



DIPARTIMENTO DI INGEGNERIA ELETTRICA  
E TECNOLOGIE DELL'INFORMAZIONE

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

# ELABORATO DI LAUREA IN

## INGEGNERIA

## DELL'AUTOMAZIONE

IMPLEMENTAZIONE DI SEMPLICI TASK DI MOVIMENTAZIONE  
PER UNA PIATTAFORMA MOBILE

**Relatore**

Chiar.mo Prof.  
Gianmaria De Tommasi

**Candidato**

Azzeddine Mouchtakiri  
N39/001708

**Correlatore**

Pietro Spadaccino

Anno Accademico 2024/2025

# Abstract

Il seguente elaborato affronta la progettazione, l'implementazione e l'analisi di un sistema robotico mobile autonomo per il tracking di oggetti. Il sistema è stato sviluppato su una piattaforma di edge computing a basso costo. L'architettura del sistema si basa su un controllo distribuito a due livelli, che distingue un nodo Master e un nodo Slave, e che vede un single-board computer Raspberry Pi 3 Model B+ quale nodo di elaborazione principale e un microcontrollore Arduino UNO R3 dedicato all'attuazione a bassa latenza e alla gestione della sicurezza. Il nucleo sperimentale di questo progetto consiste nell'implementazione e nell'analisi comparativa di due distinti approcci di visione artificiale: un approccio di computer vision classica, basato sulla segmentazione cromatica e ottimizzato per l'efficienza computazionale, e un approccio basato su deep learning, che impiega una rete neurale convoluzionale (EfficientDet-Lite0) per il riconoscimento degli oggetti. Vengono analizzate in modo approfondito le prestazioni di entrambe le modalità in termini di throughput (FPS), latenza e robustezza qualitativa.

# Indice

<b>Abstract</b>	<b>1</b>
<b>Nomenclatura:</b>	<b>6</b>
<b>1 Introduzione</b>	<b>8</b>
1.1 Contesto Tecnologico e Motivazioni del Progetto . . . . .	8
1.2 Obiettivi e Contributo Scientifico della Tesi . . . . .	9
1.3 Struttura della Tesi . . . . .	10
<b>2 Architettura del Sistema e Piattaforme Tecnologiche</b>	<b>11</b>
2.1 Logica Progettuale: un'Architettura Ibrida a Controllo Distribuito . .	11
2.2 Breve Storia delle Piattaforme: Raspberry Pi e Arduino . . . . .	12
2.3 Analisi Dettagliata delle Piattaforme Tecnologiche Selezionate . . .	13
2.3.1 Il Nodo di Elaborazione Edge: Raspberry Pi 3 Model B+ . . . .	13
2.3.2 Il Controllore di Attuazione: Arduino UNO R3 . . . . .	14
2.3.3 Il Framework di Visione e Inferenza . . . . .	14
2.4 Protocollo di Comunicazione Edge-Actuator . . . . .	14
2.5 Principi di Controllo e Interfacciamento . . . . .	15
2.5.1 Cinematica a Guida Differenziale . . . . .	15
2.5.2 Acquisizione Dati da Sensore Ultrasonico HC-SR04 . . . . .	16
<b>3 Progetto Hardware</b>	<b>17</b>
3.1 Selezione e Descrizione dei Componenti . . . . .	17
3.2 Schema Elettrico e Logica di Interfacciamento . . . . .	19
3.2.1 Interfacciamento del Driver Motori L298N . . . . .	20
3.2.2 Interfacciamento del Sensore HC-SR04 . . . . .	20
3.2.3 Assemblaggio Meccanico e Considerazioni sul Design . . . . .	21
<b>4 Progetto Software</b>	<b>23</b>
4.1 Firmware del Controllore di Attuazione (Arduino) . . . . .	23

4.1.1	Struttura e Definizioni Globali . . . . .	23
4.1.2	Loop Principale e Logica di Sicurezza . . . . .	24
4.1.3	Implementazione della Cinematica a Guida Differenziale . . . . .	25
4.2	Software del Nodo Edge (Raspberry Pi) . . . . .	26
4.2.1	Codice Sorgente Completo e Commentato . . . . .	27
4.3	Modalitá di Esecuzione dello Script Python . . . . .	38
4.3.1	Esecuzione in Modalitá Tracking Cromatico . . . . .	38
4.3.2	Esecuzione in Modalitá Machine Learning . . . . .	38
<b>5</b>	<b>Strategie di Visione e Logica di Tracking</b>	<b>40</b>
5.1	Approccio 1: Tracking Reattivo tramite Segmentazione Cromatica . . . . .	40
5.1.1	La Scelta dello Spazio Colore HSV . . . . .	40
5.1.2	Pipeline di Elaborazione Cromatica . . . . .	41
5.2	Principi Algebrici e Analitici della Visione Artificiale Classica . . . . .	42
5.2.1	Conversione Spazio Colore BGR a HSV . . . . .	43
5.2.2	Filtraggio Gaussiano e Operazioni Morfologiche . . . . .	43
5.2.3	Estrazione Contorni e Analisi Geometrica . . . . .	44
5.3	Approccio 2: Tracking Cognitivo tramite Inferenza Deep Learning . . . . .	45
5.3.1	Scelta del Modello: EfficientDet-Lite0 . . . . .	45
5.3.2	Pipeline di Elaborazione con TFLite Support . . . . .	46
<b>6</b>	<b>Integrazione e Metodologia di Test</b>	<b>50</b>
6.1	Integrazione dei Sottosistemi Hardware e Software . . . . .	50
6.2	Metodologia di Test Sperimentale . . . . .	51
6.2.1	Test di Performance Quantitativa (Throughput e Latenza) . . . . .	51
6.2.2	Test dei Target e Osservazione del Comportamento . . . . .	52
<b>7</b>	<b>Risultati Sperimentali e Discussione</b>	<b>53</b>
7.1	Risultati Quantitativi: Analisi del Throughput e della Latenza . . . . .	53
7.2	Risultati Qualitativi: Analisi del Comportamento di Tracking . . . . .	54
7.2.1	Performance del Tracking Cromatico (con pallina gialla) . . . . .	54
7.2.2	Performance del Tracking con Machine Learning (con persona e cellulare) . . . . .	55
7.3	Discussione Critica dei Risultati . . . . .	56
<b>8</b>	<b>Conclusioni e Sviluppi Futuri</b>	<b>58</b>
8.1	Sintesi del Lavoro e Conclusioni Finali . . . . .	58
8.2	Proposte per Sviluppi Futuri . . . . .	58

# Elenco delle figure

3.1	Schema dei collegamenti elettrici realizzato in Fritzing. Sono visibili le connessioni tra Arduino UNO, il driver L298N, i motori e il sensore di distanza. . . . .	19
3.2	Vista assonometrica del modello CAD 3D del prototipo robotico, realizzato in SolidWorks. Il design integra il telaio a guida differenziale e il braccio manipolatore. . . . .	21
3.3	Vista reale del prototipo robotico, realizzato. Il robot integra il telaio a guida differenziale e il braccio manipolatore. . . . .	22
5.1	Fasi chiave della pipeline di tracking cromatico: (a) la segmentazione del colore isola i pixel di interesse; (b) l'analisi dei contorni permette di localizzare l'oggetto sul frame originale. . . . .	41
7.1	Confronto degli FPS medi tra le modalità di tracciamento HSV e ML.. Dati dalla tabella 7.1 . . . . .	54
7.2	Confronto della latenza media tra le modalità di tracciamento HSV e ML. Dati dalla Tabella 7.1. . . . .	55
7.3	Esempi di rilevamento in modalità Machine Learning. Il sistema dimostra la capacità di riconoscere e classificare correttamente oggetti di diverse categorie semantiche. . . . .	56

# **Elenco delle tabelle**

3.1 Elenco dettagliato dei componenti hardware e loro specifiche tecniche.	17
7.1 Risultati comparativi delle performance quantitative tra le due modalità di tracking.	53

# Nomenclatura:

Di seguito viene fornita una lista delle principali nomenclature, abbreviazioni e acronimi utilizzati nel presente elaborato.

**AI** Artificial Intelligence (Intelligenza Artificiale)

**BGR** Blue, Green, Red (Spazio colore)

**CAD** Computer-Aided Design

**CNN** Convolutional Neural Network (Rete Neurale Convoluzionale)

**COCO** Common Objects in Context (Dataset)

**CPU** Central Processing Unit (Unità di Elaborazione Centrale)

**FLOPs** Floating Point Operations (Operazioni in virgola mobile)

**FPS** Frames Per Second (Fotogrammi al Secondo)

**GDL** Gradi di Libertà

**GPIO** General-Purpose Input/Output

**HSV** Hue, Saturation, Value (Spazio colore)

**IA** Intelligenza Artificiale

**IoT** Internet of Things

**MCU** Microcontroller Unit (Unità a Microcontrollore)

**ML** Machine Learning (Apprendimento Automatico)

**NAS** Neural Architecture Search

**NMS** Non-Maximum Suppression

**OpenCV** Open Source Computer Vision Library

**PWM** Pulse Width Modulation (Modulazione di Larghezza di Impulso)

**RAM** Random Access Memory

**RTOS** Real-Time Operating System (Sistema Operativo in Tempo Reale)

**SBC** Single-Board Computer

**SoC** System-on-a-Chip

**TFLite** TensorFlow Lite

**TPU** Tensor Processing Unit

**URLLC** Ultra-Reliable Low-Latency Communication

# Capitolo 1

## Introduzione

### 1.1 Contesto Tecnologico e Motivazioni del Progetto

L'attuale panorama tecnologico è caratterizzato da una profonda trasformazione guida dalla convergenza di discipline un tempo separate. La robotica [1], l'Internet of Things (IoT) [2] e l'Intelligenza Artificiale (IA) [3] non sono più domini isolati, ma pilastri interconnessi del paradigma noto come Industria 4.0 [4]. In questo scenario, i sistemi robotici mobili stanno evolvendo da semplici automi programmati per eseguire task ripetitivi a veri e propri agenti autonomi, dotati della capacità di percepire, ragionare e agire in ambienti complessi e dinamici. Questa evoluzione è resa possibile da un importante cambiamento architettonico: il passaggio dall'elaborazione centralizzata su cloud a un modello di **edge computing** [5].

L'edge computing consiste nell'eseguire l'elaborazione dei dati il più vicino possibile alla fonte di generazione, ovvero "sul bordo" (edge) della rete. Per un robot mobile, l'edge è il robot stesso. Questa decentralizzazione è decisiva nel superamento dei limiti intrinseci del cloud, quali la latenza di rete, la dipendenza dalla connettività e le problematiche di privacy. Per applicazioni come il tracking di oggetti in tempo reale, dove una decisione deve essere resa possibile in frazioni di secondo, affidarsi a un server remoto non è una soluzione praticabile. In questo elaborato, l'elaborazione dell'immagine e la logica decisionale sono state eseguite internamente al Raspberry Pi, senza dipendenza da risorse cloud. Questo metodo di approcciare è un esempio di edge computing, dove il processamento avviene localmente, andando a ridurre la latenza e interazione tra la visione e l'azione.

Infatti la democratizzazione dell'hardware, guidata da piattaforme open-source come Raspberry Pi [6] e Arduino [7], ha reso la prototipazione di tali sistemi ed-

ge accessibile a ricercatori, studenti e hobbisti. Malgrado ciò, questa accessibilità porta con sé una sfida ingegneristica significativa: come eseguire algoritmi di IA, notoriamente esigenti in termini di risorse, su dispositivi a basso costo e basso consumo energetico? L'uso di modelli di deep learning efficienti, come la famiglia EfficientDet-Lite, è una strategia chiave per affrontare questa sfida, poiché questi modelli sono progettati per offrire un buon compromesso tra accuratezza e requisiti computazionali su dispositivi mobili, come dimostrato da lavori recenti in ambiti quali la navigazione assistita per persone con disabilità visive [8]

L'elaborato si inserisce esattamente in questo solco. La motivazione principale non è la semplice creazione di un robot, ma l'utilizzo del robot come **piattaforma sperimentale per investigare i compromessi fondamentali dell'Edge IA**. Nella **5G Academy** è stata maturata un'esperienza, che ha fornito un'ispirazione decisiva in tal senso, mettendo in luce l'importanza critica delle comunicazioni a bassa latenza (URLLC - Ultra-Reliable Low-Latency Communication) e dell'elaborazione distribuita. Questo progetto si propone di tradurre tali concetti in una realizzazione pratica, approfondendo come diverse strategie di visione artificiale impattano sulla reattività e sull'intelligenza di un sistema autonomo operante con vincoli hardware reali.

## 1.2 Obiettivi e Contributo Scientifico della Tesi

Gli obiettivi di questo elaborato sono stati definiti per essere sia pratici che analitici, al fine di fornire un contributo tangibile e ben documentato.

1. **Progettazione e Realizzazione di una Piattaforma Robotica Modulare:** Il primo obiettivo è di natura ingegneristica e consiste nella costruzione di un prototipo robotico mobile completo e funzionante. Il sistema si basa su un'architettura ibrida, con un telaio a guida differenziale, sensori di prossimità e un sistema di visione. La modularità dell'hardware e del software è un obiettivo importante, che permette al progetto di essere versatile per futuri esperimenti.
2. **Implementazione di un Sistema di Visione Ibrido:** Il secondo obiettivo è lo sviluppo di un software di controllo che implementi due paradigmi di tracking di oggetti, attivabili in modo selettivo:

- Una **pipeline di computer vision classica**, basata sulla segmentazione cromatica nello spazio colore HSV, scelta come baseline per la sua massima efficienza computazionale.
  - Una **pipeline di deep learning**, che utilizza un modello di rete neurale convoluzionale pre-addestrato (EfficientDet-Lite0) per il riconoscimento semantico di oggetti, rappresentativa di un approccio "intelligente".
3. **Analisi Comparativa e Quantificazione dei Trade-off:** Il contributo scientifico principale della tesi risiede nel terzo obiettivo. Si mira a condurre un'analisi sperimentale rigorosa per confrontare le due strategie di visione secondo metriche oggettive e soggettive. Si intende quantificare il **trade-off tra performance (misurata in FPS e latenza) e robustezza (valutata qualitativamente in scenari di stress)**. L'obiettivo è rispondere alla domanda: "In un contesto di edge computing, quale approccio offre il miglior bilanciamento per un dato caso d'uso?"

