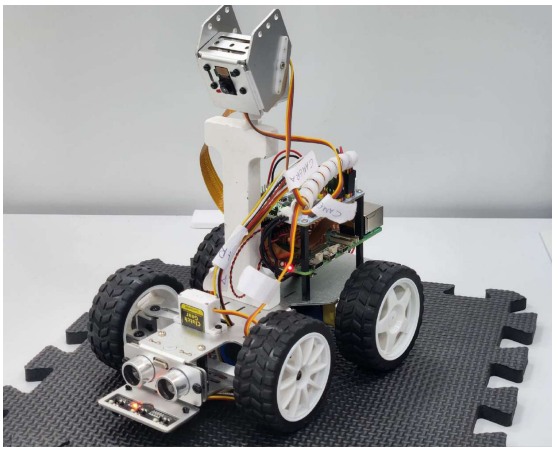


PEDESTRIAN AND VEHICLE ORIENTATION DETECTION ON PICAR-X USING YOLOV11S

ANDREA SALURSO AND FRANCESCA AVALLONE

Abstract. In questo lavoro abbiamo progettato e implementato un sistema efficiente e funzionale per il riconoscimento di pedoni e l'orientamento dei veicoli su piattaforma Raspberry Pi, integrato con il robot Picar-X. Il sistema è in grado di rilevare pedoni, distinguere tra fronte, retro e lato dei veicoli e di attivare le azioni appropriate in base al contesto stradale. Per l'implementazione finale è stata utilizzata la rete YOLOv11s, scelta in quanto la più performante tra le versioni testate. I dataset utilizzati per l'addestramento sono stati creati interamente da noi, annotati tramite lo strumento Label-Studio, garantendo così un'accurata e specifica rappresentazione delle classi di interesse. Il risultato è un sistema rapido e preciso, adatto ad applicazioni real-time su dispositivi a risorse limitate come il Raspberry Pi.

Date: June 27, 2025.

Contents

1. Introduzione

L'obiettivo principale del nostro team è stato la progettazione e realizzazione di un sistema per il riconoscimento di pedoni e l'orientamento dei veicoli, finalizzato al controllo autonomo della Picar-X. Il sistema, progettato per operare su piattaforma Raspberry Pi, è in grado di identificare oggetti rilevanti tramite la videocamera integrata e, in base alla loro natura (pedone, fronte, retro e lato di un'auto), di eseguire specifiche azioni di risposta. La distanza dagli oggetti viene invece stimata grazie ai sensori ad ultrasuoni montati sul veicolo.

La fase iniziale del progetto si è concentrata sulla scelta dei dataset da utilizzare per l'addestramento dei modelli. Dopo un'attenta analisi di quelli presenti in letteratura, abbiamo riscontrato che nessuno di essi fosse adeguato al nostro caso d'uso specifico, in particolare per quanto riguarda la rilevazione di piccoli pedoni in plastica, impiegati per la simulazione su pista. Inoltre, è emersa una completa assenza di dataset o modelli preaddestrati in grado di distinguere tra lato destro e lato sinistro di un veicolo, un'informazione fondamentale per il corretto orientamento della Picar-X nel contesto operativo previsto. Di conseguenza, abbiamo deciso di costruire da zero dei dataset ad hoc, raccogliendo immagini sia da fonti online, sia attraverso sessioni fotografiche effettuate in laboratorio utilizzando la videocamera della Picar-X. La fase di annotazione delle immagini è stata svolta tramite lo strumento Label-Studio, che ci ha permesso di creare un set di dati coerente e accuratamente etichettato, includendo specifiche classi per il riconoscimento del fronte, del retro, dei due lati del veicolo e dei pedoni.

Una delle sfide principali è stata la scelta del modello più adatto al contesto di deployment su Raspberry Pi, un ambiente caratterizzato da risorse computazionali limitate. Inizialmente abbiamo testato YOLOv8, ma le sue prestazioni in termini di frame rate si sono rivelate insoddisfacenti. Successivamente abbiamo valutato modelli basati su TensorFlow Lite, che, nonostante la leggerezza, non hanno raggiunto risultati accettabili né in termini di velocità né di precisione. Il giusto compromesso è stato trovato con YOLOv11s (versione "small"), che si è dimostrato più rapido nei tempi di inferenza. Tuttavia, per ottenere prestazioni realmente adatte all'ambiente embedded, è stato necessario convertire il modello nel formato ONNX, ottimizzandolo per l'inferenza su dispositivi a bassa potenza come Raspberry Pi.

In sintesi, il progetto ha richiesto un equilibrio costante tra precisione, reattività e leggerezza computazionale. Trovare una soluzione efficace per l'elaborazione in tempo reale su hardware limitato ha rappresentato una delle sfide più significative, ma anche uno degli aspetti più stimolanti del lavoro svolto.

2. Stato dell'Arte

L'elaborazione delle immagini in tempo reale su dispositivi embedded a risorse limitate, come il Raspberry Pi, rappresenta una sfida tecnica particolarmente rilevante nell'ambito dei sistemi autonomi. L'interesse verso queste soluzioni è cresciuto negli ultimi anni grazie alla disponibilità di hardware economico e alla crescente ottimizzazione di modelli di deep learning leggeri. In questo contesto, la scelta di reti neurali compatte e ottimizzate rappresenta una condizione fondamentale per ottenere un sistema affidabile, reattivo e adatto all'elaborazione in tempo reale.
Modelli troppo complessi, infatti, pur offrendo alta accuratezza, risultano spesso inutilizzabili su piattaforme a bassa potenza come Raspberry.

2.1. **Architetture per Object Detection in Contesti Embedded.**

L'elaborazione di immagini in tempo reale su dispositivi embedded richiede l'impiego di modelli di deep learning ottimizzati per garantire un buon compromesso tra accuratezza e velocità di inferenza.

2.1.1. *YOLO.*

Tra le architetture più diffuse in questo ambito si annovera la famiglia di modelli **YOLO** (You Only Look Once), nota per l'elevata efficienza nell'object detection. Le versioni più recenti, come **YOLOv8** e **YOLOv11**, includono ottimizzazioni architetturali che migliorano la capacità di generalizzazione e la velocità, a scapito, in alcuni casi, dell'adattabilità su dispositivi con risorse computazionali ridotte.

- **YOLOv8** è progettato per offrire alta precisione grazie a una struttura profondamente ottimizzata. Tuttavia, la sua complessità architetturale lo rende più adatto a esecuzioni su GPU o ambienti edge avanzati.
- **YOLOv11n** e **YOLOv11s** rappresentano versioni più leggere della stessa famiglia, pensate per ambienti a bassa potenza. In particolare, YOLOv11s si distingue per la sua capacità di mantenere buone prestazioni in scenari real-time, pur con una ridotta complessità computazionale.
- Le versioni **YOLOv11m** e **YOLOv11l**, pur garantendo maggiore accuratezza, risultano meno adatte a contesti embedded a causa del maggiore impatto sulle risorse hardware.

