e segmentate manualmente dai vari utenti che vi accedono. Al suo interno sono contenuti numerosi materiali di scarto suddivisi in 28 categorie (bottiglie, sacchetti in plastica, ecct.), suddivise a loro volta in base alle diverse classi di rifiuto (carta, plastica, ecct.). Per ciascun oggetto il dataset contiene circa 2500 immagini.

## 1.3 Prospettive e Sfide future

L'applicazione della computer vision per la raccolta differenziata sta guadagnando sempre maggiore attenzione. Vi è però la mancanza di alcuni dettagli e attenzioni per un'evoluzione dei sistemi attualmente presenti.

Innanzitutto, gli studi attuali mancano di set di dati di immagini proiettate e definite nel mondo reale, capaci di descrivere le svariate applicazioni industriali. In quest'ottica è fondamentale una condivisione dei set di dati da parte di chi si occupa di sviluppare modelli, così da permettere un avanzamento in termini di accuratezza.

Secondariamente, gli studi esistenti hanno semplificato gli obiettivi e le condizioni di lavoro degli algoritmi proposti: evitare la sovrapposizione degli oggetti o la lettura dei rifiuti uno ad uno consente di definire delle situazioni ideali e semplificate della realtà industriale. Semplificazioni troppo vincolanti possono rendere gli approcci sviluppati incompatibili con le strutture di smistamento effettive, dato che i rifiuti possono essere distribuiti in modo casuale in uno sfondo disordinato.

Nonostante la versatilità della computer vision nel distinguere un'ampia gamma di materiali, un ultimo limite è rappresentato dall' incapacità di caratterizzare le proprietà fisico-chimiche degli oggetti di scarto. I materiali con diverse proprietà fisico-chimiche possono presentare caratteristiche simili a livello visivo: il vetro, ad esempio, potrebbe sembrare simile ad un pezzo

di plastica trasparente, ma chiaramente si tratta di due materiali diversi. Il questi casi pertanto è difficile fare affidamento sui sistemi di visione.

Prospettive future possono prevedere la necessità di avere dataset di immagini orientate verso le esigenze pratiche dell'industria fornendo foto in contesti reali ben definiti. Altrettanto importante risulta la necessità di rendere pubblici i risultati e le informazioni raccolte dagli studi dell'intero settore, al fine di permettere una continua evoluzione in termini di accuratezza dei sistemi. L'obiettivo ultimo è la costruzione di un database centralizzato in cui le immagini di vari tipi di rifiuti e le loro annotazioni possano essere condivise. Questo attualmente risulta essere difficilmente raggiungibile in quanto incombono ancora questioni legali legate ai diritti d'autore e alla riservatezza. Risulta quindi necessaria la formulazione di un protocollo comune da utilizzare come guida per la gestione di queste immagini. Sarebbe interessante aggiungere anche campi dati aggiuntivi quali la geolocalizzazione e la provenienza dei rifiuti.

Si desidera concentrare gli studi verso l'aspetto pratico-applicativo della questione. In particolare, lo smistamento dei rifiuti nei nastri trasportatori richiede un'analisi molto più intensa di quella fatta per il riconoscimento dei materiali da un'immagine che raffigura rifiuti con sfondi monocromatici. In alcuni scenari, oltre che all'accuratezza del riconoscimento è richiesto anche il rispetto di rigide tempistiche. Ulteriormente si deve tener conto che in certi impianti di smaltimento dei rifiuti si può ricevere fino a qualche migliaia di tonnellate di prodotti al giorno e che la velocità di selezione influisce direttamente sulla produttività complessiva dell'impianto. È quindi fondamentale lo sviluppo di algoritmi accompagnati da un hardware con potenze di calcolo opportunamente dimensionate in relazione alle prestazioni finali desiderate.

In aggiunta è necessario pensare a sistemi di rilevamento degli oggetti che integrino sistemi di visione con altre classi di sensori, come ad esempio spettroscopi, misuratori di peso o sistemi induttivi, per consentire la determinazione delle proprietà fisico-chimiche dei materiali di scarto da classificare. La predisposizione di dati provenienti dalla misurazione delle diverse peculiarità dei materiali di scarto può aumentare l'accuratezza e migliorare la robustezza dei sistemi.

# 2

# Aspetti teorici degli strumenti utilizzati

## 2.1 Object detection nella computer vision

Quando si parla di sistemi di riconoscimento oggetti è importante chiarire i concetti di *image classification* e di *object localization*. Il primo termine fa riferimento ad un processo di visione artificiale in grado di classificare un'immagine in base al suo contenuto. Il secondo termine invece si rifà alla localizzazione degli oggetti contenuti nelle immagini e definisce invece la capacità di rilevazione della specifica posizione dell'oggetto ritrovato all'interno del frame. Il rilevamento degli oggetti, in gergo definito come *object detection*, fornisce gli strumenti per trovare le informazioni di posizione degli stessi in un' immagine e di disegnarne i bounding box. Un'altra esperienza è poi quella definita dal concetto di *image segmentation* che, oltre a definire la posizione, sottolinea con precisione anche i contorni degli oggetti rilevati. In figura 2.1 viene espressa una rappresentazione delle differenti tipologie appena definite.

Lo scopo ultimo di questo progetto è quello di implementare un' applicazione per il riconoscimento oggetti in ambito della raccolta differenziata. Si decide quindi di scartare gli algoritmi basati sulla segmentazione delle immagini. Questa scelta di esclusione è motivata dal fatto che, per i metodi che prevedono la segmentazione, lo sforzo computazionale per la creazione e l'elaborazione dei dataset di training risulta molto più articolato.

A livello progettuale inoltre alla rete neurale sarà richiesto come solo aspetto fondamentale il riconoscimento degli oggetti e la definizione della loro posizione. Per quanto riguarda la deter-

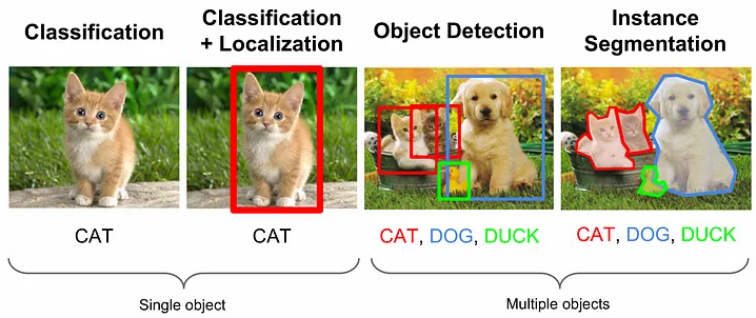



**Figura 2.1:** Metodi di riconoscimento oggetti

minazione dell' orientazione e dei dettagli geometrici degli oggetti ci si appoggia ad algoritmi e logiche implementate esternamente alla rete neurale. Questa scelta è stata presa per garantire una maggiore flessibilità e permettere di adattare il progetto a specifici prodotti da classificare e riconoscere.

Gli algoritmi per l'object detection possono essere classificati in due diversi gruppi:

- **Algoritmi basati sulla classificazione**: sono implementati in due fasi separate. Si svolge per prima una selezione delle regioni d'interesse (*Region of Interest, ROI*) andando ad ipotizzare delle zone al cui interno possono essere contenuti gli oggetti dell'immagine e scartando le zone considerate come non fondamentali. Successivamente si classificano le zone ROI attribuendone le etichette delle classi riconosciute, utilizzando reti neurali convoluzionali. La soluzione di questi algoritmi può risultare lenta in quanto la fase di riconoscimento delle regioni richiede un elevato sforzo computazionale. Uno dei principali algoritmi di questa classe è il *RetinaNet*.

- **Algoritmi basati sulla regressione**: prevedono di determinare le classi e i bounding box degli elementi riconosciuti in un' unica soluzione. A questa categoria appartengono gli algoritmi della famiglia YOLO (*You Only Look Once*) e SSD (*Single Shot multibox Detection*).

Nel progetto svolto si utilizza la versione 7 degli algoritmi YOLO. YOLO v7 è un algoritmo realtime di riconoscimento oggetti basato sul concetto di rete neurale convoluzionale che sfrutta il metodo di regressione: viene applicata una rete neurale per l'analisi dell'intera immagine per l'estrapolazione delle informazioni di posizione e classificazione degli oggetti.

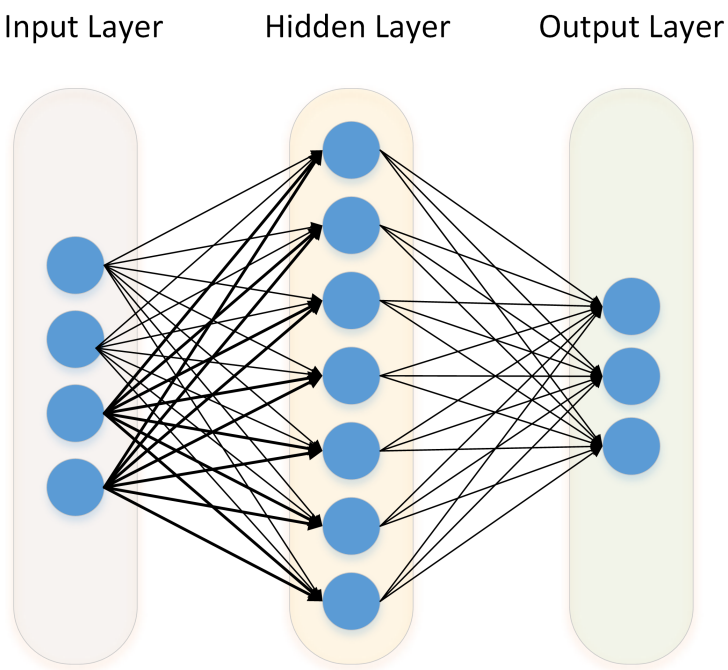



**Figura 2.2:** Schematizzazione semplificata dei layer in una rete neurale

## 2.2 Rete neurale YOLO v7

### 2.2.1 Introduzione alle reti neurali

Le reti neurali sono utilizzate in diversi sistemi di visione per compiti come la classificazione delle immagini, il rilevamento degli oggetti e il tracciamento dei movimenti. Una rete neurale è un modello computazionale il cui funzionamento prende ispirazione dal funzionamento del cervello umano: si compone di un insieme di nodi, considerati come unità logiche, interconnessi tra loro e capaci di processare informazioni in modo periferico e distribuito.

L'organizzazione logica che collega questi nodi, un cui esempio è presentato in figura 2.2, è suddivisa in strati (*layer*) che elaborano l'input in modo graduale producendo l'output finale. L'informazione in ingresso viene presentata al primo strato della rete, l'*Input Layer*, dove i nodi iniziano l' elaborazione dei dati e li trasmettono agli strati successivi, definiti come *Hidden Layers* o livelli nascosti, fino al raggiungimento dell'output finale, *Output Layer*. Ciascun layer calcola quindi il vettore di input per il livello successivo.

Questa struttura a livelli è definita per permettere un' elaborazione parallela e ramificata delle informazioni. Ad esempio, per una specifica applicazione di reti neurali utilizzate nel ricono-

scimento facciale, i primi livelli si occupano di riconoscere figure geometriche all'interno dell' immagine (come segmenti o ovali), i successivi livelli si occupano di interpretare il significato delle forme trovate identificando ad esempio se un certo ovale possa corrispondere alla forma di un occhio.

A seconda della loro architettura e dello scopo per cui vengono utilizzate esistono diverse tipologie di reti neurali. Per il riconoscimento oggetti, ambito d'interesse per il progetto che si sta esponendo, ci si concentra sulla trattazione delle reti neurali convoluzionali (*CNN, Convolutional Neural Network*). Queste sono progettate per l'elaborazione e il riconoscimento di pattern all' interno di immagini o di altre strutture dati sviluppate a griglia. La CNN è composta da una serie di strati di nodi, ognuno dei quali esegue operazioni matematiche su una porzione di input che riceve.

Nello specifico, i neuroni che costituiscono i vari layer delle CNN lavorano l'immagine in modo graduale e preciso:

- **Input Layer**: l'immagine viene caricata come input nel primo layer della rete.
- **Strati Convoluzionali**: questi strati eseguono la convoluzione dell'immagine applicando una serie di filtri ai pixel che compongono l'immagine. Ogni filtro scansiona una piccola parte dell' immagine alla volta, cercando di identificare delle caratteristiche, come ad esempio bordi, linee, forme geometriche, texture, ecct.. Tali filtri sono solitamente rappresentati da matrici di numeri adattati e ottimizzati durante la fase di allenamento della rete (per la definizione di allenamento della rete neurale si invia al paragrafo2.2.3).
- **Strati di pooling**: questi strati si occupano di ridurre la dimensione dell' immagine, riducendo il numero di pixel e mantenendo al contempo le caratteristiche essenziali dell' immagine. Questo passaggio rende l'elaborazione più veloce ed efficiente.
- **Strati completamente connessi**: questi strati prendono in input le caratteristiche identificate nei passaggi precedenti e le combinano per produrre una previsione finale. Questi strati cioè cercano di dare significato alle forme riconosciute nel layer precedenti: identificano gli oggetti o le forme nella foto basandosi sulle caratteristiche che hanno trovato nei passaggi precedenti.
- **Output Layer**: la previsione viene fornita come output nell'ultimo strato della rete.

Sono gli strati convoluzionali a rendere possibile il riconoscimento degli oggetti all'interno dell' immagine essendo per questo motivo uno dei componenti fondamentali delle CNN. Questi strati utilizzano un set di filtri, chiamati *kernel* o filtri di convoluzione, per scansionare il

frame in ingresso. Ogni filtro esegue un' operazione matematica su una piccola porzione dell' immagine, con l' obiettivo di rilevare una specifica caratteristica dell' immagine. Ad esempio, un filtro potrebbe cercare di identificare una linea orizzontale nell' immagine, cercando i cambiamenti di intensità dei pixel lungo una serie di linee orizzontali. Un altro filtro potrebbe cercare una forma geometrica, individuando i bordi di una figura.

Questi e altri metodi sono resi possibili dalla specifica codifica che caratterizza l'elaborazione delle immagini: si utilizzano immagini RGB per le quali il frame è strutturato secondo una forma matriciale a griglia dove ciascuna casella rappresenta un pixel. Per ogni pixel durante l'acquisizione dell'immagine si raccolgono tre valori che stanno ad identificare l'intensità cromatica dei tre colori primari, rosso verde e blu, presentata nel punto: si esprime con un valore da 0 a 255 per ciascuno dei tre colori da specificare. La rete, confrontando valori di codifica RGB di pixel vicini, comprende se possa o meno esserci una corrispondenza riconoscendo le forme nell'immagine.

Nella pratica, un filtro di convoluzione è una matrice numerica, di dimensioni ridotte rispetto all'immagine di input, che viene convoluta con l'immagine. Il filtro scorre lungo l'immagine, pixel per pixel, eseguendo l'operazione di convoluzione elemento per elemento, tra il filtro e la porzione dell'immagine corrispondente. Il risultato di ogni convoluzione viene assegnato allo specifico pixel della mappa delle caratteristiche corrispondenti. La mappa che raccoglie le informazioni delle forme riconosciute viene poi utilizzata dagli strati successivi della CNN per l'identificazione degli oggetti.

Come accennato in precedenza, i nodi che compongono la rete sono interconnessi tra loro secondo logiche che li suddividono in strati e li classificano a seconda della loro funzione. I collegamenti che permettono queste connessioni sono gestiti da dei parametri noti come pesi che definiscono la tipologia e la robustezza della connessione.

La precisione e le prestazioni raggiungibili da una rete neurale sono tali grazie alla struttura interna della stessa. Il modello che elabora le immagini permette un'elaborazione molto efficiente delle informazioni fornite come input e permette la produzione di un risultato molto preciso e accurato. Grazie alla loro capacità di apprendere dai dati, le reti neurali sono in grado di produrre modelli di apprendimento automatico altamente efficaci e generalizzabili, in grado di affrontare problemi complessi e di grande scala.

Questi livelli di qualità sono possibili solo se la rete che si utilizza è adeguatamente settata e, nello specifico, se i pesi che regolano i nodi e le loro interconnessioni sono opportunamente

tarati. In questi termini, la rete neurale svolge inizialmente un processo di apprendimento nel quale vengono modificati e ottimizzati i pesi in modo da minimizzare l'errore di classificazione.

La rete neurale viene allenata al riconoscimento degli oggetti grazie ad una serie di immagini, accompagnate da etichette e posizioni degli oggetti contenuti, fornite come input durante questa prima fase di apprendimento. Il sistema risulterà poi in grado di riconoscere gli oggetti ricercati in immagini nuove, nella fase di *Detect*. La prima fase di allenamento prevede una regolazione dei pesi delle connessioni attraverso l'uso di algoritmi di apprendimento automatico. In questo modo, la rete neurale è in grado di apprendere la relazione tra l'input e l'output desiderato e quando le saranno date in pasto immagini nuove sarà in grado di riconoscere gli oggetti contenuti al suo interno.

### 2.2.2 Introduzione a Yolo v7

YOLO (You Only Look Once) è un popolare algoritmo di rilevamento oggetti utilizzato nella computer vision. L'algoritmo adotta un approccio *one-shot*, cioè osserva l'intera immagine una sola volta e produce in uscita un insieme di bounding box che rappresentano la posizione e la dimensione degli oggetti rilevati.

Questa rete neurale, evoluta negli anni in diverse versioni, basa il suo funzionamento sulla divisione dell'immagine in ingresso in una griglia di celle. Per ogni cella prevede un numero fisso di bounding box che potrebbero contenere un oggetto. Ciascun bounding box è associato ad un punteggio di confidenza, che rappresenta la probabilità che il contenitore contenga o meno l'oggetto con un certo grado di accuratezza. L'algoritmo prevede di eliminare le caselle ridondanti e sovrapposte mantenendo solamente quelle con valori di confidenza sopra un certo valore, fino ad ottenere le aree d'interesse dove riconosce gli oggetti ricercati.

YOLO utilizza una rete neurale convoluzionale (CNN) per effettuare le previsioni di riconoscimento. La rete è composta da un'architettura di base che estrae le mappe delle caratteristiche dall'immagine di input.

Uno dei principali vantaggi di YOLO è la sua velocità: poiché elabora l'intera immagine in una sola volta può rilevare gli oggetti in tempo reale. YOLO è anche noto per la sua alta precisione, in particolare per gli oggetti piccoli, grazie sempre alla sua capacità di sfruttare le informazioni contestuali dell'intera immagine.

YOLO rappresenta un'innovazione rispetto ai precedenti metodi di identificazione degli oggetti: propone un metodo *end-to-end* che prevede la definizione simultanea dei bounding box

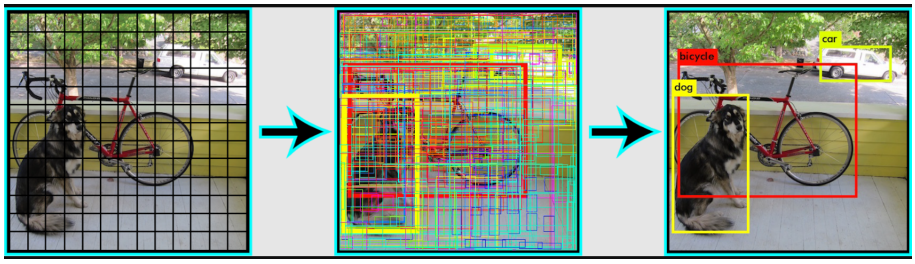



**Figura 2.3:** Concetto base del funzionamento di YOLO, [1]

e delle probabilità con cui viene rilevata l'etichetta di una certa classe per ciascuna delle aree stabilite.

L'immagine in ingresso viene suddivisa in N griglie, ciascuna con un settore dimensionale SxS di uguali dimensioni. Ciascuna di queste N griglie ha il compito di rilevare e localizzare l'oggetto in essa contenuto.

In figura 2.3 viene rappresentato quanto si sta descrivendo. In particolare si possono visualizzare la griglia definita inizialmente e le relative suddivisioni che vengono svolte sull'immagine. Come si può notare, inizialmente, per ciascuna cella vengono espresse le bounding box con una ridondanza molto elevata: molte caselle si sovrappongono e si rifanno a delle aree identiche o poco diverse. A ciascuno di questi riquadri si accostano le dimensioni del box che ne delimita l'area, le relative coordinate della cella e il nome dell'elemento che si ipotizza aver riconosciuto (espresso in figura dai diversi colori dei riquadri) collegato al valore di probabilità.

Poichè molte celle fanno riferimento alla stessa sezione e predicono quindi lo stesso elemento con multiple previsioni dello stesso riquadro di delimitazione, grazie ad una combinazione dei vari riquadri vicini e una trattazione matematica delle probabilità annesse, è possibile eliminare le zone con percentuali più basse mantenendo e definendo il bounding box preciso dell'oggetto come sovrapposizione e composizione dei bounding box primitivi. Quindi, con l'obiettivo di evitare previsioni duplicate, è fondamentale sopprimere tutti i riquadri di delimitazione con punteggi di probabilità inferiori ad un valore di riferimento prestabilito e definito come *Non-Maximal Suppression* (NMS). Questo processo viene utilizzato per ridurre il numero di bounding box mantenendo solamente quelli con un livello di accuratezza maggiore ed eliminando quelli ridondanti. La selezione continua fino alla definizione dei bounding box finali che delimitano gli oggetti.

Grazie a questa tecnica viene ridotto notevolmente lo sforzo computazionale dato che il rilevamento e il riconoscimento degli oggetti sono gestiti simultaneamente dalle sottocelle dell' immagine.

Negli anni sono state sviluppate diverse versioni di YOLO adattate alle diverse necessità progettuali. Di seguito se ne riportano le versioni più comuni.

YOLO v1 è la prima versione presentata nel 2015 nella pubblicazione [20]. L'articolo descrive l'architettura della rete neurale, composta da diversi layer di convoluzione che terminano con un layer di output che fornisce le coordinate dei bounding box e le probabilità associate agli oggetti riconosciuti. Viene inoltre specificato il metodo di allenamento della rete che sfrutta una funzione di costo per la penalizzazione delle classificazioni errate e delle imprecisioni dei bounding box. Vengono espressi i risultati dell'algoritmo ottenuti dall'elaborazione di diversi dataset, dimostrando la capacità di produrre degli output in tempo reale a prestazioni comparabili con gli altri metodi di rilevamento oggetti. Grazie alle ottime prestazioni in termini di velocità, precisione e capacità di apprendimento, YOLO ha avuto un successo rapidissimo dominando l'area dell'identificazione degli oggetti e diventando l'algoritmo più utilizzato. Gli autori, per rendere possibile il riconoscimento degli oggetti, hanno affrontato il tema come un problema di regressione dei riquadri creati e valutati tramite la logica vista sopra con l'utilizzo di una singola rete neurale. Il primo algoritmo riesce ad elaborare fino a 45 fotogrammi al secondo garantendo un funzionamento realtime e un output con mappatura a risoluzione doppia rispetto ad altri rilevatori in tempo reale. Si estende la prima versione prevedendo delle librerie di riconoscimento capaci di riconoscere fino a 9000 diverse categorie di articoli, definendo così la seconda versione YOLO v2. Nella terza generazione dell'algoritmo, YOLO v3, si svolgono alcune migliorie, come ad esempio l'aumento del numero di layer convoluzionali, per rendere possibile il riconoscimento di oggetti di dimensioni ridotte e dei loro più piccoli dettagli. Nel 2020 è stata presentata YOLO v4 per la quale è stato reso possibile un incremento della precisione nel riconoscimento degli oggetti di un 10% rispetto alla generazione precedente. La quinta versione dell'algoritmo, YOLO v5, garantisce una completa visione *open source* mettendo a disposizione una serie di modelli anche preallenati da specifici dataset.

L'ultima versione rilasciata, YOLO v7, risulta essere estremamente veloce e performante oltre che di semplice utilizzo. Un' importante nota di innovazione è stata introdotta dal concetto di assegnazione dinamica dei bounding box.

In aggiunta a quanto visto finora, dopo la scrematura e l'eliminazione dei riquadri ridondanti, i bounding box selezionati vengono ridimensionati e riposizionati in modo dinamico in base alla forma e alle dimensioni dell'oggetto rilevato al fine di garantire una migliore copertura e una maggiore precisione nella definizione della regione di interesse.

Una seconda novità introdotta da questa versione è la riparametrizzazione del modello. Questa tecnica di ottimizzazione viene utilizzata per sfoltire il numero di parametri interni ad una rete neurale così da migliorare l'efficienza computazionale e la velocità di esecuzione del modello. Nella versione specificata si sostituiscono alcuni layer convoluzionali della rete con altri più efficienti garantendo lo stesso grado di predizione degli oggetti nell'immagine. Un esempio di riparametrizzazione utilizzata spesso in questa settima generazione consiste nella sostituzione di alcuni layer convoluzionali completi con layer di dimensione ridotta, layer *Depth-wise Separable Convolution*, garantendo appunto un abbassamento dei costi computazionali richiesti alla rete.

### 2.2.3 Fase di Training

La prima fase richiesta nell'implementazione di una rete neurale è la fase di *Training* durante la quale si predispone l'addestramento di tutti i parametri e i pesi necessari alla rete per il riconoscimento degli oggetti nelle future immagini che riceverà in input. Per attuare un' idonea rilevazione degli oggetti, la rete ha la necessità di processare una grande quantità di dati, dai quali deve ricavare specifiche informazioni utili all'applicazione in esame. I dati che deve elaborare corrispondono ad immagini scattate in due diverse situazioni, sia in scene ideali create su misura con sfondi monocromatici e situazioni di luce controllata, sia in ambienti operativi reali con un contesto ben definito nel quale l'oggetto è immerso e contornato da dettagli anche complessi. Le immagini che andranno a formare il *Training Set* devono contenere informazioni utili alla rete per poter rilevare gli oggetti, devono quindi contenere delle informazioni sui tipi di oggetti contenuti e sulle posizioni che questi hanno all'interno dell'immagine. Questi frame inseriti in input alla rete fungono da esempi grazie ai quali l'algoritmo apprende le caratteristiche degli oggetti che nelle esecuzioni future saprà riconoscere autonomamente.

#### Acquisizione delle immagini

Due aspetti fondamentali nella preparazione e nella scelta delle immagini da fornire come dataset di input alla fase di addestramento della rete sono la varietà e la diversità delle immagini selezionate. È infatti fondamentale fornire all'algoritmo un insieme completo ed eterogeneo di immagini, le quali possano raccogliere tutti i diversi dettagli e tutte le diverse condizioni in cui può essere fotografato l'oggetto nella scena, quando sarà richiesto il suo riconoscimento all'interno dello spazio di lavoro. Si deve essere in grado di garantire una completa acquisizione di quello che è il dataset di informazioni da fornire alla rete per il training. Se non fosse raggiunto

un certo dettaglio in questi termini, se si utilizzassero cioè immagini simili che ritraggono gli oggetti da una sola angolazione e nelle stesse condizioni, la rete neurale quando dovrà essere utilizzata nelle sue condizioni operative post allenamento, si limiterà a rilevare gli oggetti solo nell'ambientazione e con le caratteristiche su cui è stata allenata: non si garantirebbe un funzionamento completo della rete, la quale risulterebbe non in grado di svolgere il compito del riconoscimento con le prestazioni richieste. È quindi preferibile effettuare una raccolta di immagini diversificata, adattando così l'algoritmo a diverse condizioni per garantire un alto livello di affidabilità del sistema.