### 1.3 Struttura della Tesi

Il presente documento è organizzato in nove capitoli per guidare il lettore attraverso il processo di ricerca, progettazione e analisi.

- **Capitolo 2:** Illustra l'architettura logica del sistema, definendo il modello di controllo distribuito e i flussi di comunicazione.
- **Capitolo 3:** Documenta il progetto hardware, includendo l'elenco dei componenti, lo schema elettrico dettagliato e le considerazioni sulla gestione dell'alimentazione.
- **Capitolo 4:** Analizza l'implementazione del software, sia il firmware per l'Arduino che l'applicazione Python ibrida per il Raspberry Pi.
- **Capitolo 5:** Si concentra sull'analisi approfondita delle due pipeline di visione.
- **Capitolo 6:** Presenta la metodologia di integrazione e di test, e riporta i risultati sperimentali, sia quantitativi che qualitativi.
- **Capitolo 7:** Discute criticamente i risultati ottenuti, analizzando i trade-off emersi e confrontando le performance con sistemi simili.
- **Capitolo 8:** Trae le conclusioni finali del lavoro e propone diverse direzioni per futuri sviluppi e miglioramenti del prototipo.

# Capitolo 2

## Architettura del Sistema e Piattaforme Tecnologiche

### 2.1 Logica Progettuale: un'Architettura Ibrida a Controllo Distribuito

La progettazione di sistemi robotici autonomi per applicazioni reali, specialmente su piattaforme a risorse limitate, richiede un'architettura che bilanci efficacemente la complessità computazionale con la necessità di un controllo reattivo e sicuro. Un approccio monolitico, in cui un singolo processore gestisce sia la "percezione" che l'"azione", è intrinsecamente fragile su dispositivi come il Raspberry Pi. L'esecuzione di un sistema operativo multitasking come Linux, sebbene flessibile, introduce latenze non deterministiche (jitter) che possono compromettere la stabilità del controllo fisico. Per superare queste limitazioni, per questo progetto è stata adottata una **architettura a controllo distribuito a due livelli gerarchici**. Questa scelta progettuale, ampiamente validata nella letteratura scientifica sui sistemi embedded e robotici [9, 10], si basa sul principio del disaccoppiamento funzionale. I compiti vengono suddivisi in base ai loro requisiti temporali e computazionali:

- **Livello Superiore (Cognitivo/Strategico):** Affidato a un Single-Board Computer (SBC), questo livello gestisce le operazioni complesse che non richiedono un determinismo temporale stretto. Le sue responsabilità includono l'analisi di flussi di dati sensoriali complessi (immagini), l'esecuzione di modelli di intelligenza artificiale, la pianificazione di traiettorie e la formulazione di comandi strategici di alto livello (es. "insegna il target a sinistra").

- **Livello Inferiore (Reattivo/Real-Time):** Affidato a un microcontrollore (MCU), questo livello si occupa esclusivamente dell’interfacciamento diretto con l’hardware (attuatori e sensori semplici) e dell’esecuzione di loop di controllo a tempo reale. La sua architettura bare-metal (o con un Real-Time Operating System, RTOS) garantisce risposte a bassa latenza e prevedibili, fondamentali per la sicurezza e la stabilità del sistema.

Questa architettura non solo ottimizza l’uso delle risorse, ma crea un sistema intrinsecamente più robusto. Come verrà dettagliato, il controllore di basso livello può implementare meccanismi di sicurezza (come un arresto di emergenza basato su un sensore di prossimità) che operano indipendentemente dallo stato del livello superiore, garantendo una rete di sicurezza fondamentale.

## 2.2 Breve Storia delle Piattaforme: Raspberry Pi e Arduino

Il **Raspberry Pi** è una serie di computer a scheda singola (SBC) a basso costo, delle dimensioni approssimativamente di una carta di credito, sviluppati nel Regno Unito dalla Raspberry Pi Foundation. È stato originariamente concepito nel 2012 con l’obiettivo principale di promuovere l’insegnamento dell’informatica di base nelle scuole e di rendere il computing più accessibile e divertente per persone di tutte le età. Il suo successo ha superato di gran lunga le aspettative iniziali. Dalla sua prima versione, il Raspberry Pi si è evoluto attraverso diverse generazioni (Raspberry Pi 1, 2, 3, 4, oltre a modelli più piccoli e specializzati come il Raspberry Pi Zero e il microcontrollore Raspberry Pi Pico). Ogni nuova iterazione ha generalmente portato miglioramenti significativi in termini di potenza di calcolo, memoria RAM e connettività, mantenendo al contempo un’ampia disponibilità di pin General-Purpose Input/Output (GPIO) per l’interfacciamento hardware. Il sistema operativo più comune è Raspberry Pi OS, una distribuzione Linux ottimizzata. La sua natura open-source e il robusto supporto per linguaggi di programmazione come Python lo rendono estremamente flessibile. [6]

Anche **Arduino** è una piattaforma di prototipazione open-source, basata su hardware e software flessibili e facili da usare. È nata nel 2005 in Italia con l’obiettivo di fornire un modo semplice ed economico per studenti e non professionisti di creare dispositivi interattivi o prototipi elettronici. Si basa su microcontrollori ATmega di Atmel e offre un ambiente di sviluppo integrato (IDE) semplificato. La sua

popolarità è dovuta alla sua semplicità, al costo contenuto e alla vasta comunità di supporto, rendendola ideale per progetti didattici e amatoriali, ma anche per prototipazione rapida in ambito professionale. [7]

Entrambe le piattaforme sono diventate pilastri del "physical computing" e dell'elettronica digitale, democratizzando l'accesso a sistemi embedded e all'IoT, e permettendo lo sviluppo di prototipi complessi con una barriera d'ingresso molto bassa.

## 2.3 Analisi Dettagliata delle Piattaforme Tecniche Selezionate

### 2.3.1 Il Nodo di Elaborazione Edge: Raspberry Pi 3 Model B+

Per il ruolo di nodo cognitivo è stato selezionato un Raspberry Pi 3 Model B+. Sebbene esistano versioni più recenti, questo modello rappresenta un eccellente compromesso tra costo, disponibilità e potenza di calcolo sufficiente per l'inferenza di modelli di IA leggeri. Le sue caratteristiche tecniche salienti sono:

- **Processore:** Un System-on-a-Chip (SoC) Broadcom BCM2837B0, che integra una CPU ARM Cortex-A53 quad-core a 64-bit operante a 1.4 GHz. La presenza di più core è fondamentale per questo progetto, poiché permette di dedicare thread specifici a compiti paralleli, come la gestione della comunicazione seriale e l'elaborazione video.
- **Memoria:** 1 GB di RAM LPDDR2, una risorsa critica per la gestione di frame video e per il caricamento in memoria dei modelli di machine learning.
- **Sistema Operativo:** Il dispositivo esegue Raspberry Pi OS (precedentemente Raspbian), una distribuzione Linux basata su Debian. Questo fornisce un ambiente di sviluppo potente e flessibile, con accesso a un vasto repository di pacchetti software e a strumenti di scripting avanzati.

La scelta del Raspberry Pi è quindi motivata dalla sua natura di vero e proprio computer, capace di supportare lo stack software necessario per la visione artificiale moderna, un compito irrealizzabile per un semplice microcontrollore [11].

### 2.3.2 Il Controllore di Attuazione: Arduino UNO R3

Per il controllo a basso livello è stato scelto un Arduino UNO, una piattaforma basata sul microcontrollore a 8-bit ATmega328P. Le sue caratteristiche lo rendono ideale per il ruolo di "slave" reattivo:

- **Esecuzione Deterministica:** L'assenza di un sistema operativo multitasking assicura che il codice ("sketch") venga eseguito in un loop prevedibile e con tempi di risposta costanti. Questo è essenziale per generare segnali di controllo precisi per i motori.
- **Semplicità di Interfacciamento Hardware:** Arduino espone pin di General-Purpose Input/Output (GPIO) che permettono un interfacciamento diretto e semplice con componenti elettronici come driver per motori e sensori.
- **Robustezza:** La sua architettura semplice è intrinsecamente robusta e meno soggetta a crash software, rendendolo un componente affidabile per la gestione della sicurezza fisica.

### 2.3.3 Il Framework di Visione e Inferenza

- **OpenCV (versione 4.x):** È la spina dorsale di tutta l'elaborazione delle immagini. Fornisce implementazioni ottimizzate per operazioni come la conversione degli spazi colore (`cv2.cvtColor`) e l'analisi morfologica [12].
- **TensorFlow Lite Runtime:** È la versione "edge" del framework TensorFlow di Google [13]. Il suo scopo è permettere l' **inferenza** di modelli di deep learning. Un concetto chiave abilitato da TFLite è la **quantizzazione**, un processo che converte i pesi a 32-bit del modello in interi a 8-bit, riducendo drasticamente le dimensioni del file e accelerando l'esecuzione su CPU a basso consumo [14, 15].

## 2.4 Protocollo di Comunicazione Edge-Actuator

La comunicazione tra il Raspberry Pi (master) e l'Arduino (slave) avviene tramite un'interfaccia seriale su USB, orchestrata da un protocollo applicativo custom.

- **Canale Fisico e Logico:** A livello fisico, si utilizza un cavo USB. A livello di sistema operativo, il driver dell'Arduino crea una porta seriale virtuale sul Pi (es. `/dev/ttyACM0`), gestita in Python [16] tramite la libreria **PySerial**.

- **Definizione del Protocollo:** È stato definito un protocollo **testuale, asincrono e basato su comandi**. Ogni comando è una stringa ASCII (es. "avanti"), terminata da un carattere di newline (\n), che agisce come delimitatore di pacchetto. L'Arduino legge la seriale fino a incontrare il carattere \n e interpreta la stringa. Questo approccio è robusto e semplifica enormemente il debug.

## 2.5 Principi di Controllo e Interfacciamento

Per una comprensione approfondita del sistema, è fondamentale analizzare i principi matematici e ingegneristici che sottostanno al controllo del movimento e all'acquisizione sensoriale.

### 2.5.1 Cinematica a Guida Differenziale

La piattaforma mobile utilizza una configurazione a guida differenziale, comune in molti robot mobili. Questo significa che il movimento del robot è determinato dalle velocità relative delle due ruote motrici indipendenti (destra e sinistra). La semplicità implementativa di un controllo on/off (avanti, indietro, stop per ciascuna ruota) per generare movimenti di base è la seguente:

- **Movimento in avanti (vaiAvanti()):** Entrambe le ruote girano in avanti alla stessa velocità. Matematicamente, se  $v_L$  è la velocità della ruota sinistra e  $v_R$  è la velocità della ruota destra, allora  $v_L = v_R = V_{max}$ , dove  $V_{max}$  è la velocità massima raggiungibile dalle ruote con la tensione applicata. La velocità lineare del robot  $V$  sarà  $V = V_{max}$  e la velocità angolare  $\omega = 0$ .
- **Movimento all'indietro (vaiIndietro()):** Entrambe le ruote girano all'indietro alla stessa velocità:  $v_L = v_R = -V_{max}$ . La velocità lineare del robot sarà  $V = -V_{max}$  e  $\omega = 0$ .
- **Sterzata sul posto a sinistra (giraSinistra()):** La ruota destra gira in avanti e la ruota sinistra gira all'indietro:  $v_R = V_{max}$ ,  $v_L = -V_{max}$ . Questo genera una rotazione sul posto. La velocità lineare del robot è  $V = 0$ , mentre la velocità angolare  $\omega$  dipende da  $V_{max}$  e dalla distanza tra le ruote.
- **Sterzata sul posto a destra (giraDestra()):** La ruota destra gira all'indietro e la ruota sinistra gira in avanti:  $v_R = -V_{max}$ ,  $v_L = V_{max}$ . Simile alla sterzata sinistra, con rotazione in direzione opposta.

- **Stop (`stopMotori()`):** Entrambe le ruote sono ferme:  $v_L = v_R = 0$ . La velocità lineare e angolare sono entrambe zero.

In questo progetto, poiché il controllo della velocità è binario (on/off) e non proporzionale (PWM), le equazioni cinematiche inverse per il controllo preciso di traiettorie complesse non sono direttamente applicate, ma il principio di controllo differenziale rimane valido per i movimenti discreti implementati.

### 2.5.2 Acquisizione Dati da Sensore Ultrasonico HC-SR04

Il sensore HC-SR04 opera basandosi sul principio del tempo di volo (Time-of-Flight - ToF) degli ultrasuoni. La distanza di un oggetto viene calcolata misurando il tempo impiegato da un'onda sonora per viaggiare dal sensore all'oggetto e tornare indietro [17]. La sequenza operativa è:

1. Il microcontrollore (Arduino) invia un impulso HIGH di 10 microsecondi al pin `Trig` del sensore. Questo avvia l'emissione di 8 cicli di ultrasuoni a 40 kHz.
2. Il sensore emette gli ultrasuoni e mette il pin `Echo` in stato HIGH.
3. Quando l'onda sonora torna indietro e viene rilevata dal ricevitore, il pin `Echo` torna in stato LOW.
4. L'Arduino misura la durata dell'impulso HIGH sul pin `Echo` (il tempo di volo,  $t$ ).

La distanza  $D$  viene quindi calcolata utilizzando la formula:

$$D = \frac{v_s \times t}{2}$$

dove  $v_s$  è la velocità del suono nell'aria (circa 343 metri al secondo a 20°C, ovvero 0.0343 cm/μs) e  $t$  è il tempo di volo misurato. Il tempo viene diviso per 2 perché rappresenta il viaggio di andata e ritorno dell'onda. La soglia di sicurezza imposta (20 cm) permette al microcontrollore di attivare un override di sicurezza, fermando il robot prima di una potenziale collisione.