2.1.2. *TENSORFLOW.*

In parallelo, per l'esecuzione su dispositivi embedded, trova ampio impiego anche **TensorFlow Lite**, una versione ottimizzata del framework TensorFlow, progettata per garantire inferenze rapide su CPU ARM. TFLite permette l'esecuzione di modelli convertiti in formato compatto, spesso utilizzati in combinazione con acceleratori hardware (come Coral o Edge TPU) per migliorare ulteriormente le prestazioni. Tuttavia, modelli troppo semplificati rischiano di sacrificare l'accuratezza, in particolare nel riconoscimento di oggetti piccoli o in condizioni ambientali complesse.

2.2. **Ottimizzazione dei Modelli.**

Per adattare modelli di deep learning a dispositivi embedded a risorse limitate, è fondamentale applicare tecniche di ottimizzazione che bilancino accuratezza e requisiti computazionali. Tra le metodologie più diffuse troviamo:

- **Quantizzazione**: consiste nella riduzione della precisione numerica dei pesi e delle attivazioni del modello, passando da valori a 32-bit floating point a formati più leggeri come 16-bit o 8-bit interi. Questa tecnica permette di ridurre la dimensione del modello e accelerare l'inferenza, spesso con un impatto minimo sull'accuratezza finale.
- **Conversione in formati ottimizzati** come **ONNX** (Open Neural Network Exchange) e **TensorFlow Lite** (TFLite): tali formati permettono di esportare e ottimizzare modelli per ambienti a bassa potenza, migliorandone l'efficienza computazionale. La conversione facilita l'uso di librerie e runtime specifici che sfruttano accelerazioni hardware e ottimizzazioni software.

L'adozione combinata di queste tecniche consente di ottenere modelli leggeri e veloci, in grado di eseguire inferenze in tempo reale anche su hardware con capacità computazionali ridotte come il Raspberry Pi.

2.3. **Preprocessing delle Immagini.** Nel contesto dell'elaborazione delle immagini su dispositivi embedded come il Raspberry Pi, le tecniche di preprocessing rivestono un ruolo cruciale per migliorare sia la qualità dei dati in input sia l'efficienza del modello. Tra le tecniche più comuni troviamo:

(1) **Ridimensionamento delle immagini**: consiste nell'adattare le dimensioni delle immagini di input alle specifiche richieste del modello di deep learning. Questa operazione permette di ridurre il carico computazionale e uniformare il formato dei dati, facilitando l'addestramento e l'inferenza.
(2) **Augmentation**: generazione di varianti artificiali delle immagini originali mediante trasformazioni come rotazioni, traslazioni, variazioni di luminosità e riflessioni. L'augmentation aumenta la quantità e la varietà dei dati di addestramento senza necessitare di nuove acquisizioni, migliorando la capacità del modello di generalizzare e riducendo il rischio di overfitting.

Queste tecniche sono particolarmente importanti quando si lavora con dataset limitati, come spesso avviene in applicazioni embedded.

2.4. **Strumenti Software e Hardware della Picar-X.** La **Picar-X**, sviluppato da *SunFounder*, rappresenta una piattaforma educativa avanzata, pensata per introdurre studenti e sviluppatori al mondo della robotica e dell'intelligenza artificiale basata su *Raspberry Pi* [2].

Dal punto di vista **hardware**, il Picar-X integra un insieme completo di componenti:

- **Telecamera frontale**: principale sensore per l'acquisizione visiva, utilizzata per l'elaborazione in tempo reale nei task di visione artificiale.
- **Sensori a ultrasuoni**: montati frontalmente, permettono il rilevamento della distanza da ostacoli e supportano funzionalità di evitamento.
- **Sensori a infrarossi**: utili per il line following e il riconoscimento di bordi e tracce sul terreno.
- **Servomotori**: controllano la direzione della telecamera e consentono il tracciamento dinamico di oggetti.

- **Motori DC con encoder**: gestiscono la locomozione del robot e permettono di rilevare la velocità e la distanza percorsa.
- **Batteria ricaricabile agli ioni di litio**: fornisce l'alimentazione al sistema; include protezioni integrate e indicatori LED di carica.
- **Modulo audio**: impiegato per la generazione di notifiche acustiche.
- **Modulo di comunicazione**: sfrutta la connettività Wi-Fi e Bluetooth del Raspberry Pi per abilitare il controllo remoto e il trasferimento dati.

Sul lato **software**, il robot è supportato da un ecosistema completo fornito da SunFounder:

- La libreria `SunFounder_PiCar-X` permette di interagire con tutti i dispositivi integrati, offrendo un'interfaccia ad alto livello per il controllo di motori, LED, servomotori e sensori.
- La libreria `Vilib` [3] fornisce strumenti per l'elaborazione delle immagini in tempo reale. Include funzionalità di face detection, color tracking e object tracking, ottimizzate per Raspberry Pi. Tuttavia, nonostante le buone prestazioni in termini di frame rate, nei nostri test ha mostrato una percentuale elevata di *false positives*, rendendola meno adatta a compiti complessi.
- Il sistema è compatibile con modelli in formato `TensorFlow Lite`, `ONNX` e framework come PyTorch, grazie alle ottimizzazioni software e all'uso di modelli leggeri adattati a dispositivi embedded.
- SunFounder mette anche a disposizione un'applicazione mobile (disponibile per Android e iOS) che consente il **controllo remoto** del robot. Tramite l'app è possibile accedere al feed video della telecamera, controllare la direzione e attivare modalità automatiche come l'inseguimento o l'evitamento ostacoli.

Complessivamente, il Picar-X costituisce una piattaforma ideale per la sperimentazione in ambito *robotico embedded*, fornendo un'integrazione completa tra hardware modulare e strumenti software ben documentati [1].

2.5. **Scelte progettuali.** La scelta del modello di riconoscimento più adatto per la Picar-X è stata effettuata tramite una serie di test comparativi tra diverse soluzioni basate su YOLO e TensorFlow. Tra i modelli valutati, YOLOv11s (versione "small") si è rivelato il migliore in termini di bilanciamento tra accuratezza e velocità di inferenza, requisito fondamentale per l'esecuzione su Raspberry Pi. Sono stati anche testati modelli sviluppati con TensorFlow e convertiti in TensorFlow Lite: sebbene questi modelli si distinguessero per la leggerezza e la rapidità di esecuzione, hanno mostrato performance inferiori nell'accuratezza del riconoscimento rispetto a YOLOv11s.

Parallelamente, abbiamo sperimentato le librerie Vilib, sviluppate da SunFounder per il rilevamento oggetti con la videocamera della Picar-X. Queste librerie si sono dimostrate molto efficienti in termini di velocità e integrazione con l'hardware, ma la precisione ottenuta non è stata soddisfacente a causa di un elevato numero di false dismissals, che compromettevano l'affidabilità del sistema. Anche in questo

scenario, YOLOv11s ha evidenziato una superiore robustezza e precisione.
Per massimizzare le prestazioni in ambiente embedded, abbiamo infine convertito il modello in formato ONNX, ottimizzandolo per ottenere un elevato frame rate e una maggiore efficienza computazionale, essenziali per il deployment su Raspberry Pi.