Nello specifico, in ambito industriale, è fondamentale catturare delle immagini che abbiano delle caratteristiche diverse in termini di luminosità, così da simulare le scansioni in diversi momenti della giornata e in diverse situazioni di luce che possono variare a seconda di dove viene installato il sistema. La luce naturale cambia in base all'ora del giorno, la fonte luminosa può essere artificiale con diverse caratteristiche di colori e orientazioni.

È altrettanto fondamentale scattare delle immagini immortalando l'oggetto da diverse angolazioni: l' angolo tra la fotocamera e il soggetto dell'immagine può variare da caso a caso ed è fondamentale considerare questa variabile in quanto nelle possibili applicazioni gli oggetti possono presentarsi raggruppati o comunque giungere orientati in modo molto differente. Per la stessa motivazione di variabilità delle situazioni, anche la distanza tra la fotocamera e il soggetto può variare considerevolmente.

Quando si effettua la scelta delle immagini è perciò fondamentale considerare tutti questi aspetti, scattando quindi le foto con diverse condizioni di luce, da diversi punti di vista, variando angolazioni e distanze.

### Image labeling

Il processo di apprendimento, durante il quale la rete neurale impara a riconoscere gli oggetti e a classificarli all'interno delle immagini fornite in input, può essere svolto utilizzando due diverse classi di apprendimento: supervisionato (*supervised learning*) o non supervisionato (*unsupervised learning*).

Nel processo di apprendimento supervisionato, la rete viene addestrata tramite l'utilizzo di un modello di training per il quale in input si riportano il set di immagini e il loro risultato atteso. In questo caso le immagini fornite come dataset d'ingresso rappresentanti gli oggetti da riconoscere sono accompagnate da etichette che forniscono informazioni sulla posizione e sulla tipologia di oggetto presente al loro interno. I frame in questo caso sono definiti come *labeled*.

Per altri sistemi è possibile anche l'apprendimento non supervisionato. Questi modelli sono in grado di apprendere le informazioni di training senza la necessità di dati classificati e predeterminati. L'algoritmo impara autonomamente la rappresentazione interna alle immagini o le caratteristiche importanti che la definiscono per definire relazioni o strutture contenute al suo interno.

Nel progetto che si sta esaminando viene utilizzato un modello di training supervisionato. Di conseguenza, una volta archiviate le immagini, è necessario referenziare ed esprimere le informazioni contenute in ogni frame raccolto: si crea quindi un *label* per ciascuna foto fornendo le informazioni dell' oggetto da riconoscere salvandone la posizione, espressa dalle coordinate di due vertici del bounding box creato e l'etichetta relativa alla classe di appartenenza definendo l'oggetto contenuto nella cornice.

L' etichettatura delle immagini è il processo di identificazione e marcatura dei vari dettagli in un' immagine. Il grado di accuratezza dell'algoritmo è conseguenza dell' attenzione riposta in questa fase: si devono definire con cura e attenzione i *label* degli oggetti all'interno delle immagini. Vanno definiti i contorni cercando di delimitare con precisione i prodotti. È fondamentale non tagliare porzioni degli stessi o al contrario includere nelle aree selezionate zone dell'immagine non comprendenti i soggetti d'interesse: più preciso e stretto attorno all'oggetto è il riquadro e più accurato sarà il processo di riconoscimento operativo futuro. L'algoritmo, sfruttando opportuni software grafici (nel progetto si utilizza *labelImg*), preleva le immagini e da esse estrae le caratteristiche delle aree d'interesse esportandone le informazioni in un file di testo *.txt* separato, con lo stesso nome dell'immagine per associare correttamente le etichette alle immagini a cui fanno riferimento. Questo file contiene la posizione dei due vertici opposti che identificano univocamente il bounding box e la classe di appartenenza relativa all'oggetto evidenziato.

Come si esporrà nel capitolo 3, il database creato nel progetto che si sta presentando pone attenzione ad un ristretto range di rifiuti: questo può porre le basi per applicazioni future che hanno la possibilità di ampliare il dataset di immagini inserendo nuovi prodotti in modo da ampliare e diversificare le classi di oggetti riconoscibili dalla rete.

### Training della rete neurale

Si procede ora con la fase di addestramento vera e propria. Nello specifico questa comporta diverse iterazioni attraverso le quali lo stesso set di dati viene analizzato più volte per fare in modo che ad ogni ciclo la rete possa addestrarsi in modo sempre più accurato. In questa fase vengono quindi aggiornati i pesi relativi ai nodi e i parametri della rete in modo da diminuire l'errore tra

le previsioni e i valori di output desiderati. Un singolo ciclo viene definito come epoca. L'addestramento per essere considerato funzionale deve comprendere un numero elevato di epoche, anche se aumentando le epoche aumenta anche il tempo necessario e lo sforzo computazionale richiesto al sistema.

Il primo step di preparazione dei dati si conclude con la divisione delle immagini, e delle relative informazioni, in diverse sottocartelle richieste dai protocolli di YOLO. Nello specifico le immagini vengono suddivise in *Training Set*, *Validation Set* e *Testing Set*. La prima categoria di immagini serve per il vero allenamento della rete. Il set di validazione e quello di test servono per valutare la performance della rete nei vari momenti dell'allenamento. Prima dell'esecuzione della prima epoca, vengono inizializzati i pesi della rete in modo completamente casuale.

A livello computazionale l' apprendimento si compone di diverse fasi. Nella prima, detta propagazione in avanti (*forward propagation*), vengono elaborate le informazioni per il calcolo di un risultato approssimativo. Confrontando poi la stima svolta dalla rete e l'output desiderato viene calcolato l'errore di predizione utilizzando un'opportuna funzione di costo. Questo errore viene poi propagato all'indietro (*backward propagation*) sfruttando il concetto di derivata andando ad aggiornare i pesi dei vari nodi e layer che compongono la rete utilizzando un algoritmo di ottimizzazione così da ridurre l'errore nelle predizioni successive. Come conclusione di un ciclo computazionale avviene la valutazione delle performance durante la quale la rete viene analizzata sfruttando il set di dati di convalida e test: se la performance non è soddisfacente, il processo di allenamento viene ripetuto con un set di parametri e pesi diversi. Ad ogni epoca la rete impara a riconoscere sempre più oggetti e ne migliora la definizione della loro posizione, dimensione e forma.

Durante l'esecuzione delle prime epoche la rete può mostrare una certa instabilità e incertezza nelle rilevazioni; con l'aumentare delle epoche la precisione migliora e le predizioni risultano essere più affidabili. In generale, il numero preciso dipende dal set di dati di addestramento, dalla qualità delle immagini e nella precisione con cui si sono definiti i bounding box per le informazioni di posizione nella fase di labeling.

Un numero di epoche basso porta con se un' imprecisione nel riconoscimento degli oggetti. Una rete neurale non addestrata abbastanza a causa di un numero insufficiente di epoche avrà una precisione ridotta. Ciò significa che la rete potrebbe non essere in grado di riconoscere correttamente gli oggetti o le immagini su cui è stata addestrata. Anche se riporta tempi di addestramento più brevi, si ottiene un risultato impreciso e il più delle volte inutilizzabile.

Numerose epoche sono quindi necessarie per ottenere un buon livello di addestramento della rete. Il numero di epoche è direttamente correlato al tempo necessario per la computa-

zione dell'algoritmo: più epoche richiedono più tempo, ma sono necessarie per raggiungere prestazioni adatte all'applicazione.

Se, d'altro canto, il numero di epoche di addestramento per una rete risulta essere troppo elevato si può incorrere nel problema dell' *overfitting*. Una rete neurale addestrata per un numero eccessivo di epoche potrebbe adattarsi troppo ai dati di addestramento e non generalizzare bene su nuovi dati. Secondo questo fenomeno la rete neurale memorizza i dati di addestramento e, nelle fasi successive, quando le si richiederà di riconoscere gli oggetti in immagini nuove non riuscirà a farlo perchè l'overfitting genera una riduzione della capacità di generalizzazione dei dati. La rete risulta in questo caso troppo specializzata sui dati di addestramento perdendo la capacità di riconoscere oggetti sui nuovi dati. Altra problematica di un numero troppo elevato di epoche è l'elevato tempo richiesto per l'addestramento: aumenta il tempo necessario per addestrare la rete elevando anche i costi computazionali.

Va quindi ricercato un numero di epoche ottimo, in modo da ottenere un buon livello di allenamento della rete per garantire un'efficienza nel riconoscimento degli oggetti ed evitare le problematiche sopra illustrate. La fase di training termina quando la rete neurale ha raggiunto un livello di accuratezza soddisfacente sul set di dati di training.

### Parametri e output della fase di training

Durante lo svolgimento della fase di training, YOLO mette a disposizione alcuni parametri che per ogni epoca permettono di valutare le prestazioni raggiunte dalla rete. Per la ricerca del corretto numero di epoche è necessario eseguire l'allenamento della rete diverse volte così da eliminare i tentativi in cui l'algoritmo converge a dei risultati non abbastanza precisi o al contrario con un contenuto di errore dovuto all'overfitting. Il confronto tra i diversi test e la loro valutazione possono essere svolti in diversi modi. In questa analisi si decide di confrontare i parametri di output dell'addestramento forniti ad ogni epoca. Vengono di seguito definiti i parametri principali utilizzati dalla rete neurale YOLO:

- **Intersection over Union (IoU)**: noto anche come indice Jaccard, è una delle metriche più utilizzate. L'accuratezza della previsione fatta nella definizione del bounding box viene valutata da questo parametro, calcolato come segue:

$$IoU = \frac{Area\ dell'intersezione}{Area\ dell'unione} \qquad IoU \mapsto 1$$

  Nello specifico, più si ha un valore alto di IoU e più il bounding box previsto dalla

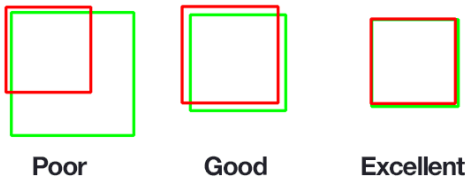



**Figura 2.4:** Esempio di IoU. In verde l'oggetto e in rosso la previsione.

rete corrisponde con il reale contorno dell'oggetto (figura 2.4). L'indice appena definito dovrebbe convergere al valore 1.

- **Objectness (Obj)**: concetto di oggettività. Rappresenta una misura della probabilità che un oggetto esista realmente in una regione di interesse proposta. Per *Obj* elevati il bounding box definito contiene un oggetto con una buona probabilità.

$$Obj \mapsto 1$$

- **Classification (Cls)**: legato al concetto di classificazione delle immagini, specifica la qualità con la quale il sistema ha effettuato l'assegnazione dell'etichetta al relativo bounding box. Il valore dovrebbe convergere a 1 con l'aumentare del numero di epoche.

$$Cls \mapsto 1$$

Questi parametri sono fondamentali per riuscire a quantificare la qualità delle prestazioni della rete. Vengono spesso usati per valutare le previsioni durante l'addestramento della rete. Il parametro IoU ad esempio, calcolato e assegnato ad ogni bounding box definito, definisce la qualità con cui sono stati definiti i singoli contorni: si sfrutta per distinguere i contorni validi e corretti da quelli sbagliati. Si mantengono validi quei bounding box che hanno un valore di IoU maggiore rispetto ad una soglia specificata così da scartare quelli imprecisi e non adatti di scarsa qualità.

Durante la successione delle epoche, questi valori forniscono quindi le informazioni su come YOLO stia eseguendo il Training rilevando eventuali problemi. Vi sono poi altri parametri, noti come vere e proprie metriche di valutazione, relegate a quantificare la precisione di riconoscimento degli oggetti lungo le epoche. Tra queste si definiscono:

- **Precision (P)**: definisce quanto la predizione sia accurata, ovvero la percentuale di previsioni corrette (*True Positive, TP*), sul totale delle previsioni fatte, che include i True

Positive e i *False Positive (FP)*.

$$Precision = \frac{TP}{TP + FP} \quad P \mapsto 1$$

Questo parametro indica quanti oggetti rilevati dal modello sono effettivamente e realmente oggetti nell'immagine elaborata. Risulta essere una metrica importante per la valutazione della qualità delle predizioni fatte dalla rete. Precisione elevata indica un modello capace di individuare opportunamente gli oggetti d'interesse e ridurre il numero di falsi positivi.

- **Recall (R)**: misura la proporzione di oggetti riconosciuti correttamente dal modello rispetto al totale degli oggetti presenti realmente nell'immagine. Il totale degli oggetti presenti nell'immagine è dato dalla somma del numero di oggetti riconosciuti (*True Positive, TP*) e gli oggetti non riconosciuti dal modello (*False Negative, FN*).

$$Recall = \frac{TP}{TP + FN} \quad R \mapsto 1$$

Un valore elevato di Recall indica che il modello è in grado di individuare un elevato numero di oggetti di interesse presenti nell'immagine.

- **F1**: è una metrica composta che tiene conto del bilanciamento tra Precision e Recall. È infatti importante bilanciare i due parametri poichè massimizzarne uno dei due potrebbe ridurre il secondo. Fornisce un' indicazione complessiva della capacità del modello di individuare correttamente gli oggetti. Avere un valore circa unitario di F1 significa avere un buon bilanciamento dei valori di Precision e Recall.

$$F1 \mapsto 1$$

- **mAP**: rappresenta la media delle precisioni (*Mean Average Precision*) nel rilevamento degli oggetti. Viene espressa sia relativamente alle singole classi che con un riferimento generale a tutti gli oggetti da riconoscere. Questo parametro tiene conto sia della correttezza della classificazione dell'oggetto, valutando quindi la precisione con cui viene attribuita l'etichetta, che della precisione della localizzazione dell'oggetto. Se alcuni degli oggetti correttamente rilevati come True Positive vengono localizzati in maniera imprecisa nello spazio, l' mAP avrà un valore inferiore. L' mAP fornisce una misura complessiva della capacità di rilevazione e classificazione degli oggetti con alta precisione e accuratezza.

$$mAP \mapsto 1$$


**Figura 2.5:** Esempio di riconoscimento oggetti con definizione dei bounding box

Questa prima fase di preparazione si conclude con un' elaborazione di un file di output che sarà poi utilizzato nei successivi step dalla rete. Questo output è un file contenente i pesi ottimizzati della rete neurale da utilizzare poi nella fase di *detect* per l'individuazione e la classificazione degli oggetti nelle immagini. YOLO v7 durante il training produce anche diverse metriche di valutazione per il monitoraggio delle performance della rete. Si includono specifiche di precisione (P), recall (R), mAP e F1.

### 2.2.4 Fase di Detect

Una volta concluso l'allenamento della rete, tutti i parametri necessari al riconoscimento degli oggetti sono pronti per essere utilizzati per l' elaborazione di nuove immagini. Grazie all' addestramento dei pesi relativi ai nodi della rete, concluso nella sezione di Training, è possibile, grazie ai protocolli interni alla rete neurale, procedere con la rilevazione e la classificazione degli oggetti contenuti nelle nuove immagini scattate. Le immagini acquisite vengono quindi inserite come input alla rete neurale ed elaborate per la restituzione delle informazioni relative alla posizione e alla classificazione degli oggetti in esse contenute come output.

Nello specifico, in questo processo si identificano e si localizzano all' interno delle immagini le classi di oggetti per le quali la rete è stata addestrata. Per ciascun elemento individuato, verranno definiti il bounding box, il nome della classe di appartenenza e la percentuale di confidenza con la quale l'oggetto è stato individuato. Questo ultimo valore numerico, compreso tra 0 e 1, rappresenta la probabilità che l'identificazione sia stata fatta correttamente. Un esempio di rilevamento degli oggetti viene espresso in figura 2.5.

Perchè si possa considerare una rilevazione di successo, un oggetto deve essere riconosciuto dal sistema con una percentuale di confidenza vicina all' unità, così da garantire un elevato

valore di precisione. Se nell'immagine scattata vengono immortalati contemporaneamente più oggetti, questi possono essere riconosciuti dalla rete in modo simultaneo ed essere elaborati singolarmente producendo delle metriche di confidenza diverse.

A questa fase di rilevamento appartengono tutte quelle operazioni che possono risultare considerate come test del modello. Nei paragrafi precedenti si è definito che, durante la fase di training, è stato necessario ripetere l' addestramento diverse volte, definendo cicli di epoche diversi in caso di prestazioni non soddisfacenti. Risulta fondamentale, ad ogni allenamento effettuato, testare la configurazione dei pesi raccolti su un insieme di immagini rappresentative.

Seguendo una presentazione teorica della fase di detect in esame, la si può descrivere come una composizione di diversi passaggi.

Per prima cosa l' immagine di input viene ridimensionata e adattata alle caratteristiche necessarie per l' input della rete neurale (*image processing*). La successiva *forward propagation* prevede di analizzare l'immagine pre-elaborata attraverso la rete convoluzionale, sfruttandone i pesi ottimizzati. Applicando una serie di filtri e attivando opportune funzioni matematiche si ottengono delle mappe contenenti informazioni di posizione dei possibili bounding box. Si evidenziano quindi gli oggetti riconosciuti con gli appositi contorni sfruttando delle operazioni fatte sui pixel e sulle informazioni cromatiche che li definiscono. Come nella fase di training, anche qui avviene una soppressione dei bounding box ridondanti e duplicati tramite opportuni algoritmi che permettono di mantenere solamente i riquadri con probabilità di confidenza maggiore. Infine, ogni bounding box viene classificato secondo la classe di appartenenza dell'oggetto a cui fa riferimento.

## 2.3 Intel Realsense Depth Camera D435

Nella visione artificiale, nella robotica e in ambito industriale, le camere 3D vengono utilizzate per la determinazione delle informazioni sull'orientamento e sulla posizione dei componenti e degli oggetti presenti nell'area di lavoro per un controllo di qualità, per comandare azioni di pick o per permettere di gestire una traiettoria di movimento evitando eventuali ostacoli presenti.

Nella specifica applicazione che si sta esponendo in questo testo, il sistema di visione viene sfruttato per il riconoscimento degli oggetti e una successiva elaborazione dei dati di posizione ed orientazione degli stessi in relazione a specifici sistemi di riferimento per rendere possibile le azioni di pick e place.

È quindi fondamentale stabilire come vengono codificate le informazioni all'interno delle

**Figura 2.6:** Fotocamera Intel Realsense D435, Vista frontale, [2]

immagini acquisite dal sistema di visione e come queste possono essere elaborate e combinate tra loro per poter così riprodurre lo spazio di lavoro e gli oggetti in esso contenuti a livello software.

Il sistema di visione, come sottolineato dalle specifiche di progetto, viene scelto escludendo le proposte integrate a dei sistemi robotici. In questo progetto infatti si decide di non utilizzare un sistema di visione proprietario relativo ad una qualsiasi casa produttrice di robot, ma si vuole implementare una soluzione che in alternativa proponga un modello open source, snella e che permetta di modificare ed adattare all'applicazione le specifiche che la compongono. In questi termini si è ben coscienti del fatto che, procedendo verso la scelta di un sistema dotato di specifiche tecniche molto meno performanti rispetto a quelle disponibili in commercio, le potenzialità risultano essere molto limitate. Questo non va ad intaccare la produttività o la precisione del sistema che si sta descrivendo: tutte le attività richieste, dalla misurazione alla gestione delle informazioni, sono garantite con alti livelli di affidabilità. Si esprime inoltre come la soluzione scelta possa in qualche modo rispondere a delle specifiche di costo molto meno rilevanti rispetto a sistemi completi presenti nel mercato. La camera utilizzata nel progetto è una *Intel Realsense Depth Camera D435*, mostrata in figura 2.6.

Il sistema di visione scelto concede quindi di soddisfare tutte le specifiche richieste dal progetto, escludendo comunque svariati altri metodi e tecnologie che sarebbero messi a disposizione

da sistemi di visione completi. Normalmente, nelle celle robotiche utilizzate nel bin picking, i sistemi di visione sono costituiti da un proiettore e una o più fotocamere. Queste sono tutte caratteristiche presenti nelle specifiche della camera scelta, come visualizzato in figura 2.7.

### 2.3.1 Caratteristiche della camera

La fotocamera Intel Realsense D435 è una camera che sfrutta la tecnologia stereoscopica per creare immagini a colori e misurazioni di profondità in tempo reale. La camera è in grado inoltre di combinare le informazioni consentendo di creare mappe 3D dettagliate di oggetti e ambienti di lavoro garantendo degli standard di precisione e accuratezza nel range di profondità che va dai 20 centimetri fino ad un massimo 3 metri.

#### Immagine RGB a colori

La fotocamera D435 è innanzitutto capace di acquisire immagini a colori in molteplici codifiche a seconda delle applicazioni nella quale viene implementata. Questi frame raccolgono le informazioni bidimensionali dello spazio di lavoro in una matrice numerica, la quale sarà poi utilizzata per la successiva elaborazione nella fase di riconoscimento degli oggetti. È quindi fondamentale indicizzare ciascun pixel che compone l'immagine in modo che il codice possa attribuirgli delle informazioni spaziali.

Il sistema di riferimento relativo ad un'immagine a colori indica come sono organizzati i pixel all' interno dell'immagine. Le immagini digitali sono composte da una griglia di pixel, dove ogni pixel identifica un punto di colore tramite una specifica codifica. Come esempio prendiamo un'immagine RGB: questa per ogni pixel prevede di assegnare un valore da 0 a 255 per ciascuno dei tre colori primari, la composizione dei tre determina il colore completo da assegnare a quel pixel.

La posizione del pixel nella griglia è descritta dalle due coordinate x e y. La posizione di origine (0,0) la si trova nell'angolo in alto a sinistra dell'immagine, mentre l'angolo in basso a destra corrisponde alla posizione (larghezza-1, altezza-1), dove larghezza e altezza sono le dimensioni dell'immagine in pixel.

Si deve utilizzare il sistema di riferimento in pixel quando è necessario specificare la posizione di uno o più pixel che sono stati selezionati. In questo modo sarà possibile svolgere le elaborazioni delle immagini catturate e valutare la presenza di oggetti nell'ambiente immortalato. Nello specifico, il riconoscimento degli oggetti sarà possibile attuando un confronto tra i valori della

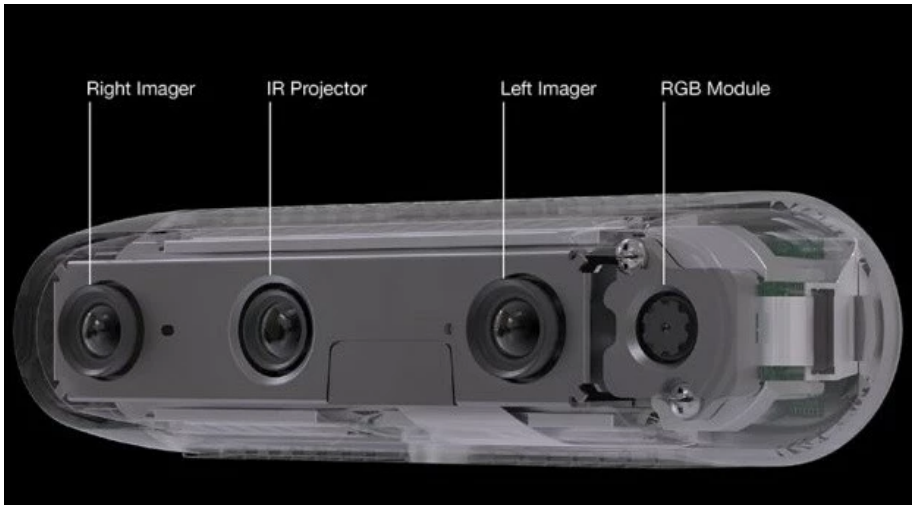



**Figura 2.7:** Fotocamera Intel Realsense D435, dettagli delle focali, [2]

codifica RGB relativa a pixel adiacenti, in modo da ricercare delle continuità di colore o texture in zone ristrette dell'immagine.

### Misurazione della profondità

Il concetto di misurazione stereoscopica deriva da un' idea molto semplice: eseguendo due immagini dello stesso oggetto da posizioni diverse, si possono combinare i due frame e triangolando le informazioni è possibile ricavare le informazioni sulle coordinate 3D relative ad un preciso pixel che compone l'immagine. Questa tecnologia è utilizzata per la determinazione delle dimensioni, delle forme e delle posizioni degli oggetti.

A livello implementativo, la visione stereo è permessa da una coppia di camere identiche configurate con i medesimi settaggi (*left imager* e *right imager*) e un proiettore ad infrarossi. Quest'ultimo permette di migliorare le informazioni di profondità rilevate. Viene proiettata una luce strutturata nel piano, formata da un preciso pattern di punti. Attraverso la misurazione di come la disposizione di questi punti viene deformata quando a contatto con le superfici degli oggetti, si determina la posizione e l' orientazione dei prodotti inquadrati. Questa tecnica, combinata alle camere stereo, risulta essere una delle più affidabili nella ricostruzione delle superfici degli oggetti e per questo molto sfruttata nel settore della robotica.

In questo modo la camera riesce a calcolare la distanza di ogni punto nell'immagine, creando una mappa di profondità ad alta risoluzione.

Considerando le misurazioni di profondità, la fotocamera è in grado di catturare fino a 30 fotogrammi al secondo con una risoluzione massima di 1280 x 720 pixel. L'interfaccia per la comunicazione e l'invio dei dati raccolti è resa disponibile da un cavo USB 3.0 che consente di trasferire le immagini e le informazioni raccolte fino al computer in cui viene eseguita l' elaborazione.

Il prodotto Intel porta con se un pacchetto software (*SDK: Software Development Kit*) che mette a disposizione applicazioni e interfacce grafiche per la calibrazione e la gestione guidata della camera. Sfruttando queste estensioni, è possibile comprendere le varie proprietà della camera e settare a dovere i parametri per progettarne un corretto funzionamento. Questa piattaforma supporta diversi linguaggi di programmazione tra cui Matlab, C/C++, ROS, Python. A livello progettuale si effettua una scelta tra questi diversi ambienti in base alla specifica applicazione in cui la camera può essere implementata.

In questa trattazione ci si sofferma maggiormente sulle applicazioni software incentrate sull'utilizzo del linguaggio di programmazione Python in modo da mantenere una fluidità nella gestione del progetto: l'interfaccia mette a disposizione apposite librerie e funzioni grazie alle quali è possibile un' ottima gestione e fruizione della camera per l'elaborazione delle informazioni ricavate dai frame.

### 2.3.2 Parametri da settare e principio di funzionamento

Come anticipato, si pone l'attenzione sulla gestione della camera e dei suoi parametri tramite il linguaggio di programmazione Python: si riescono così a gestire le specifiche necessarie per un corretto utilizzo. In particolare, per l'implementazione della camera Intel Realsense all'interno dell'ambiente di programmazione, è necessario importare la libreria Pyrealsense. Questa permette di settare e specificare alcune informazioni sulla geometria del sensore, dati necessari per la fase di calibrazione e per la correzione delle indicazioni acquisite. Queste proprietà da definire, espresse all'interno della libreria come parametri a priori (_*intrinsics*), includono la lunghezza focale, i coefficienti di distorsione della lente, le dimensioni in pixel dell'immagine e tutte quelle informazioni per adattare quanto misurato dalla camera alla scena reale fotografata: vanno tarati i parametri così che distanze e profondità dell'ambiente siano misurate e memorizzate correttamente.