# Capitolo 3

## Progetto Hardware

### 3.1 Selezione e Descrizione dei Componenti

La realizzazione fisica del prototipo si basa su una selezione mirata di componenti elettronici e meccanici, scelti per ottimizzare il rapporto tra costo, performance e facilità di integrazione. Ogni componente svolge un ruolo specifico all'interno dell'architettura descritta nel capitolo precedente. La Tabella 3.1 fornisce un elenco completo e una descrizione tecnica per ciascun elemento.

Tabella 3.1: Elenco dettagliato dei componenti hardware e loro specifiche tecniche.

Componente	Descrizione Tecnica e Ruolo nel Progetto
Raspberry Pi 3 Model B+	Unità di calcolo principale (SBC) basata su SoC Broadcom BCM2837B0 (CPU ARM Cortex-A53 quad-core a 1.4 GHz, 1GB RAM LPDDR2). Esegue il sistema operativo Raspberry Pi OS e ospita lo stack software di alto livello per la visione artificiale e la logica decisionale.
Arduino UNO R3	Piattaforma a microcontrollore basata su ATmega328P a 16 MHz con 2KB di SRAM. Agisce come controllore di basso livello, gestendo l'attuazione dei motori e la lettura dei sensori in tempo reale. La sua architettura garantisce un'esecuzione deterministica del firmware.

*Continua nella pagina successiva*

**Tabella 3.1 continua dalla pagina precedente**

<b>Componente</b>	<b>Descrizione Tecnica e Ruolo nel Progetto</b>
Driver Motori L298N	Modulo basato sul circuito integrato a ponte H duale L298N. È in grado di pilotare due motori DC indipendentemente, controllandone direzione e velocità (quest'ultima tramite segnali PWM, anche se in questo progetto si è utilizzato un controllo on/off). Supporta tensioni per i motori fino a 35V e correnti fino a 2A.
Motori DC	Quattro motori a corrente continua operanti a 6V, ciascuno dotato di un riduttore meccanico integrato. Il riduttore diminuisce la velocità di rotazione dell'albero di uscita, aumentando al contempo la coppia. Questa caratteristica è indispensabile per garantire al robot la forza necessaria per muoversi e superare piccole imperfezioni della superficie. Nel sistema implementato, tutti e quattro i motori vengono attivati per i movimenti in avanti e indietro. Per le manovre di sterzata (sinistra e destra), la logica di controllo implementa il principio della guida differenziale, attivando selettivamente i motori su ciascun lato (es. motori a destra in avanti e motori a sinistra indietro, o viceversa) per ottenere la rotazione desiderata.
Sensore Ultrasuoni HC-SR04	Sensore di prossimità che opera secondo il principio del sonar. Emette un breve impulso sonoro a 40 kHz e misura il tempo trascorso per la ricezione dell'eco. La distanza dall'ostacolo viene calcolata dalla formula $distanza = (velocità\_del\_suono \times tempo)/2$ . È utilizzato come sistema anti-collisione primario.
Webcam USB	Dispositivo di acquisizione video standard UVC (USB Video Class), con risoluzione impostata a 640x480 pixel e un frame rate nominale di 30 FPS. Funge da "occhio" del robot, fornendo il flusso di dati visivi al Raspberry Pi.

*Continua nella pagina successiva*

**Tabella 3.1 continua dalla pagina precedente**

Componente	Descrizione Tecnica e Ruolo nel Progetto
Power Bank	Fonte di alimentazione esterna da 20000 mAh, capace di erogare una corrente stabile di 5V / 2.4A. È dedicata all'alimentazione del solo Raspberry Pi e della webcam, per isolarlo dal rumore elettrico generato dai motori.
Pacco Batterie 4xAA	Portabatterie contenente 4 pile stilo Alcaline da 1.5V in serie, per una tensione totale di 6V. Questa fonte di alimentazione è dedicata esclusivamente al circuito di potenza del driver L298N per alimentare i motori.

## 3.2 Schema Elettrico e Logica di Interfacciamento

L'interconnessione dei componenti è un aspetto critico del progetto. Lo schema elettrico completo, realizzato con il software Fritzing, è riportato in Figura 3.1. Di seguito si analizza la logica di collegamento per ciascun sottosistema.

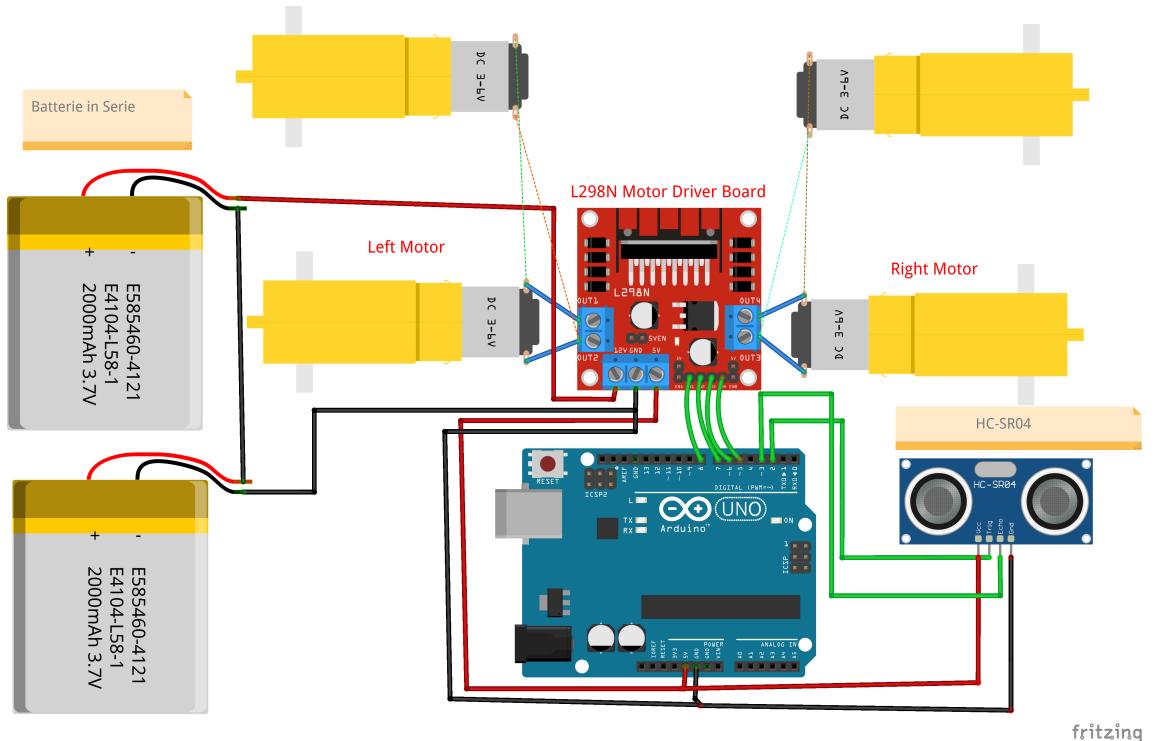


Figura 3.1: Schema dei collegamenti elettrici realizzato in Fritzing. Sono visibili le connessioni tra Arduino UNO, il driver L298N, i motori e il sensore di distanza.

### 3.2.1 Interfacciamento del Driver Motori L298N

Il modulo L298N è il ponte tra la logica a bassa corrente dell'Arduino e i motori ad alta corrente.

- **Alimentazione Logica:** Il pin **VCC** del driver è collegato al pin **5V** dell'Arduino. Questo alimenta la logica interna del chip L298N.
- **Alimentazione di Potenza:** Il pin **VS** (o **+12V** in alcune versioni) è collegato al polo positivo del pacco batterie da 6V. Il pin **GND** del driver è collegato sia al polo negativo del pacco batterie sia a un pin **GND** dell'Arduino. Quest'ultimo collegamento, detto **massa comune**, è fondamentale per garantire che i segnali logici inviati dall'Arduino siano interpretati correttamente dal driver.
- **Collegamenti di Controllo:** Per la guida differenziale, i motori su ciascun lato del robot sono controllati come un'unica unità.
  - I pin **IN1** e **IN2** del driver, che controllano il motore (o i motori) del lato sinistro, sono collegati a due pin digitali dell'Arduino.
  - I pin **IN3** e **IN4** del driver, che controllano il motore del lato destro, sono collegati ad altri due pin digitali dell'Arduino.
- **Abilitazione Motori:** I pin **ENA** e **ENB**, che abilitano rispettivamente ciascun motore, sono stati collegati direttamente a un pin **5V** tramite un jumper, per mantenerli costantemente attivi. Sebbene questi pin possano essere usati per il controllo della velocità tramite PWM, in questo progetto si è optato per un controllo on/off per semplicità.

### 3.2.2 Interfacciamento del Sensore HC-SR04

Il sensore a ultrasuoni richiede quattro collegamenti:

- **VCC** collegato al pin **5V** dell'Arduino.
- **GND** collegato a un pin **GND** dell'Arduino.
- **Trig** (Trigger) collegato a un pin digitale dell'Arduino, configurato come **OUTPUT**. L'invio di un impulso HIGH su questo pin avvia la misurazione.
- **Echo** collegato a un altro pin digitale dell'Arduino, configurato come **INPUT**. L'Arduino misura la durata dell'impulso HIGH su questo pin per calcolare la distanza.

### 3.2.3 Assemblaggio Meccanico e Considerazioni sul Design

Il prototipo è stato assemblato su un telaio a 4 ruote motrici, tale configurazione che, sebbene non strettamente onnidirezionale con il controllo implementato, offre una buona manovrabilità per la guida differenziale a causa dei 4 motori DC con riduttore. Il modello CAD 3D completo del robot, realizzato in SolidWorks, è mostrato in Figura 3.2. Questo modello include anche un braccio manipolatore a 4 Gradi di Libertà (GDL), mentre il prototipo invece realizzato e assemblato è mostrato in Figura 3.3, che sebbene non controllato in questo lavoro di tesi, fa parte della piattaforma fisica e rappresenta un'importante via per futuri sviluppi, come discusso nel capitolo finale. La disposizione dei componenti sul telaio è stata ottimizzata al meglio per bilanciare il peso e posizionare anche la camera frontalmente per una percezione ottimale, attraverso un Pan-Tilt stampato in 3D, per posizionarla e anche movimentarla con l'uso di alcuni servomotori.

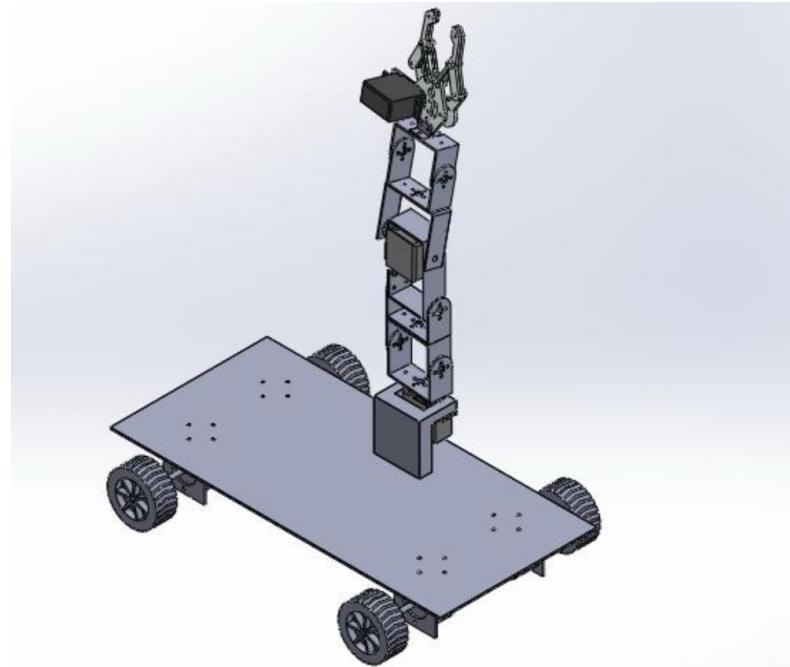


Figura 3.2: Vista assonometrica del modello CAD 3D del prototipo robotico, realizzato in SolidWorks. Il design integra il telaio a guida differenziale e il braccio manipolatore.

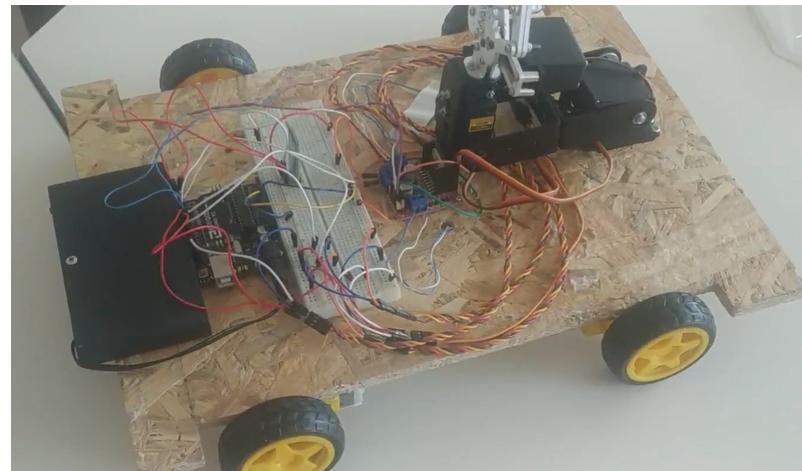


Figura 3.3: Vista reale del prototipo robotico, realizzato. Il robot integra il telaio a guida differenziale e il braccio manipolatore.

# Capitolo 4

## Progetto Software

Il software del sistema è il cuore del progetto, responsabile della trasformazione dei dati sensoriali in azioni meccaniche. Esso è suddiviso in due componenti distinte ma strettamente interconnesse, che riflettono l'architettura hardware a due livelli: il **firmware a bassa latenza** per il microcontrollore Arduino e l' **applicazione di alto livello** per il single-board computer Raspberry Pi. Questa sezione analizza in dettaglio l'implementazione, la struttura e la logica di entrambi.

### 4.1 Firmware del Controllore di Attuazione (Arduino)

Il firmware sviluppato per l'Arduino UNO è stato scritto in C++, utilizzando il framework Wiring e l'IDE ufficiale di Arduino. È stato progettato con tre obiettivi primari: reattività, semplicità e sicurezza. Il suo ruolo non è quello di "pensare", ma di agire come un esecutore fedele e istantaneo dei comandi impartiti dal Raspberry Pi, garantendo al contempo un livello di sicurezza hardware indipendente.