Per quanto riguarda il preprocessing, prima della fase di training abbiamo applicato una serie di tecniche di data augmentation volte ad aumentare la varietà e la quantità di dati a disposizione, migliorando così la capacità del modello di generalizzare. Contestualmente, le immagini sono state uniformemente ridimensionate per garantire coerenza degli input e ottimizzare le prestazioni complessive del sistema.

3. Sviluppo del Sistema

3.1. **Creazione del Dataset.** La prima fase dello sviluppo ha previsto un'attenta ricerca di dataset già annotati che potessero soddisfare le esigenze del progetto. Tuttavia, non sono stati trovati dataset sufficientemente ricchi, in particolare per quanto riguarda la distinzione tra le diverse orientazioni dei veicoli. Di conseguenza, abbiamo deciso di procedere con la creazione di un dataset personalizzato, costruito ad hoc per le specifiche esigenze del nostro sistema.

Le immagini utilizzate per l'addestramento e la validazione dei modelli provengono sia da risorse online pubblicamente accessibili, in particolare i dataset ***Roboflow*** e ***COCO*** *(Common Objects in Context)*, che offrono una buona varietà di scenari, condizioni ambientali e qualità visiva, sia da acquisizioni effettuate direttamente tramite la telecamera integrata del robot *Picar-X*. Queste ultime, seppur caratterizzate da una risoluzione inferiore, hanno garantito una rappresentazione più realistica del dominio operativo target. Inoltre, il dataset complessivo comprende sia immagini di veicoli reali sia di modellini in scala ridotta, al fine di aumentare la diversità morfologica e dimensionale dei soggetti riconosciuti dal sistema.

L'annotazione del dataset è stata effettuata utilizzando Label Studio, strumento open-source per l'etichettatura di immagini. Sono state definite cinque classi principali:

- car_front
- car_rear
- car_side_left
- car_side_right
- person

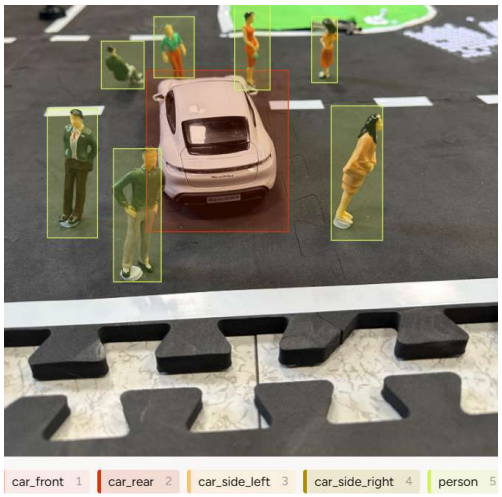


Figure 1. Label

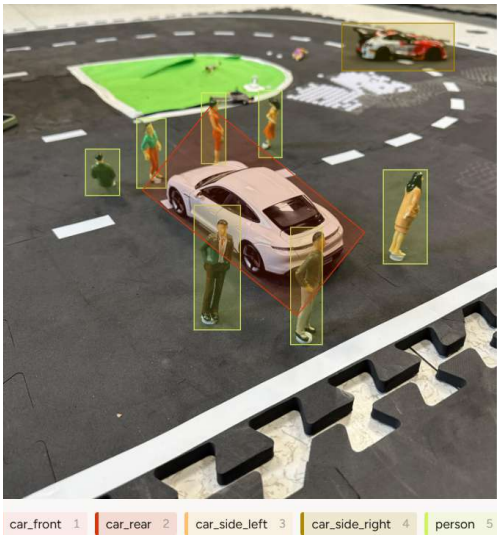


Figure 2. Label

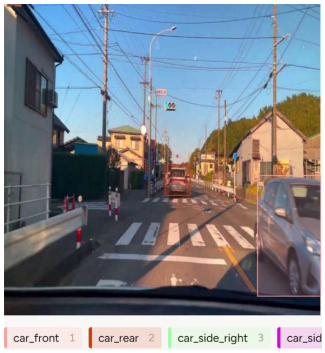


FIGURE 3. Label su Strada

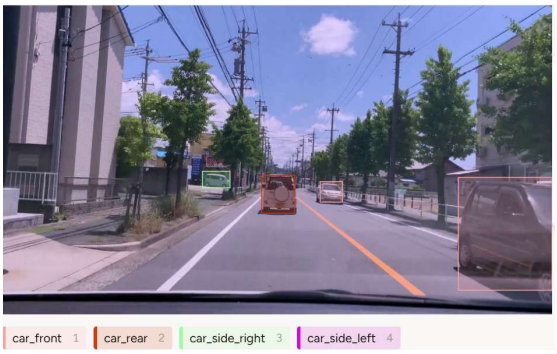


FIGURE 4. Label su Strada

FIGURE 5. Coordinate Bounding Box

Nelle Figure 1 e 2 è illustrato il processo di annotazione delle immagini tramite *Label Studio*, utilizzato per la creazione del dataset personalizzato. In particolare, viene mostrato come gli oggetti di interesse vengano selezionati ed etichettati all'interno dell'immagine.
Le Figure 3 e 4 mostrano invece esempi di annotazione di etichettatura applicata a uno scenario reale. In particolare, nella Figura 4 è visibile un caso di etichettatura sulle auto che include: car_rear, car_front, e car_side_right.

La Figura 5, invece, evidenzia il risultato dell'annotazione con la generazione automatica delle coordinate spaziali delle *bounding box* da parte del software. Questo consente una registrazione precisa delle aree rilevanti, garantendo coerenza e accuratezza nel processo di etichettatura.

Il dataset risultante comprende circa 1000 immagini per la classe 'person', 300 immagini per ciascuna vista laterale e circa 250 per car_front e car_rear. Questo bilanciamento è stato pensato per migliorare la precisione del modello nei casi più ambigui, in particolare nelle viste laterali.

Nella Figura 17, l'istogramma ci fornisce una chiara panoramica di quante volte ciascun tipo di oggetto appare nel dataset. Nel caso specifico delle classi relative alle automobili, il numero di istanze è stato intenzionalmente bilanciato: le categorie car_front e car_rear presentavano inizialmente una quantità significativamente superiore di immagini, ma sono state ridotte per uniformarsi al numero più contenuto di esempi disponibili per car_side_left e car_side_right. Questa scelta è

stata dettata dalla difficoltà nel reperire online immagini affidabili per le visuali laterali e dalla volontà di evitare un'eccessiva dipendenza da immagini sintetiche o autoprodotte, il che avrebbe potuto causare overfitting e ridurre la capacità del modello di generalizzare su dati reali e variabili.