La fotocamera D435 ha un campo visivo (*FOV, Field of Viev*) ampio: raggiunge i 91° x 65° x 100° per immagini RGB e 85° x 58° x 90° per le misurazioni di profondità riducendo al minimo

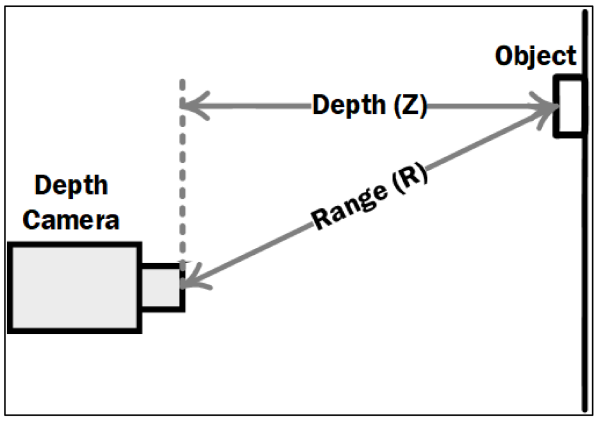



**Figura 2.8:** Principio per la determinazione della profondità

i punti ciechi nella mappa di profondità. Il sensore di profondità montato della camera la rende adatta ad applicazioni in cui non sono richiesti alti livelli di accuratezza e precisione, nei quali risulta fondamentale un'esperienza visiva globale e completa della scena che si sta analizzando.

Accuratezza e precisione, parametri importanti per i sensori di profondità, sono osservabili dalle mappe create dalla camera. La valutazione visiva delle misure è ampiamente utilizzata nella pratica per una prima caratterizzazione delle immagini e dei dati risultanti dalle misurazioni. La suite Intel messa a disposizione dall' SDK precedentemente citato permette di comprendere l'importanza di quanto si sta definendo. Altra specifica della fotocamera in oggetto è l'otturatore globale che consente l'acquisizione delle immagini senza distorsioni anche in condizioni di movimento rapido o vibrazioni della fotocamera o dell'oggetto inquadrato. Questo particolare sensore acquisisce tutta l'immagine in uno stesso istante di tempo, senza scansioni a zone del frame. In ambito robotico questa peculiarità risulta essere molto utile in quanto si possono avere movimenti molto rapidi. Avere quindi dispositivi di visione che possano garantire misurazioni accurate anche con dinamiche di processo elevate è di sicuro un ottimo aspetto.

Come specificato, la camera Intel Realsense produce mappe di profondità mettendo in relazione il prodotto dei frame a colori RGB uniti alle misurazioni di profondità. Queste mappe di profondità hanno delle qualità maggiori in termini di accuratezza e precisione se gli oggetti nella scena sono posizionati ad una distanza di 40-70 centimetri rispetto alla fotocamera. I dati rilevati dal campo dai sensori ottici vengono elaborati all'interno di un microprocessore. Sulla base dei dati ricevuti, l'elaboratore di immagini determina, tramite opportuni calcoli, i valori di profondità di ogni singolo pixel dell'immagine registrata correlando così i valori relativi alla ca-

| Caratteristiche | D435 |
|---|---|
| Risoluzione di Profondità | 16 bit |
| Massima Risoluzione RGB | 1920x1080 pixel |
| Range di Profondità | 0.2m - 3m |
| FOV | 85°x58°x90° |

**Tabella 2.1:** Caratteristiche principali Intel RealSense D435

mera di destra con quelle di sinistra. Collegando poi in successione i frame RGB e di profondità è possibile generare un flusso video di profondità.

Come espresso in figura 2.8, il valore che viene elaborato come parametro di profondità deriva da una misurazione indiretta. La camera stabilisce dei dati di *Range (R)* come valori di distanza degli oggetti dal punto centrale della focale, considerato come origine del sistema. Questi valori, tramite un'opportuna elaborazione effettuata internamente dal microprocessore, vengono proiettati e trasformati come valori di profondità (Z) definiti come distanza dal piano parallelo al sensore ottico che effettua le scansioni. Nel sistema di misura *range*, i punti nello spazio che hanno uno stesso valore di distanza sono appartenenti ad una superficie sferica centrata nell'obiettivo della focale. Nel sistema di riferimento finale invece, come dal concetto di profondità nella visione 3D, punti con profondità identica appartengono ad uno stesso piano, parallelo al piano passante per il sensore di profondità. Un ruolo cruciale nel funzionamento del sensore di profondità in esame è svolto dal processore di segnale (DSP) integrato sfruttato per l'elaborazione delle immagini. Questo dispositivo può elaborare fino a 36 milioni di valori di profondità al secondo. Sfruttando queste elevate prestazioni, i sensori D435 possono essere utilizzati e implementati in tutte quelle applicazioni che richiedono l'elaborazione delle immagini ad alta velocità.

In tabella2.1 vengono riassunte le principali caratteristiche dei dispositivi di visione D435 prodotte dalla Intel.

## 2.4 Kuka LBR IIWA 14 R820

Kuka è un' azienda tedesca presente a livello mondiale nei settori della robotica e dell' automazione industriale. Nel 2013 propone una soluzione collaborativa con la serie LBR IIWA: di questa linea di prodotti ci si concentra sul modello LBR IIWA 14 R820.

L'acronimo LBR sta per *LeichtBauRoboter*, ovvero robot leggero, mentre IIWA indica *Intel-*

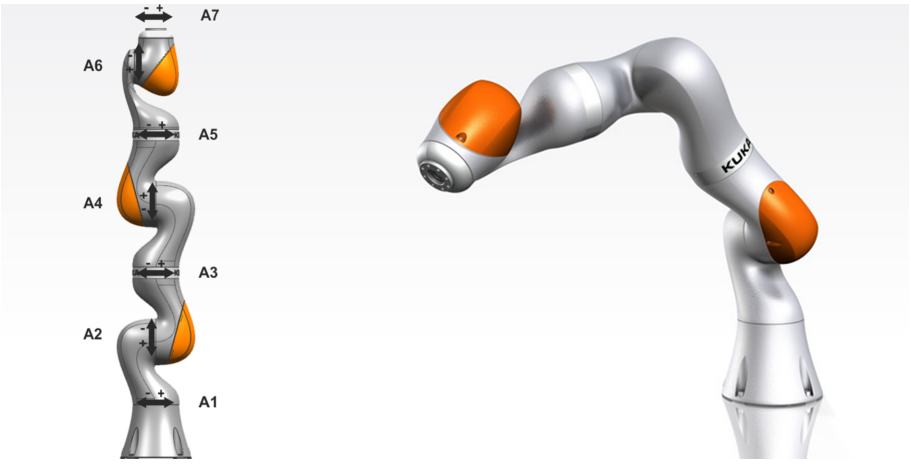



**Figura 2.9:** Robot Kuka LBR IIWA 14 R820

*ligent Industrial Work Assistant*. Questo robot è stato il primo braccio robotico collaborativo implementato a livello industriale nei processi produttivi. Grazie alla tecnologia collaborativa infatti è resa possibile la collaborazione tra uomo e macchina all'interno degli stessi spazi di lavoro senza prevedere l'obbligo di protezioni o distanziamenti di sicurezza. In questo modo è quindi possibile massimizzare la produttività e l'efficienza delle linee produttive andando a minimizzare l'ingombro e gli spazi di lavoro. Il raggio d'azione complessivo è di 820 millimetri (*R820*). In figura 2.9 viene riportata una rappresentazione del cobot (*collaborative robot*) in esame.

È utilizzato nell'industria in svariate implementazioni tra cui le operazioni di pick e place di oggetti. Questo braccio antropomorfo a 7 assi ha una capacità di carico di 14 chilogrammi. La presenza di sensori di coppia integrati su ciascuno dei 7 giunti al robot permette di rilevare il contatto con oggetti o persone presenti nelle zone circostanti e di conseguenza attuare degli stop di emergenza o ridurre le velocità di movimento. Le informazioni che vengono presentate in questo paragrafo sono state raccolte dal sito produttore Kuka [3].

Nel robot antropomorfo articolato a sette assi, l'asse aggiuntivo consente di raggiungere qualunque punto dello spazio di lavoro da qualunque angolazione e con infinite disposizioni del braccio. Tutti i cavi e le unità di azionamento si trovano all'interno del robot così da evitare danneggiamenti o interferenze durante il funzionamento e rendendo sicura l' eventuale collaborazione con operatori umani. Ogni asse contiene diversi sensori che forniscono i segnali necessari al controllo del robot. I sensori di posizione garantiscono che ogni asse si stia muovendo correttamente, i sensori di coppia, come già anticipato, garantiscono il funzionamento collaborativo e assicurano che non vengano superati i carichi impostati su ogni asse. I sensori

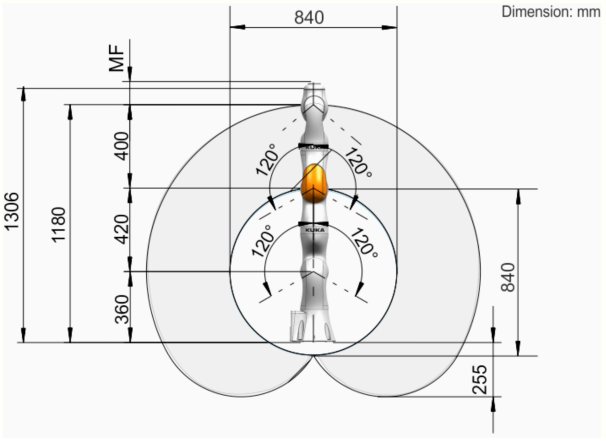



**Figura 2.10:** Scheda Tecnica KUKA IIWA 14 R820, Spazio di lavoro, [3]

di temperatura monitorano i valori termici per rimanere al di sotto dei limiti per un corretto funzionamento dell'elettronica interna.

Alla serie di robot presentata da Kuka appartengono tre diverse versioni: LBR IIWA 7 R800, LBR IIWA 7 R800 CR e LBR IIWA 14 R820.

Le prime due classi presentano due robot molto simili tra loro, con carico massimo applicabile di 7 chilogrammi e un' area di lavoro con raggio massimo di 800 millimetri e una ripetibilità di $\pm 0,1$ millimetri. La versione LBR IIWA 14 R820 è il modello di dimensioni maggiori della serie con un carico massimo di 14 chilogrammi e uno sbraccio di 820 millimetri, con un leggero peggioramento della ripetibilità garantita sui $\pm 0,15$ millimetri. Data la sua massa limitata di soli circa 29,9 chilogrammi, può essere montato a pavimento, a muro oppure a soffitto.

Il carico massimo dipende dalla distanza del centro di gravità del pezzo movimentato dalla flangia sulla quale viene montato l'end-effector con un minimo di 8 chilogrammi per distanze in Z maggiori di 200 millimetri fino al massimo di 14 chilogrammi quando il centro di massa dell'oggetto è in prossimità della pinza. Il sistema robotico richiede una temperatura dell'ambiente di utilizzo compresa tra i 5°C e i 45°C. Il livello di protezione dell'ambiente esterno è classificato come IP54. Il range di movimento dei sette assi è piuttosto ampio e genera un volume di lavoro pari a 1,8m$^3$. Mediamente ciascun asse ha una rotazione consentita di circa $\pm 170°$, ad esclusione della rotazione A6 limitata ai $\pm 120°$. Essendo valori inferiori ai 360° il volume di lavoro non è sferico, lo si riporta in figura 2.10.

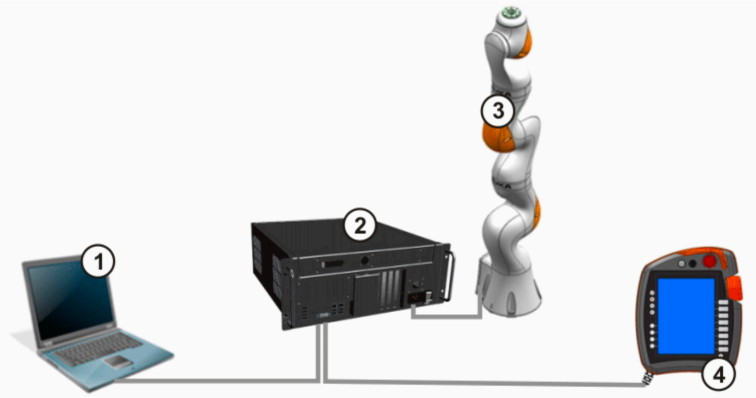



**Figura 2.11:** Kuka Sunrise: separazione tra attività di programmazione e controllo, [3]

### 2.4.1 Kuka Sunrise

Per il controllo, la gestione e il comando del robot Kuka preso in considerazione sopra, si utilizza un'interfaccia integrata al sistema Kuka che rende disponibile un applicativo software che comprende una vera e propria suite: si prende in considerazione la versione Kuka Sunrise Workbench OS 1.16.

Kuka Sunrise.OS è un pacchetto software di sistema integrato per robot industriali nel quale le attività di programmazione e controllo da parte dell'operatore sono separate l'una dall'altra (figura 2.11).

Il programma di movimento viene sviluppato all'interno dell'interfaccia Workbench: la suite software utilizzabile da computer mette a disposizione un compilatore Java entro il quale è possibile definire tutti i parametri per la movimentazione del braccio sfruttando le numerose librerie proprietarie fornite. Tramite l'utilizzo di questo pacchetto software è possibile gestire il progetto implementando le logiche di sistema e impostando le azioni da svolgere durante l'esecuzione del ciclo macchina.

La stazione robotica è composta da un controllore, da un manipolatore e dagli accessori che compongono la sensoristica e la struttura di sicurezza del sistema.

Il controllo del robot, lo *start/stop* delle varie applicazioni, la gestione degli allarmi e dei warning è gestita tramite il pannello di controllo Kuka smartPAD.

La logica generale di funzionamento sfrutta questo flusso di informazioni:

- si sviluppa il codice che definisce la logica di movimento al computer sfruttando Sunrise Workbench e il linguaggio di programmazione Java,

- si invia il programma tramite cavo ethernet o tramite connessione wifi al controllore che gestisce il robot,
- tramite la teach pendant smartPAD si gestisce il controllo del robot, resettando gli errori e inizializzando l'esecuzione del codice quando lo si ritiene necessario,
- sempre tramite smartPAD si verifica la corretta esecuzione del codice oppure si gestiscono gli eventuali errori.

Nella normale gestione del robot, lo smartPAD risulta essere necessario per tutte quelle attività che non possono essere eseguite tramite la programmazione su Sunrise Workbench: ad esempio lo si utilizza per controllare e muovere gli assi manualmente, cambiare e calibrare nuovi utensili e insegnare punti, si possono eseguire task di manutenzione, ma non possono essere effettuate in alcun modo modifiche alla configurazione di sicurezza o al codice implementato tramite programmazione.

## 2.5 Organo Terminale

L' organo terminale deve essere scelto o realizzato ad hoc a seconda della specifica applicazione. Le diverse proposte presenti nel mercato possono essere classificate in due macro categorie: la prima raccoglie gli organi di presa per la manipolazione, la seconda invece racchiude tutti gli organi terminali utilizzati per specifiche lavorazioni.

Gli organi di presa a loro volta si caratterizzano a seconda del contatto che avviene tra il gripper e l'oggetto da movimentare. Nello specifico è possibile un contatto monolaterale, con tecnologie magnetiche, adesive o pneumatiche (tramite l'utilizzo di ventose). In alternativa si può definire un contatto bilaterale sfruttando griffe parallele, angolari o radiali con pinze o griffe. Infine è possibile definire organi che stabiliscono un contatto multilaterale. Nella famiglia degli organi per le lavorazioni invece si trovano specifici *end-effector* a seconda del task richiesto dal robot: torce di saldatura, pistole a spruzzo per la verniciatura, avvitatori e diversi altri utensili.

Non esiste un metodo universale per la selezione dell'organo di presa ottimale. Questo è dovuto al fatto che molteplici applicazioni hanno requisiti e standard diversi ai quali far fronte. Devono essere prese in considerazione le proprietà dell' oggetto e le condizioni di lavoro stimando le forze necessarie e i valori di carico tipici della specifica applicazione.

Di seguito vengono presentate due diverse soluzioni: la pinza parallela bilaterale a due griffe e un organo di presa pneumatico a ventosa. Si sceglie di concentrare l'attenzione in queste

due possibili alternative in quanto sono le scelte che più si avvicinano all'applicazione che si sta presentando.

È fondamentale avere una conoscenza precisa della geometria e delle proprietà fisiche dell'oggetto per la progettazione delle operazioni di posizionamento e orientamento dello stesso nonchè per la determinazione della tipologia dell' organo di presa. A seconda del peso, delle superfici dell'oggetto e della tipologia di movimentazione che si vuole definire si determina la quantità di forza necessaria per la presa e di conseguenza anche la classe di gripper da utilizzare.

### 2.5.1 Pinza Parallela Bilaterale

Per la presa e la movimentazione di componenti di piccole dimensioni ci si può inizialmente orientare verso la classe di pinze parallele a due griffe. Il funzionamento consiste nello sfruttare il movimento lineare delle due pinze: queste muovendosi con verso simmetrico rispetto all'asse centrale della pinza permettono l'apertura e la chiusura della stessa. Il meccanismo che viene riprodotto imita il gesto umano di chiusura della mano per la presa di un oggetto. La pinza mantiene l'oggetto in posizione affinchè possa essere spostato.

Le pinze parallele possono essere utilizzate in molte applicazioni dato che è possibile creare delle griffe personalizzate in modo da realizzare degli organi di presa adatti a pezzi di forma e geometria diversa.

Si parte da un semplice azionamento elettrico che sviluppa un movimento lineare il quale, tramite il principio meccanico di pignone e cremagliera, crea il caratteristico movimento speculare. Il movimento di apertura e chiusura delle griffe, a seconda della corsa delle parti mobili, deve essere adattato agli oggetti che si devono movimentare. In questi termini devono essere ben definiti gli spazi di apertura e chiusura, tenendo sempre conto della corsa caratteristica dell' organo terminale scelto, oltre che alla forma delle stesse griffe perchè si possa definire con precisione il contatto di forza e, se necessario, il contatto di forma.

Una volta definita la tipologia di organo terminale vanno svolte delle valutazioni sulla forma delle superfici delle griffe responsabili del contatto vero e proprio con l'oggetto da prelevare. Queste superfici vengono dette superfici attive e vanno opportunamente definite a seconda dell'oggetto che si preleva.

Le griffe della pinza sono l'organo attraverso il quale viene trasmessa la forza sull'oggetto durante il processo di presa. Esistono diversi tipi di contatto tra le griffe di presa e gli oggetti dipendenti dalla morfologia di entrambe. Il contatto tra griffa e oggetto può avvenire in un solo punto, lungo una linea o tramite un' intera superficie. Un oggetto rigido libero di muo-

versi nello spazio ha sei gradi di libertà con altrettanti possibili moti traslatori e rotatori. Lo scopo del gripper di presa è quello di garantire una certa stabilità durante il processo di presa limitando il numero di gradi di libertà. Dimensionando opportunamente il numero di punti di contatto, e se necessario adattando la geometria delle griffe di presa, è possibile aumentare la stabilità consentendo anche di ridurre la forza da applicare per garantire un corretto pick del pezzo da prelevare.

La forza di presa va dimensionata garantendo che durante la chiusura delle pinze non venga danneggiato l'oggetto, ma, allo stesso tempo, in relazione ai movimenti da svolgere e quindi secondo i carichi inerziali da soddisfare, si deve garantire la stabilità del pezzo tra le griffe fino alla posizione di place dell'oggetto.

Va poi definita la tecnica di presa da utilizzare a seconda che si intenda stabilire un solo contatto di forza o si richieda anche un contatto di forma. Nel primo caso il sostegno dell'oggetto avviene unicamente per effetto delle forze di attrito tra superficie dell'oggetto e griffe di presa. Per un contatto di forma, invece, si deve adattare la geometria delle griffe alle superfici dell' oggetto da afferrare. Scegliendo questa seconda soluzione è il più delle volte possibile esercitare una forza di presa inferiore rispetto al primo caso, ma vi è il vincolo di dover conoscere con precisione la forma dell'oggetto e la necessità di poter lavorare solo con una gamma di oggetti con caratteristiche morfologiche simili.

### Pinza Collaborativa SCHUNK Co-act EGP-C 40

La ditta Schunk è una ditta tedesca specializzata in sistemi di presa. A catalogo presenta molteplici e varie soluzioni applicabili a diversi settori della robotica. La serie di pinze Co-act EGP-C raccoglie diverse configurazioni di organi terminali a contatto bilaterale con griffe parallele. Nel datasheet del prodotto [4] ne vengono definite le specifiche. *Co-act* indica che l'organo di presa ha un attuatore di movimento collaborativo: la pinza infatti è specifica per quelle applicazioni che prevedono un' interazione tra uomo e macchina per la presa di componenti di piccole dimensioni (*EGP*). Vanta la certificazione DGUV (*-C*, Assicurazione tedesca per la sicurezza sul lavoro), per garantire una maggiore sicurezza . Viene qui presentata la versione 40 che ne definisce la dimensione specifica per il robot KUKA IIWA sul quale andrà poi installata.

La pinza Co-act EGP-C, riportata in figura2.12, è ad azionamento elettrico ed è dotata di un limite di corrente per garantire che la forza di presa non superi un valore definito. È montato un rivestimento di protezione dagli urti per ridurre al minimo il rischio di lesioni durante il funzionamento collaborativo. La pinza è disponibile per una vasta gamma di robot collaborativi di diversi produttori tra cui KUKA, Universal Robots e FANUC. Per ciascun produttore vengo-

**Figura 2.12:** Pinza Collaborativa SCHUNK Co-act EGP-C 40

no forniti i materiali per il montaggio seguendo le specifiche delle geometrie e dei collegamenti elettrici.

La forza di presa applicabile all' oggetto parte da un minimo di 18 fino ad un massimo di 70 Newton (considerando una singola ganascia). La corsa di apertura/chiusura di ciascuna griffa è di 6 millimetri. La lunghezza massima delle griffe è di 50 millimetri. Questo ultimo vincolo è dovuto al fatto che, allontanandosi dal piano di scorrimento delle griffe, la forza applicabile all'oggetto diminuisce mentre aumentano le componenti inerziali: è quindi fondamentale limitare la sporgenza delle appendici da installare. Altro dato da specificare è il tempo di chiusura/apertura pari a 0.2 secondi.

### 2.5.2 Organi di presa pneumatici

I dispositivi di presa pneumatici utilizzano l'aria compressa per sollevare e trasportare oggetti. Sono comunemente utilizzati in una varietà di applicazioni in ambito industriale: importanti esempi sono la manipolazione e il picking di piccoli oggetti e componenti che richiedono delicate azioni di pick.

I gripper che sfruttano questa tecnica si possono distinguere in base alla tecnica di presa: una tipologia preleva gli oggetti con l'utilizzo di ventose, l'altra invece prevede di sfruttare l'effetto vuoto.

**Figura 2.13:** Ventose a vuoto 40mm, 15mm, 30mm

Il principio di funzionamento è molto semplice: una calotta flessibile di materiale elastomerico viene compressa ermeticamente sulla superficie dell'oggetto permettendo così di prelevarlo e muoverlo. Il vuoto necessario per la presa viene tipicamente riprodotto con pompe, soffietti o con generatori ad effetto Venturi. Sono questi ultimi ad essere maggiormente utilizzati grazie al fatto che tramite una semplice valvola si permette l' istantanea espulsione dell'oggetto.

In figura 2.13 vengono presentati degli esempi di ventose utilizzati per la manipolazione di piccoli oggetti. La presa degli oggetti avviene creando il vuoto all'interno della calotta di materiale morbido contro una superficie liscia o leggermente curva dell'oggetto. Il rilascio dell'oggetto può essere velocizzato producendo un flusso d'aria contrario capace di espellere l'oggetto nella posizione di place.

## 2.6 Comunicazione tra il sistema di controllo e il robot

I dati ricavati dalla camera ed elaborati all'interno del sistema definito in precedenza devono essere opportunamente adattati al formato corretto per l'invio del pacchetto di coordinate al robot. In questa sezione si vuole approfondire il tema della comunicazione tra sistema di acquisizione ed elaborazione dati e il robot che andrà a svolgere il movimento.

L'invio delle informazioni relative alla classe e alla posizione dell'oggetto riconosciuto da parte del sistema sviluppato in Python e la ricezione di queste da parte del robot è resa possibile grazie ad un canale di comunicazione client-server.

### 2.6.1 Comunicazione Client-Server

I sistemi client/server sono architetture di rete nella quale sono presenti due o più terminali con una caratterizzazione gerarchica ben definita. Vi sono uno o più client che si collegano

ad un server per la fruizione di un servizio o per la condivisione di risorse. È il dispositivo server a gestire gli accessi e i permessi ai vari terminali client. La relazione tra diversi dispositivi dell'architettura è un ordinamento che vede dominare il lato server nei confronti del client.

Per stabilire una connessione in un ambiente client/server vi è un software lato client che abilita il dispositivo a spedire una richiesta al server, questa richiesta viene processata e formattata in modo che il server possa riceverla e comprenderla. Dal lato server invece il software deve essere in grado di ricevere la richiesta dal client e di processarla, fornendo poi la risposta al client tramite messaggi di avvenuta ricezione del messaggio, oppure una risposta con delle informazioni. Una volta che la risposta partita dal server giunge al client questa viene riformattata per la lettura da parte del client.

Per una corretta comunicazione tra terminali di una stessa rete è necessario che le macchine utilizzino un linguaggio comune, ovvero un protocollo applicativo capace di codificare e decodificare i messaggi che vengono inviati e ricevuti dai terminali che compongono la rete.

### 2.6.2 Comunicazione TCP/IP

Per l'applicazione in esame, quindi per la comunicazione che deve avvenire tra il sistema di controllo ed elaborazione delle immagini e il sistema robotico di movimentazione, il ruolo del server è ricoperto dal braccio antropomorfo che andrà a svolgere i movimenti di pick, mentre il client è rappresentato dal sistema di acquisizione. Per quasi la totalità delle implementazioni, in questo tipo di progetti per la comunicazione, si sfrutta il protocollo TCP/IP (*Transmission Control Protocol / Internet Protocol*).

Questo modello di architettura di rete garantisce che il messaggio da inviare raggiunga la destinazione prevista. Le informazioni vengono suddivise in pacchetti che vengono poi riassemblati quando il messaggio giunge a destinazione. In questo modo è garantito un alto livello di accuratezza nella trasmissione dei dati in modo che quanto ricevuto corrisponda a quanto inviato.

L'implementazione a progetto di questo tipo di comunicazione avviene tramite l'utilizzo delle socket: strutture software che permettono l'invio e la ricezione di dati tra client e server. A seconda del linguaggio di programmazione utilizzato, questi oggetti vengono inizializzati e sfruttati all'interno del software con una logica che prevede quattro macrofasi per la loro gestione: *open*, *read*, *write* e *close*.