#### 4.1.1 Struttura e Definizioni Globali

Il codice inizia con la definizione delle costanti che mappano i pin fisici dell'Arduino alle funzioni logiche. Questa pratica migliora la leggibilità e la manutenibilità del codice. Vengono definiti i pin per il controllo dei due motori del driver motori L298N (che gestiscono il lato destro e sinistro del robot) e i pin per il sensore a ultrasuoni. Viene inoltre impostata una soglia di sicurezza critica, come mostrato nel Listato 4.1.

```

2 // --- CONFIGURAZIONE PIN MOTORI (Guida Differenziale) ---
3 // Motori LATO DESTRO
4 const int IN1_DESTRA = 7;
5 const int IN2_DESTRA = 8;
6 // Motori LATO SINISTRA
7 const int IN1_SINISTRA = 9;
8 const int IN2_SINISTRA = 10;
9
10 // --- CONFIGURAZIONE SENSORE ULTRASUONI ---
11 const int TRIG_PIN = 2;
12 const int ECHO_PIN = 3;
13 // Il robot si ferma se un ostacolo e' a meno di 20cm
14 const long DISTANZA_SICUREZZA_CM = 20;

```

Listing 4.1: Definizione delle costanti e dei pin nel firmware Arduino.

La funzione `setup()`, eseguita una sola volta all'accensione, inizializza la comunicazione seriale alla stessa velocità di baud del Raspberry Pi (9600 bps) e configura tutti i pin di controllo come `OUTPUT`, assicurando che il robot parta in uno stato di arresto con la chiamata a `stopMotori()`.

#### 4.1.2 Loop Principale e Logica di Sicurezza

Il ciclo principale `loop()` implementa un automa a stati finiti implicito, dove la sicurezza ha la priorità assoluta. Ad ogni iterazione, la prima operazione è interrogare il sensore di distanza. Se un ostacolo viene rilevato all'interno della soglia di sicurezza, il firmware entra in uno stato di "override": arresta i motori, notifica il Raspberry Pi via seriale e ignora qualsiasi altro comando, come si vede nel Listato 4.2.

```

1 void loop() {
2     // 1. Controllo di sicurezza prioritario
3     if (leggiDistanza() < DISTANZA_SICUREZZA_CM) {
4         stopMotori();
5         // Invia un messaggio di avviso solo se non e' gi'a
6         // stato inviato
7         // per evitare di intasare la seriale.
8         if (!Serial.find("OBJ_VICINO")) {
9             Serial.println("OBJ_VICINO");
10        }
11        delay(50); // Piccolo delay per debounce

```

```

11     return; // Termina l'iterazione, ignorando i comandi dal
12     Pi
13
14 // 2. Esecuzione dei comandi solo se la via e' libera
15 if (Serial.available() > 0) {
16     String command = Serial.readStringUntil('\n');
17     command.trim();
18
19     if (command == "avanti")      { vaiAvanti(); }
20     else if (command == "indietro") { vaiIndietro(); }
21     else if (command == "sinistra") { giraSinistra(); }
22     else if (command == "destra")   { giraDestra(); }
23     else { // Qualsiasi altro comando, incluso "stop",
24         arresta i motori
25         stopMotori();
26     }
27 }
```

Listing 4.2: Logica del loop principale con priorita' alla sicurezza.

#### 4.1.3 Implementazione della Cinematica a Guida Differenziale

Il controllo del movimento si basa su una cinematica a guida differenziale. Le funzioni di movimento traducono i comandi astratti in segnali digitali per il driver L298N. Per esempio, per una sterzata a sinistra sul posto, i motori del lato destro vengono fatti girare in avanti, mentre quelli del lato sinistro vengono fatti girare all'indietro (Listato 4.3). Però nella pratica, il comportamento del sistema può discostarsi dal caso ideale a causa di limitazioni economiche e altri fattori, poiché la sterzata sul posto funziona secondo il principio teorico, ma richiede calibrazione e accorgimenti per migliorare la stabilità e la precisione del movimento.

```

1 void stopMotori() {
2     digitalWrite(IN1_DESTRA, LOW); digitalWrite(IN2_DESTRA, LOW
3     );
4     digitalWrite(IN1_SINISTRA, LOW); digitalWrite(IN2_SINISTRA,
5     LOW);
```

```

4 }
5
6 void vaiAvanti() {
7     digitalWrite(IN1_DESTRA, HIGH); digitalWrite(IN2_DESTRA,
8         LOW); // Destra avanti
9     digitalWrite(IN1_SINISTRA, HIGH); digitalWrite(IN2_SINISTRA
10    , LOW); // Sinistra avanti
11 }
12
13 void vaiIndietro() {
14     digitalWrite(IN1_DESTRA, LOW); digitalWrite(IN2_DESTRA,
15         HIGH); // Destra indietro
16     digitalWrite(IN1_SINISTRA, LOW); digitalWrite(IN2_SINISTRA,
17         HIGH); // Sinistra indietro
18 }
19
20
21 void giraDestra() { // Gira sul posto a destra
22     digitalWrite(IN1_DESTRA, LOW); digitalWrite(IN2_DESTRA,
23         HIGH); // Destra indietro
24     digitalWrite(IN1_SINISTRA, HIGH); digitalWrite(IN2_SINISTRA
25         , LOW); // Sinistra avanti
26 }
27
28 void giraSinistra() { // Gira sul posto a sinistra
29     digitalWrite(IN1_DESTRA, HIGH); digitalWrite(IN2_DESTRA,
30         LOW); // Destra avanti
31     digitalWrite(IN1_SINISTRA, LOW); digitalWrite(IN2_SINISTRA,
32         HIGH); // Sinistra indietro
33 }
34

```

Listing 4.3: Implementazione delle funzioni di movimento.

## 4.2 Software del Nodo Edge (Raspberry Pi)

Prima di analizzare l'applicazione principale, è fondamentale descrivere l'ambiente di sviluppo e la metodologia di lavoro adottata per il Raspberry Pi, che ha operato in modalità "headless" (senza monitor o periferiche collegate) per l'intero ciclo di vita del progetto.

**Configurazione dell’Ambiente e Accesso Remoto** La configurazione iniziale del sistema operativo (Raspberry Pi OS) ha previsto l’abilitazione di due servizi essenziali tramite l’utility `raspi-config`: l’interfaccia della telecamera (CSI) e il server SSH (Secure Shell). L’interazione con il dispositivo è avvenuta esclusivamente tramite la rete locale Wi-Fi, utilizzando due strumenti standard:

- **SSH:** Per l’accesso a linea di comando, utilizzato per la modifica dei file, l’installazione delle dipendenze, l’avvio degli script e il monitoraggio dei log.
- **VNC (Virtual Network Computing):** Per l’accesso all’interfaccia grafica remota, indispensabile per il debug visivo in tempo reale dell’output della finestra di OpenCV (`cv2.imshow()`).

**Gestione delle Dipendenze con Ambienti Virtuali** Per garantire la stabilità e la riproducibilità del software, è stato creato un ambiente virtuale Python tramite il modulo `venv`. Questo approccio isola le librerie del progetto (come OpenCV, NumPy, pyserial e tflite-support) da quelle del sistema, prevenendo conflitti di versione. Per la modalità di Machine Learning, è stato seguito l’approccio raccomandato da Google, clonando il repository ufficiale `tensorflow/examples` ed eseguendo lo script di setup fornito (`setup.sh`), che garantisce la corretta installazione di tutte le dipendenze e delle funzioni di utilità per il modello EfficientDet-Lite0. Lo script principale del progetto è stato quindi eseguito dall’interno di questa struttura di cartelle per assicurare la corretta risoluzione dei percorsi.

Il software di alto livello, eseguito sul Raspberry Pi, è stato sviluppato in Python3. È un’applicazione complessa che gestisce la visione artificiale, la logica decisionale, la comunicazione e la visualizzazione. Per questo motivo, è stato strutturato con un approccio orientato agli oggetti per garantire modularità e manutenibilità. Di seguito viene presentato e commentato il codice completo dello script finale.

#### 4.2.1 Codice Sorgente Completo e Commentato

Il Listato 4.4 mostra il codice completo dello script Python `robot_finale.py`, che integra entrambe le modalità di tracking (cromatica e Machine Learning) e la logica di controllo del robot. Durante l’implementazione del sistema sono state utilizzate le seguenti tecnologie e fonti documentative:

- La libreria OpenCV per l’elaborazione delle immagini, in particolare per il filtraggio e la rilevazione dei colori [18, 19].

- Il modello pre-addestrato EfficientDet-Lite0 fornito da TensorFlow Hub per il riconoscimento degli oggetti [20].
- La guida ufficiale di TensorFlow Lite per la quantizzazione post-training e l'utilizzo del modello su dispositivi embedded [21].
- Il sensore a ultrasuoni HC-SR04, configurato seguendo le indicazioni disponibili su MakersLab [17].
- Il dispositivo Raspberry Pi, utilizzato come piattaforma hardware principale [22].
- Le librerie standard di Python, tra cui `pyserial` e `threading`, documentate nella reference ufficiale [23].

```

1 import cv2
2 import numpy as np
3 import serial
4 import time
5 import logging
6 import threading
7 import argparse
8 import sys
9
10 # Importazione condizionale delle librerie TFLite
11 try:
12     from tflite_support.task import core
13     from tflite_support.task import processor
14     from tflite_support.task import vision
15     TFLITE_SUPPORT_AVAILABLE = True
16 except ImportError:
17     TFLITE_SUPPORT_AVAILABLE = False
18
19 # --- CONFIGURAZIONE GLOBALE ---
20 logging.basicConfig(level=logging.INFO, format='%(asctime)s -
21
22 # -- Impostazioni Arduino e Robot --
23 SERIAL_PORT = '/dev/ttyACM0'
24 BAUD_RATE = 9600
25 CMD_FORWARD = "avanti\n"
26 CMD_STOP = "stop\n"
27 CMD_LEFT = "sinistra\n"
```

```

28 CMD_RIGHT = "destra\n"
29 CMD_BACKWARD = "indietro\n"
30 DISTANCE_CLOSE_MSG = "OBJ_VICINO"
31
32 # -- Impostazioni Video --
33 FRAME_WIDTH = 640
34 FRAME_HEIGHT = 480
35
36 # -- Configurazione per Modalita' 'color' --
37 LOWER_YELLOW = np.array([15, 100, 100])
38 UPPER_YELLOW = np.array([35, 255, 255])
39 MIN_RADIUS_COLOR = 10
40
41 # -- Configurazione per Modalita' 'ml' --
42 MODEL_ML = 'efficientdet_lite0.tflite'
43 TARGET_LABEL_ML = 'person'
44 NUM_THREADS_ML = 4
45
46 # -- Logica di Controllo Comune --
47 TARGET_RADIUS = 80
48 RADIUS_TOLERANCE = 20
49 CENTER_ZONE_PERCENT = 0.35
50
51 # -- Variabili Globali per la Comunicazione --
52 arduino = None
53 object_close_flag = threading.Event()
54 stop_thread = threading.Event()
55
56 # --- FUNZIONI DI COMUNICAZIONE CON ARDUINO ---
57 def send_command(cmd):
58     # Funzione globale per inviare comandi all'Arduino
59     if arduino and arduino.is_open:
60         try:
61             arduino.write(cmd.encode('utf-8'))
62             logging.info(f"Comando inviato: {cmd.strip()}")
63         except: pass
64
65 def serial_reader_thread():
66     # Thread che legge costantemente dalla seriale per

```

```

messaggi di sicurezza
67    global arduino
68    while not stop_thread.is_set():
69        try:
70            if arduino and arduino.in_waiting > 0:
71                line = arduino.readline().decode('utf-8').
72                strip()
73                if line == DISTANCE_CLOSE_MSG:
74                    object_close_flag.set()
75                else: object_close_flag.clear()
76            except serial.SerialException as e:
77                logging.error(f"Errore di lettura seriale: {e}")
78                break # Esci dal thread in caso di errore critico
79                seriale
80            except Exception as e:
81                logging.error(f"Errore generico nel thread
82                seriale: {e}")
83                break
84            time.sleep(0.05)

# --- CLASSE PRINCIPALE DI CONTROLLO ---
85 class HybridRobotTracker:
86     def __init__(self, mode, model_path=None, target_label=
87     None):
88         self.mode = mode
89         self.target_label = target_label
90         self.motors_running = False
91         self.motors_current_command = CMD_STOP # Inizializza
92         il comando corrente dei motori

if self.mode == 'ml':
93     if not TFLITE_SUPPORT_AVAILABLE:
94         raise ImportError("Libreria tflite-support
95         non trovata. Assicurati che sia installata (es. 'pip
install tflite-support').")
96
base_options = core.BaseOptions(file_name=
model_path, num_threads=NUM_THREADS_ML)
detection_options = processor.DetectionOptions(

```

```

    max_results=5, score_threshold=0.3)
96
        options = vision.ObjectDetectorOptions(
97    base_options=base_options, detection_options=
98    detection_options)
99
100       self.detector = vision.ObjectDetector.
101      create_from_options(options)

102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127