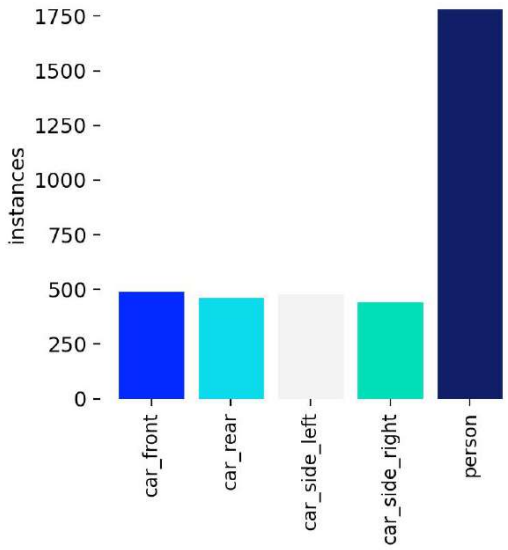


FIGURE 6. Istanze per classe

Per aumentare la robustezza e la capacità di generalizzazione del sistema, è stata implementata una fase di data augmentation. Le trasformazioni più efficaci si sono rivelate il flip orizzontale, utile per diversificare le viste laterali; rotazioni controllate, per simulare differenti pose; e leggere distorsioni geometriche, che hanno migliorato la tolleranza del modello a deformazioni ottiche o movimenti.

Questa fase ha permesso di ottenere un dataset sufficientemente ricco, bilanciato e rappresentativo, contribuendo in modo significativo all'efficacia finale del sistema, pensato per operare in tempo reale su dispositivi embedded a risorse computazionali limitate, come il Raspberry Pi.

3.2. **Addestramento del modello.** Dopo la costruzione del dataset, è stata avviata la fase di addestramento del modello per il riconoscimento dei pedoni e il rilevamento dell'orientamento dei veicoli. L'obiettivo principale era sviluppare un sistema accurato ma sufficientemente leggero da poter essere eseguito in tempo reale su piattaforme come il Raspberry Pi 5.

Abbiamo sperimentato diverse architetture della famiglia YOLO (You Only Look Once), valutando attentamente il compromesso tra precisione, velocità di inferenza e compatibilità con dispositivi embedded. Tra le versioni testate:

- YOLOv8, noto per l'elevata accuratezza e completezza architetturale, ma con requisiti computazionali più elevati e meno adatto a un deploy su CPU ARM;
- YOLOv11, in particolare le varianti Nano e Small, ottimizzate per dispositivi a risorse limitate, offrivano un buon bilanciamento tra prestazioni e leggerezza.

La versione YOLOv11-Small si è rivelata la più adatta per le esigenze del progetto, offrendo un ottimo equilibrio tra accuratezza e leggerezza architetturale. Oltre a garantire buone prestazioni di rilevamento, ha mostrato una notevole fluidità nell'elaborazione video, mantenendo un frame rate stabile e reattivo anche in fase di inferenza su CPU ARM, senza il supporto di acceleratori esterni.

3.3. **Preparazione dei dati.** Il dataset è stato composto da immagini provenienti sia da Internet, con alta risoluzione e variabilità visiva, sia da acquisizioni tramite il PiCar-X. La combinazione di veicoli reali e modellini ha contribuito a rafforzare la capacità del modello di generalizzare in scenari diversi.

Per migliorare la robustezza del modello rispetto a variazioni di angolazione, scala e occlusioni parziali, sono state applicate diverse tecniche di data augmentation, tra cui: flip orizzontali e verticali, rotazioni angolari, crop selettivi (mantenendo almeno il 20% del soggetto visibile) e distorsioni geometriche controllate.
Un'attenzione particolare è stata posta nella modifica automatica delle annotazioni durante l'augmentation, mantenendo il formato YOLO con coordinate normalizzate. Questo ha permesso di generare varianti realistiche delle immagini originali senza compromettere la coerenza delle etichette.

3.4. **Configurazione dell'addestramento.** Il modello YOLOv11-small è stato addestrato localmente su GPU, con esportazione finale in formato .pt, e successiva conversione in ONNX, più adatto all'esecuzione su CPU ARM in ambiente Raspberry Pi.
Le principali impostazioni adottate in fase di training sono state:

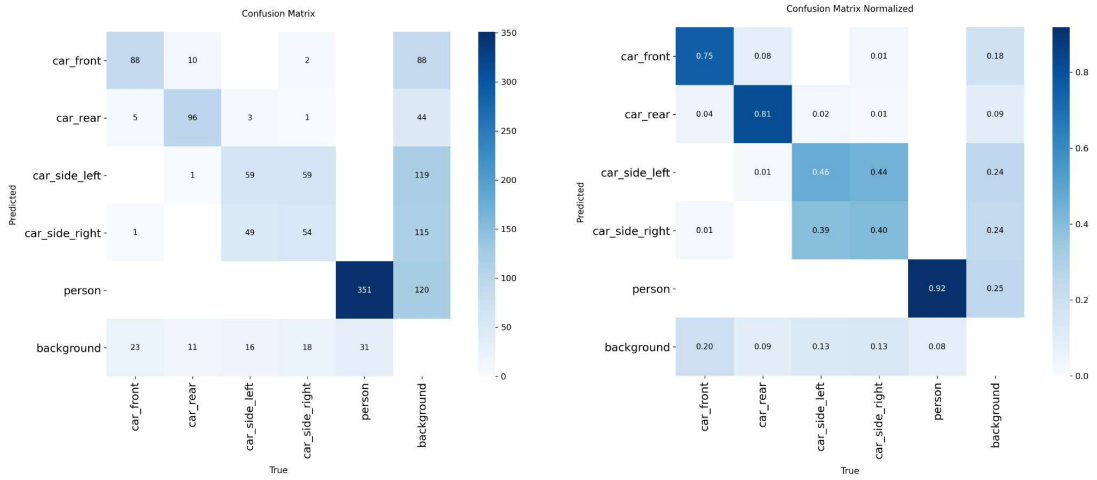
- imgsz = 320, per bilanciare il livello di dettaglio e la velocità di esecuzione;
- batch size adattato alla memoria GPU disponibile;
- loss function combinata (objectness, class, box regression), standard in YOLO;
- early stopping e salvataggio automatico del miglior modello in base alla validazione.

3.5. **Analisi delle Prestazioni del Modello di Classificazione.** L'analisi delle prestazioni del modello di classificazione è stata condotta esaminando, inizialmente la matrice di confusione, sia nella sua forma standard sia normalizzata. In particolare, la matrice normalizzata consente di valutare le performance in modo indipendente dalla distribuzione delle classi nel dataset di test.
Dall'analisi emergono i seguenti risultati chiave:

- **Performance eccellenti:** il modello dimostra una capacità di classificazione molto elevata per la classe `person`, raggiungendo un'accuratezza

(recall) del 92%. Anche la classe `car_rear` ottiene ottimi risultati, con l'81% delle istanze correttamente identificate.

- **Performance moderate:** la classe `car_front` viene classificata correttamente nel 75% dei casi. L'errore principale per questa classe consiste nella confusione con il `background`, verso cui viene erroneamente classificato il 18% delle istanze, indicando che una parte di questi oggetti non viene rilevata.
- **Criticità principale — Confusione tra viste laterali:** la debolezza più significativa del modello risiede nella distinzione tra le viste laterali dei veicoli. L'accuratezza per le classi `car_side_left` e `car_side_right` è notevolmente bassa, attestandosi rispettivamente solo al 46% e al 40%. Ciò evidenzia una forte ambiguità visiva tra le due categorie.