Tra i due interlocutori che inizializzano la comunicazione è fondamentale specificare indirizzo IP e porta da utilizzare nella comunicazione così da indicizzare il processo di invio e ricezione del messaggio. Sia client che server memorizzano indirizzo e porta della controparte in un apposito indirizzo socket per garantire che il messaggio venga recapitato al corretto destinatario.

# 3

# Descrizione del progetto

Il progetto che si propone in questo elaborato prevede l'utilizzo di diversi strumenti software e hardware per il riconoscimento di oggetti in un'area di lavoro: un sistema di visione renderà possibile il riconoscimento, la presa e il posizionamento degli stessi. L'applicazione che si definisce a progetto è quella della raccolta differenziata: vengono svolte delle immagini ad un contenitore di rifiuti diversi caratterizzati da materiali differenti. Il sistema, tramite l' elaborazione di queste immagini, riesce a classificare i prodotti presenti attribuendo loro etichette predefinite. Tramite una logica che si descriverà di seguito viene impostata una priorità ai vari oggetti riconosciuti in modo da selezionare quello che verrà prelevato per primo. Il sistema di visione 3D realizza delle scansioni dell'oggetto selezionato per comprenderne la posizione e l'orientazione nello spazio. Una volta che il sistema ha acquisito i valori delle tre coordinate e dei tre angoli dell'oggetto nel piano di lavoro, le informazioni vengono inviate al robot per iniziare il bin picking del prodotto con priorità massima. Questo, a seconda dell'etichetta che gli è stata attribuita inizialmente, viene posizionato nell'opportuno contenitore relativo alla classe di rifiuto cui appartiene.

## 3.1 Motivazione dell' applicazione alla raccolta differenziata

La scelta di voler implementare questo sistema robotico al settore della raccolta differenziata deriva dal fatto che il tema dello smistamento dei rifiuti è di cruciale importanza all'interno di una società in esponenziale crescita come quella odierna. L' applicazione della computer vision in questo specifico campo può permettere di efficientare i sistemi di riciclo dei prodotti di scarto, portando benefici all'ambiente e alla natura che ci ospita migliorando la qualità della vita.

Il primo vantaggio che ne consegue dallo sviluppo di sistemi di visione avanzati in questo ambito è una maggiore efficienza nella differenziazione della raccolta. I sistemi di visione possono identificare gli oggetti in modo più preciso e rapido rispetto ai metodi tradizionali. È inoltre possibile diminuire gli errori nella classificazione dei prodotti: si mettono a disposizione database contenenti i più svariati prodotti così da acquisire con certezza la tipologia di materiale che compone un certo rifiuto.

Queste migliorie permettono di massimizzare la sostenibilità ambientale aumentando la quantità di rifiuti riciclabili e riducendo l'invio dei materiali di scarto presso le discariche. Si ha inoltre una riduzione di quelli che sono i costi per lo smaltimento dei materiali di scarto: una raccolta differenziata più efficiente riduce i costi legati alla raccolta e al trattamento dei rifiuti, beneficiando sia l'ambiente che l'economia.

## 3.2 Definizione del set di oggetti da riconoscere

Si vuole creare un sistema per il riconoscimento di un range ristretto di rifiuti solidi per svolgerne la raccolta differenziata automatizzata. Nello specifico si prendono in considerazione cinque prodotti di scarto di uso comune elencati di seguito e riportati in figura3.1:

- Lattina di Cocacola ®(cocacola)
- Lattina di Fanta ®(fanta)
- Confezione di Nutella Biscuits ®(nutella)
- Bricchetto di succo monoporzione Santal ®(succo)
- Confezione di chewing gum Vigorsol ®(vigorsol)


**Figura 3.1:** Prodotti scelti per il progetto

La motivazione della scelta di questi articoli ricade sul fatto che sono prodotti di uso comune ipoteticamente in una zona ricreativa di un ambiente scolastico, come ad esempio una sede universitaria. In particolare poi i rifiuti scelti risultano appartenere a diverse classi di raccolta differenziata in modo da poter validare il riconoscimento e il place degli oggetti negli appositi cestini.

## 3.3 Python come linguaggio di programmazione

Il fulcro del progetto consiste nella stesura di codice che, oltre ad implementare le logiche di controllo e gestione dei processi di riconoscimento ed elaborazione delle informazioni relative agli oggetti, deve permettere di interfacciarsi con il sistema di visione e con il sistema robotico Kuka. Per rendere possibile questo è necessario utilizzare un linguaggio di programmazione potente e flessibile, che abbia a disposizione librerie dedicate alla computer vision e che dedichi risorse alla comunicazione con il sistema robotico per l'invio delle caratteristiche di posizione e di orientazione da raggiungere. Si sceglie *Python*.

In qualche modo la scelta di questo linguaggio risulta forzata da alcuni aspetti. Python propone comode interfacce per la gestione del sistema di visione scelto per il progetto: la comunicazione e la gestione delle camere Intel è semplicissima grazie all' utilizzo delle librerie messe a disposizione dal produttore. Per la gamma di fotocamere 3D messe in commercio dalla Intel,

gli sviluppatori rendono possibile utilizzare e settare i dispositivi ottici tramite l'importazione di strutture dati e funzioni pronte all'uso. Nello specifico vengono rilasciate diverse versioni della parte software relativa alle fotocamere a seconda dell'ambiente di sviluppo che l'utilizzatore vuole utilizzare: le più sfruttate in questo ambito sono quelle per Matlab e Python. Si sceglie la seconda delle due in quanto, per il progetto in esame, si dovrà svolgere la comunicazione con il robot Kuka e gestire la logica della rete neurale, tutte implementazioni gestibili all'interno di un unico programma Python.

Un'altra sorte di vincolo che accompagna la scelta di questo linguaggio di programmazione è il fatto che per il riconoscimento degli oggetti si è scelto di utilizzare la rete neurale YOLO v7. Questa viene messa a disposizione come codice open source su GitHub. Per poterla inserire nel progetto la si scarica e la si adatta all'applicazione: la versione scelta viene fornita nel linguaggio Python. Quindi risulta scontata la stesura di un programma per la gestione del progetto completo che parli la stessa lingua delle estensioni inglobate.

## 3.4 Creazione dell'ambiente di sviluppo

Vengono ora descritti i passaggi necessari per la creazione dell'ambiente di sviluppo per il codice Python che gestirà il progetto. Si deve per prima cosa scegliere e creare un ambiente virtuale per lo sviluppo e l'esecuzione del codice del sistema. Grazie alla creazione di un ambiente dedicato, si ricrea una zona protetta capace di contenere tutti i comandi e i pacchetti necessari per l'esecuzione del programma, evitando conflitti con librerie e impostazioni esterne.

In questo progetto si è scelto di utilizzare l'ambiente *Anaconda 3*. È stata scelta questa specifica estensione perchè questa consente di creare ambienti virtuali separati e fornisce la possibilità di specificare la versione di Python e le librerie dedicate. Anaconda inoltre viene fornito con una vasta gamma di librerie preinstallate, molte di queste dedicate al calcolo e all'elaborazione di grandi quantità di dati. Per questo motivo è molto utilizzata nel settore della computer vision quando si programma in Python. Una volta creato l'ambiente è necessario attivarlo.

L'ambiente di sviluppo attivato da Anaconda relativo al progetto lo si denomina con *yolov7_env* e al suo interno si installano tutte le librerie e tutte le specifiche necessarie alla creazione dell'ambiente di sviluppo dedicato partendo dalla scelta della versione di Python da utilizzare: si opta per la versione stabile *Python 3.7.16*.

## 3.5 Struttura del codice

Il codice Python sviluppato è stato diviso in diversi programmi richiamati da un file sorgente utilizzato come *main* del progetto. Si definisce il file *project.py* come codice principale dentro al quale si fa riferimento ai specifici file che compongono le varie fasi del progetto: vi sarà un codice relativo alla gestione della camera e responsabile dell'acquisizione delle immagini a colori e di profondità, una sezione di elaborazione delle informazioni raccolte per la proiezione nello spazio delle coordinate, per la definizione della priorità dell'oggetto da prendere e per la determinazione dei valori di posizione e orientazione degli oggetti riconosciuti, e infine una porzione di codice dedicata alla comunicazione delle informazioni al robot che permetterà il pick degli oggetti. L'elenco di questi sotto programmi viene completamente espresso nel capitolo 5.

### 3.5.1 Flow Chart del sistema robotico

In questa sezione si specifica quello che è il flusso che il programma segue durante l'esecuzione dei vari blocchi di codice definiti.

Si richiamano ora le sezioni in cui viene diviso il programma per presentare la struttura dello scheletro del progetto, nei paragrafi successivi verranno poi approfonditi singolarmente i vari sotto programmi. In figura3.2 viene presentato lo schema completo.

Per prima cosa si necessita di avere una rete neurale allenata per garantire un corretto e preciso riconoscimento degli oggetti nello spazio di lavoro. In questi termini, deve essere eseguito offline prima dell'esecuzione degli altri blocchi, il training della rete YOLO_v7 utilizzata: eseguo il blocco *train.py*. Ora che lo strumento di riconoscimento è pronto, la rete neurale rimane in attesa di ricevere l'immagine da elaborare.

Si inizializzano le sezioni di codice *progetto.py*, dedicato alla gestione delle scansioni e dell'elaborazione delle immagini, e *serverCamera.Java*, per la definizione della movimentazione del robot.

Dal lato del robot viene impostata la posizione di Home. Questa corrisponde alla posizione in cui effettuare le scansioni tramite la camera. Il codice Python per l'elaborazione dei dati arriva in una condizione di attesa e ci rimane fino a quando il server non richiede di effettuare una scansione.

Quando il robot si trova nella posizione di Home, il controller invia la chiamata al programma di visione per il riconoscimento degli oggetti e si mette in attesa delle posizioni e delle informazioni necessarie al pick dell'oggetto.

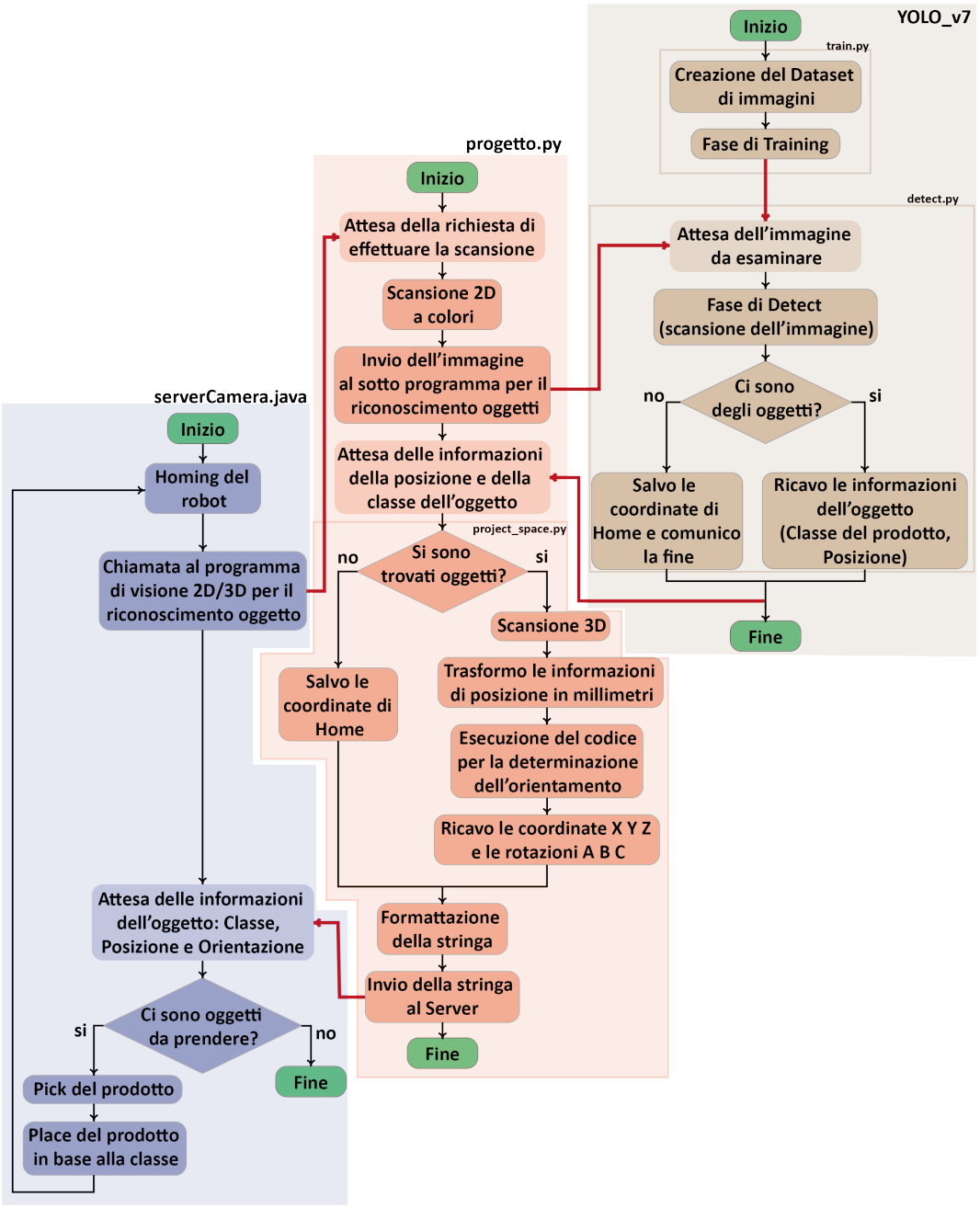



**Figura 3.2:** Flow Chart del sistema

All'interno del codice Python viene quindi inizializzato lo stream da parte della camera effettuando la scansione 2D a colori dello spazio di lavoro. Questo frame viene inviato all'interno della fase di *detect.py* della rete neurale: viene elaborata l'immagine ricercando gli oggetti al suo interno fornendo come output il bounding box e l'etichetta della classe di appartenenza degli oggetti riconosciuti. Le informazioni di posizione vengono memorizzate e all'intento della sezione *project_space.py* viene elaborata la logica per la determinazione dell'orientazione dell'oggetto nello spazio sfruttando le misurazioni tridimensionali fornite dalla camera.

Ora che le sei grandezze nello spazio sono state definite possono essere inviate al server in modo che il software per la movimentazione del robot abbia memorizzato il punto in cui effettuare il place dell'oggetto.

#### Iterazioni del flusso

Se non si trovano oggetti nello spazio di lavoro, si comunica la posizione di Home e la si invia all'interno della stringa al robot assieme alla classe 6 (tipo oggetto). In questo caso, dato che il codice java che gestisce il robot controlla la classe di appartenenza dell'oggetto rilevato, si comunica anche al server che non ci sono più oggetti. La logica implementata permette di uscire e terminare il codice se la scena viene ritenuta vuota. Nel caso invece in cui il sistema rilevi dei prodotti, il codice server viene fatto ciclare ritornando all'istruzione di attesa delle informazioni della posizione di pick per l'oggetto successivo.

## 3.6 YOLO v7: personalizzazione del codice sorgente e Fase di Training

Nel capitolo2 si è anticipato che la rete neurale implementata nel progetto è YOLO nella versione 7. Ora, per un normale utilizzo di questa rete, è necessario clonare il codice sorgente dall'apposita repository GitHub ([21]). Nel progetto che si sta presentando il codice relativo alla rete neurale è stato modificato e adattato alla specifica applicazione. Nello specifico si sono utilizzate porzioni del codice sorgente originale e mantenute tali e quali, altre sono state tolte ed eliminate (come ad esempio le sezioni che si occupano dell'analisi di video, inutile per l'applicazione in esame), altre ancora sono state modificate e integrate con sezioni necessarie per l'estrapolazione delle informazioni specifiche richieste dal progetto.

Quindi da qui in avanti, quando si fa riferimento a YOLO v7 ci si basa sul codice sorgente messo a disposizione dagli sviluppatori proprietari ([21]) ma del quale se ne effettua una versio-

ne personalizzata adattata al progetto protagonista di questa tesi. Per differenziare e riconoscere che si tratta di una versione modificata la si denomina come *YOLO_v7 CUSTOM*.

In questo e in tutti i paragrafi di questo capitolo che fanno riferimento a YOLO si espongono solo le peculiarità della rete utilizzata definendo la sua implementazione durante lo svolgimento e le esecuzioni del progetto in esame. Si è dedicata una maggiore attenzione definendo nello specifico i dettagli teorici di funzionamento in una sezione apposita nel capitolo 2.

La prima fase per l'implementazione dell'algoritmo YOLO è il training convoluzionale. Per comprendere come riconoscere gli oggetti ed individuarli all'interno delle immagini, la rete deve analizzare e studiare una grande quantità di dati, sotto forma di informazioni ricavate da appositi dataset dedicati. In questo caso le informazioni vengono ricavate da immagini degli oggetti da riconoscere: queste devono contenere dettagli utili in modo che la rete riesca a garantire il riconoscimento degli oggetti con una certa solidità.

Va specificato innanzitutto che la fase di training della rete neurale viene svolta a priori rispetto all'esecuzione del programma: come visto nella sezione teorica, l'allenamento viene svolto in fase di sviluppo del progetto in modo che, per l'esecuzione del codice, la rete sia pronta per riconoscere gli oggetti.

### 3.6.1 Acquisizione e labeling delle immagini

Come scelta progettuale si è deciso di creare manualmente il dataset di immagini da fornire come input all'allenamento della rete. Per ciascuna classe di prodotto (vedi paragrafo 3.2) si è scelto di fornire alla rete duecento foto. Cento di queste sono state scaricate da internet mentre le altre cento sono state scattate manualmente. Nelle prime si cerca di selezionare quelle immagini che coinvolgono i singoli oggetti in situazioni reali, con sfondi più o meno complessi o comunque con un contesto, preferendo comunque le immagini che raffigurano il soggetto da riconoscere posto in modo orizzontale e non con orientazioni particolari. In queste raffigurazioni si inseriscono, oltre che alle immagini appena specificate, anche dettagli che rappresentano caratteristiche del prodotto da riconoscere: ad esempio, nel bricco di succo che si vuole riconoscere è presente una raffigurazione di una pesca tagliata in due della quale si può vedere il nocciolo, nelle immagini che compongono il dataset si inseriscono anche immagini che raffigurano in dettaglio il frutto con il dettaglio sopra descritto (Figura 3.3.(a)).

**(a)** *Immagine scaricata da internet* **(b)** *Immagine scattata manualmente*

**Figura 3.3:** Esempio di immagini utilizzate per la creazione del dataset di training


**Figura 3.4:** Screen del programma LabelImg utilizzato per il labeling dei prodotti

Le foto che invece vengono scattate ad hoc per l'applicazione contengono cento frame che racchiudono gli oggetti presi singolarmente, con uno sfondo omogeneo creato su misura per riuscire a collezionare immagini dei prodotti orientati nei modi più diversi ma garantendo un contesto semplice e il più possibile monocromatico. Per fare questo si è creata una scena con sfondi bianchi e si è applicata una luce diretta tramite due diverse sorgenti luminose in modo da evitare eventuali ombre dei prodotti in scena.

Archiviate le immagini è necessario esprimere i dettagli di posizione e la classe di appartenenza dei prodotti contenuti nei frame raccolti: processo di *labeling*.

In questo progetto per creare il database di informazioni si utilizza il programma open sour-

ce *LabelImg*: questo, utilizzando un'interfaccia grafica per la visualizzazione delle immagini, permette di selezionare l'area di interesse e di definire il bounding box dell'oggetto. In figura 3.4 è proposta una schermata del programma utilizzato per la definizione dei contorni per il dataset di training.

Una volta estrapolate le informazioni e memorizzate correttamente in riferimento alle immagini sorgente è necessario svolgere un passaggio intermedio prima di effettuare il caricamento dei file nella scheda dedicata all'allenamento. Si devono adattare le informazioni raccolte contenute nei file *.txt* ad un formato che sia leggibile dalla versione di YOLO utilizzata seguendo le linee guida definite dai pacchetti di calcolo che usa la rete. A seconda della versione e delle estensioni che si utilizzano, oltre che a modificare opportunamente la formattazione delle informazioni raccolte, è richiesto di suddividere i file in apposite directory composte da cartelle e sottocartelle con metodi ben definiti: Training Set, Validation Set e Testing Set (si riporta alla trattazione teorica svolta nel capitolo 2). Per fare questo si sfrutta il sito *Roboflow* [22]. Si tratta di un servizio che permette di effettuare l'upload delle immagini e dei relativi file di testo contenenti le informazioni sopra specificate e di ottenere come risultato un output adatto ad essere sfruttato, grazie alla corretta formattazione, nella lettura da parte della specifica versione di rete neurale scelta. I file quindi vengono caricati e rielaborati secondo le preferenze di progetto, ottenendo le estensioni e le caratteristiche richieste. In particolare le immagini raccolte vengono ridimensionate in immagini di 640 x 480 pixel e suddivise in tre diverse cartelle: 662 immagini vengono selezionate come *Training Set*, 214 come *Validation Set* e le successive 124 raccolte come *Testing Set*.

La fase di allenamento della rete richiede una notevole potenza computazionale e per l' esecuzione di questo codice si necessita di CPU performanti o di GPU da dedicare allo sforzo computazionale. Per adempire a questa necessità è possibile sfruttare macchine di calcolo con caratteristiche interne adatte all'applicazione o, in alternativa, è possibile affidarsi a dei servizi cloud che mettono a disposizione potenza computazionale permettendo di sfruttare server esterni ai quali si rilega l'esecuzione del codice e dei calcoli annessi.

Nel progetto in esame si utilizza il servizio cloud *Google Colaboratory*. Si tratta di un ambiente di sviluppo integrato che permette di scrivere ed eseguire codice Python all'interno di una semplice pagina del browser, senza la necessità di possedere strumenti hardware performanti. Questo servizio fornisce l'accesso a risorse di elaborazione e, come richiesto dalla situazione in esame, ad una GPU per l'elaborazione di dati. Grazie a questa scelta è possibile eseguire i pe-

santi codici relativi all'allenamento della rete neurale da un qualsiasi computer senza richiedere grandi potenze di calcolo hardware.

Nella pagina di Google Colab relativa al training della rete neurale con dataset personalizzato ([23]) si effettua l'upload dei file relativi al dataset di immagini e informazioni creati. A questo punto è possibile eseguire il codice e permettere l'inizio della fase di allenamento della rete. Questo processo è importante in quanto permette di regolare i pesi dei nodi che compongono la rete in modo che questa impari a riconoscere gli oggetti di interesse in immagini diverse da quelle che compongono il database di training.

A livello progettuale si ripete diverse volte questa fase prima di ottenere e validare il modello finale. Come visto nel capitolo 2, questa fase si svolge seguendo la successione di diverse *epoche*. Ciascuna epoca rappresenta un'interazione completa dell'algoritmo di apprendimento.

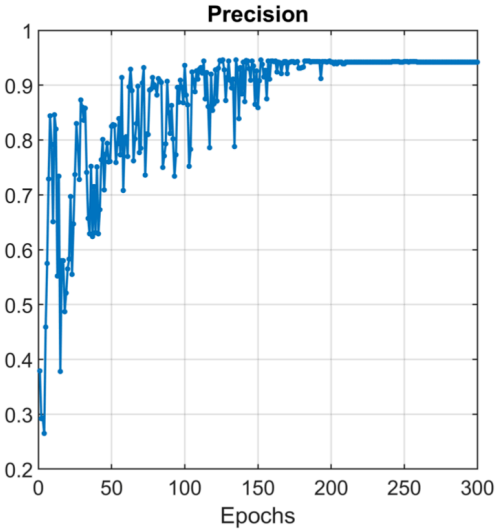
La rete YOLO cerca di adattarsi ai dati di addestramento forniti (immagini e informazioni di posizione ed etichette delle classi) attraverso l'aggiornamento dei suoi pesi e delle sue connessioni. Nel progetto, per la validazione dei pesi in output alla fase di training, si svolge l'esecuzione del codice *train.py* più volte variando di volta in volta il numero di epoche da svolgere. In questo modo è possibile stabilire il numero ottimo di epoche che corrisponde alla rilevazione di pesi con un grado di precisione adatto al progetto, garantendo un buon livello di allenamento ed evitando problemi collegati all'overfitting.

Ad ogni esecuzione del codice di allenamento della rete vengono memorizzati i parametri di ogni singola epoca. Nello specifico si forniscono parametri relativi a diversi aspetti, sia riguardanti la potenza computazionale utilizzata che le nozioni tecniche relative alla comprensione delle immagini. Vengono visualizzate inoltre le metriche per stabilire il grado di precisione a cui si è giunti.

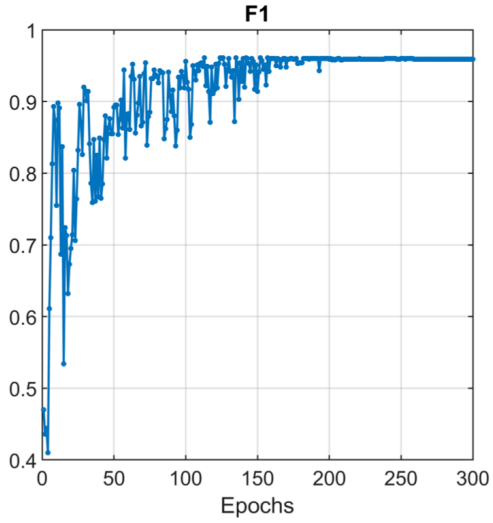
Nel progetto, per definire il numero di epoche ottimo per garantire un buon grado di allenamento della rete ci si riferisce ai valori assunti dai parametri di precisione (P) e di richiamo (*recall*, R) nelle singole epoche. In tabella 3.1 oltre a questi due parametri sono indicati anche i valori di precisione media $mAP@.5$ e $mAP@0.95$. Come visto in precedenza, per considerare una rete ben allenata si devono rilevare dei valori elevati di questi parametri, vicini all'unità.

Per quanto specificato sopra, si vuole evitare un numero di epoche troppo elevato. Si fissa quindi la soglia limite di epoche da eseguire selezionando la prima epoca cui si riscontra avere parametri circa unitari di P ed R. In questo modo si garantisce di avere un buon livello di allenamento della rete e contemporaneamente si evita l'overfitting.