```

- max\_results=5, score\_threshold=0.3)
- options = vision.ObjectDetectorOptions(
- base\_options=base\_options, detection\_options=
- detection\_options)
- self.detector = vision.ObjectDetector.
- create\_from\_options(options)
- 
- 
- def** find\_target(self, frame):
- # Delega alla funzione di ricerca corretta in base alla modalita'
- if self.mode == 'color':
- return self.\_find\_target\_by\_color(frame)
- elif self.mode == 'ml':
- return self.\_find\_target\_by\_ml(frame)
- return None, None
- 
- def** \_find\_target\_by\_color(self, frame):
- # Implementa la pipeline di visione classica
- blurred = cv2.GaussianBlur(frame, (11, 11), 0)
- hsv = cv2.cvtColor(blurred, cv2.COLOR\_BGR2HSV)
- mask = cv2.inRange(hsv, LOWER\_YELLOW, UPPER\_YELLOW)
- mask = cv2.erode(mask, None, iterations=2)
- mask = cv2.dilate(mask, None, iterations=2)
- contours, \_ = cv2.findContours(mask.copy(), cv2.
- RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)
- 
- target\_info = None
- if contours:
- c = max(contours, key=cv2.contourArea)
- ((x, y), radius) = cv2.minEnclosingCircle(c)
- if radius > MIN\_RADIUS\_COLOR:
- target\_info = {'center': (int(x), int(y)),
- 'radius': int(radius)}
- 
- visualize(frame, [], self.mode, target\_info)
- return target\_info is not None, target\_info
- 
- def** \_find\_target\_by\_ml(self, frame):
- # Implementa la pipeline di machine learning

```

128         rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
129         input_tensor = vision.TensorImage.create_from_array(
130             rgb_image)
131
132         detection_result = self.detector.detect(input_tensor)
133
134
135         # Cerca il target principale tra i rilevamenti
136         for detection in detection_result.detections:
137             category = detection.categories[0]
138             if category.category_name == self.target_label:
139                 target_detected = True
140                 bbox = detection.bounding_box
141                 center_x = bbox.origin_x + bbox.width / 2
142                 center_y = bbox.origin_y + bbox.height / 2
143                 radius = (bbox.width + bbox.height) / 4
144                 target_info = {'center': (int(center_x), int(
145                     center_y)), 'radius': int(radius)}
146
147                 visualize(frame, detection_result, self.mode, None,
148 self.target_label)
149
150             def control_robot(self, target_detected, target_info):
151                 # Logica decisionale per il movimento del robot
152                 command = CMD_STOP
153
154                 if object_close_flag.is_set():
155                     logging.warning("Ostacolo rilevato, robot fermo
156 per sicurezza.")
157
158                     command = CMD_STOP
159
160                 elif target_detected and target_info:
161                     center_x = target_info['center'][0]
162                     radius = target_info['radius']
163
164                     center_zone_width = FRAME_WIDTH *
165 CENTER_ZONE_PERCENT
166
167                     left_bound = (FRAME_WIDTH - center_zone_width) /

```

```

2
162         right_bound = left_bound + center_zone_width
163
164     # La logica da' priorita' alla sterzata
165     if center_x < left_bound:
166         command = CMD_LEFT
167     elif center_x > right_bound:
168         command = CMD_RIGHT
169     # Se centrato, regola la distanza
170     elif radius < TARGET_RADIUS - RADIUS_TOLERANCE:
171         command = CMD_FORWARD
172     elif radius > TARGET_RADIUS + RADIUS_TOLERANCE:
173         command = CMD_BACKWARD
174     else: # Target centrato e a distanza ottimale
175         command = CMD_STOP
176
177     else: # Target non rilevato
178         command = CMD_STOP
179
180     # Gestisce lo stato dei motori per evitare comandi
181     # ridondanti
182     # Invia comando solo se c'\e un cambio di stato o se
183     # il comando non \e STOP (per invii continui)
184     if (command != CMD_STOP and command != self.
185 motors_current_command) or \
186         (command == CMD_STOP and self.
187 motors_current_command != CMD_STOP):
188         send_command(command)
189         self.motors_current_command = command
190
191     # Aggiorna lo stato di funzionamento dei motori
192     self.motors_running = (command != CMD_STOP)
193
194 # --- FUNZIONI DI VISUALIZZAZIONE ---
195 def visualize(image, detection_result, mode,
196 color_target_info=None, ml_target_label=None):
197     # Questa funzione \e responsabile di disegnare i

```

```

bounding box, cerchi e testo
# sull'immagine per la visualizzazione a schermo. Il
codice effettivo
# di disegno \`e omesso per brevità in questo listato.

194 if mode == 'color' and color_target_info:
195     center = color_target_info['center']
196     radius = color_target_info['radius']
197     cv2.circle(image, center, radius, (0, 255, 0), 2)
198     cv2.circle(image, center, 5, (0, 0, 255), -1)
199     cv2.putText(image, f"Raggio: {radius}", (10, 30), cv2
200 .FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
201 elif mode == 'ml' and detection_result:
202     for detection in detection_result.detections:
203         bbox = detection.bounding_box
204         category = detection.categories[0]
205
206         # Disegna il rettangolo di contorno
207         start_point = (bbox.origin_x, bbox.origin_y)
208         end_point = (bbox.origin_x + bbox.width, bbox.
209 origin_y + bbox.height)
210         color = (255, 0, 0) # Blu
211         thickness = 2
212         cv2.rectangle(image, start_point, end_point,
213 color, thickness)
214
215         # Scrivi l'etichetta e il punteggio di confidenza
216         label = f'{category.category_name} ({category.
217 score:.2f})'
218         cv2.putText(image, label, (bbox.origin_x, bbox.
219 origin_y - 10),
220             cv2.FONT_HERSHEY_SIMPLEX, 0.7, color,
221             2)
222
223         cv2.imshow('Robot Vision', image)

# --- FUNZIONE PRINCIPALE DI AVVIO ---
def main():
    parser = argparse.ArgumentParser(formatter_class=argparse
        .ArgumentDefaultsHelpFormatter)

```

```

224     parser.add_argument('--mode', type=str, required=True,
225                         choices=['color', 'ml'], help='Modalit\`a di tracking: "color" o "ml".')
226     parser.add_argument('--model', type=str, default=MODEL_ML,
227                         help='Percorso del modello TFLite (solo per modalita "ml").')
228
229     parser.add_argument('--targetLabel', type=str, default=
230                         TARGET_LABEL_ML, help='Etichetta dell\'oggetto da
231                         tracciare (solo per modalita "ml").')
232
233     args = parser.parse_args()
234
235
236     # Inizializzazione Hardware
237     global arduino
238     try:
239         arduino = serial.Serial(SERIAL_PORT, BAUD_RATE,
240                               timeout=1)
241
242         time.sleep(2) # Attesa per la stabilizzazione della
243                     # connessione seriale
244
245         threading.Thread(target=serial_reader_thread, daemon=
246                           True).start()
247
248         logging.info(f"Connessione Arduino stabilita su {SERIAL_PORT}")
249
250     except serial.SerialException as e:
251
252         logging.critical(f"ERRORE: Connessione Arduino
253                         fallita su {SERIAL_PORT}. Errore: {e}")
254
255         sys.exit("Impossibile connettersi all'Arduino.
256                  Assicurati che sia collegato e che la porta seriale sia
257                  corretta.")
258
259     except Exception as e:
260
261         logging.critical(f"ERRORE: Errore generico nell'
262                         inizializzazione seriale: {e}")
263
264         sys.exit("Errore critico durante l'inizializzazione
265                         dell'Arduino.")
266
267
268     # Inizializzazione Tracker
269     try:
270
271         tracker = HybridRobotTracker(mode=args.mode,
272                                     model_path=args.model, target_label=args.targetLabel)

```

```

247         logging.info(f"Tracker inizializzato in modalità: {args.mode}")
248     except Exception as e:
249         logging.critical(f"ERRORE: Inizializzazione tracker fallita. Errore: {e}")
250         # Assicurati di chiudere la seriale in caso di errore critico prima di uscire
251         if arduino and arduino.is_open:
252             arduino.close()
253             sys.exit("Errore critico durante l'inizializzazione del tracker. Termino l'esecuzione.")
254         return # Non dovrebbe essere raggiunto a causa di sys.exit
255
256     # Inizializzazione Camera
257     cap = cv2.VideoCapture(0) # 0 è l'ID predefinito per la webcam
258     if not cap.isOpened():
259         logging.critical("ERRORE: Impossibile aprire la webcam. Controlla il collegamento o i permessi.")
260         # Chiudi la seriale e esci in caso di errore camera
261         stop_thread.set()
262         if arduino and arduino.is_open:
263             arduino.close()
264             sys.exit("Errore: Webcam non accessibile. Termino l'esecuzione.")
265
266     cap.set(cv2.CAP_PROP_FRAME_WIDTH, FRAME_WIDTH)
267     cap.set(cv2.CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT)
268     logging.info(f"Webcam inizializzata a {FRAME_WIDTH}x{FRAME_HEIGHT}")
269
270     start_time = time.time()
271     frame_count = 0
272     tracker.motors_current_command = CMD_STOP # Inizializza lo stato del comando motore
273
274     logging.info("Avvio del loop principale del robot...")
275     while True:

```

```

276         ret, frame = cap.read()
277
278     if not ret:
279
280         logging.error("Impossibile leggere il frame dalla"
281                     "webcam. Riprovo...")
282
283         time.sleep(0.1) # Breve attesa prima di riprovare
284
285         continue # Prova a leggere il prossimo frame
286
287
288     target_detected, target_info = tracker.find_target(
289     frame)
290
291     tracker.control_robot(target_detected, target_info)
292
293
294     frame_count += 1
295     elapsed_time = time.time() - start_time
296
297     if elapsed_time >= 1.0: # Calcola FPS ogni secondo
298
299         fps = frame_count / elapsed_time
300
301         logging.info(f"FPS: {fps:.2f}")
302
303         frame_count = 0
304
305         start_time = time.time()
306
307
308     # Interruzione con tasto 'q'
309     if cv2.waitKey(1) & 0xFF == ord('q'):
310
311         logging.info("Tasto 'q' premuto. Terminazione"
312                     "applicazione.")
313
314         break
315
316
317     # Pulizia risorse alla fine del programma
318     cap.release()
319
320     cv2.destroyAllWindows()
321
322     stop_thread.set() # Segnala al thread seriale di
323                     terminare
324
325     send_command(CMD_STOP) # Invia un comando di stop finale
326
327     if arduino:
328
329         arduino.close()
330
331         logging.info("Connessione seriale chiusa.")
332
333         logging.info("Applicazione terminata.")
334
335
336 if __name__ == '__main__':
337
338     main()

```

---

Listing 4.4: Codice sorgente completo del software su Raspberry Pi.

## 4.3 Modalità di Esecuzione dello Script Python

Lo script principale del sistema, `robot_finale.py`, è stato sviluppato per essere eseguito da linea di comando sulla piattaforma Raspberry Pi. La flessibilità del sistema risiede nella possibilità di selezionare la modalità di visione tramite un argomento da linea di comando (`--mode`), permettendo di passare agilmente tra la pipeline di computer vision classica e quella basata su deep learning.

Per avviare il sistema, è necessario accedere al Raspberry Pi tramite SSH e navigare nella directory che contiene lo script `robot_finale.py`. Successivamente, si possono utilizzare i seguenti comandi per selezionare la modalità desiderata:

### 4.3.1 Esecuzione in Modalità Tracking Cromatico

Per attivare il sistema utilizzando la pipeline di visione basata sulla segmentazione cromatica, eseguire il seguente comando nel terminale:

```
1 python robot_finale.py --mode color
```

Listing 4.5: Avvio dello script in modalità tracking cromatico

In questa configurazione, il sistema utilizzerà gli algoritmi di OpenCV per identificare il target basandosi sul colore.

### 4.3.2 Esecuzione in Modalità Machine Learning

Per attivare il sistema utilizzando la pipeline basata su deep learning con il modello EfficientDet-Lite0, eseguire il seguente comando nel terminale:

```
1 python robot_finale.py --mode ml --model efficientdet_lite0.tflite --targetLabel person
```

Listing 4.6: Avvio dello script in modalità machine learning

Questo comando specifica la modalità `ml`, il percorso del modello TFLite (`efficientdet_lite0.tflite`) e l'etichetta dell'oggetto target da riconoscere (`person`). Assicurarsi che il file del modello sia presente nella directory di esecuzione dello script o specificare il percorso completo al file. È possibile modificare l'argomento `-targetLabel` per tracciare altri oggetti supportati dal modello pre-addestrato (es. `cell phone`, `cup`, ecc.).

**Interruzione dell'Esecuzione:** Una volta avviato il sistema, sia in modalità cromatico che machine learning, l'elaborazione continuerà fino a quando non verrà interrotta manualmente. Per terminare l'applicazione, è sufficiente premere il tasto q sulla tastiera oppure CTRL(Control)+C mentre la finestra di visualizzazione della webcam (**Robot Vision**) è attiva.

# Capitolo 5

## Strategie di Visione e Logica di Tracking

Questo capitolo costituisce il nucleo tecnico-scientifico del progetto, analizzando in profondità le due pipeline di visione artificiale implementate. Sebbene entrambe mirino allo stesso obiettivo finale – localizzare un target in un’immagine – i loro approcci metodologici sono diametralmente opposti. La loro analisi comparativa permette di far emergere i compromessi fondamentali tra l’efficienza della computer vision classica e l’intelligenza semantica del deep learning in un contesto di edge computing.

### 5.1 Approccio 1: Tracking Reattivo tramite Segmentazione Cromatica

Questa strategia rappresenta la baseline prestazionale del sistema. È definita "reattiva" poiché non possiede alcuna comprensione del contenuto dell’immagine, ma reagisce unicamente a una caratteristica fisica di basso livello: il colore. La sua implementazione è interamente basata su funzioni ottimizzate della libreria OpenCV [12,18], garantendo un’esecuzione estremamente rapida sulla CPU del Raspberry Pi.

#### 5.1.1 La Scelta dello Spazio Colore HSV

Il primo passo di ogni pipeline di visione è l’acquisizione del frame, che per default avviene nel formato BGR (Blue, Green, Red). Tuttavia, lo spazio colore BGR è altamente sensibile alle variazioni di illuminazione. Una stessa superficie gialla può

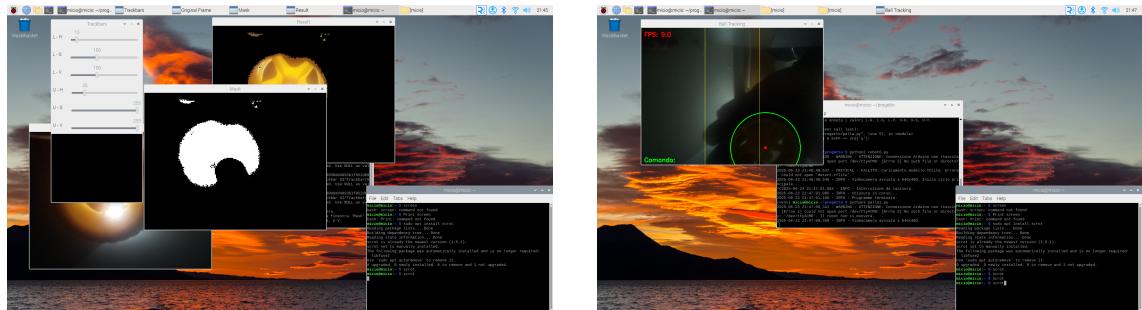
apparire con valori BGR molto diversi se illuminata da una luce al neon o dalla luce solare diretta. Per ovviare a questa instabilità, il frame viene immediatamente convertito nello spazio colore HSV (Hue, Saturation, Value) tramite la funzione `cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)`.