(A) Matrice di confusione standard (B) Matrice di confusione normalizzata

FIGURE 7. Confronto tra matrice di confusione standard (a sinistra) e normalizzata (a destra).

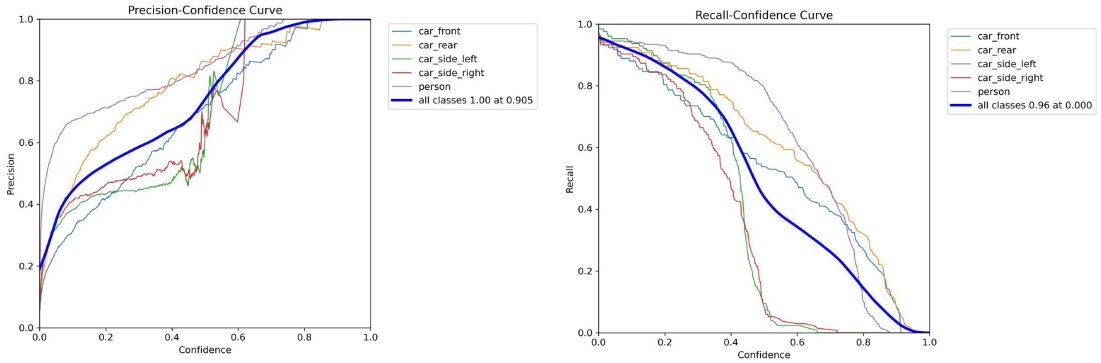
3.5.1. *Analisi delle Curve di Precisione, Recall e Precision-Recall.* Successivamente si è passati all'analisi delle curve di precisione e recall che, in funzione della soglia di confidenza ci consente di comprendere come varia la qualità delle predizioni al variare della selettività del modello.
Precision. Il grafico della precision mostra l'accuratezza delle predizioni in funzione della soglia di confidenza. In generale, si osserva un aumento della precisione con l'incremento della soglia, poiché il modello seleziona solo le predizioni di cui è più sicuro.

- **Tutte le classi**: la linea blu ("all classes") raggiunge una precisione di 1.00 a una soglia di confidenza di 0.905, indicando che oltre tale soglia le predizioni sono quasi sempre corrette.
- **Classi robuste**: la classe `person` (viola) mantiene un'elevata precisione anche a soglie più basse. Anche `car_front` e `car_rear` mostrano buone prestazioni.

- **Classi problematiche**: `car_side_left` (verde) e `car_side_right` (rosso) evidenziano un comportamento instabile, con precisione fluttuante soprattutto a soglie basse.

Recall. Il grafico del richiamo illustra la capacità del modello di individuare correttamente tutti gli oggetti effettivamente presenti.

- **Andamento generale**: come atteso, recall diminuisce all'aumentare della soglia di confidenza, poiché un modello più selettivo tende a perdere più oggetti.
- **Tutte le classi**: la curva "all classes" parte da un recall di 0.96 a soglia 0.000, quando tutte le predizioni vengono accettate.
- **Classi robuste**: `person`, `car_rear` e `car_front` mantengono una buona recall anche a soglie più alte.
- **Classi deboli**: `car_side_left` e `car_side_right` perdono rapidamente recall con soglie elevate, indicando che molti oggetti non vengono rilevati.



(A) Precision Confidence Curve (B) Recall Confidence Curve

FIGURE 8. Andamento di precisione e recall per ciascuna classe al variare della soglia di confidenza.

Curva Precision-Recall. La curva Precision-Recall consente di valutare il compromesso tra precisione e recall su tutte le soglie di confidenza, ed è utile per quantificare la qualità del modello tramite l'Area Sotto la Curva (AP).

- **Performance complessiva**: il modello presenta un valore medio di AP pari a **mAP@0.5 = 0.694**, segnalando una performance complessiva solida.
- **Classi migliori**: `person` si distingue con un AP di 0.893, seguita da `car_rear` (0.833) e `car_front` (0.706).
- **Classi critiche**: `car_side_right` (0.524) e `car_side_left` (0.514) mostrano curve che decrescono rapidamente, riflettendo una maggiore incidenza di errori.

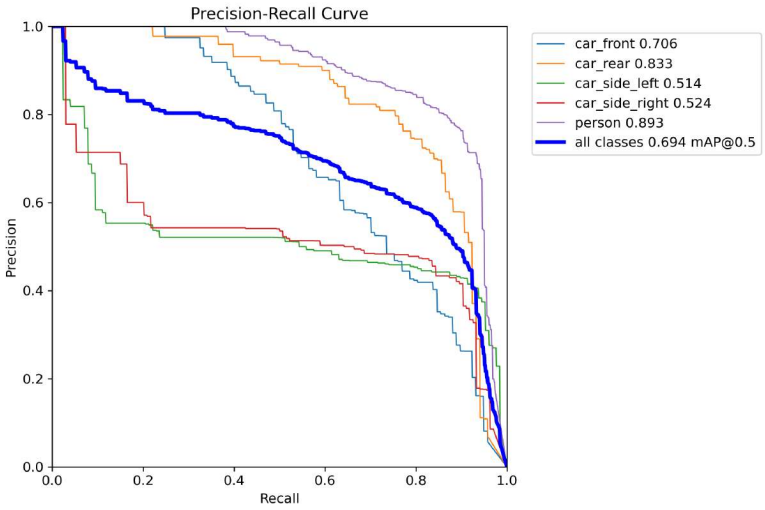


FIGURE 9. Curve Precision-Recall per ciascuna classe. Il mAP@0.5 complessivo è pari a 0.694.

3.6. **Risultati su Picar-X.** Il modello finale, una volta ottimizzato, è stato integrato all'interno del ciclo di inferenza in tempo reale, eseguito direttamente a bordo del *Raspberry Pi* equipaggiato con la fotocamera del sistema *PiCar-X*.

Le figure seguenti mostrano un esempio di output visivo generato dal sistema durante l'esecuzione real-time, evidenziando la capacità del modello di rilevare e classificare correttamente gli oggetti nell'ambiente circostante.

FIGURE 10. Il sistema rileva correttamente il retro di un'automobile con una confidenza del 68%, mentre le due persone presenti nella scena vengono riconosciute con una confidenza superiore, pari all'84%.

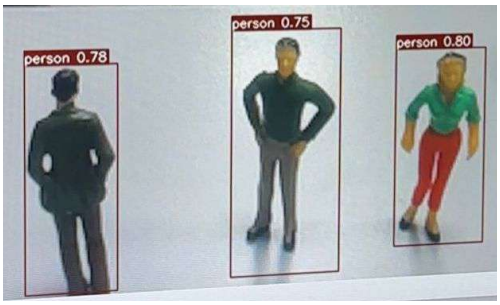


FIGURE 11. Un contesto in cui sono presenti solamente persone: il modello individua correttamente tutte e tre le persone visibili, con livelli di confidenza compresi tra il 75% e l'80%.

FIGURE 12. Riconoscimento del lato sinistro di un veicolo, ottenuto con un'elevata confidenza pari all'86%, insieme all'identificazione delle persone presenti nella scena.