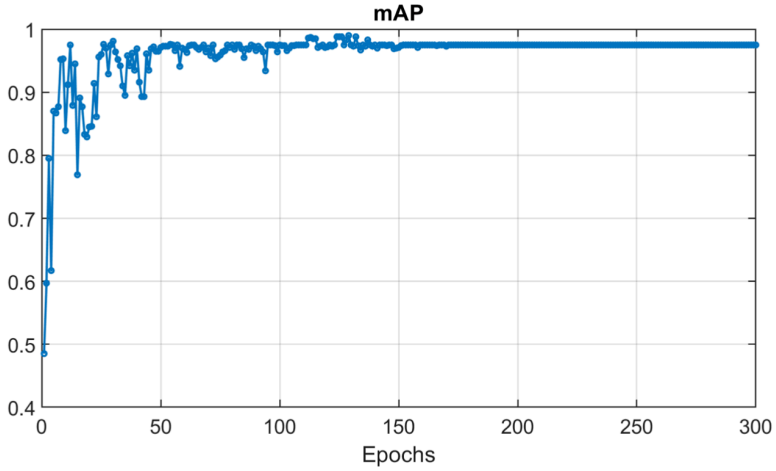
Visti i risultati riportati in figura3.5, nel progetto si sceglie un numero di epoche pari a 200,




(a) *Precision*




(b) *F1*




(c) *mAP, MeanAveragePrecision*

**Figura 3.5:** Andamento dei parametri di training lungo le varie epoche

| epoch | gpu_mem | box | obj | cls | total | labels | img _size |
|---|---|---|---|---|---|---|---|
| 199/199 | 10.6G | 0.03075 | 0.005486 | 0.00173 | 0.03797 | 10 | 640x480 |
| | **Class** | **Images** | **Labels** | **P** | **R** | **mAP@.5** | **mAP@.95** |
| | all | 214 | 232 | 0.838 | 0.811 | 0.864 | 0.572 |
| | cocacola | 214 | 35 | 0.898 | 0.755 | 0.895 | 0.57 |
| | fanta | 214 | 44 | 0.88 | 0.841 | 0.919 | 0.544 |
| | nutella | 214 | 47 | 0.778 | 0.744 | 0.813 | 0.564 |
| | succo | 214 | 38 | 0.71 | 0.789 | 0.742 | 0.486 |
| | vigorsol | 214 | 68 | 0.926 | 0.925 | 0.95 | 0.694 |

**Tabella 3.1:** Valori di output della rete neurale all'epoca 200 della fase di training

ottenendo all'ultimo ciclo i parametri elencati in tabella 3.1. Si rimarca come, per giungere a questo risultato, sia stato fondamentale svolgere la fase di training numerose volte accantonando tutti quei risultati che portassero con se errori relativi o ad una non adatta accuratezza o d'altra parte all'overfitting della rete.

Da questa prima porzione di codice si ricava il file denominato *best_200.pt* contenente i pesi ottimi relativi ai nodi che compongono la rete neurale e che verrà utilizzato per il riconoscimento degli oggetti durante l'esecuzione del progetto nel processo di *detect*.

## 3.7 Il codice di progetto Python: *project.py*

Svolta e conclusa la fase di training è possibile passare all'esecuzione vera e propria del codice che compone il progetto. Si compila quindi il file *project.py* definendo delle righe di codice che fanno riferimento all'esecuzione di diversi altri programmi:

- **detect**: vengono acquisite delle immagini a colori della scena e viene eseguito il riconoscimento degli oggetti contenuti nell'area memorizzandone l'etichetta della classe di appartenenza e le informazioni di posizione sotto forma di pixel;
- **project_space_multiple_frame**: vengono realizzate delle rilevazioni di profondità per riuscire a tradurre le coordinate in pixel in misurazioni nello spazio rispetto ad un origine prefissata, vengono inoltre ricavate le orientazioni dell'oggetto;
- **communicate_to_server**: le informazioni relative alle tre posizioni e alle tre inclinazioni nello spazio del prodotto caratterizzato da priorità massima vengono comunicate al robot per permettere l'azione di pick.

### 3.7.1 Riconoscimento degli oggetti: *detect.py*

Questa sezione è dedicata al riconoscimento degli oggetti grazie all'utilizzo della camera 3D utilizzata per l'acquisizione di frame RGB e l'elaborazione delle immagini tramite l'esecuzione della fase di detect della rete YOLO v7.

Per permettere una corretta esecuzione di questa porzione di codice è fondamentale avere nell'apposita directory il file *best_200.pt* contenente i pesi ottimizzati dei nodi della rete ricavati dal training. È altrettanto necessario avere collegata al sistema la fotocamera Intel scelta.

Vengono per prima cosa inizializzate le strutture dati nelle quali si andranno a memorizzare i dati relativi agli oggetti e vengono implementati i modelli e tutti i parametri necessari all'accensione e alla gestione della camera oltre che a definire i criteri richiesti dal detect della rete.

Viene attivato il collegamento con la telecamera, viene acquisito un primo frame RGB e lo si salva nell'apposita directory come *image.jpg*. Per fare questo si richiama la classe che permette di catturare un'immagine tramite il sistema di visione collegato, definita dal codice implementato in *realsense_camera.py*: la funzione *capture_image* permette di attivare il collegamento con la camera e di ricavarne un fotogramma a colori. Quest'immagine è il frame che viene ora preso come input dalla fase di detect della rete neurale.

La logica di riconoscimento, di cui si è analizzato il funzionamento nel capitolo 2, permette di ricercare gli oggetti precedentemente definiti nell'immagine che gli si dà come input. Tale logica, per ciascun oggetto riconosciuto, permette di definire un bounding box che racchiuda l'oggetto, attribuisce un'etichetta che definisce la classe di appartenenza e aggiunge una percentuale che specifica il grado di precisione con cui la rete ha assegnato l'etichetta a quel bounding box.

Viene quindi indirizzata l'immagine scattata dal sistema di visione alla rete neurale per il riconoscimento di tutti gli oggetti in essa presenti. Si sottolinea come, in una prima fase di test svolta durante la progettazione del codice, si sono svolte diverse prove per validare il modello di rete neurale verificando un adatto allenamento dello stesso: sono state condotte diverse simulazioni di riconoscimento mettendo in scena diverse situazioni, prima prendendo gli oggetti singolarmente e orientandoli secondo diverse angolazioni e a diverse distanze dall'obiettivo, poi accartocciando leggermente i prodotti per verificare che, anche se modificandone la forma, il sistema li riconosca con una certa precisione; infine si sono svolte delle prove per valutare il riconoscimento degli oggetti quando questi sono disposti vicini o comunque accatastati e mescolati tra loro. Dopo queste prime valutazioni è stato possibile confermare che la rete risulta

produrre degli output caratterizzati da una buona accuratezza per l'applicazione in esame.

Si ha ora a disposizione un fotogramma elaborato dalla rete neurale del quale si conoscono i prodotti riconosciuti oltre che le dimensioni e le posizioni dei bounding box ad essi associati.

Si specifica che, d'ora in avanti per la definizione della posizione degli oggetti nello spazio, partendo da informazioni in pixel e successivamente riportandole nello spazio, per ciascun rifiuto, si parte dalle coordinate del bounding box definite come un rettangolo che avvolge l'oggetto riconosciuto e si ricava il centro di questi contorni tracciando le diagonali del rettangolo, prendendo come riferimento della figura l'intersezione di queste due linee. Per tutta la successiva esecuzione del programma quindi, quando si farà riferimento al concetto di posizione dell'oggetto, ci si rifà alle coordinate relative al centro bounding box definito attorno all'oggetto stesso.

Tornando al frame e ai dettagli rilevati, le informazioni riferite ai rifiuti individuati vengono sintetizzate all'interno di una prima struttura dati definita come *data_output* di dimensione 100x5. Ciascuna riga di questa matrice sarà legata ad un singolo oggetto riconosciuto dalla rete, ipotizzando di riconoscere un massimo di 100 prodotti. Le caselle di questa struttura sono allocate secondo i seguenti dettagli:

- [:,0]: Il primo intero di ciascuna riga della matrice è riservato al concetto di priorità tra gli oggetti. È un aspetto che verrà preso in considerazione più avanti, ma si anticipa che al robot verranno inviate le informazioni relative ad un unico oggetto del quale si attribuisce una preferenza di priorità sugli altri in modo che sia il primo ad essere preso dal braccio robotico. Questa preferenza è fornita all'oggetto che risulta avere la distanza minore dall'obiettivo tra tutti quelli riconosciuti: si associa il valore 1 all'oggetto da prelevare per primo, 0 a tutti gli altri.

- [:,1]: La seconda posizione delle righe della matrice è dedicata alla classe a cui appartiene il prodotto riconosciuto e a cui fa riferimento la riga in esame. In particolare si sono definiti i valori definiti in tabella 3.2. In questo modo risulta di maggiore efficacia e comprensione l'elaborazione delle informazioni degli oggetti.

- [:,2] e [:,3]: La terza e la quarta casella di ciascuna riga corrispondono alla posizione in pixel del centro del bounding box dell'oggetto a cui fanno riferimento.

| Rifiuto individuato | Intero Assegnato |
|---|---|
| cocacola | 1 |
| fanta | 2 |
| nutella | 3 |
| succo | 4 |
| vigorsol | 5 |

**Tabella 3.2:** Codifica dei rifiuti in interi nella struttura dati *data_output*

- [:,4]: La quinta e ultima colonna della matrice corrisponde al valore intero di profondità misurato dalla fotocamera 3D inteso come centimetri di distanza dalla focale del sistema di visione.

La rete neurale elabora l'immagine e riconosce gli oggetti presenti in essa. Il sistema ne memorizza e ne codifica la classe di appartenenza, stabilisce il punto al centro del bounding box e salva le sue coordinate impostandole come coordinate dell'oggetto stesso.

Come specificato nella sezione teorica, il sistema di riferimento in pixel pone l' origine nell' angolo in altro a sinistra, quindi, per una futura elaborazione da parte del sistema di movimentazione, sarà necessaria una trasformazione.

Al momento si accetta di mantenere l'informazione sotto forma di pixel in quanto in questa fase è sufficiente memorizzare la posizione precisa dei rifiuti all'interno dell'immagine, prescindendo dal sistema di riferimento a cui si fa riferimento.

Una volta ottenuto l'elenco di tutti gli oggetti, il sistema di visione effettua una scansione 3D della situazione che sta inquadrando memorizzando per ciascun pixel un dato di profondità inteso appunto come distanza dall'ottica di visione. Si genera in questo modo una matrice delle dimensioni dell'immagine dove per ciascuna casella si memorizza il valore della distanza dalla focale. Di questa struttura si sfruttano i valori corrispondenti alle posizioni dei centri dei vari oggetti riconosciuti. In questo modo è possibile affiancare delle informazioni relative alla coordinata Z del sistema.

Come espresso in precedenza deve essere fornita una logica di priorità tra gli oggetti riconosciuti nello spazio. Per fare questo, a livello progettuale si è scelto di attribuire la massima importanza all'oggetto che risulta più vicino alla fotocamera che ha svolto le misurazioni. Si imposta quindi come oggetto preferito quello caratterizzato da un valore di profondità minore.

Se finora la struttura dati utilizzata fa riferimento a tutti gli oggetti contenuti nel piano di lavoro, d'ora in avanti si isola l'elemento prioritario scelto per il prelievo. Questa decisione è motivata dal fatto che il sistema di movimentazione prevede di prelevare un solo oggetto per volta e ad ogni ciclo macchina si deve tener presente che sono possibili spostamenti anche degli oggetti vicini a quello prelevato o altre dinamiche non considerate. Per evitare questo tipo di problematiche si decide di gestire un rifiuto per volta, effettuando ad ogni ciclo una nuova scansione ed elaborazione delle immagini.

A questo punto è terminata l'esecuzione della porzione di codice dedicata al detect degli oggetti. Si ritorna al programma principale memorizzando i dettagli di posizione degli oggetti.

### 3.7.2 Proiezione del punto nello spazio: *PROJECT_IN_SPACE* - 1

Le informazioni archiviate nella struttura dati *data_output* vengono ora filtrate per raggiungere i dettagli relativi all'oggetto con priorità massima. In questo modo è possibile proseguire con la logica di pick del sistema indirizzando l'attenzione sul primo rifiuto da raccogliere. Come sottolineato in precedenza, l'oggetto al quale si dà la priorità è quello caratterizzato da una distanza minima dal sistema che acquisisce le immagini. Questo viene evidenziato a livello software con il flag definito nel primo parametro di ogni riga nella struttura.

Dalla struttura dati completa comprendente tutti gli oggetti e le informazioni ad essi connesse, si seleziona la riga corrispondente al primo rifiuto selezionato per la raccolta, memorizzando di default il valore della priorità e la classe di rifiuto a cui appartiene l'oggetto.

#### Definizione del sistema di riferimento della camera

Un' importante sezione di questo codice racchiude la definizione del sistema di riferimento nello spazio della camera. Si sottolinea come sia necessario definire un' origine dello spazio tridimensionale come punto d'incontro degli assi e come questi debbano aver fissata un'opportuna orientazione. Questo concetto rappresenta un aspetto fondamentale prima di tutto per riportare nello spazio le coordinate finora espresse in pixel, ma certamente anche per definire un sistema di riferimento di partenza. Poi, tramite opportune traslazioni e rotazioni, ci si riporta al sistema di riferimento del robot per la comunicazione delle posizioni e delle orientazioni necessarie al sistema che effettuerà il pick degli oggetti.

Per prima cosa è necessario quindi definire gli assi definendone una loro orientazione e un'origine. Si fissa un pixel nell'immagine acquisita scegliendolo opportunamente in modo che visivamente vi sia margine dai bordi e possano essere fatte delle considerazioni per comprendere come si sviluppino le direzioni delle coordinate nel suo intorno. Si memorizzano le coordinate in pixel del punto scelto e di questo preciso punto della matrice viene ricavata la profondità tramite l'apposita funzione.

Come specificato nel capitolo precedente, è necessaria la traduzione delle informazioni di posizione memorizzate in pixel in un sistema di riferimento tridimensionale basato sulle classiche coordinate XYZ. Per svolgere questa codifica nel progetto che si sta descrivendo vengono sfruttate funzioni e proprietà messe a disposizione dalla libreria Python proprietaria Intel. Il punto definito come origine viene quindi proiettato nello spazio tramite la funzione *rs2_deproject_pixel_to_point()*.

### Definizione del punto di pick nello spazio

Una volta definita l'origine del sistema degli assi di riferimento della camera è possibile riferire tutte le future misurazioni come distanza relativa da questo punto e i vari punti individuati nello spazio. Le misurazioni che si svolgeranno sono basate sull'individuazione del punto d'interesse nella matrice di pixel, al quale si riferisce una quota di profondità e alla sua proiezione nello spazio tramite la funzione sopra citata. Per utilizzarlo in relazione al sistema di riferimento appena creato si svolge la differenza delle coordinate del punto misurate con quelle relative all'origine: in questo modo si definisce una relazione in termini relativi delle coordinate, parametrizzando tutti i valori rispetto all'origine degli assi.

Questo nel progetto è stato possibile definendo una traslazione delle coordinate a cui fa riferimento la misurazione, andando a sottrarre le quote relative all'origine:

$$x = punto[x] - x_0 \qquad y = punto[y] - y_0 \qquad z = punto[z] - z_0$$

Dove $x_0$, $y_0$ e $z_0$ sono le coordinate dell'origine nello spazio. La gestione delle posizioni dello spazio definita tramite questo modulo che passa attraverso il concetto di distanze relative tra due punti è fondamentale nel progetto. Va infatti fissato con precisione un punto nello spazio che possa essere considerato come punto di origine per i due sistemi che dovranno comunicare: sia il sistema di visione che il braccio robotico devono partire da uno stesso punto per riuscire a scambiarsi informazioni sui punti dello spazio di lavoro.

| | |
|---|---|
| *signal_output*[0] | PRIORITÀ |
| *signal_output*[1] | CLASSE DEL PRODOTTO |
| *signal_output*[2] | X |
| *signal_output*[3] | Y |
| *signal_output*[4] | Z |
| *signal_output*[5] | A |
| *signal_output*[6] | B |
| *signal_output*[7] | C |

**Tabella 3.3:** Definizione dei campi della struttura dati utilizzata per i dettagli del picking

Si compila quindi una struttura dati definita da un vettore lineare di 8 elementi. Di questi il primo e il secondo sono già stati espressi come rispettivamente il parametro di priorità e l'etichetta collegata all'oggetto riconosciuto dalla rete neurale. Le successive 3 posizioni del vettore sono composte dalle coordinate nello spazio del punto definito come centro dell'oggetto espresse come distanze in centimetri tra questo punto e quello definito come origine. Nella tabelle3.3 si esprimono i dettagli di una struttura denominata con *space[ ]*. Questa non è altro che una variabile contenente le tre coordinate nello spazio definite in modo assoluto per la telecamera e ricavate dalla misura in pixel. Questo array di valori, per quanto riguarda le coordinate di posizione, viene ricavato dalla funzione richiamata dalla libreria Intel specificata in precedenza per la proiezione di un punto nelle coordinate cartesiane. All'interno del codice si è dovuto riporre attenzione alle direzioni degli assi nel passaggio tra i diversi sistemi di riferimento: la definizione degli assi definita nel progetto non coincide con la codifica utilizzata nelle funzioni utilizzate dalle librerie proprietarie. È quindi necessario invertire l'ordine dando a ciascuna coordinata l'effettivo significato e definire le coordinate nell'ordine espresso dal sistema cartesiano di riferimento (x, y, z) riportate nella corretta orientazione.

In questo modo si è definita la posizione nello spazio dell'oggetto da prelevare: queste saranno le prime informazioni che sarà necessario inviare al robot per la comunicazione del punto nello spazio da raggiungere per il prelievo dell'oggetto.

### 3.7.3 Determinazione delle orientazioni di pick nello spazio: *PROJECT_IN_SPACE* - 2

La logica utilizzata per la determinazione dell'orientazione è articolata e completamente progettata e definita ad hoc per l'applicazione che si sta creando. Come concetto base vi è la memorizzazione di informazioni di profondità ricavate dalla camera 3D e una loro analisi ed elaborazione

per determinare come, a partire dal centro dell'oggetto, sia orientato nello spazio il rifiuto sotto esame.

Per la definizione di queste informazioni si è svolto un lavoro di progettazione e studio di una logica dedicata. Nello svolgimento e nella messa in pratica delle idee ci si è poi dovuti scontrare con delle problematiche derivanti dai limiti della strumentazione utilizzata. Di seguito si approfondirà il metodo utilizzato, spiegando i ragionamenti e il percorso fatto per arrivare alla conclusione progettuale, per descrivere infine come sono state affrontate le problematiche incontrate riportando e giustificando le varie soluzioni.

Si anticipa come nel percorso che ha portato all' algoritmo proposto ci sia stata una prima fase in cui ci si è concentrati sulla documentazione presente in letteratura osservando quali potessero essere i metodi per la determinazione della *pose estimation* messi in atto in altri progetti. Si è poi scelto di implementare una logica customizzata, adattata al progetto e ai prodotti utilizzati.

### Logica di base

Come principio per la determinazione dell' orientazione dei rifiuti nello spazio ci si basa sugli strumenti utilizzati e messi a disposizione finora: si sfrutta la camera 3D in relazione con i risultati ottenuti dalla rete neurale. La logica che struttura il codice in questa fase fa riferimento a misurazioni di profondità, opportunamente mirate per la determinazione prima di tutto dell'asse principale dell' oggetto da raccogliere e successivamente per la derivazione dei tre angoli nello spazio.

Come punto di partenza si preleva il centro dell' oggetto individuato precedentemente mantenendone le informazioni di posizione sotto forma di pixel. Si definisce un pattern di 60 punti di forma circolare che viene a distribuirsi attorno al centro con un raggio di 12 pixel. Come mostrato in figura3.6, ciascun punto è generato da un angolo incrementale di 6°. Per ciascuno di questi 60 punti, definiti sotto forma di pixel all'interno dell'immagine, se ne ricava una misurazione di profondità sfruttando la matrice *depth_frame* di dimensione 640 x 480.

Il concetto è quello di memorizzare il punto corrispondente al più alto valore in termini di altezza Z (quindi con profondità minima) come punto massimo e lo stesso per il punto dell'array che è caratterizzato da un' altezza minima. In questo modo si selezionano i due punti, massimo e minimo, che se collegati rappresentano la direzione dell'asse principale dell' oggetto proiettato nella superficie fotografata. Questo segmento poi verrà proiettato nei tre piani che definiscono lo spazio, se ne determineranno i tre angoli che ne descrivono l'orientazione nello spazio in relazione al sistema di riferimento della camera. Da questi poi, tramite opportune

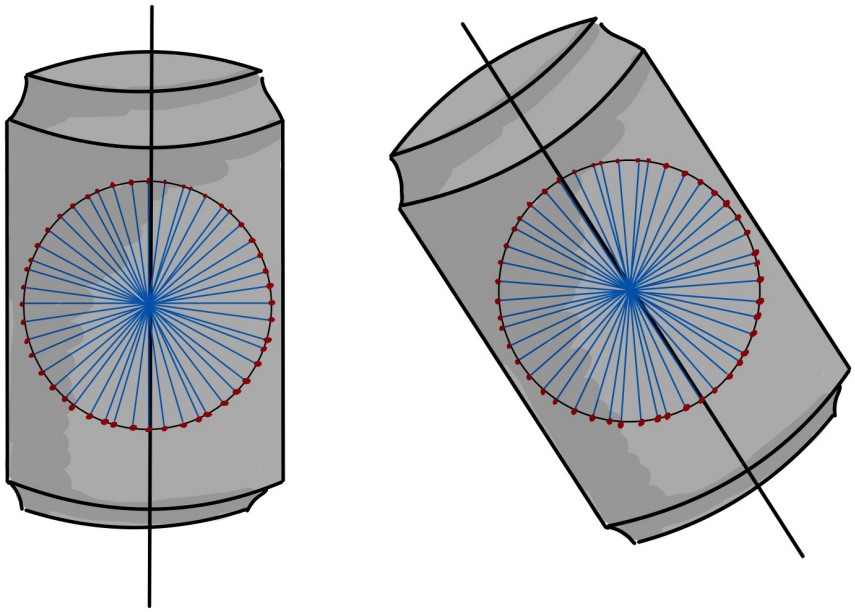


**Figura 3.6:** Dettaglio di come è stato definito il pattern di punti per la determinazione dell'asse dell'oggetto

considerazioni e trasformazioni nello spazio, se ne ricaveranno i valori delle rotazioni richieste dal robot per il pick del pezzo.

Per ciascuno dei punti appena definiti vengono quindi memorizzate le coordinate in pixel e il dato di profondità. Confrontando questo ultimo valore di distanza dalla focale per tutti i punti individuati nella circonferenza, si memorizza il punto più alto tra tutti quelli nel pattern e quello diametralmente opposto che, vista la logica e le geometrie degli oggetti da riconoscere, corrisponderà al punto più basso. Si definisce così il segmento che ha la stessa orientazione dell'asse dell'oggetto e che poggia sulla sua superficie esterna. Da questo se ne ricaveranno le rotazioni da inviare al robot per raggiungere l'oggetto partendo dalla proiezione del segmento nei tre piani dello spazio, figura3.7.

### Problematica 1: Precisione nelle misurazioni di profondità

Durante l'implementazione della fase di misurazione delle profondità che si sta descrivendo si sono incontrate alcune problematiche.

La camera 3D scelta, come visto tra gli aspetti teorici presentati nel capitolo 2, risulta essere adatta per applicazioni di visione che prevedono di catturare scene in un range di distanza tra i 20 centimetri fino ai 4 metri. Operativamente parlando e avvicinandosi a quella che è l'appli-

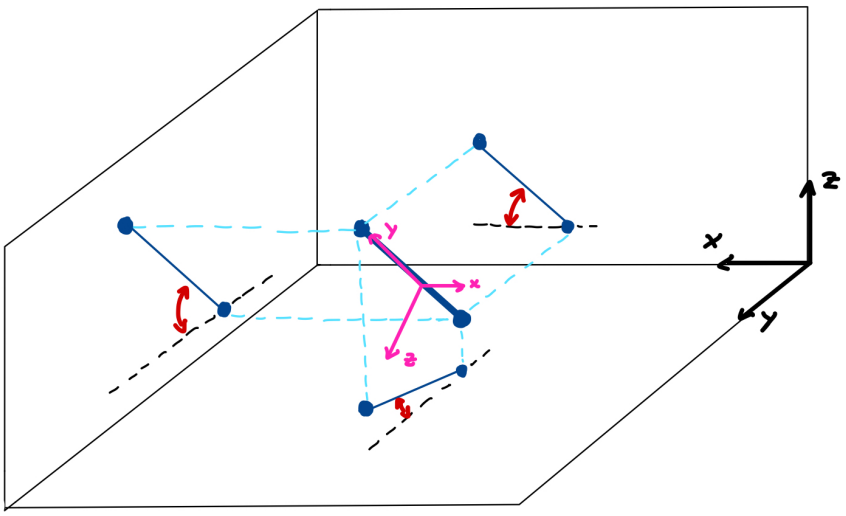



**Figura 3.7:** Asse principale dell'oggetto nello spazio e sua proiezione nei piani

cazione in esame, si devono fare delle considerazioni prestando attenzione alle caratteristiche della camera in relazione alle dimensioni fisiche degli oggetti.

Partiamo sottolineando che le immagini che si scattano ed elaborano sono di dimensione 640 x 480 pixel: questo deriva dal fatto che si ricerca una certa fluidità nell'elaborazione evitando appesantimenti dovuti a risoluzioni più alte. Concettualmente, se si sceglie uno stesso soggetto da fotografare, in un' immagine scattata da vicino, l'oggetto risulterà grande e comprenderà molti pixel. Se invece l'immagine viene fatta ad una distanza elevata rispetto al piano dell'oggetto, questo, all'interno del frame, risulterà più piccolo e andrà ad interessare un numero molto più ridotto di pixel.

Sottolineando nuovamente come la proiezione nello spazio delle posizioni degli oggetti avviene passando da informazioni raccolte in pixel, è importante comprendere come la precisione e l' accuratezza delle informazioni spaziali dipenda dal rapporto specifico tra pixel e centimetri nello spazio di lavoro. Un centimetro nello spazio può corrispondere ad una frazione di pixel se l'immagine è scattata da distante, oppure può corrispondere anche a diversi pixel se l'immagine viene fatta da vicino. È per questo fondamentale fissare la posizione in cui la camera svolge le scansioni ad una distanza ragionata dal piano in cui sono appoggiati i rifiuti: si vuole garantire una certa qualità nelle informazioni spaziali che si andranno a raccogliere.

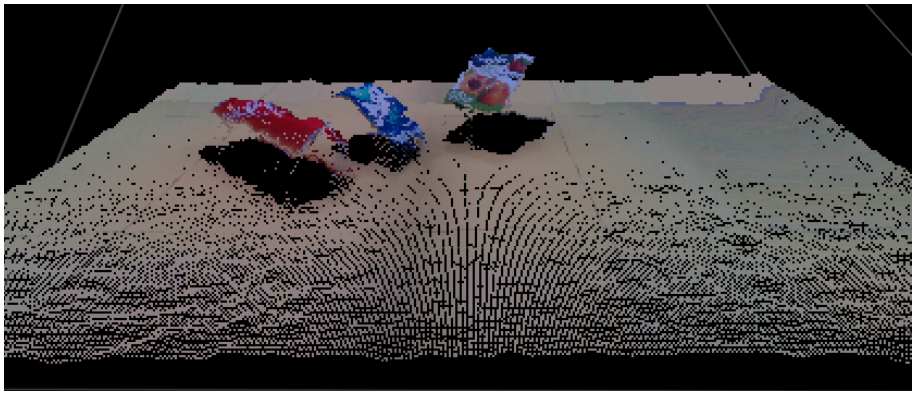


**Figura 3.8:** Scansione 3D con PointCloud che rileva il disturbo di misura delle profondità

Un' altra considerazione da fare è legata ad un aspetto di disturbo delle misurazioni di profondità, approfondita nel paragrafo successivo. Se ci si avvicina troppo agli oggetti e se ne ricavano le informazioni da distanze troppo ridotte entra in gioco una componente di disturbo e quindi di errore di misura intrinseca allo strumento che si sta utilizzando. Come garantito dalla manualistica della camera, rimanendo a distanze leggermente maggiori dei 25-30 centimetri dagli oggetti il peso relativo di questa imprecisione diminuisce, fornendo una misura più precisa.