- **Hue (H - Tonalità):** Rappresenta il colore puro (es. rosso, giallo, verde) ed è espresso come un angolo su un cerchio cromatico (da 0 a 179 in OpenCV per adattarsi a un byte). Questa componente è relativamente insensibile alle variazioni di luminosità.
- **Saturation (S - Saturazione):** Indica l'intensità o la "purezza" del colore. Un valore basso indica un colore tendente al grigio, un valore alto un colore vivido.
- **Value (V - Valore/Luminosità):** Rappresenta la luminosità del colore, da scuro (nero) a chiaro (bianco).

Lavorare in HSV permette di definire un range per il solo canale Hue, rendendo il rilevamento del giallo molto più robusto in diverse condizioni di luce.

### 5.1.2 Pipeline di Elaborazione Cromatica

La pipeline, illustrata schematicamente in Figura 5.1, si articola in passaggi sequenziali.



(a) Maschera binaria dopo la sogliatura HSV.  
(b) Rilevamento finale con cerchio di inclusione.

Figura 5.1: Fasi chiave della pipeline di tracking cromatico: (a) la segmentazione del colore isola i pixel di interesse; (b) l'analisi dei contorni permette di localizzare l'oggetto sul frame originale.

1. **Sogliatura (Thresholding):** Utilizzando `cv2.inRange`, si applica una sogliatura simultanea sui tre canali HSV. Tutti i pixel i cui valori H, S e V

rientrano nei range predefiniti (`LOWER_YELLOW` e `UPPER_YELLOW`) vengono impostati a 255 (bianco) in una nuova immagine, la maschera, mentre tutti gli altri vengono impostati a 0 (nero). Il risultato, visibile in Figura 5.1a, è una rappresentazione binaria della scena dove sono evidenziate solo le aree potenzialmente appartenenti al target.

2. **Pulizia Morfologica:** La maschera binaria ottenuta può presentare del rumore: piccoli puntini bianchi isolati dovuti a riflessi o imperfezioni del sensore, o piccoli "buchi" neri all'interno dell'area del target. Per migliorare la qualità della maschera, vengono applicate delle operazioni morfologiche. Un'operazione di **erosione** (`cv2.erode`) viene applicata per alcune iterazioni per rimuovere le piccole macchie di rumore. Successivamente, un'operazione di **dilatazione** (`cv2.dilate`) per lo stesso numero di iterazioni viene usata per ricompattare e chiudere i buchi nell'area del target principale, che potrebbe essere stata leggermente ridotta dall'erosione. [19]
3. **Estrazione e Analisi dei Contorni:** Sulla maschera pulita, la funzione `cv2.findContours` viene utilizzata per trovare i contorni di tutte le aree bianche connesse. Questa funzione restituisce una lista di tutti i "blob" (oggetti binari) presenti. Assumendo che il target di interesse sia l'oggetto giallo più grande nell'inquadratura, si itera su questa lista per trovare il contorno con l'area maggiore (`cv2.contourArea`). [19]
4. **Localizzazione Finale:** Una volta identificato il contorno principale, si utilizza la funzione `cv2.minEnclosingCircle` per calcolare il cerchio più piccolo che lo contiene. Questa funzione restituisce il centro  $(x, y)$  e il raggio di tale cerchio, fornendo una stima robusta e standardizzata della posizione e della dimensione apparente del target, come mostrato in Figura 5.1b.

## 5.2 Principi Algebrici e Analitici della Visione Artificiale Classica

Per comprendere a fondo le capacità e i limiti delle strategie di visione classica implementate, è essenziale addentrarsi nei principi matematici e ingegneristici che ne governano il funzionamento.

### 5.2.1 Conversione Spazio Colore BGR a HSV

Le immagini vengono acquisite nel formato BGR (Blue, Green, Red), dove ogni pixel è rappresentato da una combinazione di intensità dei canali blu, verde e rosso. Questo spazio colore è intuitivo ma altamente sensibile alle variazioni di illuminazione ambientale: una piccola variazione nella fonte luminosa può alterare drasticamente i valori BGR di un colore percepito come costante dall'occhio umano. Come descritto in [24] la conversione allo spazio colore HSV (Hue, Saturation, Value) mitiga questa problematica separando la cromaticità (Hue e Saturation) dalla luminosità (Value). Le formule di conversione da RGB (o BGR, che è equivalente a RGB con i canali invertiti) a HSV sono non lineari e coinvolgono:

- **Value (V):** Rappresenta la luminosità ed è dato dal valore massimo tra le componenti R, G, B normalizzate.
- **Saturation (S):** Indica la "purezza" o l'intensità del colore. È calcolata come una funzione della differenza tra il massimo e il minimo delle componenti normalizzate, divisa per il valore massimo ( $V$ ).
- **Hue (H):** L'angolo della tonalità cromatica, calcolato dalla relazione tra le componenti normalizzate. Varia da  $0^\circ$  a  $360^\circ$  (ma mappato a  $0 - 179$  in OpenCV per stare in un byte).

La chiave analitica di HSV per il tracking cromatico risiede nel canale Hue, che è relativamente invariante rispetto ai cambiamenti di illuminazione. Questo permette di definire un range specifico per il giallo (o qualsiasi altro colore) che rimane valido anche se la luce si attenua o si intensifica, a patto che il colore non diventi così scuro o sbiadito da perdere significativamente saturazione o luminosità, oltre tutto rimane un metodo, che può essere alterato facilmente da altri elementi simili ect.

### 5.2.2 Filtraggio Gaussiano e Operazioni Morfologiche

- **Filtraggio Gaussiano:** L'applicazione di un filtro Gaussiano (`cv2.GaussianBlur`) è una delle operazioni di levigatura (smoothing) più comuni in visione artificiale. È un'operazione di convoluzione, dove un "kernel" (o maschera) Gaussiano viene applicato a ogni pixel dell'immagine. Il kernel è una matrice di pesi generata da una funzione Gaussiana bidimensionale, che assegna un peso maggiore al pixel centrale e pesi che diminuiscono gradualmente all'aumentare della distanza dal centro. Il Gaussian Blur viene implementato tramite convoluzione discreta[ [25], [26]] Matematicamente, per un'immagine  $I(x, y)$  e un

kernel Gaussiano  $G(u, v)$ :

$$I'_{x,y} = \sum_{u=-k}^k \sum_{v=-k}^k I_{x+u,y+v} \cdot G_{u,v}$$

dove  $(2k + 1) \times (2k + 1)$  è la dimensione del kernel. Questo processo calcola una media pesata dei pixel vicini, riducendo efficacemente il rumore ad alta frequenza e sfumando i bordi, facilitando la successiva fase di sogliatura e di estrazione dei contorni.

- **Operazioni Morfologiche (Erosione e Dilatazione):** Queste operazioni si basano sulla teoria della morfologia matematica e sono fondamentali per la pulizia delle immagini binarie (come la maschera ottenuta dalla sogliatura). Involgono un "elemento strutturante" (un kernel di forma definita, es. un quadrato o un cerchio) che viene fatto scorrere sull'immagine. Le operazioni di erosione e dilatazione sono formalizzate nella morfologia matematica [ [25], [18]].
- **Erosione:** Riduce le aree bianche (oggetti) e ingrandisce i "buchi" neri. Un pixel nell'immagine di output sarà bianco solo se 'tutti' i pixel nell'immagine di input che si sovrappongono all'elemento strutturante sono bianchi. Questo è analogo a un'operazione di MIN logico nel vicinato, dove il risultato è 1 solo se tutti gli input sono 1.
- **Dilatazione:** Ingrandisce le aree bianche e riempie i buchi neri. Un pixel nell'immagine di output sarà bianco se 'almeno un' pixel nell'immagine di input che si sovrappone all'elemento strutturante è bianco. Questo è analogo a un'operazione di MAX logico, dove il risultato è 1 se almeno un input è 1.

Combinando erosione e dilatazione (es. "Opening" = Erosione seguita da Dilatazione; "Closing" = Dilatazione seguita da Erosione), si possono efficacemente rimuovere piccole macchie di rumore isolate, separare oggetti che sono quasi uniti e chiudere piccole interruzioni all'interno delle aree di interesse.

### 5.2.3 Estrazione Contorni e Analisi Geometrica

- **Estrazione Contorni (`cv2.findContours`):** Un contorno è una curva che unisce tutti i punti continui lungo il confine di regioni con la stessa proprietà (es. colore o intensità). Algoritmi efficienti, come "Suzuki and Abe" o variazioni di "Marching Squares", vengono utilizzati per identificare queste sequenze di

pixel di confine in un'immagine binaria [19], [27]. L'output di questa funzione è una lista di array NumPy, dove ogni array contiene le coordinate dei punti che formano un contorno specifico. La scelta del contorno di interesse (es. il più grande) è un'euristica basata sulla presupposizione che l'oggetto tracciato sia il "blob" più significativo.

- **Cerchio Minimo di Inclusione (`cv2.minEnclosingCircle`):** Una volta identificato il contorno principale (es. quello con l'area maggiore, calcolata tramite `cv2.contourArea`), la funzione `cv2.minEnclosingCircle` calcola il cerchio di raggio minimo che include tutti i punti di quel contorno. L'algoritmo sottostante risolve un problema di geometria computazionale, spesso basato su varianti dell'algoritmo di Welzl o su algoritmi iterativi che cercano di convergere al cerchio minimo. Il risultato è una coppia di coordinate  $(x, y)$  per il centro del cerchio e un valore per il raggio, che offrono una stima robusta e standardizzata della posizione e della dimensione apparente dell'oggetto tracciato, indipendentemente dalla sua forma irregolare.

## 5.3 Approccio 2: Tracking Cognitivo tramite Inferenza Deep Learning

L'approccio basato su deep learning rappresenta un salto concettuale significativo rispetto alla computer vision classica. Invece di definire regole e trasformazioni analitiche passo-passo, il sistema "impara" a riconoscere i pattern complessi che definiscono gli oggetti direttamente da un vasto dataset di esempi, utilizzando architetture di rete neurale.

### 5.3.1 Scelta del Modello: EfficientDet-Lite0

Per questo progetto è stato scelto il modello **EfficientDet-Lite0**, appartenente alla famiglia di architetture EfficientDet. Questa famiglia è stata specificamente progettata per un'alta efficienza (cioè, un buon equilibrio tra accuratezza e velocità di inferenza) su dispositivi con risorse computazionali limitate (edge devices). Come osservato in studi che hanno impiegato versioni simili di questa architettura per applicazioni su smartphone, come la navigazione assistita per ipovedenti [8], la priorità è stata data alla scalabilità e all'ottimizzazione per l'esecuzione su dispositivi con memoria e potenza di calcolo limitate. La scelta è motivata da:

- **Efficienza intrinseca:** EfficientDet si basa su tecniche avanzate come le "separable convolutions" (che riducono drasticamente il numero di parametri e operazioni rispetto alle convoluzioni standard) e un'architettura di ricerca neurale (NAS) per trovare la configurazione ottimale della rete. Questo permette di ottenere un'accuratezza elevata con un numero di parametri (e quindi calcoli, o FLOPs) significativamente inferiore rispetto a modelli di rilevamento oggetti più vecchi o più grandi (es. MobileNet-SSD o modelli ResNet-based).
- **Scalabilità Composta (Compound Scaling):** Una delle innovazioni chiave di EfficientDet è il "Compound Scaling", che scala in modo uniforme e bilanciato tutti i parametri della rete (risoluzione dell'input, profondità della rete e larghezza dei canali) utilizzando un unico coefficiente composto. Le versioni "Lite" (Lite0, Lite1, Lite2, etc.) sono varianti ottimizzate per il mobile/edge, con Lite0 che rappresenta la versione più leggera e veloce, ideale per il Raspberry Pi 3.
- **Rete di Feature Bidirezionale (BiFPN):** EfficientDet introduce il BiFPN (Bi-directional Feature Pyramid Network) come modulo chiave per l'aggregazione multi-scala delle feature. Le reti di rilevamento oggetti beneficiano dell'elaborazione di feature a diverse risoluzioni per rilevare oggetti di diverse dimensioni. BiFPN migliora le tradizionali Feature Pyramid Networks (FPN) consentendo un flusso di informazione bidirezionale (dall'alto verso il basso e dal basso verso l'alto) e una fusione pesata delle feature, il che porta a una migliore accuratezza con meno risorse.
- **Modello Pre-addestrato su COCO:** Il modello utilizzato è pre-addestrato sul dataset COCO (Common Objects in Context), uno dei più grandi e diversi dataset per il rilevamento di oggetti, che contiene 80/91, poiché nei ultimi anni ci sono stati alcuni aggiornamenti nella classi di oggetti comuni. Questo lo rende immediatamente utilizzabile per il rilevamento di target generici come "person" (persona), "cup" (tazza), "laptop", "cell phone" (telefono cellulare) ecc., senza la necessità di un addestramento personalizzato per il caso d'uso specifico. [28]

### 5.3.2 Pipeline di Elaborazione con TFLite Support

Per interagire con il modello EfficientDet-Lite0 sul Raspberry Pi, si è utilizzata la libreria **TFLite Support**, che fa parte dell'ecosistema TensorFlow Lite. TFLite

Support è un wrapper di alto livello che astrae gran parte della complessità della gestione manuale dei tensori e dell'interprete TensorFlow Lite, rendendo l'integrazione del modello nel codice Python più semplice.