3.7. **Effetti della Fusione dei Dataset e Analisi Comparativa.** Prima di passare alla logica funzionale, si è proceduto ad effettuare la fusione tra i 2 dataset: Object Detection per persone ed orientamento auto e Object Detection per segnali stradali e semafori.

L'evoluzione del modello, ottenuta attraverso la fusione di dataset eterogenei che includono ora nuove classi di segnaletica stradale e semafori oltre a veicoli e persone, ha portato a un significativo miglioramento del mAP complessivo, passato da 0.694 a 0.842. Questo incremento è attribuibile principalmente all'eccezionale performance del modello sulle classi di nuova introduzione (`stop_s`, `10_limits`, `20_limits`, `bidirectional`, `green_TL`, `yellow_TL`, `red_TL`), che dimostrano AP, precisione e richiamo prossimi all'eccellenza, come visibile nelle curve PR e Precision-Confidence.

Un altro risultato positivo della fusione è il drastico miglioramento delle performance per le classi `car_side_left` e `car_side_right`, le cui AP sono aumentate in modo sostanziale (da circa 0.51 a oltre 0.80), indicando che il modello è ora molto più competente nel rilevare queste viste laterali dei veicoli.

Tuttavia, questo ampliamento e ri-bilanciamento del dataset ha introdotto un compromesso: le classi `car_front` e `car_rear`, che in precedenza mostravano AP elevate (rispettivamente 0.706 e 0.833), hanno subito un calo significativo delle performance (AP scese a circa 0.530 e 0.545). La matrice di confusione rivela che per queste classi si è verificato un aumento dei falsi negativi (istanze non rilevate, classificate come "background"), suggerendo che il modello sta perdendo

più oggetti reali di tipo `car_front` e `car_rear` rispetto al passato.

Anche se la classe `person` mantiene una performance robusta (AP: 0.839), si osserva un leggero aumento nei falsi negativi anche in questo caso. Questo scenario indica che la fusione e la conseguente variazione nella distribuzione dei dati tra le classi hanno ottimizzato il modello per una gamma più ampia di oggetti e per alcune classi precedentemente deboli, ma hanno richiesto un riadattamento delle capacità di rilevamento per le classi `car_front` e `car_rear`.

Future ottimizzazioni dovranno mirare a mitigare questa regressione, potenzialmente attraverso strategie di bilanciamento del dataset o affinamento degli iperparametri specifici per queste classi.

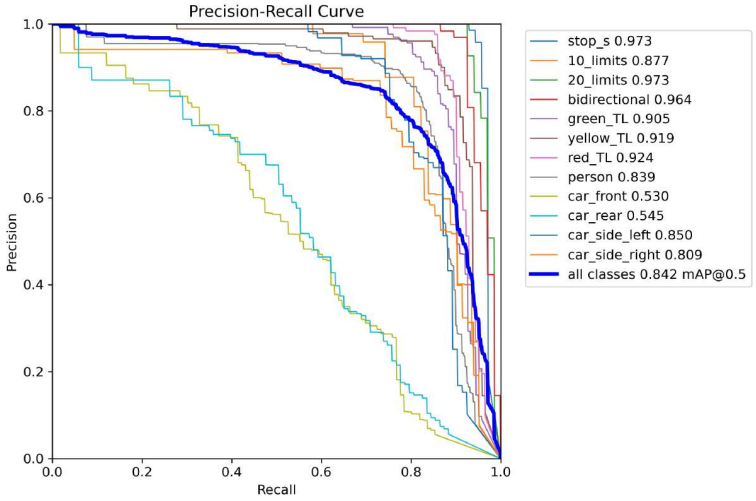


FIGURE 13. Curva Precision-Recall per ciascuna classe e aggregata (all classes).

Analisi Dettagliata delle Singole Metriche La Figura 13 presenta le curve Precision-Recall (PR) per ciascuna classe e la curva aggregata *all classes*. Questo grafico mostra che il mAP@0.5 complessivo del modello è pari a 0.842, indicando una robusta capacità di rilevamento. Le classi di segnaletica (`stop_s`, `10_limits`, `20_limits`, `bidirectional`) e semafori (`green_TL`, `yellow_TL`, `red_TL`) mostrano curve PR estremamente elevate, con AP superiori a 0.90, evidenziando prestazioni quasi ideali in termini di accuratezza e copertura.

Le classi `car_side_left` (AP: 0.850) e `car_side_right` (AP: 0.809) mostrano curve PR migliorate rispetto alle versioni precedenti del modello, segnalando un progresso nella rilevazione laterale dei veicoli. Al contrario, le curve per `car_front` (AP: 0.530) e `car_rear` (AP: 0.545) evidenziano un calo prestazionale. La classe `person` (AP: 0.839) mantiene una curva PR elevata, sebbene lievemente inferiore rispetto alle iterazioni precedenti.

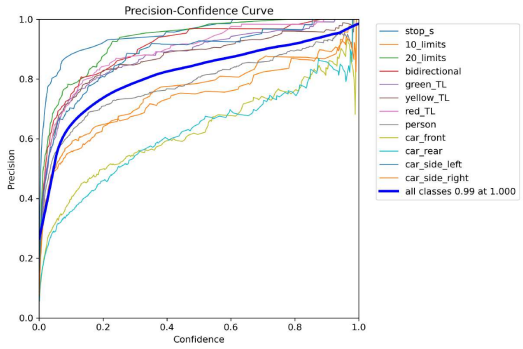


FIGURE 14

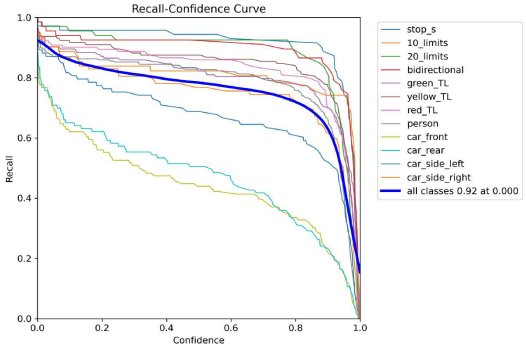


FIGURE 15

Le Figure 14 e 15 illustrano rispettivamente l'andamento della precisione e della recall del modello in funzione della soglia di confidenza.

La Figura 14 mostra che, in generale, all'aumentare della soglia di confidenza richiesta per accettare una predizione, la precisione cresce. Questo comportamento è atteso, poiché vengono mantenute solo le predizioni più sicure. La curva *all_classes* raggiunge una precisione di 0.99 per confidenza pari a 1.000, evidenziando un'elevata affidabilità del modello a soglie massime. Le classi di segnaletica e semafori raggiungono rapidamente livelli massimi di precisione anche a soglie intermedie, confermando l'efficacia del modello su queste categorie. Le curve relative a `car_side_left` e `car_side_right` risultano migliorate e più stabili rispetto al modello non unito, segnalando un progresso sostanziale. Al contrario, `car_front` e `car_rear` mostrano precisione inferiore a soglie intermedie, indicando maggiore incertezza in fase di classificazione, aspetto che non accadeva nella versione precedente. La classe `person` si mantiene stabile e robusta.