In relazione alle dimensioni degli oggetti da riconoscere e al grado di precisione scelto per le misurazioni si definisce una posizione per le acquisizioni delle immagini ad una distanza di 450 millimetri dal piano di lavoro.

### Problematica 2: Disturbo nelle misurazioni di profondità

La camera Intel RealSense D435 utilizzata nel progetto ricava le informazioni di profondità da misurazioni date da due sensori ottici che sfruttano tecnologia stereoscopica. Per migliorare la precisione delle misurazioni il sistema è dotato poi di un proiettore a raggi infrarossi. I dati acquisiti vengono elaborati all'interno del microprocessore integrato e inviati poi ai processi che ne richiedono l'utilizzo. L' utilizzo di questi dati, a seconda dei progetti, può avvenire tramite diverse interfacce o librerie a seconda del linguaggio di programmazione utilizzato. Se si utilizza il software SDK fornito dalla ditta proprietaria Intel si ha la garanzia che tutti i valori misurati subiscano una gestione postprocessing e quindi vengano opportunamente filtrati per essere correttamente letti. Se invece, come nel progetto in esame, i valori vengono estrapolati e implementati a progetto tramite l' utilizzo delle librerie Python, non si ha la completa garanzia che questi valori siano adeguatamente precisi e accurati proprio per il fatto che non viene effettuata alcuna manipolazione dei dati ricavati.


**Figura 3.9:** Proiezione delle misure dell'asse su un'immagine RGB a colori

Stando a quanto appena anticipato, durante lo sviluppo del codice per la determinazione delle orientazioni dell'oggetto, si sono incontrate alcune criticità. In particolare, effettuando scansioni video dell' area di lavoro con la camera Intel, si è potuto osservare come superfici che nella realtà sono completamente lisce senza nessun tipo di curvatura, agli occhi della camera, in seguito alla misurazione delle profondità, queste risultino avere valori di quote oscillanti per misurazioni svolte in istanti temporali successivi come riportato in figura3.8.

A livello progettuale si procede con una logica che accetta questa imprecisione nella misura e la corregge a livello computazionale. Si tiene in considerazione il fatto che per frame temporalmente successivi l' acquisizione della posizione di uno stesso punto possa dare valori diversi di profondità. L' errore dovuto al rumore si rende però minimo andando ad effettuare delle misurazioni ridondanti di tutti i punti appartenenti al pattern circolare attorno al centro dell'oggetto in istanti di tempo diversi. Per fare questo, nella misurazione delle profondità, invece che generare un solo cerchio di 60 punti, si genera una matrice composta da 60 colonne (una per ogni punto del cerchio) e 9 righe (una per ogni ciclo di misurazioni), creando quindi 9 cerchi di punti sovrapposti tra loro. Per ciascun cerchio di punti si svolge la stessa misurazione di profondità ma, per ogni cerchio, si ritarda la misura di 0.2 secondi.

L'effetto che ne consegue nelle misurazioni è possibile visualizzarlo in figura3.9: si proietta in un'immagine RGB l'orientazione della definizione dell'asse dell'oggetto, utilizzando il metodo presentato, svolgendo diversi cicli di misurazione si osserva come queste possono avere anche un'ampiezza di errore importante. Per misurazioni della stessa grandezza, svolte nei medesimi punti, si ottengono valori di profondità diversa e quindi orientazioni diverse del segmento. Si conferma quindi la presenza di rumore nelle misure.

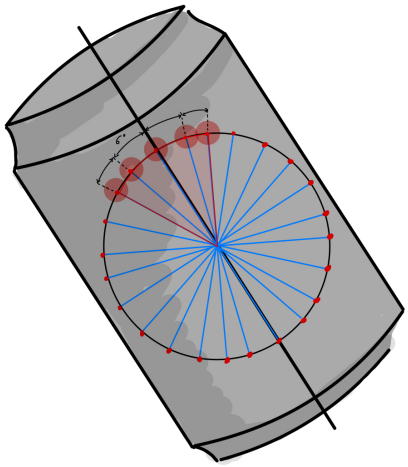


**Figura 3.10:** Problematica nella misurazione della profondità: stessa quota per punti vicini

Per rendere minima l'imprecisione e minimizzare l'errore dovuto al rumore di misura si decide di calcolare la media di tutte le misurazioni: in questo modo se l'errore affligge la misura in verso positivo o negativo, a seconda dell' istante temporale in cui viene effettuata la misura, questo viene in qualche modo compensato dalle misurazioni successive.

### Problematica 3: Quote identiche per punti vicini

Un' altra problematica incontrata nella determinazione delle orientazioni degli oggetti è stata quella che per punti del pattern vicini i valori di profondità risultassero identici.

Gli oggetti presi in considerazione per il progetto presentano superfici molto regolari e molto spesso planari. Questa caratteristica porta a delle imprecisioni nella determinazione dell'orientazione dell'asse principale dell'oggetto. La logica che definisce gli angoli di orientazione si basa sulla misurazione delle quote di profondità del pattern di punti e sulla definizione dell'asse partendo dal punto caratterizzato da una quota di profondità massima. L'acquisizione di posizioni di punti vicini tra loro può però dare come risultato profondità identiche e quindi non aggiornare il punto considerato come il più vicino alla camera tra tutti quelli che compongono il pattern circolare. In questo caso la logica proposta risulterebbe quindi inefficace.

Per sviare a questa problematica, per la determinazione dell'asse dell' oggetto, si controlla il valore della quota di un singolo punto: se questa risulta essere maggiore al valore massimo memorizzato, si aggiorna il valore con la quota corrente, andando a memorizzare l'angolo tra l'orizzontale e il segmento che collega il centro dell'oggetto al punto in questione. Si passa quin-

di al controllo della quota del punto successivo, cioè quello adiacente. Se la profondità relativa a questo punto risulta essere uguale alla quota precedente, viene aggiornato il valore della variabile *angolo_ingrease*: questo parametro tiene conto dell'ampiezza dell'angolo, a multipli di 6°, corrispondente al settore circolare individuato dai punti successivi che forniscono valori di profondità uguali, evidenziato in rosso nella figura3.10.

In questo modo l'angolazione dell'asse principale dell'oggetto è determinata partendo dal valore dell' angolo corrispondente al primo punto a quota Z maggiore al quale si aggiunge metà del valore assunto dal *angolo_increase*. Così è garantita una maggiore precisione nella determinazione dell'angolazione dell' oggetto nello spazio.

### Definizione degli angoli nello spazio e loro significato

Come già definito, le informazioni che vengono elaborate dal sistema di visione 3D comprendono il punto in cui dovrà essere svolto il pick del pezzo, espresso nelle 3 coordinate spaziali XYZ, relativo al centro del bounding box individuato dalla rete neurale.

Per quanto riguarda le orientazioni nello spazio da inviare al robot, per effettuare l'avvicinamento con le opportune inclinazioni, occorre specificare come si arriva a determinare gli angoli RX, RY, RZ da comunicare al robot partendo da dei dati che esprimono l'angolo con cui è inclinato nello spazio il segmento determinato come asse principale dell'oggetto.

Si è visto che la logica di base per la determinazione degli angoli si basa sulla definizione di un segmento nello spazio, appartenente alla superficie dell'oggetto e parallelo all'asse dello stesso. Questo poi viene proiettato nei tre piani nello spazio e se ne determinano i 3 angoli con cui questo è inclinato rispetto al sistema di riferimento definito. A partire da questi angoli, è stato possibile arrivare ad ottenere i valori di rotazioni attorno agli assi da trasmettere al robot per riuscire a far coincidere il sistema di riferimento della pinza con quello dell'oggetto e permettere quindi il pick con le corrette orientazioni.

L'analisi che si è svolta si concentra sul segmento orientato nello spazio del quale si sono definite nello spazio le coordinate XYZ relative ai suoi estremi: il *punto_A* è il punto caratterizzato da una quota più alta mentre il *punto_B* è il punto diametralmente opposto con quota più bassa.

I due punti nello spazio verranno ora proiettati nei tre piani per determinare poi, con l'utilizzo di calcoli trigonometrici, l'angolazione relativa al segmento proiettato rispetto all'asse orizzontale preso come riferimento. Partendo da questi angoli, seguiranno poi alcune elabora-

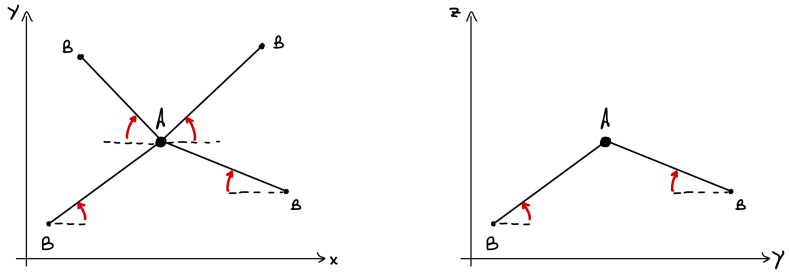



**Figura 3.11:** Casistiche di come può essere rivolto l'asse proiettato nei piani XY e YZ dello spazio

zioni per giungere ai valori di rotazione $\alpha$, $\beta$, $\gamma$ da comandare al robot per far corrispondere i sistemi di riferimento.

Nel codice si sono descritti come *theta_xy*, *theta_xz* e *theta_yz* i tre angoli ricavati dalla proiezione del segmento e dal calcolo dell'angolo rispetto all'origine tramite l'utilizzo dell' arcocoseno. Per quanto riguarda invece la definizione delle rotazioni degli assi da inviare al robot per la manipolazione del sistema di riferimento tool, ci si basa sui valori definiti dal robot Kuka utilizzato che nomina rispettivamente le rotazioni $R_z(\alpha)$, $R_z(\beta)$ e $R_z(\gamma)$ con le lettere A, B, e C.

Si sottolinea inoltre come sia fondamentale comunicare gli angoli calcolati nel corretto ordine in quanto le rotazioni, nel sistema robotico utilizzato, sfruttano la notazione di Cardano: la prima rotazione avviene attorno all'asse Z, poi si imprime una rotazione attorno all'asse Y e infine si fa ruotare l'asse X.

**Rotazione A** - $R_z(\alpha)$.
Iniziamo con l'analizzare la rotazione che deve essere impressa attorno all'asse Z per riportare il sistema di riferimento tool con la stessa orientazione del sistema di riferimento dell'oggetto.

Proiettando il segmento sul piano XY si possono ottenere le 4 situazioni, descritte nel primo grafico in figura 3.11, a seconda della posizione reciproca dei due punti che definiscono gli estremi del segmento.

Gli angoli da comunicare al robot dovranno quindi essere definiti in modo opportuno in relazione al caso in cui ci si trova.

**Rotazione B** - $R_y(\beta)$.
Per quanto riguarda la rotazione da imprimere all'asse y per riportare il sistema di riferimento della pinza nella stessa orientazione dell'oggetto si deve mantenere un angolo sempre fisso di 180° in modo che l'asse Z del tool sia entrante nell'oggetto.

$$B = 180^{\circ}$$

**Rotazione C** - $R_x(\gamma)$.

Per la rotazione attorno all'asse X si proietta il segmento nel piano YZ e, come nei casi precedenti, si calcola l'angolo tra il segmento e l'asse orizzontale. Seguendo la stessa logica utilizzata finora si riporta l'angolo calcolato per definire la rotazione da trasmettere al robot.

Il passaggio dagli angoli relativi all'orientazione del segmento fino ai parametri di rotazione A B C è stato possibile implementando una logica che considerasse caso per caso le varie combinazioni di angoli per riuscire ad orientare gli assi del sistema di riferimento del tool in modo che l'asse X fosse solidale all' orientazione del segmento individuato corrispondente all'asse principale dell'oggetto, l'asse Y fosse perpendicolare a questo e infine l'asse Z fosse entrante nell'oggetto.

## 3.8 Strumentazione aggiuntiva realizzata tramite stampa 3D

In fase di progetto è stato necessario definire nuovi componenti da aggiungere alla struttura del robot utilizzato. Si disegnano nuove geometrie e se ne sviluppano poi i modelli tramite stampa 3D a filamento.

Si deve adempiere a due necessità. La prima è quella relativa alla presa degli oggetti: date le dimensioni degli oggetti scelti per il progetto e assodato che, come si valuterà nella sezione dedicata, l'end-effector sarà composto da delle griffe parallele, risulta necessario andare ad adattare l'apertura e la chiusura della pinza con delle dita opportunamente sagomate. Il secondo componente da realizzare è la piastra per permettere il montaggio della camera a bordo del robot.

Appoggiandosi quindi alla documentazione della pinza scelta e ai datasheet relativi al robot utilizzato, è stato possibile generare i modelli dei componenti adattando così il robot antropomorfo all' applicazione che si sta definendo. In figura3.12 viene presentata una raffigurazione dei due componenti appena descritti.

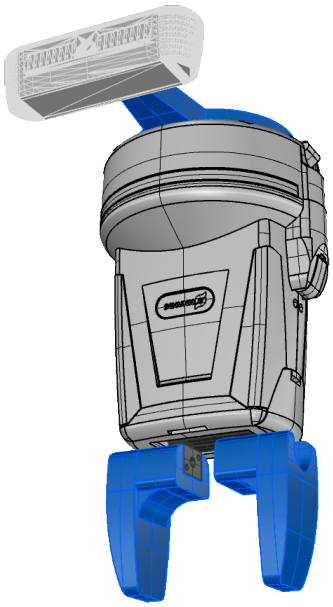


**Figura 3.12:** Pinza Schunk con evidenziate le parti stampate in 3D: griffe e piastra per montare la camera a bordo del robot

## 3.9 Intel RealSense

### 3.9.1 OpenCV

Per riuscire in una facile e completa manipolazione dei dati contenuti nelle immagini scattate, si sfrutta una libreria open source dedicata all'elaborazione delle immagini. OpenCV (*Open Source Computer Vision*), mette a disposizione una serie di modelli e algoritmi che possono essere utilizzati nelle applicazioni di visione artificiale. Grazie alle numerose librerie contenute al suo interno è possibile sfruttare e accedere a funzionalità aggiuntive e personalizzabili.

Nel progetto si sono utilizzate le potenzialità di questo pacchetto in tutte le sezioni di codice nelle quali è stato fondamentale sfruttare o elaborare le informazioni ricavate dai fotogrammi scattati dalla camera, sia in due che in tre dimensioni. Per fare un esempio, OpenCV permette di aggiungere testo e forme geometriche, come linee e punti all'interno di un' immagine. A livello progettuale si è sfruttata questa funzionalità per valutare l'effettiva correttezza degli angoli ricavati dalla logica del programma. Si aggiunge nel frame a colori memorizzato un cerchio in corrispondenza del centro bounding box e un segmento orientato secondo l'angolo dell'oggetto. In questo modo si è potuto valutare che l'orientazione del segmento corrispondesse a quella del prodotto.

La stessa rete neurale YOLO è basata sull'estrapolazione delle informazioni dalle immagini tramite l'utilizzo di questa libreria open source.

### 3.9.2 Funzioni per la gestione e l' acquisizione delle immagini

Per quanto riguarda il codice relativo alla gestione dei parametri della camera e i processi di acquisizioni delle immagini RGB e di profondità vengono sfruttate le librerie proprietarie messe a disposizione dalla casa Intel.

Nella struttura del codice viene definito un file Python subordinato nel quale vengono espresse tutte le funzioni dedicate alla cattura dei parametri ottici. All'interno del codice principale vengono richiamate queste funzioni quando si ha la necessità di svolgere delle acquisizioni. Il file di gestione della camera, di cui si fa riferimento, è il *realsense_camera.py*: qui di seguito si esprimono alcuni dettagli fondamentali di tale software.

#### Funzione *capture_image*

All'interno delle righe di questo codice si definisce per prima cosa la directory nella quale salvare l' immagine. Successivamente viene inizializzato lo stream video che permette di visualizzare a schermo quello che cattura la camera: operazione necessaria solo durante la fase di progettazione ed eventualmente da rendere disponibile per metodi di manutenzione. Una volta che da tastiera viene premuto il tasto *q* il flusso video viene interrotto e si comanda una scansione dello spazio di lavoro sfruttando la camera RBG e memorizzando nella directory definita il frame a colori acquisito.

Questa funzione permette al sistema di acquisire una prima immagine della scena da inviare in input alla rete neurale per il riconoscimento degli oggetti.

#### Funzione *define_depth_frame*

La funzione è definita per la misura di profondità di tutti i punti nello spazio di lavoro.

In particolare si utilizza la stessa risoluzione di pixel utilizzata per la gestione delle immagini RGB: si crea una matrice di dimensione 640 x 480 nella quale vengono memorizzate le informazioni di profondità per ciascun pixel. In questo modo si ha a disposizione una vera e propria mappa di punti per i quali è definita la quota in Z. Memorizzare una matrice di queste specifiche dimensioni contenente dati relativi alle quote di ciascun pixel è fondamentale per avere a disposizione i valori spaziali di tutti i punti che compongono l'immagine.

### 3.9.3 Calibrazione della camera

Come anticipato nel capitolo 2, per l'utilizzo della camera Intel Realsense D435 all'interno del progetto e per ricavarne le misurazioni dello spazio tridimensionale, è fondamentale settare e calibrare opportunamente i parametri che gestiscono i valori estrapolati dal sistema di visione. La camera 3D permette di svolgere delle acquisizioni dello spazio di lavoro ricavando immagini a colori e immagini di profondità, azioni possibili tramite l'utilizzo delle funzioni descritte nel paragrafo appena concluso. Queste misurazioni, per essere sfruttate nel progetto, devono fornire dei valori di posizione accurati e che corrispondano con precisione alle distanze e alle proporzioni dell' ambiente reale. Si è definito come le prime rilevazioni vengano svolte con riferimento ad un sistema in pixel. È fondamentale quindi che nella trasformazione di queste informazioni nel sistema di riferimento dello spazio tridimensionale, in coordinate XYZ espresse in millimetri, si utilizzino opportune funzioni che sfruttino dei parametri calibrati a seconda del range di distanze definite dal progetto.

Per la gestione del metodo e delle azioni di calibrazione viene definito un codice Python a sè stante nel quale si inseriscono tutte le operazioni da svolgere per assicurarsi di aver svolto un corretto montaggio della camera nel sistema robotico. Il file in esame è quello definito come *space_calibration.py*.

Per svolgere i task di calibrazione viene predisposta una griglia di riferimento su un foglio di dimensione A3. Questa, formata da quattro linee a due a due parallele, permette di stabilire gli assi e un'origine di un sistema di riferimento *camera*, vedi fondo della figura 3.13.

#### Processo di verifica della corretta installazione della camera

Per prima cosa si deve verificare che la camera installata nel sistema sia correttamente orientata rispetto al piano e rispetto al sistema di riferimento del robot su cui è montata.

Si implementa un metodo che permetta di verificare con precisione che la camera inquadri l'area di lavoro in modo perpendicolare. Per fare questo si installa la camera nella posizione in cui verranno scattate le immagini e si libera il piano di lavoro da qualsiasi oggetto. Inizializzando lo streaming dei frame RGB si visualizza quindi il piano di lavoro e, definendo 4 punti agli estremi dell'immagine, si effettua una misurazione di profondità sfruttando la matrice *depth_frame*.

Si confrontano le 4 quote acquisite e si ragiona sull'eventuale aggiustamento di rotazione da applicare alla camera. La camera risulterà orientata in modo corretto e perpendicolare al piano di lavoro quando i 4 valori di profondità misurati risultano essere uguali. In figura 3.13, vengono riportati in blu (A-B-C-D) i 4 punti utilizzati nel processo appena descritto.

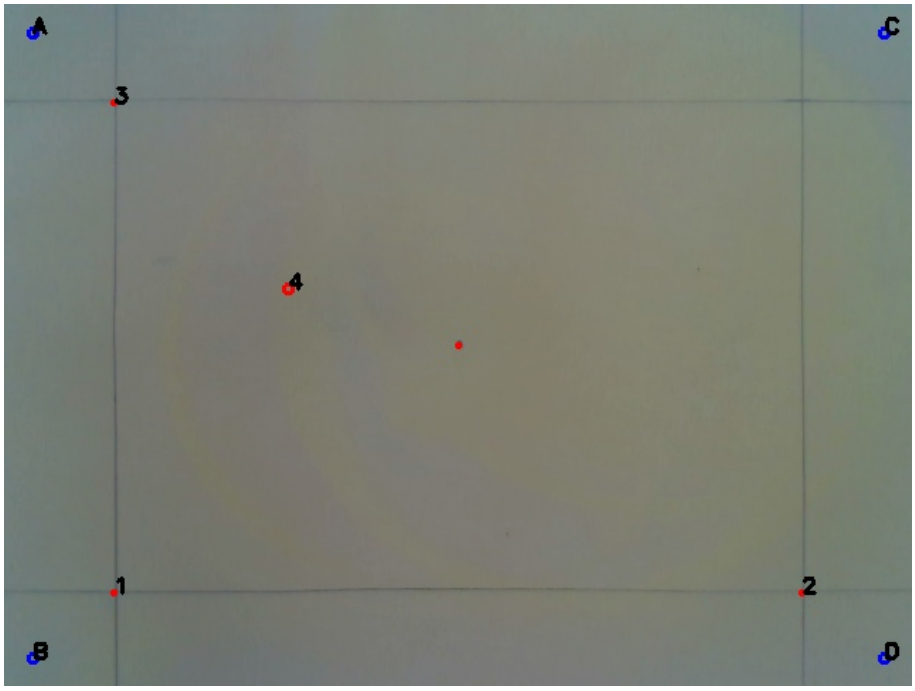



**Figura 3.13:** Griglia e punti di calibrazione

### Calibrazione dei parametri

La posizione in cui vengono scattate le immagini per la determinazione delle coordinate e le orientazioni del punto di pick da raggiungere viene fissata ad una quota di 450 millimetri. La libreria Python, messa a disposizione dalla serie di camere Intel, fornisce la lista di parametri da settare per una corretta definizione della funzione per la determinazione delle distanze nello spazio in millimetri. In particolare si devono fissare i parametri contenuti nella lista _*intrinsics*: si specificano i valori di lunghezza focale in x e in y (*fx*: lunghezza focale lungo x, *fy*: lunghezza focale lungo y). Variando questi parametri si influenza infatti la trasformazione delle informazioni spaziali e la conseguente determinazione delle lunghezze e posizioni espresse in millimetri secondo le coordinate XYZ.

Per la calibrazione dello spazio di lavoro si definiscono altri 4 punti, riportati in rosso e numerati dal 1 al 4 nella figura 3.13. Questi, escluso il 4, sono fissati in corrispondenza delle intersezioni tra le linee che compongono la griglia: la griglia che si è definita presenta due linee orizzontali parallele e distanziate 250 millimetri e altre due linee verticali parallele ad una distanza reciproca di 350 millimetri. Fissare i punti 1-2-3 in questo modo permette di verificare che la distanza ricavata tramite trasformazione ed elaborazione dei dati della camera corrisponda alla

distanza relativa nota tra i punti definiti nello spazio della griglia. Il punto 4, fissato in una posizione randomica all'interno dello spazio fotografato, viene utilizzato per verificare la misura di profondità. In particolare, in questa fase, si posiziona in corrispondenza di questo punto una confezione di forma parallelepipeda di profondità nota (144 millimetri) e si confronta questo valore con il dato ricavato dalla misura del sistema di visione.

L'obiettivo è quindi quello di ottenere, nelle tre direzioni nello spazio, dei valori di distanza tra i 4 punti definiti corrispondenti a quelle note e reali definite. Per ottenere questo si modificano i parametri caratteristici della camera fino a quando non si ottiene questa corrispondenza. Per la situazione relativa a questo progetto di sono definiti i seguenti valori:

$$fx = \frac{640}{\tan(54.6) * 2\pi/360} \qquad fy = fx$$

## 3.10 Robot Kuka

Le informazioni che sono state ricavate dal sistema di visione sono state successivamente elaborate fino a dare espressione ai dati di posizione e orientazione degli oggetti riconosciuti all'interno dello spazio di lavoro.

Nonostante l'ottica mantenuta in questo progetto fosse quella di presentare un pacchetto software che non si vincolasse ad un particolare sistema robotico, per completare l'applicazione di quanto si sta presentando, è fondamentale l' utilizzo di un apparato robotico antropomorfo per permettere di attuare le movimentazioni dei prodotti e la loro suddivisione nelle diverse classi di rifiuto.

Per rendere possibili i cicli di pick e place degli oggetti si utilizza il robot Kuka LBR IIWA 14 R820. Nel capitolo 2 sono state presentate alcune delle caratteristiche tecniche relative a questo braccio antropomorfo a 7 assi.

Il codice finora descritto ha permesso quindi di estrapolare delle informazioni relative alle coordinate dell'oggetto da prelevare nello spazio di lavoro. Tali specifiche devono essere ora inviate al controllore che comanda il braccio antropomorfo perchè sia attuata la movimentazione. Si forniscono ora alcuni dettagli sulla tipologia di messaggio da inviare e sulla modalità con cui è stato possibile far comunicare il software del progetto con il controllore industriale.

#### Collegamento tramite cavo ETHERNET

Il robot Kuka utilizzato permette di stabilire delle connessioni con un computer. In questo modo si ha la possibilità di caricare i progetti Java definiti all'interno dell'interfaccia proprietaria Kuka Sunrise Workbench.

Questo collegamento può avvenire tramite due diverse modalità: tramite una connessione fisica utilizzando un cavo Ethernet oppure tramite una connessione wifi interna al robot. Nel progetto si è sfruttata la connessione fisica via cavo, dato che si è dovuto scartare la connessione wireless per delle problematiche di instabilità del collegamento stesso.

### 3.10.1 Comunicazione Client-Server

Innanzitutto si specifica come la comunicazione tra il programma relativo al progetto e il sistema di controllo e attuazione del robot è stata possibile tramite l'implementazione di una architettura client/server.

I terminali che compongono la rete sono solamente due nella situazione in esame: il client è rappresentato dal terminale nel quale viene eseguito il codice Python che fornisce come output le informazioni di posizione, mentre il lato server è definito dal controllore che gestisce il robot entro il quale è in esecuzione il codice Java con le logiche di movimentazione.

La connessione è resa possibile dall'utilizzo della comunicazione tramite socket: la trasmissione e la ricezione dei dati avviene per mezzo di una porta che permette di interconnettere i processi che lavorano su due dispositivi fisicamente separati. La socket può essere vista come il contenitore entro il quale inserire il messaggio da scrivere e successivamente leggere tra i due terminali connessi.

Nell'utilizzo di questi strumenti software ci sono dei vincoli da rispettare per garantire che innanzitutto avvenga e si concluda la connessione tra i dispositivi, ma soprattutto anche che il contenuto del messaggio rimanga integro dal momento della scrittura fino alla lettura da parte del ricevente. In questi termini è fondamentale aggiungere al codice tutte le fasi di apertura, inizializzazione e chiusura delle comunicazioni e, fattore determinante, formattare adeguatamente la stringa da inviare come messaggio.