## Architettura e Meccanismi di Inferenza delle CNN

Una Rete Neurale Convoluzionale (CNN) è un tipo di rete neurale profonda particolarmente efficace per l'analisi di immagini. La sua architettura è ispirata al sistema visivo degli animali e si compone di strati specializzati:

- **Strati Convoluzionali (Convolutional Layers):** Sono il cuore della CNN. Ogni strato convoluzionale applica un insieme di "filtri" (o kernel) all'immagine di input. Un filtro è una piccola matrice di pesi che scorre sull'immagine, eseguendo un prodotto scalare tra i suoi pesi e i pixel sottostanti. Il risultato è una "feature map" che evidenzia specifici pattern (es. bordi, angoli, texture). Matematicamente, l'output  $O$  di un'operazione di convoluzione per un pixel  $(x, y)$  e un filtro  $F$  su un input  $I$  è:

$$O_{x,y} = \sum_{u,v} I_{x+u,y+v} \cdot F_{u,v}$$

La formula utilizzata si ricava implicitamente dalla convoluzione, come descritto in [25]. Le CNN eseguono convoluzioni tra immagine e kernel, come descritto in [29], [30]. Questi strati apprendono gerarchie di feature: i primi strati estraggono feature di basso livello (es. bordi orizzontali/verticali), mentre gli strati più profondi combinano queste feature per riconoscere pattern più complessi (es. parti di un volto, ruote di un veicolo).

- **Strati di Attivazione (Activation Layers):** Dopo ogni operazione convoluzionale, viene solitamente applicata una funzione di attivazione non lineare (comunemente ReLU - Rectified Linear Unit). Questa non linearità è cruciale per permettere alla rete di modellare relazioni complesse e non lineari nei dati, che altrimenti non sarebbero possibili con sole operazioni lineari.
- **Strati di Pooling (Pooling Layers):** Questi strati riducono le dimensioni spaziali delle feature map (downsampling), riducendo la quantità di calcolo necessaria negli strati successivi e rendendo la rete più robusta a piccole variazioni di posizione o scala degli oggetti (translation invariance). Il Max Pooling, ad esempio, seleziona il valore massimo all'interno di una piccola finestra.

- **Strati di Rilevamento (Detection Heads):** Nelle reti di rilevamento oggetti come EfficientDet, gli strati finali (spesso chiamati "Heads") sono progettati per produrre due tipi di output per ogni oggetto rilevato:
  - **Regressione del Bounding Box:** Prevedono le coordinate del riquadro di contorno (bounding box) che racchiude l'oggetto. Questo è un problema di regressione, dove il modello impara a mappare le feature visive a valori numerici (es. le coordinate  $(x_{min}, y_{min}, x_{max}, y_{max})$  o il centro e le dimensioni).
  - **Classificazione dell'Oggetto:** Prevedono la probabilità che l'oggetto all'interno del bounding box appartenga a ciascuna delle classi pre-addestrate (es. "persona", "cellulare"). Questo è un problema di classificazione, dove spesso una funzione softmax converte i punteggi raw in probabilità che sommano a 1.

L'**inferenza** è il processo di far passare un nuovo input (un frame video) attraverso tutti questi strati della rete pre-addestrata per ottenere le previsioni di bounding box e classe in output.

## Quantizzazione e Ottimizzazione per Edge

Un aspetto fondamentale dell'efficienza di EfficientDet-Lite0 e della sua esecuzione su piattaforme embedded come il Raspberry Pi è la **quantizzazione**. Tradizionalmente, i modelli di deep learning sono addestrati con pesi e attivazioni rappresentati da numeri in virgola mobile a 32 bit (`float32`) [21]. Questa precisione è elevata, ma richiede molta memoria e potenza di calcolo.

La quantizzazione è un processo che converte i pesi e le attivazioni del modello a minore precisione, tipicamente interi a 8 bit (`int8`).

- **Principio Analitico:** La quantizzazione non è una semplice troncazione, ma una mappatura del range dinamico dei valori in virgola mobile a un range di interi. Questo viene fatto definendo un fattore di scala (`scale`) e un punto zero (`zero_point`) per ogni tensore (array di dati) all'interno della rete:

$$\text{int8\_value} = \text{round} \left( \frac{\text{float32\_value}}{\text{scale}} + \text{zero\_point} \right)$$

Durante l'inferenza, le operazioni (moltiplicazioni, somme) vengono eseguite utilizzando questi interi a 8 bit, e poi il risultato può essere de-quantizzato per ottenere un valore float.

- **Benefici per l’Edge Computing:**

- **Riduzione delle Dimensioni del Modello:** Un modello `int8` occupa 4 volte [21] meno spazio in memoria (es. da 100 MB a 25 MB) rispetto alla sua controparte `float32`. Questo è cruciale per dispositivi con RAM limitata come il Raspberry Pi.
- **Minore Consumo Energetico:** Le operazioni su interi richiedono significativamente meno energia rispetto a quelle in virgola mobile, estendendo l’autonomia della batteria per i sistemi mobili.
- **Maggiore Velocità di Inferenza:** Le CPU a basso consumo e le unità di calcolo dedicate all’IA (come le Google Coral TPU o le NPU integrate) [31] sono ottimizzate per eseguire calcoli su interi a 8 bit molto più velocemente rispetto ai calcoli in virgola mobile. Questo si traduce direttamente in un throughput (FPS) più elevato e una latenza inferiore.

Sebbene la quantizzazione causa una minima perdita di precisione nel modello, per architetture come EfficientDet-Lite0, è studiata per minimizzare l’impatto sull’accuratezza finale, rendendola una tecnica essenziale per l’implementazione efficiente di AI su dispositivi embedded. La libreria TensorFlow Lite Runtime è specificamente progettata per eseguire questi modelli quantizzati in modo ottimale.

# Capitolo 6

## Integrazione e Metodologia di Test

La fase di integrazione e test è un momento cruciale in qualsiasi progetto di ingegneria robotica, in cui i sottosistemi hardware e software, sviluppati e testati individualmente, vengono assemblati per la prima volta per operare come un'unica entità coesa. Questa sezione documenta il processo di integrazione seguito e definisce i protocolli di test sperimentali progettati per valutare le performance del sistema in modo oggettivo e ripetibile.

### 6.1 Integrazione dei Sottosistemi Hardware e Software

L'integrazione del prototipo ha seguito un approccio incrementale e bottom-up, volto a minimizzare la complessità e facilitare il debug.

- 1. Test del Livello di Attuazione:** Inizialmente, il sottosistema Arduino-L298N-Motori è stato testato in isolamento. Utilizzando l'IDE Arduino e il suo Serial Monitor, sono stati inviati manualmente i comandi testuali (`avanti\n`, `sinistra\n`, etc.) per verificare la corretta risposta meccanica del robot. Questa fase ha permesso di calibrare la cinematica differenziale e di assicurare che ogni comando logico producesse il movimento fisico atteso.
- 2. Test della Comunicazione Edge-Actuator:** Successivamente, è stato stabilito il collegamento seriale tra il Raspberry Pi e l'Arduino. Un semplice script Python è stato utilizzato per inviare la sequenza di comandi e verificare che l'Arduino li ricevesse e li eseguisse correttamente, validando l'intero protocollo di controllo.

3. **Test del Sottosistema di Sicurezza:** Il sensore a ultrasuoni è stato integrato e testato. Avvicinando un ostacolo al robot, si è verificato che il firmware Arduino entrasse correttamente nello stato di override, arrestando i motori e inviando il messaggio `OBJ_VICINO` al Raspberry Pi, indipendentemente dai comandi inviati da quest'ultimo.
4. **Integrazione Completa:** Solo al superamento dei test precedenti, è stato eseguito per la prima volta lo script Python completo, integrando la pipeline di visione con la logica di controllo. Questa metodologia ha permesso di isolare e risolvere i problemi a livello di singolo sottosistema prima di affrontare la complessità del sistema integrato.

## 6.2 Metodologia di Test Sperimentale

Per condurre un'analisi comparativa rigorosa tra le due modalità di visione, sono stati definiti protocolli di test specifici. Gli esperimenti sono stati condotti in un ambiente interno standard (una stanza), sotto illuminazione artificiale (luce accesa). Le condizioni sono state mantenute il più possibile uniformi per le comparazioni tra le due modalità.

### 6.2.1 Test di Performance Quantitativa (Throughput e Latenza)

L'obiettivo di questo test è misurare la velocità di elaborazione del sistema in ciascuna modalità.

- **Metrica:** La metrica principale è il throughput, misurato in **Fotogrammi al Secondo (FPS)**. Un valore di FPS più alto indica una latenza inferiore per ogni ciclo percezione-azione.
- **Procedura:** Lo script Python è stato instrumentato per calcolare gli FPS. Un contatore di frame viene incrementato ad ogni ciclo del loop principale. Ogni secondo, il numero di frame elaborati viene diviso per il tempo trascorso, ottenendo il valore istantaneo di FPS. Per ottenere un dato stabile, sono state eseguite sessioni di test per ciascuna modalità, registrando i valori medi di FPS e la loro deviazione standard.

### 6.2.2 Test dei Target e Osservazione del Comportamento

Durante i test di performance, è stato anche osservato il comportamento del robot in relazione ai diversi target per valutare la sua capacità di tracciamento e la robustezza dei due approcci:

- **Modalità Cromatica:** Il robot è stato testato tracciando una **pallina gialla**. Sono state osservate la sua reattività al colore e la sua sensibilità a variazioni di illuminazione o all'introduzione di altri oggetti simili.
- **Modalità Machine Learning:** Il robot è stato testato tracciando una **persona** e un **telefono cellulare**. Sono state osservate la capacità del modello di discriminare semanticamente questi oggetti da altri elementi nella scena, la sua robustezza a piccole occlusioni e a variazioni di posa.

# Capitolo 7

## Risultati Sperimentali e Discussione

In questo capitolo vengono presentati e analizzati i dati raccolti durante la fase di test. I risultati sono suddivisi in analisi quantitativa e qualitativa, seguita da una discussione critica che interpreta i dati nel contesto degli obiettivi del progetto.

### 7.1 Risultati Quantitativi: Analisi del Throughput e della Latenza

Le misurazioni delle performance hanno fornito dati chiari e numericamente significativi sul costo computazionale delle due strategie di visione. I risultati medi sono riassunti in Tabella 7.1.

Tabella 7.1: Risultati comparativi delle performance quantitative tra le due modalità di tracking.

Metrica	Modalità <code>color</code>	Modalità <code>ml</code>
FPS Medio	10.0 FPS	~2-3 FPS
Deviazione Standard FPS	± 0.5 FPS	± 1.2 FPS
Latenza Media per Frame	~100 ms	~333-500 ms

La modalità di **tracking cromatico** ha operato con un throughput medio di **10.0 FPS**, un valore inferiore alla frequenza massima di cattura della web-cam (30 FPS), indicando che l'elaborazione, seppur leggera, introduce una latenza percepibile. Il tempo di elaborazione per frame è di circa 100 millisecondi.

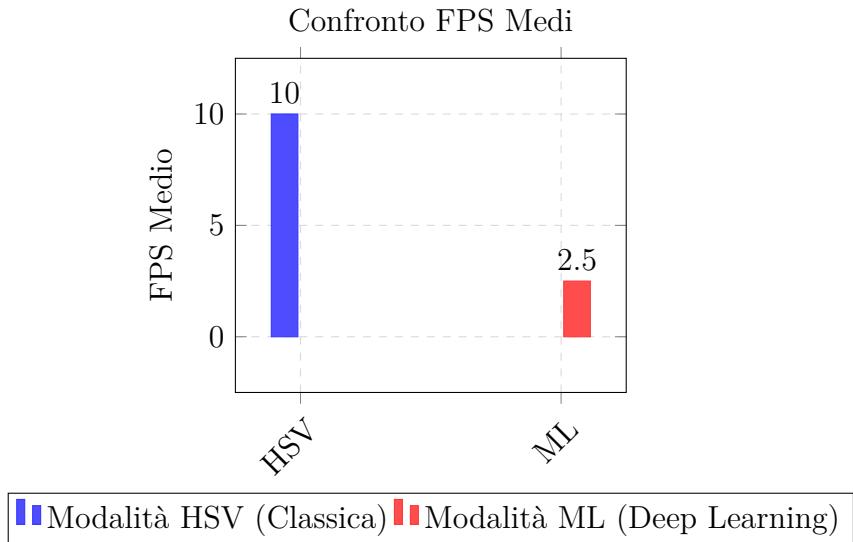


Figura 7.1: Confronto degli FPS medi tra le modalità di tracciamento HSV e ML.. Dati dalla tabella 7.1

Al contrario, la modalità **Machine Learning**, con il modello EfficientDet-Lite0 e 4 thread della CPU, ha registrato un throughput medio di **circa 2-3 FPS**. Questo crollo significativo nelle performance è interamente attribuibile al tempo richiesto per l'inferenza della rete neurale. La latenza media, che varia tra 333 e 500 millisecondi per completare un singolo ciclo di percezione-azione, è notevolmente alta e impatta sulla fluidità del controllo, rendendo il sistema inadatto a target in rapido movimento o ad applicazioni che richiedono risposte in tempo reale.

## 7.2 Risultati Qualitativi: Analisi del Comportamento di Tracking

L'osservazione del comportamento del robot durante i test ha evidenziato le differenze intrinseche tra le due strategie, fornendo un giudizio qualitativo sulla loro robustezza e intelligenza.