La Figura 15 evidenzia come la recall tenda a decrescere all'aumentare della soglia di confidenza, in quanto un filtro più rigido porta all'esclusione di predizioni meno sicure, anche se corrette. Il richiamo aggregato (*all_classes*) raggiunge 0.92 a confidenza 0.000, denotando un'elevata sensibilità iniziale. Le classi di segnaletica e semafori conservano livelli elevati di recall anche a soglie intermedie, dimostrando una bassa incidenza di falsi negativi. Le curve di `car_side_left` e `car_side_right` risultano significativamente migliorate rispetto al modello precedente, mantenendo buoni valori su un ampio range. Al contrario, `car_front` e `car_rear` evidenziano un calo rapido della recall, suggerendo una maggiore tendenza del modello a perdere istanze reali di queste classi in presenza di soglie più elevate.

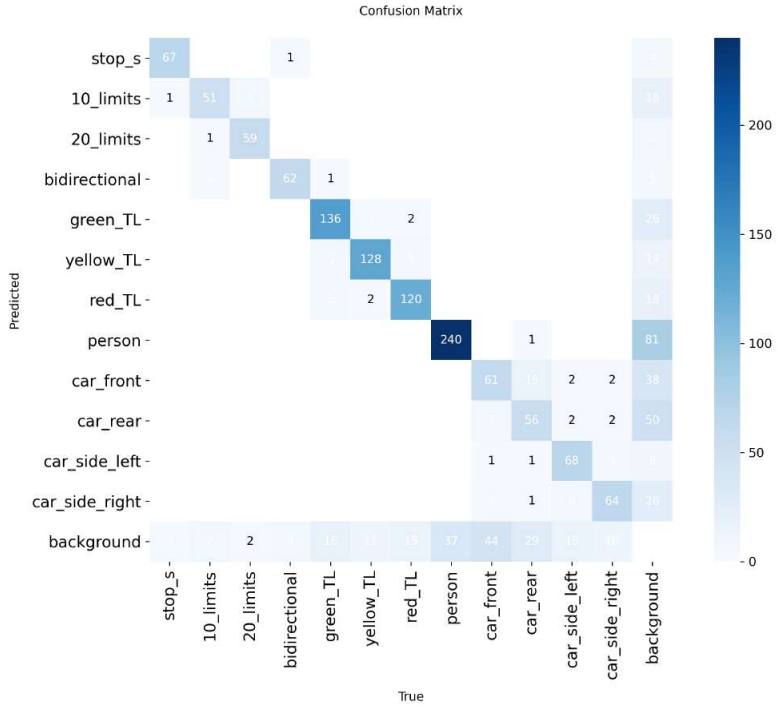


FIGURE 16. Matrice di confusione del modello unito

La Figura 16 mostra la matrice di confusione, più nello specifico:

Rilevamenti accurati (diagonale principale): Le classi relative alla segnaletica e ai semafori mostrano un numero elevatissimo di veri positivi (es. `green_TL: 136`, `yellow_TL: 128`, `red_TL: 120`), con pochissimi scambi tra di loro. Anche `person` (240), `car_side_left` (68) e `car_side_right` (64) presentano un buon numero di rilevamenti corretti.

Falsi negativi (colonna background): La classe `person` evidenzia il numero più alto di falsi negativi (81), seguita da `car_rear` (50) e `car_front` (38), indicando una difficoltà del modello nel rilevarle in modo completo.

Errori di classificazione (falsi positivi): Gli errori tra le classi di segnaletica e semafori sono minimi. Alcune confusioni si verificano tra le varie viste delle auto e tra `car_front` e `person`. Infine, la riga "background" evidenzia rilevamenti non corretti di oggetti inesistenti.

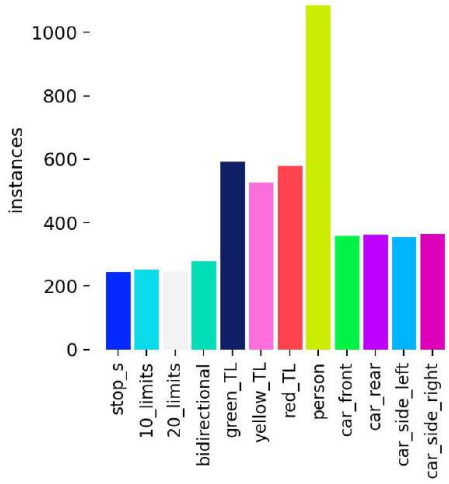


FIGURE 17. Istanze dopo la fusione

3.7.1. *Confronto delle Performance per Classe.* La Tabella 1 mette a confronto le AP ottenute per ciascuna classe prima e dopo la fusione dei dataset. Si osservano miglioramenti significativi nelle classi `car_side_left` e `car_side_right`, mentre le classi `car_front` e `car_rear` evidenziano un calo nelle prestazioni. La classe `person` mantiene invece un livello di performance sostanzialmente stabile, con una lieve diminuzione pari a 0.021. Complessivamente, il mAP finale registra un incremento di 0.148.

TABLE 1. Confronto delle AP per classe: prima vs. dopo la fusione

Classe	**AP Pre-Fusione**	**AP Post-Fusione**	**Differenza**
car_front	0.706	0.530	–0.176
car_rear	0.833	0.545	–0.288
car_side_left	0.510	0.850	+0.340
car_side_right	0.520	0.809	+0.289
person	0.860	0.839	–0.021
mAP	0.694	0.842	+0.148

4. Logica Decisionale

Dopo aver descritto nei capitoli precedenti l'architettura e i principi di funzionamento del sistema progettato, si è proceduto con una fase di sperimentazione su pista, costruita per emulare le principali condizioni di guida in uno scenario reale, seppur in scala ridotta. Questa fase rappresenta un passaggio cruciale, in quanto consente di validare empiricamente le logiche di controllo, le capacità di percezione ambientale e le reazioni dinamiche del veicolo robotico in condizioni controllate.
Il sistema implementato è basato su un *Raspberry Pi*, integrato con la piattaforma *Picar-X*, e combina componenti hardware e software al fine di ottenere una guida autonoma elementare ma funzionale. Il nucleo dell'intelligenza artificiale è costituito da due modelli di deep learning: **YOLOv11n**, utilizzato per il riconoscimento in tempo reale di segnali stradali, veicoli, persone e semafori, e un modello *TFLite* dedicato alla *Lane Detection* impiegato per determinare la traiettoria e gestire lo sterzo in modo adattivo.
All'avvio, il sistema provvede all'inizializzazione dei moduli hardware essenziali, come la fotocamera *PiCamera2*, i servomotori per la direzione e la propulsione, e un sensore a ultrasuoni frontale. Vengono inoltre avviati due *thread* paralleli: il primo è responsabile dell'acquisizione continua dei frame video, mentre il secondo esegue le inferenze YOLO in modo asincrono. L'obiettivo di questa architettura *multithread* è ridurre il carico computazionale sul thread principale, migliorando la reattività generale del sistema nonostante le limitazioni hardware del Raspberry Pi.
Due componenti svolgono un ruolo particolarmente rilevante:

- **L'inferenza visiva**, che traduce in tempo reale i dati catturati dalla telecamera in output semantici utili al processo decisionale (es. classificazione e localizzazione degli oggetti);
- **Il sensore a ultrasuoni**, utilizzato sia per il rilevamento di ostacoli (persone o veicoli), sia per innescare manovre specifiche, come il sorpasso, qualora l'oggetto venga rilevato a distanze inferiori a soglie predefinite (15 cm per persone, 25 cm per veicoli).