#### Stringa inviata

Nello specifico i dati di posizione e orientazione del punto da raggiungere per il place e tutte le informazioni relative all'oggetto riconosciuto sono state memorizzate in una struttura lineare definita come un array di interi. Si è esclusa una prima versione nella quale le informazioni

erano fornite come float: si ottiene una precisione maggiore, ma non significativa data l'applicazione. Quindi, a prescindere dal tipo assegnato alle singole variabili, le informazioni contenute in questa struttura devono essere uniformate e adattate per l'invio all'interno della socket per la comunicazione con il robot.

[*Priority Object_Type X_Value Y_Value Z_Value A_Value B_Value C_Value*]

In questo modo è possibile gestire in modo univoco l' invio ed essere certi che la codifica lato client e la decodifica della stringa lato server avvengano in modo corretto mantenendo integre le informazioni inviate.

#### Limite nella comunicazione Client-Server

Nel ciclo di pick e place degli oggetti si deve far partire manualmente l' esecuzione del sistema di visione per l'acquisizione delle immagini e dei dati di posizione con il software di gestione dei movimenti. Sarebbe interessante valutare che il lato server inviasse al client Python un messaggio per l'inizio automatico di questa esecuzione quando il robot si trova nella posizione di Home oppure, in alternativa, implementare un client in Python che rimanga costantemente in esecuzione in attesa e che inizializzi l'acquisizione dell'immagine solamente quando riconosce di essere in posizione di Home. In questa ipotetica soluzione, il lato client potrebbe verificare di essere in posizione di Home ad esempio misurando la profondità relativa a quattro punti specifici nei quali si installano delle colonnine di altezza nota: quando per tutti e quattro i punti il robot misurasse la quota corrispondente alla situazione di Home, il codice comanderebbe l'inizializzazione dell'acquisizione dell'immagine. Nella progettazione delle traiettorie di movimento del robot è fondamentale evitare di passare per la posizione di Home in un qualsiasi altro istante se non quando non si necessita di svolgere l'acquisizione delle immagini.

### 3.10.2 KUKA Sunrise e codice Java di movimentazione del robot

Una volta che le informazioni sono state adeguatamene formattate e inserite nella stringa, possono essere inviate al controllore che gestisce la movimentazione del robot.

Come anticipato nel capitolo 2, il sistema Kuka permette la stesura di codici Java per definire le logiche dei movimenti da attuare. Si procede quindi con la scrittura del programma da

inviare al braccio ponendo attenzione ai cicli di pick e place da implementare. Nel progetto implementato il codice corrispondente è contenuto nel file *serverCamera.Java*.

Ci si rifà ad un metodo di pick-and-place standard. Il movimento è inizializzato una volta che il sistema acquisisce le informazioni della posizione di pick e le relative informazioni per categorizzare il prodotto per il place nell'apposito contenitore.

Si controlla l'apertura della pinza. Successivamente al robot viene richiesto di raggiungere la posizione relativa all'oggetto da prelevare. Questa traiettoria viene gestita tramite un comando di movimento Point-to-Point. Dato che è fondamentale il raggiungimento del punto designato, ma non ci sono vincoli sul percorso da imporre all' end-effector, viene specificata solamente la velocità, quindi si determinano il tempo e le accelerazioni. Poi, internamente al controllore, viene elaborata la combinazione di movimenti per ciascun asse che definisca la traiettoria del percorso più breve tra il punto iniziale e quello finale.

Nel ciclo in esame il punto di arrivo non è direttamente quello della posizione di pick: si definisce una certa quota di approach che permette di distanziarsi opportunamente lungo l'asse Z, orientato perpendicolarmente alla superficie l'oggetto ed entrante rispetto a questa. La fase di avvicinamento all'oggetto viene definita tramite un movimento lineare tra la posizione di approach e il centro dello stesso.

Arrivato nel punto si chiudono le pinze. Si effettua il movimento di depart dal punto, sempre tramite un movimento lineare, e si comanda al robot di giungere nel punto di pick.

Grazie alle informazioni contenute nella stringa inviata, è possibile memorizzare la classe di appartenenza dell'oggetto individuato nello spazio di lavoro. Grazie a questo dettaglio si caratterizza la tipologia di rifiuto al quale appartiene il prodotto individuato: a livello progettuale e dimostrativo si sono definite due categorie di rifiuto espresse in *carta* e *alluminio*. Per ciascuna di queste viene definita una diversa posizione di place in modo da suddividere i materiali di scarto a seconda del materiale di cui sono composti.

La fase di place degli oggetti prelevati avviene allo stesso modo. Il movimento verso la posizione di approach è definito dalla logica Point-to-Point seguito da un avvicinamento lineare. Si aprono le pinze e il pezzo viene rilasciato nel rispettivo contenitore. Dopo un allontanamento lineare si riporta il braccio nella posizione di Home, pronto per l'esecuzione di una seconda scansione e dell' eventuale ciclo di pick e place.

**Figura 3.14:** Pinza Schunk [4] e griffe personalizzate

## 3.11 Organo Terminale e griffe personalizzate

Data l'applicazione richiesta e valutate le soluzioni a disposizione si esprimono ora le considerazioni fatte per la scelta dell'organo terminale.

Nel capitolo 2 si sono presentate due diverse soluzioni per quanto riguarda questa scelta. Nello specifico si sono introdotte la pinza parallela e i sistemi pneumatici di presa a ventosa. Queste sono state le due possibilità prese in considerazione durante la progettazione del sistema. Dopo una prima valutazione la scelta si concentra verso la tecnologia che prevede l'utilizzo di aria compressa. Questa preferenza è motivata dal fatto che gli oggetti scelti e per i quali si è allenata la rete risultano essere rifiuti di peso relativamente leggero e con forme e geometrie variabili: la soluzione a ventose garantisce una certa flessibilità in termini di capacità di presa di oggetti con forme diverse.

Successivamente però, a causa di esigenze esterne, si è inserito come dettaglio definitivo un end-effector a pinze parallele bilaterali. Si rinuncia leggermente al concetto di flessibilità, ma in questo modo è stato possibile avere a disposizione nell'immediato il sistema di presa svincolandosi da eventuali attese di fornitura. La pinza collaborativa *Schunk CO-act EGP-C 40* è disponibile in laboratorio mentre per la soluzione ad aria compressa, sarebbe stata necessario un dimensionamento del sistema, una scelta a catalogo dei componenti e una richiesta di fornitura che avrebbe comportato delle attese non compatibili con lo sviluppo del progetto.

In figura3.14 si riporta una rappresentazione della scelta progettuale.

Si specifica come la decisione di utilizzare delle griffe limiti il range di oggetti che può essere

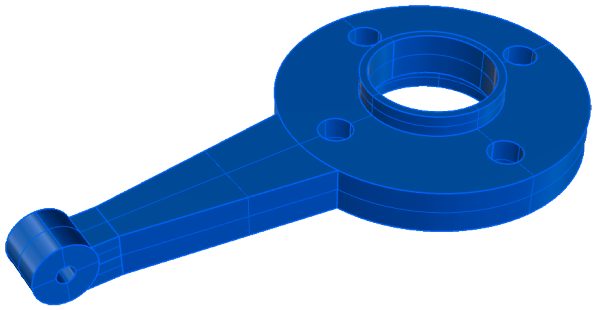


**Figura 3.15:** Modello 3D della flangia utilizzata per implementare la camera a bordo del robot

raccolto dalla pinza. In questi termini viene scartata la confezione di Nutella Biscuits (Nutella) a causa della geometria e dalla composizione e flessibilità del materiale di cui è composta.

## 3.12 Camera a bordo del robot

Il progetto fin ora presentato a livello funzionale può essere suddiviso in due principali sistemi. Il primo è quello relativo alla gestione della camera, mentre il secondo è relegato alla movimentazione. La parte software di entrambi è stata ampiamente esplicitata nei paragrafi precedenti. Si tratta ora la sezione hardware che permette di integrare questi due sotto sistemi in un'unica soluzione.

I due sistemi infatti riescono a lavorare insieme e a costituire parti diverse di uno stesso progetto, integrato in una soluzione unica. In generale, in ogni sistema robotico che svolge attività di bin picking con sistemi di visione integrati, la sezione hardware relativa alla computer vision può assumere due diverse configurazioni.

La prima proposta è quella di sistemi di visione industriali montati in strutture rigide fisse esternamente al sistema robot. In questo caso è possibile implementare sistemi di visione anche complessi e performanti garantendo un sistema di riferimento statico. La seconda alternativa prende piede in quelle soluzioni nelle quali il sistema di visione è montato a bordo del robot, lo stesso che eseguirà poi le movimentazioni.

Si vuole sottolineare come entrambe le soluzioni presentino delle buone caratteristiche. Per i sistemi in cui la visione è installata in strutture fisse si garantisce che le informazioni che si ricavano dalle misure siano riferite ad un sistema di riferimento che si mantiene costante e immutato nel tempo. I sistemi invece con camera a bordo permettono di svolgere delle scansioni della scena da posizioni ed angolazioni diverse a seconda di come è orientato il robot nello spazio.

A livello progettuale si sceglie di procedere montando la camera nel robot permettendo così di installare nel robot l'hardware relativo alla visione. Questa soluzione viene presa perchè, integrando il sistema di visione nel braccio robotico, le misurazioni dello spazio di lavoro risultano più flessibili in quanto è possibile definire nel dettaglio il punto in cui svolgere le foto. Risultano inoltre ridotti anche i tempi di ciclo, in quanto se si adottasse una soluzione con camera fissa in una struttura esterna, si dovrebbe far muovere il robot al di fuori dello spazio di lavoro in modo che la camera scatti delle immagini dell'intero ambiente, senza che il braccio occupi l'obiettivo, attendendo quindi l'esecuzione della traiettoria di uscita.

# 4

# Setup Sperimentale

## 4.1 Esecuzione del progetto

In questo paragrafo si elencano tutti i passaggi da svolgere per l'esecuzione del codice relativo al progetto. Si vuole simulare una sequenza di azioni da mettere in atto per far sì di ottenere la presa degli oggetti e la loro suddivisione in categorie di rifiuti sfruttando il modello presentato in questo progetto.

Si considera che i file relativi al codice nei linguaggi di programmazione utilizzati siano opportunamente sistemati nella directory e sia correttamente preparato l'ambiente di sviluppo per l'esecuzione dei programmi. Deve essere inoltre predisposta e installata tutta la parte hardware relativa al sistema: la camera e le griffe devono essere montate nel braccio.

1. **Posizionare la griglia di calibrazione in corrispondenza delle direzioni XY del robot**
   Il sistema di riferimento e gli assi XYZ del robot sono fissi e ben definiti. È fondamentale per prima cosa far coincidere questo sistema di riferimento con le direzioni XY della camera. Per fare questo, dato che la griglia di calibrazione servirà poi per le calibrazioni nello spazio della camera, si deve fissare al piano di lavoro il foglio con stampata la griglia in modo da avere le direzioni XY del robot corrispondenti alle linee impresse nel foglio.

2. **Portare il robot nella posizione di Home**
   Prima di iniziare la calibrazione della camera è fondamentale porre il robot nella posizione prevista per lo stream delle immagini. Nel progetto tale posizione è definita proprio

come punto di Home dello spazio:

$$[-200 \quad -600 \quad 475 \quad 0.6 \quad 160 \quad 0]$$

3. **Verificare le quote dei punti A-B-C-D**
   Facendo eseguire il codice *space_calibration.py* si deve ottenere lo stesso valore di profondità in relazione alle misurazioni effettuate nei quattro punti evidenziati nel frame in prossimità dei quattro estremi dell'immagine. Facendo coincidere questi parametri è possibile avere la certezza che la camera inquadri correttamente il piano di lavoro in modo perpendicolare evitando inclinazioni anomale tra il piano e il sistema di visione.

4. **Posizionare il *punto_centro* in corrispondenza del punto al centro della griglia**
   Per verificare che il centro del frame catturato dalla camera corrisponda con il centro della griglia, e quindi essere certi che i due sistemi facciano riferimento ad uno stesso punto, si deve controllare che il punto definito nel codice come *punto_centro* corrisponda con il punto stampato nel foglio al centro della griglia.

5. **Posizionare i punti 1-2-3**
   Si devono modificare le coordinate relative ai punti definiti all'interno del codice Python fino ad avere una corrispondenza tra i punti definiti nel codice con quelli stampati nel foglio e definiti come le intersezioni delle linee che compongono la griglia.

6. **Verificare le distanze tra i punti**
   I punti presenti nel foglio che indicano le intersezioni degli assi della griglia di calibrazione sono a delle distanze specifiche nel piano. Una volta definite le posizioni in pixel di questi punti nel codice si deve verificare che le distanze definite dalla griglia corrispondano a quelle calcolate dal codice. Per fare questo si agisce modificando i parametri che gestiscono la trasformazione delle coordinate nello spazio interne alle librerie Intel definite dal prefisso _*intrinsics*. Si variano i valori di lunghezza focale orizzontale e verticale fino a quando non si ottengono le seguenti distanze:

$$distanza\_1-2 = 350mm \quad distanza\_1-3 = 250mm$$

È ora conclusa la calibrazione iniziale della camera e la sua corretta installazione nel sistema. Si può procedere con l'esecuzione del codice attivando i software dei due terminali: *serverCamera.Java* e *progetto.py*. Nel ciclo di pick-and-place si devono eseguire manualmente entrambi i codici, sia quello che gestisce il sistema di visione che quello per la movimentazione del robot.

# 5

# Codice Implementato

In questo capitolo vengono elencate in dettaglio tutte le righe del codice utilizzate nel progetto. Si includono sia le porzioni di codice scaricato dalla directory Github riguardante la gestione della rete neurale che quelle definite durante lo svolgimento delle diverse task.

## 5.1 SPACE_CALIBRATION.PY

```
import cv2
import pyrealsense2 as py
import numpy as np
import os
from realsense_camera import *
import math
import time


cap = cv2.VideoCapture(1);
saving_path = 'C:/Users/riccardo/Documents/Riccardo_DB/Progetto_Tesi'

rs = RealsenseCamera()

while (cap.isOpened()):

```

```
    ret_z, bgr_frame = rs.get_frame_stream()
    ret_depth, depth_frame = rs.define_depth_frame()
    ret, frame = cap.read()

    point_centro =(320,240)
    circle = cv2.circle(bgr_frame, point_centro,1,(0,0,255),2)

    point_1 = (77,414)
    circle = cv2.circle(bgr_frame, point_1,1,(0,0,255),2)
    point_coord= cv2.putText(bgr_frame,'1',point_1,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_2 =(562,414)
    circle = cv2.circle(bgr_frame, point_2,1,(0,0,255),2)
    point_coord= cv2.putText(bgr_frame,'2',point_2,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_3 =(77,69)
    circle = cv2.circle(bgr_frame, point_3,1,(0,0,255),2)
    point_coord= cv2.putText(bgr_frame,'3',point_3,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_4 =(200,200)
    circle = cv2.circle(bgr_frame, point_4,3,(0,0,255),2)
    point_coord= cv2.putText(bgr_frame,'4',point_4,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    #verifica perpendicolarità asse z della camera con il piano
    point_A =(20,20)
    circle =cv2.circle(bgr_frame, point_A,3,(255,0,0),2)
    point_coord= cv2.putText(bgr_frame,'A',point_A,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_B =(20,460)
    circle =cv2.circle(bgr_frame, point_B,3,(255,0,0),2)
    point_coord= cv2.putText(bgr_frame,'B',point_B,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_C =(620,20)
    circle =cv2.circle(bgr_frame, point_C,3,(255,0,0),2)
```

```
    point_coord= cv2.putText(bgr_frame,'C',point_C,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    point_D =(620,460)
    circle =cv2.circle(bgr_frame, point_D,3,(255,0,0),2)
    point_coord= cv2.putText(bgr_frame,'D',point_D,cv2.FONT_HERSHEY_SIMPLEX
    ,0.5,(0,0,0),2)

    px_1 = point_1[0]
    py_1 = point_1[1]
    pz_1 = depth_frame[point_1[1],point_1[0]]

    px_2 = point_2[0]
    py_2 = point_2[1]
    pz_2 = depth_frame[point_2[1],point_2[0]]

    px_3 = point_3[0]
    py_3 = point_3[1]
    pz_3 = depth_frame[point_3[1],point_3[0]]

    px_4 = point_4[0]
    py_4 = point_4[1]
    pz_4 = depth_frame[point_4[1],point_4[0]]

    cx = 640/2.0
    cy = 480/2.0
    fx = 640 / (2.0 * math.tan(float(54.6) * math.pi / 360.0))
    fy = fx

    _intrinsics = py.intrinsics()
    _intrinsics.width = 640
    _intrinsics.height = 480
    _intrinsics.ppx = cx
    _intrinsics.ppy = cy
    _intrinsics.fx = fx
    _intrinsics.fy = fy
    _intrinsics.model = py.distortion.none
    _intrinsics.coeffs = [i for i in [0, 0, 0, 0, 0]]

    space_1 = py.rs2_deproject_pixel_to_point(_intrinsics,[py_1,px_1],pz_1)
    x_1 = int(space_1[1])
```

```
    y_1 = int(-space_1[0])
    z_1 = int(-space_1[2])

    space_2 = py.rs2_deproject_pixel_to_point(_intrinsics,[py_2,px_2],pz_2)
    x_2 = int(space_2[1])
    y_2 = int(-space_2[0])
    z_2 = int(-space_2[2])

    space_3 = py.rs2_deproject_pixel_to_point(_intrinsics,[py_3,px_3],pz_3)
    x_3 = int(space_3[1])
    y_3 = int(-space_3[0])
    z_3 = int(-space_3[2])

    space_4 = py.rs2_deproject_pixel_to_point(_intrinsics,[py_4,px_4],pz_4)
    x_4 = int(space_4[1])
    y_4 = int(-space_4[0])
    z_4 = int(-space_4[2])

    if ret == True:
        cv2.imshow('frame', bgr_frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()

cv2.destroyAllWindows()

a = np.array((x_1, y_1, z_1))
b = np.array((x_2, y_2, z_2))
c = np.array((x_3, y_3, z_3))
d = np.array((x_4, y_4, z_4))

dist_12 = np.linalg.norm(a-b)
dist_13 = np.linalg.norm(a-c)
diff_quote = z_1 - z_4

print('verifica perpendicolarità camera-piano. Le profondità devono essere simili:')
print('punto_A:' + str(depth_frame[20,20]))
print('punto_B:' + str(depth_frame[460,20]))
```

```
print('punto_C:' + str(depth_frame[20,620]))
print('punto_D:' + str(depth_frame[460,620]))

print('Primo punto: (' + str(x_1) + ',' + str(y_1) + ',' + str(z_1) + ')')
print('Secondo punto: (' + str(x_2) + ',' + str(y_2) + ',' + str(z_2) + ')')
print('Terzo punto: (' + str(x_3) + ',' + str(y_3) + ',' + str(z_3) + ')')
print('Quarto punto: (' + str(x_4) + ',' + str(y_4) + ',' + str(z_4) + ')')

print('Distanza 1-2 (target: 350mm): ' + str(dist_12))
print('Distanza 1-3 (target: 250mm): ' + str(dist_13))
print('Differenza quote (target: 144mm): ' + str(diff_quote))

out_path = 'C:/Users/riccardo/Documents/Riccardo_DB/Project/calibration'
cv2.imwrite(os.path.join(saving_path , 'calibrazione.jpg'),bgr_frame)
```

## 5.2 GLOBAL_VAR.PY

```
import numpy as np
import math

def initialize_data():
    global data_output
    data_output = np.zeros((100,5))
    data_output[0,0] = 1

    return data_output

def define_circle(center): #Costruisco 9 cerchi di raggio 12 formati ciascuno da 60 punti
    global circle_point
    circle_point =np.zeros((9,60,2))
    i=0
    j=0
    raggio=12
    while i<=8:
        theta=0
        while j<=59:
            circle_point[i,j,0] = center[0]+raggio*math.cos(math.radians(theta))
            circle_point[i,j,1] = center[1]+raggio*math.sin(math.radians(theta))
            j+=1
```

```
            theta+=6
        i+=1
        j=0
    return circle_point
```

## 5.3 REALSENSE_CAMERA.PY

```
import pyrealsense2 as rs
import numpy as np
import cv2
import time
import os


class RealsenseCamera:
    def __init__(self):
        # Configure depth and color streams
        print("Loading Intel Realsense Camera")
        self.pipeline = rs.pipeline()

        config = rs.config()
        config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
        config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)

        # Start streaming
        self.pipeline.start(config)
        align_to = rs.stream.color
        self.align = rs.align(align_to)

    def capture_image(self):
        cap = cv2.VideoCapture(1);
        saving_path = 'C:/Users/riccardo/Documents/Riccardo_DB/Progetto_Tesi/
    yolov7_custom'

        while (cap.isOpened()):

            ret, bgr_frame =RealsenseCamera.get_frame_stream(self)

            if ret == True:
                cv2.imshow('frame', bgr_frame)
```

```
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break

        cv2.imwrite(os.path.join(saving_path , 'image.jpg'), bgr_frame)
        cap.release()
        cv2.destroyAllWindows()

    def get_frame_stream(self):
        # Wait for a coherent pair of frames: depth and color
        frames = self.pipeline.wait_for_frames()
        aligned_frames = self.align.process(frames)
        color_frame = aligned_frames.get_color_frame()

        if not color_frame:
            # If there is no frame, probably camera not connected, return False
            print("Error, impossible to get the frame, make sure that the Intel
    Realsense camera is correctly connected")
            return False, None

        # Apply filter to fill the Holes in the depth image
        spatial = rs.spatial_filter()
        spatial.set_option(rs.option.holes_fill, 3)

        color_image = np.asanyarray(color_frame.get_data())

        return True, color_image

    def define_depth_frame(self):
        # Wait for a coherent pair of frames: depth and color
        frames = self.pipeline.wait_for_frames()
        aligned_frames = self.align.process(frames)
        depth_frame = aligned_frames.get_depth_frame()

        if not depth_frame:
            # If there is no frame, probably camera not connected, return False
            print("Error, impossible to get the frame, make sure that the Intel
    Realsense camera is correctly connected")
            return False, None

```

```
        # Apply filter to fill the Holes in the depth image
        spatial = rs.spatial_filter()
        spatial.set_option(rs.option.holes_fill, 3)
        filtered_depth = spatial.process(depth_frame)
        hole_filling = rs.hole_filling_filter()
        filled_depth = hole_filling.process(filtered_depth)

        # Create colormap to show the depth of the Objects
        colorizer = rs.colorizer()
        depth_colormap = np.asanyarray(colorizer.colorize(filled_depth).get_data())

        # Convert images to numpy arrays
        depth_image = np.asanyarray(filled_depth.get_data())

        return True, depth_image

    def release(self):
        self.pipeline.stop()
```

## 5.4 PROGETTO.PY

```
import detect
import project_space_multiple_frame
import numpy as np
import communicate_to_server

print('Start 2D Scan')
data_output = detect.detect()
print('Detection complete.')
if (not data_output[0,2]==0):

    print("[       Priority  Object_Type  X_pixel     Y_pixel    Depth_Value ]")
    i=0
    while data_output[i,1]!=0:      #Stampa di tutti gli oggetti riconosciuti
        print(data_output[i,:])
        i+=1
    print("\nFocus on the priority object")

    print("Start 3D scan")
    i=0
```

```
    while data_output[i,1]!=0:
        if data_output[i,0]==1:
            signal_output = project_space_multiple_frame.project_in_space(
    data_output[i,:])
        i+=1
    print('Project in space complete.\n')

    print("Position and orientation value in space")
    print("Format the string to send to the robot\n")

    print("[       Priority  Object_Type  X_Value     Y_Value      Z_Value
    A_value     B_Value     C_Value ]")
    print(signal_output)
else:
    print("No more object in the space")

    signal_output= [ 0,           6,      -188,        -707,         336,         0,
             180,         0]
    print("Send Home position")

    print("[       Priority  Object_Type  X_Value     Y_Value      Z_Value
    A_value     B_Value     C_Value ]")
    print(signal_output)

communicate_to_server.client_program(signal_output)
```

## 5.5 DETECT.PY

```
import argparse
import time
from pathlib import Path
import os
import numpy as np

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

import global_var
```

```
import pyrealsense2 as py
from realsense_camera import *

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow,
    non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized,
    TracedModel

import warnings

warnings.filterwarnings("ignore")

def detect(save_img=False):

    #define image
    rs= RealsenseCamera()
    rs.capture_image()

    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='best_200.pt',
    help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='image.jpg', help='source')
    parser.add_argument('--img-size', type=int, default=640, help='inference size (
    pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.59, help='object
    confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU threshold
     for NMS')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or
    0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.
    txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences
    in --save-txt labels')
    parser.add_argument('--nosave', action='store_true', help='do not save images/
    videos')
```

```
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--no-trace', action='store_true', default=True, help='don`t trace model')
    opt = parser.parse_args()
    print('Start Detect')
    source, weights, view_img, save_txt, imgsz, trace = opt.source, opt.weights, opt.view_img, opt.save_txt, opt.img_size, not opt.no_trace
    save_img = not opt.nosave and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok))  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Initialize
    data_output = global_var.initialize_data()
    zeta = 4000
    set_logging()
    device = select_device(opt.device)
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
```

```
    imgsz = check_img_size(imgsz, s=stride)  # check img_size

    if trace:
        model = TracedModel(model, device, opt.img_size)

    if half:
        model.half()

    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt', map_location=
    device)['model']).to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride)

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

    # Run inference
    if device.type != 'cpu':
        model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.
    parameters())))  # run once
    old_img_w = old_img_h = imgsz
    old_img_b = 1

    t0 = time.time()
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
```

```
            img = img.unsqueeze(0)

        # Warmup
        if device.type != 'cpu' and (old_img_b != img.shape[0] or old_img_h != img.shape[2] or old_img_w != img.shape[3]):
            old_img_b = img.shape[0]
            old_img_h = img.shape[2]
            old_img_w = img.shape[3]
            for i in range(3):
                model(img, augment=opt.augment)[0]

        # Inference
        t1 = time_synchronized()
        with torch.no_grad():   # Calculating gradients would cause a GPU memory leak
            pred = model(img, augment=opt.augment)[0]
        t2 = time_synchronized()

        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
        t3 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)

        # Process detections
        for i, det in enumerate(pred):
            if webcam:
                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
            else:
                p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # img.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]