### 7.2.1 Performance del Tracking Cromatico (con pallina gialla)

Il tracker cromatico si è dimostrato reattivo e veloce nel rilevamento della pallina gialla, ma estremamente fragile e limitato in scenari variabili:

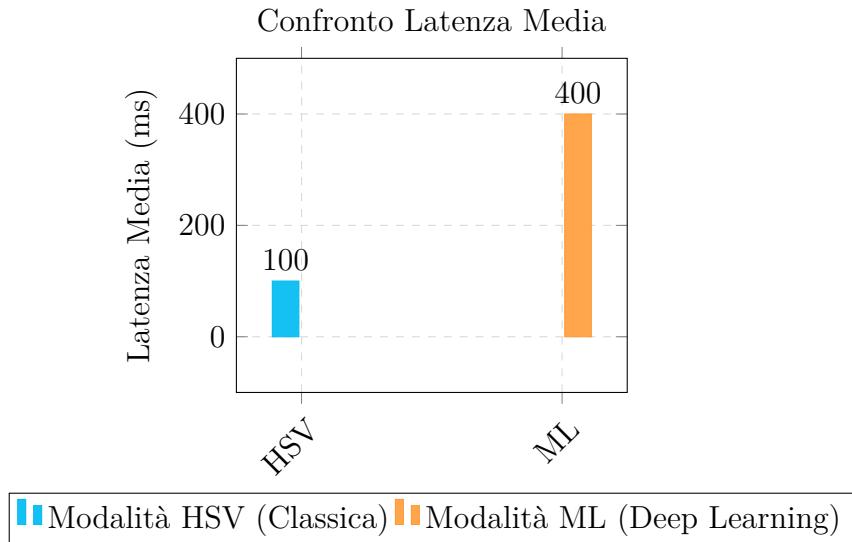


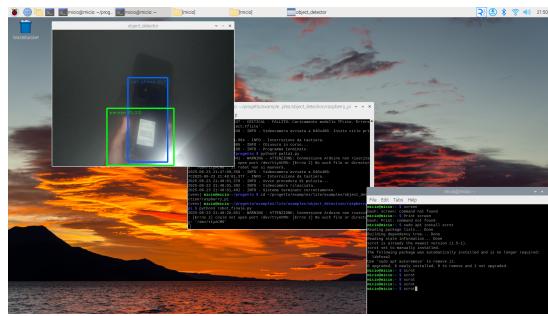
Figura 7.2: Confronto della latenza media tra le modalità di tracciamento HSV e ML. Dati dalla Tabella 7.1.

- **Sensibilità al Colore:** Il sistema tracciava efficacemente la pallina gialla in condizioni di illuminazione stabili. Tuttavia, la sua performance è stata fortemente compromessa da variazioni di luminosità o dall'introduzione di oggetti di colore simile.
- **Mancanza di Comprensione Semantica:** Essendo basato unicamente sul colore, il sistema non aveva alcuna "comprensione" della pallina come oggetto. Ciò significa che qualsiasi altro oggetto o superficie gialla nel campo visivo poteva agire da "distrattore", causando il passaggio del tracciamento a un nuovo bersaglio o la perdita del tracking se la pallina veniva parzialmente oscurata o appariva troppo piccola. Per esempio il sistema rilevava, che la maglia gialla doveva essere seguita e in quel istante perdeva controllo della situazione, questo accade poiché una volta agganciata la maglia, la pallina veniva persa dal campo visivo causando alla perdita del target, questo illustra perfettamente la mancanza di comprensione in scenari reali o dinamici, infatti come sottolineato questo è il punto di gran differenza tra i due modi di tracking.

### 7.2.2 Performance del Tracking con Machine Learning (con persona e cellulare)

Il tracker ML, pur con un throughput inferiore, ha mostrato una robustezza e una capacità di discriminazione semantica notevolmente superiori, rendendolo più adatto a scenari del mondo reale:

- **Selettività Semantica:** Il modello EfficientDet-Lite0 ha dimostrato una chiara capacità di distinguere tra classi di oggetti. Ha tracciato con successo le persone e i telefoni cellulari, ignorando oggetti di altri colori o forme non pertinenti. Questo è un vantaggio cruciale in ambienti complessi dove la distinzione del target non può basarsi solo su caratteristiche semplici come il colore. La Figura 7.1(a) e (b) dimostrano la capacità di discriminare tra diverse classi di oggetti.
- **Robustezza all’Illuminazione e all’Occlusione:** Il modello ha mantenuto un rilevamento più stabile anche con una significativa riduzione della luminosità o con occlusioni parziali (ad esempio, una persona parzialmente coperta). Le feature apprese dal deep learning sono meno dipendenti dalle condizioni di luce assolute e possono inferire la presenza di un oggetto anche da parti di esso.



(b) Rilevamento di un oggetto `cell phone`.

Figura 7.3: Esempi di rilevamento in modalità Machine Learning. Il sistema dimostra la capacità di riconoscere e classificare correttamente oggetti di diverse categorie semantiche.

### 7.3 Discussione Critica dei Risultati

I risultati ottenuti quantificano in modo inequivocabile il trade-off fondamentale dell’Edge AI. Non esiste un approccio "migliore" in assoluto, ma un approccio "ottimale", poiché dipendente principalmente dal caso d’uso e dai vincoli del sistema. Per un’applicazione industriale in ambiente controllato, dove un robot deve seguire un marcatore di un colore specifico su una linea di produzione, l’approccio cromatico potrebbe essere preferibile per la sua semplicità, velocità e il basso impatto sulle risorse. Tuttavia, la sua fragilità in condizioni variabili ne limita l’applicabilità.

Invece per un'applicazione più complessa e dinamica, come la sorveglianza autonoma o l'interazione uomo-macchina, dove il sistema deve identificare un target non cooperativo o in un ambiente non strutturato, il costo computazionale del machine learning diventa un compromesso necessario e giustificato per ottenere l'affidabilità e l'intelligenza richieste, nonostante la latenza più elevata e le difficoltà computazionali. Questo studio ha messo in evidenza la necessità di trovare un compromesso tra le esigenze di reattività con la capacità di comprensione del sistema.

# Capitolo 8

## Conclusioni e Sviluppi Futuri

### 8.1 Sintesi del Lavoro e Conclusioni Finali

In conclusione, questo lavoro di tesi ha portato alla progettazione, implementazione e validazione di un sistema robotico autonomo capace di eseguire il tracking di oggetti tramite due diverse metodologie di visione. Si è dimostrata l'efficacia di un'architettura distribuita per disaccoppiare il controllo real-time dall'elaborazione complessa. L'analisi comparativa ha quantificato il trade-off tra l'efficienza degli algoritmi di computer vision classica e l'intelligenza dei modelli di deep learning su una piattaforma a risorse limitate, confermando che la scelta tecnologica deve essere guidata da un'analisi critica dei requisiti specifici dell'applicazione. Il progetto ha raggiunto con successo i suoi obiettivi, fornendo una piattaforma funzionale e un'analisi sperimentale chiara dei compromessi dell'IA su dispositivi edge.

### 8.2 Proposte per Sviluppi Futuri

Le limitazioni emerse durante la sperimentazione aprono la strada a interessanti sviluppi futuri che potrebbero migliorare significativamente le capacità del sistema.

- **Integrazione Funzionale del Braccio Manipolatore:** Lo sviluppo futuro più diretto e significativo consiste nell'integrare a livello software il braccio robotico a 4 GDL, già presente fisicamente sul prototipo. Il sistema di visione potrebbe essere esteso non solo per tracciare un oggetto, ma anche per calcolarne la posa 3D (sfruttando tecniche di "perspective-n-point" se le dimensioni dell'oggetto sono note) e guidare il braccio per eseguire task di *pick-and-place* [1], trasformando il robot da un semplice inseguitore a un agente di manipolazione mobile.

- **Controllo Ibrido Dinamico (Tracking-by-Detection):** Per ottimizzare le risorse, si potrebbe implementare una logica che combini i due approcci: utilizzare il tracker cromatico, veloce, per un "aggancio" rapido e a basso costo del target, e attivare il modello ML, più lento, solo sull'area di interesse identificata per confermarne periodicamente la classe.
- **Accelerazione Hardware Dedicata:** Per superare il collo di bottiglia della CPU, uno sviluppo cruciale sarebbe l'integrazione di un co-processore dedicato all'IA, come un Google Coral TPU. Questo dispositivo, collegato via USB, si farebbe carico dell'inferenza, potendo aumentare drasticamente gli FPS della modalità ML.
- **Integrazione con Digital Twin:** Uno sviluppo metodologico avanzato consisterebbe nella creazione di un "gemello digitale" del robot in un ambiente di simulazione realistico come CoppeliaSim o ROS/Gazebo. Un Digital Twin permetterebbe di testare e validare nuovi algoritmi di controllo e di visione in modo rapido, sicuro e ripetibile, potendo simulare migliaia di scenari senza usurare l'hardware fisico. [32]
- **Integrazione Cloud-Based per il Monitoraggio e il Controllo Remoto del Robot tramite Framework Web:** Un ulteriore sviluppo riguarda l'integrazione del sistema con un ambiente cloud-based , utilizzando framework come Flask o FastAPI per esporre i dati raccolti dal robot e permettere il controllo remoto tramite interfaccia web [16]. Questo consentirebbe non solo di monitorare lo stato del robot da remoto, ma anche di effettuare analisi centralizzate dei dati raccolti, ad esempio per il logging storico o la generazione di report automatici.

# Bibliografia

- [1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. London: Springer, 2nd ed., 2010.
- [2] K. Ashton, “That ‘internet of things’ thing,” *RFID Journal*, vol. 22, no. 7, pp. 97–114, 2009. Origin of the term "Internet of Things".
- [3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ: Prentice Hall, 3rd ed., 2010.
- [4] J. Lee, H. Bagheri, and H.-A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems,” in *Manufacturing Letters*, vol. 3, pp. 18–21, Elsevier, 2015.
- [5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [6] Wikipedia contributors, “Raspberry pi.” [https://it.wikipedia.org/wiki/Raspberry\\_Pi](https://it.wikipedia.org/wiki/Raspberry_Pi), 2024. Accessed on: 10 luglio 2025.
- [7] Wikipedia contributors, “Arduino (hardware).” [https://it.wikipedia.org/wiki/Arduino\\_\(hardware\)](https://it.wikipedia.org/wiki/Arduino_(hardware)), 2024. Accessed on: 10 luglio 2025.
- [8] B. Kuriakose, R. Shrestha, and F. E. Sandnes, “SceneRecog: A Deep Learning Scene Recognition Model for Assisting Blind and Visually Impaired Navigate using Smartphones,” in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2021.
- [9] S. Aditya and V. K. S. G., “Raspberry pi and arduino: A technical comparison for building iot projects,” *International Journal of Computer Applications*, vol. 180, no. 33, pp. 29–32, 2018.
- [10] W. J. Johnson Jr., “Design, implementation, and control of an autonomous robot,” master’s thesis, California State University, San Bernardino, 2018. Accessed on: 10 luglio 2025.

- [11] G. A. Al-Suhail and A. Al-Naji, “Android-arduino based mobile robot for object detection and tracking,” in *2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT)*, pp. 233–238, 2017.
- [12] V. Kazemi, S. K. Singh, and A. L. Das, “Learning image processing with opencv,” 2015.
- [13] Google, “Tensorflow.” <https://www.tensorflow.org/?hl=it>, 2024. Accessed on: 10 luglio 2025.
- [14] S. Jain, V. M. Manikandan, and P. K. D. V. Yarlagadda, “Efficient machine learning models for resource-constrained devices: A survey,” *Annual Reviews in Control*, vol. 52, pp. 416–431, 2021.
- [15] P. Warden *et al.*, “Tensorflow lite: A lightweight solution for on-device inference,” in *O'Reilly AI Conference*, 2017.
- [16] G. Van Rossum and F. L. Drake Jr., *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.
- [17] MakersLab, “Sensore di distanza ad ultrasuoni hc sr04 con arduino.” <https://www.makerslab.it/sensore-di-distanza-ad-ultrasuoni-hc-sr04-con-arduino/>, 2024. Accessed on: 10 luglio 2025.
- [18] The OpenCV Foundation, “Opencv 4.x documentation.” <https://docs.opencv.org/4.x/>, 2024. Accessed on: 10 luglio 2025.
- [19] The OpenCV Foundation, “Image filtering.” [https://docs.opencv.org/4.x/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html), 2024. Accessed on: 10 luglio 2025.
- [20] Google, “Efficientdet-lite0 detection model for tensorflow lite.” <https://tfhub.dev/tensorflow/lite-model/efficientdet-lite0/detection/default/1>, 2024. Accessed on: 10 luglio 2025.
- [21] TensorFlow Developers, “Post-training quantization.” [https://www.tensorflow.org/lite/performance/post\\_training\\_quantization](https://www.tensorflow.org/lite/performance/post_training_quantization), 2024. Accessed on: 10 luglio 2025.
- [22] Wikipedia contributors, “Raspberry pi.” [https://it.wikipedia.org/wiki/Raspberry\\_Pi](https://it.wikipedia.org/wiki/Raspberry_Pi), 2024. Accessed on: 10 luglio 2025.

- [23] Python Software Foundation, “Python 3 documentation.” <https://docs.python.org/3/>, 2024. Accessed on: 10 luglio 2025.
- [24] YouTube Video, “Hsv color space explained - opencv tutorial,” 2023. Accessed: Feb. 15, 2025.
- [25] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, NJ: Pearson, 4th ed., 2018.
- [26] Wikipedia contributors, “Gaussian blur.” [https://en.wikipedia.org/wiki/Gaussian\\_blur](https://en.wikipedia.org/wiki/Gaussian_blur), 2024. Accessed on: 10 luglio 2025.
- [27] The OpenCV Foundation, “Opencv 4.x documentation - quantization support (experimental).” <https://docs.opencv.org/4.x/>, 2024. Accessed on: 10 luglio 2025.
- [28] T.-Y. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” *CoRR*, vol. abs/1405.0312, 2014.
- [29] F.-F. Li, J. Johnson, and S. Yeung, “Cs231n: Convolutional neural networks for visual recognition.” <https://cs231n.github.io/convolutional-networks/>, 2024. Accessed on: 10 luglio 2025.
- [30] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [31] Google, “The benefits of on-device ml with coral.” <https://coral.ai/docs/edgetpu/benefits/>, 2024. Accessed on: 10 luglio 2025.
- [32] A. Mazumder, M. Sahed, Z. Tasneem, P. Das, F. Badal, M. Ali, M. Ahamed, S. Abhi, S. Sarker, S. Das, M. Hasan, M. Islam, and M. Islam, “Towards next generation digital twin in robotics: Trends, scopes, challenges, and future,” *Heliyon*, vol. 9, 2023.