Durante il ciclo operativo continuo, ciascun frame acquisito è analizzato per due scopi principali: la *localizzazione della corsia*, che consente di calcolare l'angolo di sterzata ottimale, e il *riconoscimento semantico della scena* tramite YOLO. La gestione dello sterzo è soggetta a vincoli predeterminati, che limitano l'angolo massimo per evitare comportamenti instabili o eccessivamente bruschi.
Il modulo YOLO è responsabile del rilevamento e della classificazione di elementi chiave dell'ambiente stradale: segnali di stop, limiti di velocità, semafori, segnali di direzione, veicoli e pedoni. In funzione di questi input, il veicolo modula dinamicamente la propria velocità. Ad esempio, in presenza di un semaforo rosso o di una persona in prossimità, il sistema arresta completamente il movimento; in caso di semaforo giallo o limite di velocità, la velocità viene opportunamente ridotta.

Particolare attenzione è stata dedicata alla **gestione della manovra di sorpasso**. Qualora venga rilevata un'auto ferma, il sistema effettua una breve pausa per la stabilizzazione della direzione, quindi esegue una sequenza predefinita in quattro fasi: deviazione verso sinistra, avanzamento rettilineo, rientro a destra e ripresa del tracciato. Ogni fase è calibrata sia in termini di durata che di angolazione, al fine di garantire una traiettoria fluida e stabile. Durante il sorpasso, tutte le altre logiche vengono temporaneamente disattivate per evitare interferenze.

La sicurezza è garantita tramite il monitoraggio continuo del sensore a ultrasuoni: se una persona viene rilevata a distanza ravvicinata, il robot si ferma e rimane in attesa finché l'ostacolo non viene rimosso, continuando nel frattempo a eseguire le inferenze per verificare in tempo reale l'evoluzione della scena.

A supporto della fase di debugging e analisi, viene impiegata la libreria *OpenCV*, che consente di visualizzare in tempo reale l'output del modello YOLO sovrapposto ai frame acquisiti, includendo annotazioni (bounding box, etichette, probabilità) e il frame rate corrente, utile per valutare le prestazioni del sistema.
Il codice è inoltre dotato di controlli robusti per la gestione delle eccezioni, con blocchi `try/except` che garantiscono una maggiore stabilità durante l'esecuzione.
Nonostante la completezza funzionale del sistema, è doveroso segnalare alcune **limitazioni prestazionali**, in particolare legate alla bassa capacità computazionale del Raspberry Pi. La latenza nell'elaborazione delle immagini e la limitata fluidità nella risposta agli eventi ambientali rappresentano i principali colli di bottiglia. Si prevede che l'impiego di hardware più performante e un'ulteriore ottimizzazione degli algoritmi possa significativamente migliorare la reattività e l'affidabilità generale del sistema.

5. Conclusioni

Il lavoro svolto ha permesso di raggiungere pienamente l'obiettivo iniziale, ovvero il riconoscimento accurato di pedoni e veicoli con la conseguente esecuzione di azioni specifiche. Tale risultato è il frutto di scelte metodologiche attente e mirate, che hanno riguardato l'intero processo: dalla creazione e annotazione del dataset ad hoc, passando per l'ottimizzazione e l'implementazione del modello su piattaforma Raspberry Pi.
Le performance ottenute si sono dimostrate soddisfacenti sia in termini di accuratezza del riconoscimento sia di velocità di esecuzione, elementi essenziali per garantire un funzionamento efficace in ambiente embedded. Nonostante le difficoltà incontrate, in particolare per il riconoscimento differenziato dei lati destro e sinistro del veicolo, il sistema ha dimostrato un'affidabilità elevata nel distinguere correttamente tali classi.
Infine, la Picar-X, operante in condizioni di simulazione su pista, ha risposto in modo coerente e fluido alle azioni predisposte, mostrando stabilità operativa e assenza di crash durante le sessioni di prova.

5.1. **Implementazioni Future.**

Nonostante i risultati ottenuti siano stati soddisfacenti, il sistema presenta margini di miglioramento sia in termini di capacità di riconoscimento sia di prestazioni operative. Un primo ambito di sviluppo riguarda l'ampliamento delle classi di oggetti riconosciuti, includendo ad esempio biciclette, motocicli, autoarticolati o anche animali, al fine di rendere il sistema più versatile e applicabile a scenari di guida autonoma più complessi.

Un ulteriore miglioramento potrebbe riguardare il riconoscimento delle parti laterali del veicolo, in particolare lato sinistro e destro, rendendolo più robusto attraverso l'addestramento su un numero significativamente maggiore di esempi, al fine di ridurre gli errori e aumentare l'affidabilità in condizioni variegate.

Un ulteriore aspetto di sviluppo riguarda il miglioramento della manovra di sorpasso dell'auto, rendendola molto più automatica e dinamica, superando l'attuale approccio statico e basato su regole fisse. Questo permetterebbe al sistema di adattarsi in tempo reale alle condizioni del traffico e alle caratteristiche della strada, aumentando l'efficienza e la sicurezza delle manovre.
Inoltre, la maggior parte delle immagini presenti nel dataset è stata acquisita in ambiente controllato di laboratorio, limitando la varietà e la complessità delle condizioni visive. Un arricchimento e diversificazione del dataset con immagini raccolte in contesti reali e ambienti differenti migliorerebbe ulteriormente la generalizzazione del modello.

Infine, per quanto riguarda le prestazioni computazionali, si potrebbero esplorare tecniche avanzate di pruning e quantizzazione del modello, o l'impiego di acceleratori hardware dedicati come TPU edge, con l'obiettivo di incrementare il frame rate e ridurre il consumo energetico senza compromettere la precisione.
Questi sviluppi futuri consentirebbero di aumentare l'affidabilità e l'efficacia del

sistema, ampliandone il campo di applicazione e migliorandone la robustezza in situazioni reali.

References

[1] SunFounder. Picar-x official documentation. Online; accessed 2025-06-26, 2023. https://docs.sunfounder.com/projects/picar-x/en/latest/.
[2] SunFounder. Picar-x - ai self-driving robot car kit. Online; accessed 2025-06-26, 2023. https://www.sunfounder.com/products/picar-x.
[3] SunFounder. Vilib python library for raspberry pi. Online; accessed 2025-06-26, 2023. https://github.com/sunfounder/vilib.

Department of Computer Science, University of Salerno, Italy
Email address: A.SALURSO3@STUDENTI.UNISA.IT
Email address: F.AVALLONE28@STUDENTI.UNISA.IT