```

```
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).
    round()

                # Print results
                s = 'Object in space:\n'
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)} "  # add to string
                    s +='\n'
                #Export Data
                l= 0
                s+='\n'

                for *xyxy, conf, cls in reversed(det):
                    c1, c2 = (int(xyxy[0]),int(xyxy[1])), (int(xyxy[2]),int(xyxy[3])
    )
                    center_point = round((c1[0] + c2[0])/2), round((c1[1] + c2[1])
    /2)

                     #Codifico la classe riconosciuta con un intero
                    if int(cls) == 0:           #cocacola
                        data_output[l,1] = 1
                    if int(cls) == 1:           #fanta
                        data_output[l,1] = 2
                    if int(cls) == 2:           #nutella
                        data_output[l,1] = 3
                    if int(cls) == 3:           #succo
                        data_output[l,1] = 4
                    if int(cls) == 4:           #vigorsol
                        data_output[l,1] = 5

                    data_output[l,2] = center_point[0]  #Imposto la x
                    data_output[l,3] = center_point[1]  #Imposto la y

                    ret, depth_frame = rs.define_depth_frame()
                    depth_for_priority = int(depth_frame[center_point[1],
    center_point[0]])

                    data_output[l,4] = depth_for_priority
```

```
                if data_output[l,4]<zeta:               #Imposto la priorità
                    zeta = data_output[l,4]
                    data_output[:,0]= 0
                    data_output[l,0]= 1

                l+=1

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
                    with open(txt_path + '.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')
                if save_img or view_img:  # Add bbox to image
                    label = f'{names[int(cls)]} {conf:.2f}'
                    plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=2)

        # Print time (inference + NMS)
        print(f'{s}Done. ({(1E3 * (t2 - t1)):.1f}ms) Inference, ({(1E3 * (t3 - t2)):.1f}ms) NMS')

        # Stream results
        if view_img:
            cv2.imshow(str(p), im0)
            cv2.waitKey(1)  # 1 millisecond

        # Save results (image with detections)
        if save_img:
            if dataset.mode == 'image':
                cv2.imwrite(save_path, im0)
                out_path = 'C:/Users/riccardo/Documents/Riccardo_DB/Progetto_Tesi'
                cv2.imwrite(os.path.join(out_path , 'detect_BB.jpg'),im0)

if save_txt or save_img:
    s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir
```

```
     / 'labels'}" if save_txt else ''

    return data_output
```

## 5.6 project_space_multiple_frame.py

```
import cv2
import pyrealsense2 as py
import numpy as np
import os
from realsense_camera import *
import math
import global_var
import time

def project_in_space(data):
    cap = cv2.VideoCapture(1);

    rs=RealsenseCamera()

    if cap.isOpened():

        origine = (77,414)
        ret_z, bgr_frame = rs.get_frame_stream()
        ret_loop, depth_frame_loop=rs.define_depth_frame()

        depth_origine=depth_frame_loop[origine[1],origine[0]]

        cx = 640/2.0
        cy = 480/2.0
        fx = 640 / (2.0 * math.tan(float(54) * math.pi / 360.0))
        fy = fx

        _intrinsics = py.intrinsics()
        _intrinsics.width = 640
        _intrinsics.height = 480
        _intrinsics.ppx = cx
        _intrinsics.ppy = cy
        _intrinsics.fx = fx
        _intrinsics.fy = fy
```

```
        _intrinsics.model = py.distortion.none
        _intrinsics.coeffs = [i for i in [0, 0, 0, 0, 0]]

        #Definisco i valori dell'origine in pixel e l'informazione sulla profondità
        pixel_x_0= origine[0]
        pixel_y_0= origine[1]
        pixel_z_0= depth_origine

        #Proietto l'origine nello spazio
        space_0 = py.rs2_deproject_pixel_to_point(_intrinsics,[pixel_y_0,pixel_x_0],
    pixel_z_0)
        x_0 = int(space_0[1])
        y_0 = int(-space_0[0])
        z_0 = int(-space_0[2])

    cap.release()

    cv2.destroyAllWindows()

    signal_output =np.zeros(8)

    #Inizializzo priorità e classe
    signal_output[0]= int(data[0])
    signal_output[1]= int(data[1])


    #Proietto le coordinate del centro nello spazio
    space = py.rs2_deproject_pixel_to_point(_intrinsics,[data[3],data[2]],data[4])

    print("Projection of position coordinates in space")

    signal_output[2] = 7.74 - (space[1]-x_0)
    signal_output[3] = -592.74 - (-space[0]-y_0)
    signal_output[4] =  (-space[2]-z_0)
    center = int(data[2]) , int(data[3])    #(in pixel)

    #Correggo la Z in base alle dimensioni dell'oggetto
    if signal_output[1]==1:                                 #Cocacola
                    signal_output[4]=signal_output[4]-29
    if signal_output[1]==2:                                 #Fanta
                    signal_output[4]=signal_output[4]-29
```

```
    #nutella
    if signal_output[1]==4:                                     #succo
                    signal_output[4]=signal_output[4]-18
    if signal_output[1]==5:                                     #vigorsol
                    signal_output[4]=signal_output[4]-9

    circle = cv2.circle(bgr_frame, center,5,(0,0,0),2)

    point_max= np.zeros((9,3))
    point_min= np.zeros((9,3))

    pixel_max= np.zeros((9,2))
    pixel_min= np.zeros((9,2))

    theta_x= np.zeros(9)
    theta_y= np.zeros(9)
    theta_z= np.zeros(9)

    angle=np.zeros(9)
    raggio=12

    theta_x_avg= np.zeros(20)
    theta_y_avg= np.zeros(20)
    theta_z_avg= np.zeros(20)

    point_1_avg=np.zeros(2)
    point_2_avg=np.zeros(2)

    measure = np.zeros(60)
    circle_point = global_var.define_circle(center)

    i=0
    j=0

    while i<=8:
        theta=0
        angolo_increase=0
        massimo=0
        angolo_del_massimo=0
        time.sleep(0.2)
        ret_loop, depth_frame_loop=rs.define_depth_frame()
```

```
        while j<=59:
            space_depth = py.rs2_deproject_pixel_to_point(_intrinsics,[int(
    circle_point[i,j,1]),int(circle_point[i,j,0])],depth_frame_loop[int(circle_point
    [i,j,1]),int(circle_point[i,j,0])])
            measure[j]= int(-space_depth[2]-z_0)
            if measure[j]>=massimo:
                #determinazione angolo di spam
                if (massimo!=measure[j]):
                    angolo_increase=0
                    massimo=measure[j]
                if(j>=1 and measure[j]==measure[j-1]):
                    angolo_increase+=6
                if(j>1 and measure[j]==measure[j-2] and measure[j]!=measure[j-1]):
                    angolo_increase+=12

                angolo_del_massimo=theta

            j+=1
            theta+=6
        angle[i]=angolo_del_massimo-(angolo_increase/2)

        pixel_max[i,0]=center[0]+((raggio)*math.cos(math.radians(angle[i])))
        pixel_max[i,1]=center[1]+((raggio)*math.sin(math.radians(angle[i])))

        space_max= py.rs2_deproject_pixel_to_point(_intrinsics,[int(pixel_max[i,1]),
    int(pixel_max[i,0])],depth_frame_loop[int(pixel_max[i,1]),int(pixel_max[i,0])])
        point_max[i,0] = int(space_max[1]-x_0)
        point_max[i,1] = int(-space_max[0]-y_0)
        point_max[i,2] = int(-space_max[2]-z_0)

        pixel_min[i,0]=center[0]+((raggio)*math.cos(math.radians(angle[i]+180)))
        pixel_min[i,1]=center[1]+((raggio)*math.sin(math.radians(angle[i]+180)))

        space_min= py.rs2_deproject_pixel_to_point(_intrinsics,[int(pixel_min[i,1]),
    int(pixel_min[i,0])],depth_frame_loop[int(pixel_min[i,1]),int(pixel_min[i,0])])
        point_min[i,0] = int(space_min[1]-x_0)
        point_min[i,1] = int(-space_min[0]-y_0)
        point_min[i,2] = int(-space_min[2]-z_0)

        #Determinazione degli angoli
        #theta_x
```

```
        punto_A= [point_max[i,1],point_max[i,2]]
        punto_B= [point_min[i,1],point_min[i,2]]
        diametro= abs(punto_A[0]-punto_B[0])
        distanza= math.sqrt((punto_A[0]-punto_B[0])**2+(punto_A[1]-punto_B[1])**2)
        if distanza>0:
            theta_yz=math.acos(float(diametro/distanza))
        else:
            theta_yz = 0
        theta_x[i]= math.degrees(theta_yz)

        #theta_y
        punto_A= [point_max[i,0],point_max[i,2]]
        punto_B= [point_min[i,0],point_min[i,2]]
        diametro= abs(punto_A[0]-punto_B[0])
        distanza= math.sqrt((punto_A[0]-punto_B[0])**2+(punto_A[1]-punto_B[1])**2)
        if distanza>0:
            theta_xz=math.acos(float(diametro/distanza))
        else:
            theta_xz = 0
        theta_y[i]= math.degrees(theta_xz)
        if (punto_A[0]>punto_B[0] and punto_A[1]>punto_B[1]):
            theta_y[i]= -theta_y[i]

        #theta_z
        punto_A= [point_max[i,0],point_max[i,1]]
        punto_B= [point_min[i,0],point_min[i,1]]
        diametro= abs(punto_A[0]-punto_B[0])
        distanza= math.sqrt((punto_A[0]-punto_B[0])**2+(punto_A[1]-punto_B[1])**2)
        if distanza>0:
            theta_xy=math.acos(float(diametro/distanza))
        else:
            theta_xy = 0
        theta_z[i]= math.degrees(theta_xy)

        if (punto_A[0]<punto_B[0] and punto_A[1]<punto_B[1]):
            theta_z[i]=-(90-theta_z[i])

        if (punto_A[0]>punto_B[0] and punto_A[1]<punto_B[1]):
            theta_z[i]=90-theta_z[i]

        if (punto_A[0]>punto_B[0] and punto_A[1]>punto_B[1]):
```

```
            theta_z[i]=-(90-theta_z[i])

        if (punto_A[0]<punto_B[0] and punto_A[1]>punto_B[1]):
            theta_z[i]=90-theta_z[i]

        j=0
        i+=1

    theta_x_avg= np.mean(theta_x[:])
    theta_y_avg= np.mean(theta_y[:])
    theta_z_avg= np.mean(theta_z[:])

    angolo_per_disegno=theta_z_avg

    point_1_avg[0]= center[0]+40*math.cos(math.radians(angolo_per_disegno))
    point_1_avg[1]= center[1]+40*math.sin(math.radians(angolo_per_disegno))

    point_2_avg[0]= center[0]+40*math.cos(math.radians(angolo_per_disegno+180))
    point_2_avg[1]= center[1]+40*math.sin(math.radians(angolo_per_disegno+180))

    signal_output[2]= int(signal_output[2])
    signal_output[3]=int(signal_output[3])
    signal_output[4]= int(signal_output[4])

    signal_output[5]= + -16 + int(theta_z_avg)
    signal_output[6]= int(180) #theta_y_avg
    signal_output[7]= + int(theta_x_avg)

    line_avg= cv2.line(bgr_frame,[int(point_1_avg[0]),int(point_1_avg[1])],[int(
    point_2_avg[0]),int(point_2_avg[1])],(0,255,255),2)
    out_path = 'C:/Users/riccardo/Documents/Riccardo_DB/Progetto_Tesi'
    cv2.imwrite(os.path.join(out_path , 'detect_out.jpg'),bgr_frame)

    return signal_output
```

## 5.7 COMMUNICATE_TO_SERVER.PY

```
import socket


```

```
def client_program(data_input):
    #Local
    #host = socket.gethostname()
    #port = 5000

    #KUKA
    host = '172.31.1.10'
    port = 30003

    client_socket = socket.socket()
    client_socket.connect((host, port))

    new_string =""
    for x in data_input:
        new_string += str(x)
        new_string += " "
    print (new_string)

    msg1 = new_string

    client_socket.send(msg1.encode())
    data = client_socket.recv(1024).decode()
    print('Received from server: ' + data)

    client_socket.close()

    return
```

## 5.8 serverCamera.java

```
package RiccardoDB;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.inject.Inject;

import com.kuka.common.ThreadUtil;
```

```
import com.kuka.generated.ioAccess.PinzaIOGroup;
import com.kuka.roboticsAPI.applicationModel.RoboticsAPIApplication;
import static com.kuka.roboticsAPI.motionModel.BasicMotions.*;
import com.kuka.roboticsAPI.deviceModel.LBR;
import com.kuka.roboticsAPI.geometricModel.Frame;

public class ServerCamera extends RoboticsAPIApplication {
    @Inject
    private LBR lBR_iiwa_14_R820_1;

    @Inject
    private PinzaIOGroup pinza;
    private ServerSocket server;
    private Socket s1;
    public DataInputStream in;
    public DataOutputStream out;

    @Override
    public void initialize() {
    }

    @Override
    public void run() throws IOException {
        String rifiuto = "";
        startServer();
        try {

            Frame home_pos = new Frame(-200, -600, 475, Math.toRadians(0.6), Math.toRadians(180), Math.toRadians(0));
            Frame plastica_pos = new Frame(250, -720, 230, Math.toRadians(0), Math.toRadians(180), Math.toRadians(0));
            Frame carta_pos = new Frame(250, -480, 230, Math.toRadians(0), Math.toRadians(180), Math.toRadians(0));
            Frame place_pos = home_pos;

            System.out.println("Homing");
            lBR_iiwa_14_R820_1.move(lin(home_pos).setCartVelocity(50));
            System.out.println("Posizione di Home: " + home_pos.getX() + " " + home_pos.getY() + " " + home_pos.getZ()
                    + " " + Math.toDegrees(home_pos.getAlphaRad()) + " " + Math.toDegrees(home_pos.getBetaRad()) + " "
```

```
                    + Math.toDegrees(home_pos.getGammaRad()));
            System.out.println("Arrivato nella posizione di HOME.");
            System.out.println("Scatto la foto");
            System.out.println("Waiting for client");

            s1 = server.accept();
            System.out.println("Master connection opened");

            // definisci gli stream di input/output
            in = new DataInputStream(s1.getInputStream());
            out = new DataOutputStream(s1.getOutputStream());
            while (true) {
                String raw = "";
                // Leggo tutta la riga
                do {
                    raw += (char) in.readByte();
                } while (s1.getInputStream().available() > 0);
                // Stampo la riga

                System.out.println("Received string ---> " + raw);
                String command = raw;

                command = command.replace(".0", "");

                // definisco la posizione di place
                char c = command.charAt(2);
                int disc = Character.getNumericValue(c);
                switch (disc) {
                    case 1: // cocacola
                        rifiuto = "cocacola";
                        place_pos = plastica_pos;
                        break;
                    case 2: // fanta
                        rifiuto = "fanta";
                        place_pos = plastica_pos;
                        break;
                    case 3: // nutella
                        rifiuto = "nutella";
                        place_pos = plastica_pos;
                        break;
                    case 4: // succo
```

```
                        rifiuto = "succo";
                        place_pos = carta_pos;
                        break;
                    case 5: // vigorsol
                        rifiuto = "vigorsol";
                        place_pos = carta_pos;
                        break;
                    case 6: // NON CI SONO OGGETTI
                        rifiuto = "nessuno";
                        place_pos = home_pos;
                        break;
                }
                if (!(disc == 6)) {
                    System.out.println("Rifiuto riconosciuto: " + rifiuto);

                    // identifico posizione di pick

                    Frame posizione = decode(command);
                    System.out.println("Posizione inviata: " + posizione.getX() + "
" + posizione.getY() + " "
                            + posizione.getZ() + " " + Math.toDegrees(posizione.
getAlphaRad()) + " "
                            + Math.toDegrees(posizione.getBetaRad()) + " " + Math.
toDegrees(posizione.getGammaRad()));

                    Frame pick_pos = new Frame(posizione, 10, 0, -188);

                    System.out.println("Posizione di Pick: " + pick_pos.getX() + " "
 + pick_pos.getY() + " "
                            + pick_pos.getZ() + " " + Math.toDegrees(pick_pos.
getAlphaRad()) + " "
                            + Math.toDegrees(pick_pos.getBetaRad()) + " " + Math.
toDegrees(pick_pos.getGammaRad()));

                    // PICK
                    apriPinza();
                    // Approach di 70mm
                    appro(pick_pos, 0.1, 70);
                    System.out.println("Appro fatto");

                    lBR_iiwa_14_R820_1.move(lin(pick_pos).setCartVelocity(50));
```

```
                    System.out.println("Arrivato nella posizione di pick");

                    // Preparo per output
                    out = new DataOutputStream(s1.getOutputStream());

                    System.out.println("Arrivato alla posizione");
                    out.writeChars("Done: " + rifiuto);

                    // Chiudi Pinza
                    chiudiPinza();
                    ThreadUtil.milliSleep(200);
                    lBR_iiwa_14_R820_1.move(lin(appro_pos).setCartVelocity(50));

                    // Depart di 70mm
                    appro(pick_pos, 0.1, 70);

                    // PLACE
                    appro(place_pos, 0.1, 70);
                    System.out.println("Appro fatto");
                    lBR_iiwa_14_R820_1.move(lin(place_pos).setCartVelocity(50));
                    System.out.println("Arrivato nella posizione di place");
                    apriPinza();
                    ThreadUtil.milliSleep(200);
                    appro(place_pos, 0.1, 70);

                    lBR_iiwa_14_R820_1.move(lin(home_pos).setCartVelocity(50));
                } else {
                    System.out.println("Non ci sono più oggetti");
                }
            }
        } catch (IOException e) {
            System.out.println("Master closed");
        } finally {
            in.close();
            out.close();
            s1.close();
            server.close();
        }
    }

    public void apriPinza() {
```

```
164         if (pinza.getClose()) {
165             pinza.setClose(false);
166         }
167         pinza.setOpen(true);
168     }
169 
170     public void chiudiPinza() {
171         if (pinza.getOpen()) {
172             pinza.setOpen(false);
173         }
174         pinza.setClose(true);
175     }
176 
177     public void appro(Frame terna, double speed, double dz) {
178         Frame app = new Frame(terna, 0, 0, -dz);
179         System.out.println(
180                 terna.getX() + " " + terna.getY() + " " + terna.getZ() + " " + Math.
    toDegrees(terna.getAlphaRad()));
181         System.out.println(app.getX() + " " + app.getY() + " " + app.getZ() + " " +
    Math.toDegrees(app.getAlphaRad()));
182         lBR_iiwa_14_R820_1.move(ptp(app).setJointVelocityRel(speed));
183     }
184 
185     public Frame decode(String command) {
186         // decodifico messaggio
187         String pick_pos = command.substring(4);
188         String[] temp = pick_pos.split(" ");
189         double x = Double.parseDouble(temp[0]);
190         double y = Double.parseDouble(temp[1]);
191         double z = Double.parseDouble(temp[2]);
192         double rz = Double.parseDouble(temp[3]);
193         double ry = Double.parseDouble(temp[4]);
194         double rx = Double.parseDouble(temp[5]);
195         Frame pos = new Frame(x, y, z, Math.toRadians(rz), Math.toRadians(ry),
196                 Math.toRadians(rx));
197         return pos;
198     }
199 
200     private boolean startServer() {
201         try {
202             server = new ServerSocket(30003);
```

```
            server.setSoTimeout(90000);
        } catch (IOException ex) {
            ex.printStackTrace();
            return false;
        }
        System.out.println("Server ready");
        return true;
    }
}
```

# 6 Conclusioni

## 6.1 Limiti di progetto

Si è testato come la rete neurale allenata con lo specifico dataset riesca a riconoscere anche oggetti accartocciati e consumati dall'utilizzo. Questo aspetto e fondamentale in quanto gli oggetti che giungono nei punti di raccolta non presentano certamente superfici regolari.

Vengono di seguito presentati alcuni limiti che il progetto sviluppato porta con sè.

### 6.1.1 Logica nella determinazione dell'orientazione

Inizialmente si è scelto il settore di applicazione concentrandosi sul tema della raccolta differenziata. In questo campo si sono poi dovuti preferire alcuni prodotti per poterli esaminare, inserire nel dataset di training della rete neurale e conseguentemente utilizzare nei test.

I prodotti scelti sono stati selezionati tra tutti quelli possibili, perchè presentavano tutti una comune caratteristica: hanno tutti un asse principale ben definito. Questa peculiarità aiuta la logica per la determinazione degli angoli con cui è orientato l'oggetto nello spazio dato che questa si fonda proprio sulla ricerca dell' asse principale per estrarne le informazioni spaziali.

#### Possibile soluzione

Se ci si pone come obiettivo quello di definire un sistema di visione che permetta un pick degli oggetti in modo più efficiente si può rinunciare, in termini di precisione, al calcolo degli angoli

andando a determinare solamente la rotazione dell'asse Z e definendo l'orientazione dell'oggetto come la rotazione nel piano XY. La determinazione di questo angolo nel piano può avvenire tramite l'utilizzo di logiche che sfruttano puramente il riconoscimento di pattern o punti specifici degli oggetti. Per rendere questo possibile è necessario un uso più massiccio della rete neurale allenandola con diversi label. Come prima conseguenza si deve accettare che il pick degli oggetti avvenga con avvicinamenti alle posizioni sempre con direzioni parallele all'asse Z del robot, perdendo quindi un grado di libertà nei movimenti.

### 6.1.2 Griffe come organo terminale

Il fatto che si sia scelta una pinza parallela come organo terminale obbliga la presa ai soli oggetti che rientrano nelle dimensioni entro la corsa di apertura e chiusura delle griffe. Si è dovuta infatti fare una selezione di oggetti che rientrassero in un certo range di dimensioni.

#### Possibile soluzione

Una possibile soluzione alla problematica è quella di installare come organo terminale un sistema a ventosa che sfrutti il vuoto prodotto dall'aria compressa per la presa dei prodotti di rifiuto da raccogliere. In questo modo risulterebbe possibile prelevare oggetti di dimensioni e forme molto diversi tra loro ampliando la produttività del sistema proposto.

## 6.2 Conclusioni

Questo progetto di tesi mira allo sviluppo di un sistema di visione 3D per bin picking, progettando e validando un metodo che possa essere in grado di fornire le coordinate di posizione e orientazione degli oggetti riconosciuti nello spazio di lavoro fino alla comunicazione e all'attuazione del ciclo di pick-and-place.

Gli obiettivi fissati all'inizio del progetto sono stati raggiunti con successo in quanto si è giunti alla definizione di un software che riesce a fornire le informazioni richieste con una certa precisione permettendo di attuare i movimenti di presa e raccolta dei rifiuti. Il vantaggio del sistema proposto consiste nella possibilità di poter addestrare la rete neurale per il riconoscimento degli oggetti con un numero di immagini relativamente ridotto, ma ottenendo ottimi risultati nell'identificazione. Altra caratteristica di flessibilità è proposta dal fatto che sia possibile con estrema facilità modificare e rafforzare il dataset di classi di prodotti di scarto aggiungendo tutti

quei rifiuti che si dovessero incontrare durante l'utilizzo del sistema. Queste specifiche rendono il sistema adatto alle applicazioni prese in considerazione inizialmente adattandosi a tutti gli ambienti con una finestra di prodotti di scarto noti.

Il pacchetto software proposto, nelle sue diverse sotto sezioni, è risultato consono alle richieste di progetto, fornendo dati e prestazioni efficaci. Studi futuri potrebbero fornire delle logiche migliori nella determinazione delle orientazioni.

Il modello che si è sviluppato in questo progetto mantiene come aspetto fondamentale la peculiarità di mantenersi distaccato rispetto alla scelta applicativa del robot su cui viene implementato.

La progettualità porta con sè la volontà di svincolarsi da un qualsiasi sistema integrato. L' ottica con cui si sono elaborate le varie fasi dello sviluppo del software è stata quella di generalizzare il linguaggio senza ricadere in specifiche relative ad una certa azienda produttrice di sistemi robotici.

La caratteristica di essere un sistema flessibile per diverse celle robotiche è confermata dal fatto che lo sviluppo del codice si articola nella sua totalità all'interno di uno stesso programma Python. La comunicazione dei messaggi avviene tramite protocolli TCP/IP: strumento implementabile in ogni sistema robotico. Eventualmente dovrà essere adattata la sintassi per le diverse applicazioni, ma il codice e il sistema rimangono gli stessi.

# Bibliografia


[1] (2022) Introduction to yolo: Real time object detection. [Online]. Available: https://hashdork.com/yolo/

[2] (2022) Intel realsense depth camera d435. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435

[3] (2023) Lbr iiwa. [Online]. Available: https://www.kuka.com/it-it/prodotti-servizi/sistemi-robot/robot-industriali/lbr-iiwa

[4] (2023) Pinza parallela co-act egp-c. [Online]. Available: https://schunk.com/it/it/sistemi-di-presa/pinza-parallela/co-act-egp-c/c/PGR_3995

[5] (2020) La nuova definizione di rifiuti urbani. [Online]. Available: https://www.lapam.eu/notizie/normative/la-nuova-definizione-dei-rifiuti-urbani/

[6] (2016) Municipal waste management across european countries. [Online]. Available: https://www.eea.europa.eu/publications/municipal-waste-management-across-european-countries

[7] (2022) Eurostat: in europa aumentano i rifiuti urbani. [Online]. Available: https://www.nonsoloambiente.it/eurostat-in-europa-aumentano-i-rifiuti-urbani

[8] J. Bobulski and M. Kubanek, "Waste classification system using image processing and convolutional neural networks," in *Advances in Computational Intelligence: 15th International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, June 12-14, 2019, Proceedings, Part II 15*. Springer, 2019, pp. 350–361.

[9] L. Huiyu, O. G. O, and S.-H. Kim, "Automatic classifications and recognition for recycled garbage by utilizing deep learning technology," in *Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City*, 2019, pp. 1–4.

[10] M. Toğaçar, B. Ergen, and Z. Cömert, "Waste classification using autoencoder network with integrated feature selection method in convolutional neural network models," *Measurement*, vol. 153, p. 107459, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224119313260

[11] S. Faibish, H. Bacakoglu, and A. Goldenberg, "An eye-hand system for automated paper recycling," in *Proceedings of International Conference on Robotics and Automation*, vol. 1, 1997, pp. 9–14 vol.1.

[12] R. Mattone, G. Campagiorni, and A. Wolf, "Fuzzy-based processing of 3d information for items localization in the automated sorting of recyclable packaging," in *1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228)*, vol. 2, 1998, pp. 1613–1618 vol.2.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS'12, vol. 1. Curran Associates Inc., 2012, p. 1097–1105.

[14] M. Abdallah, M. Abu Talib, S. Feroz, Q. Nasir, H. Abdalla, and B. Mahfood, "Artificial intelligence applications in solid waste management: A systematic research review," *Waste Management*, vol. 109, pp. 231–246, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X20302269

[15] H. nan Guo, S. biao Wu, Y. jie Tian, J. Zhang, and H. tao Liu, "Application of machine learning methods for the prediction of organic solid waste treatment and recycling processes: A review," *Bioresource Technology*, vol. 319, p. 124114, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960852420313882

[16] Z. Yang and D. Li, "Wasnet: A neural network-based garbage collection management system," *IEEE Access*, vol. 8, pp. 103 984–103 993, 2020.

[17] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019. [Online]. Available: https://arxiv.org/abs/1905.05055

[18] W.-L. Mao, W.-C. Chen, C.-T. Wang, and Y.-H. Lin, "Recycling waste classification using optimized convolutional neural network," *Resources, Conservation and Recycling*, vol. 164, p. 105132, 2021.

[19] P. F. Proença and P. Simoes, "Taco: Trash annotations in context for litter detection," *arXiv preprint arXiv:2003.06975*, 2020.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[21] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022. [Online]. Available: https://github.com/WongKinYiu/yolov7

[22] (2022) Roboflow: create new project. [Online]. Available: https://app.roboflow.com/thesis-q5dgm

[23] (2022) How to train yolov7 on a custom dataset. [Online]. Available: https://colab.research.google.com/drive/10QHurkk2cdR1G6GE8XGSc7R_TJRzmc33